ChronoMap

Technical_documentation

Software Engineering – Group5

https://github.com/IamRichPeng/ChronoMap.git

Prepared by: Ruicheng Peng, Shusheng Song,

Haofan Cui, Wenhao Li

Nov. 2, 2018

Code design: Login

1. class design

   In IOS development, every screen, or View for technological terms, is specific to a class and should be written to a different source file for better code management. For the user management part, we designed two seperate screens. One for the login screen and another one for the sign in screen. As a result, there will be two classes for the two screens. The first one is LoginViewController and the second one is SignInViewController.

   An account class consists of two data members and one function member to check whether the information is correct and matched. The function will be inherited in the LoginViewController super class. Because this is the only place that the function will be called. As mentioned in Report 2 and shown below

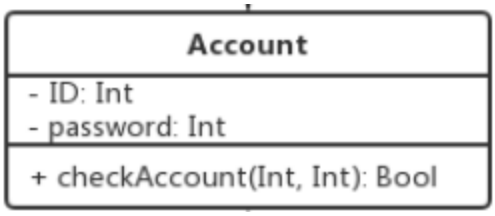

Figure 1

2. Pre-conditions

   As this part is the screens when the user launches the app, the only pre-condition will be the app is successfully installed and run.

   There is no post-condition.

3. LoginViewController

   There are two text input fields for user to type in his account, one for the user ID and the other for password. And there will be three buttons on the screen too. The user can either click “Login” or “Use as a guest” button to go into the next screen. If the user wants to use his account, he will need to click “Sign in” button to create the account for the first time use. So in this main class, there will be three functions accordingly to each

of the three buttons. The corresponding function will be called when any of the buttons is clicked.

```
//The class structure for the login screen looks like this
class LoginViewController: UIViewController
{
        Function LoginButton(){
        }       //The function that verifies the information input by    the user.
        Function SignInButton(){}       //Jumps to the Sign in screen.
        Function UseAsAGuest(){}      //Jumps to the Mission list screen.
}
```

4. SignInViewController

   On this screen, there are also two text input fields for user to type the user ID and password respectively. There is one “back” button on the top left corner which will jump back to the login screen. There is another button called “confirm” to create a new account, which will bring the user back to the login screen after the account is created successfully.

```
//The class structure for the sign in screen looks like this:
Class SignInViewController: UIViewController
{
  Function backbutton() {}          //Jumps to the login screen.
  Function confirm() {}             //Create a new account and jump to the login screen.
        }
```

Code Design: Mission List

1.The precondition of Mission list module is successfully Logged in or Use as a guest.

2.Then the scheduleViewController is the main class or you can say as a main view of the mission list module.

The first thing needs to be mentioned is that I store the data in core data (local data), which you can see in schedule2.xcdatamodeld or Mission+CoreDataClass.swift. Now there is an Assignment class data which store the name, time and priority of an event.

And in addMissionViewController, user can add an assignment into table view. Data are retrieved from textfield and pickerview and store in core data.

Then, you can see this view is basically filled by a TableView, the contents of it is fetched by "fetchRequest" function and there are other basic functions to construct the table view.

- addNewAssignment function is the action function of clicking the + button at the right-up corner which allows you to add a new event. When you click it, program will navigate to addMissionView and ask user to enter name and time and select the priority of the event. (When it runs on a simulator, you can click "shift+command+k" to shows the keyboard, you will notice that you can only input number at the text field of time) Once you finish your input and click "Confirm", the new mission will be added to the mission list(table view)
- prepare function is a segue function which passes the "index" of table view to the scheduleDetailsViewController.
- deleteAssignment function provides delete operation in core data.

3. Last, the scheduleDetailsViewController shows the detail of a specific event when you click the event on the table view. There are two buttons in it. When you click "cancel", you will go back to mission list. When you click "conform", you will be navigated to count-down page.

Code Design: Count Down

In my count-down page, **countdownViewController.swift** is the main file that control all the action in this page. I use **CoreData** to get the time to be send as timer**. CountdownTimer** function calculate the leftover time, with two button **startTimer** and **stopTimer**.
In the **CustomColors.swift**, I unique customization the colors to be showed in timer with timer pass.
In **ProgressBar.swift,** I set the animation needed for timer to run. What's more, all the animation func is set as public, so that **countdownViewController.swift** can use them.
In CountdownTimer.swift, I draw the circle needed for timer. Because I didn't find a library provide in swift had this features, I calculation the time with manually.

Code Design: Store

Class design:
Store is designed for users to use the reward which can motivate users to get rid of cell phones.

Belows are data types and operation signatures:
Store Data
Variables:
• gold: (Double). This variable stores the current gold of the user.
Functions:
• buy (Double): Double. This function will change the gold value (reduce the merchandise's price) when user choose to buy an item.
• checkBalance (Double,Double): Bool. This function will compare the current gold of user with the merchandise's price.

BUY

- Precondition: Users go to store page and want to buy a commodity

- Postcondition: System pops up a window to confirm the shopping

Confirm

- Precondition: Users choose a commodity to buy
- Postcondition: System will compare the current property with the price then offer the result.

Because our reward system function has not completed yet (we'll complete it before demo-2), the codes looked very sample now (just pop up "Purchase Failed"). In the future, we'll complete the whole function and calculate the property.