OTUS

iOS Developer. Professional

Вся мощь Swift 5.x для развития профессиональных навыков уровня Middle/Senior iOS Developer

Длительность курса: 128 академических часов

1 SwiftUI и основы Combine

1 Проектируем UI декларативно с SwiftUI. В чем отличия UIKit и SwiftUI

Цели занятия:

после занятия студент сможет:
настроить окружение для работы на курсе и выполнения домашних заданий;
использовать Xcode;
создавать базовые интерфейсы на SwiftUI/Combine.

Краткое содержание:

возможности и ограничения SwiftUI;
property Wrappers: @State, @Binding, @ObservedObject и ObservableObject, @EnvironmentObject, @Environment;
основы Combine: @Published;
решение организационных вопросов;
настройка environment: Xcode, git, scripts.

Домашние задания

1 Создание каркаса приложения на SwiftUI

Цель: Студент
1. Будет целостно понимать навигационный стек SwiftUI/Combine
2. Получит умение сборки иерархии экранов на SwiftUI/Combine

Создать флоу экранов на SwiftUI
1. Добавить TabView

2. На втором табе сделать List с обернутый в NavigationView

2.1 Из листа должны быть переходы с NavigationLink

3. На третьем табе должна быть кнопка открывающая модальное окно
4. На первом табе должна быть кнопка открывающая второй таб и один из пунктов там
5. Протестировать на iPad/iPhone симуляторах,

		со сменой ориентации девайса 6. Добавить один UIKit компонент через UIViewRepresantable
2	**SwiftUI List, List с кастомным лейаутом, Hosting ViewControllers**	**Цели занятия:** после занятия студент сможет: понимать, в каком состоянии приложение; добавлять логику на изменение состояния приложения; будет знать как устроен UIKit и его иерархия классов; делать навигацию в SwiftUI разными способами. **Краткое содержание:** Life cycle приложения; UIResponder; SceneDelegate; Hosting ViewControllers и зачем они; Combine.
3	**Использование NavigationView, TabView. Создание собственного стека Навигации**	**Цели занятия:** после занятия студент сможет: использовать NavigationView и TabView в SwiftUI; создавать собственный стек Навигации; использовать Transitions. **Краткое содержание:** особенности навигации в SwiftUI; способы решения проблем с навигацией.

4 **Создание кастомных Shape, SwiftUI Drawing and Animation API**

Цели занятия:

после занятия студент сможет:
работать с CALayer и понимать систему координат, используемую в CoreGraphics;
программно рисовать Shapes в SwiftUI;
использовать анимацию в SwiftUI.

Краткое содержание:

разница CoreGraphics и Drawing API SwiftUI;
создание кастомных фигур на Drawing API SwiftUI;
анимация интерфейса в SwiftUI.

Домашние задания

1 Реализация пейджинга на реальном API

Цель: создать список с пейджингом, работающий на реальном серверном API

1. Используйте открытое API https://github.com/public-apis/public-apis
2. Сделайте несколько рубрик по разным запросам новостей или городов по погоде (переключение через горизонтальный ScrollView либо SegmentedControl)
3. При переключении рубрик должен изменять содержимое List, пейджинг должен работать
4. Сделать глубину в 3 экрана с помощью кастомного навигейшен-стека
5. При выборе ячейки на каждом экране(разделе SegmentedControl) сделать анимацию улетающей ячейки, например, по кривой вниз (типа анимация сохранения в избранное)

5 **Отображение структурированных данных, List, пейджинг, кастомные компоненты на UIViewRepresentable**

Цели занятия:

после занятия студент сможет:
– реализовывать пейджинг на SwiftUI;
– кодогенерить Network слой в ДЗ;
– привязывать List к реальному API с помощью кодогенерации.

Краткое содержание:

использование List в SwiftUI;
реализация пейджинга для List.

2 Современная архитектура мобильных приложений

1 **Необычная система типов Swift, структуры данных, Generics**

Цели занятия:

после занятия студент сможет:
создать кастомные структуры данных.

Краткое содержание:

теория типов и Compound и Named типы;
метатип и вложенные типы;
protocol Composition;
generics.

---

2 **Связывание разных частей приложения Observing, Signals, Callbacks. PATs (Protocol with Associated Types)**

Цели занятия:

после занятия вы сможете:
работать с разными подходами связи сущностей в приложении: коллбеками , сигналами , классическим Observing.

