Capstone Project 2019:

TheraComm

Prepared by

Christian John	200360001
Iden Ellia	200370502
Joe Samano	200365260

Supervised by

Dr. Timothy Maciag

1 Abstract

Having to meet up face-to-face with a therapist can be inconvenient and uncomfortable for some patients. Some patients may only have a limited amount of time to spare. As a result, patients tend to manage their issues themselves. TheraComm is an iOS-based mobile application that provides patients with a convenient way to communicate with their therapist. TheraComm incorporates the core features of an email system and a chat system. Thus, allowing the patient to not only communicate with the therapist but also have communication in real time.

Table of Contents

2 Introduction

TheraComm is an iOS-based mobile application that incorporates the core features of an email system and a chat system. Thus, allowing the patient to not only communicate with the therapist without having to meet up face-to-face, but also have the communication in real time.

2.1 Background

Depression and anxiety are common disorders among post-secondary students. Some students seek treatments by meeting up with a therapist, but many are not able due to various reasons such as limited time, concerns about privacy, uncomfortability, and many more. Furthermore, students that do seek treatments are often faced with long waiting lists and may only be able to receive a limited number of sessions. It is also important to consider that what if the student needs immediate care but therapists are not available. In order to fix this problem, a team from the psychology department called the Online Therapy Unit is providing students with transdiagnostic internet-delivered cognitive behaviour therapy (ICBT) program through a web application. This program allows students to receive access to online lessons that contain the same information and skills as meeting up with a therapist. This program provides courses which contain core lessons and assignments for student to go through. Furthermore, the students are also able to go through case stories, additional resources, and have contact to the clinicians via email or phone. By providing students with online access care, students will be able to cope with their depression and anxiety on their own.

2.2 Purpose

The purpose of TheraComm is to improve the current system that the Online Therapy Unit is using to deliver students online mental health care. TheraComm provides a user-friendly experience using the email system. TheraComm also provides an option for patients and therapists to have real-time communication through a chat system. Furthermore, using TheraComm also improves the responsiveness of therapists to patients especially during emergencies when patients need to talk to their therapist.

3 Requirement and Specifications

3.1 Project Scope

The scope of the project is to create an iOS-based application which will include the core features of an email system and a chat system. The following are the requirements for this project.

3.1.1 Email Messaging Requirements

- Present user with a login prompt (user/password)
- Using the combination above, obtain valid API token.
- Show list of messages to the client. Each message in the listing would have: subject, from/to fields, portion of the message body, urgent/system label (if applicable).
- View particular message: Show all of the above in addition to the full body of the message.
- Navigate between folders (Inbox, Sent, Drafts).
- Save a draft message
- Update draft message
- Send a message (new or one that was drafted earlier)

- Delete a message
- Display any broadcast messages
- Ability to dismiss broadcast messages (if dismiss-able).
- In general, application needs to handle errors appropriately when requests were not successful.

3.1.2 Chat Messaging Requirements

- Send Message
- Delete Message
- Chat History
- Typing Indicator
- Enable one-to-one conversation
- Encrypted traffic
- View Chat List
- Push Notifications
- Display if user is online/offline

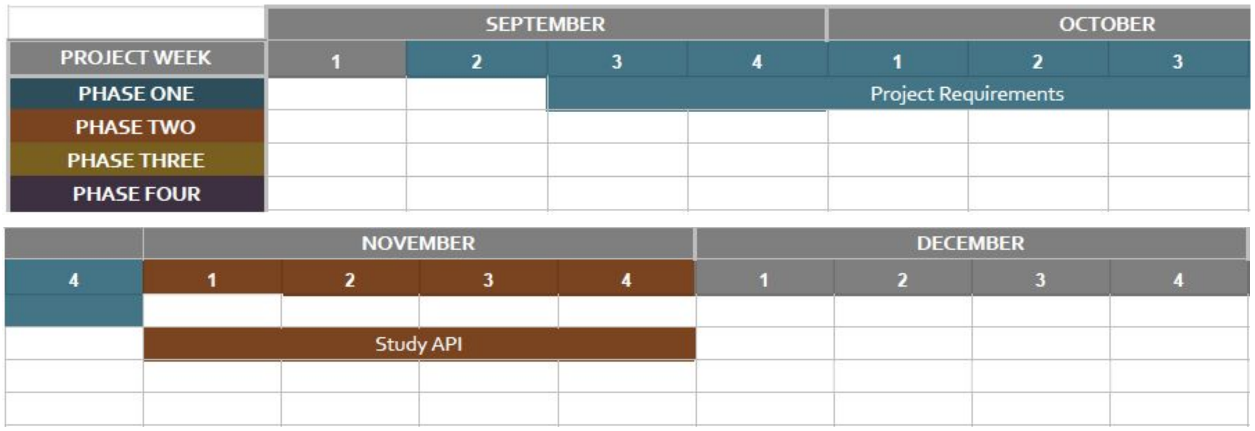
3.1.3 Additional Requirements

- Folder management (add, edit, remove) folders for client
- Message management: e.g. move messages between folders, mass delete of messages

- Show current or past program enrolments.
- Let client respond to next survey that is due (this could be very challenging requirement)
- Let client browse course materials
- Staff member can login using username/password.
- Listing of messages and folder management (as above)
- Display current caseload (list of active clients)
- Display a particular client's case file.

3.2 Project Timeline

Figure 1 below shows the general view of the project timeline. The first semester, from September to December, focuses on refining the project requirements, researching and learning technologies, and studying the Online Therapy Unit's Email API. The second semester, from January to April, focuses on development and testing.



	SEPTEMBER				OCTOBER				NOVEMBER				DECEMBER			
PROJECT WEEK	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
PHASE ONE			Project Requirements													
PHASE TWO									Study API							
PHASE THREE																
PHASE FOUR																

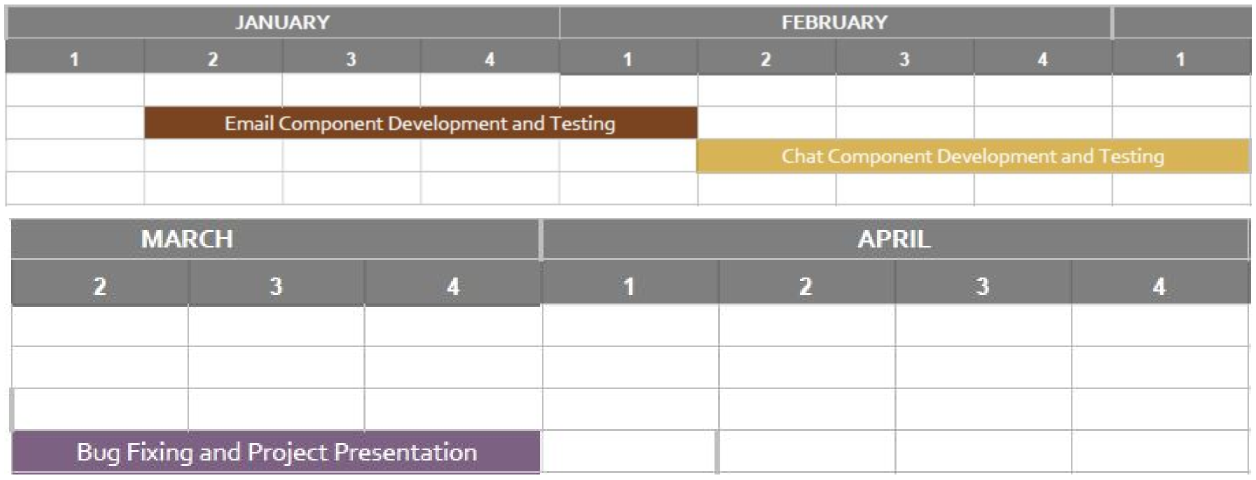
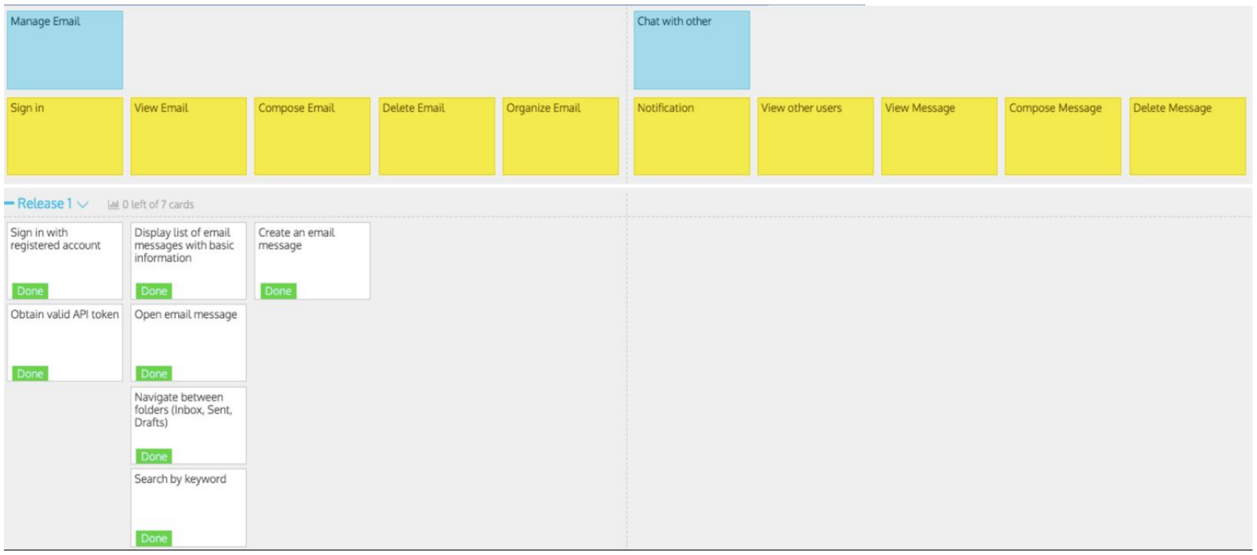


Figure 1. Project Timeline

3.3 User Story Map

User Story Map is used as a guideline for this project to keep track on features to work on in each releases. A user story map is used to see how the user would interact with the product. As such, Figure 2, 3, and 4 below shows the activities of the user, the steps to achieve those activities, and the features necessary for each step and these are separated into three releases or sprints for this project.



