Technical Manual

# 

# 

[**A description of the general structure of the software**](#_4nolaagjrd4x) **3**

[ViewController](#_sq4j8s19hr3y) 13

[StatisticsPage](#_ofsdjch0hniv) 13

[MissionListCell.swift](#_8e5a7decvw9) 13

[StatisticsOverviewPage.swift](#_zh0bkhh7ooyy) 14

[StatisticsDetailsPage.swift](#_hjcfswl2e0q0) 15

[MissionCardCell.swift](#_lfrib4jz4iee) 16

[GoalCounterPage](#_e3znaz4cvv4a) 16

[GoalCounterPage.swift](#_c5mf9l2nxmc4) 16

[GoalCounterSettingsPage.swift](#_vna6ynjt5596) 19

[NormalCounterPage](#_6wtrh52386wy) 23

[NormalCounterPage.swift](#_91cby4o6mfl4) 23

[File Generation](#_mgxwxp51dw9c) 32

[Contents of Files](#_a8j35htgzr6a) 33

[Compiling](#_euao6eiarxnl) 35

[Development tools](#_43sosvqg8l4w) 35

[BluefruitConfig.h](#_33wf757zlx4s) 36

[Rehabilitation\_buttons.ino](#_ywc6z7hjwvf) 36

# 

# A description of the general structure of the software

The general structure of our application follows a Model-View-Controller concept, in which the user interacts with a controller, which then makes changes within different models in the code. Finally these changes are displayed to the user by updating different Views. In terms of physical navigation of the application, the user is simply able to move between the main counter screen, the goal counter screen, the goal setting screen, and the user profile screen.

The workflow of the application and its Model-View-Controller architecture works as follows:

1. The user interacts with the view (e.g: main counter screen)
2. The view alerts the controller of a particular event (the user presses a button)
3. The controller updates the model (presses are recorded for statistics)
4. The model alerts the view that it has changed
5. The view grabs the models data and updates itself (counter increments)

The following three diagrams are from the agile phase 1 report and show the initial design methodology that the project team has continued to develop by.

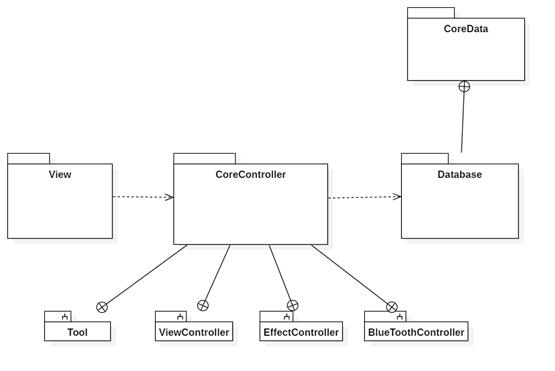
****

Figure 1. the controller UML diagram used for application design initiation

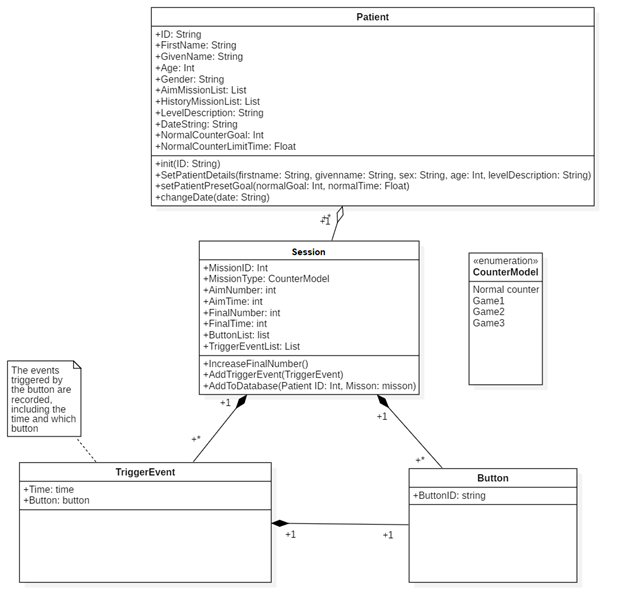
****

Figure 2. Class UML diagram of data storage solution

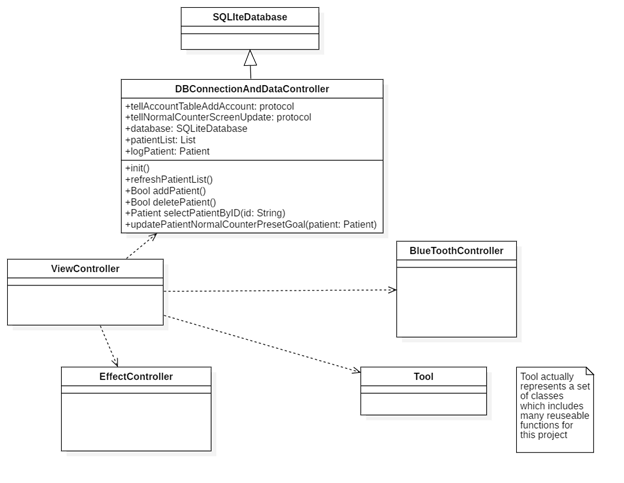
****

Figure3. Database controller and connection

**APP PROGRAM**

**Source:**[**https://github.com/yinzixie/Stroke-Rehabilitation**](https://github.com/yinzixie/Stroke-Rehabilitation)

**AppDelegate.swift**

Save BLEConnection page in the background, monitor exit and enter app actions, and write data into records.

**Tool**

The Tool folder is an auxiliary function catalogue that includes three main .swift files. They are Alert.swift, TimeInfo.swift and TimerFormattor.swift.

