2018-6-25

MiniList

Design Document

POLITECNICO
MILANO 1863

Computer Science and Engineering

Design and Implementation of Mobile Applications

2017/2018

Prof. Luciano Baresi

ZHOU YINAN
10551242

Contents

ZHOU YINAN

Design and Implementation of Mobile Applications

1. Introduction

1.1 Purpose

This document is intended to help readers to understand and evaluate the design idea for the project “MiniList” developed for the course “Design and Implementation of Mobile Application ”in academic year 2017/2018.

MiniList is an easy-use time management app for iOS. The app is fully gesture based. Users can create various planlists to track their daily goals. The app is developed using Swift 4 and Xcode.

1.2 Scope

MiniList is designed as a to-do list app. Compared with various similar apps in the market, MiniList is fully gesture based and easy to use. The main features are:

- Create multiple planlists for different task types (ex, study, shopping…)
- Modify each planlist(ex, add new plans, finish a plan …)
- Enter Pomodoro-like timer to help users focus
- Past histoy tracked in calendar

1.3 Definitions, Acronyms, Abbreviations

MiniList : App name

MVC : Model View Controller

Today Extension : Widget in the home screen

1.4 Goals

The app provides the following features:

- [G1] Pull down to add a new plan
- [G2] Swipe right to mark a plan as finished
- [G3] Swipe left to delete the plan
- [G4] Single tap on the plan to enter to focus mode
- [G5] Focus mode is a timer. Users can choose a count-down or count-up timer
- [G6] When count-down timer finishes, a notification is sent to ask the user whether to mark the plan as finished
- [G7] Double tap on the plan to edit the name of the plan
- [G8] Shake to archive all completed plans
- [G9] Swipe right from the edge to segue to planlists collection view where users can create different planlists

- [G10] Users are able to create/delete planlists except the default one
- [G11] Pull up to segue to setting page
- [G12] Users are allowed to enable/disable TouchID/FaceID
- [G13] Users are allowed to switch language between Chinese and English
- [G14] Users are allowed to switch theme between day/night
- [G15] CalendarView where users are allowed to check the history of completed plans
- [G16] Today extension in iOS to let users have quick access to the app
- [G17] Users are allowed to change background for each planlist

1.5 Document Structure

The design document is divided into the following parts:

- Introduction: This section gives a brief description of MiniList
- Architecture Design: This section contains the details of design and implementation decisions
- User Interface Design: This section presents the user interface

2. Architecture Design

2.1 Overview

In this section, a detailed description of the app is provided. It begins by introducing the design pattern. Then different components of the app are described. Finally, use case analysis is provided.

2.2 Design Pattern

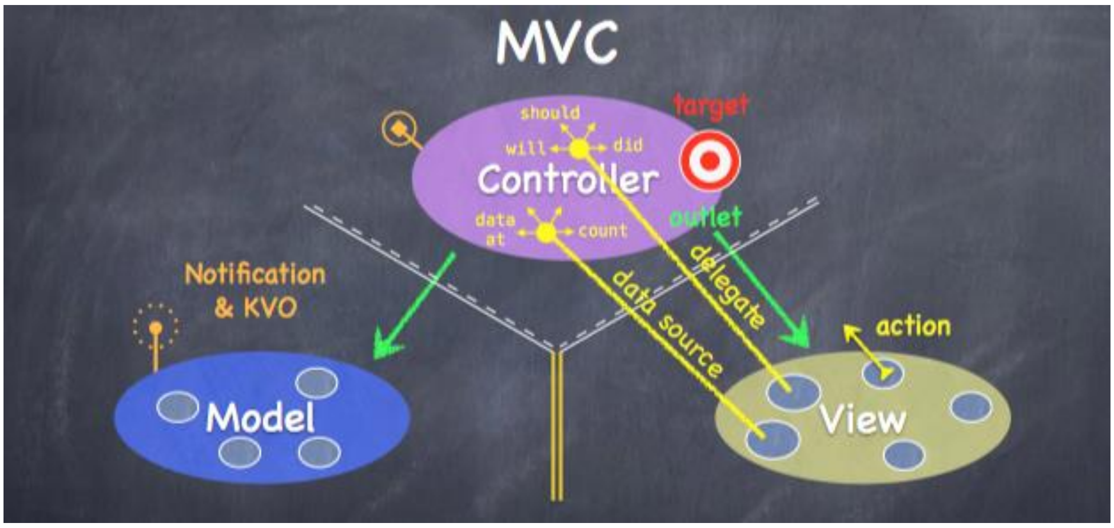
iOS applications use MVC by nature.

- Model: The model defines and represents data. Separating model from views allows code re-use. Also views are easier to change without modifying the data.
- View: The view represents UI. All UI staff should be separated from model.
- Controller: Controller acts as a bridge between model and view. It defines how your model is presented to user.

Note that the communication between these camps is crucial. Controllers can always talk directly to their model. Controllers can also talk directly to their view. The model and view should never speak to each other. The view can send message to controller in an indirect way. The controller set itself as the view's delegate. The view sends action when things happen in the UI. Sometimes view needs to synchronize with the controller. The model talks to the controller via a "radio station" like broadcast mechanism.

Sometimes the model is so simple that we do not need to implement it in a different file. Thus we can store the model in the controller directly. The important staff is that model and view should never talk to each other because model is UI-independent.

iOS MVC pattern is described as:



2.3 Components

2.3.1 Controllers

- MultiListViewController

It is responsible for managing different planlists. Users create/delete planlists via this controller.

- MiniListViewController

It is responsible for maintaining current plans. The core element of this controller is a table view which holds current plans and completed plans. This is the base ViewController for the app. Users can enter all other ViewController from here.

