МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Розробка iOS-додатку для контролю калорійності дієти**

Виконав: студент 4 курсу, групи 6.1210-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
Д. Р. Нєстєров
(ініціали та прізвище)
Керівник к.т.н., доцент, доцент кафедри ЕІС та ПЗ
Н. П. Полякова
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»
П. О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя

2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра Електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти перший(бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Т. В. Критська
“ 01 ” 2024 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Нєстєрову Денису Романовичу
(прізвище, ім’я, по батькові)

1.Тема роботи Розробка iOS-додатку для контролю калорійності дієти

керівник роботи Полякова Наталія Петрівна, к.т.н., доцент
(прізвище ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від 26.12.2023 № 2215-с

2. Строк подання студентом кваліфікаційної роботи 14.06.2024
3. Вихідні дані кваліфікаційної роботи бакалавра
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- створення програмного продукту та його опис;
- дослідження поставленої проблеми та розробка висновків.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Аналіз предметної області	03.01 – 25.01.24	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	26.01 – 29.01.24	виконано
3	Аналіз існуючих методів рішення	01.01 – 07.02.24	виконано
4	Огляд та збір літератури стосовно теми кваліфікаційної роботи	08.02 – 22.02.24	виконано
5	Огляд та аналіз існуючих рішень та аналогів	23.02 – 28.02.24	виконано
6	Аналіз сучасних технологій для створення інтерфейсу застосунку	01.03 – 07.03.24	виконано
7	Аналіз Firebase СУБД	08.03 – 13.03.24	виконано
8	Огляд Edamam API	14.03 – 18.03.24	виконано
9	Опрацювання функціональних вимог до системи	19.03 – 26.03.24	виконано
10	Розробка архітектури системи та проєктування додатку	27.03 – 23.04.24	виконано
11	Програмна реалізація застосунку та його апробація	24.04 – 10.05.24	виконано
12	Оформлення звіту	11.05 – 29.05.24	виконано

Студент _____ Нєстєров Д. Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Полякова Н.П.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 69

Рисунків — 19

Таблиць — 1

Джерел — 30

Нєстєров Д. Р. Розробка iOS-додатку для контролю калорійності дієти: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Н. П. Полякова. Запоріжжя : ЗНУ, 2024. 69 с.

У даній роботі здійснюється аналіз технологій Swift, UIKit, а також можливостей Edamam API та Firebase Firestore щодо розробки iOS-додатку для контролю калорійності дієти. Розглядаються ключові функціональні можливості Edamam API для отримання інформації про продукти та їх калорійність, а також Firebase Firestore для зберігання даних користувачів та їх дієтичних показників.

На основі здобутих знань і аналізу сучасних тенденцій у галузі здорового харчування створюється архітектура iOS-додатку з використанням Swift та UIKit. Програмна реалізація включає в себе функціонал для обліку спожитих калорій та створення дієтичних планів спираючись на обрану ціль.

Здійснено тестування працездатності створеного iOS-додатку. Проведено порівняння функціональності додатку з наявними ринковими рішеннями, а також оцінено задоволеність користувачів від використання додатку для контролю калорійності дієти.

Ключові слова: *Edamam API, iOS-додаток, дієта, контроль дієти, здорове харчування, контроль калорійності*.

ABSTRACT

Pages — 69

Drawings — 19

Tables — 1

Sources — 30

Niestierov D. R. Development of an iOS Application for Diet Calorie Control: Bachelor's Thesis in Specialty 121 "Software Engineering" / Scientific Supervisor N. P. Polyakova. Zaporizhzhia: ZNU, 2024. 69 p.

This paper analyzes the technologies Swift, UIKit, and the capabilities of Edamam API and Firebase Firestore for developing an iOS application to control diet calorie intake. The key functional capabilities of the Edamam API for obtaining information about products and their caloric content, as well as Firebase Firestore for storing user data and their dietary metrics, are being examined.

Based on the acquired knowledge and analysis of current trends in the field of healthy nutrition, the architecture of the iOS application is developed using Swift and UIKit. The software implementation includes functionality for tracking consumed calories and creating dietary plans based on the selected goal.

The functionality of the developed iOS application was tested. A comparison was made with existing market solutions, and user satisfaction with the application for controlling diet calorie intake was evaluated.

Keywords: *Edamam API, iOS application, diet, diet control, healthy nutrition, calorie control*.

ЗМІСТ

ВСТУП

Актуальність теми

У сучасному світі кожна людина має доступ до необмеженого вибору продуктів харчування. Перш ніж продукція потрапить на прилавок магазину чи супермаркету, вона проходить безліч різних етапів контролю виробництва відповідно до загальноприйнятих норм якості та безпеки. Всі ці заходи спрямовані на те, щоб споживач отримав найкращий продукт який буде не тільки безпечним для здоров’я, а ще й смачним. Але навіть при врахуванні усіх цих факторів, розуміння важливості правильного харчування залишається необхідним для кожної людини.

Протягом останнього десятиліття відсоток людей із зайвою вагою значно збільшився. Це свідчить про те, що багато людей не дотримуються збалансованого раціону та не мають достатніх знань про потреби їхнього організму. Насправді це стає дуже важливим аспектом, оскільки безпосередньо впливає на наше здоров'я та загальне самопочуття [1]. Зараз, у наш цифровий вік, проблема здорового харчування стає більш доступною для самостійного вирішення завдяки створенню десктопних та мобільних додатків. Ці додатки спрощують процес початкового ознайомлення з темою збалансованого харчування, допомагаючи звичайному користувачеві швидко та ефективно знаходити інформацію про дієту, калорійність продуктів, методи схуднення, набору маси та інше. В їх функціонал можуть входити такі елементи як рекомендації, ведення журналів, нагадувати про приймання їжі та допомога з встановленням особистої цілі людини. Такий підхід робить здоровий спосіб життя більш досяжним та контрольованим для кожного.

Наразі майже у кожної людини є свій особистий мобільний пристрій. Це швидкий та зручний спосіб комунікації, розваг та навчання який завжди поряд. Отже, ринок мобільних технологій стрімко та безперервно розвивається, безперешкодно витісняючи усі десктопні застосунки [2].

Основними платформами мобільних додатків є Android та iOS, де Android займає 70 відсотків ринку, тоді як iOS — 29 відсотків за 2022 рік. За останні роки iOS відзначається стрімким зростом, демонструючи збільшення кількості нових користувачів майже на 10 відсотків порівняно з Android [3].

Підбиваючи підсумок, можна сказати що розробка мобільного додатку для відстеження раціону та калорійності на платформі iOS стає актуальною та цікавою темою.

Мета дослідження

Розробка iOS-додатку для контролю калорійності дієти.

Завдання дослідження

Вивчити існуючі рішення та аналоги додатків для контролю калорійності дієти. Проаналізувати сучасні підходи та технології, які використовуються для розробки додатків такого типу. Розробити прототип додатку та провести його тестування. Оцінити ефективність розробленого додатку та надати рекомендації для його подальшого вдосконалення.

Об’єкт дослідження

Об’єктом дослідження є процес розробки iOS-додатку для контролю калорійності дієти засобами сучасних технологій мобільної розробки.

Предмет дослідження

Предметом дослідження є технології створення власного мобільного додатку для контролю харчування, а також методи та підходи до розробки програмного забезпечення, що сприяють підвищенню ефективності роботи користувачів.

Методи дослідження

Теоретичні — аналіз існуючих технологій та підходів до створення мобільних додатків для контролю калорійності; порівняння різних методів розробки та їх ефективності.

Практичні — розробка та тестування прототипу додатку, порівняльний аналіз різних версій додатку для вибору оптимальних рішень.

Практичне знання одержаних результатів

Практична цінність результатів цього дослідження полягає у створенні iOS-додатку, який дозволяє користувачам зручно та ефективно контролювати калорійність своєї дієти. Додаток забезпечує простий спосіб відстеження калорійності спожитих продуктів, планування харчування та досягнення особистих цілей щодо ваги та здоров'я. Використання цього інструменту допомагає краще розуміти калорійний вміст їжі, підтримувати здорові харчові звички та покращувати загальний фізичний стан. Отримані результати дослідження можуть бути застосовані для подальшого розвитку додатку, розширення його функціональних можливостей та впровадження інновацій у сфері дієтології та здорового способу життя.

Апробація

Роботу було представлено на конференції «Молода наука-2024» [4].

Глосарій

Swift — мова програмування від Apple для розробки додатків на iOS, macOS, watchOS, tvOS.

UIKit — фреймворк для створення графічних інтерфейсів користувача в iOS-додатках.

Edamam API — інтерфейс для отримання даних про харчову цінність і продукти.

Firebase Firestore — хмарна база даних від Google для зберігання і синхронізації даних.

API ключ — унікальний ідентифікатор для автентифікації запитів до API.

Дієта — набір харчових продуктів для досягнення певних цілей.

Контроль калорійності — відстеження споживаних калорій для управління вагою.

Нутрієнт — речовина, необхідна організму для підтримки життя та нормального функціонування, включаючи білки, жири, вуглеводи, вітаміни, мінерали та воду.

BMR (Basal Metabolic Rate) — базовий рівень метаболізму, тобто кількість калорій, яку організм спалює в стані спокою для підтримання життєво важливих функцій.

TDEE (Total Daily Energy Expenditure) — сумарні денні енерговитрати організму, що включають калорії, спалені під час всіх видів активності та фізичних навантажень протягом дня.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд літературних джерел

Зростання надмірної ваги та ожиріння у всьому світі є значною загрозою для здоров'я населення. За даними Всесвітньої організації охорони здоров'я, у 2022 році понад 2,5 мільярда дорослих мали надмірну вагу, з яких 890 мільйонів страждали від ожиріння. Це явище стає причиною розвитку серйозних хронічних захворювань, таких як діабет, серцево-судинні захворювання та рак [5].

Підходи до контролю ваги

Для контролю ваги основними методами є дотримання дієт, підвищення фізичної активності та, в разі необхідності, використання лікарських засобів. Однак, для багатьох людей слідкувати за споживанням калорій є викликом. За дослідженням Американської асоціації дієтологів, яке було проведено у 2022 році, значна частина населення неправильно оцінює свої калорійні потреби, що ускладнює підтримку здорової ваги [6]. Тому розробка мобільних додатків для контролю калорійності дієти може значно полегшити цей процес.

