МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА IOS ПРИЛОЖЕНИЙ НА ЯЗЫКЕ SWIFT

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Преображенской Яны Алексеевны

Научный руководитель
к. т. н., доцент _____ В. М. Соловьев

Заведующий кафедрой
к. ф.-м. н., доцент _____ А. С. Иванов

Саратов 2020

ВВЕДЕНИЕ

В настоящее время сложно представить человека, у которого нет мобильного телефона. Почти все современные люди их имеют и активно пользуются. Поэтому разработка мобильных приложений крайне важна. Мобильные приложения намного удобнее веб-страниц, потому что они сразу разрабатываются специально для мобильных устройств и имеют удобный дизайн, кроме того они содержат всю необходимую информацию по теме приложения, из-за чего пользователю не нужно тратить время на поиск этой информации в сети интернет.

Помимо мобильных телефонов люди часто имеют домашних питомцев, таких как кошки. Но заводя пушистого друга далеко не все задумываются о правильном уходе и поддержании его здоровья. Поэтому крайне важно дать людям возможно узнать о них побольше. С этой целью планируется разработка приложения «Happy Cats», которое позволит хозяевам кошек больше узнать о породах кошек и заболеваниях, которым подвержена конкретная порода и о всех кошачьих заболеваниях в целом. Специально для данного приложения будет разработана экспертная система, которая позволит пользователям по проявившемся у их питомцев симптомам, узнать наиболее вероятное заболевание. Планируется, что данное приложение поможет людям лучше узнать своих питомцев и стать более ответственными за их жизнь.

Данное приложение будет разработано для пользователей продукцией Apple, а именно для пользователей смартфонов iPhone под управлением операционной системы iOS. Для разработки приложения будет использован язык программирования Swift 5-ой версии и среда разработки Xcode.

Основной целью выпускной квалификационной работы является:

— Разработка клиент-серверного приложения для мобильной операционной системы iOS.

В рамках основной цели рассматриваются следующие задачи:

— Изучение основ проектирования iOS приложений на языке Swift и необходимых для этого инструментов;
— Изучение основных архитектурных паттернов iOS приложений;
— Изучение фреймворков, позволяющих реализовать возможности реактивного программирования.

Выпускная квалификационная работа состоит из введения, 2 разделов,

заключения, списка использованных источников и 2 приложений. Общий объем работы — 84 страницы, из них 65 страниц — основное содержание, включая 29 рисунков, цифровой носитель в качестве приложения, список использованных источников информации — 28 наименований.

1 Краткое содержание работы

Первый раздел «Основы разработки iOS приложений на языке Swift» посвящен основной теории по разработке приложений под управлением мобильной операционной системы iOS на языке программирования Swift. Данный раздел состоит из 5 подразделов.

Первый подраздел «Язык программирования Swift» посвящен обзору языка программирования Swift.

В 2014 году 2 июня компания Apple представила новый язык программирования с названием Swift. В 2019 году 20 сентября была представлена 5-ая версия языка Swift, которая остается актуальной на данный момент [1].

Swift — открытый мультипарадигмальный компилируемый язык программирования общего назначения. Язык swift предназначен для разработки под iOS и macOS, оптимизирован для использования фреймворков серии Cocoa, обладает совместимостью с основными кодами Apple. Swift проектировался как более читаемый и надежный с точки зрения реакции на ошибки программиста язык, чем его предшественник Objective-C [2].

На презентации своего нового языка программирования компания Apple рассказала, что Swift позаимствовал многое из Objective-C, однако часть функций языка выполняется намного быстрее по сравнению с самим Objective-C и другими языками программирования. Например, по скорости выполнения сортировок сложных объектов он в 4 раза превосходит Python, и 1,5 — Objective-C [3]. Код на Swift совместим с кодами на C и Objective-C [4].

Второй подраздел «Среда разработки Xcode» посвящен обзору среды разработки Xcode, которая использовалась для написания приложения.