- CalendarViewController

It is responsible for managing calendars. The controller contains a calendar view for scanning dates and a table view for displaying plan history on a selected date.

- LockViewController

It is responsible to handle TouchID and FaceID access.

- SettingTableViewController

It is responsible to manage various settings. It contains a static table view. Users are allowed to customize their settings for : TouchID/FaceID, ScreenAlwaysOn, Langauges, Theme, Font Size, Background Paper.

- LanguageTableViewController

It is responsible for managing current supported languages.

- TimerViewController

It is responsible for managing a count-down timer.

- UpTimerViewController

It is responsible for managing a count-up timer.

- TimerUITabBarController

It is responsible for customizing timer tab bar appearance.

- NewPlanViewController

It is responsible for adding new plans to a planlist.

2.3.2 Views

- CalendarCustomCell

Calendarview is based on collection view. This custom view cell enables custom setting for each cell. Each cell is characterized by three things: dateLabel, dotLabel, selectedView.

- PlanIdCollectionViewCell

In MultiListViewController, users can maintain various planlists using collection view. Each cell is defined here.

- ProgressBarView

ProgressBarView defines how animation works in timer view.

2.3.3 Model

- AuthenticateManager

  It manages how TouchID/FaceID works.

- CountDownTimer

  It defines the timer behaviour.

- StorageManager

  In this app, data is saved using UserDefaults. StorageManager handles all the data staff.

2.3.4 Util

- Style

  It defines view appearance, including font color, font size and theme.

- PlanData

  It defines data structure for data storage.

- ZoomInAnimationController

  It defines zoom in interaction transition

- ZoomOutAnimationController

  It defines zoom out interaction transition

- SwipeInteractionController

  It defines how swipe interaction transition works.

- Constant

It defines some useful constants within the app, helping different components retrieve the same environment setting.

2.3.5 Today Extension

Today extension is what Apple calls a "Widget". By using today extension, users do not need to open the app to do some simple interaction. In MiniList, I implement today extension to let users have a quick access of current plans. Users can check the current plans and mark it as finished.

Today extension is a just a "small" version of the app. In order to share data with the main app, I create a group and use UserDefaults for data synchronization.

2.4 Use Case Analysis

This part of the document describes how users interact with MiniList in order to reach goals. Event flow, output and exception are provided.

2.4.1 Insert a new plan

Actor	Users
Goal	Insert a new plan into a certain planlist
Event Flow	Pull down to show textField for input Type a new plan Hit return button
Input Conditions	User in a certain planlist
Output Conditions	A new plan is inserted A new plan is saved using UserDefaults
Exception	New plan name is empty

Solutions: If user input invalid plan names, the input operation is ignored.

2.4.2 Mark a plan as completed

Actor	Users
Goal	Mark a plan as completed
Event Flow	Swipe right to mark the plan as completed
Input Conditions	The plan is not finished

Output Conditions	The plan is marked as finished
Exception	None

2.4.3 Mark a plan as incomplete

Actor	Users
Goal	Mark a completed plan as incomplete
Event Flow	Swipe right to mark the plan as unfinished
Input Conditions	The plan is finished
Output Conditions	The plan is marked as unfinished
Exception	None

2.4.4 Delete a plan

Actor	Users
Goal	Delete a plan from current planlist
Event Flow	Swipe left to delete a plan
Input Conditions	None
Output Conditions	The plan is deleted
Exception	None

2.4.5 Single tap a plan to show timer

Actor	Users
Goal	Show timer view

Event Flow	Single tap a plan Segue to timer view
Input Conditions	The plans are not finished
Output Conditions	None
Exception	None

2.4.6 Double tap a plan to edit plan name

Actor	Users
Goal	Change plan name
Event Flow	Double tap a plan Show textField to modify plan name
Input Conditions	The plan is not finished
Output Conditions	The plan name is changed
Exception	The new input name is empty

Solution: If the input name is invalid, the input operation is ignored.

2.4.7 Pull up to show setting page

Actor	Users
Goal	Show setting page
Event Flow	Pull up gesture
Input Conditions	None
Output Conditions	Segue to setting page
Exception	None

2.4.8 Swipe right from edge to show mutiple planlists page

Actor	Users
Goal	Show multiple planlists page
Event Flow	Swipe to right from edge
Input Conditions	None
Output Conditions	Segue to Multiple planlists view
Exception	None

2.4.9 Tap add button to add new planlist

Actor	Users
Goal	Add a new planlist
Event Flow	Single tap the add button cell
Input Conditions	None
Output Conditions	A notification box shows to let user input name of the new planlist A new planlist cell is added
Exception	The name of the new planlist already exists

Solution: If an input name is invalid, the input operation is ignored.

2.4.10 Long press a planlist cell to delete

Actor	Users
Goal	Delete a planlist cell
Event Flow	Long press a planlist cell

Input Conditions	None
Output Conditions	The selected cell is deleted
Exception	Users try to delete the default cell