Manage Email					Chat with other				
Sign in	View Email	Compose Email	Delete Email	Organize Email	Notification	View other users	View Message	Compose Message	Delete Message
Release 1 0 left of 7 cards									
Sign in with registered account Done	Display list of email messages with basic information Done	Create an email message Done							
Obtain valid API token Done	Open email message Done								
	Navigate between folders (Inbox, Sent, Drafts) Done								
	Search by keyword Done								

Figure 2. User Story Map - Sprint 1

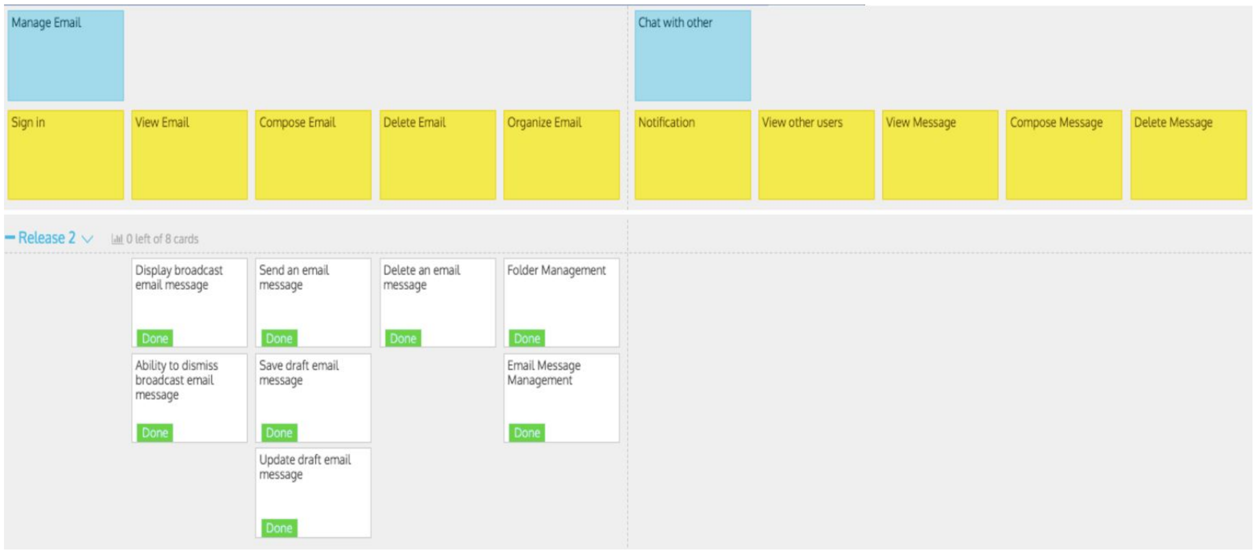


Figure 3. User Story Map - Sprint 2

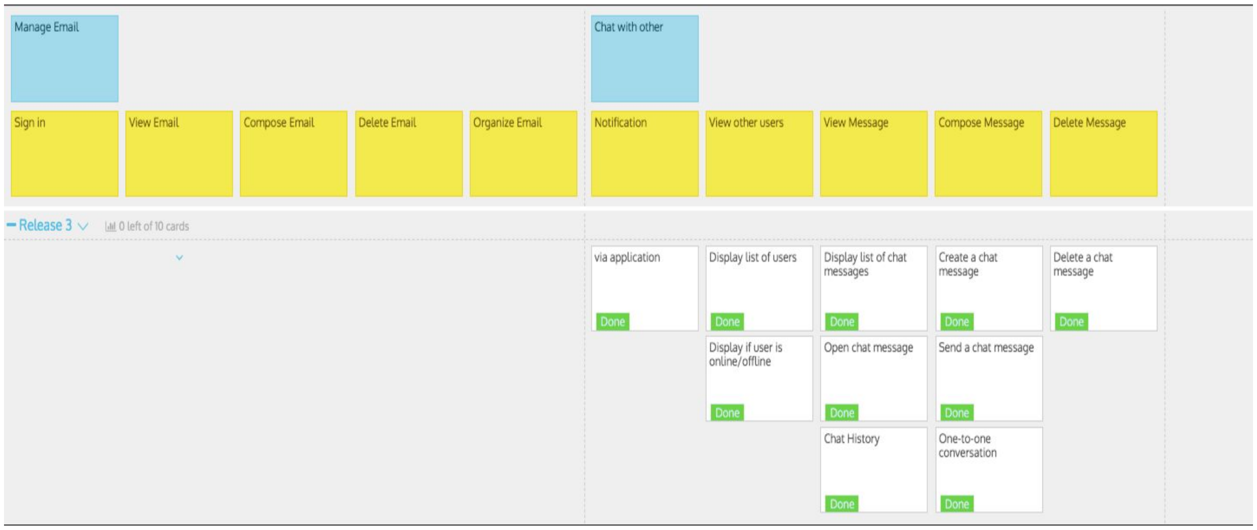


Figure 4. User Story Map - Sprint 3

4 System Design

4.1 Design Goals

The design goals for this project are the following:

- The system must be an iOS-based application built in Swift programming language.

- The implementation of the email system must follow how it is implemented using the API used in the Online Therapy Unit's email system.
- The implementation of the chat system must use a third-party API particularly Chatkit.
- The user interface of the application must be user-friendly in terms of ease of use and visually appealing.
- The system must be extensible.

4.2 Current Software Architecture

The system being used by the Online Therapy Unit is implemented in a web application. Their service platform is based on microservice architecture. The structure of the architecture is a collection of loosely coupled services that each provides an API. These services are the following as shown in Figure 5 below:

- Hippo - API Proxy
- Snitch - Activity Feeds
- Scribe - Client notes management
- Tango - Client/staff matchings
- Ledger - Program enrollments and per-enrollment client data store
- Postoffice - Internal messaging system
- Babylon - Program management
- Curator - Module/content management
- Ridler - Client surveys
- Trumpet - Site-wide system notifications aka broadcasts

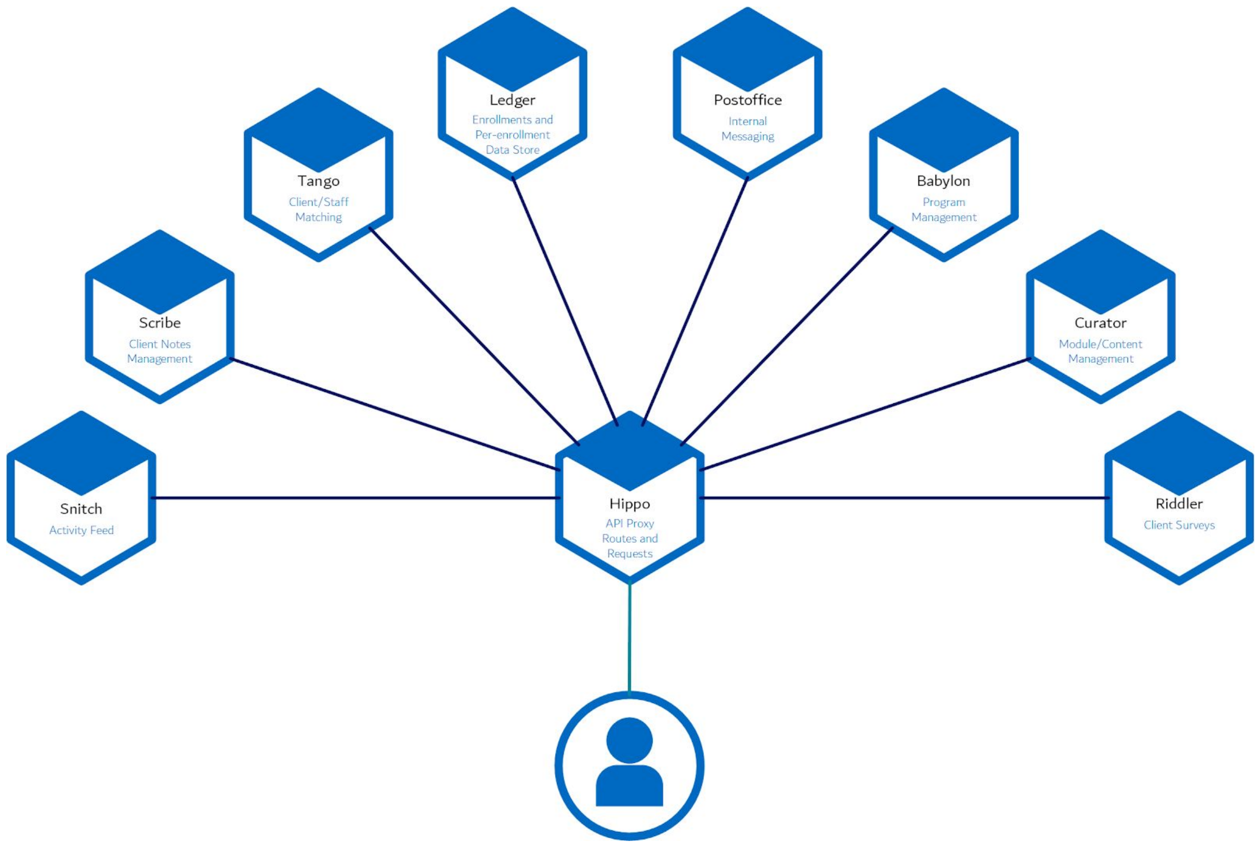


Figure 5. Online Therapy Unit Architecture

4.3 Proposed Software Architecture

4.3.1 Overview

This system is separated into three components: Application, Chat API, and the Online Therapy Unit API. The application is the user interface that will be containing the email and chat services in order to interact with the users. The Chat API is provided by a third-party application called Chatkit to provide us with the core features of the chat part of the application. The Online Therapy Unit API provides the email features of the email part of the application. Figure 6 below shows the high-level view of the architecture for this project.

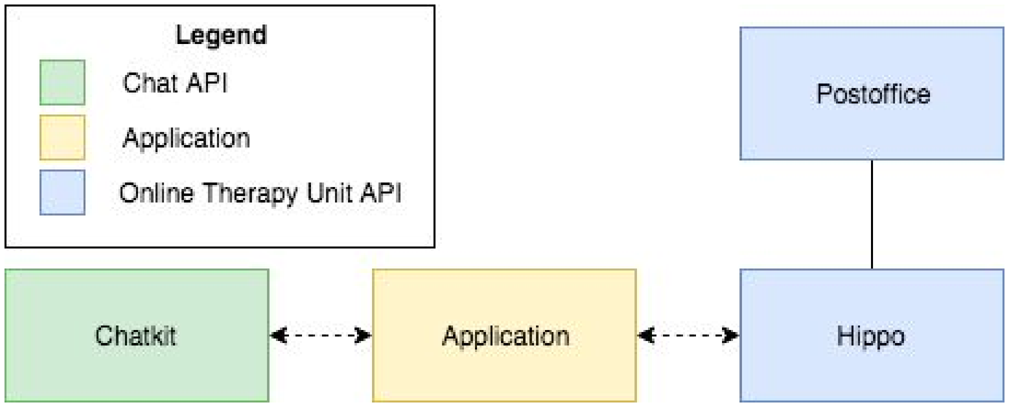


Figure 6. High-Level View Architecture

4.3.2 Subsystem Decomposition

4.3.2.1 Application

The application is built using the Swift language and using an IDE called Xcode. Furthermore, the application is going to be an iOS-based application. The main responsibility of this component is to provide the user with an interface in order to use the email and chat features of the application.

4.3.2.2 Online Therapy Unit API

The Online Therapy Unit API's main responsibility is to provide the application with the email features. In order to gain access and interact to Postoffice, an API for internal messaging and email system, the application needs to communicate with Hippo, a public-facing API Proxy/Gateway.

4.3.2.3 Chat API

The Chat API is provided by Chatkit, a third party extensible API to build in-app messaging. The main responsibility of this component is to provide the application with the chat features. These features include being able to create rooms to have one-on-one or group chats, create or send a chat message, push notifications, and typing indicator.

4.3.2.3.1 Advantages of using Chatkit

- Good Documentation
- The Sandbox Plan is free which allows up to 1000 users per month, 500,000 messages per month, and contains the core features necessary for a chat system.
- Can be built with multiple languages such as Swift, Java, Javascript, etc.
- No need to make any changes in the Online Therapy Unit API to make Chatkit work.

4.3.2.3.2 Disadvantages of using Chatkit

- Costly when upgrading plan
- End-to-end encryption is still in development

4.4 Access Control

There are going to be three people who are involved in the system: software developer, therapist, and the patient. The software developer will be managing the system, while the therapist and patient will only be able to access the application. The software developer can edit the Online Therapy Unit API in order to make necessary changes for the email system. Furthermore, the software developer can manage the rooms and users using Chatkit for the chat system. The software developer can also make changes in the user interface of the application. The therapist and patient can only interact and make use of the services in the application. Figure 7 below show the high-level view of the access control.

