МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЛИТВИНЮК ВОЛОДИМИР СЕРГІЙОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
к. т. н., доцент
_____ Т. В. Нескородєва
«_____» _____ 2021р.

РОЗРОБКА СИСТЕМИ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ ДЛЯ УСУНЕННЯ ПРОБЛЕМ ІЗ ТРАНСПОРТНИМ ЗАСОБОМ

Спеціальність 122 «Комп'ютерні науки»

Кваліфікаційна (бакалаврська) робота

Науковий керівник:
Антонов Ю.С., доцент кафедри
інформаційних технологій,
к. фіз.-мат. н., доцент

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2021

Анотація

Литвинюк В.С. Ця дипломна робота присвячена розробці мобільного додатку для визначення несправностей автомобіля. Так, як на даний момент технології розвиваються дуже швидко, усі намагаються автоматизувати процеси, які до недавна робились виключно людьми. Робота складається із 3 розділів, 40 літературних джерел, 13 рисунків. Загальний обсяг роботи складає 62 сторінки.

У вступі наведено актуальність розробки такого мобільного додатку, як він може бути корисним та які механізми та інструменти було вивчено в ході роботи

Другий розділ присвячено вибору інструментів для розробки та їх опису. Також там розглянуто їх функціонал та обгрунтовано чому саме цей інструмент обрано серед аналогів.

Третій розділ містить в собі розробку самого мобільного додатку та його тестування. В ньому детально описано який функціонал притаманний цій системі та як розробляти подібні системи. Розписано, як розроблявся та будувався дизайн додатку. Також описано алгоритм за яким працює додаток.

У висновку підбитий підсумок проведеної роботи та обгрунтована значимість цієї системи.

Ключові слова: несправність, мобільний, XCode, Swift, Apple, метод Електра.

Lytvyniuk V This thesis is devoted to the development of a mobile application for troubleshooting the car. As technology is evolving very fast at the moment, everyone is trying to automate processes that until recently were done exclusively by humans. The work consists of 3 chapters, 11 literary sources, 13 drawings. The total volume of work is 62 page.

The introduction shows the relevance of developing such a mobile application, how it can be useful and what mechanisms and tools were studied in the course of work.

The second section is devoted to the selection of tools for development and their description. Also, their functionality is considered there and it is substantiated why this tool was chosen among analogues.

The third section includes the development of the mobile application and its testing. It describes in detail what functionality is inherent in this system and how to develop such systems. Describes how the application design was developed and built. The algorithm by which the application works is also described.

In conclusion, the results of the work are summarized and the significance of this system is substantiated.

Keywords: fault, mobile, XCode, Swift, Apple, Electra method.

ЗМІСТ

ВСТУП

Актуальність роботи: На сьогоднішній день у людства існує дуже багато проблем та багато із них люди навчились вирішувати самостійно, але бувають ситуації коли людина не має достатньо досвіду та навичок для вирішення того чи іншого питання. На такі випадки люди винайшли системи підтримки прийняття рішень. Вони допомагають нам з вами знайти відповіді на запитання, які нас цікавлять дуже швидко та зручно. Ця галузь дуже стрімко розвивається, але існує ще багато нюансів та підводних каменів, які нам потрібно знайти.

Переважна більшість людей має власне авто та користується ним відносно часто, але лише деякі мають достатні знання та навички для того, щоб власноруч та без допомоги його відремонтувати. Такі проблеми здебільшого вирішуються на станціях технічного обслуговування автомобілів.

Несправності іноді бувають складними та потребують складного ремонту, який зробити без відповідних інструментів не можливо та велика кількість несправностей вирішується легко та швидко без замінних запчастин та інструментів. На дослідження таких випадків і спрямована бакалаврська робота.

Ймовірність таких несправностей можливо передбачити в деякій мірі. Це можливо зробити зібравши в достатньому обсязі інформацію про те, що може спричинити ту чи іншу несправність. Також корисною є інформація про «симптоми», що свідчать про несправність того чи іншого характеру. Таким чином дізнавшись, що саме відбувалося та відбувається за автомобілем можна сказати, що ймовірніше всього потребує ремонту. Дослідивши найбільш часті та ймовірні несправності ми можемо надати інформацію про те, яким чином їх можна усунути.

Мета дослідження: Розробка підтримки прийняття рішень для усунення проблем із транспортним засобом.

Завдання дослідження:

- Огляд існуючих аналогів

- Ознайомлення з мовою програмування Swift
- Ознайомлення з функціоналом баз даних FireBase
- Розробці системи підтримки прийняття рішень для усунення проблем з транспортним засобом
- Тестування системи

Об’єкт дослідження: Різноманітні несправності що виникають під час експлуатації транспортних засобів.

Предмет дослідження: Система підтримки прийняття рішень, що дозволяє вирішувати та усувати проблеми із транспортним засобом.

Практичне значення одержаних результатів: створення незалежного додатку та взаємодія із користувачем для вирішення проблем.

Зв’язок роботи з науковими програмами, планами, темами: Зв’язок роботи з науковими програмами, планами, темами. Наведені у магістерській роботі дослідження пов’язані з фундаментальною науково-дослідною роботою «Дослідження та комп’ютерно-математичне моделювання складних систем та процесів у науці, освіті та інформаційно-комунікаційній діяльності підприємств» (№ держреєстрації 0116U002394, 2018-2022 рр.).

Структура кваліфікаційної роботи: Бакалаврська робота складається із вступу, трьох розділів та висновків до них, списку використаних джерел та одного додатку.

У першому розділі бакалаврської роботи наведено детальну інформацію про історію створення та теорію із характеристиками експертних систем.

У другому розділі розглянуто постановку задачі, яку потрібно вирішити та проведено огляд аналогів таких систем.

У третьому розділі проведено детальний опис та дослідження інструментів, що було використано для розв’язання поставленої задачі та розглянуто процес розробки додатку.

Бакалаврська робота включає в себе 62 сторінку, 13 рисунки і список літератури із 40 джерела.

РОЗДІЛ 1. ІСТОРІЯ ТА АРХІТЕКТУРА ЕКСПЕРТНИХ СИСТЕМ

В даному розділі буде описано та охарактеризовано теоретичні відомості експертних систем.

1.1 Введення в теорію експертних систем

Експертна система (ЕС) – це програма, що має предметну конкретну вузьку предметну галузь та надає найефективніше рішення задачі. Основною частиною ЕС є база знань (БЗ), що накопичується в процесі розробки та побудови.

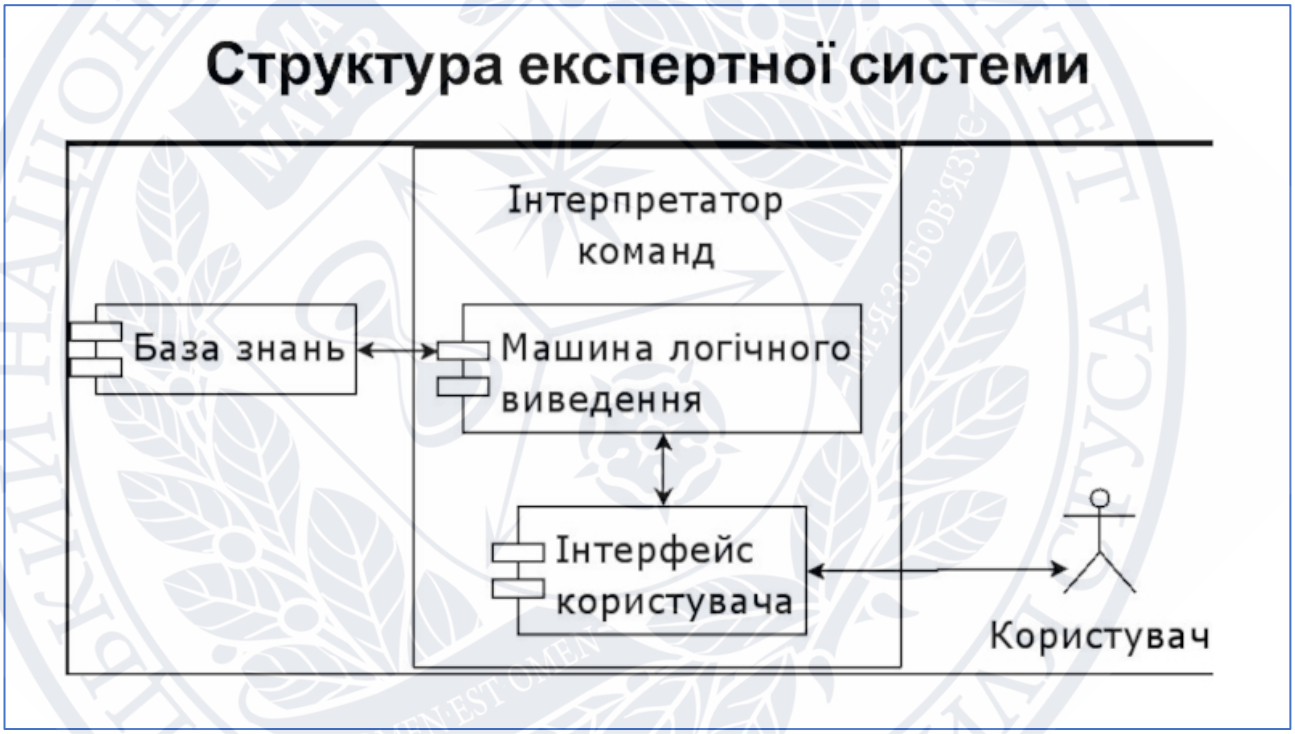

Рисунок 1.1. Структура ЕС [19]

Основні властивості ЕС:

1. Досвід – використання якісних знань експертів у конкретній предметній галузі.

2. Передбачення – можливість передбачати результат в залежності від ситуації.

3. Навчання – можливість навчати людей та надавати якісні знання у конкретній галузі за вже існуючою БЗ реалізованою у ЕС.

Експерт – людина, яка має великий багаж знань та стажу у конкретній галузі. Він здатен знаходити рішення та відповіді на запитання стосуючі конкретної предметної галузі за значно коротший час, використовуючи деякі хитрощі та уловки.

Інженер знань – людина, що займається безпосередньо опитуванням, категоризацією та архітектурними рішеннями побудови бази знань. Він безпосередньо контактує та допомагає розробнику із процессом написання програми.

Засіб побудови – програмний інструмент, що відрізняється від звичайних мов програмування тим, що підтримує значно легше представлення складних понять та використовується розробником в процесі написання програмного коду ЕС.

Види користувачів:

- Розробник
- Інженер знань
- Експерт
- Клерк

Відмінність експертних систем від традиційних програм:

- Компетентність – досягнення експертного рівня
- Символьні міркування – представлення знань в символьному вигляді
- Глибинність – використання складних правил та робота над розв’язком складних завдань
- Самосвідомість – пояснення та дослідження своїх дій

1.2 Технологія розробки ЕС

Процес розробки має 6 етапів:

1. Ідентифікація
2. Концептуалізації

3. Формалізації

4. Виконання

5. Тестування

6. Дослідження та експлуатації

Послідовності дій, які відносятся до кожного з етапів:

1) Етап ідентифікації містить в собі:

- визначити задачі та цілі
- знайти експертів

2) Етап концептуалізації:

- Змістовний аналіз предметної області
- Виділення основних понять
- Визначення методів розв'язання

3) Етап формалізації:

- Обрання програмних засобів розробки
- Визначення способів подання знань
- Основні поняття.

4) Етап виконання заключається в наповнені експертом БЗ

5) Етап тестування полягає в перевірці на компетентність ЕС

6) Етап дослідницької експлуатації перевіряє ЕС на придатність для кінце вих користувачів.

Коефіцієнт довіри – це число, яке означає ступінь впевненості з якою можна вважати даний факт або правило достовірним чи правдивим.

Існує ще одна важлива відмінність ЕС від звичайних програм. Якщо звичайні програми розробляються так, щоб кожен раз отримувати правильний результат, то ЕС розроблені, щоб поводитися як експерти. Вони, як правило, дають правильні відповіді, але іноді, як і люди, здатні помилятися.

Традиційні програми для вирішення складних завдань, теж можуть робити помилки. Але їх дуже важко виправити, оскільки алгоритми, що лежать в їх основі, явно в них не сформульовані. Отже, помилки нелегко знайти і виправити. ЕС, подібно людям, мають потенційну можливість вчитися на своїх помилках.

Також слід відмітити, що ЕС поділяються на 2 види: статичні, динамічні та квазідинамічні.

Статичні ЕС – це ЕС в яких база знань не змінюється у часі та є стабільними.

Динамічні ЕС – це ЕС в яких база знань наповнюється у режимі реального часу з постійною інтерпритацією даних.

Квазідинамічні ЕС – інтерпретують ситуацію, що змінюється з деяким фіксованим інтервалом часу.

1.3 Найвідоміші ЕС та їх опис

MYCIN [20] була ранньою ЕС розробленою за 5 чи 6 років на початку 1970х років в Стенфордському університеті. Вона була написана на Ліспів як докторська дисертація Едварда Шортлайфа під керівництвом Брюса Бучанана, Стенлі Н. Коена та інших. У цій же лабораторії була раніше створена експертна система Dendral, але цього разу увагу було акцентовано на використанні вирішальних правил з елементами невизначеності. MYCIN було спроектовано для діагностування бактерій, що викликають складні інфекції, такі як бактеремія і менінгіт, а також для рекомендації необхідної кількості антибіотиків, в залежності від маси тіла пацієнта. Назва системи походить від суфіксу «-міцин», який часто зустрічається в назвах антибіотиків. Також Mycin використовувався для діагностики захворювань згортання крові.