**Alert.swift**

This file contains the Alert {} class which defines three different alert types in three functions; a warning alert, a message alert and a “yes” or “no” alert:

static func warningAlert()

This alert will appear when a user performs an incorrect operation. For example, when they are using the application, if they press the “Trigger” button before the “Arm” button has been pressed, the application will pop-up a warning window to tell them to press the “Arm” button first.

static func messageAlert()

This alert shows a message. For example, the user can set a time and goal before they start practising, and when the time or goal has been achieved, the application will pop-up a message to show them that they have timed out or that their goal has been achieved.

static func yesOrNoAlert()

This creates a pop-up Yes or No window.

**TimeInfo.swift:**

This file contains the TimeInfo{} class and contains functions that get, set and format times and dates:

static func currentYear()

Use this function to get the current year.

static func currentMonth()

Get the current month.

static func currentDay()

Get the current day.

static func currentWeekDay()

Get the current day of the week.

static func countOfDaysInMonth()

Get how many days there are in the current month.

static func getCountOfDaysInMonth()

Get how many days there are in a given month.

static func getFirstWeekDayInMonth()

Get the first week day in a given month.

static func dateToDateString()

Convert a date to a string.

static func dateStringToDate()

Convert a date string of the format “yyyy-MM-dd” to a date format.

static func timeStringToDate()

Convert a time string of the format “yyyy-MM-dd HH:mm:ss” to a date format.

static func dateDifference()

Calculate discrepancy between two dates.

static func compareOneDay()

Compare two dates to see which one comes first.

static func stringToTimeStamp()

Convert time string of the format “yyyy-MM-dd HH:mm:ss” to a timestamp.

static func timeStampToString()

Convert timestamp to “yyyy-MM-dd” string format.

static func timeStampToStringDetail()

Convert timestamp to a time string of the format “yyyy-MM-dd HH:mm:ss”.

static func timeStampToHHMMSS()

Convert timestamp to “HH:mm:ss” string format.

static func getStamp()

Get the current timestamp of the system.

static func numberToChina()

Convert month number to Chinese characters.

static func add0BeforeNumber()

Add 0 before number.

static func compareCurrentTime()

Show the time as ~ minutes ago, ~ hours ago, ~ days ago.

static func secTransToHourMinSec()

Convert seconds to an “HH:MM:SS” string and add a zero in front of each number if it is less than 10.

**TimerFormattor.swift:**

This file contains the TimerFormatter{} class and only has one function, an initialiser for formatting the timer.

**Widget**

The UI folder contains 7 classes which provide unique views and effects. They were found from online resources. All included classes follow the MIT license. They are as follows:

**Spring**

Contains Animations for the UI which spring UI elements in as views are created.

From: <https://github.com/MengTo/Spring>

**AFBlurSegue**

Segue with a blur effect.

From: <https://github.com/AlvaroFranco/AFBlurSegue>

**CalendarView**

Monthly calendar widget view.

From: <https://github.com/mmick66/CalendarView>

Changes:

*CalendarDayCell.swift:*

1. *Added code from line 43-53 to show or hide dots on the calendar if exercises were performed on that day.*

*CalendarView+DataSource.swift :*

1. *Added code at line 61*
2. *Comments line 62*
3. *Added code from line 64-69*
4. *Added code from line 174-179*

**SCLAlertView**

Alerts with different styles used to create larger customised dialogues.

From: <https://github.com/vikmeup/SCLAlertView-Swift>

**CardStackView**

Card stack view used for therapy session details.

From: <https://github.com/gomfucius/CardStackView>

**ZFRippleButton**

Button with some ripple effects.

From: <https://github.com/zoonooz/ZFRippleButton>

**LTMorphingLabel**

Label with some animation effects.

From: <https://github.com/suzuki-0000/CountdownLabel>

**UICircularProgressRing**

Circle progress ring UI used in goal exercise screen and therapy session details.

From: <https://github.com/luispadron/UICircularProgressRing>

**Extension:**

This file contains classes that extend existing classes, including Time, String, UIView, Image and ViewController.

**ExtensionDate.swift:**

This file contains the Date{} extension, extending the Date{} class and adds the following functions:

func year\_()

Gets the current year.

func month\_()

Gets the current month.

func day\_()

Gets the current day.

func weekDay\_()

Gets what day of the week the current day is. For example, today is Tuesday.

func countOfDaysInMonth()

Gets the number of days in the current month.

func firstWeekDay()

Gets the first day of the current month, and what day of the week that day is.

func isToday()

Checks if the given day is today or not.

func isThisMonth()

Checks if the given day occurs in the current month or not.

**ExtensionString.swift:**

This file contains the Date{} extension, extending the Date{} class and adds the following functions:

func containsVersion2()

Compares strings.

func containsIgnoringCase()

Search by string and ignore case of string.

**ExtensionUIView.swift:**

This file is about view setting and style, such as border setting, card view, or hide cards.

func cardView()

Card view style and layer, such as background color, radius, and style of shadow.

func removeCardView()

Remove card and its’ styles, such as background color and shadow styles.

func SetMutiBorderRoundingCorners()

This function for setting border rounding corners of UIView, that includes such as top-right, top-left, bottom-right, bottom-left and so on.

func SetBorder()

The purpose of this function is to set the border of UIView, that includes frame, size, width and height.

**ExtensionUIImage.swift:**

This file is to set image styles, such as size, width, height and so on.

func resizeImage()

This function for generating circle image and processing the circle image.

func resizeImage()

Resize image (width and height), and return a new size image.

func imageByRemoveWhiteBg()

Return a UIImage and change the background color from white to transparent.

func imageByRemoveBlackBg()

Return a UIImage and change the background color from black to transparent.

func transparentColor()

Return a UIImage and change the background color from a specified color to transparent.