Мобільні додатки набувають популярності через їхню зручність та ефективність. Вони надають користувачам інформацію про калорійність їжі, допомагають відслідковувати харчування та фізичну активність, а також мотивують досягати поставлених цілей. Дослідження Університету Північної Кароліни показало, що користувачі, які активно використовують такі додатки, досягають кращих результатів у контролі ваги та покращенні загального стану здоров'я [7]. Аналогічні результати отримано в дослідженні Університету Оксфорда у 2023 році, яке підтвердило, що мобільні додатки є ефективним інструментом для зниження ваги [8].

Ефективність різних підходів до контролю ваги

Різноманітні дослідження, проведені в різних країнах, підтверджують ефективність поєднання дієти та фізичної активності для зниження ваги. Наприклад, дослідження, опубліковане в Journal of the American Medical Association, показало, що комбінування низькокалорійної дієти з регулярними фізичними навантаженнями є одним з найефективніших способів зниження ваги [9]. Інше дослідження, опубліковане в The Lancet, виявило, що середземноморська дієта у поєднанні з фізичною активністю значно покращує стан серцево-судинної системи та сприяє зниженню ваги [10].

Висновок

Комплексний підхід до контролю ваги, що включає дієти, фізичну активність, мобільні додатки та, за необхідності, лікарські засоби, є найефективнішим. Мобільні додатки, зокрема, роблять процес контролю калорійності дієти більш доступним та ефективним для широкого кола користувачів. Використання таких технологій може допомогти досягти значних успіхів у боротьбі з надмірною вагою та покращенні загального здоров'я населення.

1.2 Аналіз програмних продуктів-аналогів

На поточному ринку мобільних додатків є немало застосунків темою яких є стеження за харчуванням користувача, проте лише кілька з них є дійсно особливими та зручними в користуванні. Ці програми допомагають користувачам відстежувати калорії, нутрієнти та інші важливі показники особистого харчування.

Проаналізуємо деякі з них, а саме MyFitnessPal, Lose It!, Lifesum.

MyFitnessPal

MyFitnessPal — популярний додаток для відстеження харчування та фізичної активності, що надає калькулятор калорій, живий журнал харчування та спільноту для підтримки здоров'я та фітнесу [11]. Додаток містить багато корисних функцій, які роблять його популярним серед користувачів, надаючи при цьому корисні інструменти для аналізу та контролю.

Інтерфейс додатку можемо побачити на рисунку 1.

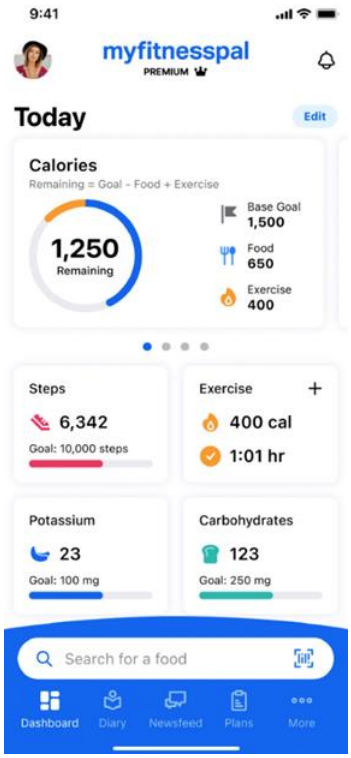

Рисунок 1 — *Інтерфейс MyFitnessPal*

Переваги:

1. Пропонує об’ємну базу даних продуктів для споживання, яка забезпечує легкий доступ до інформації про спожиті користувачем страви.
2. Надає можливість докладно вести журнал спожитих продуктів і витрачених користувачем калорій, а також записувати в журнал свої фізичні активності.

3. Взаємодіє з різноманітними фітнес-трекерами, автоматично синхронізує дані про фізичну активність та забезпечує їх взаємодію.
4. Можливість користувачів переглядати графіки та статистику щодо змін у власній вазі, спожиття та витрачення калорій та інших характеристик.
5. Можливість відстеження макроелементів.

Недоліки:

1. Наявність пасивної реклами.
2. Обмежені функції у безкоштовній версії додатку які доступні тільки за окрему плату.
3. База даних продуктів може містити помилки чи неповну інформацію, що може впливати на точність обліку калорій.

Lose It!

Lose It! — додаток для відстеження харчування та фітнесу, який пропонує калькулятор калорій, журнал харчування, спільноту користувачів та інструменти для досягнення особистих цілей фітнесу [12]. Додаток надає широкий спектр функцій, що допомагають користувачам підтримувати здоровий спосіб життя та контролювати свій прогрес.

Переваги:

1. Аналізує дані користувача і надає персоналізовані рекомендації з харчування та фізичної активності для досягнення конкретних цілей.
2. Користувачі можуть об'єднуватися в групи та брати участь в змаганнях.
3. Можливість моніторингу рівня глюкози та інших показників здоров'я.
4. Можливість швидкого додавання продуктів до журналу шляхом сканування штрих-кодів, що спрощує процес обліку харчування.

Недоліки:

1. У порівнянні з іншими додатками, база даних продуктів може бути менш обширною та оновлюватися не так швидко.

2. Деякі функції можуть бути обмежені або недоступні на певних пристроях чи операційних системах.

Інтерфейс додатку зображено на рисунку 2.

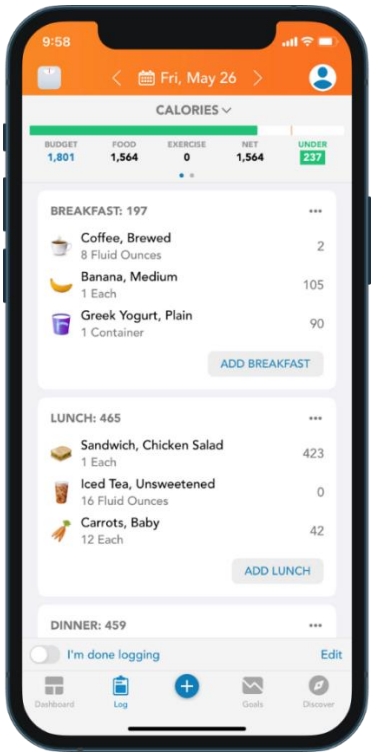

Рисунок 2 — *Інтерфейс Lose It!*

Lifesum

Lifesum — інноваційний додаток для підтримки здорового способу життя, який вирізняється своїм унікальним підходом до підтримки користувачів у досягненні їхніх цілей здоров'я та фітнесу, забезпечуючи персоналізовані поради для таких ключових аспектів життя, як сон, стрес та гідратація [13]. Додаток надає комплексний підхід до здоров'я, інтегруючи різні аспекти життя користувачів для досягнення оптимальних результатів.

Інтерфейс додатку Lifesum можна побачити на рисунку 3.

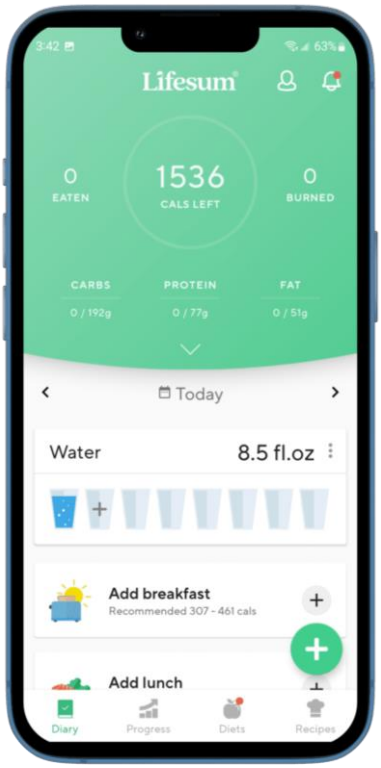

Рисунок 3 — *Інтерфейс Lifesum*

Переваги:

1. Вирізняється сучасним та інтуїтивним дизайном інтерфейсу, що робить користування приємним та легким.
2. Аналізує індивідуальні харчові та фізичні потреби, пропонуючи персоналізовані поради та рекомендації для досягнення конкретних цілей.
3. Пропонує відстеження не лише харчування та фізичної активності, але й інших аспектів, таких як сон, стрес та гідратація.
4. Пропонує обширні можливості соціальної взаємодії, такі як: об'єднання у групи для взаємного підтримання та викликів, групові виклики, активні форуми.

Недоліки:

1. У деяких регіонах може відсутність деяких місцевих чи регіональних харчових продуктів у базі даних.

2. На деяких пристроях може спостерігатися неоптимальна робота додатку, що викликає певні труднощі для користувачів цих пристроїв.
3. Інтеграція з фітнес-трекерами тільки у платній версії.

Порівняння характеристик цих аналогів та результати аналізу наведено в таблиці 1.

Таблиця 1 — *Порівняння характеристик MyFitnessPal, Lose It! та Lifesum*

Властивість	**MyFitnessPal**	**Lose it!**	**Lifesum**
База даних	Обширна	Обширна	Задовільна
Спільнота	Обмежена; форум з обмеженою взаємодією	Задовільна; групи для обговорення	Обширна; групи для обговорення, групові виклики, форуми
Базовий функціонал	Калькулятор, журнал	Калькулятор, журнал	Калькулятор, журнал, відстеження сну та стресу
Персоналізація та аналітика	Високий рівень	Високий рівень	Високий рівень
Інтеграція	Широкі можливості	Широкі можливості	Тільки у платній версії
Підтримка мов	Наявна, обширна	Наявна, обширна	Наявна, обширна
Вартість	Безкоштовний	Безкоштовний	Безкоштовний
Платна підписка	Наявна, $9.99/місяць	Наявна, $3.33/місяць	Наявна, $3.99/місяць
Функції платної підписки	Розширена статистика, додатковий функціонал	Розширена статистика	Додатковий функціонал, наявна інтеграція

Згідно з результатами порівняння кращих додатків на ринку мобільних додатків для iOS, а саме MyFitnessPal, Lose it! та Lifesum, основними темами яких є схуднення, дієта та фітнес, можна зробити висновок про необхідність створення

власної системи яка забезпечить стабільне та зручне відстеження свого раціону та калоражу.

