Smart Prompter Documentation

Likhon Gomes (Leo), Dr Chiu Tan, Sarah Lehman

How to Install the app, go to this github link : https://github.com/likhongomes/Smart-Prompter-iOS and git clone the repository and open it using xcode. click the run button or press cmd+R on keyboard to run the app, on a simulator or your iPhone.

Requirement: iOS 10 or above and xcode version 10 and above.

Smart Prompter Patient App

This is a basic app that let's patients respond to the alarms set by the caretake in Caretaker app.

Classes & ViewControllers

AppDelegate

The app delegate is the root class of the app. When the app is executed, the app delegate sets all of the configuration, all of the connections to the services used and run. In the app delegate I have declared all of the necessary variables and constants that are going to be universally accessible throughout the app, for example "userID:Auth.auth().currentUser?.uid" established connection with firebase and fetch's the current userID that's logged into the app.

- didFinishLaunchingWithOptions functions is the main function of the app, in this function I have specified the the app to show sign up page to show up if the user is logged out. If the user is already logged in, then the home screen is shown. Next in the same function I have called a function registerForPushNotification, which asks for permission from the user and enables push notification.
- performFetchWithCompletionHandler function fetches data from the firebase server when the app is in background mode. The os lets the app to execute this function to fetch data from firebase. Whenever this function is run, the app has to let the OS know about its completions status using the completion handler. The OS sets the apps' background execution priority based on the frequency of completion success from completion handler.
- `didRegisterForRemoteNotificationWithDeviceToken` function registers the device with a device token for the app to deliver notifications.
- `didFailToRegisterForRemoteNotificationsWithError` this function is executed if the app fails to register for notification.
- `setupDatabase` this functions is used to setup GRDB database to store alarms in the phone's persistent memory. It creates a connection in the apps documents folder and creates a file called "SmartPrompter.sqlite". This is where the alarm data are stored, but currently it's inactive because we shifted to firebase. I still kept this function just in case if we use it in future.
- `registerForPushNotifications` function configures the notification type served by the app, for example, this functions sets the alert type, sound and the badge.

If anyone wants to change the sound or anything related to push notifications, this is where they have to do it.

SignInVC

This is the viewcontroller that the user sees when they open the app and they are not logged in. This view and all other views in this app were made using programmaticUI. That means I have coded each and every single UI element. Most of the functions for UIElement setup are same. Therefore I will describe the function for one UI Element.

- `loginButtonSetup` function add the UIButton element loginButton to the view, next auto layout is activated, and the location of the button is specified in relevance of locations from other UIElements. Next, the button's height and width is specified and then the background color and a title when the title is in normal mode. Finally addTarget function call adds the action taken when the button is pressed.

- `loginButtonTapped.` Function is executed only when loginButton is tapped, note before the function it says @objc, which means the function has been exposed to Objective C as apple uses objective C as the foundation of iOS. Insite the function, we try to sign in using Auth.auth().signIn(…) function call. I used guard let to make sure to catch any errors during signup process. If the signin session is successful, it does a transition to next view controller which is MainVC in this case. If signin fails, it prints an error code to the console.

- `signUpButtonTapped` function is similar to loginButtonTapped but in this function I have used Auth.auth().createUser(…) function call to create a user account on firebase platform. Before going ahead with the signup process, I have implemented condition statements to check whether the text fields are empty or not. If the text fields are empty, it won't proceed to the next step.

- `viewWillAppear` function is a built in function from iOS, its executed when the viewcontroller is loaded to execute code ahead of time. In this function I have called a handle function called Auth.auth().addStateDidChangeListener(…) which looks whether the user is already signed in and listens for activities, the closes thing I can compare this function to is implementing cookies in a web browser.

- `keyboardWillShow` function listens for keyboard appearance on the screen and make adjustment of the textfields and raise them above the keyboard so that the textfield is not hidden behind the keyboard.

- `keyboardWillHide` similar to keyboardWillShow function, this function listens when the keyboard disappears from the screen and puts back the textfield back in their spot by setting their frame origins to zeros.

FirebaseUtil class

This class is the firebase management class in the app. It's a work in progress, made this class to have one class to maintain all of the firebase actions throught the whole app.