**ExtensionUIViewController.swift:**

This file is about controlling the on-screen keyboard as well as other contact with the keyboard.

func hideKeyboardWhenTappedAround()

When the user taps on white space in the application, the on-screen keyboard will hide.

func dismissKeyboard()

Close the keyboard.

**Resource:**

**Sounds:**

This file includes 2 sounds that correspond with the “Arm” and “Trigger” buttons.

**CoreStruct:**

This folder includes different data structures used to define objects and events that occur as the application runs. It contains the following files:

**Button.swift:**

Allows buttons to be defined with a string identifier.

Definition:

ButtonID is “String”.

**Patient.swift:**

Data structure that defines a patient and stores data about them and their recorded therapy sessions.

Definition:

Patient ID is “String”

Patient Name is “String”

Normal counter goal is “Int”

Normal counter time is “Int”

**UserDefaultKeys.swift:**

Some default values stored as UserDefaultKeys strings.

**ButtonTriggerEvent.swift:**

Data structure that defines a button trigger event and stores information about the event, such as the patient, the session ID, a timestamp and the button that was pressed.

**NormalCounterMission.swift:**

Data structure that defines a therapy session and stores information about it. Contains functions to set its values.

Definition:

Mission ID is “Int”

Aim number is “Int”

Aim time is “Int”

Final Number is “Int = 0”

Initialize all user mission values, such as AimGoal = 0, AimTime = 0

## **ViewController**

The ViewController folder contains all of the visual aspects of the application as well as the logic that controls those views. All user input and output comes through the ViewController classes and is processed and presented to the user. This is in accordance with the MVC design pattern that Swift development typically utilises.

### StatisticsPage

This folder contains the classes that manage the statistics page of the application. It is accessible from the user login page after a user has logged in.

#### MissionListCell.swift

Contains the MissionListCell{} class which displays and manages a list of exercise session information of the logged in user. It extends the UITableViewCell{} class. This class is no longer in use and has been replaced by the StatisticsDetailsPage{} class. The MissionListCell{} class contains the following functions:

override func awakeFromNib()

Initialises the UITableViewCell{} class and any required parent classes to create the view.

func setLabels()

Fills all labels in the view with relevant data about the selected exercise session. This data is taken from the NormalCounterMission{} class which contains information about an exercise session. Timestamps are converted to readable strings using functions from the TimeInfo{} class.

#### StatisticsOverviewPage.swift

Contains the StatisticsOverviewPage{} class which extends the UIViewController{} class and shows the calendar popup after the user presses the Show Statistics button in the user profile page. The user can select days on the calendar and view summarising information about therapy sessions performed on that day. The StatisticsOverviewPage{} class contains the following functions:

override func viewDidLoad()

Initialises the view and any required parent classes when the statistics overview page is opened. Initialises and formats the CalendarView{} class and the views contained within.

override func viewDidAppear()

Overrides the standard viewDidAppear() function of the UIViewController{} class to set the displayed date on the calendar to today if no session data has been added for the selected day.

func setStatisticOverViewInformation()

Calculates all relevant information to be displayed on the calendar view. Calculates and fills the labels at the bottom of the card and determines the colour and direction of the arrows. If the selected day contains no exercise information, the labels are set to zero and the arrows are coloured red. This function executes whenever the calendar is opened or changed.

@IBAction func onValueChange()

Updates the displayed date when a new date is selected.

@IBAction func goToPreviousMonth()

Updates the calendar display to the previous month when the previous month is selected via a swipe action on the calendar.

@IBAction func goToNextMonth()

Updates the calendar display to the next month when the next month is selected via a swipe action on the calendar.

@IBAction func back()

Dismisses the calendar view when the user taps outside of the calendar view.

override func prepare()

Transfers the currently selected day’s session details to the StatisticsDetailsPage{} class, when its segue is called.

func startDate()

Returns a Date value which is used to limit the range of the calendar. Currently set so the calendar begins in March 2019.

func endDate()

Returns a Date value which is used to limit the range of the calendar. Currently set so the calendar ends two months from the current date.

func calendar()

Called whenever the calendar is opened, is refreshed or a date is selected. Pulls session information for the selected day from the DBAdapter{} class and calls the setStatisticOverViewInformation() function. Disables the “Show Details” button if there is no exercise data for the selected day.

#### StatisticsDetailsPage.swift

Contains the StatisticsDetailsPage{} class which extends the UIViewController{} class and creates the page which shows extensive details about the therapy sessions that occurred on the selected day. This class is called when the user taps the “Show Details” button in the Statistics Overview Page. This view contains swipeable cards for each of the sessions that were performed on the selected day, as well as a calendar widget that shows the selected date. Each card shows a range of details for that session. The StatisticsDetailsPage{} class contains the following functions:

override func viewDidLoad()

Initialises the view and any required parent classes when the statistics details page is opened. Loops through every session for the selected day, calculates the required values to be displayed on that day’s card, formats and creates a card and appends it to an array of UIViews. The function then creates a CardStackView() from the UIViews array and displays it. Finally, the accompanying calendar widget is generated by calling the setCalendarCard() function.

override func viewDidAppear()

Adds animations to the swipeable cards and the calendar widget.

private func setCaledarCard()

Formats the calendar widget, adds the date of the selected sessions and displays it on the screen.

@IBAction func closePage()

Dismisses the statistics details page when the Close button is tapped.

func getLabelWidth()

Returns a label width value appropriate for the length of the input string and font size. Used to create correctly sized labels for the statistics details cards.

#### MissionCardCell.swift

Contains an unused and empty class MissionCardCell{} which extends UICollectionViewCell{}.

### GoalCounterPage

Contains all classes that directly manage the Goal Counter screen and its settings page. This folder contains 2 files, GoalCounterPage.swift and GoalCounterSettingPage.swift.