1.3 Постановка задачі

Сьогодні мобільні додатки відіграють важливу роль у багатьох аспектах нашого життя, допомагаючи не тільки у комунікації та розвагах, але й у підтримці здорового способу життя. У світі, де технології стрімко розвиваються, особливо важливо мати доступ до інструментів, які дозволяють ефективно контролювати своє харчування та фізичну активність. Саме з цією метою стоїть задача розробки зручного, сучасного та швидкого мобільного додатку, що надає користувачам можливість персоналізовано стежити за своїм раціоном та досягати поставлених цілей.

Основні задачі, які необхідно вирішити під час розробки додатку, включають:

1. Вхід та реєстрація користувачів:

Забезпечення можливості увійти за допомогою існуючого облікового запису або створити новий. Процес має бути розподілений на кілька екранів: головна сторінка для вибору способу входу, сторінка входу з можливістю відновлення паролю та сторінка реєстрації.

2. Калькулятор калорій:

Створення можливості введення особистих даних користувача, таких як вік, зріст, вага, рівень фізичної активності та стать. Крім того, необхідно забезпечити вибір бажаної цілі: схуднення, набір ваги чи підтримка поточної ваги для розрахунку якої повинен використовуватися відповідний алгоритм для автоматичного розрахунку необхідного калоражу з використанням визначених формул та надання деяких рекомендацій для досягнення цілі.

3. Відстеження споживаних продуктів:

Додаток повинен мати у собі функції ведення щоденного журналу раціону, що дозволить користувачам вводити інформацію про спожиті продукти та напої. Має використовуватися пошуковий елемент для введення назви продукту та відображення списку відповідних продуктів з деталями про білки, жири та вуглеводи. Також, має бути можливість додавання продуктів до корзини, зміни кількості та оновлення списку спожитих продуктів.

4. Інтерфейс користувача:

Розробка має на меті інтуїтивно зрозумілий та зручний інтерфейс, що дозволить користувачам легко взаємодіяти з додатком. Важливо забезпечити логічну структуру навігації та привабливий дизайн.

5. Безпека даних:

Забезпечення захисту особистих даних користувачів шляхом використання сучасних методів шифрування та автентифікації. Розробка політики конфіденційності та заходів для забезпечення безпеки даних.

6. Точність розрахунків:

Використання перевірених формул для розрахунку калоражу та інших показників, що гарантує точність та надійність результатів. Потрібно провести тестування алгоритмів та забезпечити їх відповідність встановленим стандартам.

7. Адаптивність та масштабованість:

При розробці додатку необхідно ураховувати можливість його подальшого розширення та адаптації до нових функціональних вимог, забезпечити можливість легкого додавання нових функцій та модулів.

Таким чином, завдання розробки додатку для контролю калорійності дієти включає визначення ключових функцій, технічних рішень та забезпечення зручності використання, що в сукупності створює ефективний інструмент для покращення способу життя користувачів.

2 ДОСЛІДЖЕННЯ ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ДОДАТКУ

2.1 Огляд мов програмування

Для розробки додатку на платформі iOS можна використовувати кілька мов програмування, кожна з яких має свої переваги та особливості. Основні мови програмування, які підтримуються для створення iOS-додатків, включають Objective-C та Swift. Розглянемо детальніше кожну з них, щоб визначити, яка з мов найкраще підходить для розробки сучасного мобільного додатку для контролю калорійності дієти.

Objective-C

Objective-C — це одна з перших мов програмування, яку Apple використовувала для створення застосунків на платформах macOS, iOS, watchOS та tvOS. Вона була розроблена у 1983 році Бредом Коксом та є розширенням мови програмування C [14]. Мова програмування втілює об'єктно-орієнтований підхід, що дозволяє створювати потужні та ефективні програми, використовуючи ключові концепції ООП, але у деяких моментах розробки використовує характерно застарілі підходи.

Основні риси Objective-C:

1. Має свій власний специфічний синтаксис, який іноді вважається менш зручним порівняно з іншими сучасними мовами програмування.
2. Використовує синтаксис мови програмування C та впроваджує об'єктно-орієнтований підхід.
3. Дозволяє розробникам створювати потужні та ефективні програми за допомогою ключових об'єктно-орієнтованих концепцій, таких як спадкування, поліморфізм та інкапсуляція.

4. Використовує механізм автоматичного підрахунку посилань для зручного управління пам'яттю, полегшуючи уникання витоку пам'яті та ручного визначення моменту видалення об'єктів.
5. Дозволяє розширювати функціональність класів за допомогою категорій та використання протоколів для визначення набору методів.

Swift

Swift — це сучасна та інноваційна мова програмування, яка була розроблена компанією Apple для розробки широкого спектру програм для macOS, iOS, watchOS, та tvOS. Вона була представлена у 2014 році і виступила як сучасна заміна Objective-C [15]. Основною метою Swift було полегшення розробки програм та покращення безпеки коду.

Основні риси Swift:

1. Має відповідний сучасним стандартам синтаксис, який робить код більш зрозумілим та легко-читабельним, використовуючи безперервні ключові слова та типізацію.
2. Підтримує основні концепції об'єктно-орієнтованого програмування, такі як спадкування, інкапсуляція та поліморфізм, що сприяє розробці модульного та чіткого коду.
3. Надає високу безпеку коду завдяки вбудованим механізмам, таким як безпечні опціонали та безпечні інтерполяції рядків, що допомагає уникати багатьох помилок.
4. Завдяки своїй структурі, Swift полегшує командну роботу над проєктами, роблячи код більш зрозумілим і легким для підтримки.
5. Розробникам надається можливість використовувати інтерактивні консолі, що дозволяють експериментувати, тестувати код без необхідності компіляції всього проєкту.

6. Підтримує функціональні програмувальні парадигми, такі як замикання, вищі функції та неявні функції, що робить мову більш гнучкою та потужною.
7. Використовує автоматичну збірку сміття, що значно полегшує управління пам'яттю та зменшує ймовірність витоку пам'яті.

Беручи до уваги представлену інформацію, можна сказати що у 2024 році вибір Swift для розробки додатків для платформ Apple буде кращим вибором через його активний розвиток, безпеку коду, сучасно-адаптований синтаксис та широкі можливості для реалізації як об'єктно-орієнтованих, так і функціональних концепцій програмування. Саме через ці критерії мова програмування Swift була обрана для написання застосунку.

2.2 Огляд інструментів розробки

При розробці iOS-додатків важливу роль відіграє вибір інструментів, які, в свою чергу, дозволяють ефективно створювати інтерфейси користувачів та забезпечувати взаємодію з додатком. На платформі iOS доступні декілька потужних фреймворків, кожен з яких пропонує різні підходи та можливості для реалізації функціоналу. Основні фреймворки для розробки інтерфейсів включають UIKit та SwiftUI. Проаналізуємо їхні особливості та переваги, щоб обрати найоптимальніший інструмент, який сприятиме створенню зручного та ефективного додатку для контролю калорійності дієти.

UIKit

UIKit — це фреймворк що працює на платформах Apple, таких як iOS, macOS, який слугує для створення користувацьких інтерфейсів [16]. Цей фреймворк має змогу обробляти різні події викликані користувачем, керуючи таким чином взаємодією між користувачем та застосунком. UIKit був розроблений у 2007

році під час випуску першого iPhone і став ключовим інструментом для розробки iOS-додатків на основі мови програмування Objective-C.

Основні характеристики UIKit:

1. Можливість створення інтерфейсу використовуючи конструктор інтерфейсу: за допомогою безпосереднього розміщення елементів інтерфейсу та їх конфігурації у візуальному середовищі storyboard.xib.
2. Можливість створення усіх елементів інтерфейсу вручну за допомогою програмного коду. Це може бути вигідно для складних ефектів, динамічної логіки та гнучкого керування елементами інтерфейсу.
3. Має вбудовані контролери навігації, такі як UINavigationController, які спрощують реалізацію та управління навігацією між екранами додатка. Це дозволяє зручно організовувати інтерфейс та навігацію.
4. Надає потужні інструменти для реалізації анімацій та взаємодії з користувачем які в свою чергу не завжди потребують комплексний та важкий код. Це включає UIView-анімації, Core Animation та інші технології, що дозволяють створювати привабливі та динамічні ефекти в інтерфейсі.

SwiftUI

SwiftUI — це сучасний фреймворк який слугує для створення користувацьких інтерфейсів для платформи Apple. Він був представлений у 2019 році і розроблений на мові програмування Swift [17]. SwiftUI пропонує декларативний підхід до розробки інтерфейсів, у якому розробник описує вказаний стан інтерфейсу, а фреймворк автоматично генерує відповідний UI.

Основні характеристики SwiftUI:

1. SwiftUI використовує декларативний синтаксис, що дозволяє описувати, як повинен виглядати інтерфейс, а не як його створити крок за кроком.

2. Надає можливість використання простих та сучасних конструкцій для створення будь-яких елементів інтерфейсу, що спрощує процес розробки та покращує читання коду.
3. Автоматично відслідковує зміни стану інтерфейсу та оновлює його відповідно.
4. Доступність інтегрування з існуючими UIKit-компонентами, що дозволяє розробникам поетапно переходити від UIKit до SwiftUI.
5. Можливість розробки в реальному часі з Live Preview.

Для застосунку було обрано більш надійний та сталий фреймворк — UIKit завдяки його гнучкості, широким можливостям кастомізації та тривалій історії використання в екосистемі Apple. Фреймворк вже давно зарекомендував себе як стабільний і добре підтримуваний інструмент, що дозволяє створювати складні та інтерактивні інтерфейси користувача. Важливо відмітити, що цей фреймворк активно використовується в багатьох популярних додатках, що підтверджує його надійність та ефективність для реалізації найрізноманітніших завдань та проєктів.

2.3 Огляд архітектурних патернів

Для будь-якої мови програмування важливим етапом розробки є вибір та застосування певної архітектури або патерну для ефективної та структурованої розробки застосунку. Архітектура дозволяє програмісту розділити відповідальності та структурувати код у логічні компоненти, тим самим покращуючи організованість проєкту. Крім того, вибір правильного архітектурного шаблону допомагає забезпечити масштабованість проєкту для адаптивного впровадження нового функціоналу або зміни існуючого, а також робить підтримку проєкту іншими розробниками легкодоступною.