Призначення цієї програми - бути асистентом лікаря, який не є вузьким спеціалістом в області застосування антибіотиків. В процесі роботи програма формує гіпотези діагнозу і надає їм певні ваги, але самостійно, як правило, не робить остаточного вибору. Після 1976 року система неодноразово модифікувалася і оновлювалася, але базова версія складалася з п'яти компонентів.

- База знань містить фактичні знання, що стосуються предметної області, і відомості про наявні невизначеностях.

- Загальні бази даних пацієнтів містять інформацію про конкретних пацієнтів та їх захворювання.
- Консультуюча програма ставить запитання, виводить укладення системи і дає поради для конкретного випадку, використовуючи інформацію про пацієнта і статичні знання.
- Пояснююча програма відповідає на запитання і дає користувачеві інформацію про те, на чому ґрунтуються рекомендації або висновки, сформульовані системою. При цьому програма приводить трасування процесу вироблення рекомендацій.
- Програма сприйняття знань служить для оновлення знань, що зберігаються в системі, в процесі її експлуатації.

Висновки 1 розділу

В даному розділі описано теоретичні відомості експертних систем, технології іх розробки та класифікації. Також наведено трохи інформації про найвідоміші експертні системи на сьогоднішній день.

РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД АНАЛОГІВ

Даний розділ містить в собі постановку задачі, визначення потрібного функціоналу та огляді існуючих аналогів.

2.1 Постановка задачі

Необхідно розробити мобільний додаток із системою підтримки прийняття рішень для усунення проблем із транспортним засобом для девайсів із операційною системою IOS. Дана система повинна мати зрозумілий для користувача дизайн із мінімальною кількістю лише необхідної ввідної інформації та кнопок. Реалізувати для даної системи декілька тем тестувань із списком запитань, коефіцієнтів та можливих поломок автомобіля. Кожне пройдене тестування повино мати в результаті відповідь із можливою несправністю або ж навпаки із повідомленням про те, що несправність потребує візиту на сервіс технічного обслуговування.

Система включає в себе наступні ключові моменти:

1) Визначення найбільш ймовірної несправності автомобіля
2) Підтримка декількох девайсів із операційною системою IOS
3) Інтуїтивно зрозумілий дизайн
4) Можливість змінювати текст запитань безпосередньо в базі даних

Доступний функціонал:

1) Вибір теми діагностики автомобіля
2) Проходження діагностики
3) Зв’язок із розробником

Результати діагностики не зберігаються та їх можна переглянути лише один раз після проходження діагностики.

2.2 Огляд аналогів

В якості прикладів аналогів було обрано декілька додатків та перший із них «Nike+ Training Club». Цей додаток розроблено для людей, які стежать

за своїми тренуваннями та здоров’ям. Він допомагає отримати повну інформацію про минулі тренування. Також його функціонал надає можливість створити персональний список тренувань, які підходять індивідуально для кожного. Таким чином цей додаток пропонує фізичні вправи, їх комбінації для досягнення максимального результату від тренувань.

Переваги даного додатку між іншими:

- Найбільш інтуїтивно зрозумілий дизайн
    - Дизайн виконано без непотрібної інформації. Він містить лише найбільш необхідну інформацію та користувач, який вперше відкрив додаток інтуїтивно розуміє де знаходиться та чи інша кнопка
- Велика кількість різноманітних тренувань
    - Цей додаток має на борту величезний список стандартних тренувань, які підходять для багатьох одночасно
- Можливість створювати персональні тренування
    - Якщо ж вам не підходять стандартні тренування або ж з якоїсь іншої причини вам потрібно створити тренування, яке підходить індивідуально саме для вас. Цей сервіс надає таку можливість
- Можливість відслідковувати якість тренувань
    - Сервіс надає можливість переглянути усі показники часу і не тільки минулих тренувань
- Присутня можливість слідкувати за показниками ккал та серцебиття
    - Додаток має можливість слідкувати за станом вашого здоров’я та його показниками. Таким чином сервіс може надсилати вам повідомлення за необхідності попереджувати про небезпечні перенавантаження та інше
- Присутні навчальні відеозаписи для новачків із правильним виконанням усіх вправ

- Сервіс підходить не тільки для людей, які вже знають як правильно виконувати усі вправи, а також для тих хто не має усіх навичок в повному обсязі

- Стабільність роботи
  - Цей сервіс знаходиться на ринку вже дуже давно та має велику базу клієнтів, тому оновлення виходять регулярно та усі несправності виправляються дуже швидко. Також він підтримується на достатньо великій кількості різноманітних девайсів

В якості наступного прикладу розглянемо «Таблиця калорійності». Цей додаток допомагає визначити вашу норму кілокалорій на день та скільки кілокалорій ви вживаєте в середньому. Також даний сервіс допомагає зібрати аналітику ваших показників, які зміни раціону потрібно зробити та від яких продуктів слід відмовитись, а які навпаки споживати більше.

Переваги та недоліки даного сервісу:

- Мінімалістичний дизайн
  - Дизайн даного додатку має мінімальну кількість лише потрібної інформації та не містить реклами взагалі
- Не вибагливий до місця використання
  - Користувачеві не потрібен інтернет для того, щоб використовувати даний додаток. Він оновлює локальну базу даних кожного разу при підключенні до інтернету.
- Легкий у використанні
  - Кожен користувач повинен лише правильно ввести дані про свій організм та раціон, а далі додаток самостійно визначить усі необхідні показники та надасть результат роботи сервісу.
- Безкоштовність
  - Цей сервіс є абсолютно безкоштовним, тобто для того, щоб його використовувати користувач не повинен оплачувати його покупку, також відсутні будь-які платежі по підписці та інші

види платежів. Для використання потрібно лише завантажити його на свій мобільний телефон

- Стабільність у роботі
    - Цей додаток знаходиться на ринку вже давно та має дуже щільно протестований функціонал та виправлені усі критичні недоліки

Висновки до 2 розділу.

В даному розділі була чітко сформована постановка задачі та визначена функціональність системи підтримки прийняття рішень, яку було розроблено та досліджено. Також було оглянуто декілька систем-аналогів, які допомагають прийняти рішення в різних ситуаціях. Було чітко описані їх переваги та недоліки в порівнянні із конкурентами в суміжних галузях.

РОЗДІЛ 3. ОПИС ТА ПІДБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ. ОГЛЯД ФУНКЦІОНАЛЬНОСТІ ІНСТРУМЕНТІВ ТА ПРОЦЕС РОЗРОБКИ

Цей розділ присвячено інструментам розробки, що було використано для розробки саме даної системи підтримки прийняття рішень та огляду їх можливостей й функціоналу, що вони надають розробникам. Сьогодні існує дуже багато схожих технологій між собою по функціоналу та можливостям, тому в цьому розділі буде описано також чому було обрано саме ці інструменти розробки.

3.1 IOS

IOS (IPhone OS) – це операційна система розроблена компанією Apple для мобільних девайсів таких як IPhone та IPad[11].

Відмінними рисами системи є[12]:

1. **Конфіденційність**. Користувач повністю контролює доступ до своїх особистих даних, тому жодна з програм їх не отримає без Вашої згоди.
2. **Безпека**. В системі безпеки IOS все продумано до дрібниць, тому смартфону не будете отримувати будь віруси, шпигунські програми, втрата даних або їх злом. Кожен користувач має свій особистий кабінет з логіном і паролем, який дає доступ до даних тільки йому. Є можливість встановити пароль, який захистить весь вміст телефону від сторонніх.
3. **Вбудовані функції.** Завдяки деяких функцій пристрій може допомогти впоратися з багатьма завданнями. Серед вбудованих функцій слід виділити наступні:

- **Touch ID** - система сканування відбитків пальця, яка гарантує захист даних від чужих людей. При цьому є можливість особисто додати відбитки знайомих або родичів, якщо це необхідно.

- **Face ID** - новітня система розпізнавання обличчя користувача. З'явилася технологія зовсім недавно в 2017 році і поки доступна тільки в останньому iPhone X.

- **Siri** - це особистий помічник, який встановлений на всіх iphone. Додаток обробляє мова користувача, відповідає на питання чи виконує команди, які йому задають.

- **Знайти IPhone** - функція, яка дозволить знайти втрачене пристрій. Вона визначає з точністю розташування і відтворює на ньому звуковий сигнал.

Метою цієї роботи було розробити систему підтримки прийняття рішень для операційної системи IOS.

3.2 Середовище розробки XCode

XCode – інтегрована середа розробки (IDE) програмного забезпечення під системи MacOS, tvOS, IOS, WatchOS, розроблена компанією Apple[]. Перша версія була випущена у 2003 році. Стабільну та актуальну версію цього середовища розробки може встановити кожен охочий через Mac App Store на свій комп'ютер під керуванням операційної системи MacOS. Сертифіковані розробники також мають доступ до бета версій через веб-сайт Apple Developer.

Особливості XCode:

- Інтерфейс Xcode інтегрує редагування коду, проект призначеного для користувача інтерфейсу, управління активами, тестування і налагодження в єдиному вікні робочої області.
- Пакет Xcode включає в себе змінену версію вільного набору компіляторів GNU Compiler Collection і підтримує мови C, C ++, Objective-C, Objective-C ++, Swift, Java, AppleScript, Python і Ruby з різними моделями програмування, включаючи Cocoa, Carbon і Java.
- Так само до складу входять більша частина документації розробника від Apple і Interface Builder - додаток, що використовується для створення графічних інтерфейсів.

Storyboard (рис. 3.1) – чудова функція Xcode, яка була представлена разом з iOS 5 і заощадила багато часу для створення призначеного для користувача інтерфейсу ваших додатків.

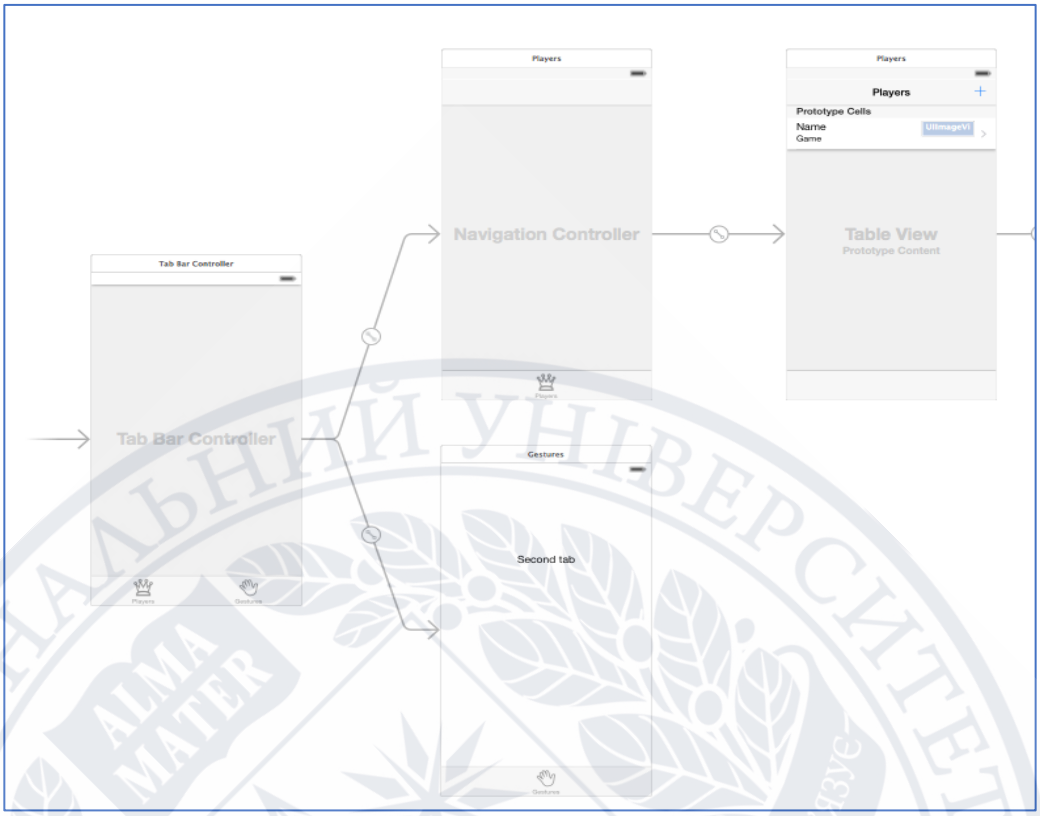

Рисунок 3.1. Приклад схема storyboard [2]

Сторіборди мають ряд переваг:

1. У storyboard дуже зручно спостерігати загальну картину вашого застосування і взаємозв'язку між його сторінками. У storyboard можна відстежувати все що завгодно, тому що загальний дизайн програми міститься в одному єдиному файлі, а не розподілений між декількома файлами nib.
2. Storyboards можуть описувати переходи між різними вікнами. Ці переходи називаються "segues", які створюються шляхом з'єднання двох сторінок прямо в storyboard. Завдяки цим segues, ви пишіть менше коду для вашого призначеного для користувача інтерфейсу.
3. Storyboards полегшують вашу роботу з табличними типами, з осередками. Ви можете створити ваші таблиці практично повністю з storyboard, але ж знову таки це саме те, що зменшує в рази ваш код, який вам би довелося написати.
4. Storyboards спрощують роботу при використання автопозиціонування. Автопозиціонування - потужна функція, яка дозволяє визначати

математичні взаємини між елементами, які мають певні розміри і позиції, і так само спрощує роботу по відображенню вашого застосування на різних пристроях з різними розширеннями екрану.