#### GoalCounterPage.swift

Contains the GoalCounterPage{} class which extends the UIViewcontroller{} class; and creates and manages the Goal Counter page in the application. It holds variables to manage all labels and required values for this page. This class calls functions which initialise Bluetooth and Audio functionality. The GoalCounterPage{} class contains the following functions:

override func ViewDidLoad()

Initialises the view and any required parent classes when the goal counter page is opened. Prepares the “Arm” and “Trigger” buttons for use. Sets up the popup dialogues. Sets up goal reps and goal timers if they have been set.

override func viewDidAppear()

Updates labels whenever the page is reaccessed.

override func viewDidLayoutSubviews()

Resets the gradient colour button feedback when the page is reaccessed.

func runTimer()

Updates the updateTimer() function function every second. Called when the session begins.

func resetTimer()

Ends the running timer and re-initialises it.

@objc func updateTimer()

Increments or decrements the timer value depending if a timer was set and updates the label. Updates the circular progress bar with timer info if no goal was set. Ends the session when the counter hits zero.

func arm()

Arms the counter. If it is the first time in the session the arm button was pressed, starts the session.

func trigger()

Increments or decrements the counter and updates the label if Arm has already been pressed. Updates the circular progress bar if a goal number of reps has been set. Resets Arm, ready for use again. Ends the session and shows a dialogue if the goal number of reps reaches zero.

@objc func pressArmButton()

Function runs when on-screen arm button is first touched before release, or Bluetooth arm button is first pressed before release. Arms the button, plays a sound, runs the colour gradient feedback animation and appends the arm event to a list of events.

@objc func releaseArmButton()

Clears the colour gradient feedback from the screen once the arm button is released.

@objc func pressTriggerButton()

Function runs when on-screen trigger button is first touched before release, or Bluetooth trigger button is first pressed before release. Triggers the button, plays a sound, runs the colour gradient feedback animation and appends the trigger event to a list of events.

@objc func releaseTriggerButton()

Clears the colour gradient feedback from the screen once the trigger button is released.

@IBAction func endMissionButtonTrigger()

Runs when the END TASK button is tapped. If a session is in progress, show a confirmation dialogue before ending the session.

func missionStart()

Starts the session. Called on the first “Arm” press of the session. Hides unneeded labels and buttons from the view. Starts the timer and logs the time that the session began.

func missionFinished()

Ends the session. Called when the timer runs out, the goal reps has been met, or END TASK has been tapped. Reveals the hidden labels and buttons. Resets the timer and labels back to their originally set values (zero or a preset goal). Stores session data in the database.

@IBAction func backPreviousPage()

Closes the view with an animation when the back button is pressed.

override func prepare()

Transfers the set goal from the GoalCounterSettingsPage{} class to this page and updates the session data, when its segue is called.

func updateMission()

Resets and prepares for a new session using currently set data parameters. Resets labels and progression bars, gets session and patient data from the database. Sets session parameters based on whether or not time and rep goals have been set.

|  |
| --- |
| func updateIncomingData() |

Listens for incoming Bluetooth data and triggers the “Arm” or “Trigger” touch down and touch up events when the appropriate Bluetooth button is pressed and released.

func peripheralManagerDidUpdateState()

Checks to see if the Bluetooth peripheral device manager is working and if so, prints a message to the console.

func peripheralManager()

Checks if a device is connected to the peripheral manager.

func peripheralManagerDidStartAdvertising()

Checks if the Bluetooth peripheral has successfully started advertising its data and prints an error to the console if it has not.

func enterPageAnimation()

Called when the view is loaded. Sets up and performs animations on the buttons and views when the page is opened.

func leavePageAnimation()

Called when the view is dismissed. Sets up and performs animations on the buttons and views when the page is closed.

func changeBackgroundColor()

Called when the “Arm” and “Trigger” buttons are pressed down. Sets up and performs the gradient colour button feedback animations on the left or right side of the screen, depending on the button pressed.

func removeBackgroundColour()

Called when the “Arm” and “Trigger” buttons are released. Clears the gradient colour button feedback animations.

override var preferredStatusBarStyle

Keeps the status bar a light colour to allow it to blend in.

@objc func minimiseView()

Animates the goal counter page and shrinks it slightly when the goal settings page is opened.

@objc func maximiseView()

Animates the goal counter page and expands it back to full size when the goal settings page is closed.

#### GoalCounterSettingsPage.swift

Contains the GoalCounterSettingsPage{} class which presents and manages the settings page view of the goal counter screen. It holds variables to manage all labels and required values for this page. The GoalCounterSettingsPage{} class contains the following functions:

override func viewDidLoad()

Initialises the view and any required parent classes when the goal settings page is opened. Gets rep and time goal values, displays them, and shows or hides settings if they have previously been set.

override func viewDidAppear()

Animates the settings screen and the goal counter screen when the goal settings screen is opened.

@IBAction func hideGoalArea()

Hides the rep goal value and setting buttons when the rep goal settings switch is tapped off.

@IBAction func hideTimeArea()

Hides the time goal value and setting buttons when the time goal settings switch is tapped off.

@IBAction func negateFive()

Reduces the rep goal value by five, but not less than zero. Updates the rep goal label. Called when the -5 reps button is tapped.

@IBAction func negateTen()   
Reduces the rep goal value by ten, but not less than zero. Updates the rep goal label. Called when the -10 reps button is tapped.

@IBAction func negateFifteen()   
Reduces the rep goal value by fifteen, but not less than zero. Updates the rep goal label. Called when the -15 reps button is tapped.

@IBAction func negateTwenty()   
Reduces the rep goal value by twenty, but not less than zero. Updates the rep goal label. Called when the -20 reps button is tapped.

@IBAction func goalReset()   
Resets the rep goal value to zero. Called when the rep goal reset button is tapped.

@IBAction func incrementFive()   
Increases the rep goal value by five. Updates the rep goal label. Called when the +5 reps button is tapped.