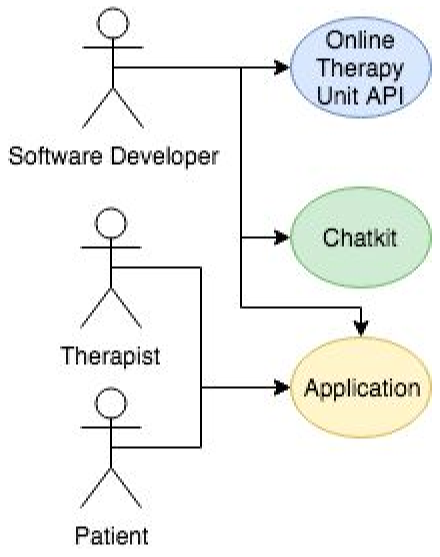


Figure 7. Access Control

5 Object Design

5.1 Object Design Trade-Offs

The trade-off decision that is made in the object design is the decision to build versus buy components for this system. We will be building the user interface of the system with the functionalities provided by existing APIs. Using existing APIs is done to save time.

5.2 Interface Documentation Guidelines

- Classes are named with nouns.
- Methods are named with verb phrases.

5.3 Design Pattern

5.3.1 Apple's MVC

The Apple's Model-View-Controller (MVC) design pattern assigns three roles to an object in an application particularly model, view, and controller. This pattern describes how the communication between the objects. The Model object encapsulates the data specific to an application and define the logic and computation that manipulate and process the data. The View object is an object that users can see. The Controller objects acts as an intermediary between one or more of an application's view objects and one or more of its model objects. Figure 1 shows the model view controller and how it communicates between components.

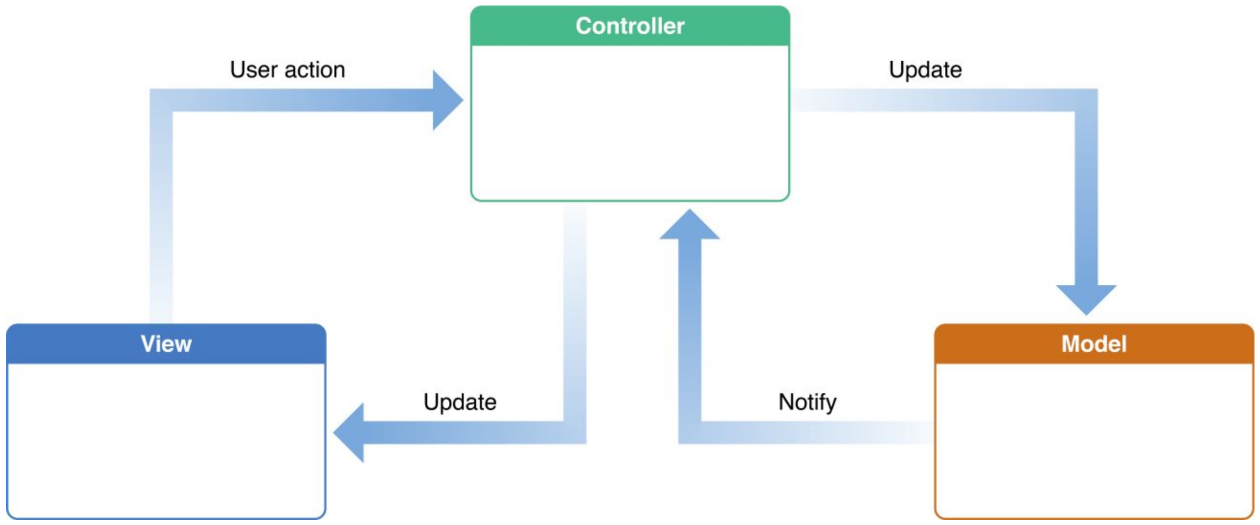


Figure 8. Model-View-Controller

5.4 Class Interfaces

Class represents an object or concept that contains variables, methods, and relationships with other classes. The classes are represented using MVC pattern as shown in Figure 9 and 10.

5.4.1 Email System

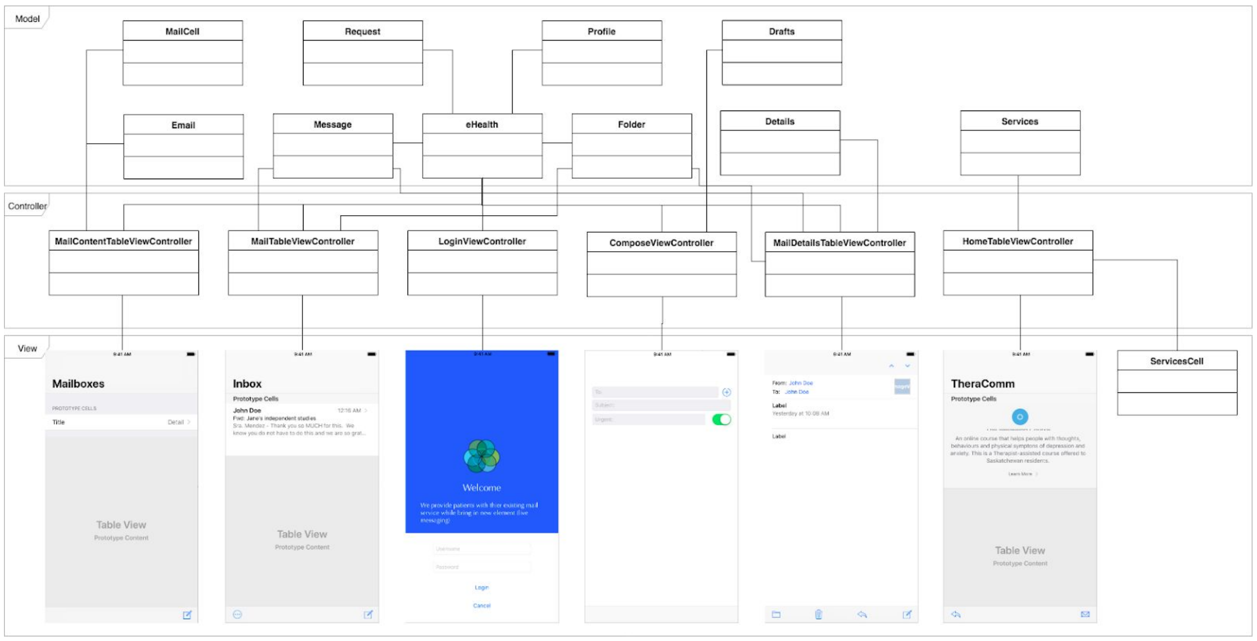


Figure 9. Class Diagram for the Email System

5.4.1.1 Model

5.4.1.1.1 Email

The email class is responsible for wrapping the HIPPO PROXY backend. The HIPPO Proxy backend uses JWT for authentication and contains several endpoints the public can use to access the internal post-office.

Functions:

Auth(User, Password)

- Parameters
    - User (String)
        - The username
    - Password (String)

        - The password

- Description
    - This function authenticates the user and stores the JWT token result in a class variable for future use. This function must be called and indicate success before using other class functions.
- Return Value
    - This function returns TRUE indicating success (BOOL)
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

MoveMessages(from_folder, to_folder, message_ids)

- Parameters
    - From_folder (String)
    - To_folder (String)
    - Message_ids ([String])
- Description
    - This function is responsible for moving messages to any user-defined folder.
- Return Value
    - This function returns TRUE indicating success (BOOL)
- Miscellaneous

    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

CreateFolder(folder_name, parent_folder_id)

- Parameters
    - Folder_name (String)
    - Parent_folder_id (Int?)
- Description
    - This function is responsible for creating a user defined folder
- Return Value
    - This function returns an optional Folder.SingleResult structure type. If the optional return value is nil that indicates a failure.
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

SaveDraft(recpt_ids, body, subject, reply_to_id, urgent)

- Parameters
    - Recpt_ids ([String])
    - Body (String)
    - Subject (String)
    - Reply_to_id (String)
    - Urgent (bool)

- Description
    - This function is responsible for saving a message to the draft folder for future use.
- Return Value
    - This function does not return any value to indicate success or failure
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

ComposeMessage(recpt_ids, body, subject, reply_to_id, urgent)

- Parameters
    - Recpt_ids ([String])
    - Body (String)
    - Reply_to_id (String)
    - Urgent (Bool)
- Description
    - This function is responsible for sending a message to an internal user
- Return Value
    - This function returns a Message.ComposeResult optional value to indicate success. If this function returns a nil optional value this indicates failure.
- Miscellaneous

    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

GetFolders()

- Parameters
    - This function takes no parameters
- Description
    - This function returns a list of folders for the currently authenticated user
- Return Value
    - An optional Folder.Result structure that indicates success and contains the folders.
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

GetMessages()

- Parameters
    - Folder_id (string)
- Description
    - This folder is responsible for retrieving all of the messages in a given folder
- Return Value

    - An optional Message.Result structure type that indicates success and contains the messages
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

GetMessage(folder_id, message_id)

- Parameters
    - Folder_id (string)
    - Message_id (string)
- Description
    - This function returns a specific message in full
- Return Value
    - This function returns an optional Message.SingleMessage.Result to indicate success
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

GetMatchings()

- Parameters
    - This function takes in no parameters
- Description

    - This function returns the matching profiles for the current authenticated user
- Return Value
    - This function returns an optional Profile.MatchUserResult to indicate success.
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

GetProfile()

- Parameters
    - This function takes in no parameters
- Description
    - This function is responsible for returning the current user’s profile
- Return Value
    - This value returns a Profile_Information structure always
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

GetSenderInformation(Message)

- Parameters
    - Message (Message.SingleMessage.Result)

- Description
    - This method parses the TO information of the message
- Return Value
    - This function returns an optional sender_information structure to indicate success
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

GetSenderInformation(messages, msg_id)

- Parameters
    - Messages (Message.Result)
    - Msg_id (String)
- Description
    - This method parses the FROM information of the message
- Return Value
    - This function returns an optional sender_information structure to indicate success
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

GetToInformation(Message)

- Parameters
    - Message (Message.SingleMessage.Result)
- Description
    - This method parses the TO information from the specific message
- Return Value
    - This method returns an array of sender_information structure always
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

GetToInformation(messages, msg_id)

- Parameters
    - messages (Message.Result)
    - Msg_id (String)
- Description
    - This method parses the TO information from the specific message
- Return Value
    - This method returns an array of sender_information structure always
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

DeleteMessage(folder_id, message_id)

- Parameters
    - Folder_id (string)
    - Message_id (string)
- Description
    - This method deletes a specific message in a specific folder
- Return Value
    - This method returns a BOOL (TRUE) which indicates success
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

DeleteFolder(folder_id)

- Parameters
    - Folder_id (string)
- Description
    - This method deletes a specific folder
- Return Value
    - This method returns a BOOL (TRUE) which indicates success
- Miscellaneous
    - This function is thread-safe and synchronous. The ASYNC I/O is awaited for by using a semaphore object.

5.4.1.1.2 Request

This class is responsible for encapsulating the URLRequest class for processing HTTP(/S) requests

Functions:

HTTPPostJSON(url, data, callback(string, string?))

- Parameters
    - Url (string)
    - Data (data)
    - Callback (function pointer to a type of function that takes two parameters of string and returns void)
        - String
        - String?
- Description
    - This function processes an HTTP Post request with JSON formatted data
- Return Value
    - None

HTTPsendrequest(request, callback(string, string?))