- `fetchOneObject (FirebaseID:String)` function fetches only one alarm data from firebase given the firebase ID. This function goes to the storage location and fetches the Alarm data and them puts them in an Alarm data structure for maintainability. It also looks for the alarms status, if its active, the alarm is added to activeAlarm array. It also schedules the alarm for notification from the fetched data and time from firebase. Later in the function, the I printed out the snapshot just to check in the console if the data is being fetched properly.

AlarmScheduler class
This class was written to take care of all actions related to alarm scheduling.

- `scheduleNotification(title:String, dateComponents:DateComponents, id:String?)` function schedules a notification in the OS. It takes three params, they are title of the function, dateComponents and the id. At first the function checks for the iOS version and whether the iOS device can schedule an alarm (only devices with iOS version 10 and above can schedule alarms). Next the function creates a Notification content using the params passed, Next the notification content is passed into a calendarTrigger which Is a data structure for the OS to trigger the notification even when the app itself is not running. Next the calendarTrigger is wrapped around a request along with an identifier and the metadata of the notification which can be retrieved later. At this stage the notification request is added in NotificationCenter for iOS to be delivered at a specific date and time. If the whole operation fails, the error is printed out on the console.

AlarmVC
This class is the view controller where the user responds to an existing alarm. It provides the user with the details of the alarm and a slider to complete or request extension in the alarm.

- `showDatePicker()` function controls the appearance of the datePicker carousel. It creates and shows the done/cancel button and get the value from the datePicker for later use.
- `showTimePicker()` function is same as the datePicker, except this one is only for picking time.
- `donedatePicker()` function takes the value selected from the picker and formats it into specific order and then closes the datePicker.
- `donetimePicker()` function is same as the time picker's
- `cancelDatePicker()` cancels the datePicker and ignores the value picked.
- `sliderSetup()` function sets up the slider and adds it on the view, and then the function sets the value for the sliders from 0 being all the way to the left to 100 being all the way to the right, and the slider being on 50 by default.

- `imageViewSetup()` function is provided by iOS, its only executed when the user takes picture using the camera. After a picture is taken, the user is taken back to the previous screen and the image is shown in the imageView. Next the image is compressed so that it can be uploaded to firebase. Then the picture is uploaded to firebase using storageRef.putData(…)
- `changeValue(_ sender: UISlider)` Here the param UISlider is a default from iOS. function is executed only when the slider is moved, it checks if the value is 0, then the scheduler is called and notification is put back into the NotificationCenter with an added delay time (It's still work in progress), In the meantime the label is updated to remind me later. When the slider value is 100, the label updated to "On my way" and camera is triggered and the user is asked to take a picture, after he user takes a picture it's shown on the imageView. After this the alarmData on firebase is updated with completionTime, and status.

Alarm class

This class serves as the data structure for the alarm data. It inherits from PersistableRecord, Codable and FetchableRecord. These are all GRDB specific classes. GRDB is the sqlite class that enables a local database in the persistent storage of the device. The data that are stored in the data structure are:

e
Id: Int
FirebaseID:String
Label:String
Year:Int
Month:Int
Day:Int
Hour:Int
Minute:Int
Active:Bool
Status:String

- `encode(to container: inout PersistenceContainer)` function is given by GRDB, this helps the data to be loaded into the sqlite database. It puts the value using the key value into the database.

AlarmDB Class

This class manages the input and output of alarm data into the sqlite database.

- Initializer checks if the table already exists, if not then it creates a new table using the name provided in the class.
- `insert(user:Alarm)` function takes an alarm as a param. Then it writes the alarm data into the db.
- `getAll() -> [Alarm]` functions fetches all of the entries from the table in the database, loads them in an array and then returns it.

- `getActiveAlarms() -> [Alarm]` function gets all the active alarms in the database and returns an array loaded with active alarms.
- `getInactiveAlarms() -> [Alarm]` function is same as Active. It returns inactive alarms.

MainVC extension inheriting UNUserNotificationCenterDelegate
This is just an extension that connects the user notificationcenter to mainVC. So that MainVC can handle all of the notification tasks for the app.