@IBAction func incrementTen()   
Increases the rep goal value by ten. Updates the rep goal label. Called when the +10 reps button is tapped.

@IBAction func incrementFifteen()   
Increases the rep goal value by fifteen. Updates the rep goal label. Called when the +15 reps button is tapped.

@IBAction func incrementTwenty()   
Increases the rep goal value by twenty. Updates the rep goal label. Called when the +20 reps button is tapped.

@IBAction func resetTimer()   
Resets the time goal value to zero. Called when the timer goal reset button is tapped.

@IBAction func plusTenSecs()   
Increases the time goal value by ten seconds. Updates the time goal label. Called when the +10s button is tapped.

@IBAction func plusOneMins()   
Increases the time goal value by sixty seconds. Updates the time goal label. Called when the +1m button is tapped.

@IBAction func plusTenMins()  
Increases the time goal value by ten minutes. Updates the time goal label. Called when the +10m button is tapped.

@IBAction func minusTenSecs()   
Reduces the time goal value by ten seconds, but not less than zero. Updates the time goal label. Called when the -10s button is tapped.

@IBAction func minusOneMins()   
Reduces the time goal value by sixty seconds, but not less than zero. Updates the time goal label. Called when the -1m button is tapped.

@IBAction func minusTenMins()   
Reduces the time goal value by ten minutes, but not less than zero. Updates the time goal label. Called when the -10m button is tapped.

@IBAction func finishSetGoal()

Saves the session’s data and stores it in the database. Updates the goal counter page with the set parameters. Dismisses the goal settings view. Called when the “Save” button is tapped.

@IBAction func cancel()

Discards inputted values and dismisses the goal settings view. Called when the “Cancel” button is pressed.

**BLEConnectionPage:**

This file is concerned with connecting Bluetooth to the application, that includes refreshing Bluetooth list, scanning for Bluetooth devices, connecting to Bluetooth and so on. The following section describes details about those functions.

**BLEConnectionPage.swift:**

Follow the instructions from:<https://learn.adafruit.com/crack-the-code/overview>

@IBAction func refreshAction()

Refresh the Bluetooth list when the ‘refresh’ button is tapped.

override func viewDidLoad()

Connects the buttons with the application, and checks which button is “Arm” and which is “Trigger” by reading its sensor ID.

override func viewDidAppear()

Connect button with application via Bluetooth. Once the application detects the Buttons via Bluetooth, you may tap it to connect.

func startScan()

Application starts scanning nearby Bluetooth devices.

@objc func cancelScan()

Stop scanning for Bluetooth devices and display a list of available peripherals.

func refreshScanView()

Refresh nearby Bluetooth devices.

func disconnectFromDevice()

Disconnect the Bluetooth connection between the application and the connected device.

func restoreCentralManager()

Refresh discovered peripherals.

func centralManager()

Manage the Bluetooth peripheral list by refreshing the list when a peripheral is either discovered or lost.

func connectToDevice()

Connect to selected peripheral devices.

func centralManager()

After connecting to the device, display a message to notify the user, and display peripheral information.

func centralManager()

If unable to connect to a peripheral device, then display the message “failed to connect to peripheral”.

func disconnectAllConnection()

Disconnect all connected peripheral devices.

func peripheral()

Show details about the connected peripheral device; such as name, date and connection state.

func tableView()

Utilise the table view style for the peripheral list.

func centralManagerDidUpdateState()

Check to see if Bluetooth is currently enabled on the device.

@IBAction func swapID()

Swap ID between “Arm” button and “Trigger” button and swap sensor0 to sensor1.

**PeripheralListCell.swift:**

This file is about peripheral list.

override func awakeFromNib()

Initialization code.

override func setSelected()

Configure the view for the selected state.

### NormalCounterPage

Contains the class that directly manages the Normal Counter screen. This folder contains a single file, NormalCounterPage.swift.

#### NormalCounterPage.swift

Contains the NormalCounterPage{} class which extends the UIViewController{} class; and creates and manages the Normal Counter page in the application. It holds variables to manage all labels and required values for displaying this page. This class calls functions which initialise Bluetooth and Arduino functionality. The GoalCounterPage{} class contains the following functions:

override func viewDidLoad()

Initialises the view and any required parent classes when the normal counter page is opened. Initialises the database. Prepares the “Arm” and “Trigger” buttons for use. Sets up the popup dialogues.

func super\_init()  
Calls the function that initialises the database, DBAdaper().

override func viewDidAppear(\_ animated: Bool)   
Sets the user icon label to the currently logged in patient. Animates the transition to the page. Occurs whenever the page is reaccessed.

override func viewDidLayoutSubviews()  
Resets the gradient colour button feedback when the page is reaccessed.

@IBAction func bluetoothSettings(\_ sender: Any)

Opens the Bluetooth settings page when the Bluetooth button is tapped.

func setTimer()

Prepares and formats the on-screen counter. Calls scheduledTimer() which runs a function every second that increments the counter. Called when the session begins.

@objc func pressArmButton()

Function runs when on-screen arm button is first touched before release, or Bluetooth arm button is first pressed before release. If it is the first time in the session the arm button was pressed, start the session. Arms the button, plays a sound, runs the colour gradient feedback animation and appends the arm event to a list of events.

@objc func releaseArmButton()

Clears the colour gradient feedback from the screen once the arm button is released.

@objc func pressTriggerButton()

Function runs when on-screen trigger button is first touched before release, or Bluetooth trigger button is first pressed before release. Triggers the button, plays a sound, runs the colour gradient feedback animation and appends the trigger event to a list of events.

@objc func releaseTriggerButton()

Clears the colour gradient feedback from the screen once the Trigger button is released.