Основними архітектурними шаблонами для Swift є MVP, MVVM та VIPER. Архітектурні підходи MVP та MVVM мають спільні елементи, що включають:

1. Модель (Model) — відповідає за представлення даних та бізнес–логіку застосунку. Включає операції, пов’язані з роботою бази даних, мережевими запитами та обробкою інших даних.
2. Представлення (View) — забезпечує відображення та інтерефейс користувача. Визначає зовнішній вигляд інтерфейсу та відображення даних з моделі.
3. Presenter або ViewModel — відповідає за управління взаємодією між моделлю та представленням. Містить логіку відображення даних, обробку подій користувача та взаємодію з моделлю.

MVP

MVP (Model-View-Presenter) — це архітектурний підхід, що дозволяє досить чітко розділити відповідальності у застосунку та полегшити тестування коду [18]. Архітектура розділяє обов'язки та забезпечує однозв'язність, розбиваючи компоненти застосунку на три основні частини: Модель (Model), Представлення (View), Presenter.

Схему архітектури можна побачити на рисунку 4.

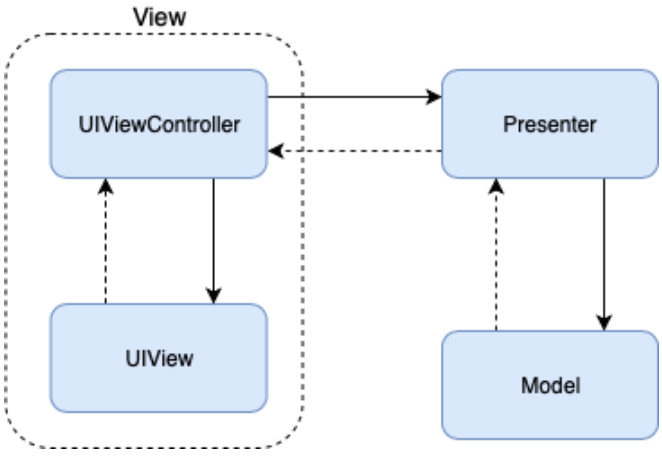

Рисунок 4 — *Схема архітектури MVP*

MVP чітко розділяє відповідальності, де модель представляє дані та логіку, представлення відображає інтерфейс, а презентер взаємодіє з обома, обробляючи події користувача та організуючи взаємодію з моделлю.

MVVM

MVVM (Model-View-ViewModel) — це архітектурний підхід, який спеціально призначений для розробки інтерфейсів користувача в SwiftUI з низкою переваг, які спираються на особливості цього фреймворку, такі як властивості ObservableObject та Published для автоматичного оновлення інтерфейсу при зміні даних у ViewModel [19]. Архітектура дозволяє ефективно розділити компоненти застосунку на три частини: Модель (Model), Представлення (View), ViewModel. Модель містить дані, представлення визначає вигляд інтерфейсу, а ViewModel є посередником між ними, містить логіку та забезпечує асинхронні оновлення інтерфейсу.

Схему архітектури можна побачити на рисунку 5.

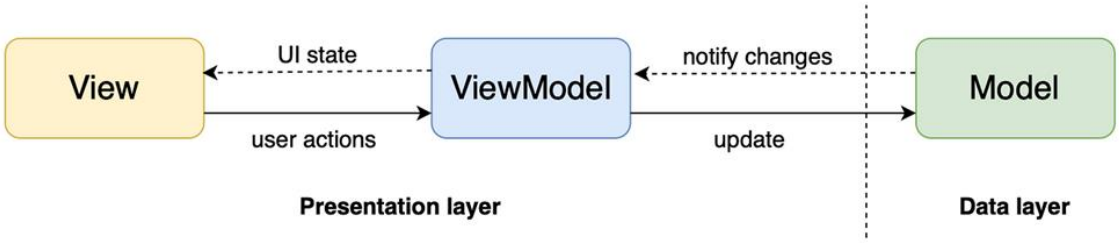

Рисунок 5 — *Схема архітектури MVVM*

VIPER

VIPER (View, Interactor, Presenter, Entity, Router) — це архітектурний шаблон, розроблений для покращення структури та обслуговування великих і складних проєктів [20]. Кожен елемент архітектури виконує конкретну роль, що сприяє

підтримці коду та зрозумілості структури проєкту. VIPER складається з п’яти компонентів:

1. Модель (Entity) — відповідає за представлення даних та може містити в собі базові об'єкти даних або структури, які представляють дані та обробляють операції, пов'язані з цими даними.
2. Представлення (View) — відповідає за відображення та інтерфейс користувача. Визначає зовнішній вигляд інтерфейсу та відображення даних з моделі.
3. Презентер (Presenter) — управляє логікою відображення даних, обробкою подій користувача та взаємодіє з моделлю. Реалізує проміжний шар між представленням і інтерактором.
4. Інтерактор (Interactor) — відповідає за обробку бізнес-логіки та взаємодію з моделлю. Робить запити до моделі та обробляє отримані дані перед передачею до презентера.
5. Маршрутизатор (Router) — відповідає за навігацію між екранами та взаємодію з іншими модулями. Використовується для переходу між різними частинами програми.

   Схему архітектури можна побачити на рисунку 6.

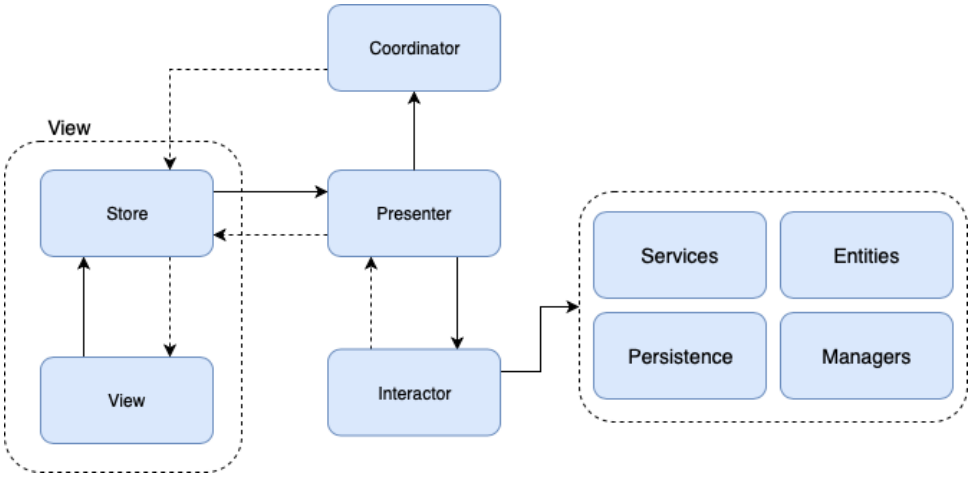

Рисунок 6 — *Схема архітектури VIPER*

Для застосунку було обрано архітектуру MVP. Вибір архітектури базується на вдалій сумісності та особливостями фреймворку UIKit, надаючи відносну простоту та гнучкість при використанні.

2.4 Огляд системи керувань пакунками

Менеджер пакетів — це інструмент, створений для ефективного керування пакунками в програмному забезпеченні. Основною метою менеджера пакетів є забезпечення простого способу додавання, видалення та оновлення сторонніх бібліотек, фреймворків або інших компонентів, що використовуються в розробці програмного продукту.

Популярними менеджерами пакетів які належать до мови програмування Swift є: Swift Package Manager (SPM), CocoaPods та Carthage. Кожен з них має свої переваги та особливості та вибір між ними може залежати від конкретних вимог та уподобань розробників.

Розглянемо кожен з запропонованих менеджерів детальніше.

SPM

SPM — це офіційний менеджер пакетів від Apple, який є інтегрованим у Swift та Xcode [21]. Він надає простий спосіб управління залежностями, використовуючи файл “Package.swift”. SPM автоматично управляє версіями та вирішує залежності, що робить його використання зручним для проєктів, які розробляються в екосистемі Apple.

CocoaPods

CocoaPods — це один з найпопулярніших менеджерів пакетів у спільноті розробників на мові програмування Swift та Objective-C [22]. Менеджер має велику базу пакетів, що полегшує пошук інструментів, інтеграцію зі сторонніми

бібліотеками та роботу з різними версіями бібліотек. CocoaPods отримав свою популярність через те що SPM на початкових версіях мала невелику базу бібліотек та була не найстабільнішою серед менеджерів.

Carthage

Carthage — це ще один популярний менеджер пакетів для мов програмування Swift та Objective-C [23]. Основна особливість Carthage полягає у його підході “без зв'язування”. Carthage не включає код залежностей безпосередньо в ваш проєкт. Замість цього, бібліотеки компілюються окремо як фреймворки або динамічні бібліотеки, і вручну додаються до вашого проєкту. Це дозволяє зберігати контроль над процесом компіляції та використовувати гнучку конфігурацію, але робить менеджер пакетів більш специфічним та важким у використанні.

Для застосунку було обрано менеджер пакетів SPM для управління залежностями проєкту через відносну простоту та вбудовану підтримку в середовищі розробки XCode.

2.5 Огляд СКБД

Система керування базами даних (СКБД) — це програмне забезпечення, яке забезпечує створення, управління та маніпулювання базами даних. СКБД надають розробникам інструменти для зберігання, отримання та обробки даних у їхніх додатках. У сучасній розробці додатків популярними є такі СКБД: Firebase, PostgreSQL та MongoDB. Кожна з цих систем має свої переваги та особливості, і вибір між ними залежить від конкретних вимог проєкту.

Firebase

Firebase — це хмарна платформа від Google, яка пропонує широкий спектр інструментів для розробки та управління додатками, включаючи базу даних

Realtime Database та Firestore [24]. Основною перевагою Firebase є його інтеграція з іншими сервісами Google, що забезпечує зручність та простоту у використанні. Платформа підтримує функціональність у реальному часі, дозволяючи створювати додатки з миттєвим оновленням даних. Крім того, Firebase має безкоштовний тарифний план, що дає змогу розпочати роботу з обмеженим обсягом даних і поступово масштабуватися за потреби.

PostgreSQL

PostgreSQL — це потужна об'єктно-реляційна СКБД, яка є однією з найвідоміших у світі [25]. Вона славиться своєю надійністю, розширюваністю та дотриманням стандартів SQL. PostgreSQL підтримує складні запити, транзакції та розширення, що робить її ідеальним варіантом для великих проєктів з високими вимогами до продуктивності та надійності. Однією з головних переваг PostgreSQL є її відкритий вихідний код, що дозволяє розробникам налаштовувати та оптимізувати СКБД та структуру під свої потреби.