У XCode є вбудовані віджети для розробки дизайну та функціоналу.

Основні віджети XCode: Label, Button, Text Field, Slider, Switch, Table, View, Navigation Bar, View Controller, Navigation Controller.

Основні характеристики базових віджетів буде описано нижче: [2]

Button:[2] коли ви натискаєте кнопку або вибираєте кнопку, яка має фокус, кнопка виконує будь-які прикріплені до неї дії. Ви повідомляєте призначення кнопки за допомогою текстової мітки, зображення або того й іншого. Зовнішній вигляд кнопок можна налаштувати, тому ви можете тонувати кнопки або форматувати заголовки відповідно до дизайну вашого додатка. Ви можете додавати кнопки до свого інтерфейсу програмно або за допомогою Interface Builder.

Додаючи кнопку до свого інтерфейсу, виконайте такі дії:

- Встановіть тип кнопки під час створення.
- Надайте рядок або зображення в заголовку; розмір кнопки відповідно до вашого вмісту.
- Підключіть один або кілька методів дій до кнопки.
- Налаштуйте правила автоматичного макетування, щоб регулювати розмір і положення кнопки у вашому інтерфейсі.
- Надайте інформацію про доступність та локалізовані рядки.

Налаштування зовнішнього вигляду кнопки

Тип кнопки визначає її основний вигляд та поведінку. Ви вказуєте тип кнопки під час створення, використовуючи init(type:) метод або у своєму файлі розкадрування. Після створення кнопки ви не можете змінити її тип. Найбільш часто використовуваними типами кнопок є користувацькі та системні типи, але за потреби використовуйте інші типи.

Label:[2] Ви можете налаштувати загальний вигляд тексту мітки та використовувати атрибутивні рядки для налаштування зовнішнього вигляду

підрядків у тексті. Додайте та налаштуйте мітки у своєму інтерфейсі програмно або за допомогою інспектора атрибутів у Interface Builder.

Виконайте такі дії, щоб додати мітку до свого інтерфейсу:

- Укажіть рядок або атрибутивний рядок, що представляє вміст.
- Якщо ви використовуєте неатрибутивний рядок, налаштуйте зовнішній вигляд мітки.
- Налаштуйте правила автоматичного макетування, щоб регулювати розмір та положення етикетки у вашому інтерфейсі.
- Надайте інформацію про доступність та локалізовані рядки.

Text Field:[2] Ви для багатьох різних типів введення, таких як звичайний текст, електронні листи, номери тощо. Текстові поля використовують механізм цільової дії та об'єкт-делегат, щоб повідомляти про зміни, внесені в процесі редагування.

На додаток до основної поведінки редагування тексту, ви можете додавати накладені подання до текстового поля, щоб відображати додаткову інформацію та надавати додаткові елементи керування. Ви можете додати власні види накладання для таких елементів, як кнопка закладок або піктограма пошуку. Текстові поля надають вбудований режим накладання для очищення поточного тексту. Використання власних подань накладання необов’язкове.

Після додавання текстового поля до вашого інтерфейсу ви налаштовуєте його для використання у вашому додатку. Конфігурація передбачає виконання деяких або всіх наступних завдань:

- Налаштуйте одну або кілька цілей і дій для текстового поля.
- Налаштуйте пов'язані з клавіатурою атрибути текстового поля.
- Призначте об'єкт-делегат для обробки важливих завдань, таких як:
    - Визначення, чи слід користувачеві дозволяти редагувати вміст текстового поля.
    - Перевірка тексту, введеного користувачем.
    - Відповідь на натискання кнопки повернення клавіатури.
    - Переадресація введеного користувачем тексту в інші частини вашої програми.

- Зберігайте посилання на текстове поле в одному з об’єктів контролера.

Інформацію про методи об'єкта-делегата текстового поля див .UITextFieldDelegate

Зовнішній вигляд текстового поля:

Ви налаштовуєте клавіатуру текстового поля, використовуючи властивості протоколу, який приймає клас. UIKit підтримує стандартні клавіатури для поточної мови користувача, а також підтримує спеціалізовані клавіатури для введення цифр, URL-адрес, електронних адрес та інших конкретних типів інформації. Ви використовуєте властивості цього протоколу для налаштування рис клавіатури, таких як:UITextInputTraitsUITextField

- Тип клавіатури для відображення
- Поведінка автокапіталізації клавіатури
- Поведінка автокорекції клавіатури
- Тип клавіші повернення для відображення

Slider:[2] Під час переміщення **великого пальця** повзунка він передає своє оновлене значення будь-яким діям, прикріпленим до нього. Зовнішній вигляд повзунків можна налаштувати; Ви можете тонувати доріжку та великий палець, а також забезпечити зображення, що відображатимуться на кінцях повзунка. Ви можете додати повзунки до свого інтерфейсу програмно або за допомогою Interface Builder.

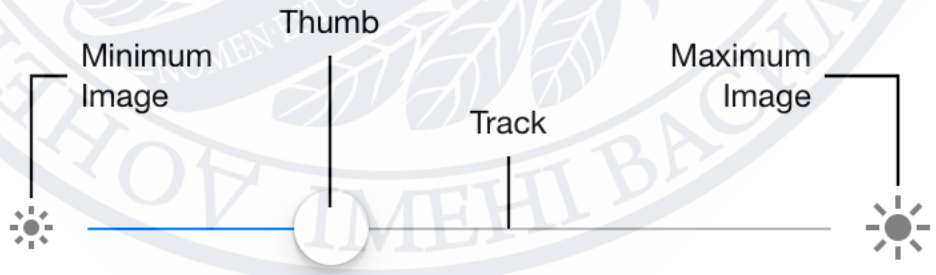

Рисунок 3.2. Slider [2]

Для додавання повзунка до вашого інтерфейсу потрібно виконати наступні дії:

- Вкажіть діапазон значень, який представляє повзунок.
- За бажанням налаштуйте зовнішній вигляд повзунка з відповідними кольорами відтінків та обмежте зображення.

- Підключіть один або кілька методів дій до повзунка.
- Налаштуйте правила автоматичного макетування, щоб регулювати розмір і положення повзунка у вашому інтерфейсі.

Switch:[2] UISwitchКлас оголошує властивість і метод, щоб контролювати його стан вкл / викл. Як і у випадку UISlider, коли користувач маніпулює керуванням перемикачем («перевертає» його), це ініціює подію .valueChanged

Ви можете налаштувати зовнішній вигляд перемикача, змінивши колір, який використовується для тонування перемикача, коли він увімкнений або вимкнений.

View Controller:[2] Клас визначає загальну поведінку, яке є загальним для всіх контролерів уявлення. Ви рідко створюєте екземпляри класу безпосередньо. Натомість ви підклас і додаєте методи та властивості, необхідні для управління ієрархією подання контролера перегляду .UIViewControllerUIViewControllerUIViewController

Основні обов'язки контролера перегляду включають наступне:

- Оновлення вмісту подань, як правило, у відповідь на зміни основних даних.
- Відповідь на взаємодію користувачів з поданнями.
- Зміна розміру подань та управління макетом загального інтерфейсу.
- Координація з іншими об’єктами, включаючи інші контролери перегляду, у вашій програмі.

Контролер перегляду тісно пов'язаний з поданнями, якими він керує, і бере участь у обробці подій у своїй ієрархії подання. Зокрема, контролери подання є UIResponder об'єктами і вставляються в ланцюжок відповідачів між кореневим видом контролера подання та супер переглядом цього подання, який зазвичай належить іншому контролеру перегляду. Якщо жоден з подань контролера перегляду не обробляє подію, контролер перегляду має можливість обробити подію або передати її до супер перегляду.

Контролери перегляду рідко використовуються ізольовано. Натомість ви часто використовуєте кілька контролерів перегляду, кожному з яких належить частина користувацького інтерфейсу вашого додатка. Наприклад, один контролер перегляду може відображати таблицю елементів, тоді як інший контролер перегляду

відображає вибраний елемент із цієї таблиці. Зазвичай одночасно видно лише види з одного контролера перегляду. Контролер перегляду може представляти інший контролер перегляду для відображення нового набору переглядів, або він може виконувати функцію контейнера для вмісту інших контролерів перегляду та анімувати подання, як хоче.

Navigation Controller: Контролер навігації - це контролер перегляду контейнера, який керує одним або кількома контролерами дочірнього перегляду в інтерфейсі навігації. У цьому типі інтерфейсу одночасно видно лише один дочірній контролер подання. Вибір елемента в контролері перегляду висуває на екран новий контролер перегляду за допомогою анімації, тим самим приховуючи попередній контролер перегляду. Натискання кнопки "Назад" на панелі навігації у верхній частині інтерфейсу видаляє контролер вигляду зверху, тим самим відкриваючи контролер перегляду внизу.

Використовуйте навігаційний інтерфейс, щоб імітувати організацію ієрархічних даних, керованих вашим додатком. На кожному рівні ієрархії ви надаєте відповідний екран (керований користувацьким контролером перегляду) для відображення вмісту на цьому рівні. На малюнку (рис. 3.3) наведено приклад навігаційного інтерфейсу, представленого додатком Налаштування в iOS Simulator. Перший екран представляє користувачеві список програм, що містять налаштування. Вибір програми відкриває окремі установки та групи установок для цієї програми. Вибір групи дає більше налаштувань тощо. Для всіх, крім кореневого подання, контролер навігації забезпечує кнопку повернення, щоб дозволити користувачеві повернутися назад вгору по ієрархії.

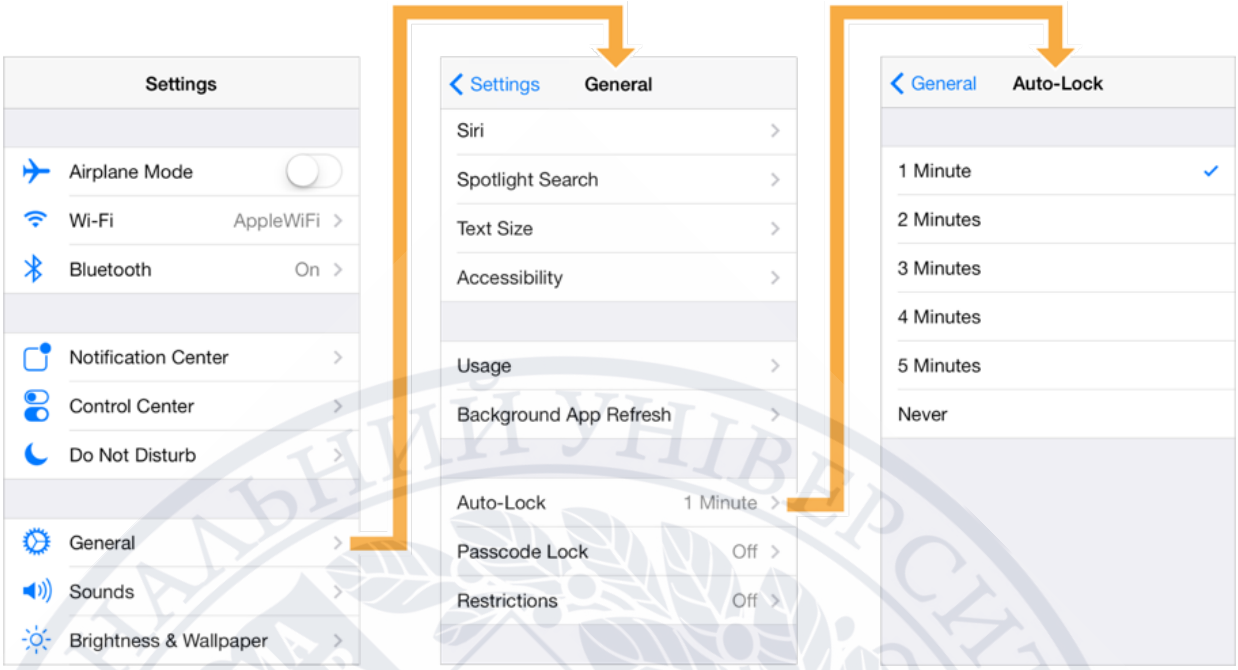

Рисунок 3.3. Navigation Controller [2]

Об'єкт контролера навігації управляє своїми дочірніми контролерами подання за допомогою упорядкованого масиву, відомого як **стек навігації** . Перший контролер подання в масиві є кореневим контролером подання і представляє нижню частину стека. Останній контролер перегляду в масиві є найвищим елементом стека і представляє контролер перегляду, який зараз відображається. Ви додаєте та видаляєте контролери перегляду зі стеку за допомогою segues або за допомогою методів цього класу. Користувач також може видалити верхній контролер перегляду за допомогою кнопки "Назад" на навігаційній панелі або жесту пальцем по лівому краю.

Контролер навігації управляє панеллю навігації у верхній частині інтерфейсу та додатковою панеллю інструментів внизу інтерфейсу. Панель навігації завжди присутня і нею керує сам контролер навігації, який оновлює панель навігації, використовуючи вміст, наданий її контролерами дочірнього перегляду. Коли властивість є, навігаційний контролер аналогічно оновлює панель інструментів із вмістом, наданим найвищим контролером перегляду.isToolbarHiddenfalse

Контролер навігації координує свою поведінку зі своїм delegate об'єктом. Об'єкт-делегат може замінити натискання або вискакування контролерів подання, забезпечити власні переходи анімації та вказати бажану орієнтацію

навігаційного інтерфейсу. Об’єкт делегата, який ви надаєте, повинен відповідати протоколу.UINavigationControllerDelegate

На малюнку (рис. 5) показано взаємозв'язок між навігаційним контролером та об'єктами, якими він керує. Використовуйте вказані властивості контролера навігації для доступу до цих об’єктів.