@objc func addTime()

Adds one second to the timer and updates the label. Called every second by scheduledTimer().

func arm()

Arm the counter.

func trigger()

Increments the counter and updates the label if Arm has already been pressed. Resets Arm, ready for use again.

@IBAction func endTask()

Ends the session. Called when the END TASK button is tapped. Shows a confirmation dialogue before ending the session.

@IBAction func goToUserPage()

Animates the transition from the normal counter page to the user list and login page. Called when the user icon on the top left of the screen is called.

@IBAction func goToGoalCounterPage()

Ends the currently running therapy session and animates the transition from the normal counter page to the goal counter page. Called when the arrow button is tapped.

override func prepare()

Unused segue function used to transfer data between this view and other views.

func updateMission()

Resets values and prepares for a new session. Called when the page is opened and when a session is ended.

func MissionStart()

Starts the session. Begins the timer and logs the time that the session began. Called when the “Arm” button is pressed for the first time.

func missionEnd()

Stops the timer. Logs the time length of the session and the number of completed reps. Updates the database with the new data and confirms it with a dialogue. Resets the screen ready for a new session.

func enterPageAnimation()

Called when the view is loaded. Sets up and performs animations on the buttons and views when the page is opened.

func leavePageAnimation()

Called when the view is dismissed. Sets up and performs animations on the buttons and views when the page is closed.

func changeBackgroundColor()

Called when the “Arm” and “Trigger” buttons are pressed down. Sets up and performs the gradient colour button feedback animations on the left or right side of the screen, depending on the button pressed.

func removeBackgroundColour()

Called when the “Arm” and “Trigger” buttons are released. Clears the gradient colour button feedback animations.

|  |
| --- |
| func updateIncomingData() |

Listens for incoming Bluetooth data and triggers the “Arm” or “Trigger” touch down and touch up events when the appropriate Bluetooth button is pressed and released.

func peripheralManagerDidUpdateState()

Checks to see if the Bluetooth peripheral device manager is working and, if so, prints a message to the console.

func peripheralManager()

Checks if a device is connected to the peripheral manager.

func peripheralManagerDidStartAdvertising()

Checks if the Bluetooth peripheral has successfully started advertising its data and prints an error to the console if it has not.

func adaptivePresentationStyleForPresentationController()

Sets the presentation style to none when opening the Bluetooth settings page.

**UserListAndLoginPage:**

This file is about user management, including user login, new user registration, user settings and user list management. It records information about users, such as how long they spend performing therapy and how many sessions they performed.

**UserListCell.swift:**

User list and that includes user ID, user name and user image.

override func awakeFromNib()

Initialization code

override func setSelected()

Configure the view for the selected state.

**AddNewUserCell.swift:**

Clicking the “+” symbol will add a new user, with basic information such as; User Name and User ID.

override func awakeFromNib()

Initialization code.

override func setSelected()

Configure the view for the selected state.

**UserSettingPage.swift:**

This file is about user settings, and deleting users.

override func viewDidLoad()

Do any additional setup after loading the view.

@IBAction func switchTrigger()

Turn on and turn off delete protection.

**UserListAndLoginPage.swift:**

User list and login page layout.

override func viewDidLoad()   
Set the visual layout of the user list. Includes size of button, text, title and window. User list table colour, width and height.

func setMilestone()

Calculates and displays information about the currently logged in user’s session history.

func login()

Login page, connects with the internal database.

@IBAction func loginAs()

Login as current user.

@IBAction func backPreviousPage()

Back to previous page.

func tableView()

Setting user table layout.

func deleteAction()

Delete current user.

@objc func addNewUser()

Clicking the “+” symbol will add a new user, allows the user to enter a new users User Name and User ID.

**EffectController:**

This file handles the auditory and visual feedback within the application.

**AudioEffectController.swift:**

Plays the applications sounds. When the user presses the “Arm” and “Trigger” buttons, the application will play different sounds.

**ColorEffectController.swift:**

Colours will display when “Arm” and “Trigger” buttons are pressed.

**BluetoothController:**

Bluetooth connection controller.

**UUIDKey.swift:**

Defines string identifiers used to set up and connect Bluetooth.

**BLEAdapter.swift:**

Check button connection state.

func refreshDevice()

Refresh nearby Bluetooth devices.

static func checkValue()   
Check Bluetooth value.

static func setArmID()

Check Arm button sensor ID.

static func setTriggerID()

Check Trigger button sensor ID.

**DatabaseController:**

Use sqlite3 to store users’ information

**SQLiteDatabase.swift:**

Create a database.

private func printCurrentSQLErrorMessage()

If SQL has any error, print those error messages.

private func createTables()

Create database table.

private func dropTables()

Add names in database table, that includes “Patient”, “ButtonTriggerEvent” and “NormalCounterMission”.

func onUpdateDatabase()

Check version of database, update version of database if there is a new version.

private func createTableWithQuery()

Create table with Structured Query Language (SQL) to check if database creation was successful or not.

private func dropTable()

Check if a database table already exists.

private func insertWithQuery()

Insert a row in database table.

private func selectWithQuery()

Select an element of database table.

private func updateWithQuery()

Update value of database table.

func createPatientTable()

Create a patient table in the database, that includes ID, Name, Normal Counter Goal and Normal Counter Limit Time.

func createButtonTriggerEventTable()

Create button trigger event table in the database, that includes EventID, PatientID, MissionID, ButtonID and Trigger Time.

func createNormalCounterMissionTable()

Create normal counter mission table in database, that includes MissionID, PatientID, Aim Goal, Aim Time, Start Time, Final Achievement, and Final Time.

func insertPatient()

Add a new patient in the Patient table, that includes ID, Name, Normal counter goal and Normal counter limit time.

func deletePatient()

Delete patient information from the patient table in the database.

func selectAllPatient()

Select all patient information from the table. This will assist in changing large amounts of information.

func selectPatientByID()

Admins can search by specific patient by ID.

func updateNormalCounterPresetGoal()

Update normal counter goal in the patient table of the database.

func insertNormalCounterMission()

Insert normal counter mission in the NormalCounterMission table of the database.

func selectAllNormalCounterMissionsThroughPatientID()

Select all normal counter sessions for a PatientID ordered by StartTime.

func insertButtonTriggerEvent()

Insert value in the buttonTriggerEvent table of the database.

func selectButtonTriggerEventsThroughMissionID()

Select all from Button Trigger Events table by MissionID.

