Задания курса «Разработка под iOS. Взлетаем»

Задание 1. GCD

Переходим к задачам:

- Создайте 2 потока, используя `Thread`.
- Реализуйте увеличение общего счетчика в цикле от 0 до 10000 в каждом потоке. Проверьте, что значение счетчика не равно 20000 после завершения функции.
- С помощью `NSLock` измените код, чтобы счетчик стал считаться корректно. Проверьте, что теперь значение счетчика равно 20000.
- Придумайте, как с помощью вызова метода `.sync()` у GCD очереди спровоцировать deadlock.

Задание 2. Операции

Реализуйте аналог функции `sleep` с помощью `DispatchGroup` и метода `asyncAfter()`.
То есть необходимо написать функцию с именем `wait(for: TimeInterval)`. Функция принимает на вход вещественное число секунд и блокирует текущий поток до тех пор, пока указаное число секунд не пройдет.
Для проверки необходимо до и после вызова `wait(for:)` напечатать текущее время.

Задание 3. Приложение Заметки. Создаём операции

Мы продолжаем развивать приложение, которое вы создали в первом курсе «Разработка под iOS. Начинаем».
Возьмите предоставленную базовую реализацию асинхронной операции: скачать .zip (35 Кб). И реализуйте сабклассы операций специфичных для приложения:

- Получение и отправка данных на сервер: без реализации, только интерфейс. Реализацию вы сможете написать после модуля «Работа с сетью», который

посвящен работе с сервером. А сейчас все операции должны возвращать ошибку о недоступности сервера:
— Базовая операция для работы с бекендом `BaseBackendOperation`;
— Операция сохранения заметок в бекенд `SaveNotesBackendOperation`;
— Операция загрузки списка заметок из бекенда `LoadNotesBackendOperation`.

- Загрузка и сохранение заметок на локальный диск. В реализации этих операций необходимо использовать записную книжку`FileNotebook`, вы создали её в рамках первого курса «Разработка под iOS. Начинаем»:
— Базовая DB операция `BaseDBOperation`;
— Операция сохранения заметки в DB `SaveNoteDBOperation`;
— Операция загрузки списка заметок из DB `LoadNotesDBOperation`;
— Операция удаления заметки из DB `RemoveNoteDBOperation`.
- Агрегирующие операции для работы из UI:
— Операция сохранения заметки `SaveNoteOperation`. Должна вызывать операции `SaveNotesBackendOperation` и `SaveNoteDBOperation`. Должна вызываться из UI по событию окончания редактирования.
— Операция загрузки списка заметок `LoadNotesOperation`. Должна вызывать `LoadNotesBackendOperation` и `LoadNotesDBOperation`. Если полученные с сервера данные не соответствуют данным локально, то источником истины нужно считать сервер. В этом сценарии нужно обновить локальные данные. Она вызывается при отображении списка заметок.
— Операция удаления заметки `RemoveNoteOperation`. Должна вызывать `SaveNotesBackendOperation` и `RemoveNoteDBOperation`. Должна вызываться из UI по событию удаления заметки.
- **Важно!** Каждая операция должна быть в отдельном файле, имя которого совпадает с именем операции.

Задание 4. Сетевой запрос в iOS

Загрузите три любые картинки с известными url из сети и отобразите их в таблице (UITableView).
Примеры URL:
1. https://avatars.mds.yandex.net/get-bunker/60661/5ec62cb755193c37a6ec19a826b3891780eead2a/orig
2. https://avatars.mds.yandex.net/get-bunker/135516/7a83e9a35b9c1537ba8fe3a5cf1ef182838a11be/orig

Задание 5. Протокол Decodable на практике

Подготовка
Для решения задачи вам понадобится логин на github. Если вы ещё не зарегистрированы на github.com, пожалуйста, сделайте это. Также потребуются созданные gists в интерфейсе github gists. Достаточно будет создать хотя бы один-два gist.
Инструкция
С помощью github API получите список публичных gists и выведите полученную информацию в консоль.
Структура, описывающая gist должна содержать поля со следующими данными:

- дата создания;
- логин владельца gist;
- количество комментариев;
- информация о файлах gist.

Для получения публичных gists можно использовать один из методов:
GET /gists
GET /gists/public
Описание API для работы с gists есть по ссылке.

Заготовка
Ниже вы найдёте структуру кода, которая поможет вам реализовать задание.
В качестве ответа на задачу скопируйте ваш код в текстовое поле или прикрепите файл с кодом.

```
import Foundation
import PlaygroundSupport

PlaygroundPage.current.needsIndefiniteExecution = true

struct Gist: Decodable {

    // implement Gist struct
    // ...

    let files: [String: GistFile]
    let owner: Owner
}

struct Owner: Decodable {

    // implement Owner struct
    // ...

}

struct GistFile: Decodable {

    // implement GistFile struct
    // ...

}

func load() {

    // load gists data with URLSession.shared.dataTask and print loaded
gists info
    // ...

}

load()
```

Задание 5. Получение своих файлов с github

Подготовка

Для решения задачи вам понадобится логин на github. Если вы ещё не зарегистрированы на github.com, пожалуйста, сделайте это. Также потребуются созданные gists в интерфейсе github gists. Достаточно будет создать два gist (хотя бы один из них должен быть приватным).

