МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІКТ

ВИПУСКНА РОБОТА

на тему:

«Особистий менеджер повсякденних подій на iOS»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Шаповалов С. П.**

Студент гр. ІН-61 **Левкутник Д. О.**

Суми 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІКТ

Затверджую_____

Зав. кафедрою Довбиш А.С.

“_____” _____2020 р.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи ІН-61 спеціальності “Інформатика” денної форми навчання Левкутника Дмитра Олеговича.

Тема: “ **Особистий менеджер повсякденних подій** ”

Затверджена наказом по СумДУ
№_____ від _____2020 р.

Зміст пояснювальної записки:1)огляд існуючих рішень; 2)постановка завдання й формування завдань дослідження;3)проектування та розробка мобільного додатку;4) програмна реалізація та її опис;5)висновки.

Дата видачі завдання “_____”_____2020 р.
Керівник випускної роботи_____ Шаповалов С.П.
Завдання прийняв до виконання _____ Левкутник Д. О.

РЕФЕРАТ

Записка: 56 стор., 35 рис., 6 табл., 1 додаток, 4 джерела.

Об’єкт дослідження —iOS додаток на мові Swift.

Мета роботи — придбання теоретичних знань і практичних умінь з розробки мобільних додатків під iOS на мові Swift, а також придбання навичок з використання популярних фреймвоків, таких як UIKit, CoreData, UserNotifications та CloudKit.

Методи дослідження — технології створення iOS додатків.

Результати — розроблено мобільний додаток під систему iOS на мові Swift. Додаток має сучасний дизайн та зрозумілий інтерфейс. Під час написання коду було використано патерн проектування MVVM(Model-View-ViewModel), це дозволило написати зручний для масштабування та легкий для розуміння код, який легко буде підтримувати у майбутньому. У ході тестування проблем не виявлено.

IOS, SWIFT, МОБІЛЬНИЙ ДОДАТОК, ICLOUD, МЕНЕДЖЕР, APPLE, IPHONE.

ЗМІСТ

ВСТУП

На сьогоднішній день смартфон є одним з найпопулярніших девайсів у світі. Загалом, у цивілізованих країнах майже кожна людина носить у своїй кишені смартфон. Технології не стоять на місці, тож такі всесвітньо-відомі компанії як Samsung та Apple оновлюють лінійки смартфонів кожного року. Тому з кожною новою моделлю ці девайси стають швидшими та здобувають новий функціонал. Таке широке розповсюдження смартфонів дозволило створити новий ринок – ринок мобільних додатків. На кінець 2017 року у App Store(магазин мобільних додатків компанії Apple) опубліковано 2,1 млн додатків. Мінімальна ціна мобільного додатку у App Store складає $1, тож це дає величезні можливості для творчості окремих розробників, адже з'являється можливість працювати лише на себе. Ліцензія розробника у Apple коштує $100, тож кожний бажаючий має змогу публікувати свої програми у магазин App Store.

Реліз мови Swift відбувся у 2014 році. Мова призначена для написання коду на iOS та macOS та працює з фреймворками Cocoa та Cocoa Touch. Swift має змогу працювати з кодом на C та Objective-C у рамках одного й того ж проекту. На момент написання дипломної роботи опублікована версія 5.2.2 цієї мови.

iCloud – хмарне сховище даних від компанії Apple. Сервіс дозволяє безкоштовно розміщувати дані користувачів у хмарі. Це дозволяє зберігати дані при переході на новий девайс та забезпечує синхронізацію даних на всіх пристроях користувача, де встановлено додаток. Сервіс дозволяє створювати резервні копії iOS пристроїв.

Патерн Model-View-ViewModel – шаблон проектування під час проектування архітектури програми. MVVM полегшує відокремлення розробки графічного інтерфейсу від бізнес логіки. Модель представлення є

частиною, яка зв’язує інтерфейс користувача за бєк-єндом і оброблює більшість логіки відображення даних. Це дозволяє змінювати бєк-єнд та дизайн програми незалежно один від одного, у цьому й полягає перевага моделі, на відміну від MVC.

Темою випусної роботи є “Особистий менеджер повсякденних подій”, призначення якого організувати та полегшити повсякденне життя користувача. Функції додатку повинні стати у пригоді, а комфортний інтерфейс та компактність усього необхідного для користувача у одній програмі дозволить додатку довше пробути встановленим на девайсі користувача.

Для реалізації проекту було пройдено довгий шлях у оволодінні навичок iOS розробника, була вивчена мова Swift та основні інструменти написання коду та створенні інтерфейсу у середі XCode.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Вимоги до розроблюваного додатку

Загальні вимоги до розроблюваного додатку:

- Календар та можливість планувати події з опцією отримання повідомлення у час початку.
- Можливість створювати листи-нагадування з тим, що потрібно виконати (to-do list).
- Можливість ведення користувачем власного щоденника.
- Можливість ведення користувачем списку своїх витрат та прибутків.
- Зберігання усіх даних на хмарному сервісі iCloud.

Структура додатку має бути виконана у моделі проектування MVVM, що дозволить легко розширювати функціонал та дасть можливість працювати з моделлю даних та представленням окремо одне від одного. У програмі повинні бути передбачені можливість зберігання даних локально, на хмарному сервісі та отримання повідомлень про початок події.

Графічний інтерфейс додатку має бути інтуїтивно зрозумілим. Усі представлення додатку мають бути виконані в одному стилі.

Програма має складатися з 4 розділів, які об’єднані у Tab Bar Controller(переключання між розділами відбувається внизу екрану). Розділи мають назви:

1. Calendar
2. To-do list’
3. Diary
4. Finance

У кожному розділі має бути можливість додавання користувачем нової інформації, тому мають бути створені відповідні ViewControllers, або AlertConrtollers з полями для введення даних.

1.2 Мова Swift

Swift – надзвичайно зручний спосіб написання програм для смартфонів та планшетів, смарт-годинників, комп'ютерів Apple, серверів та інших пристроїв. Swift – безпечна, швидка та інтерактивна мова. Swift увібрала у себе ідеї найкращих розробників у світі. Ця мова є open-source, тож кожен бажаючий може прийняти участь у її розробці. Мова є C-подібною, тож код буде зрозумілий не тільки розробникам Swift, а й тим, хто знає щось про програмування. Swift має надзвичайно зрозумілий синтаксис, сама мова є легкою для оволодіння, але високий поріг входження у середу написання: як мінімум розробник повинен працювати на комп'ютері від компанії Apple, щоб отримати усі необхідні інструменти для написання повноцінних програм. Написання коду у пісочниці Playground дозволить експерементувати та миттєво бачити виконану роботу коду без необхідності компілювати та запускати програму.

Swift виключає велику кількість розповсюджених у програмуванні помилок:

- Змінні завжди ініціалізуються до того, як вони будуть використані
- Індекси масивів завжди перевіряються на out-of-bounds помилки
- Опціонали гарантують, що значення nil будуть явно оброблені
- Автоматичне управління пам'яттю

1.3 Огляд існуючих програм-аналогів

Серед існуючих програм аналогів одним з найяскравіших представників є додаток **Calendars от Readdle**. Програма має такий вигляд:

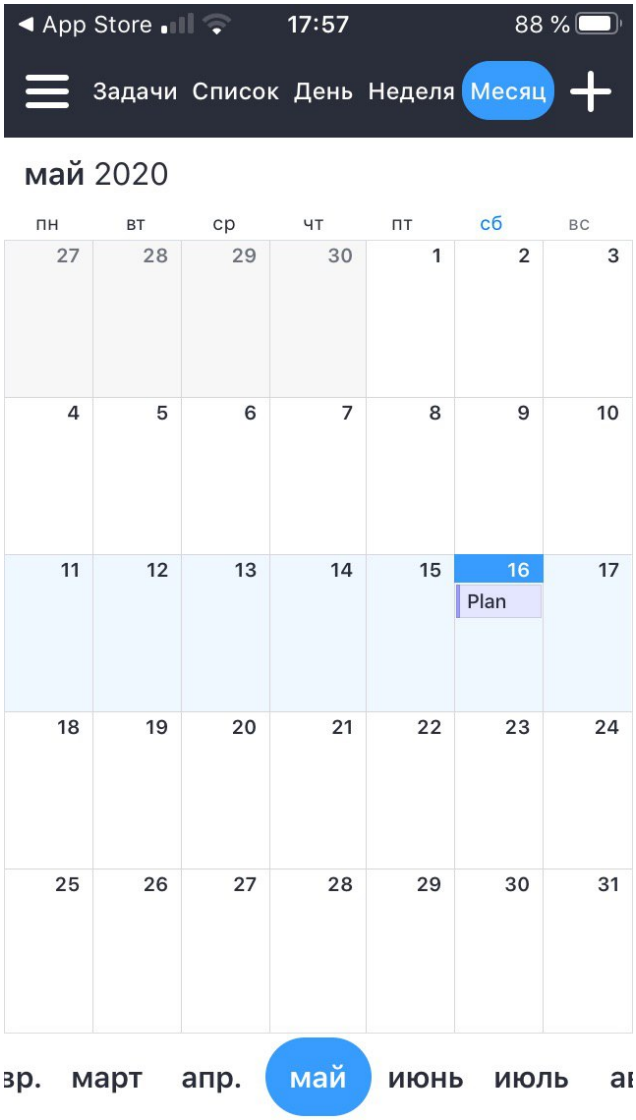

Рисунок 1.1 – Calendars от Readdle

Програма дозволяє планувати події дуже явним та цікавим способом. Основним інтерфейсом є календар, у який користувач додає події та отримує повідомлення про них. Також програма синхронізована з iCloud

Одним з найкращих представників програм типу ToDoList є програма **Todoist**. Додаток має приємний та сучасний дизайн, дуже швидку працездатність та легка для використання.

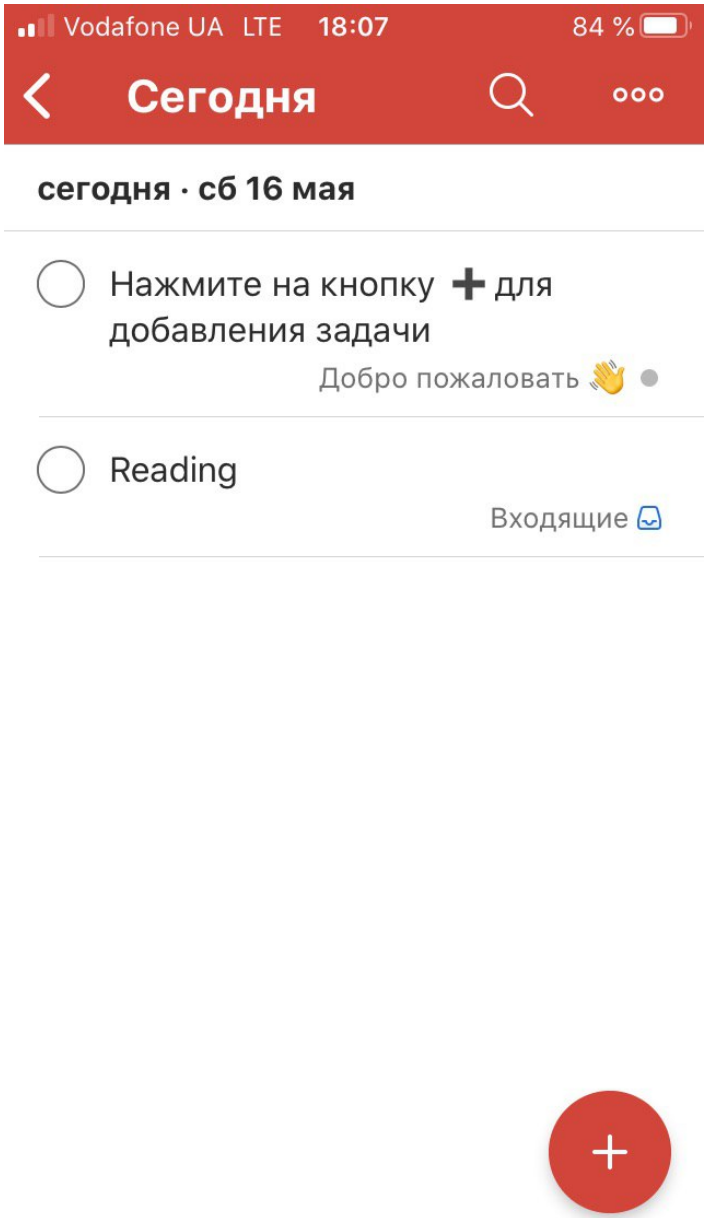

Рисунок 1.2 – Todoist

Список можна продовжити ще величезною кількістю платних та безкоштовних програм. Усі вони мають щось спільне, та кожна має ціль залишитися довше на девайсі користувача і стати для нього незамінною.

1.4 Постановка задачі

Основною метою випускної роботи є надбання навичок створення додатків на смартфон з підтримкою мови Swift, у даному випадку програма написана під iPhone. Також у процесі виконання роботи повинні бути освоєні навички роботи з Core Data – фреймворку від компанії Apple, який дозволяє

взаємодіяти з локальною базою даних; навички роботи з локальними повідомленнями(User Notifications); підключення додатку до хмарного сервісу iCloud та зберігання даних на ньому а також синхронізація даних на усіх iPhone користувача. Також повинні бути здобутими упевнене вміння користування середи розробки XCode.

Предметом дослідження є технологія створення iOS додатку.