Перегляди контролера навігації

Контролер навігації є контролером перегляду контейнера, тобто він вбудовує вміст інших контролерів перегляду всередину себе. Ви отримуєте доступ до подання контролера навігації з його view властивості. Цей вигляд включає панель навігації, додаткову панель інструментів та подання вмісту, що відповідає верхньому контролеру перегляду. На малюнку (рис. 3.4) показано, як ці види збираються для представлення загального навігаційного інтерфейсу. (На цьому малюнку навігаційний інтерфейс додатково вбудований всередину інтерфейсу панелі вкладок.) Хоча зміст переглядів панелі навігації та панелі інструментів змінюється, а самі подання - ні. Єдиний вигляд, який насправді змінюється, - це власний вигляд вмісту, наданий найвищим контролером перегляду у стеці навігації.

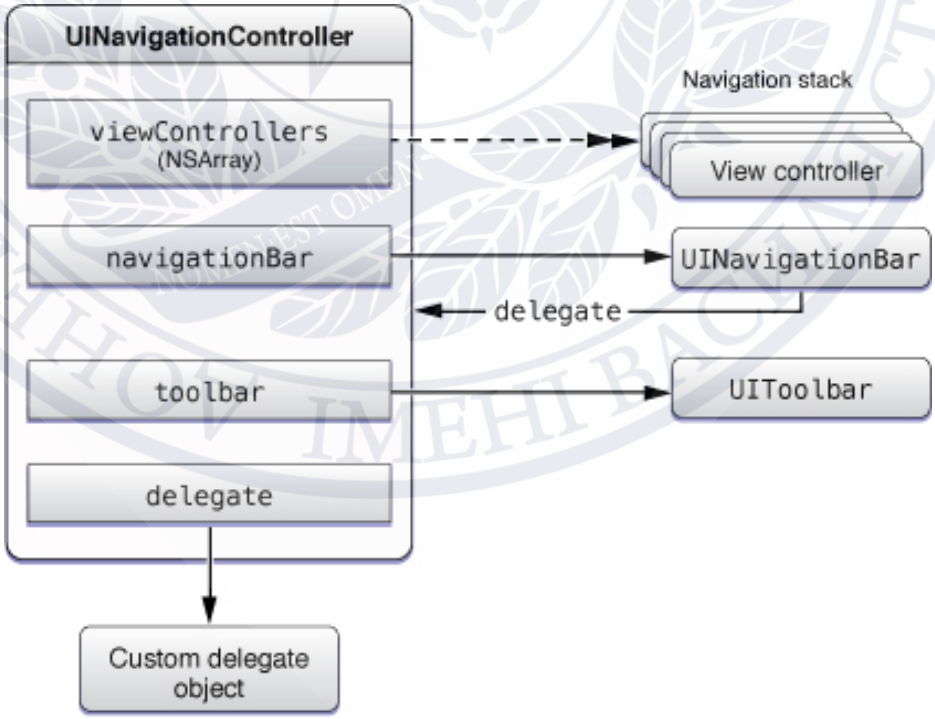

Рисунок 3.4. Структура Navigation Controller

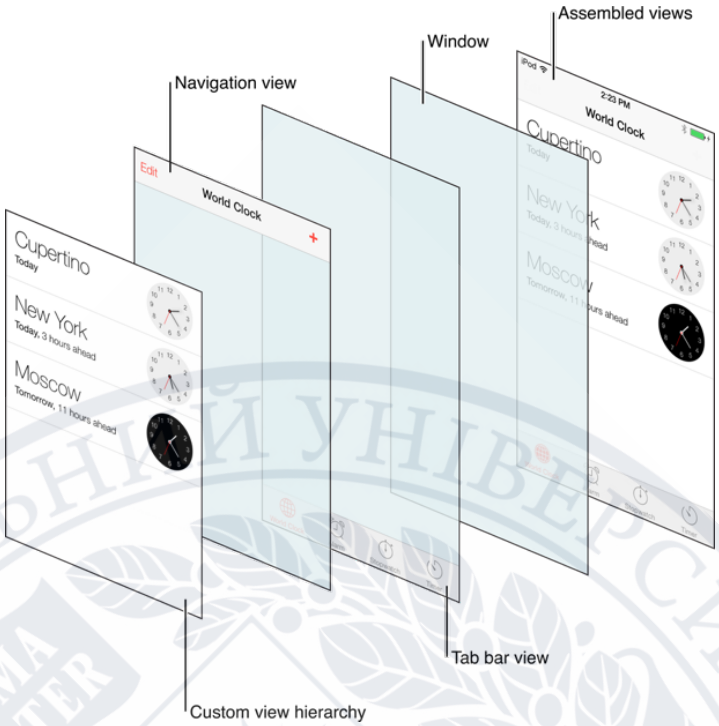

Рисунок 3.5. Види Navigation Controller

Navigation Item: При побудові навігаційного інтерфейсу кожен контролер перегляду, який ви натискаєте на стек навігації, повинен мати об’єкт, який містить кнопки та види, які ви хочете відобразити на панелі навігації. Керуючий об'єкт використовує елементи навігації двох верхніх контролерів перегляду, щоб заповнити навігаційну панель вмістом. UINavigationItemUINavigationController

Елемент навігації завжди відображає інформацію про пов'язаний з ним контролер подання. Елемент навігації повинен містити заголовок, який відображатиметься, коли контролер перегляду знаходиться вгорі на стеці навігації. Елемент може також містити додаткові кнопки для відображення на правій (або завершальній) стороні панелі навігації. За допомогою властивості можна вказати кнопки та подання, які відображатимуться на лівій (або передній) стороні панелі інструментів, але навігаційний контролер відображає ці кнопки лише тоді, коли є вільне місце.leftBarButtonItems

Властивість елемента навігації відображає кнопки Назад ви хочете відобразити, коли поточний контролер уявлення трохи нижче верхнього контролера уявлення. Кнопка "Назад" не відображається, коли поточний контролер подання знаходиться вгорі. backBarButtonItem

Визначаючи кнопки для елемента навігації, ви повинні використовувати об'єкти. Якщо ви хочете відобразити власні подання на панелі навігації, ви повинні обернути ці подання всередині об’єкта, перш ніж додавати їх до елемента навігації. UIBarButtonItemUIBarButtonItem

Toolbar: Для створення елементів панелі інструментів використовуйте клас. Щоб додати елементи панелі інструментів на панель інструментів, використовуйте метод. UIBarButtonItemsetItems(_:animated:)

Зображення панелі інструментів, що представляють нормальний та виділений стан елемента, походять від зображення, яке ви встановили за допомогою успадкованої image властивості класу. Зображення забарвлене за допомогою панелі інструментів. UIBarItemtintColor

Якщо вам потрібні елементи керування стилем перемикача, використовуйте клас замість. UITabBarUIToolbar

Налаштування зовнішнього вигляду

Ви використовуєте методи, перелічені в Налаштування зовнішнього вигляду, щоб налаштувати зовнішній вигляд панелей інструментів. Ви надсилаєте повідомлення установника проксі-серверу зовнішнього вигляду ([UIToolbar appearance]), щоб налаштувати всі панелі інструментів, або певному UIToolbarекземпляру. Коли властивість залежить від метричних показників, зазвичай слід вказувати значення як для, так і для .UIBarMetrics.defaultlandscapePhone

Table View: Подання таблиць на iOS відображають один стовпець вертикально прокручуваного вмісту, розділений на рядки. Кожен рядок таблиці містить один фрагмент вмісту вашої програми. Наприклад, програма Контакти відображає ім’я кожного контакту в окремому рядку, а головна сторінка програми Налаштування відображає доступні групи налаштувань. Ви можете налаштувати таблицю для відображення одного довгого списку рядків, або згрупувати пов’язані рядки за розділами, щоб полегшити навігацію вмістом.

Таблиці зазвичай використовуються програмами, дані яких структуровані або організовані ієрархічно. Додатки, що містять ієрархічні дані, часто використовують

таблиці разом із контролером перегляду навігації, що полегшує навігацію між різними рівнями ієрархії. Наприклад, програма Налаштування використовує таблиці та навігаційний контролер для організації системних налаштувань.

UITableViewкерує основним виглядом таблиці, але ваш додаток надає клітинки (об’єкти), які відображають фактичний вміст. Стандартні конфігурації комірок відображають просту комбінацію тексту та зображень, але ви можете визначити власні комірки, які відображають будь-який вміст, який ви хочете. Ви також можете надати подання верхнього та нижнього колонтитулів, щоб надати додаткову інформацію для груп комірок .UITableViewCell

Додавання подання таблиці до вашого інтерфейсу

Щоб додати перегляд таблиці до вашого інтерфейсу, перетягніть об’єкт «Перегляд таблиці» ( ) на свою розкадрування. Xcode створює нову сцену, яка включає в себе як контролер перегляду, так і табличний вигляд, готовий до налаштування та використання. UITableViewController

Перегляди таблиць керуються даними, зазвичай вони отримують свої дані від об’єкта джерела даних, який ви надаєте. Об'єкт джерела даних керує даними вашої програми та відповідає за створення та налаштування комірок таблиці. Якщо вміст вашої таблиці ніколи не змінюється, замість цього ви можете налаштувати цей вміст у своєму файлі розкадрування.

Інформацію про те, як вказати дані таблиці, див. У розділі Заповнення таблиці даними.

Збереження та відновлення поточного стану таблиці

Перегляди таблиць підтримують відновлення програми UIKit. Щоб зберегти та відновити дані таблиці, призначте властивість подання таблиці непусте значення . Коли його батьківський контролер подання зберігається, подання таблиці автоматично зберігає шляхи індексу для поточно вибраних та видимих рядків. Якщо об'єкт джерела даних таблиці приймає протокол, таблиця зберігає унікальні ідентифікатори, які ви надаєте для цих елементів, замість шляхів їх індексу. restorationIdentifierUIDataSourceModelAssociation

Інформацію про те, як зберегти та відновити інформацію про стан програми, див. У розділі Збереження користувацького інтерфейсу програми під час запусків.

3.3 Мова програмування SWIFT

SWIFT – багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C та бути стійкішою до помилкового коду[1]. Мова побудована з LLVM компілятором вбудованим в XCode 6 beta. У електронному книжковому магазині iBooks є безкоштовний посібник із документацією до цієї мови програмування.

Дана мова програмування дозволяє писати код та розробляти програми для девайсів під управлінням операційних систем розроблених компанією Apple для своїх гаджетів, а саме MacOS, IOS, WatchOS, tvOS та інші.

Swift успадкував найкращі елементи мов C і Objective C, саме тому синтаксис дуже звичний для розробників, що з ним вже знайомі, але в той самий час різниться використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду.

Попри це все код написаний мовою програмування Swift компілюється у машинний код, що забезпечує швидкодію. За офіційною статистикою компанії Apple однаковий код написаний двома мовами Swift та Objective C працює в 1,3 рази на мові Swift. Також швидкодія забезпечується засобами підрахунку посилань на об’єкти, що використовується замість збирача сміття Objective C.

Окремо хотілось би відмітити, що не слід плутати мову програмування Swift розроблену компанією Apple із скриптовою мовою Swift націленою на багатонитеве програмування, що поставляється із вільною ліцензією Apache.

Плюси в порівняні із Objective C:

1. Швидкодія
2. Безпека
3. Функціональність

4. Більша кількість розробників, що використовують Swift

Для розробки цією мовою програмування компанією Apple було розроблено середовище розробки під назвою XCode[4]. Воно має великий функціонал та дуже гнучкі налаштування індивідуально під розробника. Також слід відмітити, що Swift може бути викликано із термінала, уможливлює розробку цією мовою програмування не тільки на девайсах під управлінням операційної системи MacOS, а також дає змогу розробляти на гаджетах під управлінням операційної системи Linux.

В даній роботі цю мову програмування використано, як основний інструмент розробки. Приблизно вісімдесят відсотків усієї роботи написано мовою програмування SWIFT

3.4 Макет дизайну

Дизайн даної системи виконано із дотриманням усіх вимог із розділу 1. Було прийнято рішення розробляти дизайн за мотивами веб-додатку розробленого в минулому році. Основні кольори видимої частини для користувачів чорний білий та сірий. [15]

Основна сторінка(рис. 10) містить на собі назву, коротку інформацію про те, що це за додаток, для чого він потрібен та які функції він виконує. Також на першій сторінці присутня кнопка «Тести» натиснувши яку користувач перейде на сторінку вибору тестування. Після натискання активується «меню навігації» зверху завдяки якому користувачі можуть переходити на попередні сторінки.

На наступній сторінці вибору тестування(рис. 10) користувачі мають змогу обирати на яку тематику тестування їм потрібно проходити. Вибір проводиться за допомогою натискання на кнопки із відповідними назвами тестувань. Окрім цієї інформації ця сторінка не містить нічого іншого для того, щоб не перенавантажувати інтерфейс та не лякати користувача великою кількістю інформації.

Наступна сторінка(рис. 3.6) – це сторінка тестування із запитаннями та варіантами відповідей «Так» або «Ні». На цій сторінці користувач може обирати варіанти відповідей натискаючи відповідні кнопки.