Краткое содержание:

observing и broadcasting, нужен ли нам KVO;
плюсы и минусы Delegation, виды делегатов;
callbacks;
sgnals ans Slots и причем здесь Rx;
PATs (Protocol with Associated Types).

---

3 **Protocol Oriented Programming (POP), SOA, модуляризация, архитектурные Rx паттерны: Flux/Redux**

Цели занятия:

после занятия вы сможете:
применять паттерн Inversion of Control;
использовать ServiceLocator;
объяснить в чем польза слоистой архитектуры.

Краткое содержание:

inversion of Control паттерн;
serviceLocator и инжектинг;
разделение архитектуры на слои и что это дает;
модуляризация приложения, способы: Frameworks, SwiftPM;
protocol Oriented Programming (POP);
type constraints.

Домашние задания

1 Реализация Service Locator и Dependency Injection и изолированности слоев в приложении

Цель: Научиться внедрять DI, понять плюсы подхода.

1. Создать ServiceLocator (можно на дженериках)
2. Перевести существующие сервисы на него
3. Добавить инжектинг в переменные инстанса класса, чтобы в каждом классе можно было видеть зависимости, не скролля файл
4. Выделить уровень Core сервисов (сеть, парсинг, хранение).
*5. Вынести Core в пакет Swift PM

---

4 **MVx, VIP, анализ архитектурных паттернов. SOLID, GRASP, Coupling/Cohesion принципы.**

Цели занятия:

после занятия вы сможете:
разобраться в семействе MV(x) паттернов;
объяснить, что такое модуляризация и как ей пользоваться;
использовать Clean Architecture подход;
рассказать о паттернах в мобильной разработке.

Краткое содержание:

MV(X) архитектурные паттерны;
архитектурные Rx паттерны
SOLID и как получить от него пользу;
другие принципы: KISS, DRY/DIE, YAGNI, BDUF, SOC;
необходимые паттерны для сегодняшней мобильной разработки: Adapter, Memento, Observer, Strategy, Factory, Command, Composite, Iterator, Mediator, Proxy, Template Method, Singleton и где они приненимы на iOS.

Домашние задания

1 Создание модуляризированного приложения с SOA/MVVM/Flux/MobX

Цель: Прокачать умение делать архитектурный рефакторинг всего приложения.
Научиться внедрять Архитектурные паттерны, понять плюсы подхода

Можно использовать в качестве основы реализованное домашнее задание по прошлому занятию.

1. Выполнить рефакторинг приложения до SOA/MVVM/Flux/MobX. При реализации SwiftUI использовать стандартные механизмы Combine/Signals/EventBus
2. Модуляризировать свое приложение одним из известным способом. Вынести UI компоненты в отдельный модуль и импортить его в местах использования
3. Создать ServiceLocator(SOA)/MVVM(вложенные вьюмодели)/Flux(вложенные сторы)/MobX(контейнер-медиатор)
4 Перевести существующие сервисы на него
5. Добавить инжектинг в переменные инстанса класса, чтобы в каждом классе можно было видеть зависимости, не скролля файл
* 6. Выделить уровень Core сервисов (сеть, парсинг, хранение).
* 7. Core и другие сервисы вынести в модуль

3 Foundation без сторонних фреймвоков и Swift 5 Standard Library

1 **Sequences и коллекции, асимптотический анализ: O(1), O(N), O(N*log(N)), O(n^2)**

Цели занятия:

после занятия вы сможете:
использовать Sequence и Collection для реализации собственных структур данных;
объяснить как работают lazy collection, будете понимать концепцию type-erased.

Краткое содержание:

sequence и IteratorProtocol;
type-erased типы: AnySequence, AnyIterator, AnyCollection;
lazy Wrappers;
wrappers for Algorithms;
асимптотический анализ встроенных и кастомных структур данных.

---

2 **Использование всей мощи String: StringInterpolation, Expressible. Региональные форматы.**

Цели занятия:

после занятия студент сможет:
работать с utf8 и utf16 представлениями;
использовать подстроки и Ranges, StringProtocol;
работать с единицами измерения и валютами.

Краткое содержание:

Custom String Interpolation;
сравнение суффиксов и префиксов и другие способы сравнения строк;
regex.
парсинг и представление телефонных номеров;
форматирование дат согласно региону и локали, POSIX спецификация;
корректная локализация приложения на несколько языков и регионов.

---

3 **Ассоциативные типы, Type Erasure, «сахарные» типы данных, диспетчеризация вызовов в Swift 5**

Цели занятия:

после занятия вы сможете:
объяснить, что такое Method Dispatch и почему это важно.
создавать generic протоколы.
объяснить как Swift работает с типами.

