НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА ЭКОНОМИКИ

ДОМАШНЕЕ ЗАДАНИЕ №3

посвященный исследовательскому семинару

«Погружение в iOS-разработку»

МОСКВА 2024

Тема: Добавить функцию записи желаний, которые можно исполнить позже.

Цель: Отточить навыки работы с таблицами и научиться сохранять простую информацию в пользовательских настройках по умолчанию.

Описание задачи

Вы продолжите работу над приложением WishMaker. Если вы пропустили эту задачу, попросите помощника предложить вам лучшее решение. Вам нужно будет добавить вторую функцию исполнения желаний: возможность сохранять желания для последующего исполнения! Для этого потребуется табличный режим. Вы будете разрабатывать такое же приложение WishMaking и в HW4, поэтому постарайтесь написать код так, чтобы он не мешал вам в будущем.

Требования к заданию

• Для выполнения этой задачи необходимо удалить Storyboard и использовать Swift + UIKit. •
Необходимо использовать одну из следующих архитектур: MVC, MVVM или VIPER. • Чтобы использовать код
из этого руководства или из интернета, необходимо сначала его понять.

Оценка

Оценка задачи	
1	При запуске WishMakerViewController отображается кнопка записи.
2	При нажатии кнопки отображается новый WishStoringViewController.
3	WishStoringViewController имеет табличное представление.
4	Таблица имеет ячейки.
5	Ячейки разделены на две группы: AddWishCell и WrittenWishCell.
6	Пожелания, созданные с помощью AddWishesCell, добавляются в список написанных желаний.
7	Написанные пожелания сохраняются в UserDefaults и отображаются при повторном открытии приложения.
8	Письменные пожелания могут быть удалены.
9	Письменные пожелания можно редактировать.
10	CoreData используется для сохранения желаний или распространения желаний через Share.

Отказ от ответственности

Просроченная сдача работы наказывается снижением оценки на количество дней, прошедших с даты сдачи. Максимальное наказание — минус 5 баллов. Пожалуйста, помните об этом и планируйте свою неделю соответствующим образом.

Задавайте любые вопросы по этой задаче и разработке iOS в целом. Наша цель — помочь вам стать лучшим iOS-разработчиком . Преподаватели и ассистенты всегда готовы помочь!

Материалы, содержащие необычные решения, исключительный стиль кода, шаблоны Использование и развлекательные функции могут привести к начислению дополнительных баллов. Это не подлежит обсуждению. Повторение или копирование решений, за которые ранее были получены бонусные баллы, не приносит бонусных баллов.

Начав выполнять это домашнее задание, ваш плохой код снизит вашу оценку. Не забудьте убрать магические числа и писать чистый код.

Учебное пособие

Продолжайте работать над проектом из предыдущего домашнего задания. Но при создании коммитов пишите HW-3: в сообщениях к коммитам. Старайтесь выполнять коммиты по каждому пункту отдельно, чтобы получить бонусные баллы и добиться блестящего успеха. позже в (реальной) жизни.

Пункт 1:

Вы уже знаете, как добавить кнопку. Давайте сделаем это вместе ещё раз, в последний раз.

Создайте addWishButton следующим образом:

```
private пусть addWishButton: UIButton = UIButton(тип: .system)
```

Предоставление типа переменным, полям и свойствам сокращает время сборки проекта. Автоматизация этого — хорошая привычка не только в Swift. Мы также видим систему типов при инициализации кнопки. Это добавляет к кнопке анимацию нажатия, что позволяет нам выполнять меньше действий. Вы ведь любите меньше действий, верно?