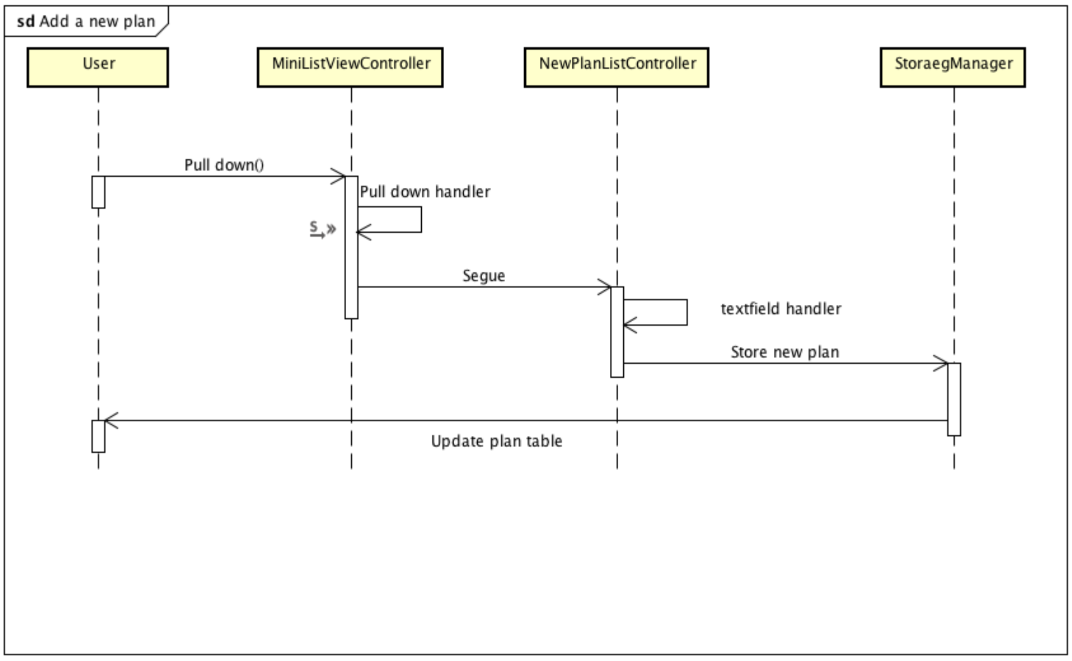
2.4.11 Single Tap a planlist cell to detailed planlist

Actor	Users
Goal	Show detailed planlist
Event Flow	Single tap a planlist cell
Input Conditions	None
Output Conditions	Segue to detailed planlist view
Exception	None

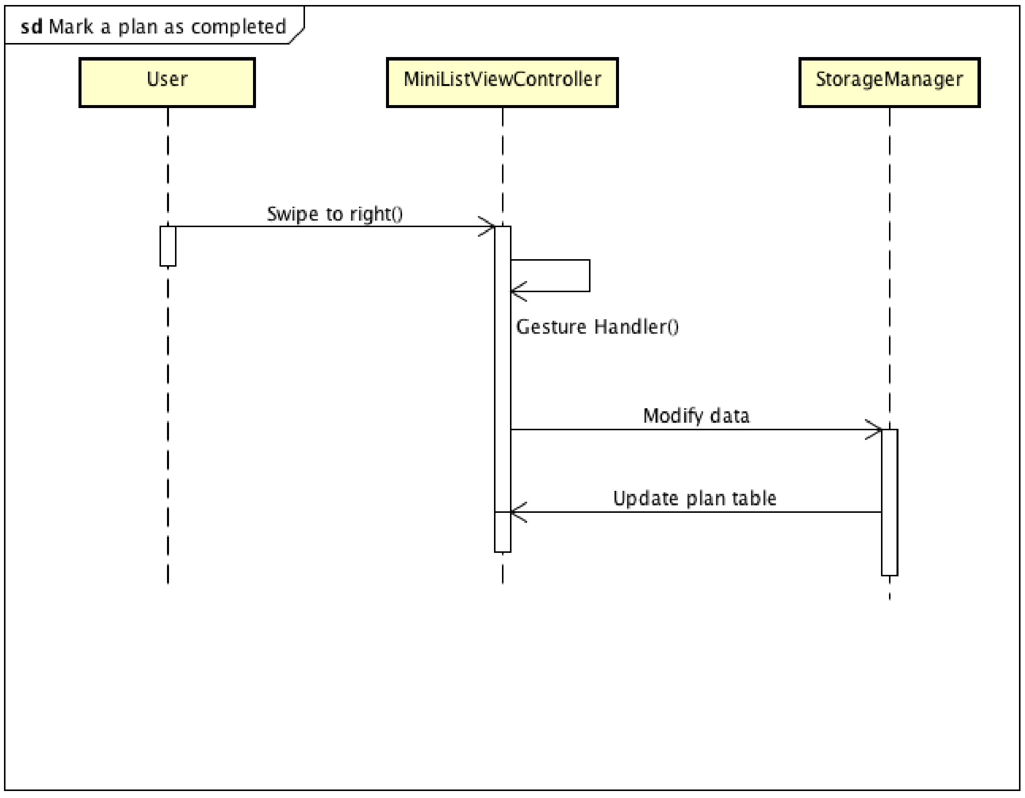
2.5 Sequential Diagram

In this section, some typical sequence diagrams are shown to illustrate how the operation flow works.