Xcode — интегрированная среда разработки (IDE) программного обеспечения для платформ macOS, iOS, watchOS и tvOS, разработанная корпорацией Apple. Первая версия IDE была выпущена в 2003 году.

Последней актуальной версией является Xcode 11. Xcode 11 включает SDK для iOS 13, macOS Catalina 10.15, watchOS 6 и tvOS 13. Xcode 11 поддерживает разработку для устройств под управлением iOS 13.1. Xcode 11 поддерживает отладку на устройстве для iOS 8 и новее, tvOS 9 и новее, а также watchOS 2 и новее. Xcode 11 требует Mac под управлением MacOS Mojave 10.14.4 или новее [5].

Третий подраздел «Экспертная система» посвящен обзору экспертных

систем и возможности реализации их интерфейсов в мобильном приложении.

Экспертная система — программный комплекс, аккумулирующий знания специалистов в конкретных предметных областях и тиражирующие их эмпирический опыт для консультации менее квалифицированных пользователей.

Экспертные системы используются вместе с базами данных для обеспечения распознавания по тем же принципам, по которым это делает человек, и вместе с автоматизированными системами принятия решений для выявления знаний с помощью анализа скрытых закономерностей в данных и создания таким образом интеллектуальной базы знаний.

Также говорят, что экспертная система — это компьютерная система, которая эмулирует способности экспертна к принятию решений. Термин «эмулирует» означает, что система обязана действовать как эксперт. Конечно еще не удалось создать систему, которая могла бы принимать в качестве универсального решателя задач, но существующие сейчас экспертные системы весьма успешно действуют в своих областях [6].

Для демонстрации возможностей экспертной системы, в приложении, которое разрабатывается как практическая часть данной работы, был создан раздел «Экспертная система». Данная экспертная система позволяет по симптомам, проявившимся у кошки, выявить наиболее вероятное заболевание.

Для удобного взаимодействия с экспертной системой интерфейс должен содержать элементы, которые будут интуитивно понятны любому пользователю. Для решения данной задачи было принято решение сделать отдельные экраны для каждого из задаваемых вопросов, которые будут последовательно сменять друг друга. На экране вопроса содержится сам текст вопроса и 2 кнопки для выбора ответа. Для предоставления ответа были выбраны именно кнопки, а не текстовые поля, потому что таким образом пользователю будет необходимо только нажать на соответствующую кнопку из-за чего исключается возможность ошибки при вводе ответа. К тому же, экспертная система предусматривает только ответы «да» и «нет», поэтому нет смысла давать пользователю возможность отвечать как-то иначе.

Четвертый подраздел «Основные архитектурные паттерны iOS приложений» посвящен обзору архитектуры приложений в целом и наиболее часто используемых в iOS разработке паттернов проектирования.

Архитектура приложения — это архитектурный шаблон проектирования, охватывающий все приложение или какую-то его часть, которую обычно называют модулем. Из этих модулей обычно и строится приложение. В качестве модуля могут быть как отдельный экран приложения, так и несколько связанных между собой экранов [7].

Использование определенной архитектуры при разработке приложений крайне важно, так как при таком подходе приложение становится легче и поддерживать и уменьшается время для привлечения новых разработчиков. Хорошо спроектированная программа обладает достаточной гибкостью и расширяемостью при изменении и эволюции.

В iOS разработке существует 4 основных архитектурных паттерна: MVC, MVP, MVVM и VIPER. Сразу бросается в глаза, что MVC, MVP и MVVM очень похожи. Все эти 3 паттерна разделают сущности приложения на 3 типа:

— Models (M) — отвечает за данные или за слой доступа к данным, который ими манипулирует.

— View (V) — отвечает за уровень представления, для окружающей среды iOS это все, что начинается с префикса UI.

— Controller(C) / Presenter (P) / ViewModel (VM) — посредник между Model и View. В основном отвечает за изменения Model, реагируя на действия пользователя, выполненные на View, и обновляет View, используя изменения из Model [8].