- `userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void)` function is a builtin iOS function. It's only executed when the notification is delivered to the user while the app is running in the foreground. This is where I tried to reschedule the repeat notification feature, It works whever the app is running in the foreground, unfortunately doesn't work in the background because this app is never executed then.
- `func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive response: UNNotificationResponse, withCompletionHandler completionHandler: @escaping () -> Void)` function is a builtin iOS function, it's executed when the user responds to the notification tap. In this function I have fetched the notification information and then passed them into the next ViewController and and present the viewcontroller whenever the user touches the function, whether in background or foreground. Once this function is executed, the alarm data is updated by adding acknowledgement time of the alarm.

MainVC
This is the main viewcontroller of the app. The heart and the backbone, most of the actions happen in this viewcontroller. This is the Viewcontroller the user sees right after signing into the app.

- `tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int` function is builtin from iOS. It prepares the number of rows on the table based on the number of items in the column returned.
- `tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell` function sets what text to show on the table elements.
- `tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath)` function is only executed when the user selects and element from the table and then AlarmVC is presented where the user gets to see all of the details for the notification and the task that needs to be completed.
- `viewDidLoad()` is the main function of the viewcontroller, it loads all of the the GUI of the app and then fetches all the alarms from firebase and reloads the table to show

all of the fresh data fetched. Then there are three more function calls that specifies the style of date and time formatter and then the third function call schedules a timer to run on the MainVC to show the current time.

- `logoutButtonClicked()` signs the user out of firebase using firebaseAuth.signOut() function call. And then takes the user back to signup/signin screen.
- `fetchFromFirebase()` function is same as the one if FirebaseUtilsClass, only exception is, this functions reloads the table everytime a new entry is added to the table.

Extension UIView

This is not a class, rather an extension to UIView class. This gives me extra added function into the uiview class.

- `addNavigationBar(viewControllerName:String, leftButton:UIButton = UIButton(), rightButton:UIButton = UIButton())` function adds a topbar to every single viewcontroller wherever this function is called. It places the name of the viewcontroller in the center of the top bar from the given string in the parameter and places a left button and a right button if any are passed in the parameter, otherwise it's left to be blank.

AppDatabase

It's a class provided by GRDB, it connects the app to the database file in the device's documents folder and then returns the file to be opened by GRDB.

Smart Prompter Caretaker App

This is a basic app that allows a caretaker to remotely set alarm to remind the patient they are taking care of to do certain tasks on specifically on time. This app has a lot of similarity, because they were made from the same foundation and they have similar viewcontrollers that I explained in Patient app. So I will explain the unique viewcontrollers in this app, and refer to patient app for explanation regarding similar viewcontrollers.

SignInVC

Signin vc is exactly same as the one in Patient App, only difference is, there an extra sign up button in this app where the user can sign up a new patient-caretaker account.

- `signUpButtonTapped()` when signup button is tapped, it creates a new account on Firebase database using email address and password. Then the view controller presents the user with UserInfoVC where the user can input information about the caretaker and the patient.

UserInfoVC

This viewcontroller consists of four text fields to collect Caretaker's first/last name and Patient's first/last name. After the user is done inputting the information, they can tap on done button to proceed to next viewcontroller, which is MainVC.

MainVC
This is the main heart and soul of the application, most of the important operations take place in this viewcontroller and it shares a lot of its DNA from the other app's MainVC.

- `logoutButtonClicked()` function is only executed when the logout button is clicked. This logs out the user from existing firebase user session using firebaseAuth.signOut() function call. Then the user is taken back to SignInVC.
- `newAlarmButtonSetup()` function is only executed when newAlarmButton is clicked. When executed, this button takes the user to AlarmVC where the user can create a new alarm.
- `viewAlarmButtonClicked()` function is executed when viewAlarmButton is clicked. Similarly to new AlarmButton it takes the user to CurrentAlarmVC where they can see the alarms currently active.
- `pastAlarmButtonClicked()` function is executed when pastAlarmsButton is clicked. It takes the user to PastAlarmsVC which is exactly similar to CurrentAlarmVC. Here they can see all of the past completed alarms.

AlarmVC
This view controller is same as the one in Patient app. Only difference is, in this app the user can interact with textfields. That's because of one Boolean variable called editable, when editable is true, the users can interact with the textfields, when false, users can't interact with the text fields. In this app the variable is set to true and in Patient app the variable is set to false. This ways I was able to reuse the same code for both of the apps.