MongoDB

MongoDB — це документо-орієнтована СКБД, яка використовує гнучкий формат JSON для збереження даних [26]. Вона відзначається високою швидкістю та масштабованістю, що робить її чудовим вибором для додатків з великими обсягами даних та потребою у швидкій обробці запитів. MongoDB дозволяє легко управляти складними даними і підтримує горизонтальне масштабування, що забезпечує високу продуктивність при збільшенні навантаження на систему.

Для додатку було обрано Firebase завдяки його популярності, доступності та безкоштовному плану. Firebase забезпечує легкий старт для розробників завдяки інтеграції з іншими сервісами Google і підтримці функціональності у реальному часі, що дозволяє створювати сучасні додатки з миттєвим оновленням даних.

2.6 Огляд продуктового API для контролю калорійності дієти

API (Application Programming Interface) — це сукупність інструментів і правил, які дозволяють різним програмам взаємодіяти одна з одною. API забезпечує розробників можливістю вбудовувати функції сторонніх сервісів у свої програми без потреби розробляти ці функції самостійно. У створенні додатків для роботи з харчовими продуктами та рецептами часто використовуються такі API, як Edamam, Spoonacular і Nutritionix. Кожен із них має свої особливі переваги, і вибір залежить від конкретних вимог проєкту.

Edamam API

Edamam API — потужний інструмент для отримання даних про продукти, рецепти та їхню харчову цінність [27]. Основною перевагою цього API є велика база даних, яка містить інформацію про мільйони продуктів та рецептів. Він підтримує різні типи запитів, включаючи пошук за інгредієнтами, рецепти та дані калорійності, нутрієнтів та вітамінів у стравах, продуктах. Edamam забезпечує швидку та точну відповідь на запити, що робить його надійним вибором для додатків, які потребують оперативного доступу до великої кількості даних. Важливо зазначити, що Edamam API дозволяє робити до 100 000 запитів на місяць безкоштовно, що робить його економічно вигідним для стартапів та невеликих додатків.

Spoonacular API

Spoonacular API — інструмент, що пропонує широкий спектр функцій, зокрема пошук рецептів, інформацію про продукти, харчову цінність та планування харчування [28]. У API є можливість інтеграції з іншими сервісами для планування меню та створення списків покупок, що робить його ідеальним вибором для додатків, які прагнуть надати користувачам комплексні рішення з

харчування. Spoonacular має чудово структуровану та широку документацію, що спрощує інтеграцію у додатки. Безкоштовний план Spoonacular API дозволяє робити до 150 запитів на день, що підходить для невеликих проєктів.

Nutritionix API

Nutritionix API — програмний інтерфейс, який пропонує дані про харчову цінність продуктів та страв, а також можливість створювати та керувати базами даних користувацьких продуктів [29]. Особливою перевагою Nutritionix є його фокус на точності та актуальності даних, оскільки він використовує інформацію, зібрану від відомих брендів і ресторанів. Це робить Nutritionix чудовим вибором для додатків, що потребують високої точності даних про продукти, особливо на ринку США. API має безкоштовний план, який дозволяє до 2000 запитів на день, що є привабливою опцією для невеликих додатків та стартапів.

Для додатку було обрано Edamam API через його великий обсяг безкоштовних запитів, що робить його економічно вигідним для додатків-стартапів. Крім того, Edamam API має велику базу даних продуктів та рецептів, що дозволяє створювати багатофункціональні додатки з широкими можливостями для користувачів.

2.7 Висновки з 2 розділу

Для розробки застосунку було ретельно відібрано найсучасніші та найбільш надійні засоби, що забезпечують високу якість та ефективність цього продукту в цілому. Основою стала мова програмування Swift, обрана завдяки активному розвитку, безпеці коду та сучасному синтаксису, що дозволяє реалізувати як об'єктно-орієнтовані, так і функціональні концепції програмування.

Для створення інтерфейсу користувача було обрано UIKit, який відзначається стабільністю, гнучкістю та можливостями кастомізації, що робить його ідеальним для складних і, крім того, інтерактивних додатків.

Архітектурний паттерн MVP забезпечує чітке розділення логіки додатку, надаючи простоту та гнучкість у розробці, тестуванні та підтримці.

Менеджер пакетів SPM (Swift Package Manager) було обрано для управління пакетами проєкту через його простоту та інтеграцію з XCode, що полегшує роботу зі сторонніми бібліотеками.

Для забезпечення роботи з базою даних та збереження необхідним даних на сервері було використано Firebase, який забезпечує функціональність у реальному часі, широкий спектр сервісів та безкоштовний план, що є особливо вигідним для стартапів.

Наостанок, для інтеграції інформації про продукти та рецепти було обрано Edamam API, який пропонує значну кількість безкоштовних запитів та обширну базу даних, що робить його економічно вигідним та функціонально багатим рішенням.

Таким чином, вибір засобів розробки, включаючи Swift, UIKit, MVP, SPM, Firebase та Edamam API, забезпечує високу якість, стабільність та функціональність застосунку, відповідно до сучасних вимог користувачів та ринку.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1 Опис предметної області

У сучасному світі, де здоровий спосіб життя набирає популярності, виникає потреба у зручних та ефективних інструментах для моніторингу калорійності дієти. Сьогодні існує багато додатків, які допомагають відстежувати харчування та фізичну активність, проте вони часто не задовольняють всі потреби користувачів. Серед основних проблем — складність у використанні, неточність даних про продукти та обмежений безкоштовний функціонал. Інноваційний iOS-додаток дозволить користувачам легко контролювати калорійність свого раціону, отримувати індивідуальні поради щодо харчування та досягати поставлених цілей у сфері здорового харчування. Додаток буде корисним як для професійних спортсменів, так і для звичайних людей, які прагнуть підтримувати здоровий спосіб життя або потребують спеціальних дієтичних рекомендацій.

Розробка додатку передбачає використання сучасних технологій, таких як Swift, UIKit, MVP, Firebase та Edamam API. Це забезпечує високу продуктивність та зручність користування. Додаток розроблено таким чином, щоб користувачі могли легко вводити та переглядати дані, а також отримувати докладні звіти про свій прогрес. Завдяки цьому, користувачі можуть бачити реальні результати своїх зусиль і коригувати свій раціон для досягнення оптимальних результатів.

Основною метою додатку є надання користувачам зручного та точного інструменту для контролю за харчуванням. Додаток дозволяє вводити особисті дані для розрахунку необхідної кількості калорій, вести детальний щоденник харчування та отримувати інформацію про калорійність і склад продуктів. Інтеграція з надійними джерелами даних, такими як Edamam API, забезпечує точність і актуальність інформації про харчові продукти. Це гарантує, що користувачі отримують найсвіжіші дані, необхідні для прийняття обґрунтованих рішень щодо свого

раціону. Крім того, використання Firebase для бекенд-інфраструктури гарантує безпечне зберігання персональних даних користувачів, що є критично важливим аспектом для додатку такого типу.

Додаток допомагає користувачам ефективно контролювати своє харчування, досягати бажаних результатів та підтримувати здоровий спосіб життя. Завдяки точним даним та індивідуально підібраним порадам, він стає незамінним інструментом для всіх, хто прагне покращити своє харчування та зберігати оптимальний баланс калорій. Додаток має на меті задовольнити потреби найвибагливіших користувачів і стати важливим інструментом на шляху до здорового способу життя.

3.2 Архітектура системи

Архітектура iOS-додатку для контролю калорійності дієти спирається на сучасні технології та передові архітектурні шаблони, що забезпечують високу продуктивність, зручність використання та точність даних. Ключовими елементами цієї архітектури є мова програмування Swift, фреймворк UIKit, архітектурний шаблон MVP, Firebase для бекенд-інфраструктури, Edamam API для доступу до харчової інформації, Alamofire для роботи з HTTP-запитами та Kingfisher для обробки зображень.

Основу додатку складають мова програмування Swift та фреймворк UIKit. Swift, як сучасна мова програмування, забезпечує високу продуктивність та безпеку коду, що особливо важливо для додатків, які обробляють персональні дані користувачів. Використання Swift дозволяє розробляти ефективний та стійкий до помилок код, який легко підтримувати та розширювати. UIKit, у свою чергу, надає чудовий потужний набір інструментів для створення інтуїтивно зрозумілих та естетично привабливих інтерфейсів для сучасного користувача. З точки зору розробника, з його допомогою можна створювати гнучкі та адаптивні

інтерфейси, що підлаштовуються під різні розміри екранів, які в першу чергу є надійними та досить простими для створення. Комбінація Swift та UIKit дозволяє створювати додатки, які швидко реагують на дії користувачів, забезпечуючи високу продуктивність та зручність використання.

Основним архітектурним підходом, що використовується у додатку, є шаблон MVP (Model-View-Presenter). Цей шаблон забезпечує чіткий розділ відповідальності між трьома ключовими компонентами, що сприяє підвищенню модульності та тестованості коду.

Використання архітектурного шаблону MVP має кілька ключових переваг:

1. Розділ відповідальності: MVP чітко розділяє логіку додатку на три окремі компоненти — модель, уявлення та презентер. Це робить код більш організованим та легшим для розуміння і підтримки.
2. Тестованість: Кожен з компонентів MVP можна тестувати незалежно один від одного. Це спрощує процес написання юніт-тестів та підвищує якість коду.
3. Модульність: Чітке розділення компонентів дозволяє легко замінювати або оновлювати окремі частини додатку без впливу на інші частини. Це особливо корисно при розширенні функціональності додатку.
4. Повторне використання коду: Компоненти MVP можуть бути легко використані повторно в інших проєктах або частинах додатку, що знижує дублювання коду та покращує його підтримку різними системами.

Компоненти MVP є наступними:

1. Модель (Model): Модель відповідає за управління даними та бізнес-логікою. Вона отримує дані з бази даних, API або інших джерел, обробляє їх і передає презентеру для подальшої роботи. Модель також може виконувати різноманітні обчислення та обробляти запити від презентера, забезпечуючи точність та актуальність даних.