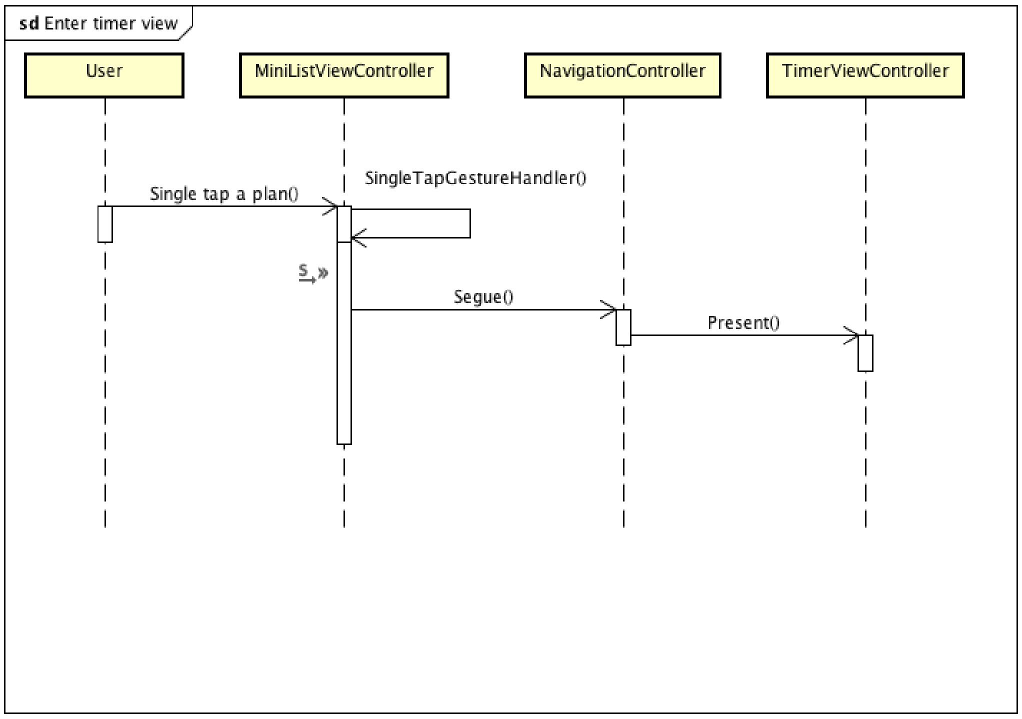
2.5.1 Add a new plan



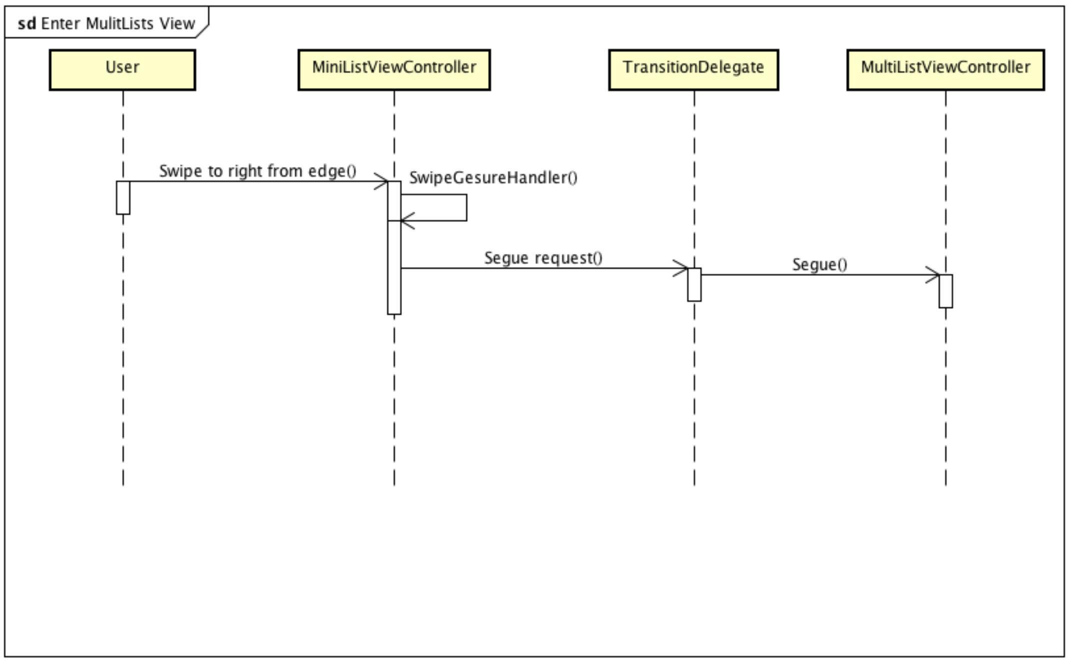
2.5.2 Mark a plan as completed



2.5.3 Enter timer view



2.5.4 Enter Multilists view

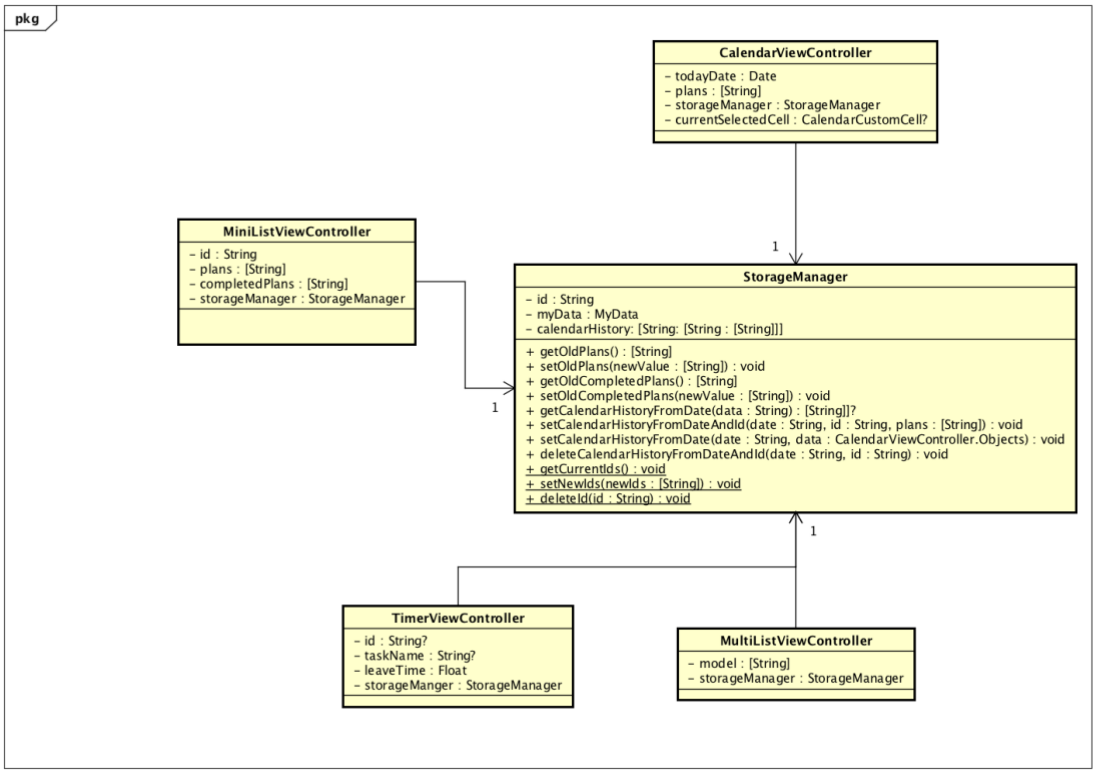


2.6 Class Diagram

Two class diagrams are shown to illustrate the software architecture.

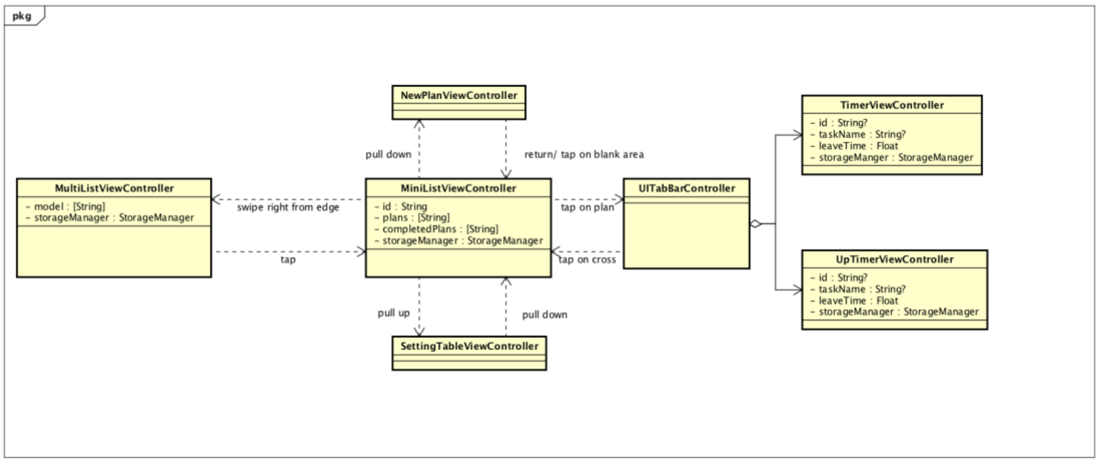
2.6.1 Data Storage Class Diagram

Data persistence is crucial in MiniList. StorageManager is used to manage all data persistence staff.


pkg
CalendarViewController
- todayDate : Date
- plans : [String]
- storageManager : StorageManager
- currentSelectedCell : CalendarCustomCell?
MiniListViewController
- id : String
- plans : [String]
- completedPlans : [String]
- storageManager : StorageManager
1
1
StorageManager
- id : String
- myData : MyData
- calendarHistory: [String: [String : [String]]]
+ getOldPlans() : [String]
+ setOldPlans(newValue : [String]) : void
+ getOldCompletedPlans() : [String]
+ setOldCompletedPlans(newValue : [String]) : void
+ getCalendarHistoryFromDate(data : String) : [String]]?
+ setCalendarHistoryFromDateAndId(date : String, id : String, plans : [String]) : void
+ setCalendarHistoryFromDate(date : String, data : CalendarViewController.Objects) : void
+ deleteCalendarHistoryFromDateAndId(date : String, id : String) : void
+ getCurrentIds() : void
+ setNewIds(newIds : [String]) : void
+ deleteId(id : String) : void
1
TimerViewController
- id : String?
- taskName : String?
- leaveTime : Float
- storageManger : StorageManager
MultiListViewController
- model : [String]
- storageManager : StorageManager

2.6.2 Segue Relation Class Diagram

MiniListViewContoller is the center of all segue operations. All the other views can be reached from MiniListViewController. Note that in the class diagram, some navigation controllers are not drawn for simplicity.