- Parameters
    - Request (URLRequest)
    - Callback

            - String
            - String?
- Description
    - This function performs the sending of the URLRequest
- Return Value
    - None

HTTPPOSTJSONAPI(url, token, data, callback(string, string?))

- Parameters
    - Url (string)
    - Token (string)
    - Data (data)
    - Callback
        - String
        - String?
- Description
    - This function is responsible for sending an authenticated JSON POST request
- Return Value
    - None

HTTPGETJSONAPI(url, token, data, callback(string, string?))

- Parameters
    - Url (string)
    - Token (string)
    - Data (data)
    - Callback
        - String
        - String?
- Description
    - This function is responsible for sending an authenticated JSON GET request
- Return Value
    - None

HTTPDELETEJSONAPI(url, token, data, callback(string, string?))

- Parameters
    - Url (string)
    - Token (string)
    - Data (data)
    - Callback
        - String
        - String?

- Description
    - This function is responsible for sending an authenticated JSON DELETE request
- Return Value
    - None

HTTPPUTJSONAPI(url, token, data, callback(string, string?))

- Parameters
    - Url (string)
    - Token (string)
    - Data (data)
    - Callback
        - String
        - String?
- Description
    - This function is responsible for sending an authenticated JSON PUT request
- Return Value
    - None

5.4.1.1.3 Message

This class is responsible for modelling an interface for the JSON request & responses for handling of the messages. The structures are directly mirrored with the requests and responses. This class uses the Decodable and Codeable interfaces.

5.4.1.1.4 Profile

This class is responsible for modelling an interface for the JSON request & responses for the handling of the user’s profile. The structures are directly mirrored with the requests and responses. This class uses the Decodable and Codeable interfaces.

5.4.1.1.5 Folder

This class is responsible for modelling an interface for the JSON request & responses for the handling of folders. The structures are directly mirrored with the requests and responses. This class uses the Decodable and Codeable interfaces.

5.4.1.1.6 Details

A UITableViewCell object that manages the details of each mail in the MailDetailsTableViewController. It contains field associated with the mail such as from: String, to: String, subject: String, date: String, body: String, index: Int, emails: [Email] (format: variable: Type).

5.4.1.1.7 Services

The Services class is a model structure managing the information displayed in the HomeTableViewController which is the homepage of the application. Its structure holds serviceImage, serviceTitle, serviceBody, serviceLink and serviceLinkImage.

5.4.1.2 View

5.4.1.2.1 MailCell

A UITableViewCell object such as MailCell is a specialized type of view that manages the content of a single table row. The TableViewCell is a the visual representation of a row in the table view. In the case is it's TableView belongs to the MailTableViewController. The MailCell is a UITableViewCell subclass with your custom behavior. Also a subclass of SwipeTableViewCell which is from a third party library (SwipeCellKit) which gives us the ability to perform various swipe actions.

5.4.1.2.2 ServiceCell

ServiceCell is a subclass of the UITableViewCell which is used to provide custom behavior for the HomeTableViewController. It takes in outlets for two UIImageViews and three UILabels. It is designed in the main storyboard and the ServiceCell class is used to populate the TableViewCell in the HomeTableViewController.

5.4.1.3 Controller

5.4.1.3.1 LoginViewController

LoginViewController is the controller associated with the login screen. LoginViewController is the initial view controller that control is being transferred to after the launch screen. The LoginView like any other login screen has a username and password text fields, a login button and a cancel button. The controller takes in the username and password from the user. When the login button is pressed, a level of validation takes place and if the username or password is wrong the user is presented an error message. If correct, a segue is performed to the home screen of the application.

5.4.1.3.2 MailTableViewController

MailTableViewController is the UITableViewController responsible for handling the folders. It populates its datasource using email.GetFolders() from the model which returns an array of the user's folders. It also displays the appropriate images to go along with the folder. The controller is embedded inside a navigation controller hence the back button to the home screen. The controller also has the ability to delete particular folder, multiple folders and creating folders. It also has the ability to compose a messages without having to go to deep into the application. Selection of a row of the tableview performs a segue to the MailContentTableViewController.

5.4.1.3.3 MailContentTableViewController

MailContentTableViewController is responsible for showing the list of the mails in the selected folder. The mails which are presented in the view are only a preview of the its actual content and hence does not contain the full body of the mail. It's cells are custom made and belong to the class MailCell. Embedded in the TableViewController is also a UISearchController which helps us in filtering the tableview datasource to match whatever is being searched. The search algorithm works for the from, subject and body fields of the MailCell. With a change in the searchBar the datasource of the tableview is modified and the tableview is reloaded showing the newly generated tableview showing the search results. Some of the features that this controller provides are as followed:

- Deleting a particular mail.
- Deleting multiple mails while in the editing mode of the tableviewcontroller.
- Showing only unread messages and has to ability to toggle its states between showing only unread and the original full list of mails.
- Composing of mail

Transition out of this controller is by clicking a particular mail. onSelection the controller performs a segue to the MailDetailsTableViewController where more details about the mail is showing including the whole mail body as opposed to the preview shown the MailContentTableViewController.

5.4.1.3.4 MailDetailsTableViewController

MailDetailsTableViewController handles presenting the details for a particular mail. The content of this controller will be explained in order of appearance on the screen. This controller contains:

- The back button to the previous controller because they are all embedded in the navigation controller.
- An up and down button which presents the next or previous mail based on the sequential order in which they appear in the MailContentTableViewController.
- Shows the full name of the sender and the receiver.
- UIImageView that generates letter initials as a placeholder for user images. In this case generating its image based on the full name of the sender. The code for this was gotten from and external source under the MIT license (DPImageView+LettersExtension)
- The subject of the mail
- The Date and Time that the mail was sent in a fully descriptive date/time format.
- The body of the mail. This is the full body of the mail as opposed to the preview displayed in the MailContentTableViewController.
- The ToolBar located and the bottom of the screen contains 4 buttons used of 4 distinct purposes:
    - Move: A bar button item used for moving particular messages from one folder to another. An action sheet AlertViewController is presented when the button is clicked and the user can move the mail to whatever folder of their choose. When the new folder is selected, it

is removed from the current folder and the next mail is presented sequentially.

- Delete: The trash button is used for deleting the particular mail and same as the move the next mail is presented sequentially after the mail has been deleted.
- Reply: The reply button is used to reply to the mail that is currently displayed.
- Compose: Used to compose a new mail.

5.4.1.3.5 ComposeViewController

This controller is responsible for sending internal emails using the Online Therapy API between users. The emails are sent using the user ID. The decision was made to make use of the user ID as opposed to the username or full name to eliminate the possibility of multiple users having the same name or username. The sending can either be the original sending of the mail or reply to a sent mail. The presentation of this controller is unconventional in its animation and transition, similar to the Mail and Apple application. This transition and presentation are handled with a third party library called SPStorkController. The ComposeViewController is accessible from various controllers which include the MailTableViewController, MailDetailsTableViewController.

5.4.1.3.6 HomeTableViewController

This is the home page for the application and has information and links to Services provided by the Online Therapy Unit.

5.4.2 Chat System

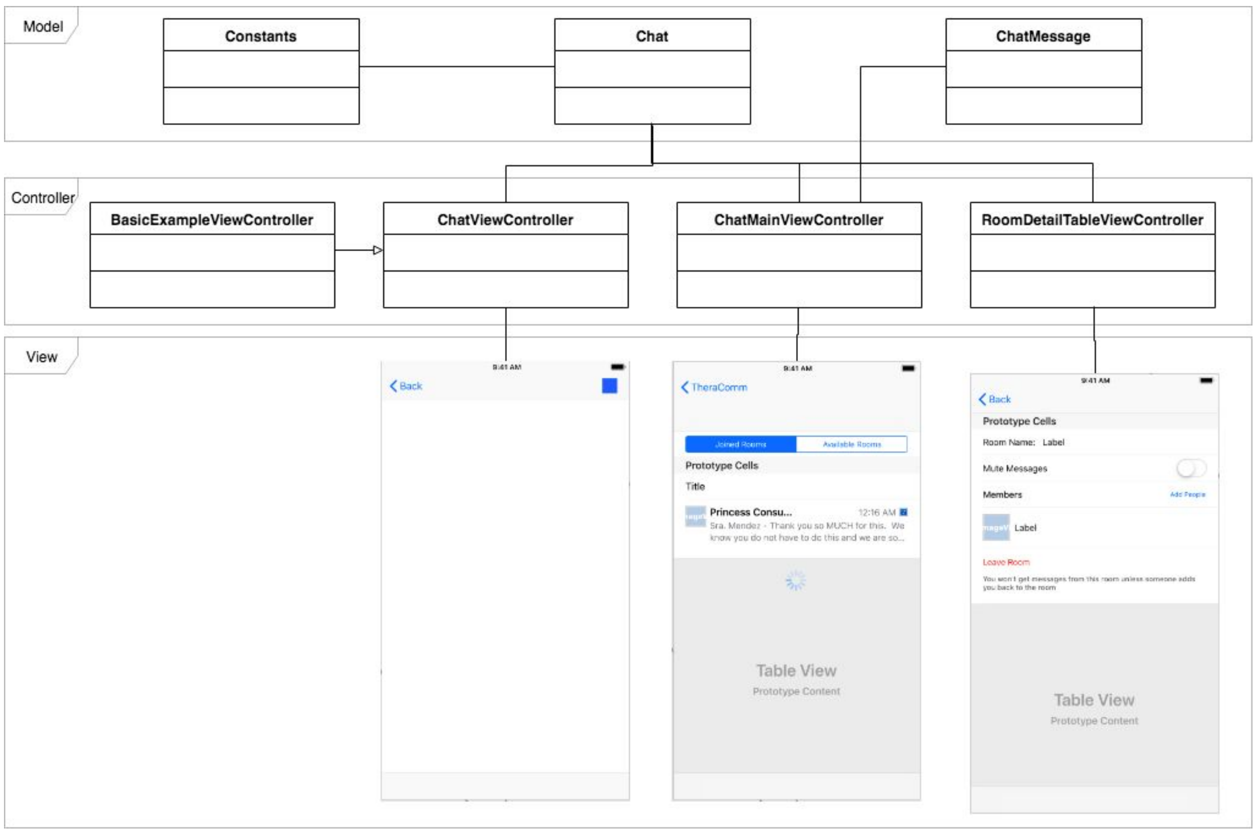


Figure 10. Class Diagram for the Chat System

5.4.2.1 Model

5.4.2.1.1 Constants

The constants class holds immutable global data such as the token provider, the instance locator, and the API path.

- Variables
  - instanceLocator
  - tokenProvider
  - idProvider

5.4.2.1.2 Chat

The chat class wraps the ChatKit library. It is responsible for setting up and using the ChatKit library functions. Authentication is required to use this class.

Functions:

GetUserId(username, password)

- Parameters
    - Username (String)
    - Password (String)
- Description
    - This function is responsible for retrieving the associated online therapy unit ID for the username
- Return Value
    - The return value is an optional string indicating success
- Miscellaneous
    - This function is thread safe and synchronous. ASYNC I/O operations are awaited through using a semaphore object.

Authenticate(username, password, delegate)

- Parameters
    - Username (String)
    - Password (String)

    - Delegate (PCChatManagerDelegate)
- Description
    - This function is responsible for authenticating the online therapy unit with the chat application
- Return Value
    - This function returns an optional PCCurrentUser to indicate success
- Miscellaneous
    - This function is thread safe and synchronous. ASYNC I/O operations are awaited through using a semaphore object.