В MVC Controller является посредником между View и Model. В итоге получается, что View и Model не знают друг о друге. Из-за этого становится трудно заново переиспользовать Controller, но в принципе это нормально, так как мы должны иметь где-то бизнес-логику, которой не место в Model.

MVP очень похож на MVC. Однако, в этом варианте View пассивна, а посредником является Presenter, который не имеет отношения к жизненному циклу View, в Presenter совершенно нет layout-кода, но он все так же отвечает за обновление View в соответствии с новыми данными и состоянием.

Кроме стандартного MVP существует также вариант MVP с «надзирающим» контроллером. Такой вариант MVP подразумевает прямое связывание View и Model, при этом Presenter все так же обрабатывает действия с View и способен ее изменять.

MVVM является самой новой из архитектурных паттернов типа MV(X).

MVVM очень похож на MVP, но здесь в качестве посредника между слоями View и Model выступает ViewModel. Так же как и в MVP View и Model ничего друг о друге не знают и все взаимодействие происходит через ViewModel, а `UIViewController` рассматривается как View. Однако, MVVM делает связывание между объектами View и ViewModel, а не между View и Model, как это делал бы MVP.

Архитектура MVVM отлично работает в связке с таким фреймворком, как ReactiveCocoa. Благодаря ReactiveCocoa можно реализовать возможности реактивного программирования и с легкостью связать элементы View с ViewModel. Связка MVVM и ReactiveCocoa — наилучший выбор, если мы хотим писать приложение с использованием реактивного программирования.

Архитектура VIPER в отличии от всех предыдущих архитектур типа MV(X) разделяет обязанности на 5 слоев, а не на уже привычные 3.

Описание слоев:

— View — уже привычная View, содержащая все настройки по кастомизации экрана;

— Interactor содержит в себе бизнес-логику, связанную с данными (Entities): например, создание новых экземпляров сущностей или получение их с сервера. Для этих целей будут использовать некоторые сервисы и менеджеры, которые рассматриваются скорее как внешние зависимости, а не как часть модуля VIPER;

— Presenter содержит бизнес-логику, связанную с UI, вызывает методы в Interactor;

— Entities — простые объекты данных;

— Router несет ответственность за переходы между VIPER-модулями.

Пятый подраздел «Реактивное программирование» посвящен обзору подхода реактивного программирования и библиотек, которые используются для реализации данного подхода при написании iOS приложения.

Реактивное программирование — это парадигма программирования с асинхронными потоками данных.

Асинхронность в программировании — это выполнение процесса в неблокирующем режиме, что позволяет программе продолжить обработку.

Потоки — это массив данных, отсортированных по времени, который может сообщать, что данные изменились. Потоки могут транслировать и под-

писываться на данные. В течение жизненного цикла потоки могут транслировать три сигнала: данные, ошибку и завершение [9].

Реактивный подход стал особенно важным для современных веб- и мобильных приложений, в которых происходит большое количество UI-событий.

RxSwift и RxCocoa [10] являются фреймворками для реализации возможностей реактивного программирования. RxSwift — это фреймворк для взаимодействия с языком программирования Swift, а RxCocoa — это фреймворк, упрощающий использование API-интерфейсов Cocoa в iOS и OS X с помощью реактивных методов [11].

RxFlow — навигационный фреймворк для iOS приложений, основанный на паттерне «Координатор».

Таким образом, в первом разделе была кратко рассмотрена теоретическая информация, необходимая для разработки iOS приложений на языке Swift.

Второй раздел «Разработка iOS приложения» посвящен процессу разработки мобильного приложения для устройств под управлением мобильной операционной системы iOS. Данный раздел состоит из 8 подразделов.

Первый подраздел «Описание» посвящен общему описанию разрабатываемого приложения.