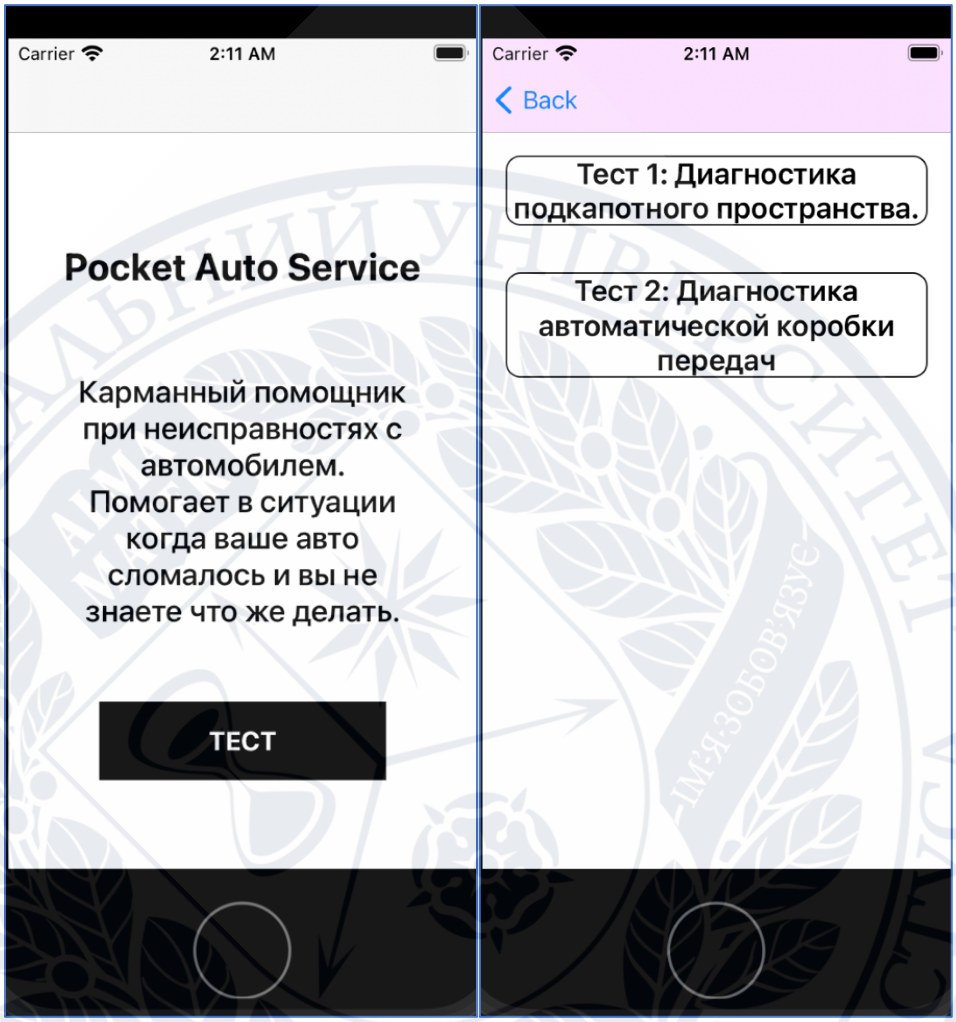

Рисунок 3.6. Головна сторінка додатку та тестувань

Для уникнення проблем із відповідями користувач може змінити свою відповідь, якщо він випадково обрав не той варіант, який хотів, натиснувши кнопку із протилежним варіантом відповіді. Слід відмітити, що поки користувач не відповів на усі запитання кнопка отримання результату буде підсвічена сірим кольором та не будет клікабельна, а якщо ж відповіді на усі запитання успішно обрані, то ця кнопка буде підсвічена чорним кольором та стане клікабельна.

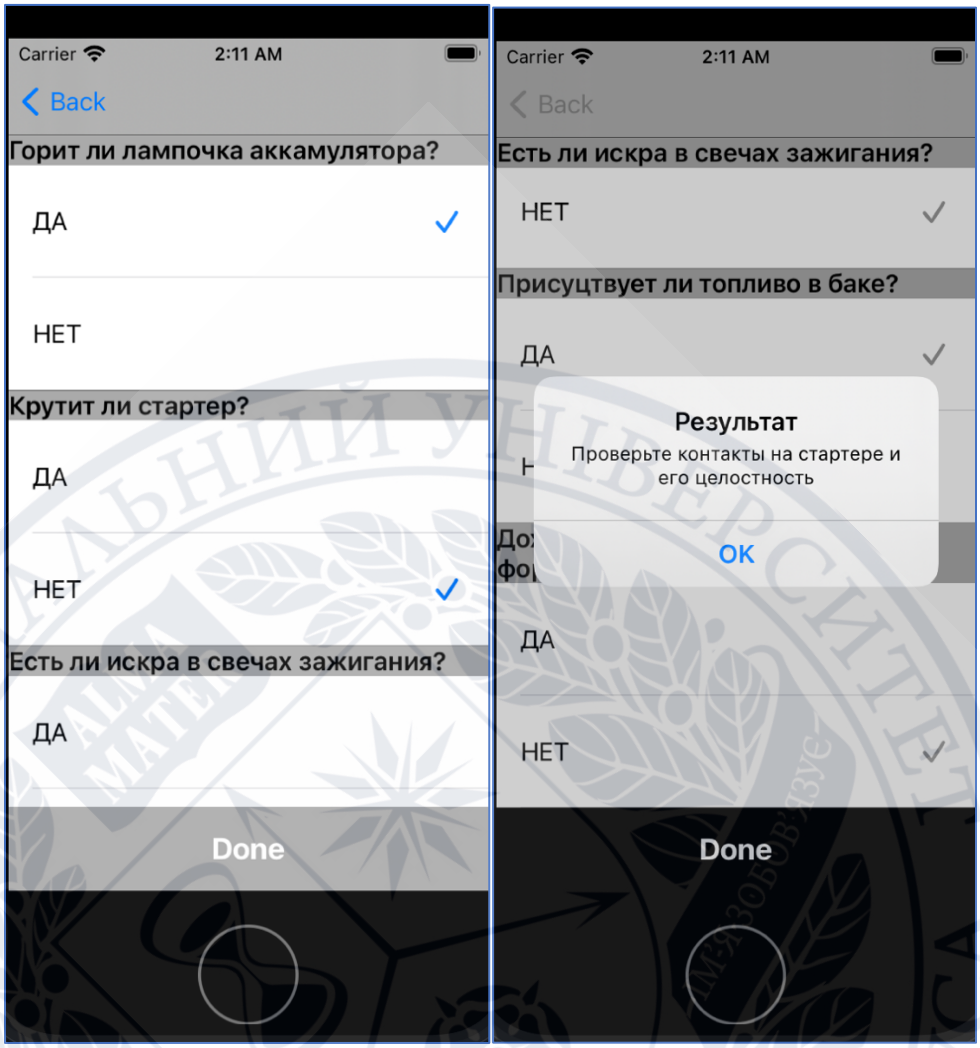

Рисунок 3.7. Сторінка тестування та результату

Натиснувши її користувач отримає результат у діалоговому вікні «Результат».

3.5 Реалізація макету в XCode

XCode має для цього відповідний файл, який має назву Storyboard. Відкривши цей файл через файловий менеджер вбудований в XCode можна побачити пусту початкову сторінку на яку вже можна додавати віджети, кнопки та різні інші елементи дизайну. Додати ці віджети неймовірно просто, потрібно лише натиснути кнопку «+» додати віджет та обрати потрібний вам віджет. Наприклад «Label» для того, щоб розмістити якусь інформацію на екрані.[16]

Також для реалізації навігації використано декілька способів, а точніше 2.

Перший спосіб полягає в тому, щоб реалізувати власні кнопки із переходами по сторінкам. Це зробити дуже просто, потрібно лише додати на перший екран кнопку та оформити її як потрібує цього постановка задачі. Далі

затиснувши ліву кнопку миші перетягнути курсор на наступний екран. Таким чином з'являється зв'язок, що пов'язує кнопку та перехід до наступного екрану.

Другий спосіб створення навігації, що було використано в даній роботі – це спосіб через Navigation Bar. Для реалізації цього способу навігації потрібно лише додати Navigation View Controller. Він додається та виглядає як іще один екран, але його не видно користувачам. По суті коли програма запускається, то вона починає свою роботу саме з цього Navigation View Controller, який є невидимим та перекидає нас на першу сорінку. Далі коли ми будем переходити на наступну сторінку зверху буде спливати Navigation Bar із кнопкою "Back" для того, щоб повернутися назад. Саме ось таким чином реалізовано навігацію даного додатку.

Наступним кроком побудови макету дизайну було опис дизайну кнопок тестувань та відповідей. Вони виконані у сірому та чорному кольорах. Але це не прості кнопки. Вони зроблені через віджет Table View так, як текст запитань, тем тестувань та відповідей із поломками загружаються з бази даних Firebase Realtime Database. Це реалізовано через JSON запроси, які отримують усю інформацію. Обробляють її функції обробки даних та передають усю необхідну інформацію для виводу на екран. Це дуже корисно коли інформацію потрібно часто змінювати та переписувати. Таким чином ми можемо зекономити дуже багато часу.

3.6 CocoaPods

CocoaPods - це потужне і одночасно витончене засіб управління залежностями Cocoa-бібліотек, які розробники використовують в своїх iOS і MacOS X проектах[6].

Згідно з офіційним сайтом, CocoaPods - це кращий засіб з управління залежностями бібліотек в Objective-C проектах.

Замість скачування коду зі сховищ бібліотеки і копіювання в папку вашого проекту ми можемо надати можливість CocoaPods зробити все за нас.

Наприклад, нам необхідно зробити додаток, які буде спілкуватися з RESTfull API якогось сервісу, використовуючи JSON.

Найбільш популярна бібліотека для роботи з HTTP-запитами - це AFNetworking (з тих пір, як ASIHTTPRequest перестала підтримуватися).

Список найбільш популярних бібліотек доступних через CocoaPods: AFNetworking, ASIHTTPRequest, BlocksKit, ConciseKit, CorePlot, EGOTableViewPullRefresh, Facebook-iOS-SDK, JSONKit, MBProgressHUD, Nimbus, QuickDialog, Reachability, SFHFKeychainUtils, ShareKit

3.7 Система контролю версій Git

Git – розподілена система контролю версій файлів та спільної роботи. Цікавим фактом є те, що дана система була створена для контролю версій при розробці системного ядра Linux та підтримується до сьогодні. Git система контролю версій є однією із найпопулярніших та найбезпечніших в світі, що базується на розгалужені та злитті гілок. Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи. [8] Також можлива прив’язка цифрових підписів розробників до тегів та комітів.

Цю систему використовують компанії різного калібру, від найменших до світових брендів. Ця система використовується надає найбільш стабільну роботу та неймовірно зручна та пристосована для того, щоб користувачі мали усі необхідні можливості. Прикладами проєктів, що використовують Git, є ядро Linux, Android, LibreOffice, Cairo, GNU Core Utilities, Mesa 3D, Wine, багато проєктів з X.org, XMMS2, GStreamer, Debian DragonFly BSD, Perl, Eclipse, GNOME, KDE, Qt, Ruby on Rails, PostgreSQL, VideoLAN, PHP, One Laptop Per Child (OLPC), АБІС Koha, GNU LilyPond та ELinks і деякі дистрибутиви GNU/Linux.

Репозиторій (repository)[9]

Часто згадується як репо. Репозиторій – це колекція файлів і папок, які ви використовуєте для відстеження git. Сховище складається з усієї історії змін

вашої команди в проекті. Це той великий склад, на якому зберігається код, який додали ви і ваша команда.

Github[9]

Найпопулярніше віддалене сховище для git-репозиторіїв. Особливості: він дозволяє вам встановлювати права доступу до проектів, відстежувати і відправляти помилки, приймати запити на поліпшення, підписуватися на повідомлення сховища, використовувати графічний інтерфейс, а не командний рядок. Репо за замовчуванням відкриті, але платні акаунти можуть мати приватні репозиторії.

Commit [9]

Думайте про це як про збереження вашої роботи. Коли ви фіксуєте репозиторій, це схоже на те, що ви збираєте файли в тому вигляді, в якому вони існують в даний момент, і поміщаєте їх в капсулу часу. Фіксація буде існувати тільки на вашому локальному комп'ютері, поки вона не буде відправлена на віддалений репозиторій.

Push [9]

Фіксація поміщає ваші файли в капсулу часу, а відправка – це те, що запускає капсулу в космос. Відправлення – це, по суті, синхронізація ваших збережень (фіксацій, коммітів) з хмарою (знову ж таки, ймовірно, Github). Ви також можете використовувати кілька коммітів одночасно. Ви можете працювати в автономному режимі, зробити багато роботи, а потім передати все це на Github.

Branch [9]

Уявіть свій git-репо у вигляді дереві. Стовбур дерева, програмне забезпечення, яке запускається, називається майстер-гілкою (Master Branch). Це те, що є онлайн. Гілки цього дерева називаються, як не дивно, гілками. Це окремі екземпляри коду, який відрізняється від основної бази коду. Ви можете відгалузити одну функцію або експериментальний патч. Розгалужуючись, ви можете зберегти цілісність основного програмного забезпечення і мати можливість відкотитися, якщо зробите щось зовсім божевільне. Це також

дозволяє вам працювати над своїм завданням, не впливаючи на вашу команду (або вона на вас).

Merge [9]

Коли гілка виправлена, не містить помилок (наскільки ви принаймні можете судити) і готова стати частиною первинної бази коду, вона буде об'єднана з головною гілкою. Об'єднання – це те, на що це схоже: злиття двох гілок. Будь-який новий або оновлений код стане офіційною частиною кодової бази. Той, хто відгалужується від точки злиття, також буде мати цей код в своїй гілці.