LeaveRoom(room)

- Parameters
    - Room (PCRoom)
- Description
    - This function is responsible for leaving a PCRoom
- Return Value
    - A bool type indicating success (TRUE)
- Miscellaneous
    - This function is thread safe and synchronous. ASYNC I/O operations are awaited through using a semaphore object.

DeleteRoom(room)

- Parameters
    - Room (PCRoom)
- Description
    - This function is responsible for deleting a PCRoom
- Return Value
    - This function returns a bool indicating success (TRUE)
- Miscellaneous
    - This function is thread safe and synchronous. ASYNC I/O operations are awaited through using a semaphore object.

CreateRoom(name, isPrivate)

- Parameters
    - Name (String)
    - isPrivate (bool)
- Description
    - This function is responsible for creating a room
- Return Value
    - An optional PCRoom value to indicate success
- Miscellaneous

    - This function is thread safe and synchronous. ASYNC I/O operations are awaited through using a semaphore object.

GetJoinableRooms()

- Parameters
    - None
- Description
    - This function is responsible for retrieving joinable rooms for the current authenticated user
- Return Value
    - An array of PCRooms
- Miscellaneous
    - This function is thread safe and synchronous. ASYNC I/O operations are awaited through using a semaphore object.

SubscribeToRoom(room_id, delegate, message_limit)

- Parameters
    - Room_id (String)
    - Delegate (PCRoomDelegate)
    - Message_limit (int)
- Description

    - This function is responsible for subscribing to a specific room to receive real-time notifications through the delegate
- Return Value
    - A bool indicating success (TRUE)
- Miscellaneous
    - This function is thread safe and synchronous. ASYNC I/O operations are awaited through using a semaphore object.

SubscribeToRoom(room, delegate, message_limit)

- Parameters
    - Room (PCRoom)
    - Delegate (PCRoomDelegate)
    - Message_limit (int)
- Description
    - This function is responsible for subscribing to a specific room to receive real-time notifications through the delegate
- Return Value
    - A bool indicating success (TRUE)
- Miscellaneous
    - This function is thread safe and synchronous. ASYNC I/O operations are awaited through using a semaphore object.

FetchMessages(room, limit)

- Parameters
  - Room (PCRoom)
  - Limit (int)
- Description
  - This function is responsible for retrieving messages from a specific room
- Return Value
  - An array of PCMultiPartMessage
- Miscellaneous
  - This function is thread safe and synchronous. ASYNC I/O operations are awaited through using a semaphore object.

5.4.2.1.3 ChatMessage

The ChatMessage object manages the details of the data which are displayed on the ChatMainViewController. It is the data model representing each cell in the tableview. Its contents includes: name (type: String): name of the room, message (type: String): most recent message sent or received, date (type: String): the date in which the most recent message was sent or in the case of no messages in the room then it holds the name of the user that created the room and what they named the room, room (type: PCRoom): the current PCRoom being handled.

5.4.2.2 View

5.4.2.2.1 ChatKitMessagesListCell

This is a view for the UITableViewCell for the ChatMainViewController which contains a list of all the rooms. The cell contains the room avatar “messageAvatar”

(ImageView showing either the placeholder image for the room or the initials for the room), the name of the room “senderName”, the label showing the most recent messages in the room “resentMessage”, the date the most recent message was sent “messagesDate”. This class can be found in the ChatMainViewController, placed there for better understanding of what controller a cell is responsible for.

5.4.2.2.2 RoomNameTableViewCell

This is a custom UITableViewCell belonging to the RoomDetailTableViewController and holds the name of the room.

5.4.2.2.3 MuteMessagesTableViewCell

This is a custom UITableViewCell belonging to the RoomDetailTableViewController and holds a switch for unsubscribing from a room which mean the current user will no longer be receiving notifications for this room and therefore has the room muted. Notifications are only sent to users when they subscribe to a room.

5.4.2.2.4 MembersTableViewCell

This is a custom UITableViewCell belonging to the RoomDetailTableViewController and is responsible for adding users to a room. When the button is clicked, An AlertController pops up asking for the name of the room and a create or cancel button. By default all rooms are public. Modifications for future iterations will include the ability to specify if the room is private or public.

5.4.2.2.5 ListOfMembersTableViewCell

This is a custom UITableViewCell belonging to the RoomDetailTableViewController and is responsible showing the list of users subscribe to the room. The cell includes the user avatar and username of the user.

5.4.2.2.6 LeaveRoomTableViewCell

A UIButton which is used to leave a room which a user is already subscribed to. Room membership is persistent, so a user remains a member of the room until they explicitly leave (or are removed). The former is done with this button.

5.4.2.3 Controller

5.4.2.3.1 ChatMainViewController

The ChatMainViewController is the controller that manages the list of rooms and a preview of the which contains the name of the room, the most recent message in the room, the date the most recent message was sent and the placeholder ImageView showing the image of the room or the initials of the room. In this case the initials because the application does not suppose uploading of images for rooms or users. The controller contains three main outlets: segmentControl (type: UISegmentedControl!), tableView (type: UITableView!), spinner (type: UIActivityIndicatorView!). The segmentControl is used to indicate which datasource to be used for the TableView. There are two segments: Joined Rooms and Available Rooms. The datasource to be used is determined by the segment which is currently active. There are two static arrays used as the tableView datasource:

- ChatSubscribedRooms (type: [ChatMessage]): used only when the active segment is Joined Rooms
- ChatUnsubscribedRooms (type: [ChatMessage]): used only when the active segment is Available Rooms.

The Joined Rooms segment indicates that the current user is subscribed to the following list of rooms and populates the ChatSubscribedRooms with room details using the GetCurrentRooms() function from the Chat model. This functions is responsible for retrieving joined rooms for the current authenticated user.

The Available Rooms segment indicates that the current user is unsubscribed (not joined) to the following list of rooms and populates the ChatUnsubscribedRooms with the room details using the GetJoinableRooms() function which is responsible for retrieving joinable rooms for the current authenticated user.

Due to the timing delays produced from the fetching of information from the server, we added a spinner widget that start spinning when the fetching process is initiated and ends when the data is retrieved. This is to provide feedback to the user that information is being retrieved. The data fetching is done on a separate thread from the main thread so as to not distort and UI actions which all run on the main thread.

This controller is also responsible for adding rooms (by clicking the plus icon on the top right part of the screen), deleting rooms (using the normal swipe action for all tableviews).

5.4.2.3.2 BasicExampleViewController

The BasicExampleViewController is a subclass of ChatViewController which is responsible for managing the appearance of the chatroom. It inherits the MessagesDisplayDelegate and the messagesLayoutDelegate and make necessary changes to their delegate functions to fit our needs.

5.4.2.3.3 ChatViewController

The ChatViewController is the controller where all in the chat interactions are being handled and shown. It shows the chat interaction between the users in the room and the chat history as well as the dates in which messages were sent. Note: this controller does not show that dates in which all the messages were sent but shows them at a message interval. After every three messages the date/time of the three message is displayed just underneath the third message. The message bubbles are displayed with the help of a third party library called MessageKit. This controller is also capable of handling scroll view contents. This is used for shown passed messages. When the controller is loaded for the first time, only 20 messages are being displayed to the user. We limited this number to 20 to minimize the load time. When the user is at the top of the controller and refreshes the screen the controller fetches for the next 20 messages in the room.

Some important functions to be described in this controller include:

- func subscriptionChoice() - when the controller is loaded this function checks if the current user is joined to this room. If the user is not member of the room an alert pop up asking the user if they want to join the room and if “Yes” they are taken back to the ChatMainViewController where they can see the room they just joined among the list of joined rooms. And if “No” they are taken back to the ChatMainViewController but the room is not added to the list of joined rooms.

- func loadFirstMessages() - this function fetches for the first 20 messages of the room beginning with the most recent message. When it gets the first 20 messages it uses this to initialize the tableview datasource and the tableview is reloaded with its initial content. After reload the view scrolls down to the buttom.

- func loadMoreMessages() - this function calls the helper function “getMessages(count: 20)” to get the next 20 messages.

- func getMessages(count: Int, completion: ([MockMessage]) -> Void) - this function get the next 20 messages on refreshes using the oldestMessageIDReceived. This is used to represent, as the name suggests, the ID of the oldest message that a client had received up until now. When it is done fetching all 20 messages it returns them its completion handler.

- func insertMessage(_ message: MockMessage) - takes in a message and inserts it to the bottom of the collection view.
- func onMultipartMessage(_ message: PCMultipartMessage) - the PCRoomDelegate function that is called when a new message is sent or recieved. This function calls the insertMessage function so the message can be added to the bottom of the collection view.
- func onUserStartedTyping(inRoom: PCRoom, user: PCUser) - a user started typing in a room in which the current user is a member. The subtitle of the navigation bar title is changed to "username" is typing...
- func onUserStoppedTyping(inRoom: PCRoom, user: PCUser) - a user stopped typing in a room in which the current user is a member. The navigation bar goes back to its original style with not subtitles.
- func onUserJoinedRoom(_ room: PCRoom, user: PCUser) - a user joined a room in which the current user is a member. The subtitle of the navigation bar title is changed to "username" just joined the room
- func onUserLeftRoom(_ room: PCRoom, user: PCUser) - a user left a room in which the current user is a member. The subtitle of the navigation bar title is changed to "username" just left the room
- func onPresenceChanged(stateChange: PCPresenceStateChange, user: PCUser) - a user came online or went offline. The subtitle of the navigation bar title displays the current state of the user. If the user is online, displays "username" is online and if the user is offline displays "username" is offline

5.4.2.3.4 RoomDetailTableViewController

The RoomDetailTableViewController is responsible for presenting more details about a particular room. It shows the name of the room, switch to mute the room, a list of users in the room and button to leave the room.

5.5 Third-Party Libraries

5.5.1 Chatkit

Chatkit is a flexible third party API that integrates and implements a live-chat service which includes:

- Live Notification System
- Creation and Deletion of rooms
- Push Notifications
- Token Provider Authentication
- Typing Indicators
- Read Cursors
- Files
- Presence Updates
- Permission System

5.5.1.1 License

Sandbox Plan (FREE)

- 3,000 Uniquely identified users/month
- 90 Days message retention

- 5 MB maximum file size transfer
- 200 Media Stored
- 1,000,000 Messages / Month
- Unlimited peak concurrent connections
- All core features listed above included

5.5.1.2 Documentation and Usage

The documentation and how-to can be found at https://pusher.com/docs/chatkit.

5.5.2 SparrowKit

It is library for projects SPPermission & SPStorkController. Later being one of the libraries used in this project. Also library have many useful extensions and classes.

5.5.2.1 License

SparrowKit is licensed under the MIT License.

5.5.2.2 Documentation and usage

The documentation and how-to can be found at https://github.com/IvanVorobei/SparrowKit.

5.5.3 SPStorkController

Modal controller like in Mail or Apple Music application. Similar animation and transition. This is the third party library used in the transition, animation and appearance of the ComposeViewController.

5.5.3.1 License

SPStorkController is licensed under the MIT License.

5.5.3.2 Documentation and usage

The documentation and how-to can be found at https://github.com/IvanVorobei/SPStorkController.

5.5.4 MessageKit

MessageKit is third party library used to:

- Provide a :rotating_light:safe:rotating_light: environment for others to learn and grow through Open Source.
- Make adding Chat:speech_balloon: to a project easy.
- Enable beautiful and customizable Chat UI's.
- Provide an awesome Open Source project for the iOS open source community.
- Help others learn.

5.5.4.1 License

MessageKit is licensed under the MIT License.

5.5.4.2 Documentation and usage

The documentation and how-to can be found at https://messagekit.github.io/#requirements.