В ходе практической части было разработано приложение под названием «Happy Cats». Данное приложение позволяет пользователям узнавать актуальную информацию про породы кошек, про заболевания которым они подвержены и про все заболевания кошек в целом. Также пользователь может воспользоваться экспертной системой и, ответив на вопросы, узнать по симптомам наиболее вероятное заболевание, которое в данный момент беспокоит его питомца. Кроме этого приложение содержит раздел с новостями, где пользователи могут прочитать последние интересные новости из мира кошек. Для просмотра всей информации пользователю необходимо авторизоваться в приложении. В своем личном кабинете пользователь может добавлять как информацию о себе, так и о своих питомцах. Данное приложение направлено на то, чтобы сделать хозяев домашних кошек более осведомленными о заболеваниях, которым могут быть подвержены их питомцы и как следствие более ответственными.

Данное приложение разрабатывалось для мобильных устройств под

управлением операционной системы iOS 12 и выше. Приложение было написано на языке Swift с использованием архитектуры MVVM и фреймворков RxSwift, RxCocoa и RxFlow. Приложение «Happy Cats» поддерживает русский и английский языки.

Второй подраздел «Локализация» посвящен описанию реализации локализации в приложении.

В приложение кроме основного английского языка дополнительно был добавлен русский язык. Для локализации основных текстовых переменных приложения были созданы файлы `Localizable.strings` для русского и английского языков соответственно.

Третий подраздел «Навигация» посвящен описанию реализации навигации по всему приложению.

Для реализации навигации в приложении был использован фреймворк RxFlow.

Для описания шагов приложения было написано перечисление `AppStep`, которое является наследником `Step` (перечисление фреймворка RxFlow). В данном перечислении описаны все возможные шаги, которые могут понадобиться при навигации в приложении.

Навигация в приложении осуществляет по нескольким потокам. Для каждого потока был создан свой класс, в котором описано поведение приложения при возможных комбинациях потока и шага.

Четвертый подраздел «Сервисы» посвящен описанию создания дополнительных сервисов, с помощью которых был реализован удобный механизм передачи данных во все части приложения.

Основной класс — класс `ServicesContainer`, который является контейнером для сервисов и содержит в себе все сервисы приложения. Данный сервис передается в каждый навигационный поток приложения, из него при переходе на определенный экран будут передаваться необходимые на данном экране сервисы.

В приложении реализован сервис для авторизации и сервис пользователя. Сервис для авторизации разработан для сохранения результатов авторизации (пользователь авторизирован или нет). Сервис пользователя разработан для сохранения токена пользователя, по которому будут осуществляться дальнейшее взаимодействие с сервером.

Пятый подраздел «Вспомогательные классы» посвящен описанию вспомогательных классов.

Вспомогательными классами являются класс констант и класс стилевых настроек.

Константы были созданы для безопасного и простого использования некоторых неизменяемых в процессе работы приложения параметров. Использование констант помогает быстро их изменять и безопасно применять. Для описания констант было создано перечисление `Constants`, которое также содержит вложенные перечисления, отвечающие за каждую часть приложения.

Класс стилевых настроек устанавливает внешний вид верхней и нижней навигационных панелей. Такая настройка позволяет один раз настроить все навигационные панели в одном месте.

Шестой подраздел «Работа с сервером» посвящен описанию механизмов работы с сервером.

Для работы с сервером были написаны классы со статическими методами, которые отправляют запросы на сервер и получают ответы сервера. Для работы с сервером был использован фреймворк Alamofire [12].

Для декодирования ответов сервера в модели, которые будет удобно в дальнейшем использовать в приложении, были написаны классы моделей для всех вариантов ответов. Каждая модель является наследником класса `Codable`. Класс `Codable` совмещается в себе классы `Decodable` и `Encodable`. Класс `Encodable` кодирует модель данных в заданный тип кодировщика. Класс `Decodable` инициализирует модель данных из предоставленного декодера. `Codable` совмещает в себе 2 данных класса и выполняет обе функции по кодированию и декодированию [13].