2.7 Some Design Decisions

2.7.1 Storage

In MiniList, storage and multi-file communication is done by UserDefaults. UserDefaults is not intended for large data storage. Usually, people use UserDefaults as a way to communicate between different classes. Also it is used to store user custom settings. Here I choose to use UserDefaults as storage method because of its simplicity. By using Codable protocal introduced in swift 4, it Is easy to store custom data.

Basically, I need to store two things for each planlist : current plans and completed plans. Given a planlist id, it should be possible to retrieve the plan data. So this information is stored in a dictionary.

For MultiPlanListView, I should get all the created planlist ids. This information is stored using list of strings.

For CalendarView, it is a bit complicated. Given me a date, I should be able to get all the created planlists id on that date. And for a given planlist id, I need to get all the archived plan data. So this information is stored using nested dictionary.

2.7.2 Background mode

In iOS, to make apps run in background is not so easy. iOS system is known for its strict control on apps. Usually the apps will be killed after 5 minutes in the background in order to save power.

Apple only allows 4 legal cases for background mode. Here in MiniList, in order for focus mode to work with screen off, I need to make it run in background. Here I choose to "fake" it as a music background player.

2.7.3 Cocoapods

In this project, I used three external projects in cocoapods.

1. GSTouchesShowingWindow-Swift

It is used only for video demonstration purpose. The place where I tap on the screen will be highlighted.