Clone [9]

Клонування репо – це майже те ж саме, як і звучить. Воно бере весь онлайн-репозиторій і робить точну копію на вашому локальному комп'ютері. Вам потрібно буде зробити це по ряду причин, і не в останню чергу для збереження сумісності.

Fork [9]

Форкінг багато в чому схожий на клонування, тільки замість того, щоб робити копію існуючого репо на вашому локальному комп'ютері, ви отримуєте абсолютно новий репо цього коду під своїм власним ім'ям. Ця функція в основному використовується для взяття існуючої кодової бази і перехід з нею в абсолютно новому напрямку, що часто трапляється в програмному забезпеченні з відкритим вихідним кодом; розробники бачать базову ідею, яка працює, але хочуть піти іншим шляхом. Форкінг дозволяє цьому статися. Ви також можете взяти участь в репозиторії іншого розробника, як у своїй особистій пісочниці. І якщо ви робите щось, що, на вашу думку, може йому сподобатися, ви можете зробити попередній запит на об'єднання.

Pull Request [9]

Запит на підтвердження – це коли ви відправляєте запит з внесеними вами змінами (або в гілці, або в відгалуженні), які повинні бути перенесені (або об'єднані) в основну гілку (Master Branch) сховища. Це великий час, і тут відбувається диво. Якщо запит на підтвердження буде схвалений, ви офіційно

внесете свій внесок в програмне забезпечення, і Github завжди буде показувати, що саме ви зробили. Однак, якщо запит відхиляється з якої-небудь причини, ревізор зможе дати відгук про те, чому запит був відхилений і що ви можете зробити, щоб він був прийнятий.

В даній роботі систему контролю версій використано для контролювання процесу розробки мобільного додатку та зберігання резервної копії програмного коду у хмарі так як це найпотужніший сервіс контролю версій на сьогоднішній день.

3.8 Операційна система MacOS

MacOS – це операційна система розроблена корпорацією Apple для їх власних комп'ютерів таких як MacBook, Mac Mini, IMac. На даний момент актуальною версією цієї операційної системи є OS X. Вона значно відрізняється від попередніх. OS X включила безліч можливостей, які роблять її більш стабільною ніж попередні. Також OS X використовує витісняючу багатозадачність і захист пам'яті які дозволяють запускати декілька процесів, що не зможуть перервати або зашкодити один одному. На архітектуру OS X вплинув OPENSTEP, який був задуманий як операційна система що легко портується. Наприклад, NEXTSTEP була портована з 68k платформи NeXT комп'ютера, до того як він — NEXTSTEP — був куплений Apple. Так і OPENSTEP був портований на PowerPC в рамках проєкту Rhapsody[10].

OS X також включає середовище розробки програмного забезпечення Xcode[4], що дозволяє розробляти програми на декількох мовах включаючи C, C++, Objective-C, і Java. Вона підтримує компіляцію в так звані «універсальні програми» (Universal Binary), які можуть запускатися на декількох платформах (x86, PowerPC), так само, як «fat binaries» використовувалися для запуску однієї програми на як на 68k, так і на PowerPC платформах.

Основи операційної системи:

- Підсистема з відкритим кодом

- Середовище програмування Core Foundation (Carbon API, Cocoa API і Java API)
- Графічне середовище Aqua
- Технології CoreImage, CoreAudio і CoreData.

Весь процес розробки було проведено на комп'ютері MacBook Pro 13 із операційною системою MacOS X (Big Sur).

3.9 Алгоритмізація та розробка функціоналу

В даній системі було реалізовано алгоритм Електра в звичайному його вигляді. Суть алгоритму Елкетра[13] в тому, що спочатку є певна кількість несправностей із коефіцієнтами 0. В кожному запитанні на яке дана відповідь, що не задовільняє систему, додається певний коефіцієнт до кожної із відповідей. Після того, як усі коефіцієнти просумувались ми визначаємо найбільший із них і виводимо його на екран користувачеві. Якщо ж в кінці діагностики виявляється, що ми маємо декілька однакових найбільших коефіцієнтів, виводяться усі ймовірні відповіді та в протилежному випадку якщо при перевірці коефіцієнтів виявилось, що вони усі нульові ми виводимо, що система не може виявити несправність і вам потрібна допомога механіка та візуальна діагностика авто.

Таблиця 3.1 Коефіцієнти та діагностичні питання

Питання	**K1**	**K2**	**K3**	**K4**	**K5**
1. Чи горить лампочка аккамулятора?	0,3	0,4	0,1	0,0	0,0
2. Крутить стартер?	0,0	0,2	0,3	0,1	0,0
3. Чи реагує авто на зажигання?	0,2	0,2	0,3	0,1	0,4
4. Чи доходить паливо до форсунок?	0,3	0,0	0,1	0,5	0,1
5. Чи підключено бензонасос правильно?	0,1	0,1	0,0	0,1	0,2
6. Чи стояла колись нештатна сигналізація?	0,1	0,0	0,2	0,1	0,3
7. Чи горить лампочка Check Engine?	0,0	0,1	0,0	0,1	0,0

Усі несправності та коефіцієнти було обговорено із людьми, що працюють в цій сфері та являються безпосередньо майстрами на автосервісі.

3.10 Бази даних Firebase

Firebase – це платформи розробки мобільних та веб- додатків[5]. Firebase надає в режимі реального часу базу даних та бекенд як службу. Ця служба надає розробникам застосунків API, який дозволяє синхронізувати дані застосунків між клієнтами та зберігати їх у хмарі Firebase. Компанія також надає клієнтські бібліотеки, які дозволяють інтеграцію із застосунками Android, iOS, JavaScript / Node.js, Java, Objective-C, Swift. База даних також доступна через REST API та прив'язки до декількох сценаріїв JavaScript, таких як AngularJS, React, Ember.js та Backbone.js. REST API використовує протокол подій із сервером, який є інтерфейсом для створення HTTP-з'єднань для отримання push-повідомлень від сервера. Розробники, які використовують Realtime Database, можуть захищати свої дані за допомогою правил безпеки, що застосовуються на сервері.

Із великого списку сервісів, що відносяться до Firebase я обрав Firebase Realtime Database. В даній роботі було використано цей сервіс для збереження бази даних із питаннями тестування, відповідями, назвами тем тестування та коефіцієнтами. Дані виглядають таким чином у Firebase(рис. 3.8).

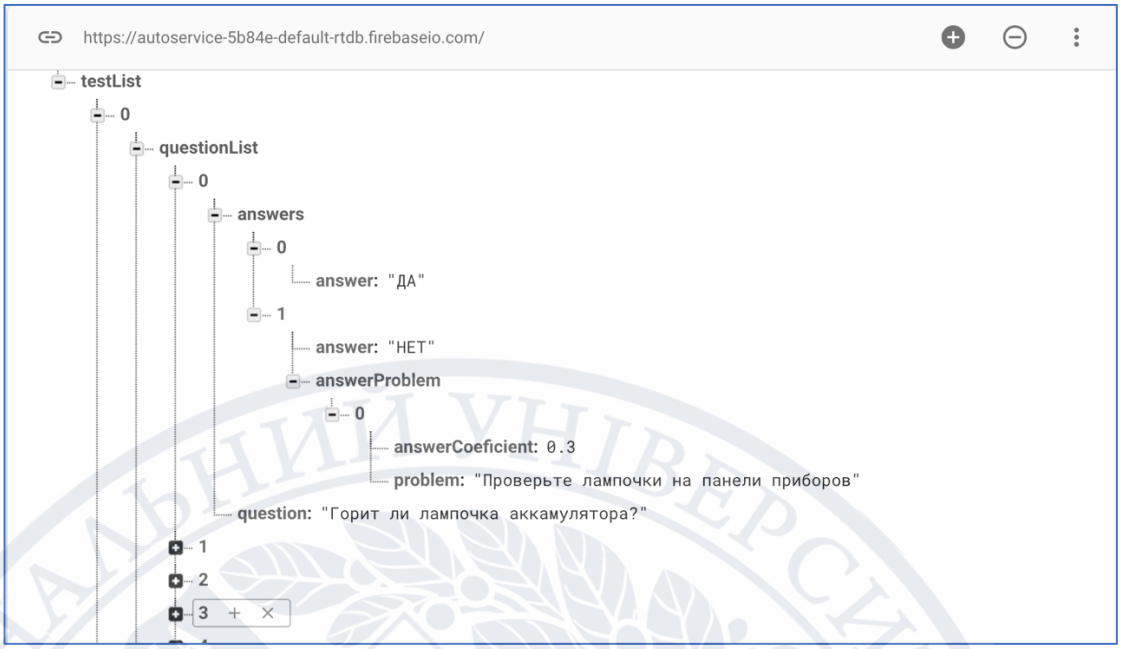

Рисунок 3.8. База даних Firebase

Серед ключових особливостей Cloud Firestore:

- Документи та колекції з потужним запитом
- SDK для iOS, Android та Web з автономним доступом до даних
- Синхронізація даних у режимі реального часу
- Автоматична багаторегіональна реплікація даних з високою послідовністю
- Пакети SDK для вузлів, Python, Go та Java

Цей інструмент дає змогу також імпортувати та експортувати базу даних. Наприклад, якщо вам потрібно швидко змінити текст запитань, виправити помилки чи ще щось змінити в базі даних, то набагато легше буде експортувати її на свій комп’ютер. В результаті буде завантажено JSON файл із базою даних. Після цього відредагувати цей файл так як потрібно та зробити експорт цього файлу. Після цього всі зміни будуть внесені в базу даних на сервері.

3.11 JSON

JSON — це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дає змогу

описувати об'єкти та інші структури даних. Цей формат використовується переважно для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією)[7].

JSON знайшов своє головне призначення в написанні веб-програм, а саме при використанні технології AJAX. JSON, що використовується в AJAX, виступає як заміна XML (використовується в AJAX) під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти.

В даній роботі цей формат даних використано для оформлення бази даних, вигляд якої видно з малюнку(рис. 6). Більш детально оглянувши базу можемо зрозуміти, що в ній присутній масив “testList” в якому ми зберігаєм об’єкти із тестуваннями. У кожному об’єкті із тестуванням мають бути присутні “questionList” та “testName”. Із назв можна зрозуміти, що це список запитань та назва тестування відповідно. Назву тестування зберігаємо із типом string, а список запитань це масив із запитаннями “question” та об’єктами відповідей “answers” відповідно кожному запитанню. Відповідей існує два види: «ТАК» і «НІ». Одна із них є позитивною та одна негативною. Позитивна відповідь не має ніяких додаткових полів, а негативна навпаки містить в собі текст можливої несправності та коефіцієнт несправності. Саме таким чином побудована уся база даних.

JSON будується на таких структурах:

- Набір пар назва та значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список із ключем або асоціативним масивом.
- Впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список або послідовність.

Це універсальні структури даних. Теоретично всі сучасні мови програмування підтримують їх у тій чи іншій формі. Оскільки JSON

використовується для обміну даними між різними мовами програмування, то є сенс будувати його на цих структурах.

У JSON використовуються такі їхні форми:

- **Об'єкт** — це послідовність пар назва/значення. Об'єкт починається з символу “{“ і закінчується символом “}” . Кожне значення слідує за “:” і пари назва/значення відділяються комами.
- **Масив** — це послідовність значень. Масив починається символом “[“ і закінчується символом “]”. Значення відділяються комами.
- **Значення** може бути рядком в подвійних лапках, або числом, або логічними “true” чи “false”, або “null”, або об'єктом, або масивом. Ці структури можуть бути вкладені одна в одну.
- **Рядок** — це послідовність з нуля або більше символів юнікода, обмежена подвійними лапками, з використанням escape-послідовностей, що починаються зі зворотної косої риски “\”. Символи представляються простим рядком.