Краткое содержание:

3 типа диспатчеризации в Swift: direct, dynamic, message;
Associated Types;
PATs и динамическая диспатчеризация;
другие способы реализации паттерна Type Erasure;
как работают типы в SIL (Swift Intermediate Language).

---

4 **Компилятор LLVM, AST, создание собственных операторов**

Цели занятия:

после занятия вы сможете:
объяснить, как работает компилятор LLVM;
создавать собственные операторы.

Краткое содержание:

как работает LLVM: SIL, IR;
как некоторые типы представлены в SIL и для чего это нужно знать;
особенности и хитрости компиляции;
перезагрузка и создание операторов.

Домашние задания

1 Создание расширения для копирования текста в приложения и построение на основе него суффиксного массива

Цель: Получить умение создавать App Extension. Вы научитесь создавать кастомные структуры данных на основе протоколов Sequence и IteratorProtocol кастомные структуры данных и решать с помощью них реальные задачи в

приложениях

1. Реализовать для приложения WidgetExtension
1.1 В виджете сделать несколько кнопок ведущих на разные части приложения(или табы), одна из на экран:
1.2 Внутри приложения поле ввода для текста
1.3 Показывать в приложении(пункт 3.1) на виджете статистику совпадения суффиксов более 3х символов, например:
"
бра – 2
кад.– 3
"
2. Перед показом View разложить все полученные слова в тексте на SuffixSequence
2.1 Как показано в уроке создать SuffixIterator
2.2 Обернуть в SuffixSequence каждое слово из полученное из шаринга
3. В этом View:
3.1 Отображать Segmented Control(Picker в SwiftUI) переключения между
3.1.1 листом всех суффиксов, повторяющиеся помечать кол-вом, остортировать по алфавиту, сделать переключение сортировки ASC/DESC
3.1.2 топом 10 самых популярных 3х буквенных суффиксов, отсортированных по кол-ву нахождений
*4. Добавить поиск на лист 2.1.1
4.1 Искать по совпадением используя debounce в 500мс из Combine

4 Многозадачность

1 **Проблемы многозадачности и способы их решения, GCD**

Цели занятия:

после занятия студент сможет:
использовать GCD: QoS, Queues, Main Queue и Main Thread.

Краткое содержание:

антипаттерны и проблемы: Priority Inversion, Race condition, Deadlock, Resource contention, Starvation, Non-deterministic and Fairness.

---

2 **Внутренности GCD(libdispatch), OperationQueue**

Цели занятия:

после занятия вы сможете:
рассказать о проблемах многозадачности;
избегать антипаттерны;
пользоваться средствами GCD.

Краткое содержание:

плюсы и минусы OperationQueue;
внутренности libdispatch: пул тредов, continuation, QoS и как зная это лучше использовать очереди.

---

3 **RunLoop & POSIX Threads, Инструменты синхронизации, Lock, Mutex**

Цели занятия:

после занятия вы сможете:
разобраться как работает RunLoop;
использовать инструменты синхронизации;
разобраться с POSIX.

Краткое содержание:

runLoop и чем сегодня он может нам быть полезен;
pthreads;
виды локов: NSLock, NSRecursiveLock, Spinlock, Mutex, Semaphore;
dispatch Barriers;
trampoline техника.

Домашние задания

1 Реализация асинхронного выполнения задач и оценка эффективности подхода

Цель: Научиться внедрять сервис очереди в существующую инфраструктуру приложения, развиваем навык рефакторинга для не-UI кода приложения

1. Сделать таб историю шарингов на основе предыдущего таба
2. Реализовать структуру данных Job Queue
3. Создать сервис Job Scheduler
4. В хедере таблицы экрана Feed сделать возможность запускать один конкретный тест по всем айтемам истории на построение суффиксного массива
5. В ячейку выводить время построение
*6. Красить в зеленый лучшее время и в красный худшее, или градацие от зеленого к красному

5 Networking и хранение данных

1 **Новый Network-фреймвок, URLSession, Codable**

Цели занятия:

после занятия вы сможете:
попробовать разные способы формированияи работы с URL;
объяснить, как работать с чистой URLSession;
познакомиться с GraphQL и как его использовать на iOS.

Краткое содержание:

network фреймвок, HTTP, REST, Sockets, GraphQL;
URLSession;
сериализация и десериализация с помощью Codable;
немного о gRPC и кодогенерации.