2. JTAppleCalendar
   It is used for easy implementation of calendar view.

3. Localize-Swift

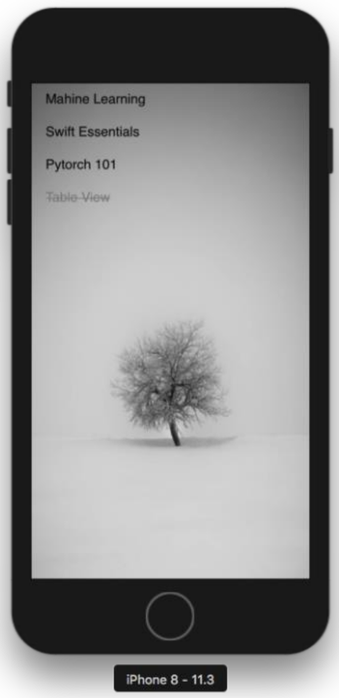
It is used for internalization. Language translation is just a mapping between source and target language.

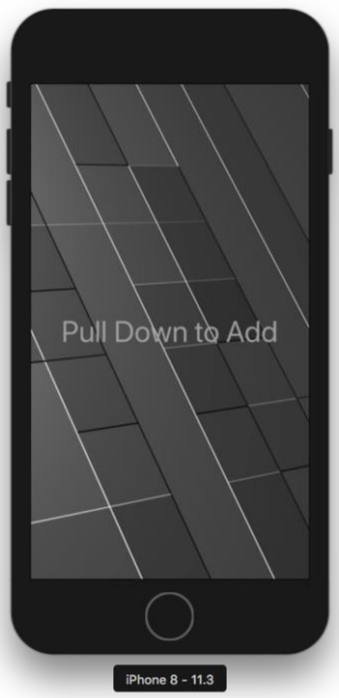
3 User Interface Design

In this section, we will see how UI is defined. MiniList is a gesture based app where the design decision is to make UI as simple as possible. All the unnecessary section lines or tabbars are deleted. To make the design compatible with the design decision, black and white color theme is chosen.

3.1 PlanListView

The following screen shots show the PlanListView. Users pull up to add new plan and pull down to segue to setting page.



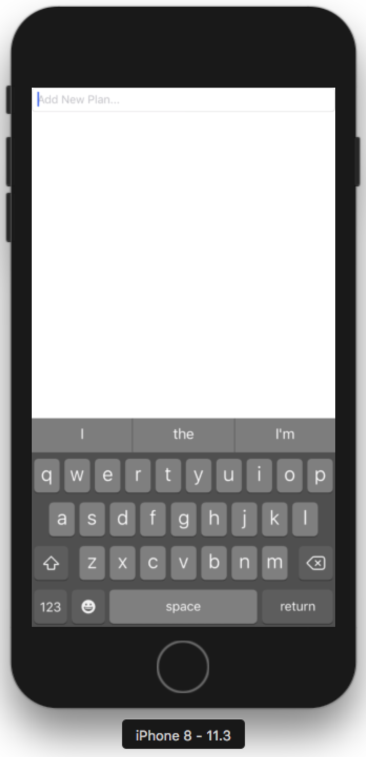


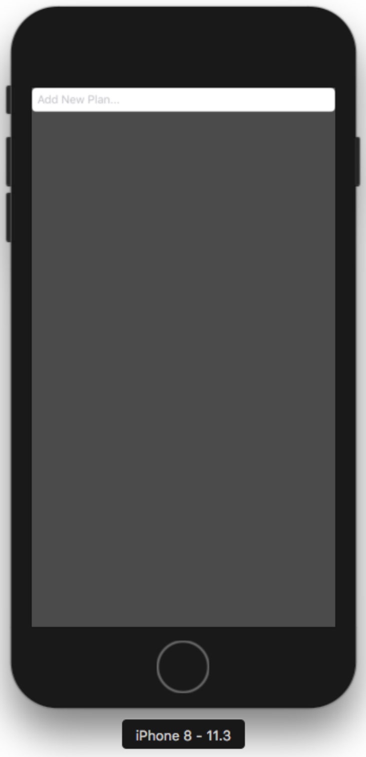
When users pull down/up, a segue will happen. There are two hidden labels at the top and bottom of the view. They are shown to give users a guide.

Users may choose different wall paper for planlists.

When there are no plans, “Pull Down to Add” label is shown.

3.2 NewPlanAddingPage

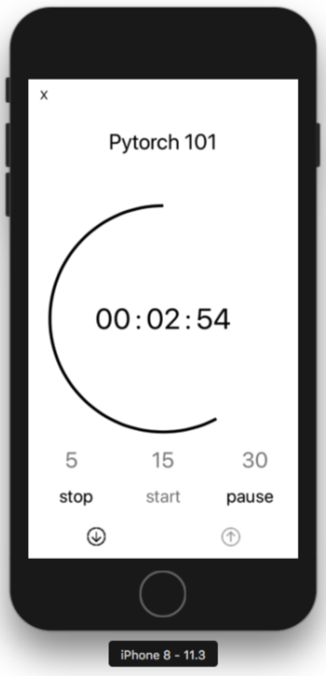


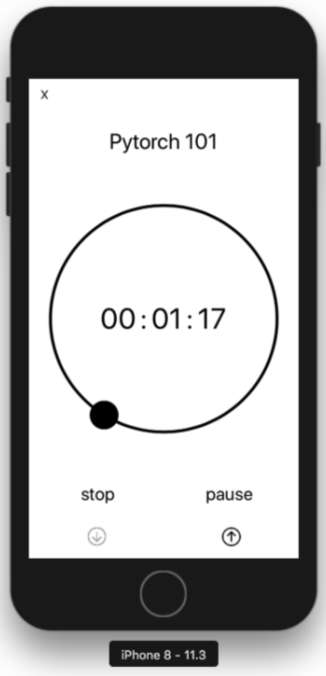


Users can go back to the planlist if he taps the return button or he taps the blank region.