Для реалізації проекту необхідно виконати наступні завдання:

- Вивчити мову Swift та здобути навички програмування у середі XCode(було виконано упродовж усіх років навчання в університеті)
- Підключити проект у XCode до iCloud за допомогою аккаунта розробника(було використано аккаунт розробника мого батька)
- Проектування моделі даних
- Написання коду програми, використовуючи патерн проектування MVVM
- Перевірка програми на можливі баги з подальшим їх усуненням

2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ДОДАТКУ

2.1 Структура додатку

Структура програми матиме такий вигляд:

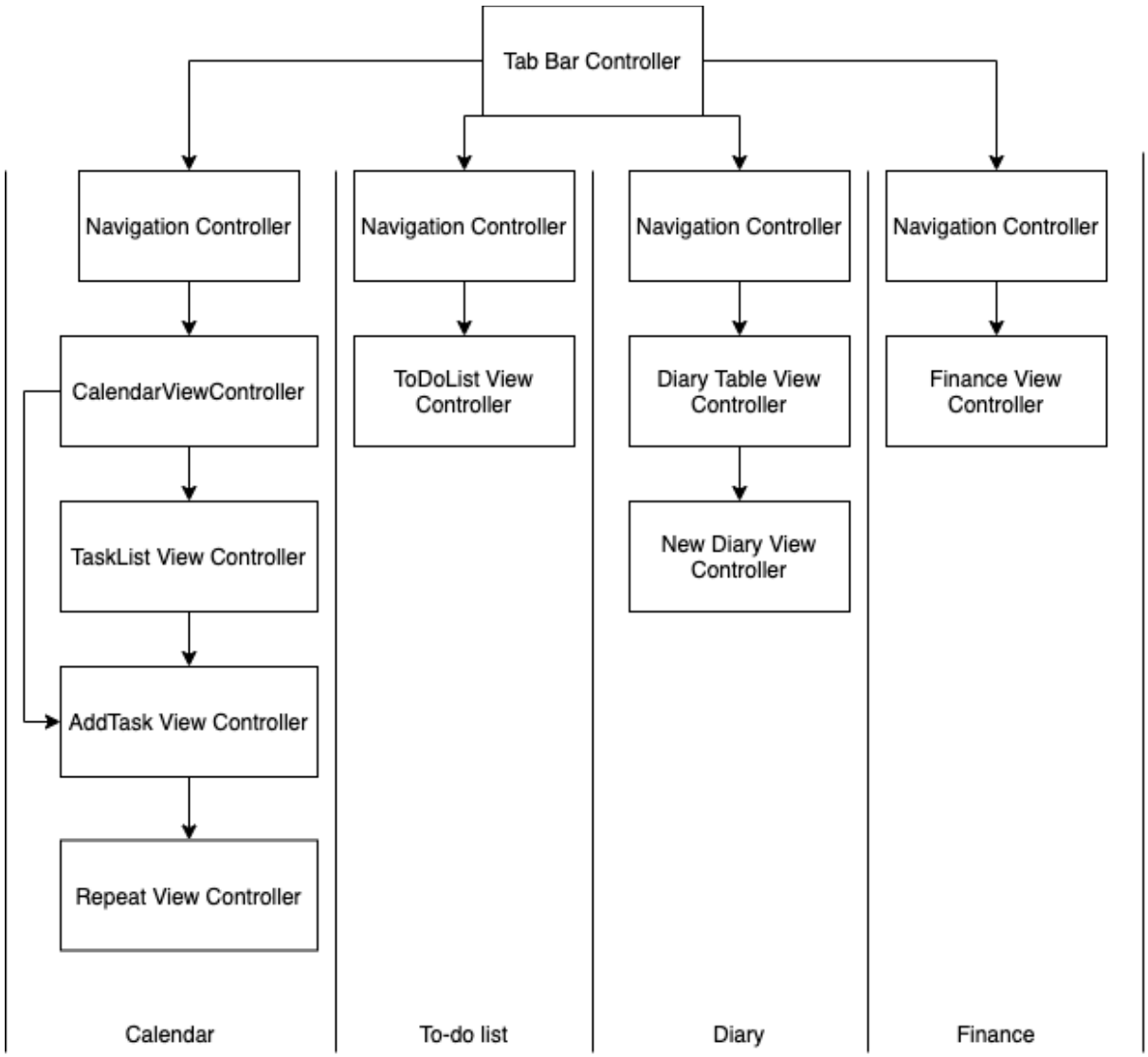

Рисунок 2.1 – Структура додатку

Тобто структура програми повністю відповідає структурі представлень у Main.storyboard, яка має такий самий вигляд:

Рисунок 2.2 – Структура додатку у Main.storyboard

У кінцевому варіанті програма нараховує 41 файл.

2.2 Модель даних

У кінцевому варіанті модель даних має такий вигляд:

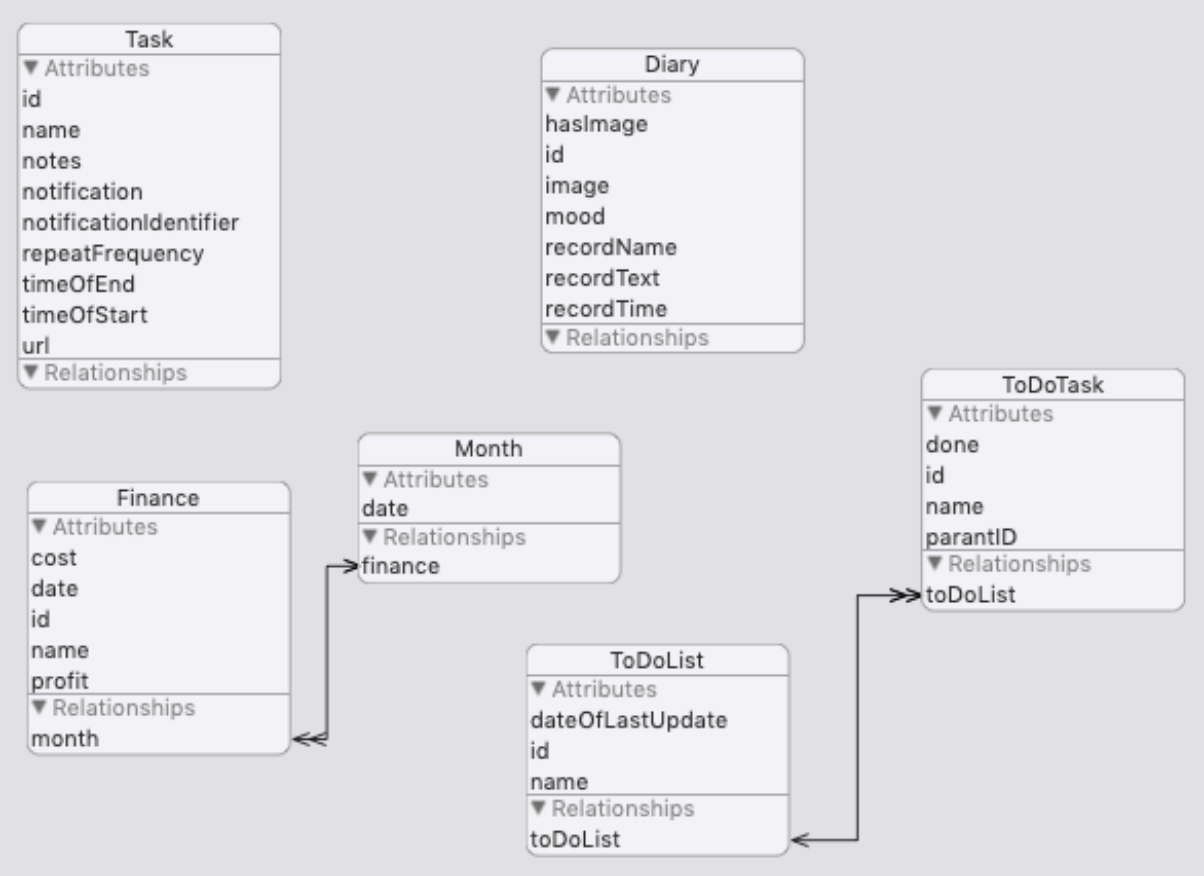

Рисунок 2.3 – ER-діаграма бази даних

База даних додатку складається з таких таблиць:

Таблиця 2.1 Task(завдання для виконання у певний час)

Назва поля	Призначення поля	Тип даних
id	Id сутності(потрібно для зберігання у хмарі)	String
name	Назва завдання	String
notification	Чи буде з’являтися повідомлення	Bool
notificationIdentifier	Ідентифікатор повідомлення	String
repeatFrequency	Частота приходу повідомлення	String
timeOfStart	Час початку	Date
timeOfEnd	Час кінця	Date
url	URL адреса	String

Таблиця 2.2 Diary(запис у щоденнику)

Назва поля	Призначення поля	Тип даних
id	Id сутності(потрібно для зберігання у хмарі)	String
hasImage	Чи є зображення	Bool
image	Зображення	Data
mood	Настрій	String
recordName	Назва запису	String
recordText	Текст запису	String
recordTime	Час запису	Date

Таблиця 2.3 ToDoList(список того, що потрібно зробити)

Назва поля	Призначення поля	Тип даних
id	Id сутності(потрібно для зберігання у хмарі)	String
name	Назва списку	String
dateOfLastUpdate	Час останнього оновлення	Date
toDoList	Список, який потрібно виконати	Relationship(one to many)

Таблиця 2.4 ToDoTask(завдання, яке потрібно виконати)

Назва поля	Призначення поля	Тип даних
id	Id сутності(потрібно для зберігання у хмарі)	String
name	Назва завдання	String
done	Чи виконане завдання	Bool
parantID	ID батьківскої сутності	String
toDoList	Посилання на батьківську сутність	Relationship(to one)

Таблиця 2.5 Month(місяць для обліку фінансів)

Назва поля	Призначення поля	Тип даних
date	Дата	Date
finance	Список доходів та витрат	Relationship(one to many)

Таблиця 2.6 Finance(запис доходу або витрати)

Назва поля	Призначення поля	Тип даних
id	Id сутності(потрібно для зберігання у хмарі)	String
name	Назва доходу або витрати	String
cost	Вартість	Double
date	Дата	Date
profit	Доход чи витрата	Bool
month	Місяць	Relationship(to one)

2.3 Підключення до iCloud

Для того щоб підключити додаток до хмарного сервісу iCloud для подальшого зберігання даних на ньому, потрібно виконати такі дії:

Обрати у проекті XCode аккаунт розробника та підключити сервіс iCloud

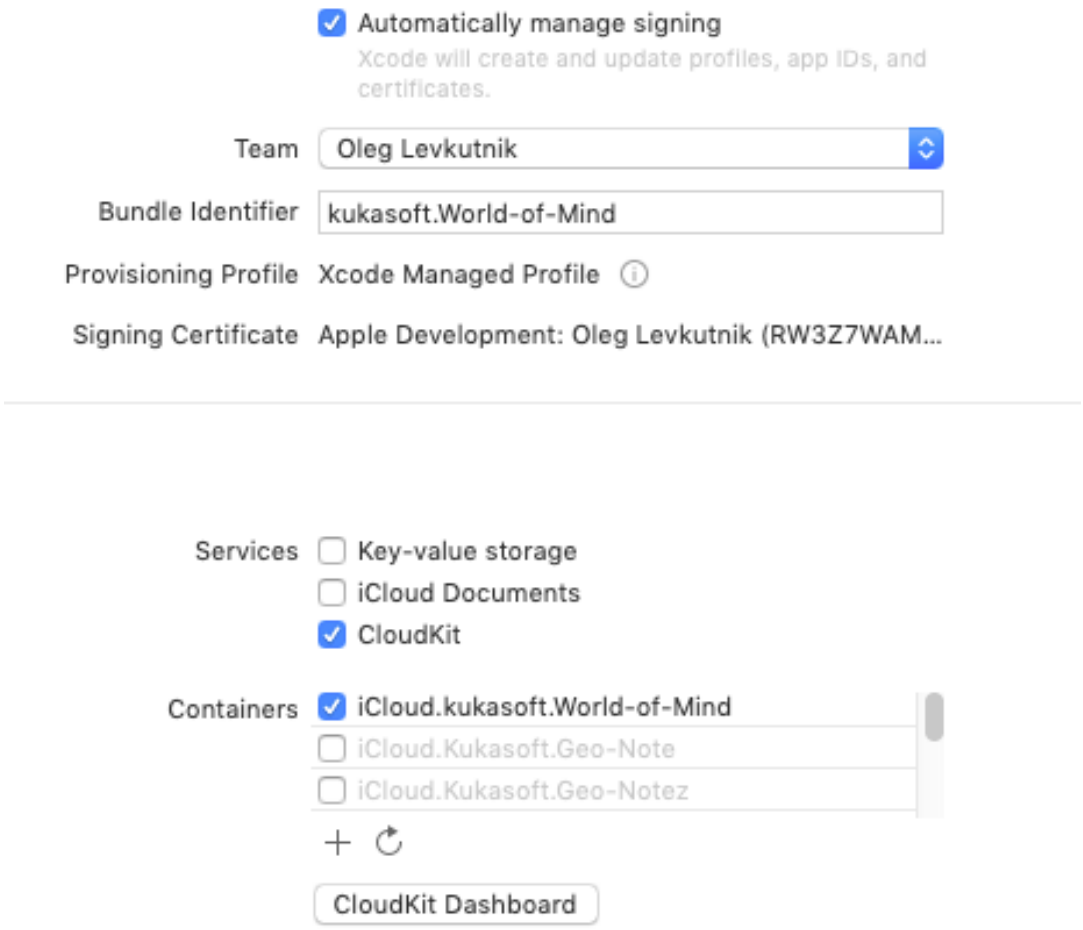

Рисунок 2.4 – Підключення до iCloud

Потім необхідно натиснути на CloudKit Dashboard. У браузері автоматично відкриється сторінка з нашим проектом на сайті icloud.developer.apple.com

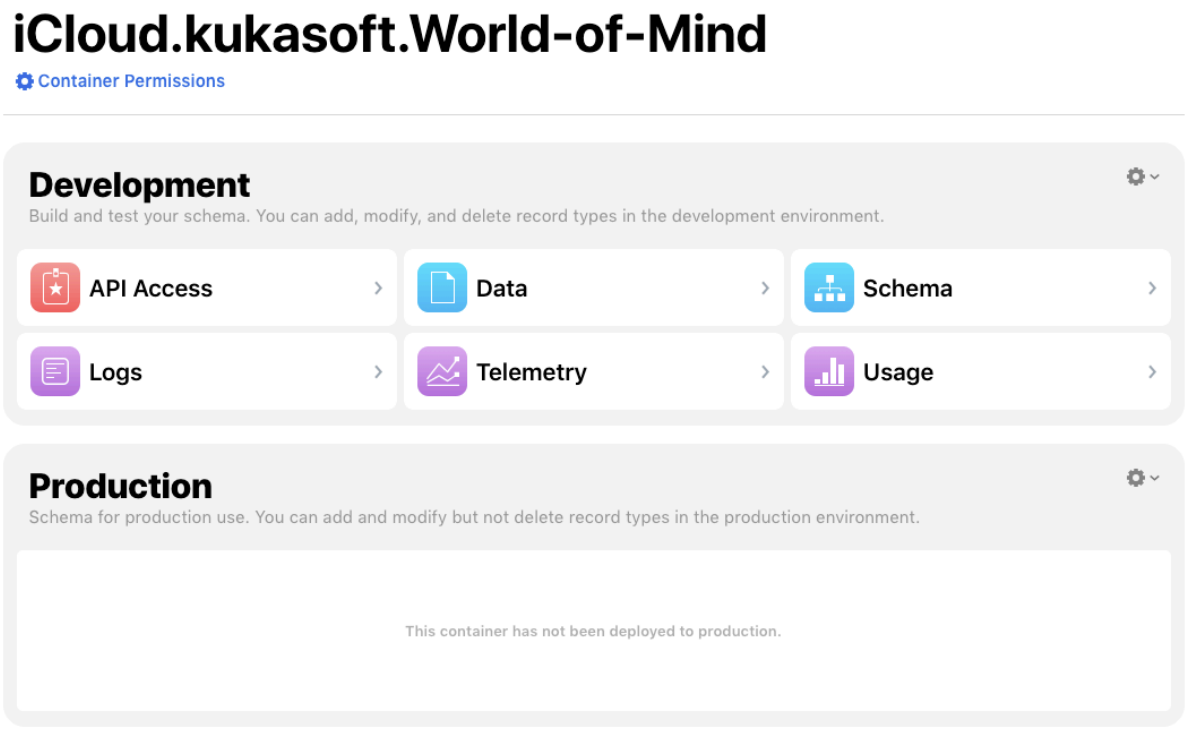

Рисунок 2.5 – Сторінка додатку у iCloud

У розділі Schema необхідно створити типи зберігання даних. Кінцевий варіант матиме такий вигляд:

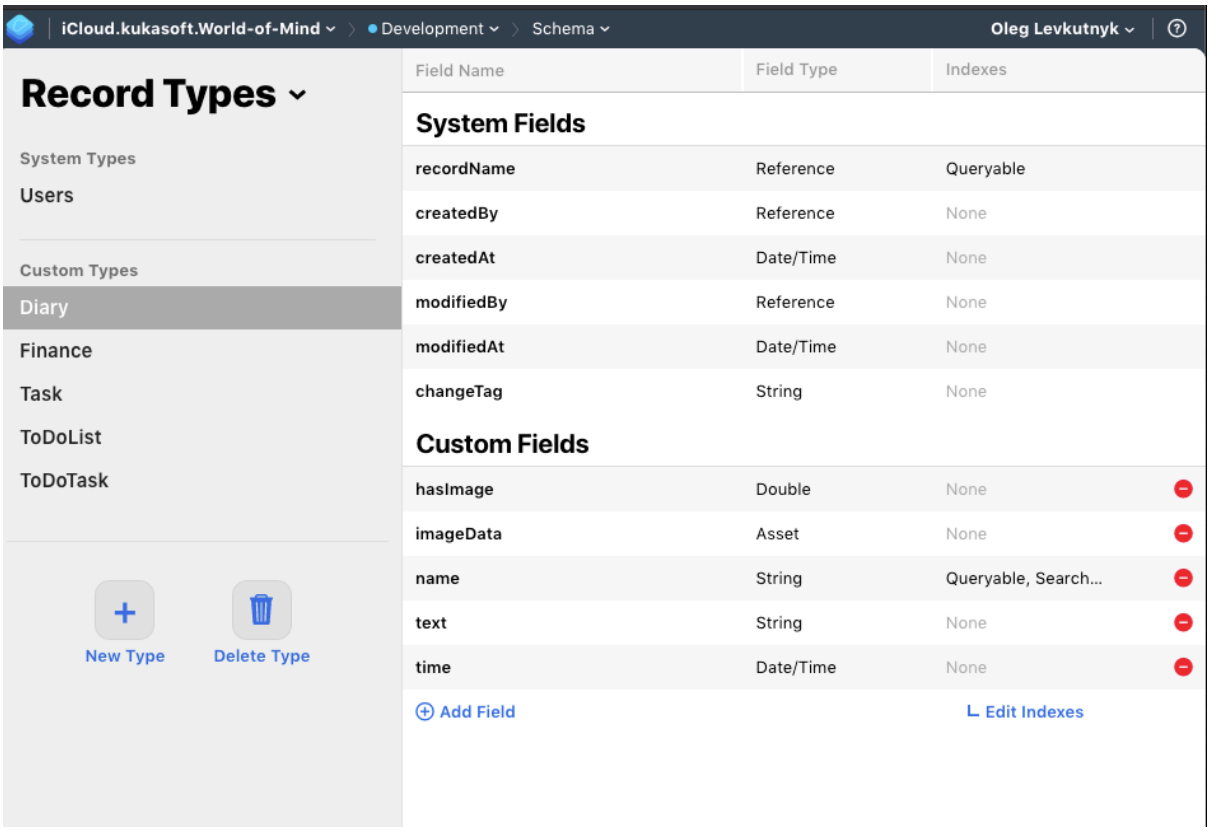

Рисунок 2.6 – Record Types

Далі необхідно створити клас CloudManager, за допомогою якого буде здійснюватися взаємодія з iCloud. Код класу буде представлений у додатку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЇЇ ОПИС

3.1 Опис проекту

3.1.1 Загальний опис початку користування додатком

У XCode проекті міститься 41 файл, не враховуючи кінцевий продукт та фреймворк CloudKit. Усі файли розміщені у папках, які відповідають логічному змісту файлів. Код написано відповідно структурі MVVM і також структуровано у відповідних папках.

Додаток має зрозумілий для непідготовленого користувача дизайн, де зрозуміло, що робить кожен елемент.

Для коректної роботи зі хмарою, користувач повинен бути авторизованим під своїм Apple аккаунтом у «Налаштуваннях»:

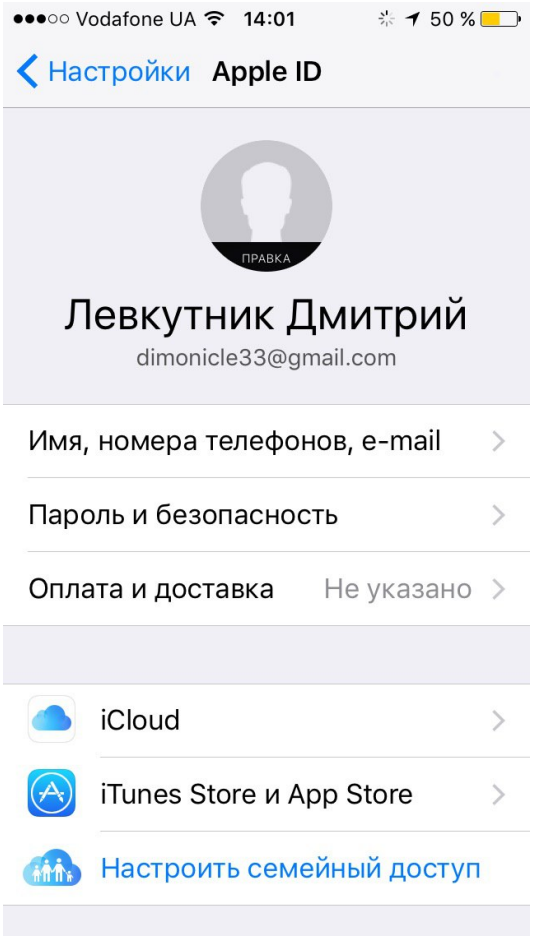

Рисунок 3.1 – Авторизований користувач

При першому запуску додатку користувач отримає повідомлення про дозвіл надсилання повідомлень:

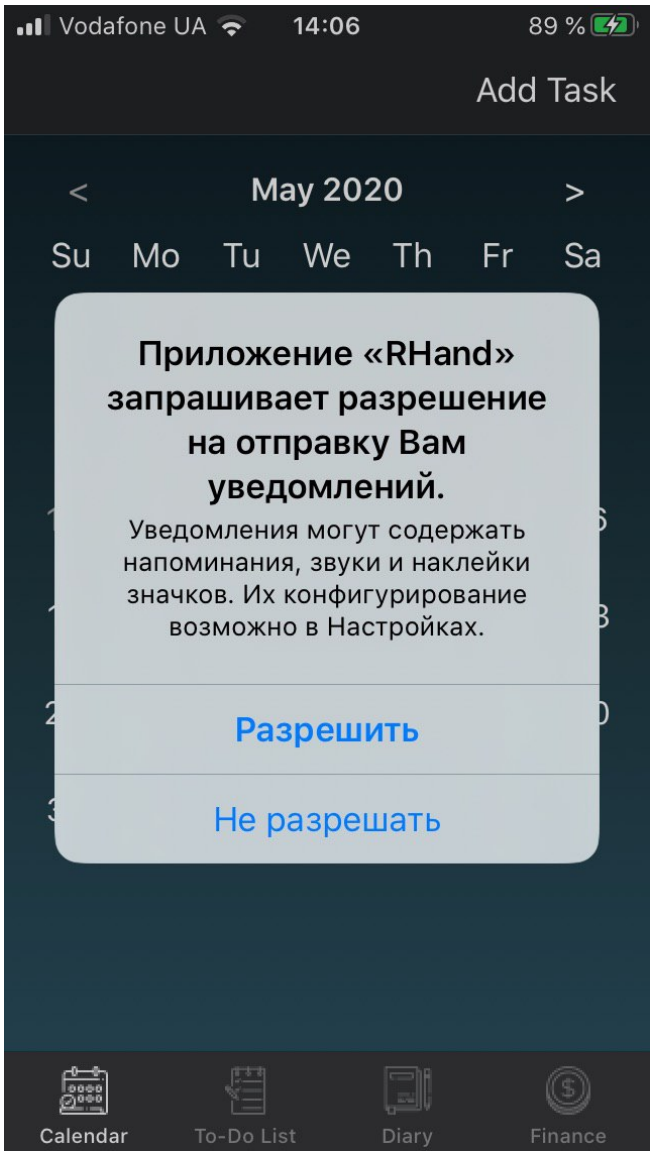

Рисунок 3.2 – Дозвіл на відправку повідомлень

Далі користувач опиняється на екрані з календарем, де він має змогу перейти на конкретну дату або одразу добавити нове завдання. Також користувач має можливість перейти на інші розділи: Список справ, Щоденник або Облік фінансів:

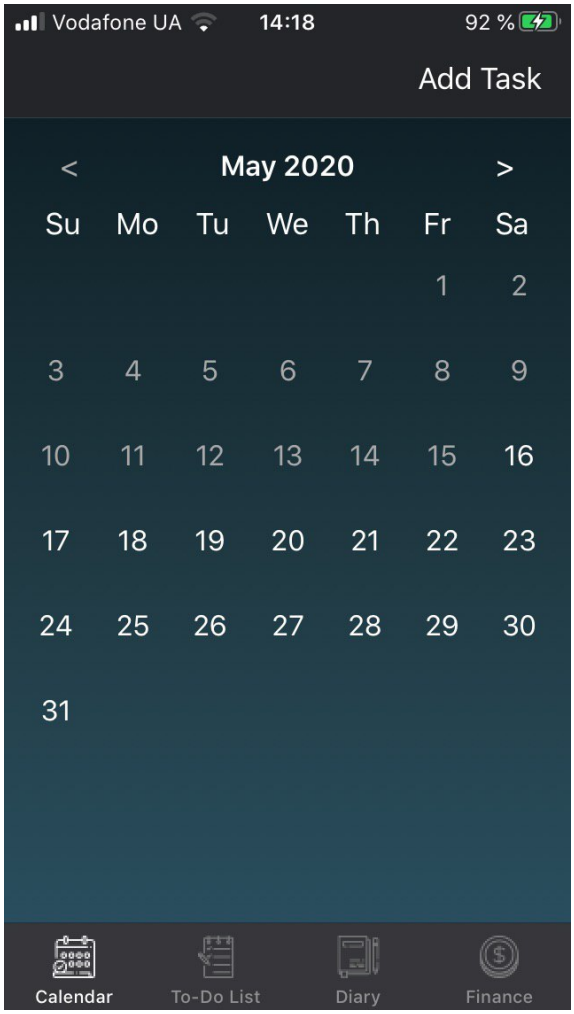

Рисунок 3.3 – Календар

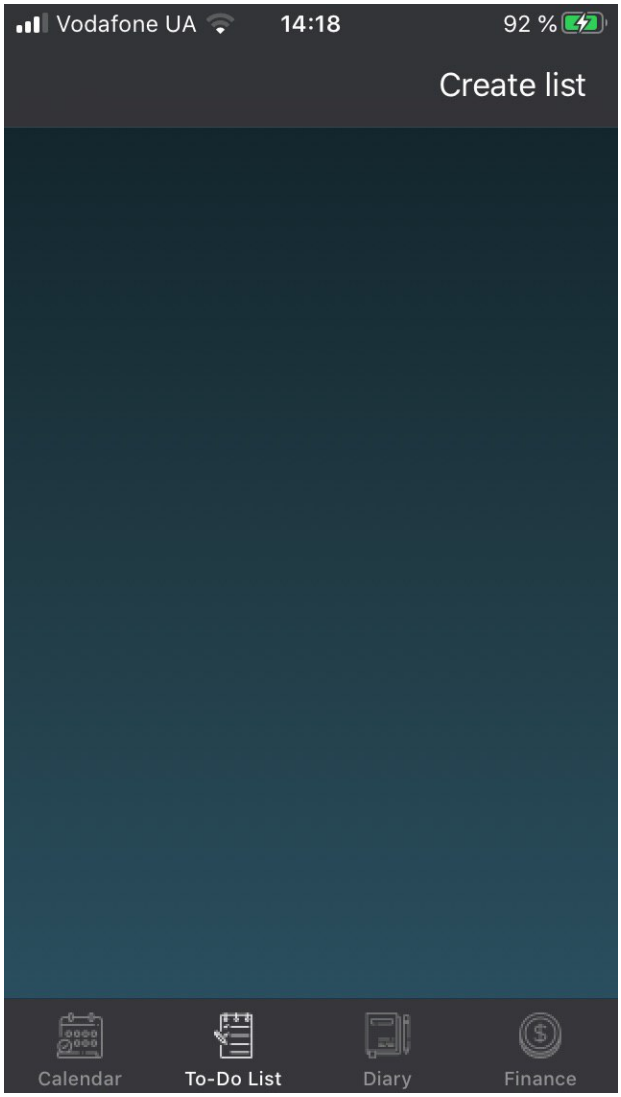

Рисунок 3.4 – Список справ

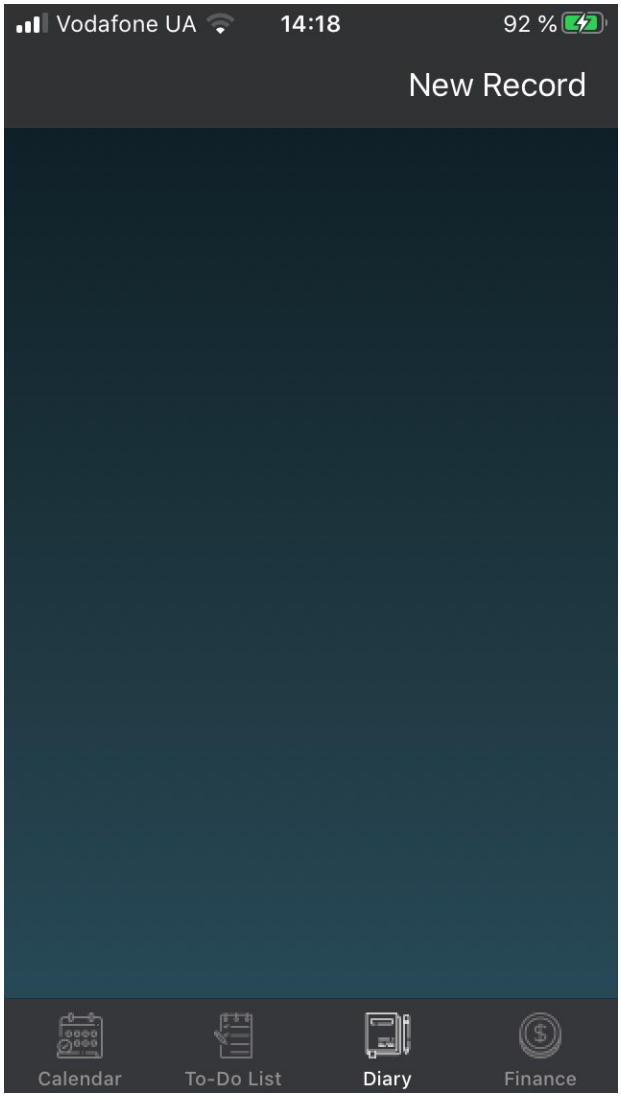

Рисунок 3.5 – Щоденник

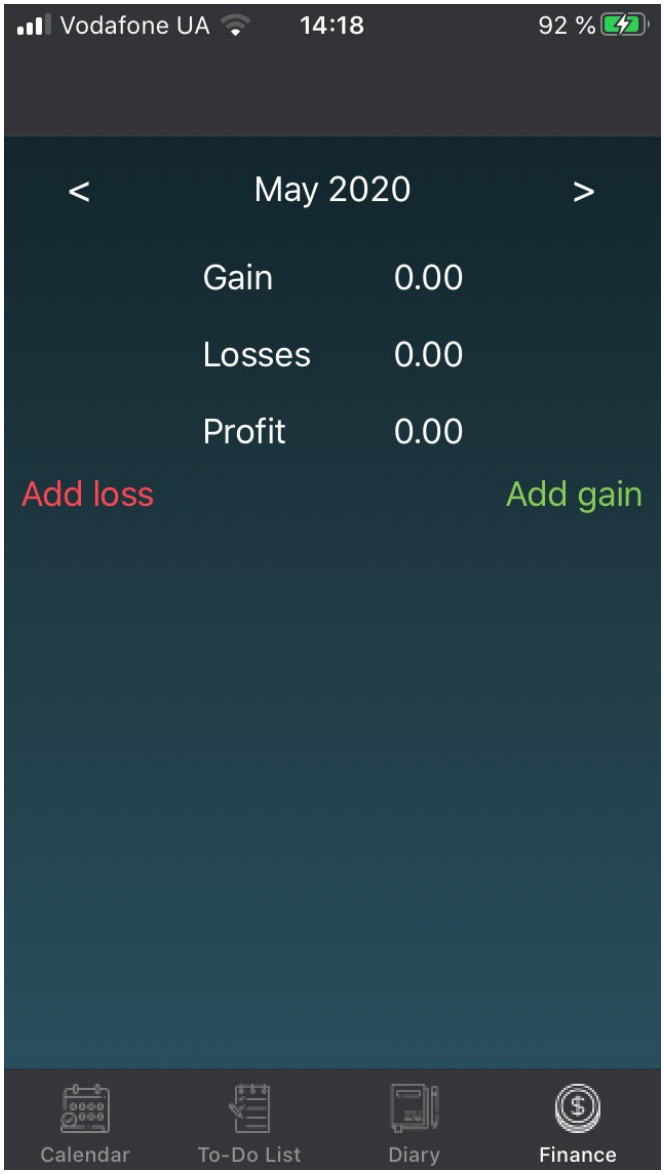

Рисунок 3.6 – Облік фінансів

Для додавання нової справи для виконання у календарі, користувач переходить на відповідне вікно та вводить усі необхідні дані та тисне на кнопку Save