---

2 **Socket.io, WebSocket и другие сокеты для чатов и мгновенных обновлений**

Цели занятия:

после занятия начнем разбираться в socket.io, WebSocket и других сокетах для чатов и мгновенных обновлений.

Краткое содержание:

мессенджеры;
пуши;
практика вместе с преподователем.

3	**SQLite, способы кеширования, Files, Сравнение CoreData и Realm**	**Цели занятия:** после занятия вы сможете: разобрать строение файловой системы iOS; познакомиться с некоторыми способами кэширования; рассмотреть такие решения для сохранения данных, как SQLite, Realm, NoSQL, CoreData **Краткое содержание:** виды кеширования; SQLite и другие DB* альтернативы; NoSQL; Files и File System.

4 **Безопасность: OAuth 2.0, Keychain и обфускация API ключей, SSL Pinning**

Цели занятия:

разберем как работает OAuth;
сохраним важные данные в Keychain;
потренируемся скрывать данные от декомпиляции ipa файла;
защищать приложение от Man-In-The-Middle аттаки.

Краткое содержание:

методы обфускации;
Keychain;
OAuth 1.0 и 2.0;
SSL Pinning.

Домашние задания

1 Реализация поддержки оффлайн режима в приложении

Цель: Научится сохранять Codable структуры в файлы, реализовывать кэш

Создать приложение, которое будет получать данные из сети.
1. Реализовать кэширование на Realm/Files/Firebase/CoreData (лучше то, что еще не использовали)
2. Проверить, что модели поддерживают протокол Codable в вашем каркасе приложения
2.1 Либо реализовать кеш другим способом (UserDefaults, NoSQL(CoachDBLite, PinCache, ...), Keychain :), Files, Core Data, Realm)
3. Реализовать сохранение в файл при возвращении из экрана тестирования структуры данных
4. Чтение сделать в момент запуска приложения или перезахода в экран тестирования структуры данных
5. Должно сохраняться предыдущее состояние тестирования даже при перезапуске приложения

6 Создание приложений для Apple Watch, TV, Mac

1 **watchOS**

Цели занятия:

после занятия студент сможет:
создать приложение для Apple Watch.

Краткое содержание:

отличия SwiftUI для watchOS;
интеграция с iOS приложением.

---

2 **tvOS**

Цели занятия:

после занятия студент сможет:
создать приложение для Apple TV.

Краткое содержание:

отличия SwiftUI для tvOS.

---

3 **Кросс-платформенный код для iOS/iPadOS, watchOS, macOS, tvOS**

Цели занятия:

после занятия вы сможете:
использовать NavigationView и TabView в SwiftUI;
делать компоненты с помощью @ViewBuilder;
использовать Transitions и Animation;
создавать собственный стек Навигации.

Краткое содержание:

нюансы SwiftUI для iPadOS/macOS;
перенос приложения на macOS.

Домашние задания

1 Приложение работающее на всех платформах Apple

Цель: Прокачать навык apple-кроссплатформенной разработки

1. Можно использовать шаблон https://github.com/shial4/SpriteKit-SwiftUI-Template или сделать свой для SwiftUI компонентов
2. Сделать меню игры с разделами
2.1 Play
2.2 Settings
2.3 Leaderboard
3. На каждом экране показывать как минимум заголовок и тестовые данные
4. Можно сделать какой-нибудь геймплей :)
*4. Добавить watchOS

7 CoreML и Vision, нейронные сети и машинное обучение

1	**CoreML, CreateML, TensorFlow использование обученных моделей нейронных сетей**	**Цели занятия:** разобраться как устроен CoreML 3+; использовать готовые обученные модели. **Краткое содержание:** CoreML разных версий отличия от нейронок ; Практическая демонстрация возможностей.
2	**CoreML: получение моделей с помощью AutoML Vision и использование их на устройстве**	**Цели занятия:** объяснить как создавать и обучать модели на AutoML и Vision. **Краткое содержание:** autoML; vision; практическое демо получения и использования модели.

8 Мультиплатформенная разработка: перенос на Android, Vulkan/Metal

1 **Мультиплатформа для Rich Media: Metal и Vulkan, разработки игр, Video/Image процессинг**

Цели занятия:

после занятия вы сможете:
писать кроссплатформенный GPU код;
настраивать окружение для этого.

Краткое содержание:

GPU конвейер;
Metal;
OpenGL;
MoltenVK.

---

2 **Jetpack Compose**

Цели занятия:

после занятия вы сможете:
сравнить SwiftUI и Compose;
переносить логику разработки на SwiftUI в Compose.