Инструкция

С помощью API github реализуйте получение списка своих приватных и публичных gists. Выведите полученную информацию в таблицу UITableView. В результате должно получиться приложение, в котором отображается информация о gists вашего аккаунта в виде списка.
В каждой ячейке таблицы отразите следующую информацию о gist:

- количество комментариев;
- дату создания;
- опционально — любые другие данные gist, например, url.

Для получения списка необходимо отправлять в запросе токен авторизации. Метод API:
GET /users/:username/gists (чтобы получить свои gists, `:username` необходимо заменить на ваш логин).
Формат для отправки токена:
`Authorization: token ВАШ_ТОКЕН`

Получение токена

Есть несколько способов получения токена для выполнения данного задания, подробно они описаны в документации. Для простоты можно использовать токен, сгенерированный в интерфейсе github. Такой способ накладывает некоторые ограничения на ваше приложение, но позволяет быстро получить токен и перейти к запросу своих приватных данных.

Важно отметить, что токен выдается на определенный **scope.** Например, для работы с gist потребуется указать scope *gist*. Описание возможных значений параметра scope можно найти в документации.
В качестве решения отправьте:

- скриншот/гифку работающего приложения;
- код с выполнением запроса к API github.

Примечание: если в коде явно указан токен авторизации (строковым литералом), его лучше подменить другой строкой, потому что авторизационный токен можно использовать для получения доступа к данным вашего аккаунта.
Дополнительный материал

- Описание API для работы с gists есть по ссылке.
- Описание процесса авторизации через github есть по ссылке.
- Описание создания токена в интерфейсе github есть по ссылке.
- Описание возможных scope есть по ссылке.

Критерии проверки
Сокурсники и преподаватели проверят скриншот/гифку вашего приложения: вся информация должна отображаться в таблице и быть зафиксирована. А также посмотрят ваш код: выполняется ли GET запрос к API github с токеном авторизации.

Задание 6. Выполнение POST запроса

Отправьте любой заранее известный текст для публикации в качестве gist на вашем github:

- Текст вынесите в строковую константу.
- Используйте github API для отправки текста:
- POST /gists
- Описание API для работы с gists есть по ссылке.

Загрузите ваш код на проверку.
Примечание: задание можно выполнить в playground, создание отдельного проекта не требуется. Токен можно поместить в строковый литерал или считывать строку с токеном из консоли. Можно использовать токен, полученный

при выполнении предыдущего задания, или сгенерировать новый на странице настроек. Перед отправкой кода на проверку убедитесь, что НЕ отправляете вместе с кодом свой токен (его можно заменить на любую строку, например, "my_secret_token")

Критерии проверки

Сокурсники и преподаватели проверят ваш код: известен ли текст до выполнения кода, выполняется ли POST запрос и отправляется ли токен авторизации.

Задание 7. Приложение Заметки. Работа с API github gists

Расширьте функциональность приложения Заметки работой с API github gists. В gist будет храниться JSON, описывающий список созданных заметок.

Описание API для работы с gists есть по ссылке.

Требования

1. Реализовать загрузку списка заметок из gist. Файл в gist для хранения массива заметок в виде JSON должен называться "ios-course-notes-db". Если при загрузке файл с именем "ios-course-notes-db" не был найден, нужно показывать локальный список заметок, иначе - пустой список.
2. Реализовать возможность создания новой заметки из приложения и сохранения ее в gist в виде JSON в файл "ios-course-notes-db".
3. Реализовать возможность редактирования заметки в приложении (после редактирования необходимо отправлять запрос для обновления файла "ios-course-notes-db" в gist).
4. Реализовать возможность удаления заметки из приложения (после удаления заметки необходимо отправлять запрос для обновления файла "ios-course-notes-db" в gist).
5. Код необходимо помещать в операции, созданные в предыдущем уроке.

6. Все используемые зависимости должны содержаться в архиве проекта. Проверяющий не будет вызывать `pod install`.

Усложнение
Реализуйте авторизацию в приложении по логину и паролю. За это можно получить дополнительные баллы.