3.1.2 Calendar

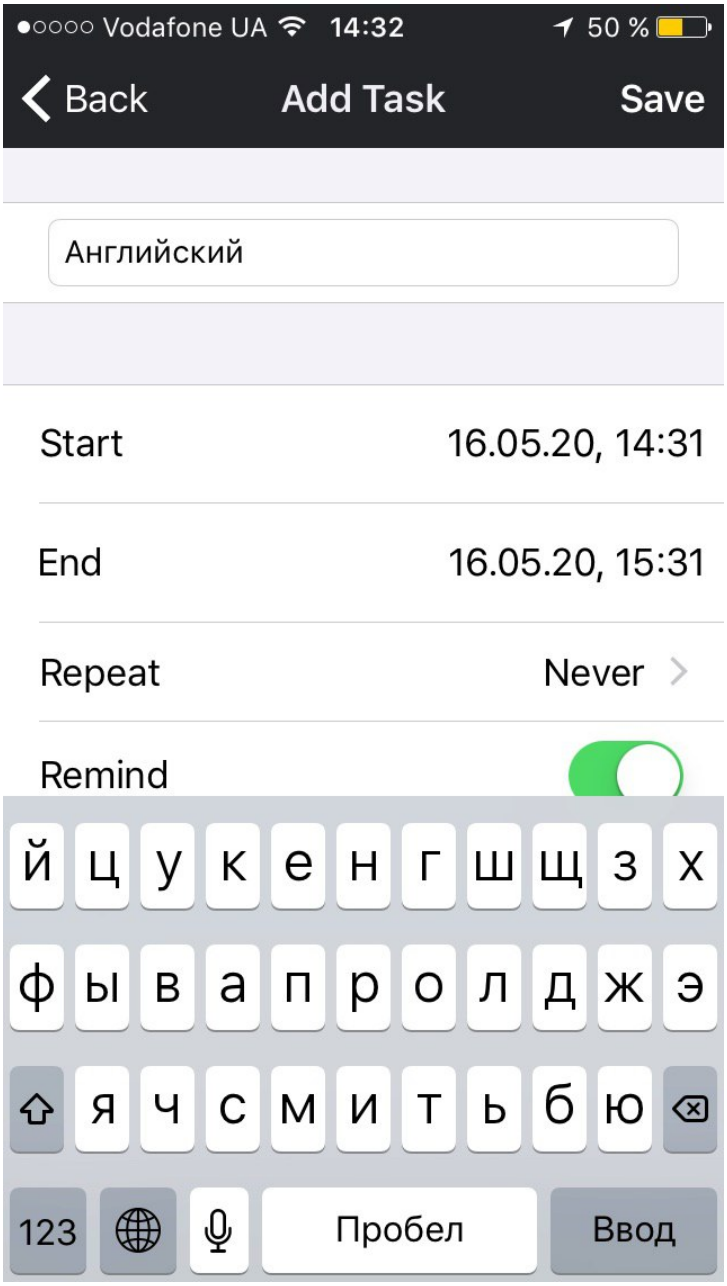

Рисунок 3.7 – Введення даних про завдання

Також користувач має змогу налаштувати частоту отримання повідомлень. Для цього він тисне на поле Repeat та обирає частоту:

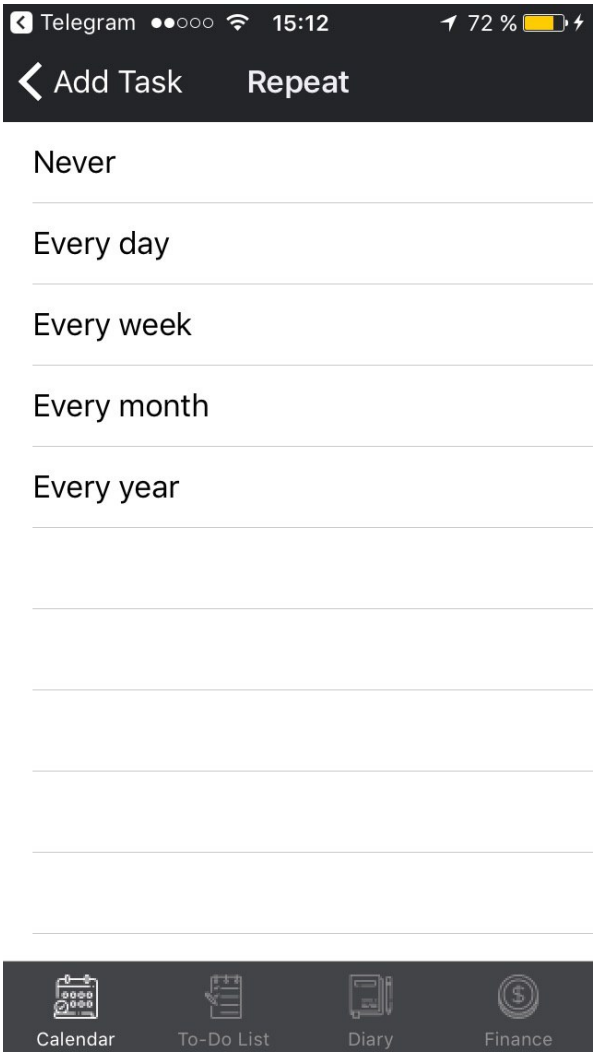

Рисунок 3.8 – Обрання частоти приходу повідомлень

Після цього завдання з’являється у списку завдань під обраною датою. У момент початку користувач отримує відповідне повідомлення з назвою завдання:

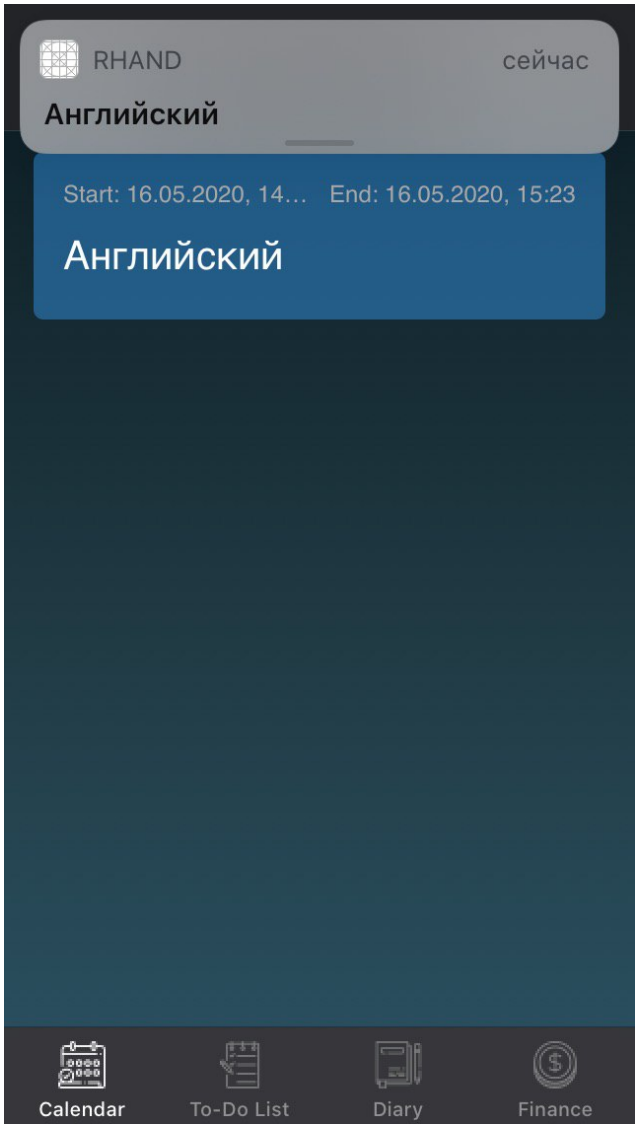

Рисунок 3.9– Отримання повідомлення про завдання

У майбутньому користувач має змогу повністю редагувати задачу.

При створенні списку справ(To-Do List) користувач натичкае на кнопку Create list та отримує відповідний AlertController:

3.1.3 To-Do List

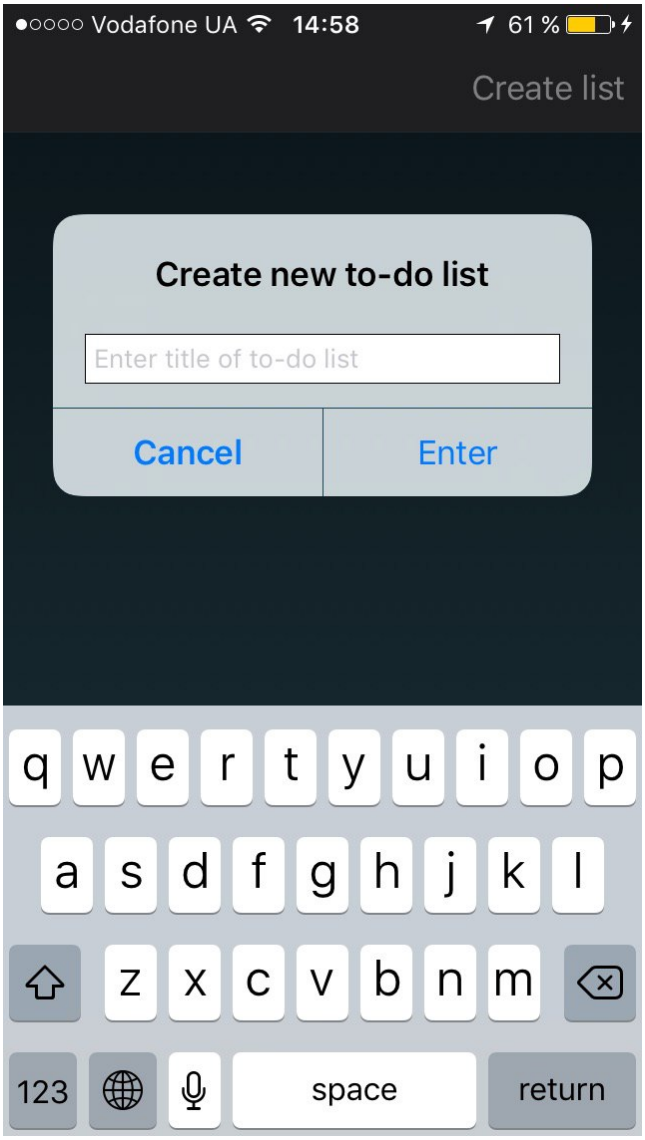

Рисунок 3.10 – Створення нового списку справ

Користувач вводить назву списку та тисне Enter, після цього з’являється новий список справ:

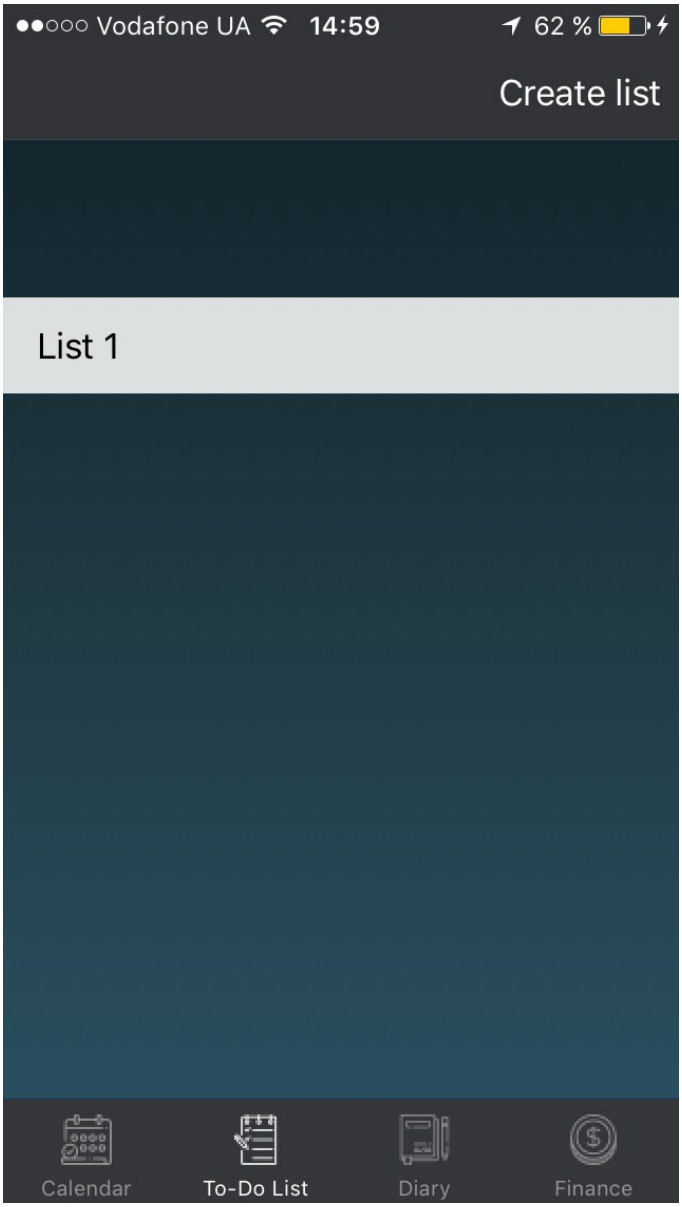

Рисунок 3.11 – Створення нового списку справ

Користувач тисне на список та у таблиці з’являється нова клітина; користувач тисне на кнопку “+” та з’являється поле для вводу назви:

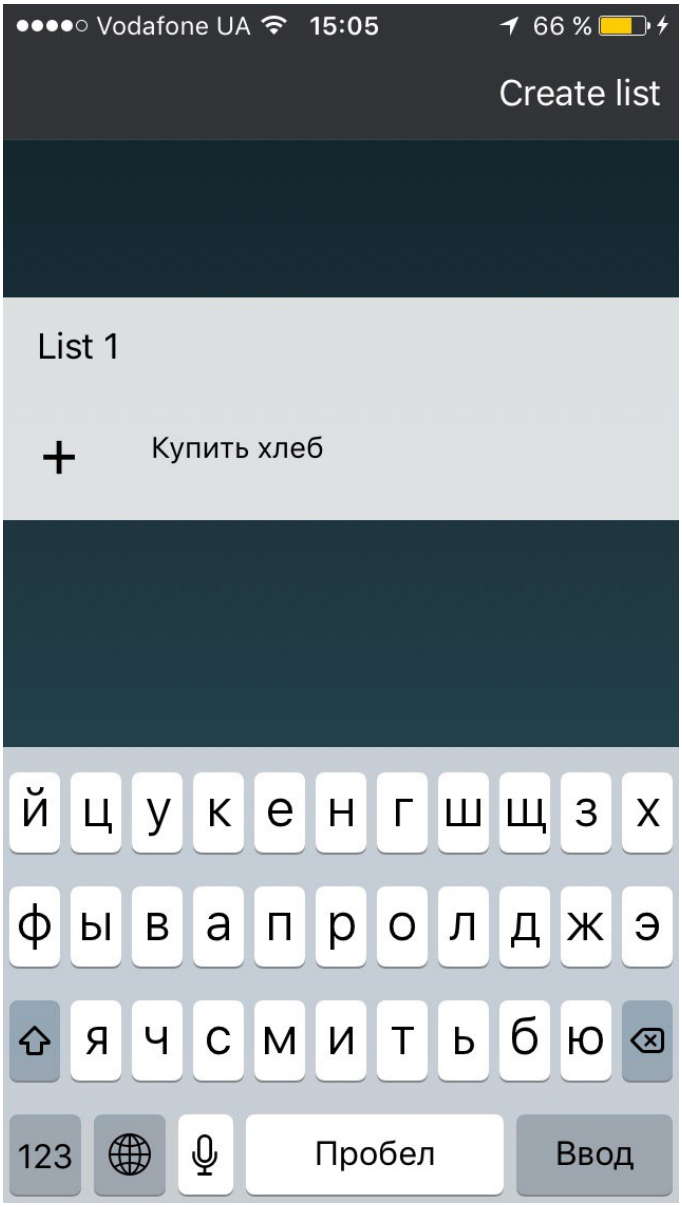

Рисунок 3.12 – Створення нової справи

Після цього користувач знову тисне на “+”. Далі користувач має можливість відмітити справу як зроблену, натиснувши на коло. Після цього у колі з'явиться галочка.