Седьмой раздел «Точка входа в приложение» посвящен описанию точки входа в приложение.

Точкой входа в приложения является класс `AppDelegate`. В данном классе описываются функции, которые будут вызываться при открытии и закрытии приложения, выводе из фонового режима, перехода в приложение через оповещения и тому подобное.

В разрабатываемом приложении было настроено поведение приложения при его открытии. В данной функции был создан основной поток на-

вигации и сервисы, передаваемые в другие части приложения. Кроме того были установленны настройки для некоторых дополнительных библиотек. Были настроены библиотеки IQKeyboardManagerSwift [14], которая делает автоматическое прокручивание страницы при отображении клавиатуры, закрывающей поле, и AlamofireNetworkActivityLogger [15], которая логирует и выводит в консоль все действия, которые совершаются при взаимодействии с сервером через фреймворк Alamofire [12].

Восьмой раздел «Модули приложения» посвящен описанию реализации всех модулей приложения.

Приложение содержит 5 модулей:

— Модуль «Авторизация»;
— Модуль «Новости»;
— Модуль «Справочник»;
— Модуль «Экспертная система»;
— Модуль «Профиль».

Каждый модуль приложения состоит из одного или нескольких экранов. Каждый экран содержит следующие составляющие:

— файла `.xib`, в котором содержится интерфейс экрана, созданный в InterfaceBuilder;
— файла `View`, в котором содержатся методы по настройке экрана, его показу и связыванию элементов с `ViewModel` данного экрана;
— файла `ViewModel` данного экрана, где происходит все обновление `View` и взаимодействие со сторонними сервисами.

Для связывания `ViewModel` и `View` необходимо передать `ViewModel` в поле класса `View` с помощью функции `setModel(model: T)`.

Каждый класс `View` имеет функции `buildUI()` для настройки элементов интерфейса и функцию `bindUI()` для связывания элементов интерфейса с `ViewModel`.

Классы `ViewModel` имеют входные значения `Input` и выходные `Output`.

Во входные значения передаются наблюдаемые элементы, это может быть нажатие кнопок, изменение полей и тому подобное. В выходных значениях отдаются тоже наблюдаемые элементы, но измененные внутри `ViewModel`. Передача входных значений во `ViewModel` и выходных значений в `View` происходит в функции `bindUI()`.

Конвертация входных значений в выходные происходит внутри функции `transform(input: Input) -> Output`. Внутри данной функции выполняется все дополнительные преобразования по проверке полей или отправке запросов на сервер. Выходные значения, полученные в результате выполнения данной функции отдаются `View`. Дальнейшая работа с ними происходит уже непосредственно во `View`.

Благодаря реализации возможностей реактивного программирования, `ViewModel` все время своего существования реагирует на происходящие события и производит некоторые действия в соответствии с этими событиями.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была достигнута следующая основная цель:

— Разработка клиент-серверного приложения для мобильной операционной системы iOS.

Также в рамках основной цели были рассмотрены и изучены следующие задачи:

— Изучение основ проектирования iOS приложений на языке Swift и необходимых для этого инструментов;
— Изучение основных архитектурных паттернов iOS приложений;
— Изучение фреймворков, позволяющих реализовать возможности реактивного программирования.

В качестве практической части выпускной квалификационной работы было разработано приложение «HappyCats». Данное приложение разрабатывалось для мобильных устройств с операционной системой iOS 12 и выше с поддержкой русского и английского языков. Были разработаны следующие модули:

— Модуль «Авторизация» для авторизации пользователей;
— Модуль «Профиль» для обновления информации о пользователях и их питомцах;
— Модуль «Новости» для просмотра интересных и актуальных новостей из мира кошек;
— Модуль «Справочник» для ознакомления с породами и заболеваниями кошек;
— Модуль «Экспертная система» для определения заболевания питомца по проявившимся симптомам.