Мы хотим, чтобы эта кнопка располагалась под ползунками, поэтому нам придётся изменить метод configureSliders и добавить два новых метода. Первый метод — configureAddWishButton. Он разместит кнопку внизу экрана и задаст её высоту, цвет, текст и т. д. Я буду использовать макет с закреплёнными элементами (pin layout). Если вы хотите использовать ограничения Vanilla или SnapKit, вам придётся сделать это самостоятельно.

```
92. private func configureAddWishButton() { view.
            addSubview(addWishButton) 93. addWishButton.
setHeight(Constants.buttonHeight) 94. addWishButton. pinBottom(to: view,
Constants.buttonBottom) 95. addWishButton. pinHorizontal(to: view, Constants.buttonSide) 96. 97. 98.
addWishButton . backgroundColor = . white


            addWishButton.setTitleColor(.systemPink, для: .normal) 99. 100.
addWishButton.setTitle(Constants.buttonText, для: .normal) 101. 102. addWishButton.layer.cornerRadius =

Constants.buttonRadius
103. addWishButton.addTarget(self, action: #selector(addWishButtonPressed), for: .touchUpInside) 104.}
```

Если вы скопируете этот код и вставите его в свой проект, возникнут ошибки сборки. Это сделано намеренно. Добавьте константы кнопок самостоятельно. В строке 93 мы добавляем нашу кнопку как подпредставление экрана WishMakerViewController . Без неё строки 94–96, определяющие положение кнопки на экране, вызовут исключение времени выполнения. Мы любим исключения времени выполнения, но в реальных проектах, а не в проекте домашнего задания. Строки 98–100 задают цвета кнопки. Строка 102 задаёт кнопке скруглённые углы. Строка 103 задаёт функцию, которая будет вызвана, как только пользователь поднимет палец внутри кнопки.

Следующий элемент addWishButtonPressed :

```
@objc
private func addWishButtonPressed() { // это будет сделано
        позже!
}
```

Измените метод configureSliders так, чтобы нижняя часть ползунка была прикреплена к верхней части кнопки. Запустите приложение и посмотрите, что оно делает.

Надеюсь, вы получили исключение во время выполнения из-за того, что не вызвали addWishButtonPressed перед configureSliders. Не забывайте делать так в будущем. После добавления кнопки она должна выглядеть примерно так:

Изображение 11

Пункт 2:

У нас уже есть метод addWishButtonPressed , который будет вызываться при нажатии кнопки. Всё, что нам нужно сделать, — это открыть новый экран. Это можно сделать двумя способами. Первый — отобразить экран с помощью метода present . Второй вариант — добавить контроллер навигации в наше приложение. Второй вариант мы рассмотрим в следующем домашнем задании, поэтому давайте его рассмотрим. Прежде чем мы это сделаем, нам нужно создать новый файл с классом WishStoringViewController . Не забудьте сделать его final и унаследовать от UIViewController.

```
окончательный класс WishStoringViewController: UIViewController {
        переопределение функции viewDidLoad() {
                вид.backgroundColor = .blue
        }
}
```

Мы можем представить наш новый экран следующим образом:

```
присутствует(WishStoringViewController(), анимированный: true)
```

Это действие разместит экран WishStoringViewController над нашим главным экраном. Это можно увидеть, используя иерархию отладочных представлений. Вы уже должны знать, что это такое и как это сделать.
Благодаря этому мы можем закрыть экран, смахнув его вниз и перетащив за верхнюю часть. Это не лучший способ, и дизайнеры обычно просят добавить сверху линию (общепризнанный индикатор экрана, который можно закрыть, перетащив его вниз), кнопку «Назад» или кнопку закрытия. Чтобы закрыть представленный экран в коде, можно использовать функцию .dismiss() . Это отличается от подхода с использованием контроллера представления навигации.

Пункт 3:

Чтобы добавить табличное представление, создадим метод configureTable():

```
приватная таблица let: UITableView = UITableView(frame: .zero)

private func configureTable() { view.addSubview(table)
        table.backgroundColor = .red
        table.dataSource = self table.separatorStyle
        = .none

        table.layer.cornerRadius = Constants.tableCornerRadius

        table.pin(to: view, Constants.tableOffset)
}
```

Помните, что функции располагаются после переопределенных методов, а поля — перед переопределенными методами. Этот код должен вызывать ошибку в строке, где мы пытаемся назначить экран источником данных для таблицы. Это связано с тем, что для предоставления данных наш экран WishStoringViewController должен соответствовать протоколу UITableViewDataSource :

```
// MARK: - расширение UITableViewDataSource
WishStoringViewController: UITableViewDataSource { ... }
```

