Object Design Document for TheraComm

Prepared by

Christian John	**200360001**
Iden Ellia	**200370502**
Joe Samano	**200365260**

Version History

Document History

Version #	Implemented by	Revision Date	Approved by	Approval Date	Reason
1	Christian John Joe Samano Iden Ellia	21/03/2019			Draft

Table of Contents

1 Introduction

1.1 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.

1.2 Interface Documentation Guidelines

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

1.3 Definitions, Acronyms, and Abbreviations

API	An application program interface (API) is a set of routines, protocols, and tools for building software applications.

1.4 References

Object Design Document Template.(2003, September 30).Retrieved from http://ee.hawaii.edu/~tep/EE467/BDLecs/html/odd.htm

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

1.5 Design Pattern

1.5.1 Model View Controller Pattern

The 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.

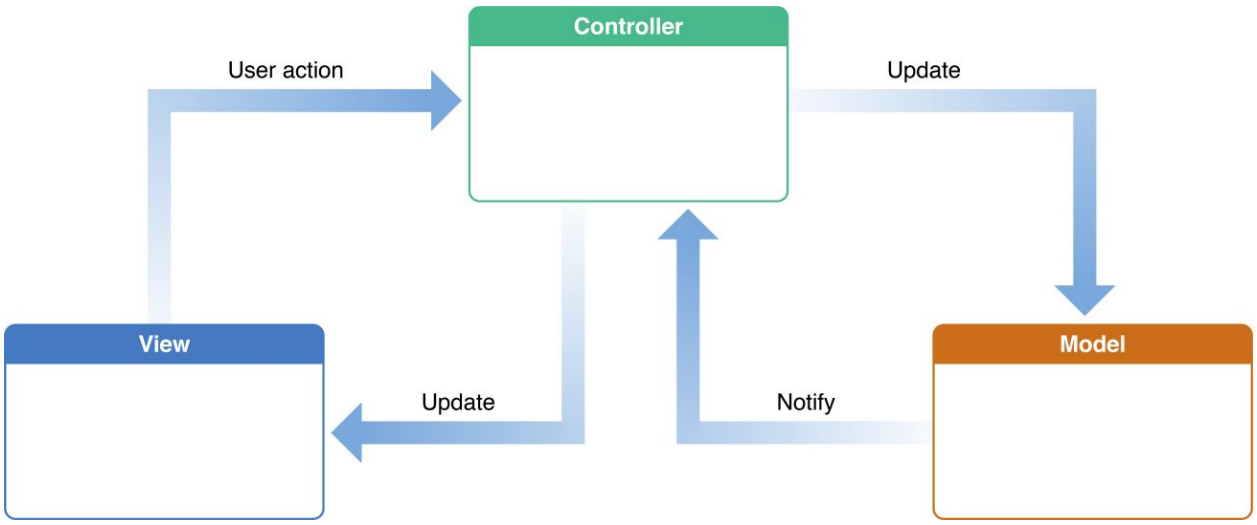

Figure 1. Model-View-Controller

3. 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 2.

3.1 Email System

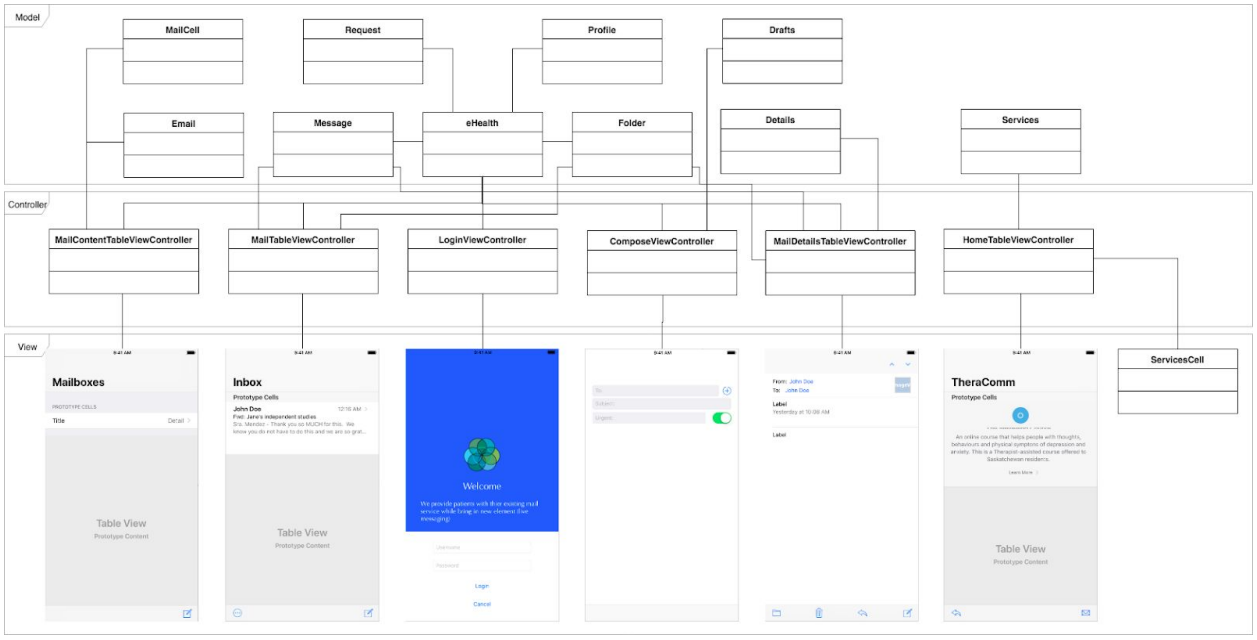

Figure 2. Class Diagram for the Email System

3.1.1 Model

3.1.1.1 Email

C

3.1.1.2 Request

C

3.1.1.3 Message

C

3.1.1.4 Profile

C

3.1.1.5 Folder

C

3.1.1.6 Drafts

C

3.1.1.7 Details

C

3.1.1.8 Services

C

3.1.1.9 MailCell

C

3.1.2 View

3.1.2.1 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.

3.1.2.2 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.`

3.1.3 Controller

3.1.3.1 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.

3.1.3.2 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.`

3.1.3.3 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`.

3.1.3.4 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.

3.1.3.5 ComposeViewController

C

3.1.3.6 HomeTableViewController

C

3.2 Chat System

3.2.1 Model

3.2.2 View

3.2.3 Controller