5.5.5 SwipeCellKit

Swipeable UITableViewCell based on the stock Mail.app, implemented in Swift. A swipeable UITableViewCell or UICollectionViewCell with support for:

- Left and right swipe actions
- Action buttons with: text only, text + image, image only
- Haptic Feedback

- Customizable transitions: Border, Drag, and Reveal
- Customizable action button behavior during swipe
- Animated expansion when dragging past threshold
- Customizable expansion animations
- Support for both UITableView and UICollectionView
- Accessibility

5.5.5.1 License

MessageKit is licensed under the MIT License.

5.5.5.2 Documentation and usage

The documentation and how-to can be found at https://github.com/SwipeCellKit/SwipeCellKit

6 Software Development Process

6.1 Agile Development

Agile development is an iterative approach to software development. It allows to adapt to changes in the requirements. It allows the involvement of clients throughout the process. It also helps in delivering minimum viable products in each releases within a specified amount of time. The following sections will show the focus for each sprint throughout the development of the project.

6.1.1 Sprint 1

The main focus of sprint 1 is to making the Online Therapy Unit's Email API work with the application and building the interface for the email system. The span for this sprint is 2 weeks from the 2nd week of January to the 3rd week of January.

6.1.2 Sprint 2

The main focus of sprint 2 is to add the functionalities and build the remaining interface of the email system. The span for this sprint is 2 weeks from the 4th week of January to the 1st week of February.

6.1.3 Sprint 3

The main focus of sprint 3 is to add the functionalities and build the interface of the chat system. The span for this sprint is 4 weeks from the 3rd week of February to the 1st week of March.

7 Test Plan and Execution

7.1 Features to be tested

The following features are based on the specifications and are necessary to fulfill the purpose of TheraComm; thus, these features are required to be tested.

Email Features:

- Logging in the application
- See the list of messages from corresponding folders (Inbox, Sent, Drafts)
- Navigation between folders
- Ability to view, send, and delete messages
- Ability to save and update draft messages
- The allowability of dismissing broadcast messages

- Ability to add and delete folders

Chat Features:

- Ability to create, send, and delete messages
- See the chat list and history
- Indication of typing activity
- Message Notifications
- Online/Offline user indicator

7.2 Features Not to be tested

The feature that is not going to be tested is the Swift features that is used to build the application since it is under the assumption that these features are already built-in the programming language.

7.3 Testing Approach

7.3.1 Functional Testing

7.3.1.1 Purpose

Functional testing is going to be used in order to verify the functionalities of the application. This approach is also done for quality assurance and to meet with the specifications.

7.3.1.2 Method

The test will be performed using the features provided by the Xcode application. This test is done by simulating the application and then performing the functionalities of the application within the simulated environment. Furthermore, this test can also be done in an Iphone device to see how the application perform in an actual device.

7.3.1.3 Pass/Fail Criteria

The pass/fail criteria of each of these functionalities is outlined in a pdf file that is provided with this test document called “TestCases.pdf ”. This file contains multiple test cases with pre-conditions, steps to take, post-conditions, and expected results in order to be used as a guide when testing each feature.

7.3.2 User Acceptance Testing

7.3.1.1 Purpose

User Acceptance Testing is necessary in order to know if the software product meets the expectation of the users not only in functionality, but also in usability particularly in terms of ease of use or ease to operate the product.

7.3.1.2 Method

The developers can be testers in this testing but the main testers of this approach is the clients. The clients will arrange a number of people in their staff to test the application. The testers will be provided with questionnaires to answer as outlined in a pdf file that is provided with this test document called “UsabilityQuestionnaire.pdf”.

7.3.1.3 Pass/Fail Criteria

The pass/fail criteria will depend on the feedbacks of the testers.

7.3.3 Unit Testing

7.3.3.1 Purpose

Unit testing is for testing each component of the application.

7.3.3.2 Method

The developers will test manually.

7.3.4 Integration Testing

7.3.4.1 Purpose

Integration testing is for testing when the email system of the application is integrated with the chat system of the application.

7.3.4.2 Method

The developers will test manually.

7.3.4 System Testing

7.3.4.1 Purpose

System testing is for testing the application as a whole.

7.3.4.2 Method

The developers will test manually.

7.4 Testing Materials

7.4.1 Hardware

- Laptop capable of running Xcode
- Iphone Device

7.4.2 Software

- Xcode version 10.1

7.5 Testing Schedule

The functionalities of the application will be tested during development to make sure everything is working until the completion of the application. There will be a final functionality testing that will take place in the second week of March. This will also be the time where the user acceptance testing will take place and estimated to be finished within a few days. Thus, this will allow the developers to fix any bugs that may arise before the Project Day.

8 Used Tools / Technologies

8.1 Swift

Swift is the programming language for developing iOS softwares.

8.2 Xcode

Xcode is the integrated development environment (IDE) for developing OS X and iOS software.

8.3 CocoaPods

CocoaPods is a dependency manager for Swift and Objective-C projects to help in scaling projects.

8.4 Online Therapy Unit API

See Section 4.3.2.2.

8.5 Chatkit

See Section 4.3.2.3.

8.6 Laravel

Laravel is a free, open-source PHP web framework to help in handling tokens and routing.

8.7 Github

Github is a web-based hosting service for version control using Git for this project.

8.8 GitKraken

GitKraken is Git graphical user interace (GUI) client to help in project planning. It contain kanban features and feature scheduling.

8.9 Toggl

Toggl is a time tracking application that helps to keep track the amount of time is done throughout the project.

8.10 StoriesOnBoard

StoriesOnBoard is a user story mapping tool for agile development. This tool is used to generate

9 Engineering Log Book

9.1 Meeting with Groupmates

Meeting	Date	Time	Summary of the meeting
1	04/10/2018	5:00 pm - 6:00 pm	• Work on Presentation
2	18/10/2018	5:00 pm - 6:00 pm	• Work on Presentation
3	01/11/2018	5:00 pm - 6:00 pm	• Work on Presentation
4	24/01/2019	5:00 pm - 6:00 pm	• Work on Presentation
5	07/02/2019	5:00 pm - 7:00 pm	• Work on Presentation, some features, and bug fixing
6	28/02/2019	5:00 pm - 7:00 pm	• Work on Presentation, some features, and bug fixing
7	14/03/2019	5:00 pm - 7:00 pm	• Work on Presentation, some features, and bug

			fixing
8	28/03/2019	5:00 pm - 9:00 pm	• Work on some features and bug fixing
9	29/03/2019	5:00 pm - 9:00 pm	• Work on some features and bug fixing
10	04/04/2019	12:00 pm - 11:00 pm	• Work on Final Presentation and practice
11	05/04/2019	4:00 pm - 5:00 pm 9:30 pm - 11:00 pm	• Work on Final Presentation and practice

9.2 Meeting with Client and Supervisor

Meeting	Location	Date	Time	Attendees
1	ED 410	03/10/2018	1:15 pm - 1:30 pm	• Tim Maciag • Christian John • Joe Samano
2	CW 126	10/10/2018	3:00 pm - 4:00 pm	• Heather Hadjistavropoulos • Marcie Nugent • Christian John • Iden Ellia • Joe Samano
3	ED 547.13	15/10/2018	12:30 pm - 2:30 pm	• Max Ivanov • Christian John • Iden Ellia • Joe Samano
4	ED 410	22/10/2018	11:45 pm - 12:00 pm	• Tim Maciag • Christian John • Joe Samano
5	CW 126	24/10/2018	2:30 pm - 2:45 pm	• Heather Hadjistavropoulos • Marcie Nugent • Christian John • Iden Ellia • Joe Samano
6	ED 410	04/02/2019	1:30 pm - 2:10 pm	• Tim Maciag • Christian John • Joe Samano
7	CW 126	11/02/2019	1:30 pm - 2:00 pm	• Marcie Nugent • Christian John • Iden Ellia • Joe Samano

8	ED 410	25/02/2019	1:30 pm - 2:00 pm	• Tim Maciag • Christian John • Joe Samano
9	ED 410	04/03/2019	1:30 pm - 2:00 pm	• Tim Maciag • Christian John • Joe Samano
10	ED 410	11/03/2019	1:30 pm - 2:00 pm	• Tim Maciag • Christian John • Joe Samano
11	ED 410	18/03/2019	1:30 pm - 1:35 pm	• Tim Maciag • Christian John
12	ED 410	18/03/2019	2:00 pm - 2:20 pm	• Tim Maciag • Joe Samano
13	ED 410	25/03/2019	1:00 pm - 2:00 pm	• Tim Maciag • Joe Samano
14	ED 441	02/04/2019	1:30 pm - 1:45 pm	• Tim Maciag • Christian John • Joe Samano
15	ED 410	03/04/2019	3:00 pm - 4:00 pm	• Tim Maciag • Joe Samano

Meeting	Summary of the Meeting
1	• We asked Tim if he could be our supervisor for our Capstone Project and Tim is okay with it. • We briefly discussed about our Capstone Project being a general real-time messaging application. • Tim asked us how is it different from other messaging applications like WhatsApp and what is the problem we are solving to which we don’t have. • Tim suggested to talk with Heather and discuss what we could work on that will work for us. • Tim will help us in setting a meeting with Heather.
2	• Heather and Marcie briefly discussed on what they currently have on their website and the some of the problems they are facing particularly how younger users are not completing their course. • We briefly discussed that we are working on a general real-time messaging application. • A messaging chat feature is one of the things that they do not have and we agreed

	that having a realtime messaging application is beneficial for them. • Brief list of possible features for the real-time messaging application: • One-to-One Chat • Encrypt messages • Save messages for therapists and users • Allow users to be able to delete messages • Track and record for the time it takes to reply • Marcie will inform the developer about this meeting so that we could discuss on how the backend works for their website and get the things we need for our project. • Heather will provide us with problem statements for our project.
3	• Max explained how the backend of the website works. • Max also explained that our project idea cannot be integrated with their system because what we wanted needed changes on their side as well. • Max proposed that instead of creating a real-time messaging application, we could build an application that contains the current features of their website. • Max explained that the project requirements that Heather gave us is too huge for us to handle, so Max will provide us with a new project requirements. • After the meeting with Max, we decided to do the idea that Max proposed to us.
4	• We asked Tim on what we could do in our situation wherein we are waiting for the project requirements and access to APIs from Max. • Tim advised us to focus on the project requirements we got from Heather and try to document and plan ahead.
5	• We discussed the changes of the project after talking with Max which includes how the real-time messaging application will not be integrated with their system, how Heather’s requirement were too broad and that we cannot handle, and that we are going to create an application with functionalities that their website can currently do particularly the email part. • We also showed them the new project requirements that we got from Max. • Heather and Marcie were fine with the changes and we will discuss in later meeting how we will proceed after doing the email part of the application.
6	• We talked about where we at in our project. • We discussed about the issue of having trouble contacting Max and Tim advised us to contact Heather. • We talked about what we are planning for the chat part of the application of using a third application and Tim suggested that we should talk with Heather about this plan as well. • We had Tim checked our Requirements and Specification Document and Tim wanted an abstract and user story map. Tim also gave us an access to a tool called StoriesOnBoard.
7	• We discussed our current progress in the project and provided a demo. • We discussed that we will be using a third-party application for the chat part of the application particularly Chatkit and explained the benefits to it. • We talked about testing our application. Marcie told us that they will prepare a number of people to test our application. We will get back to them on a later date. • We brought up the issue where Max is unresponsive to us to our emails. Marcie explained that Max is busy with some deadlines and she said that will email Max to fix

	our issues.
8	• We discussed where we at in our project. We showed our plan using a lo-fi prototype to show the design of the screens that we are working towards. • We discussed about the test plan and execution document and how we will be doing the testing. • We talked about how we are still having an issue with contacting Max and Tim suggested to try to email again and cc him in the email to keep track on the efforts we are making. Tim said that give Max a few more days and if Max still does not reply back then, Tim will email Max as well.
9	• We discussed our current progress of our project and what we are planning next. Here we showed Tim, the working interface including the home interface, email interface, and the chat interface. We also showed a separate project that implements sending message using ChatKit. • Tim discussed with us that he will set up a meeting with the people at eHealth so we could present our project and we agreed to this.
10	• We discussed our current progress of our project and what we are planning next. We did not show him anything, but described to him what we are currently working. • We also discussed regarding the eHealth presentation and that it is all set up. • We also discussed regarding documentation. Tim gave us a sample functional document for reference. Tim also asked us to start on the documentation even if it not complete by Friday.
11	• We had a short discussion of what we are currently doing regarding the project with Tim.
12	• We discussed about the documentation for the project with Tim. Tim gave advice on what to put in each of the documentation.
13	• We discussed our current progress of our project and what each member is currently doing. • We went through each documentation, what to fix, edit, or add. • We also had brief discussion regarding the presentation.
14	• We had a discussion regarding on what to add for the presentation.
15	• We had a final review regarding the presentation.