2. Уявлення (View): Уявлення відповідає за відображення інформації та взаємодію з користувачем. Воно реагує на дії користувача, такі як введення даних або натискання кнопок, і передає ці події презентеру. Завдяки цьому, View зосереджується виключно на візуальному відображенні даних та взаємодії з користувачем, що робить код більш чистим та структурованим.
3. Презентер (Presenter): Презентер діє як посередник між моделлю та уявленням. Він обробляє події від уявлення, робить відповідні запити до моделі та повертає оброблені дані назад до уявлення для відображення. Це забезпечує чітку координацію між компонентами та дозволяє легко тестувати логіку додатку.

У додатку презентер відповідає за логіку взаємодії між користувачем та даними. Наприклад, коли користувач вводить інформацію про споживану їжу, уявлення передає ці дані презентеру. Презентер обробляє ці дані, надсилає запит до моделі для зберігання або обробки інформації, а потім повертає результат назад до уявлення для відображення користувачу.

Такий підхід забезпечує не тільки чітке розділення відповідальностей, але й дозволяє швидко та ефективно обробляти дані, що надходять від користувача, надаючи йому точну та актуальну інформацію у зручному форматі.

Для забезпечення бекенд-інфраструктури використовується Firebase, який надає широкий спектр інструментів для аутентифікації користувачів, управління базою даних у реальному часі, хмарного зберігання та хостингу. Firebase Authentication забезпечує простий та безпечний спосіб аутентифікації користувачів за допомогою особистої електронної пошти та паролю, а Firestore, в свою чергу, дозволяє зберігати дані в реальному часі, забезпечуючи миттєву синхронізацію між усіма підключеними пристроями.

Інтеграція з Edamam API дозволяє додатку отримувати точні та актуальні дані про харчові продукти. API надає детальну інформацію про склад продуктів, їх калорійність та деякі інші поживні властивості. Це дозволяє користувачам

обирати з потрібних споживаних продуктів саме ті які їм потрібні з відображенням деталей про них і отримувати миттєві розрахунки їх калорійної цілі.

Для відправки запитів до Edamam API використовується Alamofire — популярна бібліотека для роботи з HTTP-запитами в Swift. Alamofire не тільки спрощує процес відправки та обробки GET-запитів, але й робить це максимально оптимізовано, використовуючи складні механізми для покращення цього процесу, що дозволяє ефективно отримувати необхідну інформацію для додатку.

Взаємодія між компонентами системи здійснюється через HTTP протокол за допомогою Alamofire. Після введення користувачем даних про продукти харчування, додаток обробляє їх за допомогою MVP архітектури, надсилаючи запити до Edamam API для отримання інформації про продукти та до Firebase для зберігання даних. Отримані відповіді відображаються користувачеві у вигляді зручних списків продуктів.

Зображення фізичного розташування компонентів застосунку та їх взаємодію з іншими системами, можна побачити на рисунку 7.

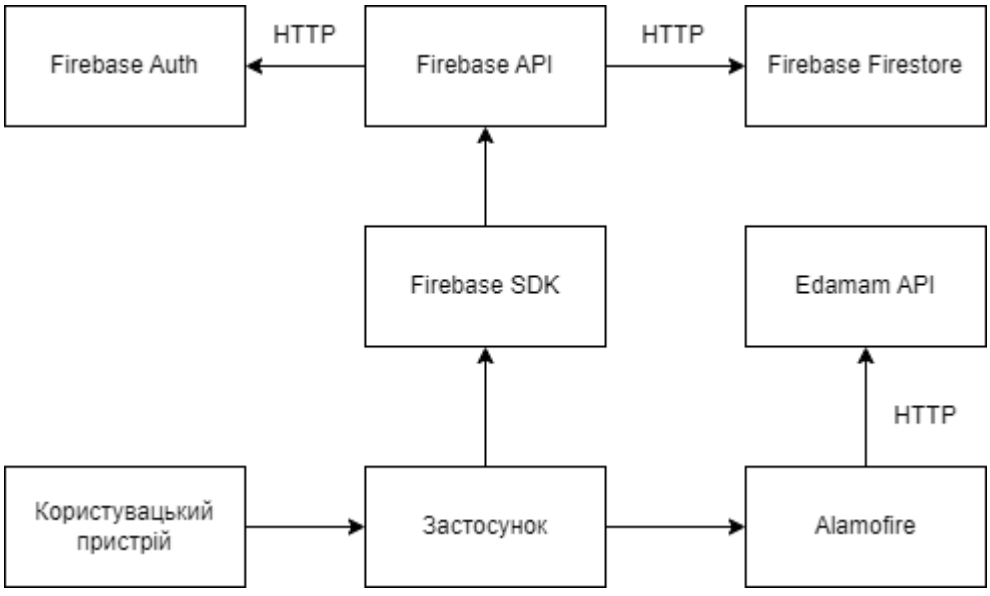

Рисунок 7 — *Архітектура системи*

Для обробки отриманих посилань на зображення використовується Kingfisher. Він відповідає за завантаження зображень через посилання, кешування та їхнє відображення, що значно покращує користувацький досвід. Тож, зображення продуктів завантажуються з Edamam API, кешуються в Kingfisher і відображаються у відповідних місцях інтерфейсу користувача. Використання Kingfisher дозволяє уникнути повторного завантаження одних і тих же зображень, що знижує навантаження на мережу і прискорює роботу додатку.

Таким чином, архітектура iOS-додатку для контролю калорійності дієти забезпечує надійний, масштабований та зручний інструмент для користувачів. Інтеграція з Firebase та Edamam API надає точні дані для прийняття обґрунтованих рішень щодо харчування, а використання MVP сприяє легкій підтримці та розширенню функціональності додатку. Використання передових технологій забезпечує надійність та ефективність роботи додатку, що робить його незамінним помічником для тих, хто прагне контролювати свій раціон та покращувати своє здоров'я.

3.3 Функціональні вимоги системи

Для забезпечення ефективного контролю калорійності дієти та досягнення поставлених цілей користувачів, розробка iOS-додатку повинна включати ряд ключових функціональних можливостей. Нижче наведені вимоги для розробки та імплементації iOS-додатку для контролю калорійності дієти з використанням Swift, UIKit, MVP, Firebase та Edamam API.

Розглянемо детальніше кожну з функціональних можливостей.

1. Аутентифікація користувачів:
    - Система повинна забезпечувати можливість користувачам реєструватися, створюючи нові облікові записи за допомогою особистої електронної пошти та пароля.

- Користувачі повинні мати змогу входити вже в існуючі облікові записи, використовуючи свої персональні дані.
- Користувачі повинні мати змогу відновити забутий пароль через електронну пошту.

2. Введення та обробка персональних даних:
    - Користувачі повинні мати змогу вводити особисті дані, такі як вік, зріст, вага, рівень фізичної активності та стать, через спеціальний інтерфейс в додатку.
    - Персональні дані користувача повинні бути збережені в базі даних та використовуватися для розрахунку індивідуальних калорійних потреб.
    - Після введення збереження персональних даних програма має автоматично розраховувати новий калорійний план згідно з оновленими користувачем даними.
3. Калькулятор калорій:
    - Додаток має містити екран вибору цілі користувача, таких як схуднення, набір ваги чи підтримка поточної ваги.
    - Користувач повинен мати змогу обрати новий калорійний план.
    - Калорійний план повинен бути розрахований та збережений до бази даних після натискання користувачем на відповідну кнопку в інтерфейсі.
4. Журнал спожитих продуктів:
    - Інтерфейс додатку повинен давати змогу користувачам обирати спожиті ними продукти харчування за допомогою пошукового елементу в інтерфейсі, а також списку який відображає результати пошуку.
    - Користувачі повинні мати доступ до інформації про продукти, включаючи їх назву, зображення, склад, калорійність, а також вміст білків, жирів та вуглеводів.

- Інформація та список продуктів має бути представлений у зручному для перегляду форматі.
- Обрані користувачем спожиті продукти харчування повинні додаватися до корзини, де користувач повинен мати змогу збільшити або зменшити кількість того чи іншого продукту за допомогою відповідних кнопок у користувацькому інтерфейсі з можливістю виключити елемент зі списку.
- Користувачі повинні мати змогу додавати продукти до свого щоденного журналу.
- Інформація про продуктовий журнал повинна автоматично оновлюватися та зберігатися в базі даних.

5. Відстеження прогресу:
    - Додаток повинен надавати можливість користувачам слідкувати за своїм прогресом у досягненні поставлених цілей на щоденній, тижневій та місячній основі.
    - Користувачі повинні мати доступ до щоденного журналу спожитих продуктів.
    - Щоденні рекомендовані калорійні показники повинні бути зручно та чітко відображені.

Описані функціональні вимоги визначають ключові можливості iOS-додатку для контролю калорійності дієти, які включають ведення харчового щоденника, розрахунок калорійної норми, встановлення персоналізованих цілей щодо споживання калорій та самостійний аналіз прогресу. Разом ці функції утворюють цілісну систему, яка забезпечує зручність використання, швидкодію та інтуїтивний інтерфейс. Реалізація цих вимог дозволить створити сучасний додаток, що якнайкраще відповідає потребам користувачів, сприяючи ефективному контролю дієти та покращенню загального самопочуття.

UseCase діаграму користувача можна побачити на рисунку 8.

Рисунок 8 — *UseCase діаграма можливостей взаємодії користувача з додатком*

3.4 Вимоги до апаратного та програмного забезпечення

Для успішного функціонування додатку на пристроях користувачів необхідні відповідні апаратні та програмні компоненти для того щоб забезпечити користувачів зручним інтерфейсом та швидким доступом до інформації. Враховуючи важливість стабільної роботи додатку, особливо при роботі з великою кількістю даних та інтеграцією з зовнішніми сервісами, а також використовуючи останні технології та підходи, дотримання мінімальних та оптимальних вимог є обов'язковим.

Мінімальні вимоги є наступними:

1. Операційна система: iOS 15.0 або вище.
2. Пристрій: iPhone 7 або новіший, iPad 5-го покоління або новіший.
3. Процесор: A10 Fusion чип або новіший.

4. Оперативна пам'ять: 2 ГБ RAM.
5. Пам’ять: 250 МБ для встановлення додатку та можливості функціонування.