Краткое содержание:

Android Studio & IntelliJ IDEA;
обзор Kotlin Multiplatform и целесообразность общего ядра;
разработка Pure Swift + Pure Kotlin.

---

3 **Одновременная реализация фич на iOS + Android. Необходимый tool-set**

Цели занятия:

после занятия вы сможете:
настраивать окружение;
собирать KMM стек для iOS;
решать проблемы с gradle.

Краткое содержание:

как Swift разработчику писать на Kotlin;
SwiftUI + Jetpack Compose;
Combine + RxAndroid/RxKotlin/RxJava.

Домашние задания

1 Создание мультиплатформенного сетевого слоя с помощью KMM и openapi-generator

Цель: Научится собирать мультиплатформу с кодогенерацией

1. Установить IDEA Community Edition https://www.jetbrains.com/idea/download
2. Создать мультиплатформенный проект, как было показано на занятии
3. Пересоздать для SwiftUI iosApp, как было показано на занятии
4. Обновить openapi-generator с помощью brew install openapi-generator
5. Можно взять из материалов recipepuppy_openapi.yaml (нужно в Info.plist разрешить http) или найти другое публично апи и написать спеку https://github.com/public-apis/public-apis
6. Сгенерировать сетевой слой openapi-generator generate -g kotlin -i recipepuppy_openapi.yaml --library multiplatform -o NetworkLayer (показывается на занятии)
7. Подключить в build.gradle io.ktor и kotlinx.serialization
8. Сбилдить KMM в IDEA и затем app фреймвок в Xcode
9. Вывести рецепты в SwiftUI лист по нескольким ингредиентам или поиском (частично показано в конце занятия)

9 Организация разработки

1 **Тестирование кода XCTest, UITest, fastlane и CI**

Цели занятия:

после занятия студент сможет:
собрать CI (Continuous Integration) на fastlane;
использовать XCTest.

Краткое содержание:

test-Driven Development (TDD);
теория тестирования;
зачем нужен UITest.

---

2 **Git-flow, TBD, автоматизизация workflow**

Цели занятия:

после занятия студент сможет:
использовать комманд git-flow.

Краткое содержание:

продвинутое использование git;
фича, хотфикс, релизный цикл, master ветка;
trunk Based Development и ветки для вич.

---

3 **Как правильно написать резюме и развивать hard-skills**

Цели занятия:

после занятия вы сможете:
корректно писать резюме;
выбирать работодателя, чтобы развивать свои hard-skills.

Краткое содержание:

какие бывают работодатели;
какие скрытые критерии отбора используются.

Домашние задания

1 Написание работающего резюме

Цель: Создать/доработать резюме, чтобы получать больше откликов и оно меньше вызывало вопросов со стороны рекрутеров и работодателей

1. Посмотреть запись занятия с разбором резюме
2. Доработать свое
3. Прислать ссылкой или файлом

10 Проектная работа

1 **Написание приложения с нуля**

Цели занятия:

выбрать и обсудить тему проектной работы;
спланировать работу над проектом;
ознакомиться с регламентом работы над проектом;
генерировать идеи для простых приложений на основе известных «болей» пользователей;
использовать iOS платформу для генерации идей для приложений.

Краткое содержание:

подбор инструментов, помощь с стартом написания приложения;
правила работы над проектом и специфика проведения итоговой защиты;
требования к результату проекта и итоговой документации.

Домашние задания

1 Написание приложения с нуля

Цель: Создание проектной работы, которая будет указана в сертификате

выбрать тему/название для приложения;
закрепить тему приложения в чат с преподавателем; обсудить тонкости;
сделать проектную работу

2 **Консультация по проектам и домашним заданиям**

Цели занятия:

получить ответы на вопросы по проекту, ДЗ и по курсу.

Краткое содержание:

вопросы по улучшению и оптимизации работы над проектом;
затруднения при выполнении ДЗ;
вопросы по программе.

---

3 **Защита проектных работ**

Цели занятия:

защитить проект и получить рекомендации экспертов.

Краткое содержание:

презентация проектов перед комиссией;
вопросы и комментарии по проектам.

Домашние задания

1 Сдать ссылку на репозиторий курсового проекта. В репозитории обязательно должен быть заполнен файл Readme.md с описание проекта.

Цель: Предложить оговоренную/одобренную ранее тему через строку в окне ниже.
Сдать ссылку на репозиторий курсового проекта. В репозитории обязательно должен быть заполнен файл Readme.md с описание проекта.