Подсказки
Редактирование и удаление заметки требуется реализовать как обновление уже существующего файла в gist.
Информация о файлах gist в Codable структуре может быть представлена так:

```
struct Gist: Codable {
    let files: [String: GistFile]
    ...
}

struct GistFile: Codable {
    let filename: String
    let rawUrl: String

    enum CodingKeys: String, CodingKey { // Позволяет использовать названия полей в структуре отличающиеся от названий ключей в JSON
        case filename
        case rawUrl = "raw_url"
    }
}
```

Получить массив заметок, загруженных по rawUrl из gist (через URLSession dataTask), можно так:

```
try? JSONDecoder().decode(Notebook.self, from: data)
```

Критерии проверки
Отправьте проект на проверку. Сокурсники будут проверять его на наборе тестов: создадут, отредактируют, удалят заметки, проверят, что список заметок в приложении синхронизируется со списком gist. Также за **реализацию авторизации** в приложении (по логину и паролю) можно получить **дополнительные баллы**.

Задание 8. Приложение для спортсменов

Создайте приложение для спортсменов. Оно позволят сохранять время, за которое спортсмен пробегает круг.
В приложении должен быть один экран, где будет отображаться список с результатами и кнопка «+».

Требования

- С интерефейсом и работой приложения вы можете ознакомиться на видео (.mov, 2.1 Мб).
- Результаты кругов между перезапусками приложения сохраняются.
- Для построения стэка Core Data используйте NSPersistentContainer.
- Для отображения списка результатов используйте NSFetchedResultsController.
- Сохранение времени круга должно происходить в отдельном background context'е.
- Контексты должны передаваться от места инициализации стэка Core Data объектам, где они используются, а не лежать в AppDelegate или любом другом синглтоне.

Критерии проверки

Отправьте проект на проверку. Проверяющие запустят ваше приложение в симуляторе, проведут ряд функциональных тестов, изучат ваш код на предмет соблюдения всех требований задачи.

Задание 9. Приложение Заметки. Локальное хранение в Core Data

Реализуйте локальное хранение заметок в Core Data.

Требования

- Использовать NSPersistentContainer для построения стэка Core Data.
- Использовать `LoadNotesDBOperation`, `SaveNoteDBOperation`, `RemoveNoteDBOperation` из «Модуля 1. Многопоточность».

- Не допускается использование NSManagedObject объектов за пределами операций, на выходе из операций должны быть Note.
- Чтение, сохранение и удаление заметок должно происходить в background context'ax.
- Контексты должны передаваться от места инициализации стэка Core Data объектам, где они используются, а не лежать в AppDelegate или любом другом синглтоне.
- Должно быть реализовано несколько версий модели данных и присутствовать миграция.
- Допускается использование lightweight миграции.

Критерии проверки
Отправьте проект на проверку. Проверяющие запустят ваше приложение в симуляторе и проведут ряд тестов. Тесты проверят работоспособность всех элементов приложения. Если используется несколько версий схем данных, укажите их порядок в текстовом поле.

Задание 10. Классическая iOS архитектура

Найдите в вашем приложении Заметки проблемные места, описанные в этом уроке. Попробуйте применить для их устранения приёмы, описанные в этом уроке. Попробуйте использовать все описанные приёмы, чтобы разобраться в каких случаях эти приемы наиболее оправданы.

Задание 11. MVP

Модифицируйте свое приложение Заметки в архитектуру MVP: весь, не касающийся графики, код вью-контроллеров абстрагируйте в презентеры. Создайте протоколы для вью и презентеров, проверьте схему владения объектами, удостоверьтесь в пассивности вью в новой реализации вашего приложения.

Задание 12. MVVM

- Найдите и изучите код одного или двух приложений, использующих MVVM и реактивные библиотеки.
- Подумайте: каким был бы код вашего приложения при использовании MVVM?
- Реализуйте один экран вашего приложения с помощью MVVM.
- Реализуйте байндинги между полями ввода данных и моделями. Сравните этот подход с MVP.

Задание 13. Координаторы

- Создайте иерархию координаторов в вашем приложении Заметки. За основу возьмите код вашего приложения в архитектуре MVP. Выделите соответствующую логику в координаторы.
- Попробуйте реализовать всю навигацию между экранами в коде координаторов, а не в storyboard. Проследите, чтобы вью-контроллеры и презентеры не были связаны друг с другом в вашем коде, а общались лишь через координаторы.

Задание 14. SOLID

Изучите код вашего приложения и выпишете те принципы, которые в вашем коде нарушаются. Дополните список пояснениями на примерах из вашего кода: что именно и почему является причиной нарушения SOLID. Подумайте, какие корректировки можно было бы внести в код вашего проекта, чтобы эти принципы не нарушались.

Задание 15. VIPER

- Ознакомьтесь с рекомендованными статьями про VIPER и примерами реализаций.
- Взяв за основу модификацию вашего приложения в архитектуре MVP, абстрагируйте из презентеров код для

роутеров и интеракторов. Создайте соответствующие протоколы для новых классов. Обратите внимание на четкое разделение ответственности между этими классами. Проверьте направления зависимостей, а также убедитесь в отсутствии retain-циклов (корректно расставьте сильные и слабые ссылки).