Так как данное приложение разрабатывалось для того, чтобы помочь хозяевам кошек больше узнать о своих питомцах, оно содержит много познавательной и обучающей информации. Благодаря экспертной системе, которая была разработана специально для этого приложения, пользователь по проявившимся симптомам может узнать предполагаемое заболевание, которое беспокоит его питомца в данный момент, и начать принимать какие-либо действия по его лечению. Данная экспертная система является главной и отличительной частью приложения «HappyCats».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Усов, В.* Swift. Основы разработки приложений под iOS и macOS / В. Усов. — Санкт-Петербург: Питер, 2018.

2 *Manning, J.* Learning Swift: Building Apps for macOS, iOS, and Beyond / J. Manning, P. Buttfield-Addison, T. Nugent. — Sebastopol: O'Reilly, 2017.

3 Национальная библиотека им. Н. Э. Баумана / Swift (язык программирования) [Электронный ресурс]. — URL: https://ru.bmstu.wiki/index.php?title=Swift_(%FF%E7%FB%EA_%EF%F0%EE%E3%F0%E0%EC%EC%E8%F0%EE%E2%E0%ED%E8%FF)&mobileaction=toggle_view_desktop (Дата обращения 30.04.2020). Загл. с экр. Яз. рус.

4 *Apple Inc.,* The Swift Programming Language / Apple Inc. — Cupertino: Apple Inc., 2014.

5 Xcode 11 Release Notes [Электронный ресурс]. — URL: https://developer.apple.com/documentation/xcode_release_notes/xcode_11_release_notes (Дата обращения 30.04.2020). Загл. с экр. Яз. англ.

6 *Джарратано, Д.* Экспертные системы. Принципы разработки и программирования / Д. Джарратано, Г. Райли. — Москва: Издательский дом «Вильямс», 2007.

7 Coursera / Популярные архитектуры [Электронный ресурс]. — URL: https://ru.coursera.org/lecture/user-interface/populiarnyie-arkhitiektury-H69v4 (Дата обращения 30.04.2020). Загл. с экр. Яз. рус.

8 Официальный блог компании Badoo / Архитектурные паттерны в iOS [Электронный ресурс]. — URL: https://habr.com/ru/company/badoo/blog/281162/ (Дата обращения 02.05.2020). Загл. с экр. Яз. рус.

9 Medium / Реактивное программирование. Начало [Электронный ресурс]. — URL: https://medium.com/@oxmap/%D1%80%D0%B5%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BD%D0%B0%D1%87%D0%B0%D0%BB%D0%BE-4ad548a7d41c (Дата обращения 05.05.2020). Загл. с экр. Яз. рус.

10 RxSwift [Электронный ресурс]. — URL: https://github.com/ReactiveX/RxSwift (Дата обращения 05.05.2020). Загл. с экр. Яз. англ.

11 Начало работы с RxSwift и RxCocoa [Электронный ресурс]. — URL: https://www.raywenderlich.com/1228891-getting-started-with-rxswift-and-rxcocoa (Дата обращения 05.05.2020). Загл. с экр. Яз. англ.

12 Alamofire [Электронный ресурс]. — URL: https://github.com/Alamofire/Alamofire (Дата обращения 22.05.2020). Загл. с экр. Яз. англ.

13 Codable [Электронный ресурс]. — URL: https://developer.apple.com/documentation/swift/codable (Дата обращения 22.05.2020). Загл. с экр. Яз. англ.

14 IQKeyboardManager [Электронный ресурс]. — URL: https://github.com/hackiftekhar/IQKeyboardManager (Дата обращения 22.05.2020). Загл. с экр. Яз. англ.

15 AlamofireNetworkActivityLogger [Электронный ресурс]. — URL: https://github.com/konkab/AlamofireNetworkActivityLogger (Дата обращения 22.05.2020). Загл. с экр. Яз. англ.