PastAlarmVC
This viewcontroller consists of a large table view where all the fetched data from firebase is shown. Mainly past alarms are shown.

- `fetchFromFirebase()` function fetches data from firebase. It filters out all the active alarms and puts the inactive alarms into an array and then provides it to be shown into the table.
- `backButtonClicked()` is only executed when the backButton is pressed. It takes the user back to MainVC

CurrentAlarmVC
This viewcontroller is same as PastAlarmVC, except the fetchFromFirebase() function in this VC filters out all the inactive alarms and adds only the active alarms into the array and provides it to be shown in the table.

Alarm class
This class is same as the one in Patient app, because both of the apps use the same database. For further reference please look at Patient app's Alarm class.

Extension UIView
This extension is similar to the one in Patient app. For further reference, please check the Extension UIView in Patient app.

Services Used

External services used in this app.

- **Firebase** – due to iOS's tight security architecture, It's nearly impossible to share data between two iOS apps, only possible way to do is to send the data out of the phone and then brining it back. Therefore, I have used firebase database service to send an receive data into iOS app. This not only connects both caretaker and patient app, but also both apps can be used from different mobile devices.
- **GRDB Database** – Initially before moving on to Firebase we planned to use in phone's persistent storage to be able to store data and share them between two apps. But due to the limitation in iOS, it was not possible. So we had to move to Firebase. But the GRDB database support is still there which can be used in future implementations to support offline usage.