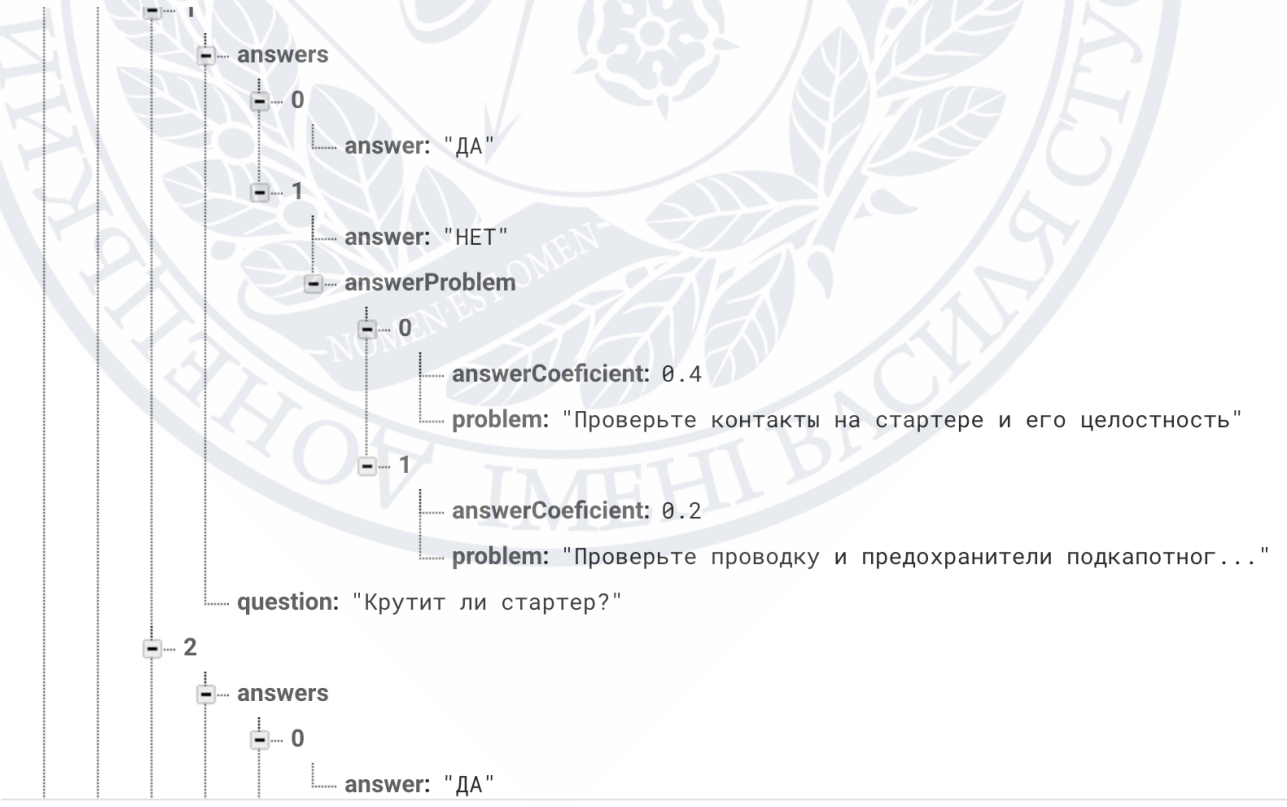

Рисунок 3.9. База даних у форматі JSON

Безпека

Хоча JSON призначений для передачі даних в серіалізованому вигляді, його синтаксис відповідає синтаксису JavaScript і це створює низку проблем безпеки. Часто для обробки даних, отриманих від зовнішнього джерела у форматі JSON, до них застосовується функція **eval()** без якої-небудь попередньої перевірки.[7]

3.12 Розробка бази даних

Для самого початку було проведено дослідницьку роботу для того, щоб розумітися краще на галузі та її специфіці. В ході ознайомлення було проведено декілька бесід із працівниками сервісних центрів та вони надали статистику несправностей та її симптомів. Таким чином було зібрано достатньо інформації для того, щоб розрахувати приблизні коефіцієнти для бази даних.

У додатку Firebase існує функція імпорту бази даних, тому спершу було створено файл із розширенням .json. В цьому файлі описано базу даних в текстовому вигляді та імпортовано у сервіс Firebase Realtime Database. Як виглядає база даних видно з малюнку (рис. 7). Цей процес не займає багато часу, але потребує чіткого розуміння, яку архітектуру вам потрібно розробити.

3.13 Тестування додатку

Для досягнення стабільної роботи додатку було проведено тестування даного сервісу через Simulator вбудований в XCode. Цей симулятор дозволяє запустити на комп’ютері візуальну копію будь-якого девайсу Apple. Тестування проводилось із використанням симуляції IPhone 11 який є відносно універсальним так як його діагональ екрану однакова із рядом інших пристроїв та він є досить актуальним й на сьогоднішній день. Діагональ IPhone 11 сягає 6.1 дюймів та має сумісність із наступним рядом девайсів: IPhone XR, IPhone 12, IPhone 12 Pro, IPhone 11 Pro. Під час тестування було виявлено декілька недосконалостей, які успішно було виправлено. Нормальну роботу додатку було зафіксовано на відеозапис та вивантажено у репозиторій на GitHub. Завантажити відео можливо перейшовши по QR коду.(рис. 3.10)

Рисунок 3.10. Відео із роботою програми

Прикладом такої несправності яку було виявлено можна привести проблема із активністю кнопка отримання результату тестування. До того вона не мала статусів активна чи ні, вона завжди була клікабельна, але це не правильно адже для початку потрібно отримати відповіді на усі поставлені запитання. Шляхів вирішення цієї проблеми було знайдено 2.

Перший шлях вирішення від якого згодом відмовились – це шлях за яким усі запитання на які не було отримано відповідь автоматично зараховувати із 0 коефіцієнтом, але це не зовсім точно так як користувач міг банально не помітити, що він не відповів на одне запитання, а результат вже не буде коректним.

Наступним шляхом вирішення цієї проблеми було обрано додати до цієї кнопки статуси та залежність. Статус може бути чи «активна» чи «не активна». Коли користувач відповів на усі запитання кнопка набуває статус «активна» та стає клікабельна. Коли ж користувач не відповів хоча б на одне питання кнопка зберігає статус «не активна» та є не клікабельною.

Ви також можете приєднатися до тестування даного сервіси перейшовши по QR коду до репозиторію на GitHub та встановивши цей додаток собі на пристрій.

Рисунок 3.11. QR код на репозиторій

Висновки розділу 3

В даному розділі було описано весь інструментарій, що було використано для розробки даного додатку. Детально описані характеристики та функціонал кожного інструменту та усі переваги поміж конкурентів, де це було потрібно. Також було описано чому саме ці технології було обрано. Також в цьому розділі було описано весь процес розробки.

Дизайн є мінімалістичним та зрозумілим, що надасть переваги поміж інших додатків із схожою тематикою, а функціонал розроблено так, щоб користувач отримав максимально коректний результат роботи програми.

ВИСНОВКИ

З усього вище сказаного можна зробити невеликий підсумок по декільком пунктам.

Перший пункт – це те, що для розробки мобільних додатків для Apple девайсів підходить лише комп'ютер із операційною системою MacOS.

Другий пункт стосовно завантаження даних. Найбільш зручний спосіб завантаження, передачі та оновлення даних – це база даних зроблена у Firebase Realtime Database із типом даних JSON.

Третій пункт стосовно дизайну та макету додатку. Він повинен мати мінімалістичний дизайн із мінімальною кількістю лише вкрай необхідної інформації для зручності користувачів. Це приваблює користувачів, так як вони одразу розуміють, що це, для чого цей додаток та як ним користуватись.

Четвертий пункт стосовно алгоритму. Було обрано найпростіший алгоритм за методом Електра(метод коефіцієнтів).[13] Це надає швидкості роботи програми та не потребує зайвого часу на підрахунки, а також позбавляє розробника від зайвої роботи у вигляді опису складних математичних формул та залежностей.

СПИСОК ДЖЕРЕЛ

1. Swift Wiki. URL: https://uk.wikipedia.org/wiki/SWIFT (дата звернення: 11.01.2021)

2. Swift Storyboard. URL: https://swiftbook.ru/post/tutorials/swift-storyboards-v-xcode-63-chast-1/ (дата звернення: 11.01.2021)

3. Swift UIKit documentation. URL: https://developer.apple.com/documentation/uikit/ (дата звернення: 12.01.2021)

4. XCode Wiki. URL: https://uk.wikipedia.org/wiki/Xcode (дата звернення: 14.01.2021)

5. Firebase Wiki. URL: https://uk.wikipedia.org/wiki/Firebase (дата звернення: 20.02.2021)

6. Cocoapods documentation. URL: https://habr.com/ru/company/luxoft/blog/149631/ (дата звернення: 23.01.2021)

7. JSON Wiki, URL: https://uk.wikipedia.org/wiki/JSON (дата звернення: 14.01.20021)

8. Git Wiki. URL:]https://uk.wikipedia.org/wiki/Git (дата звернення: 24.03.2021)

9. Особливості Git. URL: https://sebweo.com/scho-take-git-ta-github-kerivnitstvo-dlya-pochatkivtsiv/ (дата звернення: 22.02.2021)

10. MacOS Wiki. URL: https://uk.wikipedia.org/wiki/MacOS (дата звернення: 13.03.2021)

11. IOS Wiki. URL: https://uk.wikipedia.org/wiki/IOS (дата звернення: 20.04.2021)

12. Особливості IOS. URL: https://mac-win.ru/osobennosti-operatsionnoj-sistemy-ios/ (дата звернення: 05.04.2021)

13. Метод Електра. URL: https://www.semestr.ru/ks880 (дата звернення: 11.01.2021)

14. Apple. URL: https://www.apple.com/ua/ (дата звернення: 20.03.2021)

15. Дизайн та макет. URL: https://dan-it.com.ua/uk/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/ (дата звернення: 11.05.2021)

16. Swiftbook. URL: https://swiftbook.ru/ (дата звернення: 12.05.2021)

17. Варламова С.А., Федосеева К.А., Варламова Методы и средства поддержки принятия решений водителя автомобиля по ограничению скоростного режима: Том 18, № 4 (2018) https://vestnik.susu.ru/ctcr/article/view/8071 (дата звернення: 13.04.2021)

18. В.М. Ерёмин, А.О. Аристов, Комп'ютені системи прийняття рішень по управлінню транспортним засобом на автомобільних дорогах (дата звернення: 12.05.2021)

19. Експертні системи. URL: http://moodle.ipo.kpi.ua/moodle/file.php/791/lekcija5_14.pdf (дата звернення: 21.03.2021)

20. MYCIN. URL: https://uk.wikipedia.org/wiki/Mycin (дата звернення: 21.04.2021)

21. Audrey Tam & Caroline Begbie, SwiftUI Apprentice (1st Edition) From: Razeware LLC 2021, 706 с (дата звернення: 11.02.2021)

22. Ehab Amer, Marin Bencevic, Ray Fix, Expert Swift (1st Edition) From: Razeware LLC 2021, 405 с (дата звернення: 12.05.2021)

23. Adam Aspin, Pro Power BI Theme Creation: JSON Stylesheets for Automated Dashboard Formatting From: Apress 2020, 308 с (дата звернення: 12.05.2021)

24. Alexander Aronowitz, Swift Programming: The Ultimate Beginner's Guide to Learn swift Programming Step by Step, 3nd Edition From: NLN Inc 2019, 364 с

25. Antonio Bello, Bill Morefield, Audrey Tam, SwiftUI by Tutorials (3rd Edition) From: Razeware LLC 2021, 335 с (дата звернення: 20.03.2021)

26. Wei-Meng Lee, SwiftUI For Dummies From: For Dummies 2019, 419 с (дата звернення: 20.03.2021)

27. Matt Neuburg, Programming iOS 14: Dive Deep into Views, View Controllers, and Frameworks, Eleventh Edition From O'Reilly Media, Inc. 2020, 1259 с (дата звернення: 20.03.2021)

28. Hem Dutt, Interprocess Communication with macOS: Apple IPC Methods From: Apress 2021, 295 с (дата звернення: 20.03.2021)

29. Mark L. Chambers, MacBook For Dummies, Ninth Edition From: For Dummies 2021, 434 с (дата звернення: 20.03.2021)

30. Thomas Valentine, Database-Driven Web Development: Learn to Operate at a Professional Level with PERL and MySQL From: Apress 2021, 206 с (дата звернення: 20.03.2021)

31. Chengqing Zong, Rui Xia, Text Data Mining From: Springer, Tsinghua University Press 2021, 363 с (дата звернення: 21.04.2021)

32. Dhiman Deb Chowdhury, NextGen Network Synchronization From: Springer 2021, 279 с (дата звернення: 21.04.2021)

33. Mike Unwin, RSPB Spotlight Swifts and Swallows From: Bloomsbury Wildlife 2018, 198 с (дата звернення: 21.04.2021)

34. Jonathon Manning, Paris Buttfield-Addison, iOS Swift Game Development Cookbook: Simple Solutions for Game Development Problems, 3rd Edition From: O'Reilly Media 2018, 350 с (дата звернення: 21.04.2021)

35. Greg Lim, Beginning SwiftUI: updated to SwiftUI 2.0 and iOS 14 From: Independetly Published 2021, 156 с (дата звернення: 21.04.2021)

36. Swift Book. URL: https://swiftbook.ru/ (дата звернення: 07.03.2021)

37. Swift Docs. URL: https://swift.org/documentation/ (дата звернення: 05.03.2021)

38. JSON examples. URL: https://json.org/example.html (дата звернення: 06.03.2021)

39. CocoaPods. URL: https://cocoapods.org/ (дата звернення: 07.03.2021)

40. GitBook. URL: https://git-scm.com/doc (дата звернення: 07.03.2021)

41. An Introduction to Computational Systems Biology: Systems-Level Modelling of Cellular Networks, Karthik Raman From: Chapman and Hall/CRC, 2021, 359 с (дата звернення: 10.03.2021)

42. Remotely Possible: Strategic Lessons and Tactical Best Practices for Remote Work, Shawn Belling, From: Apress, 2021, 138 с (дата звернення: 11.03.2021)

43. The Road to React with Firebase: Your journey to master advanced React for business web applications, Robin Wieruch, From: Lean Publishing, 2019, 202 с (дата звернення: 07.03.2021)

44. iOS 11 & Swift 4 for Beginners (1st Edition), Fahim Farook, Matt Galloway, Eli Ganim, From: Razeware LLC, 2017, 706 с (дата звернення: 25.01.2021)