У розділі Diary користувач тисне

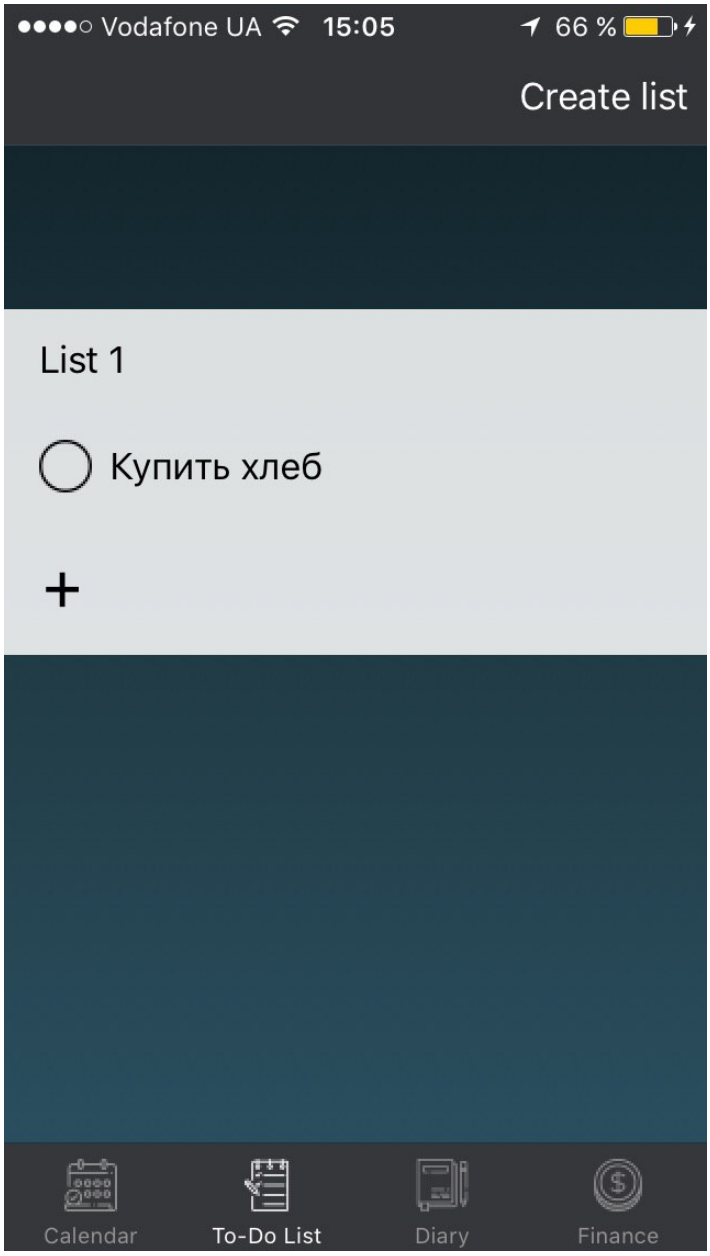

Рисунок 3.13 – Справа не виконана

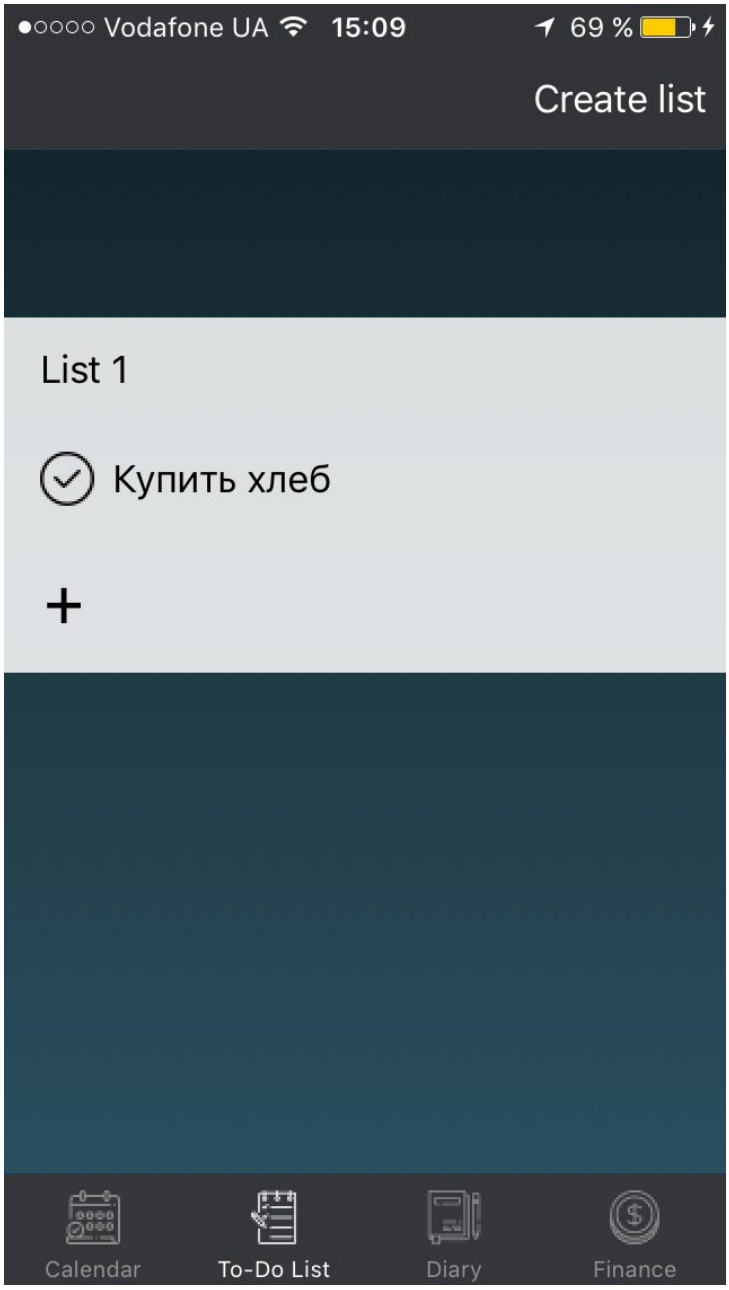

Рисунок 3.14 – Справа виконана

У розділі Diary користувач тисне на кнопку New Record та переходить на екран введення інформації до свого щоденника, де має змогу обрати фото, записати назву та ввести основний текст:

3.1.4 Diary

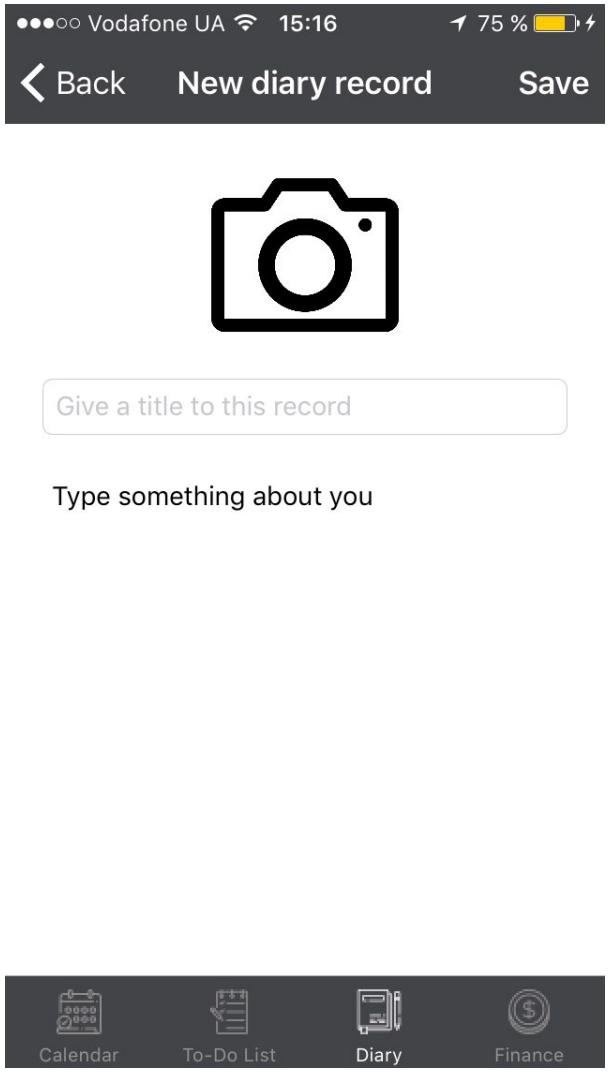

Рисунок 3.15 – Введення даних до щоденнику

Після цього запис з’являється у таблиці та має такий вигляд:

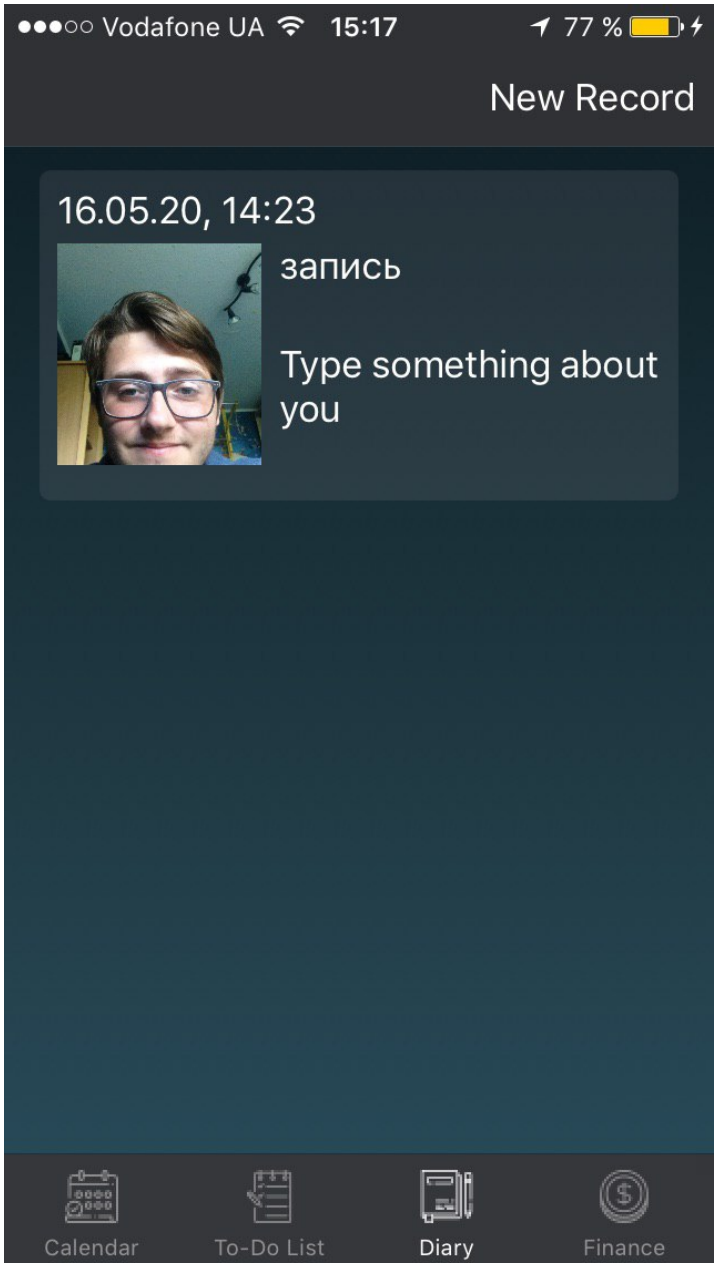

Рисунок 3.16 – Запис у щоденнику

Також у майбутньому користувач має змогу редагувати запис.

На екрані Finance користувач має змогу дивитися свої доходи та витрати по місяцям. Щоб додати витрату або прибуток необхидно натиснути відповідно **Add loss** або **Add gain**. Після цього з’явиться Alert Controller у якому необхідно вписати назву та вартість.