Это должно быть в исходном классе WishStoringViewController . Будет отображена новая ошибка, указывающая на отсутствие двух методов. Нажмите кнопку «Исправить» внутри ошибки, и отсутствующие функции будут добавлены в наше расширение. Все расширения, расширяющие протокол, лучше маркируются, что является хорошей практикой кодирования. Два новых метода, которые были добавлены, не содержат контента. Чтобы понять, что делает каждый из них, щелкните правой кнопкой мыши по неизвестной функции и выберите «Перейти к определению». Код Apple обычно содержит комментарии, описывающие назначение функции. Чтобы проверить, получили ли вы точку возврата 0 и UITableViewCell(), запустите приложение и посмотрите на красную таблицу.

Пункт 4:

Чтобы добавить ячейки, нам нужно сделать несколько вещей. Начнём с создания массива желаний с одним временным желанием:

```
private var wishArray: [String] = ["Я хочу добавить ячейки в таблицу"]
```

Этот массив будет хранить наши пожелания для сеанса приложения. Закрытие приложения удалит записи, добавленные во время сеанса. Пожелания, добавленные через код, будут появляться при каждом запуске приложения, что не идеально. Мы разберемся с этим позже. У нас есть массив, давайте свяжем количество желаний с количеством ячеек в таблице:

```
вернуть wishArray.count
```

Это отлично, но нам нужен WrittenWishCell вместо ячейки по умолчанию.

Давайте создадим пользовательскую ячейку:

```
окончательный класс WrittenWishCell: UITableViewCell {
        static let reuseId: String = "WrittenWishCell"

        Константы частного перечисления
                { static let wrapColor: UIColor = .white
                static let wrapRadius: CGFloat = 16
                static let wrapOffsetV: CGFloat = 5
                static let wrapOffsetH: CGFloat = 10
                static пусть wishLabelOffset: CGFloat = 8
        }

        private пусть wishLabel: UILabel = UILabel()

        // MARK: - Жизненный цикл
        переопределить init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) { super.init(style: style, reuseIdentifier:
                reuseIdentifier)

                configureUI()
        }

        @available(*, unavailable) требуется init?
        (coder: NSCoder) { fatalError("init(coder:) не был
                реализован")
        }

        func configure(c пожеланием: String) { wishLabel.text =
                пожелание
        }

        частная функция configureUI()
                { selectionStyle = .none
                backgroundColor = .clear

                пусть обернется: UIView = UIView()
                addSubview(wrap)

                wrap.backgroundColor = Constants.wrapColor
                wrap.layer.cornerRadius = Constants.wrapRadius
                wrap.pinVertical(to: self, Constants.wrapOffsetV) wrap.pinHorizontal(to: self,
                Constants.wrapOffsetH)

                wrap.addSubview(wishLabel)
                wishLabel.pin(to: wrap, Constants.wishLabelOffset)
        }
}
```

Жизненный цикл ячейки включает инициализатор, но также есть метод prepareForReuse . Это важно, поскольку вместо создания новых ячеек после прокрутки таблицы оптимизированы для многократного использования одной и той же ячейки. Это означает, что следует быть очень осторожным с представлениями ячеек. Мы можем случайно добавить один и тот же wishLabel каждый раз. Это негативно скажется на производительности таблицы, но хуже всего то, что текст предыдущего пожелания будет отображаться за текстом текущего пожелания.

Чтобы избежать этого, мы упорядочиваем представления после инициализации ячейки в методе configureUI . Мы также проводим косметическую подготовку. Во-первых, удаляем стиль выделения. В противном случае нажатые желания без причины становились бы серыми. Фон ячейки делается прозрачным, чтобы открывать таблицу под ней. Мы также настраиваем содержимое метки с помощью открытого метода configure.
Вместо желания эти методы обычно принимают модель.

Следующий шаг — создать ячейку желаний вместо нашего временного решения UITableView. Это не так просто, как может показаться. Для этого нам нужно удалить из очереди повторно используемую ячейку.