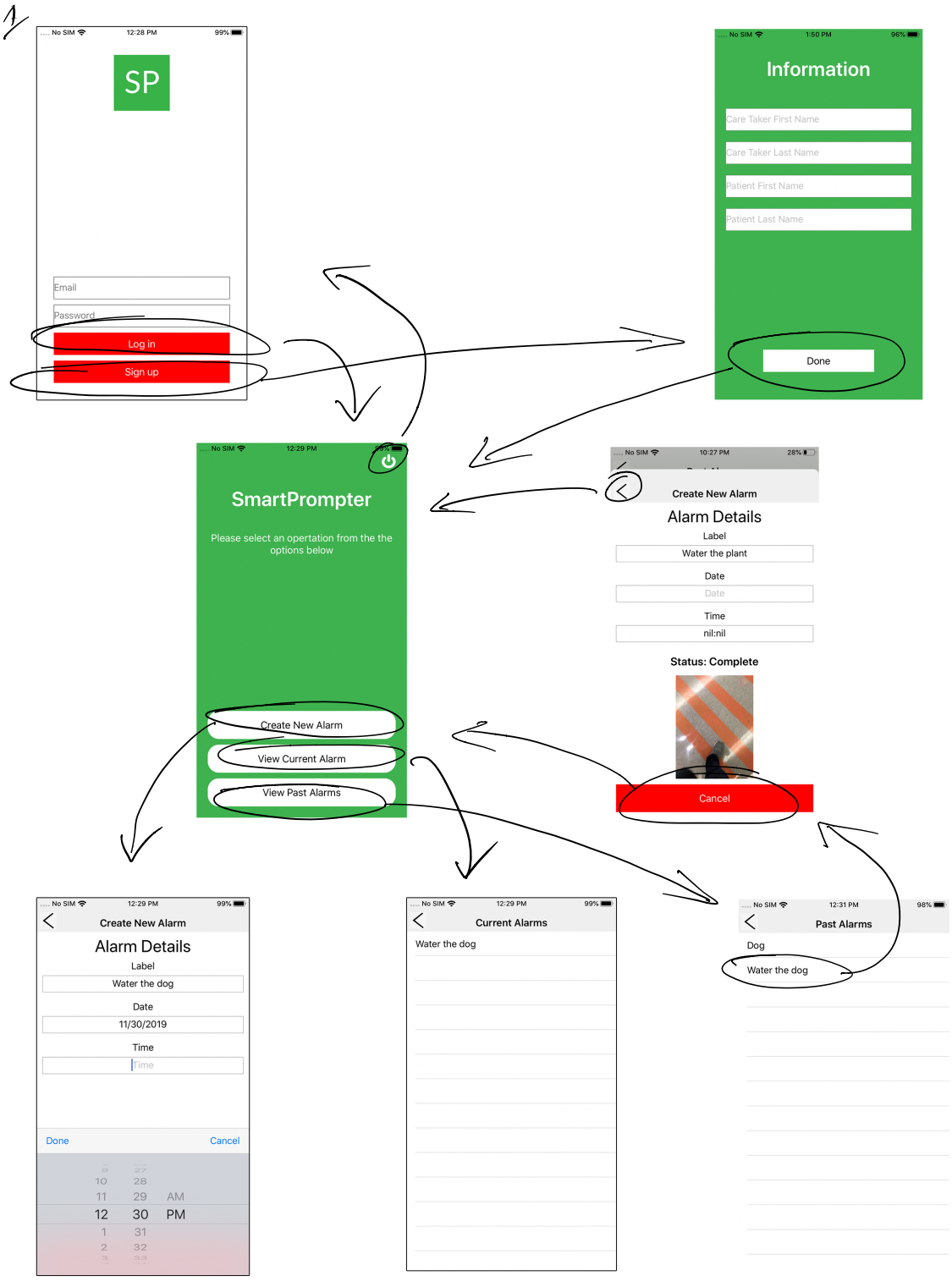
Known Issues & Backlogs

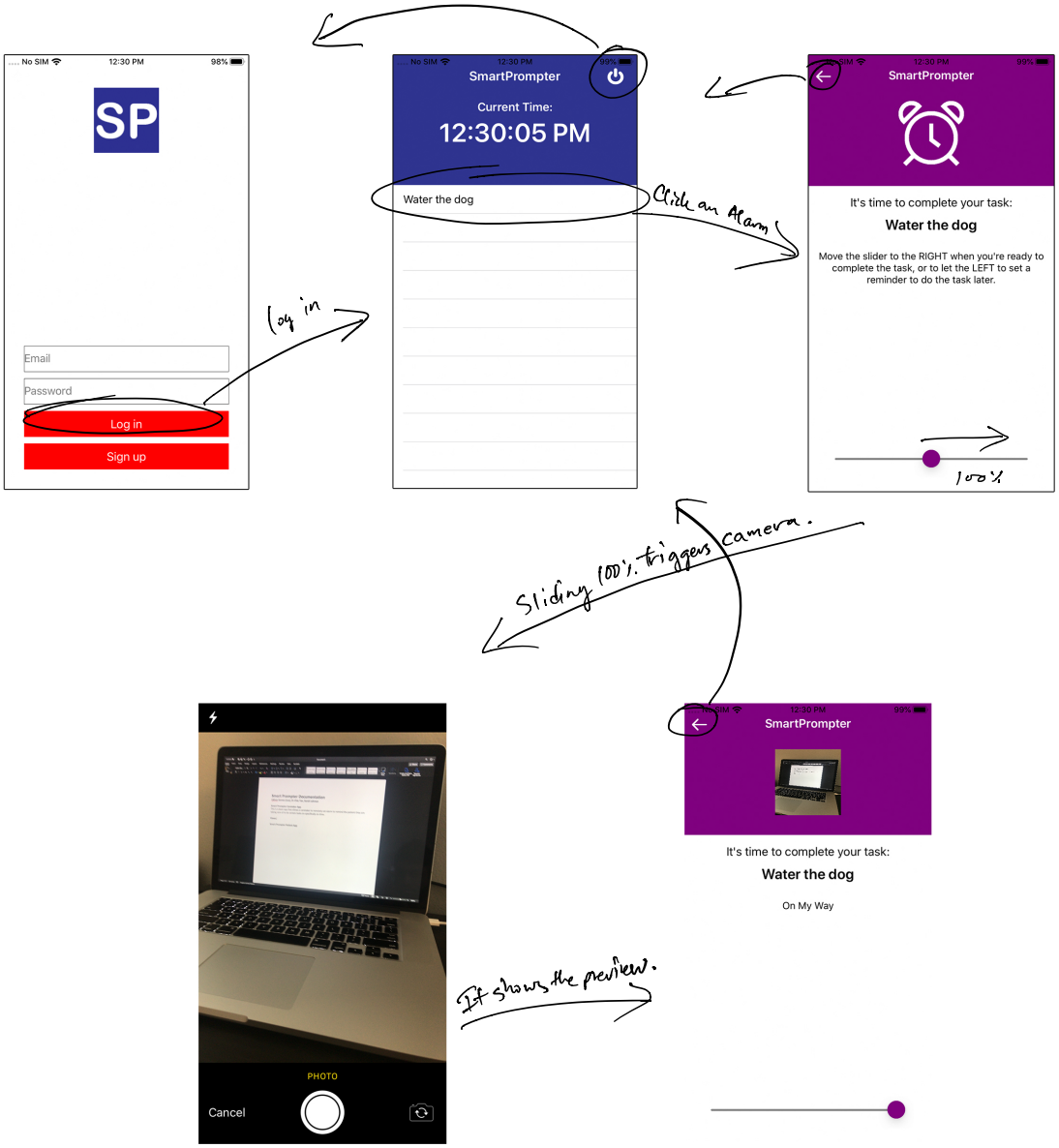
There are some bugs throughout the app, that I have noted down. Most of them are some minor bugs and can be fixed easily. They were found while writing this documentation