**DBAdapter.swift:**

Some basic database operations, including: refreshing patients’ list, adding new users, deleting users, select patient by ID or Name, check the user exists and so on.

static func refreshPatientList()

Refresh patient list in database.

static func refreshlogPatient()

Refresh logged in patient list in database.

static func refreshlogPatientData()

Refresh data of logged in patient in database.

static func addPatient()

Add new patient in database list.

static func deletePatientList()

Delete all patient information in the database.

static func selectPatientByID()

Select a patient in the database by patient ID.

static func selectPatientName()

Select a patient in the database by patient Name.

static func isUserIDExist()

Check if UserID exists in the database.

static func isExericeAtDay()

Check if user exercised that day.

static func numberOfCompletedMission()

Calculate the number of completed sessions.

static func numberOfReps()

Calculate the number of completed repetitions.

static func numberOfTimes()

Calculate the amount of time a user has spent exercising.

static func selectNormalCounterMissionForDay()   
Search Normal counter session from that day.

static func updatePatientNormalCounterPresetGoal()

Update patient counter goal in database.

static func insertNormalCounterMission()   
Add new normal counter session in the database.

**A description of each database table**;

The database of the application includes three main tables: the PatientTable, ButtonTriggerEvent table, and NormalCounterMission table. These tables are detailed below.

**Patient table:**

This table is used for storing patients’ information; that includes Name, ID, NormalCounterGoal and NormalCounterLimitTime. The ID and Name parameters are of STRING type, and ID is the primary key, NormalCounterGoal and NormalCounterLimitTime are integers. NormalCounterGoal is the repetition goal as set by the user. NormalCounterLimitTime stores the timer as set by the user. Both the patient and the doctor can view the progress of a patient’s therapy. For example, if the UserID is 12, Name is Tom, goal is 30 reps, and Timer is 70 seconds, the table will fill as below:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Name** | **NormalCounterGoal** | **NormalLimitTime** |
| 12 | Tom | 30 | 70 |

**ButtonTriggerEventTable:**

This table is populated by the button trigger event. It records all of a patients’ exercise using the buttons. The table’s values are EventID (STRING, PRIMARY KEY), PatientID (STRING), MissionID (INT), ButtonID (INT) and TriggerTime (INT). EventID records the button press actions, MissionID is the ID of the therapy session being undertaken, ButtonID records which button has been pressed of the two at that timestamp and TriggerTime is the UNIX time that the user started the exercise. An example is shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EventID** | **PatientID** | **MissionID** | **ButtonID** | **TriggerTime** |
| 1 | 11 | 1 | 2 | 13411234 |
| 2 | 11 | 2 | 2 | 15232390 |
| 3 | 12 | 3 | 2 | 24736254 |

**NormalCounterMissionTable:**

This table records patient therapy sessions (or Missions). It includes seven values, they are: MissionID (STRING, PRIMARY KEY), PatientID (STRING), AimGoal (INT), AimTime (INT), StartTime (INT), FinalAchievement (INT) and FinalTime (INT). MissionID and PatientID are as above in the ButtonTriggerEventTable. AimGoal tracks the goal number of reps that a user set, AimTime tracks the time goal that a user has set for the exercise, StartTime tracks the start time of therapy sessions, FinalAchievement shows the final number of repetitions that the user completed in the therapy session and FinalTime records the end time of users session. An example version of the table is shown below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **MissionID** | **PatientID** | **AimGoal** | **AimTime** | **StartTime** | **FinalAchievement** | **FinalTime** |
| 1 | 11 | 30 | 70 | 14286650 | 30 | 15226488 |
| 3 | 12 | 40 | 80 | 33201487 | 40 | 19364110 |

Database operations, such as insert, delete, select, select by ID and update exist within these tables. For the patient table, an administrator can insert new users into the table, delete users from the table, select all patients or select one or more specified patient(s) by the patients’ ID. For the NormalCounterMission table, an administrator can insert new sessions into this table or select one or more specific session(s) by the patientID of the user that completed them. For ButtonTriggerEvent table, administrators can insert a new event in the table or select specific events via MissionID.

**Generated external files**

As part of the data exportation function accessible via the settings menu of the user profile page, the application will generate two Comma Separated Values (.CSV) files each time the Export Data button is pressed. The user can then save the generated files either on the iPad’s physical memory, to cloud storage or via other means of file sharing supported by the iPad. These files contain detailed information about the selected user’s exercise session history and records of every button actuation. The function is explored in depth below:

### File Generation

Tapping on the “Export Data” button in the settings menu of the user profile page begins generating files containing information about the currently logged-in user. In the Swift code, tapping the buttons calls the @IBAAction func exportData() function in UserListAndLoginPage/UserSettingsPage.swift file. In addition to creating and displaying a progression notification, the function initially creates two string variables which eventually become the filenames of the two .CSV files. These variables are initialised by getting the ID of the currently logged-in user from the DBAdapter{} class’s logPatient variable. logPatient contains all the session information for the currently logged in user, queried from the SQLite database. The currently logged in patient’s ID is then appended with the strings "\_Mission\_List.csv" or "\_Button\_Events.csv" to create the filenames for each generated file. For example, if the logged-in user’s ID is 1234, the generated file names will become 1234\_Mission\_List.csv and 1234\_Button\_Events.csv

The contents of the .CSV files themselves are created with variables that are initialised with the headings of each of the columns of the .CSV file, separated by commas. The contents of the files are then generated by appending to each variable the relevant database data contained in the DBAdapter{} class, separated by commas.