Оптимальні вимоги:

1. Операційна система: iOS 15.0 або вище.
2. Пристрій: iPhone 12 або новіший, iPad Pro 3-го покоління або новіший.
3. Процесор: A14 Bionic чип або новіший.
4. Оперативна пам'ять: 4 ГБ RAM або більше.
5. Пам’ять: 500 МБ для забезпечення оптимальної роботи, збереження даних користувача та тимчасових стандартних файлів системи iOS.

Ці вимоги спрямовані на забезпечення стабільної та ефективної роботи додатку, що дозволить користувачам насолоджуватись швидкою та надійною роботою програми. Виконання цих технічних вимог гарантує, що користувачі зможуть безперешкодно отримувати доступ до всіх функцій додатку, включаючи перегляд та аналіз харчової інформації, збереження та синхронізацію даних. Таким чином, дотримання цих вимог забезпечить високу якість користувацького досвіду, зменшить ризик технічних проблем та дозволить додатку працювати безперебійно навіть під час інтенсивного використання.

3.5 Модулі і алгоритми

Важливу роль у проєктуванні та реалізації iOS-додатку відіграють алгоритми та модулі, що забезпечують його функціональність. У цьому розділі буде розглянуто такі аспекти:

1. Опис алгоритмів, які є ключовими для ефективної роботи додатку, оскільки вони визначають основні процеси та обробку даних.
2. Перелік класів з коротким описом їх функцій, що допомагає зрозуміти, яку роль виконує кожен компонент у загальній структурі додатку.

3. Детальний опис основних класів із прикладами коду, що надає глибше розуміння їх реалізації та взаємодії.
4. Опис реалізації окремих важливих функцій, які підкреслюють їх значимість у загальній архітектурі додатку.

Такий підхід сприятиме комплексному розумінню модульної архітектури додатку та принципів роботи його алгоритмів, що є необхідним для успішного впровадження та подальшого вдосконалення.

Перейдемо детальніше до кожного з аспектів.

Опис алгоритмів

В додатку головним алгоритмом є розрахунок калорійної мети, який визначає кількість калорій, необхідних користувачеві для досягнення його цілей щодо зниження, підтримки або набору ваги. Цей алгоритм враховує індивідуальні параметри користувача, такі як стать, вік, вага, зріст та рівень фізичної активності.

Основою для розрахунку калорійної мети є визначення базового обміну речовин, скорочено **BMR**. BMR — це кількість енергії, яку організм витрачає у стані спокою для підтримки життєво важливих функцій, таких як робота серця, дихання, підтримка температури тіла, мозку та інших органів. Тобто, вимірювання BMR відображає мінімальну кількість калорій, необхідну для забезпечення основних метаболічних процесів не беручи до уваги додаткові фізичні навантаження на тіло.

Для розрахунку BMR використовується формула Харріса-Бенедикта [30]. Формула була розроблена на початку 20-го століття вченими Джеймсом Артуром Харрісом і Френсісом Гано Бенедиктом. Вона враховує вагу, зріст, вік та стать користувача, що дозволяє отримати точний показник базового обміну речовин та є ключовим елементом для подальших розрахунків калорійної мети.

Формула Харріса-Бенедикта виглядає наступним чином:

1. Для чоловіків:

$$BMR = 88.362 + (13.397 * \text{вага в кг}) + (4.799 * \text{ріст в см}) - (5.677 * \text{вік в роках})$$

2. Формула для жінок:

$$BMR = 447.593 + (9.247 * \text{вага в кг}) + (3.098 * \text{ріст в см}) - (4.330 * \text{вік в роках})$$

Після розрахунку BMR також важливо враховувати рівень фізичної активності користувача. Це дозволяє додатку створити більш точні та персоналізовані калорійні цілі, оскільки фізична активність значно впливає на загальні енергетичні потреби організму.

Для розрахунків фізичної активності використовуються наступні значення, що включають в себе відповідний коефіцієнт:

1. Сидячий спосіб життя: мінімальна фізична активність, що включає повсякденну діяльність без регулярних тренувань або особливих фізичних навантажень. Прикладом є офісна робота, навчання або будь-яка інша діяльність, що не вимагає значних фізичних зусиль. Використовується коефіцієнт зі значенням 1.2.
2. Легкі фізичні навантаження: включають легкі тренування або деякі фізичні навантаження декілька разів на тиждень. Це можуть бути прогулянки, легкий біг, походи та інше. Використовується коефіцієнт зі значенням 1.375.
3. Помірні фізичні навантаження: регулярні тренування середньої інтенсивності. Це можуть бути заняття фітнесом, аеробікою, плавання, їзда на велосипеді та інші навантаження по кілька разів на тиждень. Використовується коефіцієнт зі значенням 1.55.
4. Інтенсивні фізичні навантаження: щоденні інтенсивні тренування або фізичні навантаження високої інтенсивності. Прикладами є важкі тренування в спортзалі, професійні заняття спортом або будь-які інші важкі та довготривалі навантаження. Використовується коефіцієнт зі значенням 1.725.

5. Дуже інтенсивні фізичні навантаження: дуже високий рівень фізичної активності, що включає кілька тренувань на день або важку фізичну роботу. Це може бути важка фізична праця, професійний спорт або просто більш регулярні виснажливі тренування. Використовується коефіцієнт зі значенням 1.9.

Загальні добові енерговитрати, або скорочено **TDEE**, враховують не лише базовий обмін речовин, але й енерговитрати на фізичну активність та термічний ефект їжі. TDEE відображає загальну кількість калорій, які організм витрачає протягом дня, включаючи основні метаболічні процеси, фізичну активність та травлення. Це допомагає визначити кількість калорій, необхідних для підтримки поточної ваги або досягнення поставленої цілі.

Формула для розрахунку TDEE:

$$TDEE = BMR * \text{рівень фізичної активності}$$

В залежності від мети користувача, загальні добові енерговитрати коригуються наступним чином:

1. Для досягнення мети зниження ваги, від загальної кількості калорій віднімається 700 калорій.
2. Щоб підтримувати поточну вагу, значення TDEE залишається без змін.
3. Для досягнення мети набору ваги, до загальної кількості калорій додається 500 калорій.

Цей підхід дозволяє адаптувати обрану користувачем калорійну ціль до його індивідуальних потреб, забезпечуючи ефективність досягнення поставлених результатів. Алгоритм розрахунку калорійної мети є фундаментом для функціональності додатку, надаючи точні та персоналізовані рекомендації щодо калорійності дієти, допомагаючи користувачам досягати своїх цілей.

Базова структура модуля

Кожен екран інтерфейсу iOS-додатку для обраної MVP архітектури складається з п’яти класів: ViewController, Presenter, Model, Module, Router.

Розглянемо детальніше кожен з них.

ViewController

ViewController відповідає за відображення інтерфейсу користувача та обробку взаємодії з ним. Основні функції включають відображення даних, отриманих від Presenter, обробку подій користувача, таких як натискання кнопок або введення тексту, та виклик методів Presenter для оновлення даних або виконання певних дій. ViewController містить об’єкт відповідного Presenter та взаємодіє безпосередньо з його методами.

Presenter

Presenter виступає посередником між ViewController та Model. Він отримує дані від моделі, форматує їх для відображення у ViewController та керує бізнес-логікою програми. Presenter забезпечує логіку відображення та обробки даних, звільняючи ViewController від цих завдань. Presenter тримає у собі об’єкт ViewController для виклику необхідних оновлень інтерфейсу.

Model

Model відповідає за управління даними та бізнес-логікою додатку. Основні функції включають зберігання, обробку та отримання даних, необхідних для роботи додатку. Model містить структури та методи для зберігання даних, їх обробки, включаючи обчислення, перетворення та валідацію. Вона забезпечує Presenter усіма необхідними даними для відображення та обробки, але не взаємодіє безпосередньо з ViewController. Model відокремлює логіку обробки даних від

відображення інтерфейсу користувача, що сприяє кращій організації коду та його підтримці.

Module

Module відповідає за створення та конфігурацію компонентів екрану. Його основні функції включають ініціалізацію та налаштування ViewController, Presenter і Router, ін'єкцію залежностей між компонентами та забезпечення правильного компонування компонентів та взаємодії між ними. Модуль гарантує, що всі компоненти екрану налаштовані та пов'язані між собою належним чином.

Router

Router відповідає за навігацію між екранами додатку. Його основні функції включають виконання переходів між екранами, підготовку даних для наступного екрану та взаємодію з іншими роутерами системи для забезпечення очікуваної навігації. Router полегшує переміщення користувача між різними частинами додатку, забезпечуючи послідовну структуровану розробку.

Висновок

Ці п’ять класів створюють єдину цілісну структуру для створення ефективного та компонованого інтерфейсу користувача в iOS-додатку, забезпечуючи чітке розділення відповідальності та полегшуючи підтримку та розширення коду.

Перелік наявних модулів

Розглянемо список наявних модулів у додатку:

1. WelcomeScreen — модуль вітального екрану для вибору способу авторизації користувача.
2. LoginScreen — модуль для входу користувача в додаток. Алгоритм входу виконується за допомогою Firebase Authentication.

3. RegistrationScreen — модуль реєстрації користувача в додаток. Алгоритм реєстрації виконується за допомогою Firebase Authentication.
4. PersonalInfoScreen — модуль введення користувачем особистих даних, таких як вік, стать, вага, зріст та активність. Збереження даних відбувається за допомогою сервісу Firebase Firestore.
5. TargetGoalScreen — модуль вибору користувачем поставленої цілі: схуднення, набір ваги або підтримання поточної ваги.
6. CaloriesInfoScreen — модуль відображення головної сторінки додатку, яка включає щоденну та місячну калорійна ціль для перегляду досягнень користувача.
7. FoodListScreen — модуль для додавання щоденних спожитих продуктів користувача. Модуль включає в себе клас FoodCart, який слугує для відображення корзини продуктів.

   Структуру наявних модулів можна побачити на рисунку 9.