Это можно сделать следующим образом:

```
func tableView( tableView:
        _   UITableView, cellForRowAt
        indexPath: IndexPath
) -> UITableViewCell {
        пусть ячейка = tableView.dequeueReusableCell(
                withIdentifier: WrittenWishCell.reuseId, для: indexPath

        )

        пусть wishCell = ячейка как? WrittenWishCell else { return cell }

        wishCell.configure(c: wishArray[indexPath.row])

        вернуть wishCell
}
```

Здесь мы используем метод dequeueReusableCell табличного представления для создания ячейки. Проблема в том, что она создаётся с понижением типа до UITableViewCell. Это означает, что мы не можем вызвать метод configure с пожеланием, чтобы добавить пожелание в wishLabel. Мы приводим его к классу WrittenWishCell , опционально преобразуя его в дочерний тип. Запустите этот код. Посмотрите, что появится на экране. Надеюсь, вы не прочитали эту строку перед запуском приложения, так как этот код не приведёт к исключению во время выполнения. Каждое исключение во время выполнения, которое вы получаете на этом этапе, крайне важно и показывает, на что стоит обращать внимание при разработке приложения для iOS. В данном случае, перед извлечением ячейки из очереди нам необходимо её зарегистрировать. Добавьте следующую строку в метод configureTable() :

```
table.register(WrittenWishCell.self, forCellReuseIdentifier: WrittenWishCell.reuseId)
```

Теперь вы знаете, как зарегистрировать ячейку.

Пункт 5:

Группы, на которые мы делим ячейки, называются секциями. Чтобы создать новую секцию, необходимо сделать следующее: • Переопределить
метод numberOfSections протокола UITableViewDataSource так, чтобы он возвращал значение 2.
Не забудьте добавить это число в Константы.
• В функции numberOfRowsInSection верните количество ячеек для каждой секции. В нашем случае лучше всего
У вас есть одна ячейка AddWishCell. Используйте if или switch для переменной section.
• Используйте if или switch в методе cellForRowAt . Используйте indexPath.section. В разделе 0 мы должны
создаем AddWishCell, в разделе 1 оставляем код, который мы написали для WrittenWishCell.

Некоторые из вас могут спросить: «Где или когда мы написали AddWishCell?». Мы действительно его не писали. Это задание для вас. Возможно, вам стоит следовать инструкциям пункта 6, чтобы сделать это за один раз. Удачи!

Пункт 6:

Чтобы добавить пожелание в таблицу, нам нужно, чтобы AddWishCell имел несколько вещей: • UITextView для ввода желания. • Кнопка для добавления желания.

При нажатии кнопки значение UITextView (если оно не пустое) добавляется в массив wishArray как пожелание. Для этого нам понадобится необязательное замыкание в ячейке AddWish: var addWish: ((String) -> ())?. Это позволит нам изменять wishArray изнутри ячейки.

После обновления wishArray выполните table.reloadData()

Пункт 7:

Чтобы сохранить пожелания в UserDefaults, выполните следующие действия:

Создайте сохраненный экземпляр пользовательских значений по умолчанию, чтобы не обращаться к стандарту каждый раз:

```
private пусть defaults = UserDefaults.standard
```

Каждый раз, когда вы изменяете wishArray, сохраняйте его до значений по умолчанию:

```
defaults.set(stringArray, forKey: Constants.wishesKey)
```

При загрузке WishStoringViewController получите доступ к сохраненным данным и назначьте их wishArray:

```
defaults.array(forKey: Constants.wishesKey) как? [Строка]
```

Закончив, не забудьте удалить временное желание.

Заключение:

В этом руководстве рассказывается, как использовать UITableView , и демонстрируются проблемы, с которыми вы можете столкнуться при работе с табличными представлениями. Вы больше не увидите так много кода, поскольку уже знаете, как делать некоторые вещи. В будущем объём кода в руководстве будет уменьшаться в прямой зависимости от ваших знаний.

Вы можете успешно завершить это руководство и получить 7 баллов. Это хорошая оценка. Чтобы заработать больше, наберите оставшиеся баллы, изучая и применяя что-то новое.