10 How-to-use / User Manual

10.1 General Information

10.1.1 Organization of the Manual

This document is divided into 4 sections:

- General Information: system in general terms and its purpose.
- System Summary: general overview of the system which includes the system configuration and user access levels.
- Getting Started: installation process and brief overview of the main screens.
- Using the System: detailed description of system functions.

10.2 System Summary

10.2.1 System Configuration

TheraComm operates on mobile devices with iOS operating system. The application requires an Internet connection in order to interact with the application. After installation, TheraComm can be used immediately without any further configuration.

10.2.2 User Access Levels

Only registered users are able to make use of the application particularly the patients and therapists.

10.3 Getting Started

10.3.1 Installation

The application can be downloaded from the iOS app store.

10.3.2 Logging In

A registered username and password is required to log in the interface.

10.4 Using the System

10.4.1 Email System

10.4.1.1 Sending Email

Sending an email message can be done by creating an email message and then sending the message. Click on the icon on the bottom right of the Email Interface to create an email message. An interface will show up as shown in Figure 11 where the user will be entering recipient, subject, and the body of the message. Click on the Send button and the message will be sent.

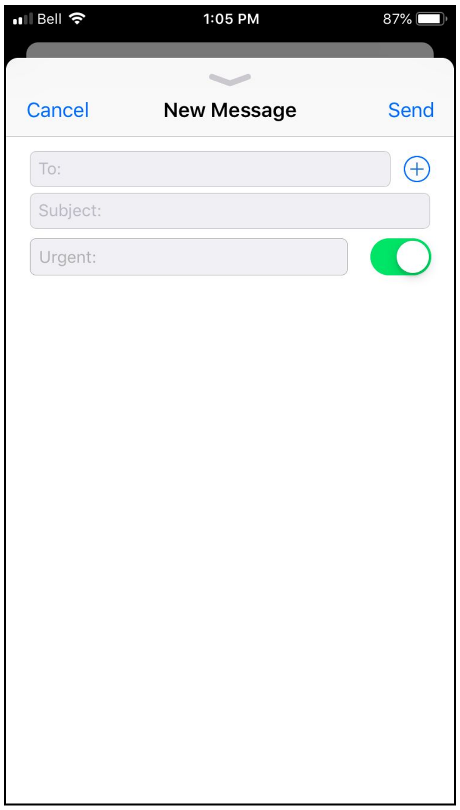


Figure 11. Compose Message Interface

10.4.1.2 Creating Folders

Creating a folder can be done by clicking + icon on the upper-left of the Email Interface. An interface will show up as shown in Figure 12 prompting the user to give a folder name. After entering the folder name, click on Create Button and the folder will be created.

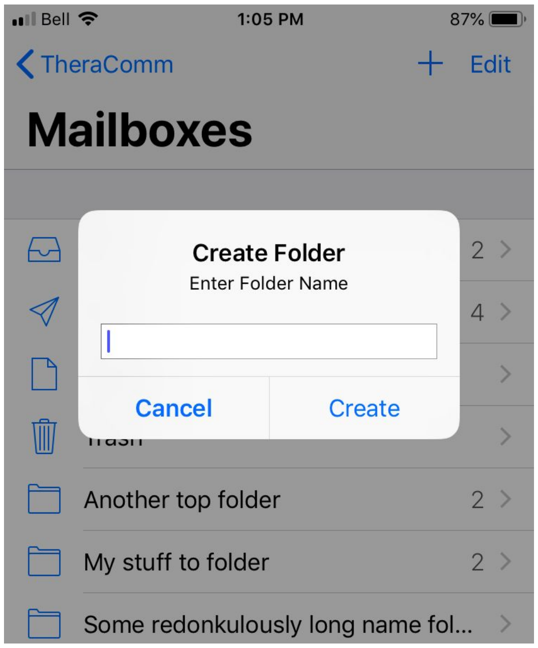
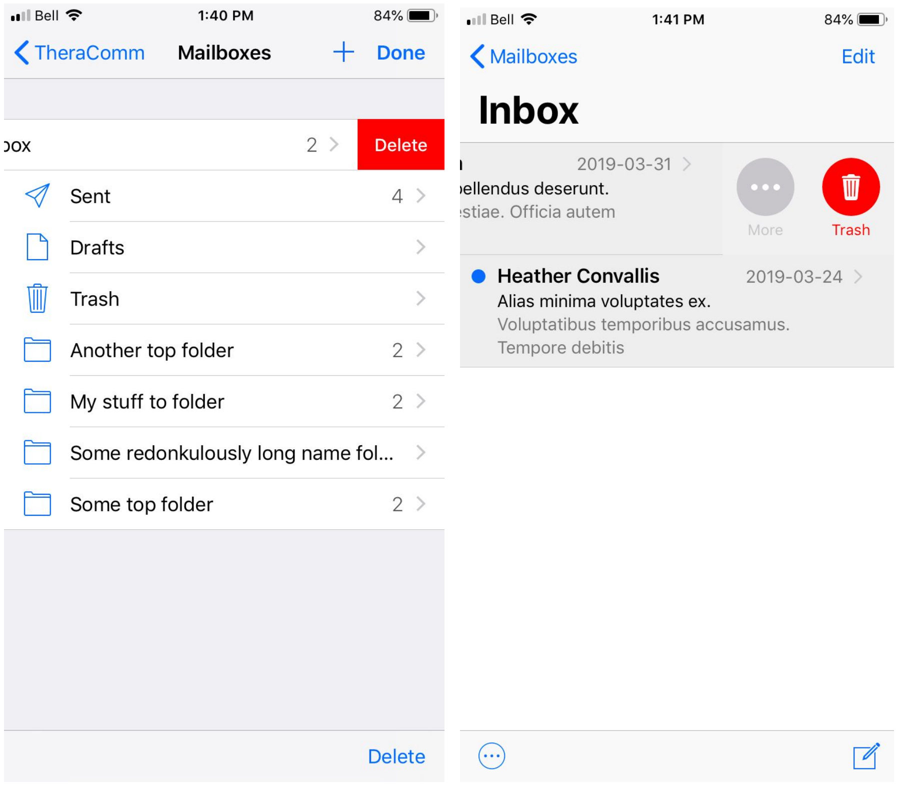


Figure 12. Create Folder Interface

10.4.1.3 Delete Email / Folders

Deleting can be done by swiping completely left on the folder or email the user wants to delete. The user can also partially swipe left and click on the red icon to delete the folder or email as shown in Figure 13.



ı 2019-03-31
ɔellendus deserunt.
ɛstiae. Officia autem

Heather Convallis 2019-03-24
Alias minima voluptates ex.
Voluptatibus temporibus accusamus.
Tempore debitis

Figure 13. Deleting Interface

10.4.1.4 Saving / Deleting Draft

Saving or deleting a draft message can be done by clicking on the Cancel button in the upper-left corner of the interface as shown in Figure 11 when an email message is being created. When clicking on the Cancel button, the user will be prompted to either save or delete the message as shown in Figure 14.

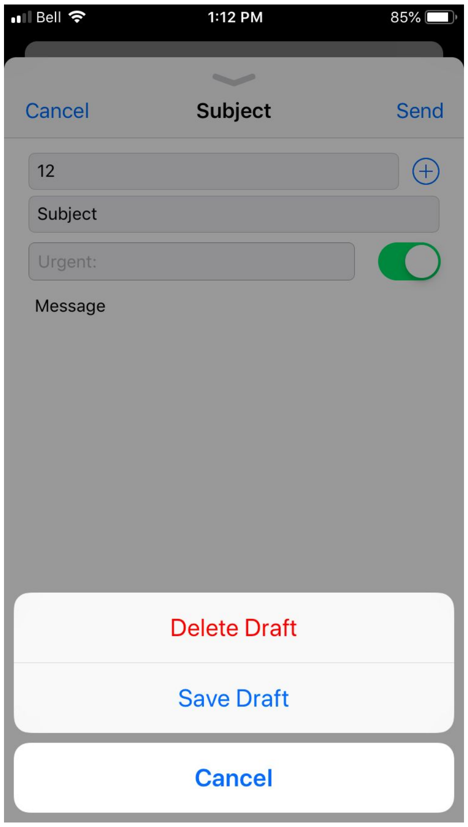
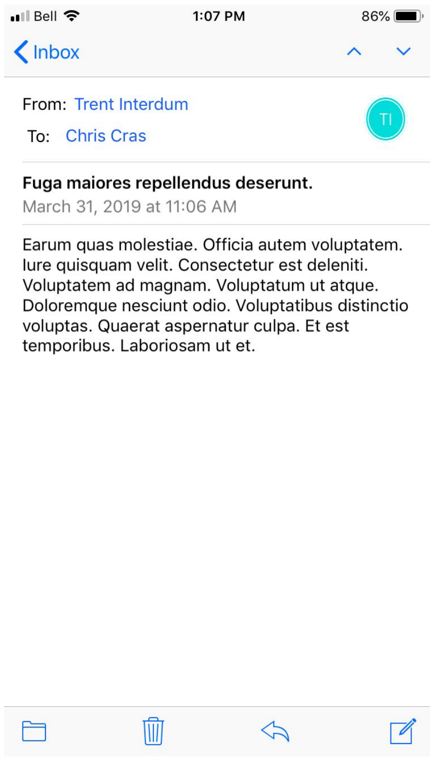


Figure 13. Draft Interface

10.4.1.5 Moving Email between Folders

Moving email messages can be done by going to email message the user wants to move and click on the icon in the lower-left corner of the Message Detail Interface as shown in Figure 14. After clicking the icon, the user will be shown with a list of folders to choose from to move as shown in Figure 15.