There are two default theme for this app: night and dark mode.

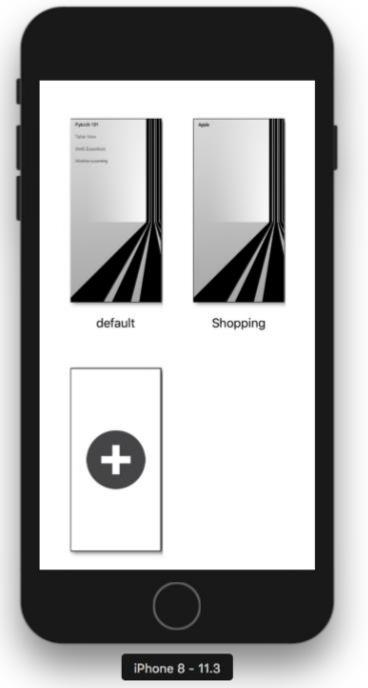
3.3 Timer View





A timer view is composed of two timers: count-down and count-up timer. They are inspired by pomodoro timers which help users to focus.

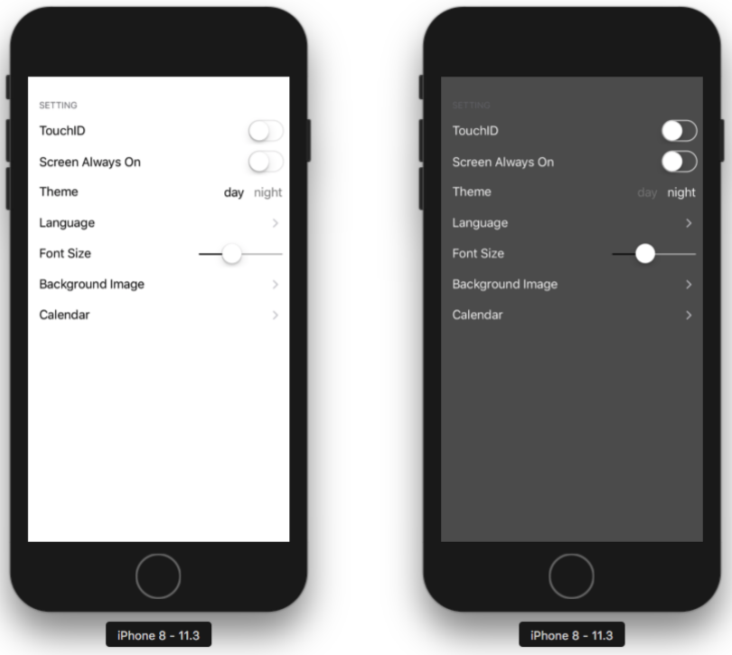
3.4 MultiPlanlists View



When User taps a planlist, a segue is performed using zoom in interaction transition. These transition interactions are custom defined, using ZoomIn/ZoomOut transition delegate.

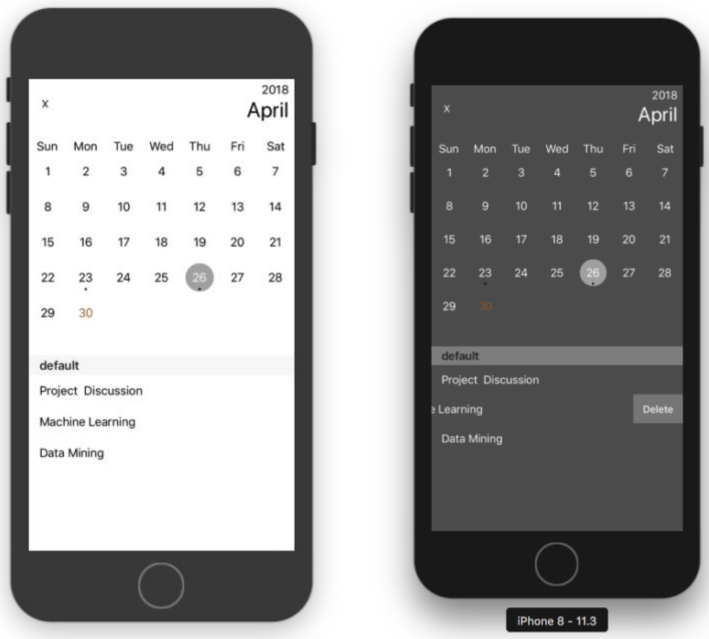
Users can create or delete different planlists. MultiPlanLists view is based on a collection view in Swift. Each cell has a snapshot to help users see what is inside each planlist. Users care allowed to add or delete an existing planlist. Note that the “default” planlist is necessary so it can not be deleted.

3.5 Settings View



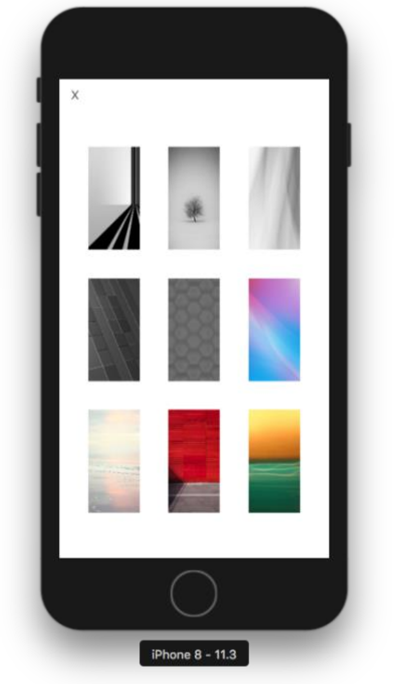
Setting View allows users to customize their app. In setting page, users can also segue to calendar view.