3.1.5 Finance

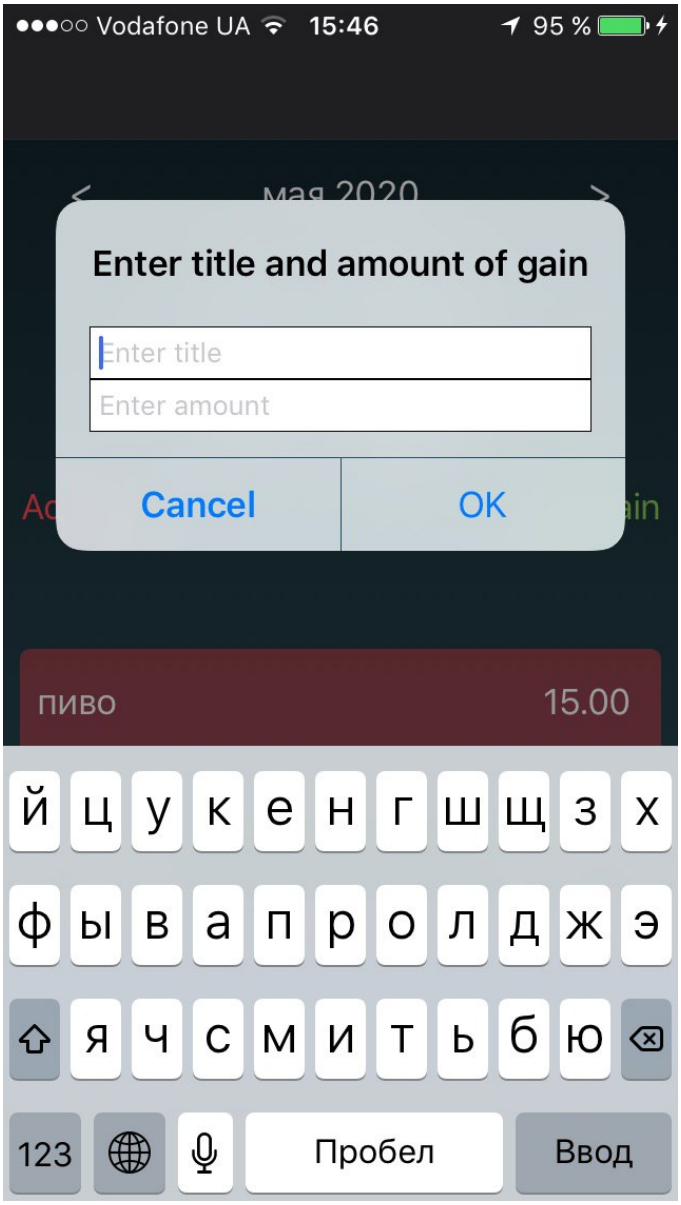

Рисунок 3.17 – Введення доходу або витрати

Після чого записи з’являться у таблиці а також буде підрахована вигода:

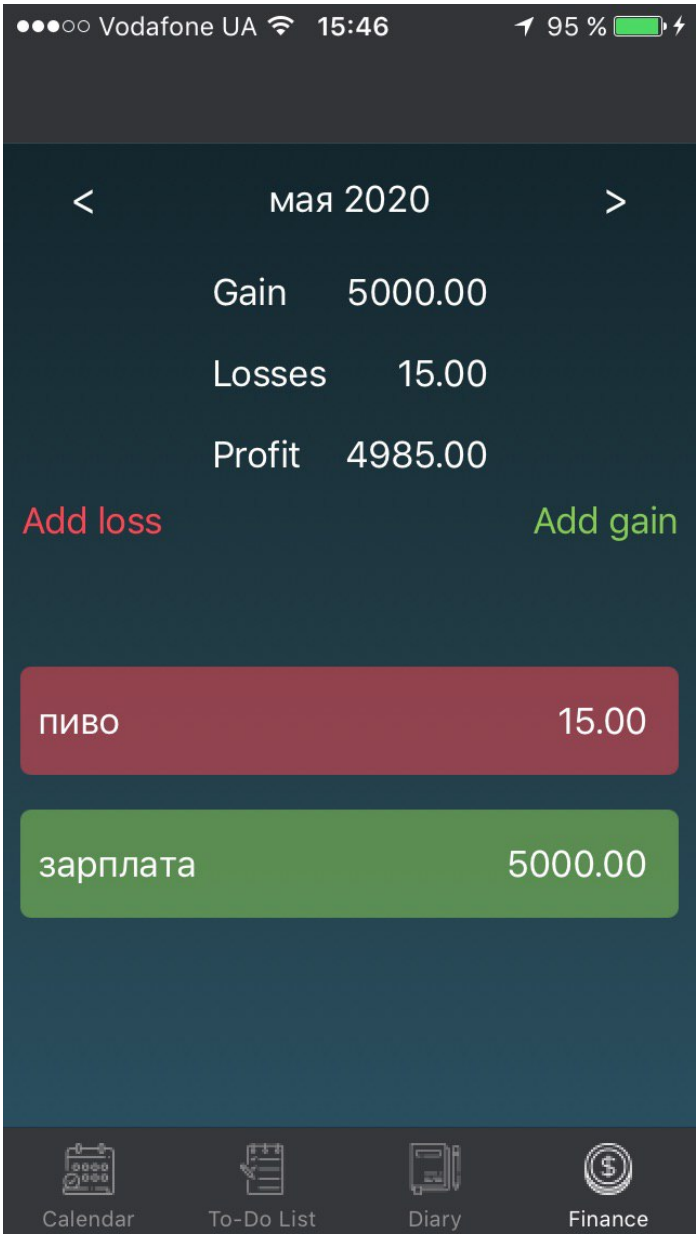

Рисунок 3.18 – Екран фінансів

3.2 Заповнення даними

Заповнена даними програма має наступний вигляд:

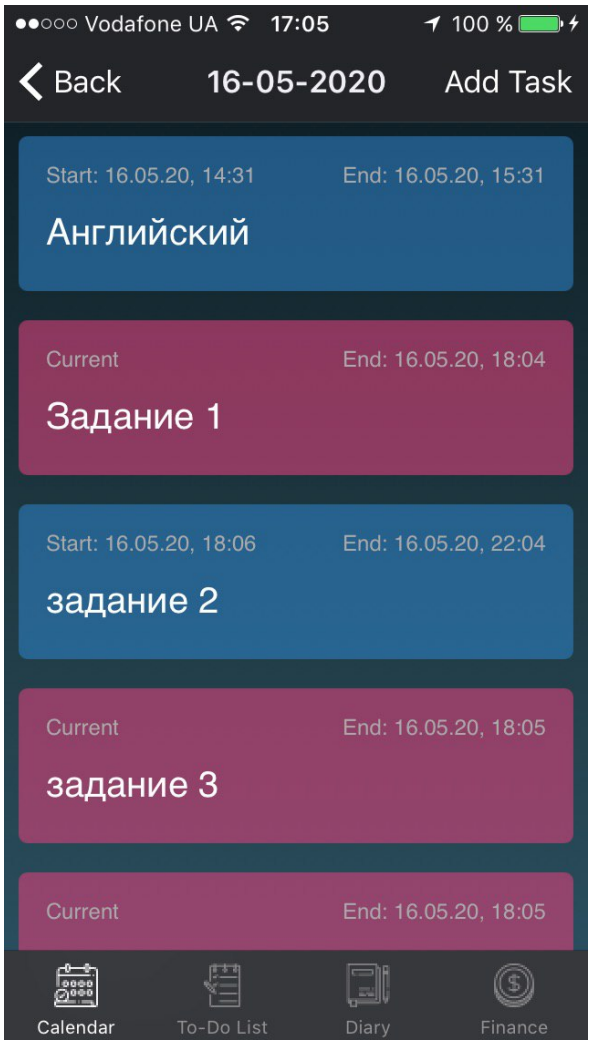

Рисунок 3.19 – Calendar(16-05-2020)

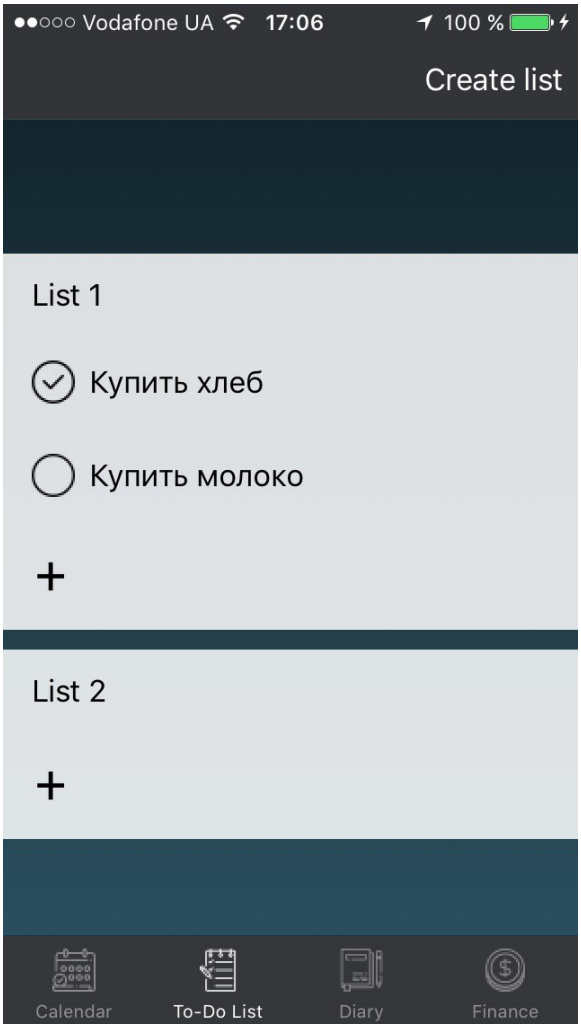

Рисунок 3.20 – To-Do List

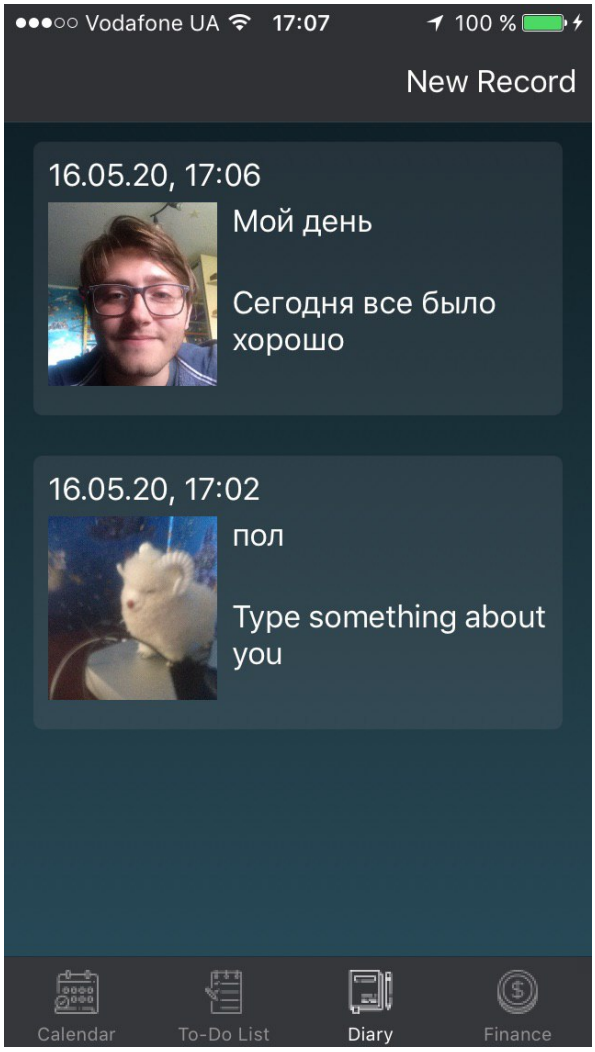

Рисунок 3.21 – Diary

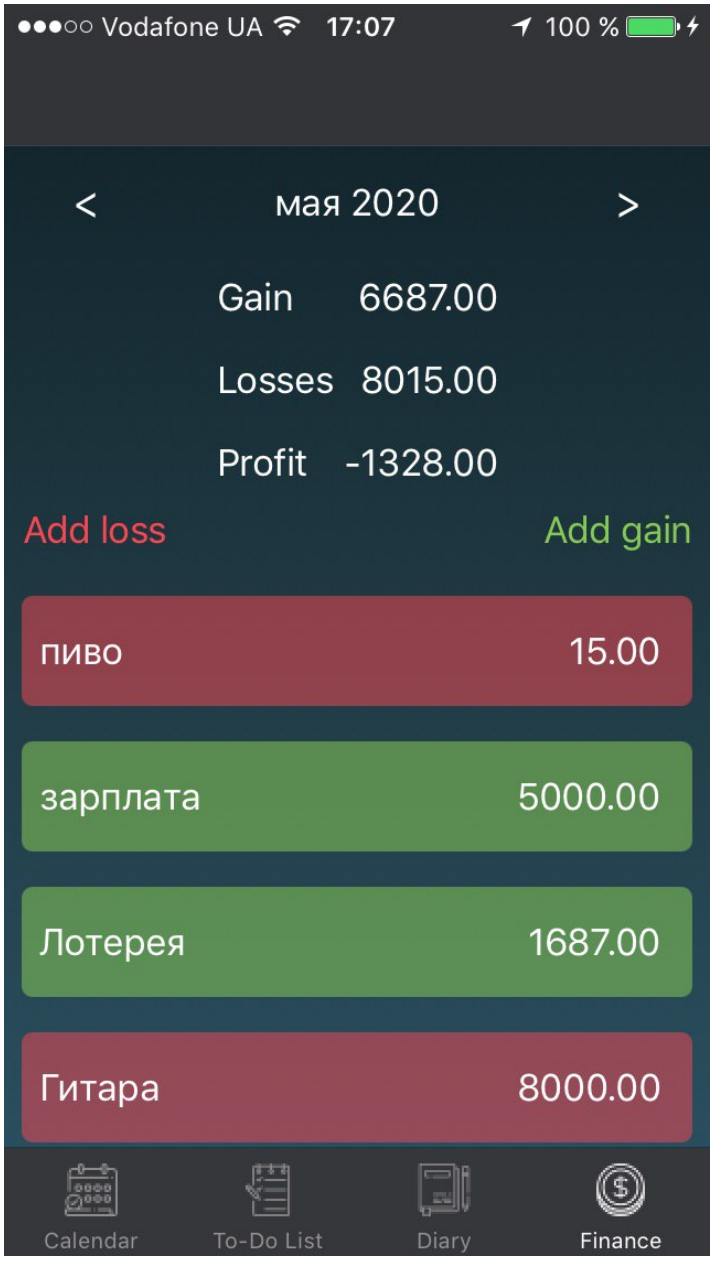

Рисунок 3.22 – Finance

Також усі дані збережені у iCloud:

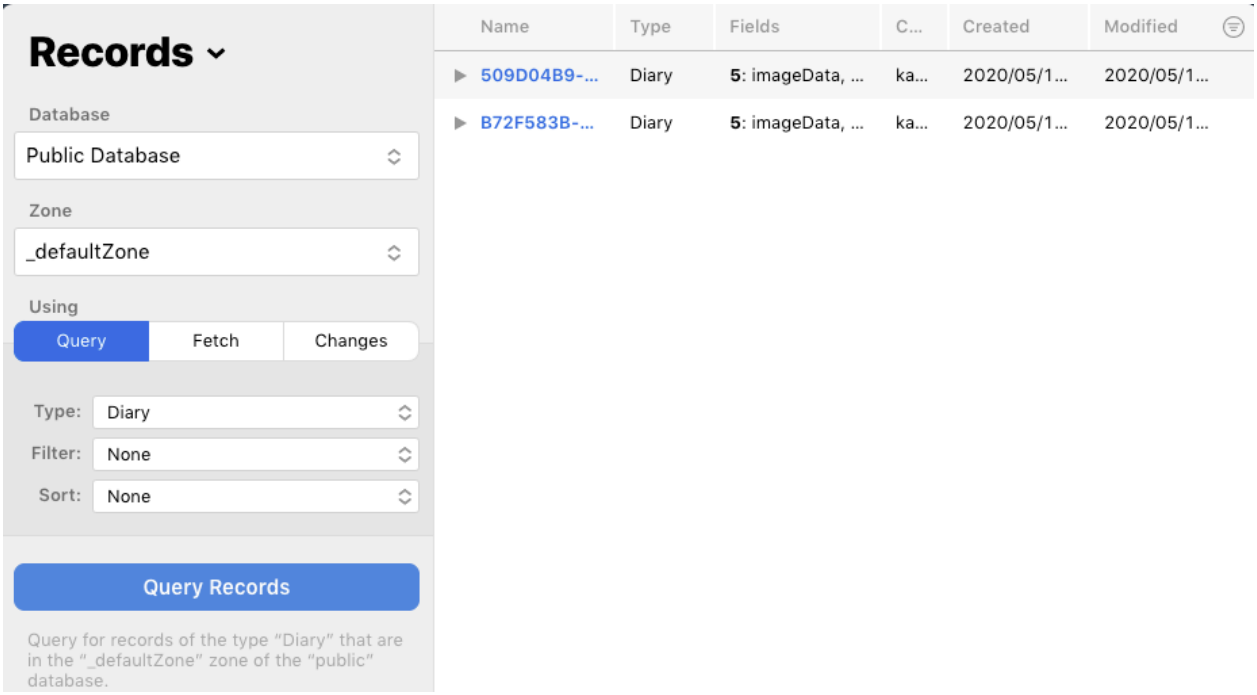

Рисунок 3.23 – Diary iCloud

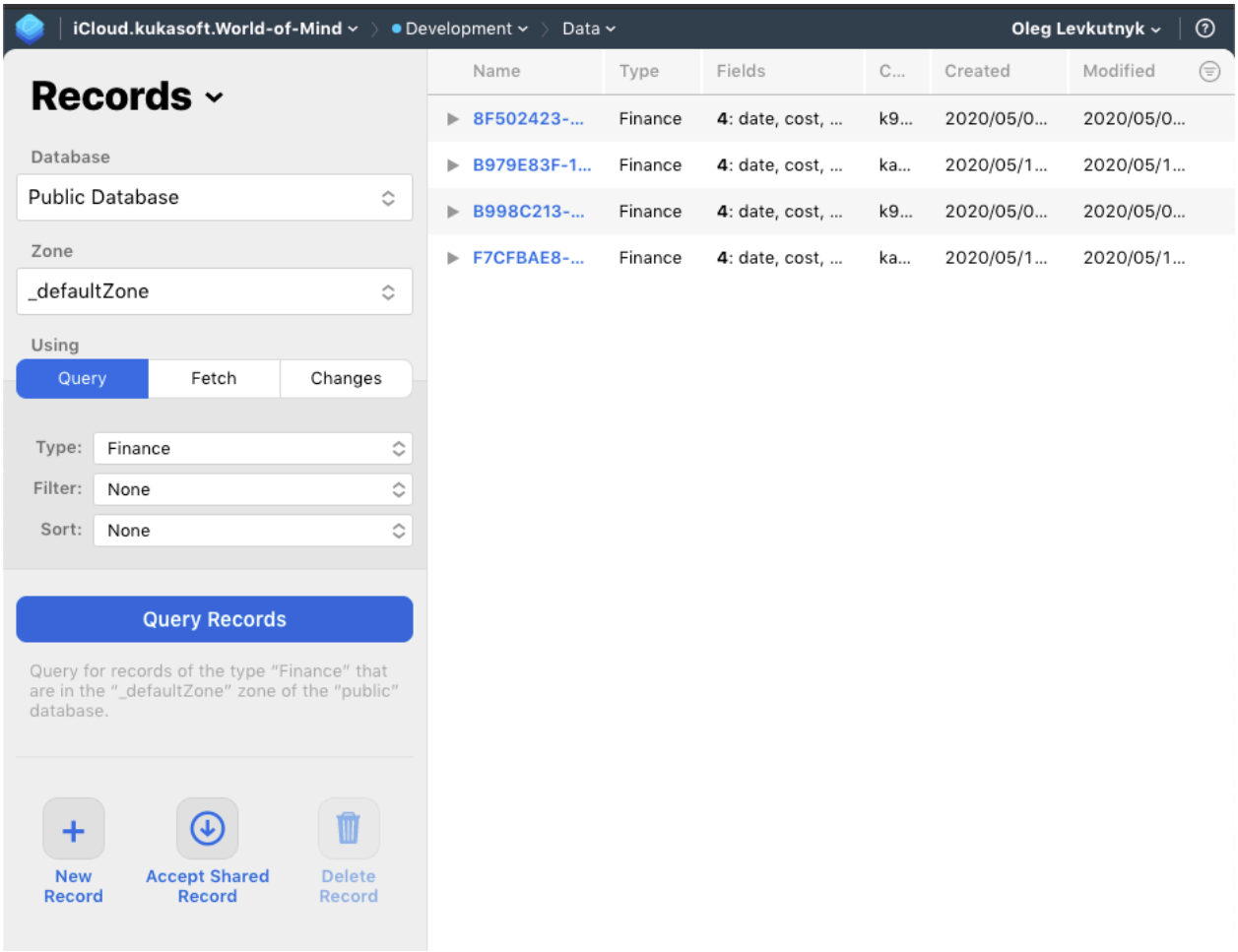

Рисунок 3.24 –Finance iCloud

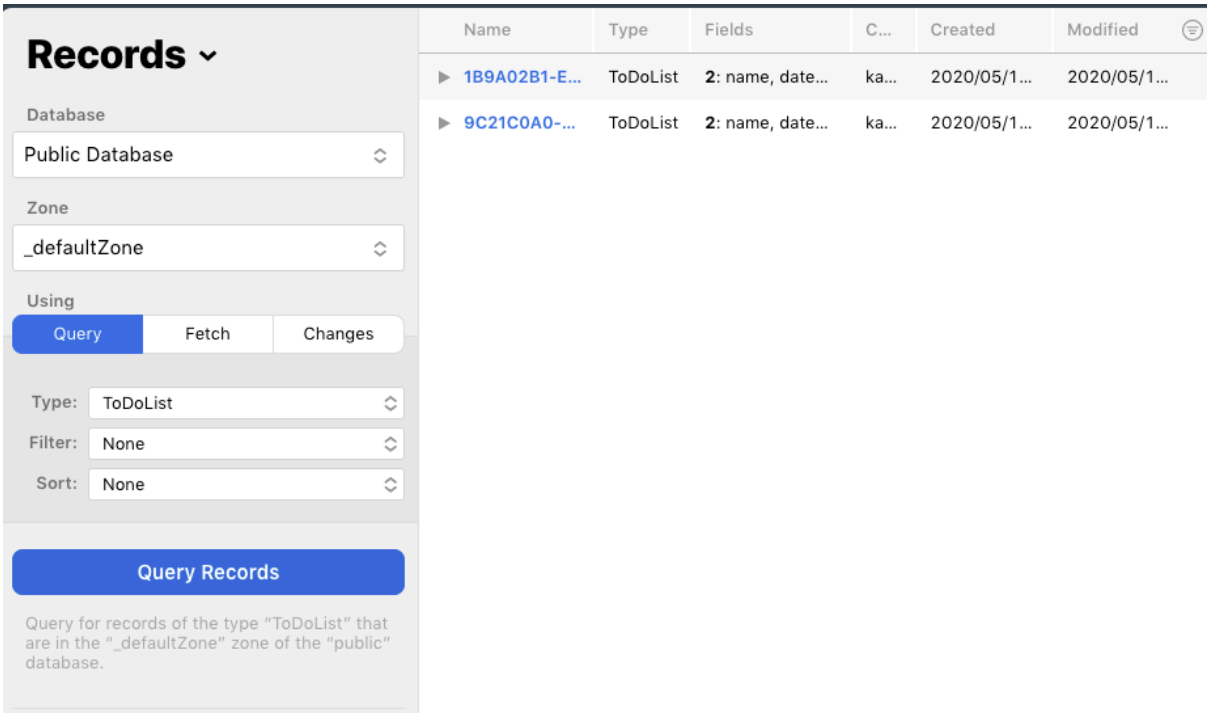

Рисунок 3.25 – ToDoList iCloud

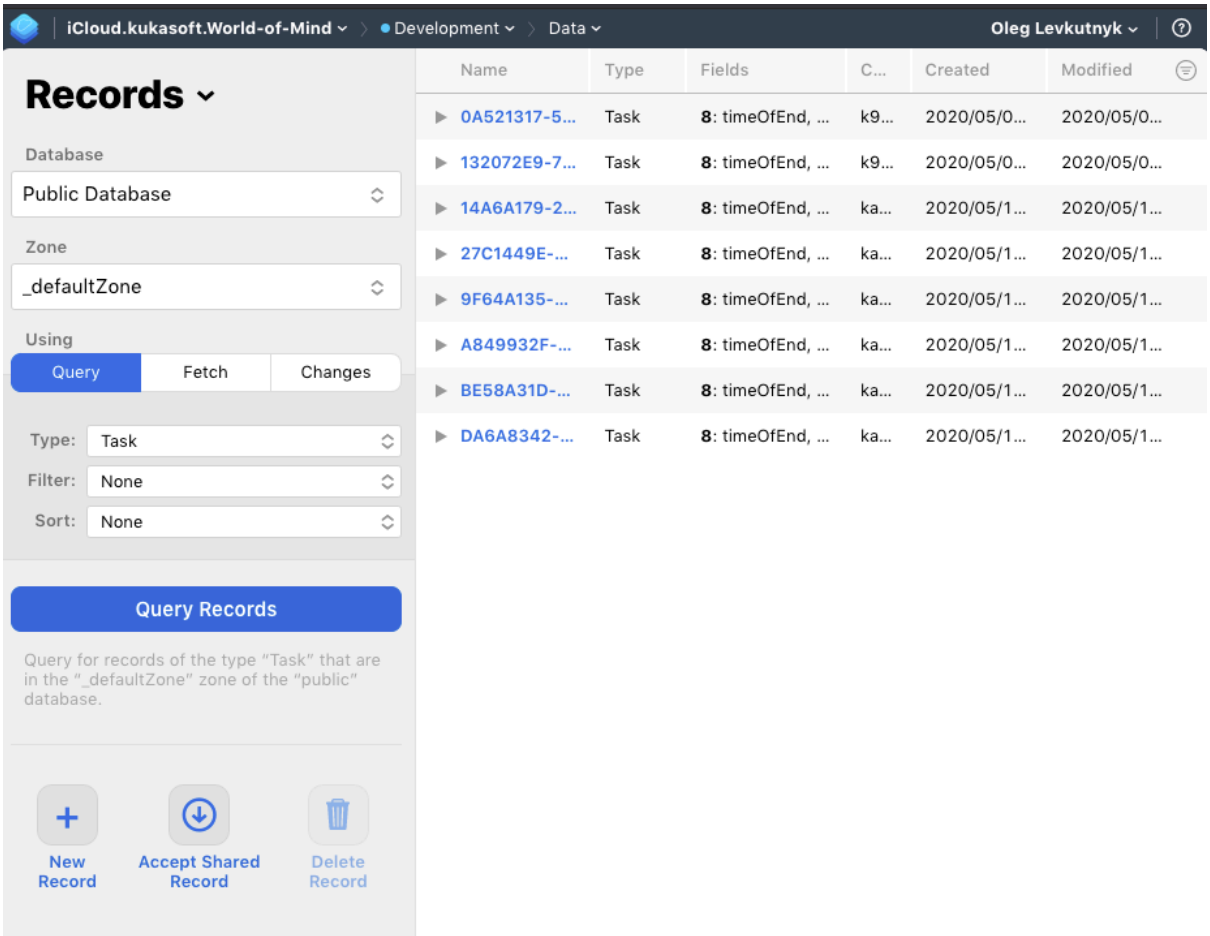

Рисунок 3.26 – Task iCloud

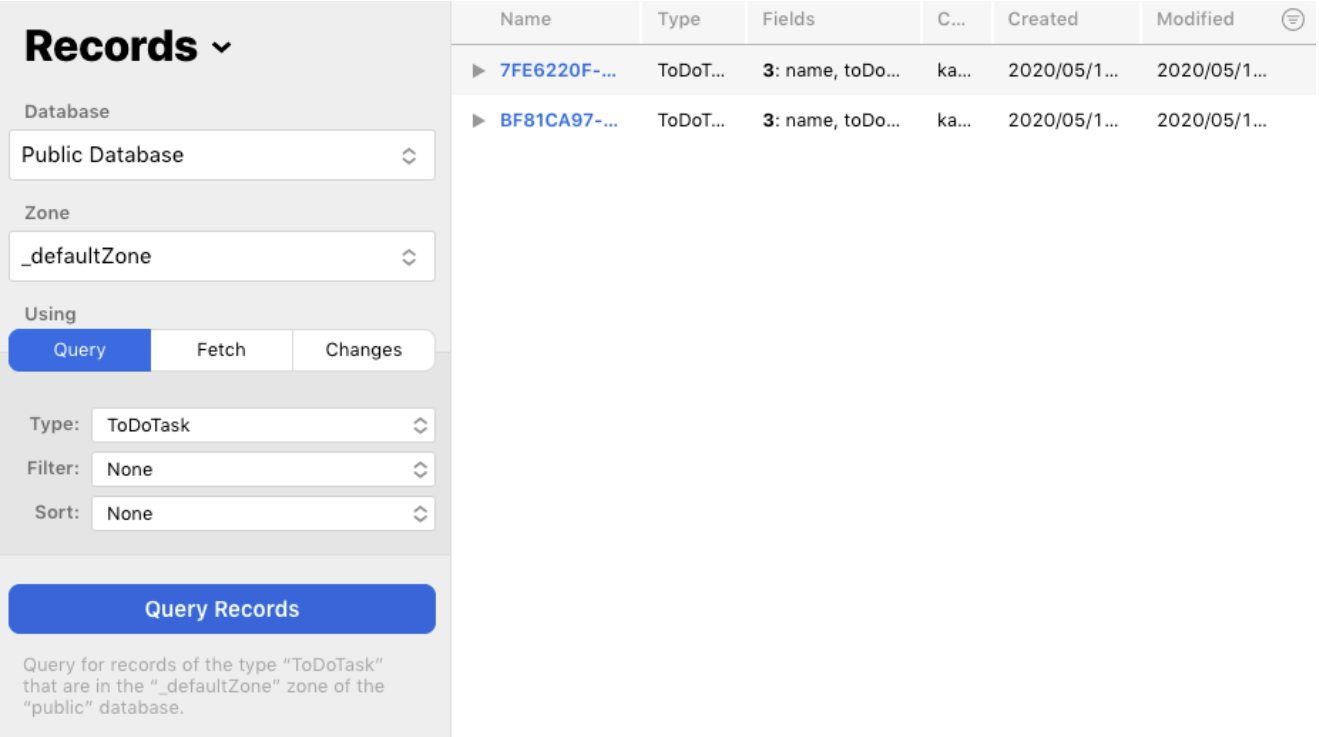

Рисунок 3.27 – ToDoTask iCloud

При запуску програми у випадку розсинхронізації даних, дані з хмари будуть автоматично отримані а таблиці для їх відображення будуть оновлені

Проект та його код можна знайти на GitHub: https://github.com/Kentukvel/World-of-Mind

ВИСНОВКИ

У ході виконання випускної роботи проведені дослідження по застосуванню багатопарадигмової компільованої мові програмування Swift в розробці iOS додатків. Створено інформаційне та програмне забезпечення мобільного додатку “RHand”, який виконує функції особистого менеджеру повсякденних подій, що відповідає завданню випускної роботи. Мобільний додаток “RHand” може бути встановлений на будь-який iPhone з версією iOS не раніше 10.0. Програма має можливість підключення до хмарного сервісу iCloud і дані користувача зберігатимуться у їх власному сховищі. Мобільний додаток пройшов тестування на iPhone SE а також у програмі Simulator. У ході тестування додатку помилок не виявлено

СПИСОК ЛІТЕРАТУРИ

1. Про мову Swift[Електронний ресурс] – Режим доступу до ресурсу: https://swiftbook.ru/content/swift-tour/about-swift/ – дата доступу: 25.04.2020

2. Документація мови Swift [Електронний ресурс] – Режим доступу до ресурсу: https://swiftbook.ru/contents/doc/ – дата доступу: 11.03.2020

3. Курси програмування на iOS від сайту swiftbook [Електронний ресурс] – Режим доступу до ресурсу: https://swiftbook.ru/courses/ – дата доступу: 25.03.2020

4. Курси програмування на iOS на сайті Udemy від Angela Yu [Електронний ресурс] – Режим доступу до ресурсу: https://www.udemy.com/course/ios11-app-development-bootcamp/ - дата доступу: – дата доступу: 14.03.2020

5. SwiftUI Essentials – iOS Edition: Learn to Develop iOS Apps using SwiftUI, Swift 5 and Xcode 11 Kindle Edition – Н. Сміт, 2019. – 430 с.