The exportData() function then writes the two string variables to a temporary path, adds the appropriate filename and extension described above, and encodes them to the UTF-8 standard. Finally, the function calls the UIActivityViewController{} class which presents on-screen a built-in iOS dialogue which allows the user to choose where to save the files

### Contents of Files

As stated above, tapping the “Export Data” button in the settings menu of the user profile page generates two .CSV files, <userID>\_Mission\_List.csv and <userID>\_Button\_Events.csv. Each file contains a different set of data associated with the logged-in user. The .CSV file was chosen because it is highly compatible as well as easily machine-readable and human-readable.

**<userID>\_Mission\_List.csv**

This file contains summarising details about every therapy session performed by the logged-in user. It contains six columns of data for each therapy session:

* **MissionID** is the key attribute in the database which lists all of the session data. It is a unique value that identifies each therapy session. It is generated by appending the ID of the user and the Unix timestamp of when the session began.
* **StartFrom** contains the Unix timestamp of the time that the therapy session began. This is also the time that the “Arm” button was first actuated.
* **FinishAt** contains the Unix timestamp of when the therapy session finishes. This is also the time the user completes their set reps, the timer runs out, or the user ends the session.
* **AimGoal** is the value representing the goal number of reps for the session set by the user. This value is zero if no rep goal is set.
* **AimTime** is the value representing the goal length of the session set by the user, in seconds. This value is zero if no time goal is set.
* **Achievement** is the number of reps that was successfully completed during the session. If AimGoal has a value > 0, the Achievement value will not exceed AimGoal. Otherwise, the Achievement value will continue to increase until the session is ended.
* **FinishTime** is the duration of the session, in seconds. If AimTime has a value > 0, the FinishTime value will not exceed AimTime. Otherwise, the FinishTime value will continue to increase until the session is ended.

The data contained in this file is the same data that is used in the application to generate summarising data about a user’s therapy activity. This generated file can therefore be used to extrapolate a great deal of information about a user’s interaction with the application.

**<userID>\_Button\_Events.csv**

This file contains data about every button press that was performed by the logged-in user before, during and after a session has begun. It contains four columns of data for each button press event:

* **MissionID** is an identifier that associates each button press event with a session. All button press events that occur during the same session will share the same MissionID. The MissionID value in this file can be used to determine which session the button press event occurred in, by cross-referencing key values in the <userID>\_Mission\_List.csv file. This value is generated by appending the ID of the user and the Unix timestamp of when the session began.
* **EventID** is a unique value that identifies every button press event. It is generated by appending the ID of the user, the Unix timestamp of when the session began, the ID of the button that was pressed, and the Unix timestamp of when that button was pressed.
* **ButtonID** is a unique identifier associated with each of the physical buttons, to record which button was pressed. Currently, there are two identifiers for each of the physical buttons; “TriggerButton” and “ArmButton” With two virtual button: “LeaveAppButton” and “EnterAppButton”. For example, when an event from the “Arm” button is registered, the ButtonID column for that event will contain “ArmButton”, when user exit app during one task session the ButtonID column for that event will contain “LeaveAppButton”.
* **TriggerTime** is the Unix timestamp of when that button press event occurred. Two button press events can have the same TriggerTime value if the two button press events occurred within the same second.

The data contained in this file is the most detailed and elemental data generated by the application. It can be extrapolated and processed to generate a great deal of information about how the user is interacting with the buttons, with a session and with the application itself. For example, information about button press speed and button press speed consistency can be generated; as well as “Arm/Trigger/Arm” consistency and accuracy.

## Compiling

As the program exists in an iPad environment, the first compile will be far more complex than future compiles. To initially load the application onto the iPad; it must first be compiled in the XCode environment, and ran as an emulated program on a physical device. This process will force application installation, and will allow the application to be used as normal for all future use.

Future application use will be as simple as selecting the icon on the iPad as normal. However, if anyone wishes to load the application on another iPad device; the same compile process must be undertaken. Hosting the application on the iOS app store would circumvent this issue, and allow the application to be used in the same way as any other.

## Development tools

Operating System: macOS Mojave 10.14

The application was developed using the latest version of macOS available at the time.

Development Environment: XCode 10.2

Using Xcode we developed the application, using the programming language Swift 5 and a little of Objective-C

iOS Version: 13

This is the earliest version of iOS that the app will work on, as certain elements of Swift 5 are not compatible with earlier versions of iOS.

**BLE PROGRAM**

**Source:**[**https://github.com/yinzixie/BLE-Button**](https://github.com/yinzixie/BLE-Button)

Simply changed from Adafruit Bluefruit LE nRF51 example file.

Website:<https://learn.adafruit.com/crack-the-code/sending-and-receiving-data-with-arduino>

## **BluefruitConfig.h**

Configuration file for Rehabilitation\_buttons.ino. A standard library that allows the arduino board to communicate via Bluetooth.

## **Rehabilitation\_buttons.ino**

bool sendSensorTriggerEvent(const String sensorID, int state)

sensorID: ID of button

State:press or release

This function decide what messages sent to device when sensor is pressed or released.

void error(const \_\_FlashStringHelper\*err)

Handles outputting error messages.

void setup(void)

Responsible for handling the initial setup of the Arduino board. Executes functions that need to run before the main loop.

void loop(void)

The main running loop of the program. Handles the boards main functions.

bool sendSensorTriggerEvent(const String sensorID, int state)

Sends the current state of the Bluetooth buttons (pressed/not pressed) to the client (iPad device).