3.6 Calendar History



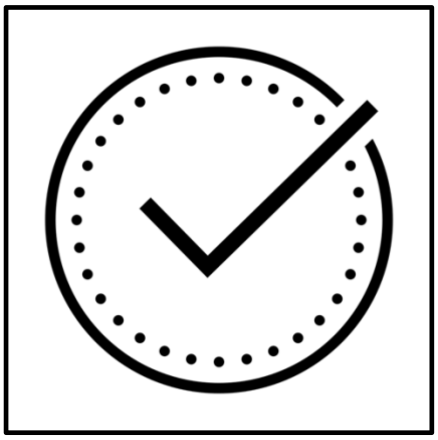
Calendar history is composed by two parts: a calendar view and a table view. When user selects a data cell, the archived plans are shown at bottom. If a certain date has archive plans, then the date cell has a dot annotation.

3.7 WallPaper Selection



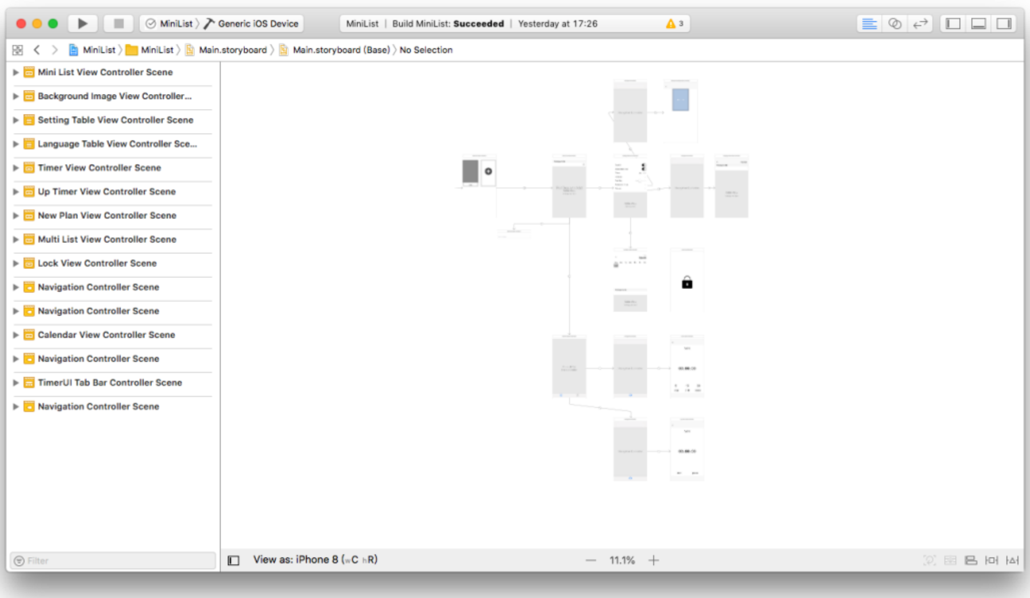
Users are allowed to choose their favorite wall paper for their planlist.

3.8 App Icon



3.8 Storyboards

Here is the overview for MiniList storyboard projects.


MiniList
Generic iOS Device
MiniList | Build MiniList: Succeeded | Yesterday at 17:26
MiniList 〉 MiniList 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 No Selection
Mini List View Controller Scene
Background Image View Controller...
Setting Table View Controller Scene
Language Table View Controller Sce...
Timer View Controller Scene
Up Timer View Controller Scene
New Plan View Controller Scene
Multi List View Controller Scene
Lock View Controller Scene
Navigation Controller Scene
Navigation Controller Scene
Calendar View Controller Scene
Navigation Controller Scene
TimerUI Tab Bar Controller Scene
Navigation Controller Scene
Filter
View as: iPhone 8 (wC hR)
11.1%

4 Summary

MiniList is a fully gesture based to-do list app. Users can easily track their daily staff and use a timer to help them focus. It is designed with minimalism design style.

The app is fully developed using Swift 4. All the functionalities except calendar and internalization are implemented using standard api.

Here is a summary of the project proposal and the final implemented app.

1.[Done] table view presenting list of plans

2.[Done] swipe left to mark the plan as finished

3.[Done] swipe right to delete the plan

4.[Done] single tap to enter focus mode

5.[Done] double tap to edit the plan

6.[Done] pull down to add new plans

7.[Modified] pull up to segue to new view with past finished plan history. In the history users are allowed to view all the finished plans and search for it by time.

History searching is implemented using calendar view. Pull up operation segues to setting page and calendar view is inside the the setting page.

8.[Modified] data persistence and sync with iCloud drive

Data persistence is implemented using UserDefaults. Sync with iCloud drive is not implemented because it requires a payed apple developer account.

9.[Done] widget in iOS which allows users to quick access without entering the app.

** New functions **

10.[Done] count-up timer is added along with the count-down timer

11.[Done] setting page is added where users are allowed to customize the app: TouchID/FaceID, screenAlwaysOn, Theme, Language, Font Size

12.[Done] multiple planlists view where users are allowed to create various types of plans. Custom ZoomIn/ZoomOut transition is implemented and snapshotview is added for each planlist.

5 Bibliography

[1] DIMA Course Website http://home.deib.polimi.it/baresi/dima.htm

[2] Stanford CS193P iOS development http://web.stanford.edu/class/cs193p/cgi-bin/drupal/

[3] Apple Official Website https://developer.apple.com