ДОДАТОК А
КОД

TestViewController.swift

```
import UIKit

final class TestViewController: UIViewController {

    @IBOutlet private var tableView: UITableView!
    @IBOutlet private var acceptButton: UIButton!

    let headerFont:UIFont = UIFont.boldSystemFont(ofSize: 17)
    var questionsList: [QuestionList] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        setupTableView()
        setupBottomButton()
    }

    private func setupTableView() {
        tableView.delegate = self
        tableView.dataSource = self

        tableView.register(AnswerTableViewCell.self)
    }

    private func configure(cell: AnswerTableViewCell, indexPath: IndexPath) {
        let answer = questionsList[indexPath.section].answers[indexPath.row]
        cell.display(title: answer.answer.rawValue)
        cell.accessoryType = (questionsList[indexPath.section].selectedIndex ==
indexPath.row) ? .checkmark : .none
    }

    private func heightOfHeaderText(text:String) -> CGFloat {
        return NSString(string: text).boundingRect (
            with: CGSize(width: self.tableView.frame.size.width, height: 999),
            options: NSStringDrawingOptions.usesLineFragmentOrigin,
            attributes: [NSAttributedString.Key.font : headerFont],
            context: nil).size.height
    }

    private func setupBottomButton() {
```

```
        if isFinishTest() {
            acceptButton.backgroundColor = .black
            acceptButton.isEnabled = true
        } else {
            acceptButton.backgroundColor = .lightGray
            acceptButton.isEnabled = false
        }
    }

    private func generateResult() -> String {
        let problems = questionsList.compactMap { question -> [AnswerProblem]? in
            if question.selectedIndex == 1 {
                return question.answers.last?.answerProblem ?? nil
            }
            return nil
        }.flatMap { $0 }
        let errorList = getProblemWithMaxCoeficient(list: checkDuplicate(list:
problems))

        var errorText: String = ""
        errorList.forEach { error in
            errorText.append("\(error.problem)\n")
        }

        if errorText.isEmpty {
            return "Нужен визит на СТО"
        }

        return String(errorText.dropLast())
    }

    private func checkDuplicate(list : [AnswerProblem]) -> [AnswerProblem] {
        guard let firstItem = list.first else { return [] }
        var errorList: [AnswerProblem] = [firstItem]
        list.forEach { answer in
            var isSingle: Bool = true
            errorList.enumerated().forEach { index, item in
                if answer.problem == item.problem {
                    errorList[index].answerCoeficient += answer.answerCoeficient
                    isSingle = false
                }
            }
            if isSingle {
                errorList.append(answer)
            }
```

```
        }
        return errorList
    }

    private func getProblemWithMaxCoeficient(list: [AnswerProblem]) ->
[AnswerProblem] {
        let maxCoeficient = list.max { one, two -> Bool in
            return one.answerCoeficient < two.answerCoeficient
        }
        guard let max = maxCoeficient?.answerCoeficient else { return [] }
        let result = list.compactMap { problem in
            return (Double(round(1000*problem.answerCoeficient)/1000) ==
Double(round(1000*max)/1000)) ? problem : nil
        }

        return result
    }

    private func isFinishTest() -> Bool {
        let list = questionsList.map { $0.selectedIndex == nil }
        return !list.contains(true)
    }

    private func showAlert() {
        let message = generateResult()
        let alert = UIAlertController(title: "Результат", message: message,
preferredStyle: .alert)

        let okAction = UIAlertAction(title: "OK", style: .cancel) { _ in
            self.navigationController?.popToRootViewController(animated: true)
        }

        alert.addAction(okAction)

        self.present(alert, animated: true, completion: nil)
    }

    @IBAction private func doneAction(_ sender: UIButton) {
        showAlert()
    }
}

extension TestViewController: UITableViewDelegate, UITableViewDataSource {
    func numberOfSections(in tableView: UITableView) -> Int {
        return questionsList.count
```

```
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int)
-> Int {
        return questionsList[section].answers.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {
        let cell = tableView.dequeue(AnswerTableViewCell.self, indexPath)
        self.configure(cell: cell, indexPath: indexPath)
        return cell
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        questionsList[indexPath.section].selectedIndex = indexPath.row
        setupBottomButton()
        self.tableView.reloadData()
    }

    func tableView(_ tableView: UITableView, heightForHeaderInSection section:
Int) -> CGFloat {
        return heightOfHeaderText(text: questionsList[section].question)
    }

    func tableView(_ tableView: UITableView, viewForHeaderInSection section: Int)
-> UIView? {
        let headerLabel: UILabel = UILabel()
        headerLabel.frame.origin = .init(x: 16, y: 0)
        headerLabel.numberOfLines = 0
        headerLabel.lineBreakMode = NSLineBreakMode.byWordWrapping
        headerLabel.font = headerFont
        headerLabel.text = questionsList[section].question
        headerLabel.backgroundColor = .lightGray
        return headerLabel
    }
}
```

TestEntity.swift

```
import Foundation

struct TestsEntity: Codable {
    let testList: [TestList]
}
```

```
// MARK: - TestList
struct TestList: Codable {
    let testName: String
    let questionList: [QuestionList]
}

// MARK: - QuestionList
struct QuestionList: Codable {
    var answers: [AnswerElement]
    let question: String
    var selectedIndex: Int?
}

// MARK: - AnswerElement
struct AnswerElement: Codable {
    let answer: AnswerEnum
    let answerProblem: [AnswerProblem]?
}

enum AnswerEnum: String, Codable {
    case да = "ДА"
    case нет = "НЕТ"
}

// MARK: - AnswerProblem
struct AnswerProblem: Codable, Hashable {
    let problem: String
    var answerCoeficient: Double
}
```

AppDelegate.swift

```
import UIKit
import Firebase

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        FirebaseApp.configure()
```

```
        return true
```

Extensions.swift

```
import UIKit

extension UIStoryboard {
    convenience init(storyboard: String) {
        self.init(name: storyboard, bundle: nil)
    }

    func instantiateViewController<T: UIViewController>(_ type: T.Type) -> T {
        let id = NSStringFromClass(T.self).components(separatedBy: ".").last!
        return self.instantiateViewController(withIdentifier: id) as! T
    }
}

extension UIViewController {
    class func instance(_ storyboard: String) -> Self {

        let storyboard = UIStoryboard(storyboard: storyboard)
        let viewController = storyboard.instantiateViewController(self)

        return viewController
    }
}

extension UIView {
    class var identifier: String {
        return String(describing: self)
    }
}

extension UITableView {

    func dequeue<T: UITableViewCell>(_ cell: T.Type, _ indexPath: IndexPath) -> T
{
        return self.dequeueReusableCell(withIdentifier: cell.identifier, for: indexPath)
as! T
    }

    func register<T: UITableViewCell>(_ cell: T.Type) {
        self.register(UINib(nibName: T.identifier, bundle: nil), forCellReuseIdentifier:
T.identifier)
```

```
    }
}
```

HomeTableView.swift

```
import UIKit

protocol HomeTableViewCellProtocol {
    func display(title: String)
}

final class HomeTableViewCell: UITableViewCell {

    @IBOutlet private var titleLabel: UILabel!

    override func layoutSubviews() {
        super.layoutSubviews()
        titleLabel.layer.borderColor = UIColor.black.cgColor
        titleLabel.layer.borderWidth = 1.0
        titleLabel.layer.cornerRadius = 10
    }
}

extension HomeTableViewCell: HomeTableViewCellProtocol {
    func display(title: String) {
        self.titleLabel.text = title
    }
}
```

HomeViewController.swift

```
//
//  HomeViewController.swift
//  AutoService
//
//  Created by Sasha Voloshanov on 18.05.2021.
//

import UIKit

final class HomeViewController: UIViewController {

    @IBOutlet private var tableView: UITableView!

    private let manager: FirebaseManagerProtocol = FirebaseManager()
    private var list: [TestList] = []

    override func viewDidLoad() {
```

```
        super.viewDidLoad()
        setupTableView()
        getTests()
    }

    private func getTests() {
        manager.getTests { [weak self] result in
            guard let self = self else { return }
            switch result {
            case .success(let list):
                DispatchQueue.main.async {
                    self.list = list.testList
                    self.tableView.reloadData()
                }
            case .failure(let error):
                print(error.localizedDescription)
            }
        }
    }

    private func setupTableView() {
        tableView.delegate = self
        tableView.dataSource = self

        tableView.register(HomeTableViewCell.self)
    }

    private func configure(cell: HomeTableViewCell, item: Int) {
        let test = list[item]
        cell.display(title: test.testName)
    }
}

extension HomeViewController: UITableViewDelegate, UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int)
-> Int {
        return list.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {
        let cell = tableView.dequeue(HomeTableViewCell.self, indexPath)
        self.configure(cell: cell, item: indexPath.row)
        return cell
    }
```

```
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        self.tableView.deselectRow(at: indexPath, animated: true)
        let item = list[indexPath.row]
        let vc = TestViewController.instance("Main")
        vc.questionsList = item.questionList
        self.navigationController?.pushViewController(vc, animated: true)
    }
}
```

Firebase.swift

```
import FirebaseDatabase
import Firebase

protocol FirebaseManagerProtocol: class {
    func getTests(completion: @escaping (Result<TestsEntity, Error>) -> Void)
}

final class FirebaseManager: FirebaseManagerProtocol {

    private var ref: DatabaseReference!

    func getTests(completion: @escaping (Result<TestsEntity, Error>) -> Void) {
        ref = Database.database().reference()

        ref.getData(completion: { error, snapshot in
            if snapshot.exists(),
                let json = snapshot.value as? NSDictionary,
                let data = try? JSONSerialization.data(withJSONObject: json, options: []),
                let model = try? JSONDecoder().decode(TestsEntity.self, from: data) {

                completion(.success(model))
            } else if let error = error {
                completion(.failure(error))
            }
        })
    }
}
```

JSON Database

```
{
  {
 "testList" : [ {
```

```
  "questionList" : [ {
    "answers" : [ {
      "answer" : "ДА"
    }, {
      "answer" : "НЕТ",
      "answerProblem" : [ {
        "answerCoeficient" : 0.3,
        "problem" : "Проверьте лампочки на панели приборов"
      } ]
    } ],
    "question" : "Горит ли лампочка аккамулятора?"
  }, {
    "answers" : [ {
      "answer" : "ДА"
    }, {
      "answer" : "НЕТ",
      "answerProblem" : [ {
        "answerCoeficient" : 0.4,
        "problem" : "Проверьте контакты на стартере и его целостность"
      }, {
        "answerCoeficient" : 0.2,
        "problem" : "Проверьте проводку и предохранители подкапотного
пространства"
      } ]
    } ],
    "question" : "Крутит ли стартер?"
  }, {
    "answers" : [ {
      "answer" : "ДА"
    }, {
      "answer" : "НЕТ",
      "answerProblem" : [ {
        "answerCoeficient" : 0.2,
        "problem" : "Проверьте контакты на стартере и его целостность"
      }, {
        "answerCoeficient" : 0.3,
        "problem" : "Проверьте проводку и предохранители подкапотного
пространства"
      } ]
    } ],
    "question" : "Есть ли искра в свечах зажигания?"
  }, {
    "answers" : [ {
      "answer" : "ДА"
    }, {
```

```
      "answer" : "НЕТ",
      "answerProblem" : [ {
        "answerCoeficient" : 0.3,
        "problem" : "Залейте топливо в бак и попробуйте заново"
      } ]
    } ],
    "question" : "Присуцтвует ли топливо в баке?"
  }, {
    "answers" : [ {
      "answer" : "ДА"
    }, {
      "answer" : "НЕТ",
      "answerProblem" : [ {
        "answerCoeficient" : 0.3,
        "problem" : "Проверьте топливную магистраль (Требует визита на СТО)"
      } ]
    } ],
    "question" : "Доходит ли топливо до форсунок двигателя?"
  } ],
  "testName" : "Тест 1: Диагностика подкапотного пространства."
}, {
  "questionList" : [ {
    "answers" : [ {
      "answer" : "НЕТ"
    }, {
      "answer" : "ДА",
      "answerProblem" : [ {
        "answerCoeficient" : 0.4,
        "problem" : "Замените или долейте масло в коробку передач"
      } ]
    } ],
    "question" : "При включении положения D происходит пинок?"
  }, {
    "answers" : [ {
      "answer" : "НЕТ"
    }, {
      "answer" : "ДА",
      "answerProblem" : [ {
        "answerCoeficient" : 0.1,
        "problem" : "Замените или долейте масло в коробку передач"
      }, {
        "answerCoeficient" : 0.2,
        "problem" : "Посмотрите на трос акселератора который идет к АКПП под капотом и поправьте его при необходимости"
      } ]
```

```
    } ],
    "question" : "При переключении передачи обороты завышены? (3000-3500 и более)"
   }, {
    "answers" : [ {
     "answer" : "НЕТ"
    }, {
     "answer" : "ДА",
     "answerProblem" : [ {
      "answerCoeficient" : 0.4,
      "problem" : "Посмотрите на трос акселератора который идет к АКПП под капотом и поправьте его при необходимости"
     }, {
      "answerCoeficient" : 0.1,
      "problem" : "Удостоверьтесь в целостности термостата контура охлаждения АКПП»"
     } ]
    } ],
    "question" : "Реакция на педаль акселератора хуже чем всегда?"
   }, {
    "answers" : [ {
     "answer" : "НЕТ"
    }, {
     "answer" : "ДА",
     "answerProblem" : [ {
      "answerCoeficient" : 0.2,
      "problem" : "Замените или долейте масло в коробку передач"
     }, {
      "answerCoeficient" : 0.3,
      "problem" : "Удостоверьтесь в целостности термостата контура охлаждения АКПП»"
     } ]
    } ],
    "question" : "емпература коробки передач выше 90 градусов в режиме езды?"
   }, {
    "answers" : [ {
     "answer" : "НЕТ"
    }, {
     "answer" : "ДА",
     "answerProblem" : [ {
      "answerCoeficient" : 0.1,
      "problem" : "Замените или долейте масло в коробку передач"
     } ]
    } ],
```

"question" : "Слышны ли характерные звуки метала при переключении положений ручки АКПП?"
}],
"testName" : "Тест 2: Диагностика автоматической коробки передач"
}]
} }