Bell 1:07 PM 86%

Inbox

From: Trent Interdum

To: Chris Cras

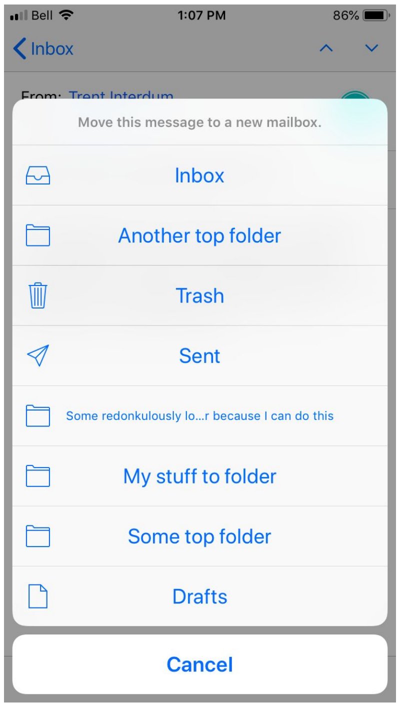
TI

Fuga maiores repellendus deserunt.

March 31, 2019 at 11:06 AM

Earum quas molestiae. Officia autem voluptatem. Iure quisquam velit. Consectetur est deleniti. Voluptatem ad magnam. Voluptatum ut atque. Doloremque nesciunt odio. Voluptatibus distinctio voluptas. Quaerat aspernatur culpa. Et est temporibus. Laboriosam ut et.

Figure 14. Message Detail Interface


Bell 1:07 PM 86%

Inbox

From: Trent Interdum

Move this message to a new mailbox.

Inbox

Another top folder

Trash

Sent

Some redonkulously lo...r because I can do this

My stuff to folder

Some top folder

Drafts

Cancel

Figure 15. List of Folders Interface

10.4.2 Chat System

10.4.2.1 Sending a Message

Sending a message can be done by going into a room and then going to the bottom of the Chat Interface as shown in Figure 16 and type a message. After typing a message, click on the Send Button to send the message.

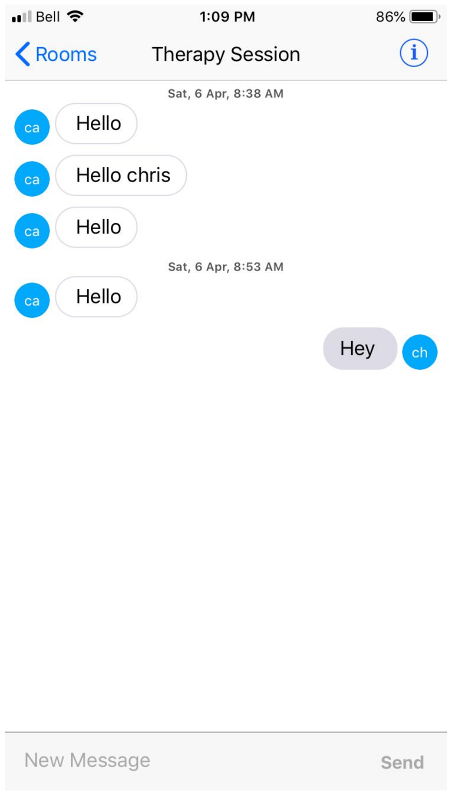
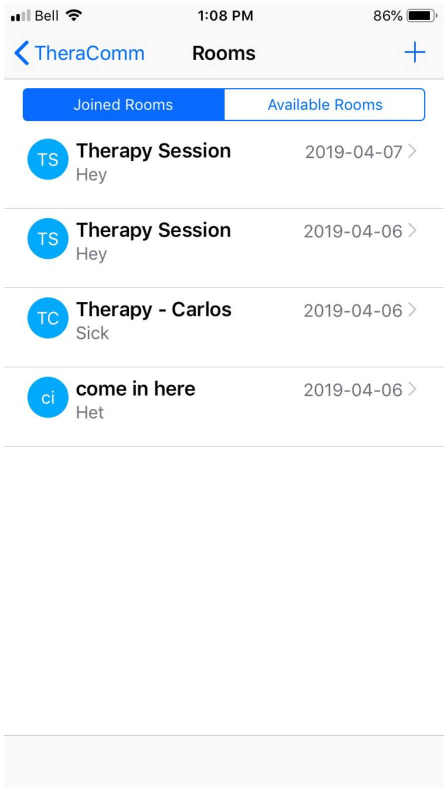


Figure 16. Chat Interface

10.4.2.2 Creating a Room

Creating a room can be done by clicking on the + icon in the upper-right corner of the Room Interface as shown in Figure 17. After clicking on the + icon, an interface will show up and prompt the user to enter the name of the room as shown in Figure 18. After entering the name, click on the Create Button to create the room.



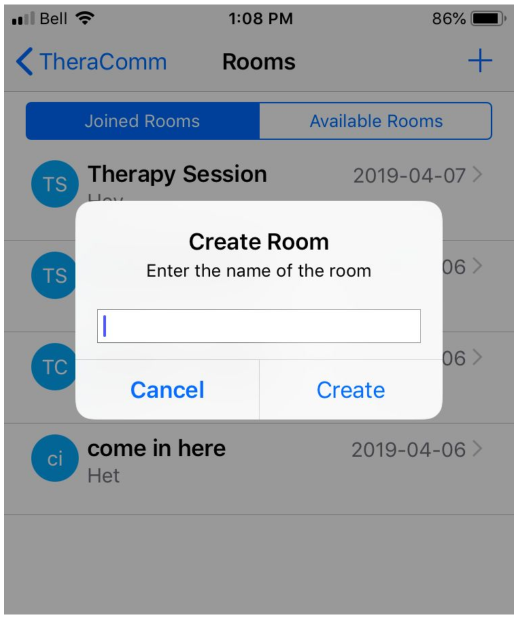
TS Therapy Session 2019-04-07
Hey

TS Therapy Session 2019-04-06
Hey

TC Therapy - Carlos 2019-04-06
Sick

ci come in here 2019-04-06
Het

Figure 17. Room Interface



Create Room
Enter the name of the room

Cancel Create

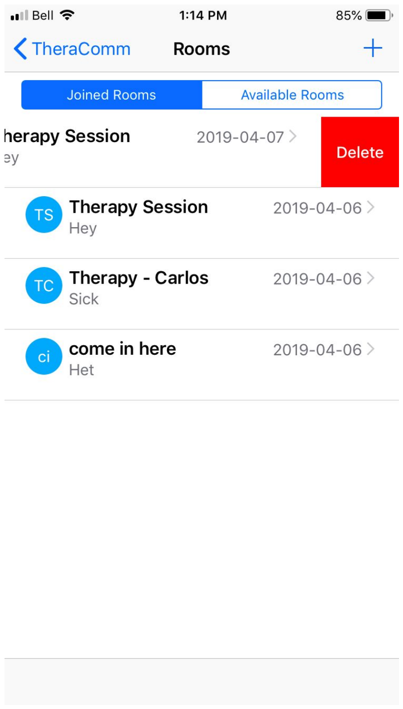
TS Therapy Session 2019-04-07

ci come in here 2019-04-06
Het

Figure 18. Create Room Interface

10.4.2.3 Deleting a Room

Deleting a room can be done similarly to section 10.4.1.3 by swiping completely left on the room or partially left and then click on the red icon as shown in Figure 20.

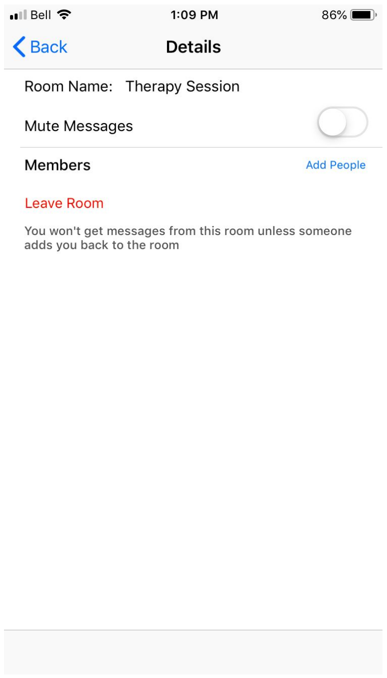


herapy Session ey	2019-04-07	
TS	Therapy Session Hey	2019-04-06
TC	Therapy - Carlos Sick	2019-04-06
ci	come in here Het	2019-04-06

Figure 20. Delete Room Interface

10.4.2.4 Adding User in a Room

Adding a user in a room can be done by going to room and then click on the ⓘ icon as shown in Figure 16. After clicking on the ⓘ icon, it will bring the user in the Chat Detail Interface as shown in Figure 21 and the user can click on the Add People button on the upper-right of the interface and enter the name of the other user. Then, the other user will be added to the room.



Room Name: Therapy Session

Mute Messages

Members

Leave Room

You won't get messages from this room unless someone adds you back to the room

Figure 21. Chat Detail Interface

11 Business Plan

11.2 Vision

Our vision for this project is to provide a more options for patients to communicate with their therapist. This project also hopes to be a good starting point to expand the current system of the Online Therapy Unit to mobile devices.

11.3 Stakeholders

This parties that are involved in this project are the following:

- Online Therapy Team
    - Dr. Heather Hadjistavropoulos - The executive director of the team
    - Marcie Nugent MSW - The coordinator of the team
    - Max Ivanov - The web developer
- Capstone Group
    - Christian John

  - Iden Ellia
  - Joe Samano

11. 4 Target Customers

- Students/Patients
- Therapist from the Online Therapy Unit Team

11.5 SWOT Analysis

11.5.1 Strengths

The strength of this project is that it improves the responsiveness of therapists to patients. It also provides patients with an option to set up a chat session with their therapist.

11.5.2 Weaknesses

The weakness of this project is that any changes that happens on the Online Therapy Unit's side may affect this project and it needs to adapt to those changes.

11.5.3 Opportunities

There a lot of opportunities for this application which include improving the current application like adding video chats, sending images, and sub-folders. The application can also be improved by adding other existing system or features from the Online Therapy Unit API.

11.5.4 Threats

The possible threat that needs to be considered is when there is going to be an iOS update that can affect the application. The application needs to be able to adapt to those changes to able to be functional.

11.6 Existing Similar Software Application

Talkspace is an existing application that is similar to this project. This application aims to provide more people with access to therapists. The main difference between TheraComm and Talkspace is that in terms of chat system, Talkspace allows audio messages, video messages, and send pictures, which TheraComm does not have but can be added in future releases. In terms of cost, Talkspace have different plans available which costs $49/week, $59/week, or $79/week with the main difference between the plans is the amount of live sessions per month. For TheraComm, depending on how the Online Therapy Unit team use the application, they may offer the services provided by TheraComm for free since the services they provided in their web application are free.

11.7 Cost

In term of costs for this project, Table below shows the total cost of the project. It is assumed that the cost for labour is $ 30 per hour. Chatkit's Sandbox Plan is free and it includes the core features for the chat system for this project. The Apple Developer License is necessary for deploying the application and to do push notifications.

Labour Cost	183 hr x $ 30/hr	$ 5490.00
Chatkit's Sandbox Plan		$ 0.00
Apple Developer License		$ 120.00
Total		$ 5610.00

Table 1. Cost of the Project

12 References

CocoaPods.(n.d.).Retrieved from https://cocoapods.org/

Github.(n.d.).Retrieved from https://en.wikipedia.org/wiki/GitHub

GitKraken.(n.d.).Retrieved from https://www.producthunt.com/posts/gitkraken-4

Laravel.(n.d.).Retrieved from https://en.wikipedia.org/wiki/Laravel

Model-View-Controller.(2018, April 6).Retrieved from https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

StoriesOnBoard.(n.d.).Retrieved from https://storiesonboard.com/

Swift.(n.d.).Retrieved from https://developer.apple.com/swift/

Talkspace.(n.d.).Retrieved from https://www.talkspace.com/

Toggl.(n.d.).Retrieved from https://toggl.com/

Xcode.(n.d.).Retrieved from https://en.wikipedia.org/wiki/Xcode