- Sign In Keyboard Glitch – when the user taps on the text field in SignIn Page, the screen raises up in order to accommodate the keyboard space, it's fine to have it for the text fields that are within keyboard space, but the texts fields above they keyboard are raised as well, therefore they end up raising too height out of the bounds of the keyboard.
- Notification glitch – On the android app, whenever a notification is fetched, it repeats until the user responds to the alarm, but on the iOS side the scenario is a bit different, while the app is able to deliver the notification, and repeat when the app is in the foreground. But the repetition is not available while the app is in the background, that's because of how iOS handles their app scheduler. The app is completely as sleep, therefore I am not able to schedule a repeating function. Although iOS has two notification scheduling APIs, one being UNCalendarNotification (this one delivers the notification on a specific date and time) and the other one is UNIntervalNotification (this one delivers notification after a specific interval from the current time). I have successfully used both of these API to deliver notification while it's running in the foreground, but because the doesn't run when in background, I am not able to execute the code to schedule UNIntervalNotification. Now, one other solution is to schedule multiple Notification after 1 minute of each other, which is possible and it works but the problem is, iOS doesn't have viable api to add minutes to a certain date & time. You can only add minutes to current date and time. Therefore, I haven't solely implemented this method because this leaves the app prone to crashing whenever the time is miscalculated, thus rendering the patient data useless. I am still doing my research, trying to find a safe way to solve this problem.
- Null Data Check – There are some viewcontrollers where the data are not being checked before uploading to firebase, therefore NIL data are being uploaded if nothing is

entered in the text field. I have to implement character checking before uploading the data to Firebase.

App Flows


No SIM 12:28 PM 99%
SP
Email
Password
Log in
Sign up
No SIM 1:50 PM 96%
Information
Care Taker First Name
Care Taker Last Name
Patient First Name
Patient Last Name
Done
No SIM 12:29 PM
SmartPrompter
Please select an opertation from the the options below
Create New Alarm
View Current Alarm
View Past Alarms
No SIM 10:27 PM 28%
Create New Alarm
Alarm Details
Label
Water the plant
Date
Date
Time
nil:nil
Status: Complete
Cancel
No SIM 12:29 PM 99%
Create New Alarm
Alarm Details
Label
Water the dog
Date
11/30/2019
Time
Time
Done
Cancel
10 28
11 29 AM
12 30 PM
1 31
2 32
No SIM 12:29 PM 99%
Current Alarms
Water the dog
No SIM 12:31 PM 98%
Past Alarms
Dog
Water the dog


No SIM
12:30 PM
98%
SP
Email
Password
Log in
Sign up
log in
No SIM
12:30 PM
99%
SmartPrompter
Current Time:
12:30:05 PM
Water the dog
Click an Alarm
No SIM
12:30 PM
99%
SmartPrompter
It's time to complete your task:
Water the dog
Move the slider to the RIGHT when you're ready to complete the task, or to let the LEFT to set a reminder to do the task later.
100%
Sliding 100%. triggers camera.
PHOTO
Cancel
It shows the preview.
No SIM
12:30 PM
99%
SmartPrompter
It's time to complete your task:
Water the dog
On My Way