ДОДАТКИ

Додаток А

Клас **CloudManager**

```swift
//
//  CloudManager.swift
//  World of Mind
//
//  Created by Левкутник Дмитрий on 4/28/20.
//  Copyright © 2020 Левкутник Дмитрий. All rights reserved.
//

import UIKit
import CloudKit
import CoreData

class CloudManager {

    private static let privateCloudDatabase = CKContainer.default().privateCloudDatabase
    //private static let publicDatabase = CKContainer.default().publicCloudDatabase

    static func saveDataToCloud(toDoList: ToDoList, closure: @escaping (String) -> ()) {
        let record = CKRecord(recordType: "ToDoList")
        record.setValue(toDoList.dateOfLastUpdate, forKey: "dateOfLastUpdate")
        record.setValue(toDoList.name, forKey: "name")


        privateCloudDatabase.save(record) { (newRecord, error) in
            if let error = error { print(error); return }
            if let newRecord = newRecord {
                closure(newRecord.recordID.recordName)
            }

        }
    }

    static func saveDataToCloud(finance: Finance, closure: @escaping (String) -> ()) {
        let record = CKRecord(recordType: "Finance")

        record.setValue(finance.cost, forKey: "cost")
        record.setValue(finance.name, forKey: "name")
        record.setValue(finance.date, forKey: "date")
        record.setValue(finance.profit ? 1 : 0, forKey: "profit")


        privateCloudDatabase.save(record) { (newRecord, error) in
            if let error = error { print(error); return }
            if let newRecord = newRecord {
                closure(newRecord.recordID.recordName)
            }

        }
    }
```

```
51      static func saveDataToCloud(toDoTask: ToDoTask,
52                                  for list: ToDoList,
53                                  closure: @escaping (String) -> ()) {
54          let record = CKRecord(recordType: "ToDoTask")
55          record.setValue(toDoTask.done ? 1 : 0, forKey: "done")
56          record.setValue(toDoTask.name, forKey: "name")
57          record.setValue(list.id, forKey: "toDoListID")
58
59          privateCloudDatabase.save(record) { (newRecord, error) in
60              if let error = error { print(error); return }
61              if let newRecord = newRecord {
62                  closure(newRecord.recordID.recordName)
63              }
64
65          }
66      }
67
68      static func saveDataToCloud(task: Task, closure: @escaping (String) -> ()) {
69          let record = CKRecord(recordType: "Task")
70          record.setValue(task.notes, forKey: "notes")
71          record.setValue(task.name, forKey: "name")
72          record.setValue(task.notification ? 1 : 0, forKey: "notification")
73          record.setValue(task.notificationIdentifier, forKey: "notificationIdentifier")
74          record.setValue(task.repeatFrequency, forKey: "repeatFrequency")
75          record.setValue(task.timeOfEnd, forKey: "timeOfEnd")
76          record.setValue(task.timeOfStart, forKey: "timeOfStart")
77          record.setValue(task.url, forKey: "url")
78
79          privateCloudDatabase.save(record) { (newRecord, error) in
80              if let error = error { print(error); return }
81              if let newRecord = newRecord {
82                  closure(newRecord.recordID.recordName)
83              }
84
85          }
86      }
87
88      static func saveDataToCloud(diaryRecord: Diary, with image: UIImage,
89                                  clouser: @escaping (String) -> ()) {
90
91          let (image, url) = prepareImageToSaveToCloud(record: diaryRecord, image: image)
92
93          guard let imageAsset = image, let imageURL = url else { return }
94
```

```
95          let record = CKRecord(recordType: "Diary")
96          record.setValue(diaryRecord.recordName, forKey: "name")
97          record.setValue(diaryRecord.recordText, forKey: "text")
98          record.setValue(diaryRecord.recordTime, forKey: "time")
99          record.setValue(diaryRecord.hasImage ? 1 : 0, forKey: "hasImage")
100         record.setValue(imageAsset, forKey: "imageData")
101
102         privateCloudDatabase.save(record) { (newRecord, error) in
103             if let error = error { print(error); return }
104             if let newRecord = newRecord {
105                 clouser(newRecord.recordID.recordName)
106             }
107             deleteTempImage(imageURL: imageURL)
108         }
109     }
110
111     static func fetchDataFromCloud(diaryRecords: [Diary],
112                                    closure: @escaping (DiaryModel) -> ()) {
113
114         let query = CKQuery(recordType: "Diary", predicate: NSPredicate(value: true))
115         query.sortDescriptors = [NSSortDescriptor(key: "name", ascending: true)]
116
117         privateCloudDatabase.perform(query, inZoneWith: nil) { (records, error) in
118
119             guard error == nil else { print(error!); return }
120             guard let records = records else { return }
121
122             records.forEach({ (record) in
123                 let newDiaryRecord = DiaryModel(record: record)
124
125                 DispatchQueue.main.async {
126                  if self.newCloudRecordIsAvailable(diaryRecords: diaryRecords,
127                                                    recordID: newDiaryRecord.id) {
128                         closure(newDiaryRecord)
129                     }
130                 }
131             })
132         }
133     }
134
135     static func updateCloudData(diaryRecord: Diary, with image: UIImage) {
136
137         let recordID = CKRecord.ID(recordName: diaryRecord.id!)
138
139         let (image, url) = prepareImageToSaveToCloud(record: diaryRecord, image: image)
140         guard let imageAsset = image, let imageURL = url else { return }
141
```

```
142         privateCloudDatabase.fetch(withRecordID: recordID) { (record, error) in
143
144             if let record = record, error == nil {
145
146                 DispatchQueue.main.async {
147                     record.setValue(diaryRecord.recordName, forKey: "name")
148                     record.setValue(diaryRecord.recordText, forKey: "text")
149                     record.setValue(diaryRecord.hasImage ? 1 : 0, forKey: "hasImage")
150                     record.setValue(diaryRecord.recordTime, forKey: "time")
151                     record.setValue(imageAsset, forKey: "imageData")
152
153                     privateCloudDatabase.save(record, completionHandler: { (_, error) in
154                         if let error = error { print(error.localizedDescription); return }
155                         deleteTempImage(imageURL: imageURL)
156                     })
157                 }
158             }
159         }
160     }
161
162     static func fetchDataFromCloud(toDoTasks: [ToDoTask],
163                                    closure: @escaping (ToDoTaskModel) -> ()) {
164
165         let query = CKQuery(recordType: "ToDoTask",
166                             predicate: NSPredicate(value: true))
167         query.sortDescriptors = [NSSortDescriptor(key: "name", ascending: true)]
168
169         privateCloudDatabase.perform(query, inZoneWith: nil) { (records, error) in
170
171             guard error == nil else { print(error!); return }
172             guard let records = records else { return }
173
174             records.forEach({ (record) in
175                 ///
176
177
178                 let doneBool = record.value(forKey: "done") as! Int
179
180                 let done = doneBool == 1 ? true : false
181                 let name = record.value(forKey: "name") as! String
182                 let parantID = record.value(forKey: "toDoListID") as! String
183                 let id = record.recordID.recordName
184
185                 let newTask = ToDoTaskModel(name: name, id: id,
186                                             parantID: parantID, done: done)
187
```

```
188                 DispatchQueue.main.async {
189                     if self.newCloudRecordIsAvailable(toDoTasks: toDoTasks,
190                                                       toDoTaskID: newTask.id) {
191                         closure(newTask)
192                     }
193                 }
194             })
195         }
196     }
197
198     static func fetchDataFromCloud(toDoLists: [ToDoList],
199                                    closure: @escaping (ToDoListModel) -> ()) {
200
201         let query = CKQuery(recordType: "ToDoList", predicate: NSPredicate(value: true))
202         query.sortDescriptors = [NSSortDescriptor(key: "name", ascending: true)]
203
204         privateCloudDatabase.perform(query, inZoneWith: nil) { (records, error) in
205
206             guard error == nil else { print(error!); return }
207             guard let records = records else { return }
208             records.forEach({ (record) in
209
210
211
212                 ///////
213                 let name = record.value(forKey: "name") as! String
214                 let id = record.recordID.recordName
215                 let dateOfLastUpdate = record.value(forKey: "dateOfLastUpdate") as! Date
216                 let newToDoList = ToDoListModel(name: name,
217                                                 id: id,
218                                                 dateOfLastUpdate: dateOfLastUpdate)
219                 DispatchQueue.main.async {
220                     if self.newCloudRecordIsAvailable(toDoLists: toDoLists,
221                                                       toDoListID: newToDoList.id) {
222                         print("done")
223                         closure(newToDoList)
224                     }
225                 }
226             })
227         }
228     }
229
230     static func fetchDataFromCloud(tasks: [Task], closure: @escaping (TaskModel) -> ()) {
231         let query = CKQuery(recordType: "Task", predicate: NSPredicate(value: true))
232         query.sortDescriptors = [NSSortDescriptor(key: "name", ascending: true)]
233
234         privateCloudDatabase.perform(query, inZoneWith: nil) { (records, error) in
235
```

```
236             guard error == nil else { print(error!); return }
237             guard let records = records else { return }
238             records.forEach({ (record) in
239
240
241
242                 ///////
243
244                 let newTask = TaskModel(record: record)
245                 DispatchQueue.main.async {
246                     if self.newCloudRecordIsAvailable(tasks: tasks, taskID: newTask.id) {
247                         pri Set the active scheme
248                         closure(newTask)
249                     }
250                 }
251             })
252         }
253     }
254
255     static func fetchDataFromCloud(finances: [Finance],
256                                    closure: @escaping (FinanceModel) -> ()) {
257         let query = CKQuery(recordType: "Finance", predicate: NSPredicate(value: true))
258         query.sortDescriptors = [NSSortDescriptor(key: "name", ascending: true)]
259
260         privateCloudDatabase.perform(query, inZoneWith: nil) { (records, error) in
261
262             guard error == nil else { print(error!); return }
263             guard let records = records else { return }
264             records.forEach({ (record) in
265
266
267
268                 ///////
269
270                 let newFinance = FinanceModel(record: record)
271                 DispatchQueue.main.async {
272                     if self.newCloudRecordIsAvailable(finances: finances,
273                                                       financeID: newFinance.id) {
274
275                         closure(newFinance)
276                     }
277                 }
278             })
279         }
280     }
281
```

```
282     private static func newCloudRecordIsAvailable(finances: [Finance],
283                                                   financeID: String) -> Bool {
284         for finance in finances {
285             if finance.id == financeID {
286                 return false
287             }
288         }
289         return true
290     }
291
292     private static func newCloudRecordIsAvailable(tasks: [Task],
293                                                   taskID: String) -> Bool {
294
295         for task in tasks {
296             if task.id == taskID {
297                 return false
298             }
299         }
300
301         return true
302     }
303
304     private static func newCloudRecordIsAvailable(toDoTasks: [ToDoTask],
305                                                   toDoTaskID: String) -> Bool {
306
307         for toDoTask in toDoTasks {
308             if toDoTask.id == toDoTaskID {
309                 return false
310             }
311         }
312
313         return true
314     }
315
316     private static func newCloudRecordIsAvailable(diaryRecords: [Diary],
317                                                   recordID: String) -> Bool {
318
319         for record in diaryRecords {
320             if record.id == recordID {
321                 return false
322             }
323         }
324
325         return true
326     }
```

```
328     private static func newCloudRecordIsAvailable(toDoLists: [ToDoList],
329                                                   toDoListID: String) -> Bool {
330
331         for toDoList in toDoLists {
332             if toDoList.id == toDoListID {
333                 return false
334             }
335         }
336         return true
337     }
338
339     static func updateCloudData(toDoTask: ToDoTask) {
340         let recordID = CKRecord.ID(recordName: toDoTask.id!)
341
342         privateCloudDatabase.fetch(withRecordID: recordID) { (record, error) in
343
344             if let record = record, error == nil {
345
346                 DispatchQueue.main.async {
347                     record.setValue(toDoTask.done ? 1 : 0, forKey: "done")
348
349
350                     privateCloudDatabase.save(record, completionHandler: { (_, error) in
351                         if let error = error { print(error.localizedDescription)
352                             return
353
354                         }
355                     })
356                 }
357             }
358         }
359     }
360
361     static func updateCloudData(task: Task) {
362
363         let recordID = CKRecord.ID(recordName: task.id!)
364
365         privateCloudDatabase.fetch(withRecordID: recordID) { (record, error) in
366             if let record = record, error == nil {
367
368                 DispatchQueue.main.async {
369                     record.setValue(task.name, forKey: "name")
370                     record.setValue(task.notes, forKey: "notes")
371                     record.setValue(task.notification ? 1 : 0, forKey: "notification")
372                     record.setValue(task.notificationIdentifier,
373                                     forKey: "notificationIdentifier")
374                     record.setValue(task.repeatFrequency, forKey: "repeatFrequency")
375                     record.setValue(task.timeOfEnd, forKey: "timeOfEnd")
```

```
423             privateCloudDatabase.add(queryOperation)
424         }
425
426         // MARK: Private Methods
427         private static func prepareImageToSaveToCloud(record: Diary,
428                                                       image: UIImage) -> (CKAsset?, URL?) {
429
430             let scale = image.size.width > 1080 ? 1080 / image.size.width : 1
431             let scaleImage = UIImage(data: image.pngData()!, scale: scale)
432             let imageFilePath = NSTemporaryDirectory() + record.recordName!
433             let imageURL = URL(fileURLWithPath: imageFilePath)
434
435             guard let dataToPath = scaleImage?.jpegData(compressionQuality: 1)
436                 else { return (nil, nil)}
437
438             do {
439                 try dataToPath.write(to: imageURL, options: .atomic)
440             } catch {
441                 print(error.localizedDescription)
442             }
443
444             let imageAsset = CKAsset(fileURL: imageURL)
445
446             return (imageAsset, imageURL)
447         }
448
449         static private func deleteTempImage(imageURL: URL) {
450             do {
451                 try FileManager.default.removeItem(at: imageURL)
452             } catch {
453                 print(error.localizedDescription)
454             }
455         }
456     }
457
```

```
                    record.setValue(task.timeOfStart, forKey: "timeOfStart")
                    record.setValue(task.url, forKey: "url")

                    privateCloudDatabase.save(record) { (_, error) in
                        if let error = error {
                            print(error.localizedDescription); return
                        }
                    }
                }
            }
        }
    }

    static func deleteRecord(recordID: String, type: ModelClass) {
        var recordType = ""
        switch type {
        case .toDoList:
            recordType = "ToDoList"
        case .toDoTask:
            recordType = "ToDoTask"
        case .task:
            recordType = "Task"
        case .finance:
            recordType = "Finance"
        case .diary:
            recordType = "Diary"
        }
        let query = CKQuery(recordType: recordType, predicate: NSPredicate(value: true))
        let queryOperation = CKQueryOperation(query: query)
        queryOperation.desiredKeys = ["recordID"]
        queryOperation.queuePriority = .veryHigh

        queryOperation.recordFetchedBlock = { record in
            if record.recordID.recordName == recordID {
                privateCloudDatabase.delete(withRecordID: record.recordID) { (_, error) in
                    if let error = error {
                        print(error.localizedDescription)
                        return
                    }
                }
            }
        }
        queryOperation.queryCompletionBlock = { (_, error) in
            if let error = error {
                print(error.localizedDescription)
            }
        }
        privateCloudDatabase.add(queryOperation)
```