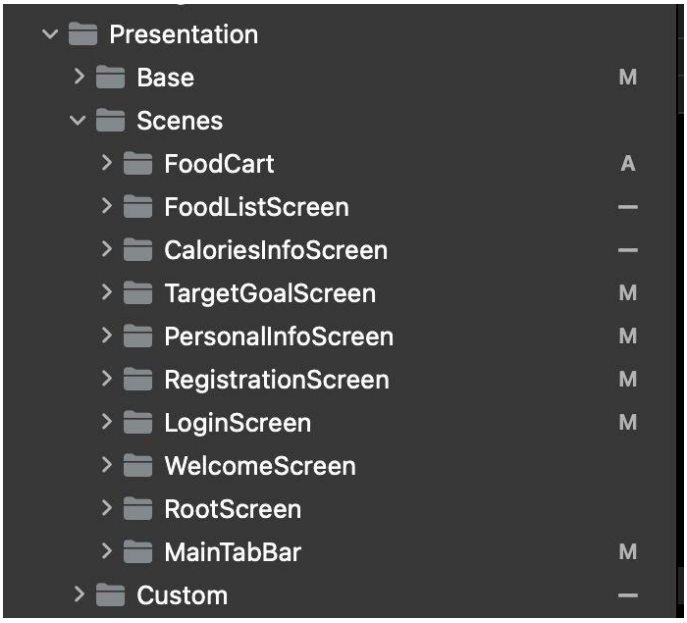

Рисунок 9 — *Структура модулів додатку*

UserService

UserService — клас призначений для керування даними користувачів, включаючи особисті дані, цілі по калоріях, та інші пов'язані дані з додатком необхідні дані. Він забезпечує механізми збереження, оновлення та отримання цих даних з Firestore. Крім того, клас надає можливість відслідковувати зміни в особистих даних користувачів, повідомляючи про ці зміни відповідні класи-спостерігачі. UserService також надає функціональність для обробки кошика їжі користувача, що включає додаткові калорії.

Перелік методів класу UserService можна побачити на лістингу 1.

Лістинг 1 *Перелік методів класу UserService*

```
final class UserService {
    func addObserver(_ observer: PersonalDataObserver)
    func removeObserver(_ observer: PersonalDataObserver)
    func getCurrentUserUid() -> String?
    func loadPersonalData(completion: @escaping
(Result<PersonalData, Error>) -> Void)
    func updatePersonalData(_ data: PersonalData, completion:
@escaping (Result<Void, Error>) -> Void)
    func updateUserTargetData(targetGoal: TargetGoal,
completion: @escaping (Result<Void, Error>) -> Void)
    func updateUserTargetData(completion: @escaping
(Result<Void, Error>) -> Void)
    func updateUserFoodCart(foodCart: [DailyFoodDetails],
additionalCalories: Int, completion: @escaping (Result<Void,
Error>) -> Void)
    func getUserTargetData(completion:
((Result<UserTargetData, Error>) -> Void)?)
}
```

Переглянемо кожен з методів класу UserService:

1. func addObserver(_ observer: PersonalDataObserver) — додає спостерігача для оновлення особистих даних користувача.
2. func removeObserver(_ observer: PersonalDataObserver) — видаляє спостерігача для оновлення особистих даних користувача.
3. func getCurrentUserUid() -> String? — повертає ідентифікатор поточного користувача.
4. func loadPersonalData(completion: @escaping (Result<PersonalData, Error>) -> Void) — завантажує особисті дані користувача з Firestore.
5. func updatePersonalData(_ data: PersonalData, completion: @escaping (Result<Void, Error>) -> Void) — оновлює особисті дані користувача в Firestore.
6. func updateUserTargetData(targetGoal: TargetGoal, completion: @escaping (Result<Void, Error>) -> Void) — оновлює дані цілі користувача з новою ціллю.
7. func updateUserTargetData(completion: @escaping (Result<Void, Error>) -> Void) — оновлює дані цілі користувача, використовуючи існуючі дані.
8. func updateUserFoodCart(foodCart: [DailyFoodDetails], additionalCalories: Int, completion: @escaping (Result<Void, Error>) -> Void) — оновлює кошик їжі користувача з додатковими калоріями.
9. func getUserTargetData(completion: ((Result<UserTargetData, Error>) -> Void)?) — отримує дані цілі користувача з Firestore.

CalorieCalculator

CalorieCalculator — клас призначений для розрахунку щоденної калорійної цілі користувача на основі його персональних даних та обраної дієтичної мети.

На лістингу 2 можна побачити методи алгоритму розрахунку калорійної цілі користувача за його персональними даними з використанням формули Харріса-Бенедикта та з урахуванням активності.

Лістинг 2 *Методи алгоритму CalorieCalculator*

```
func calculateDailyCaloriesGoal(
        for targetGoal: TargetGoal,
        with personalData: PersonalData
    ) -> Int {
        let bmr = calculateBMR(for: personalData)
        let activityFactor = personalData.activity.multiplier
        var tdee = bmr * activityFactor
        tdee = targetGoal.adjustCalories(tdee)
        return Int(tdee)
}

func calculateBMR(for personalData: PersonalData) -> Double {
        let weightFactor = personalData.gender == .male ?
13.397 : 9.247
        let heightFactor = personalData.gender == .male ?
4.799 : 3.098
        let ageFactor = personalData.gender == .male ? 5.677 :
4.330
        let baseBMR = personalData.gender == .male ? 88.362 :
447.593
        return baseBMR
        + (weightFactor * Double(personalData.weight))
        + (heightFactor * Double(personalData.height))
        - (ageFactor * Double(personalData.age))
}
```

TargetGoalScreen

TargetGoalScreen — модуль призначений для надання користувачам можливості вибору бажаної дієтичної мети, яка може включати схуднення, набір ваги або підтримання поточної ваги. Інтерфейс користувача забезпечує зручний вибір цілі шляхом натискання на відповідні кнопки.

Коли користувач обирає дієтичну мету, ViewController фіксує цю дію і передає сигнал презентеру. Presenter, у свою чергу, обробляє вибір користувача за допомогою визначеного алгоритму, використовуючи його персональні дані, збережені у Firebase Firestore.

Процес обробки даних включає кілька етапів. Після отримання персональних даних з бази, вони проходять через спеціальні методи, які визначають необхідний алгоритм і працюють згідно із заданими формулами. Цей процес дозволяє точно налаштувати дієтичні рекомендації відповідно до індивідуальних потреб користувача.

Після обробки інформації, Presenter здійснює запит до Firebase Firestore для збереження оновлених даних на сервері. На завершення цього запиту, Presenter надсилає команду до View для відображення інформаційного вікна, яке повідомляє користувача про успішність або невдачу здійсненої операції.

Важливо відмітити, що в разі відсутності необхідних даних користувача у базі даних, додаток відобразить сповіщення з вікном інформації, що вказує на потребу заповнення відсутньої інформації.

Ці методи відповідають за безперебійну роботу всієї системи, починаючи з вибору цілі, продовжуючи збереженням даних і закінчуючи інформуванням користувача. Вони ретельно обробляють кожен крок, забезпечуючи точність та ефективність у досягненні дієтичних цілей користувача.

Повний перелік методів класу TargetGoalViewController модулю TargetGoalScreen можна побачити на лістингу 3.

Лістинг 3 *Перелік методів класу TargetGoalViewController*

```
final class TargetGoalViewController {
    override func viewDidLoad()
    private func setupView()
    private func setupChangeGoalButton()
    private func setupTargetGoalButtons()
    private func setupMainStackView()
    private func setupMainContentStackView()
    private func updateButtonStates(_sender:
    TargetGoalButton)
    private func animateGoalChange(for targetGoal:
    TargetGoal)
    private func showCaloriesInfo()
    @objc func buttonTapped(_ sender: TargetGoalButton)
    @objc func changeTargetButtonTapped()
    func showTargetGoalChangedAlert()
    func handleError(_ error: BaseError)
    func setLoadingState(isLoading: Bool)
}
```

Переглянемо кожен з методів класу TargetGoalViewController:

1. override func viewDidLoad() — викликається при завантаженні представлення, налаштовує початковий стан інтерфейсу користувача.
2. handleError(_ error: BaseError) — обробляє помилки, що виникають в процесі роботи, передаючи їх у відповідний метод обробки помилок.
3. setLoadingState(isLoading: Bool) — встановлює стан завантаження для відображення індикатора завантаження.
4. private func setupView() — налаштовує основне представлення, додаючи всі необхідні елементи інтерфейсу.

5. private func setupChangeGoalButton() — налаштовує кнопку зміни цілі, встановлюючи її розмір та прив'язки.
6. private func setupTargetGoalButtons() — налаштовує кнопки вибору цілі, додаючи їх у відповідний контейнер та встановлюючи обробники подій.
7. private func setupMainStackView() — налаштовує основний контейнер, додаючи в нього всі інші елементи інтерфейсу.
8. private func setupMainContentStackView() — налаштовує основний контейнер для вмісту, додаючи в нього основний вміст та кнопку зміни цілі.
9. private func updateButtonStates(_ sender: TargetGoalButton) — оновлює стан кнопок вибору цілі, відмічаючи вибрану кнопку та анімує зміну цілі.
10. private func animateGoalChange(for targetGoal: TargetGoal) — анімує зміну вмісту відповідно до вибраної цілі.
11. private func showCaloriesInfo() — показує інформацію про калорії, перемикаючи вкладку на головний екран.
12. @objc func buttonTapped(_ sender: TargetGoalButton) — обробляє події натискання на кнопки вибору цілі, оновлюючи стан кнопок.
13. @objc func changeTargetButtonTapped() — обробляє події натискання на кнопку зміни цілі, викликаючи метод оновлення цілі користувача у презентері.
14. func showTargetGoalChangedAlert() — показує сповіщення про успішну зміну цілі користувача.

На лістингу 4 можна побачити повний перелік методів класу DefaultTargetGoalPresenter модулю TargetGoalScreen.

Лістинг 4 *Перелік методів класу DefaultTargetGoalPresenter*

```
final class DefaultTargetGoalPresenter {
    init(view: TargetGoalView)
    func updateUserGoal(with targetGoal: TargetGoal)
```

```
    private func updateUserTargetData(for targetGoal:
TargetGoal)
}
```

Переглянемо кожен з методів класу DefaultTargetGoalPresenter:

1. init(view: TargetGoalView) — ініціалізує екземпляр класу з параметром view, який відповідає за інтерфейс користувача.
2. updateUserGoal(with targetGoal: TargetGoal) — оновлює ціль користувача, викликаючи приватний метод updateUserTargetData.
3. updateUserTargetData(for targetGoal: TargetGoal) — оновлює дані цілі користувача, використовуючи UserService, обробляє результат і передає його на відповідний метод view для обробки.

Переглянути діаграму класів модулю TargetGoalScreen можна побачити на рисунку 10.

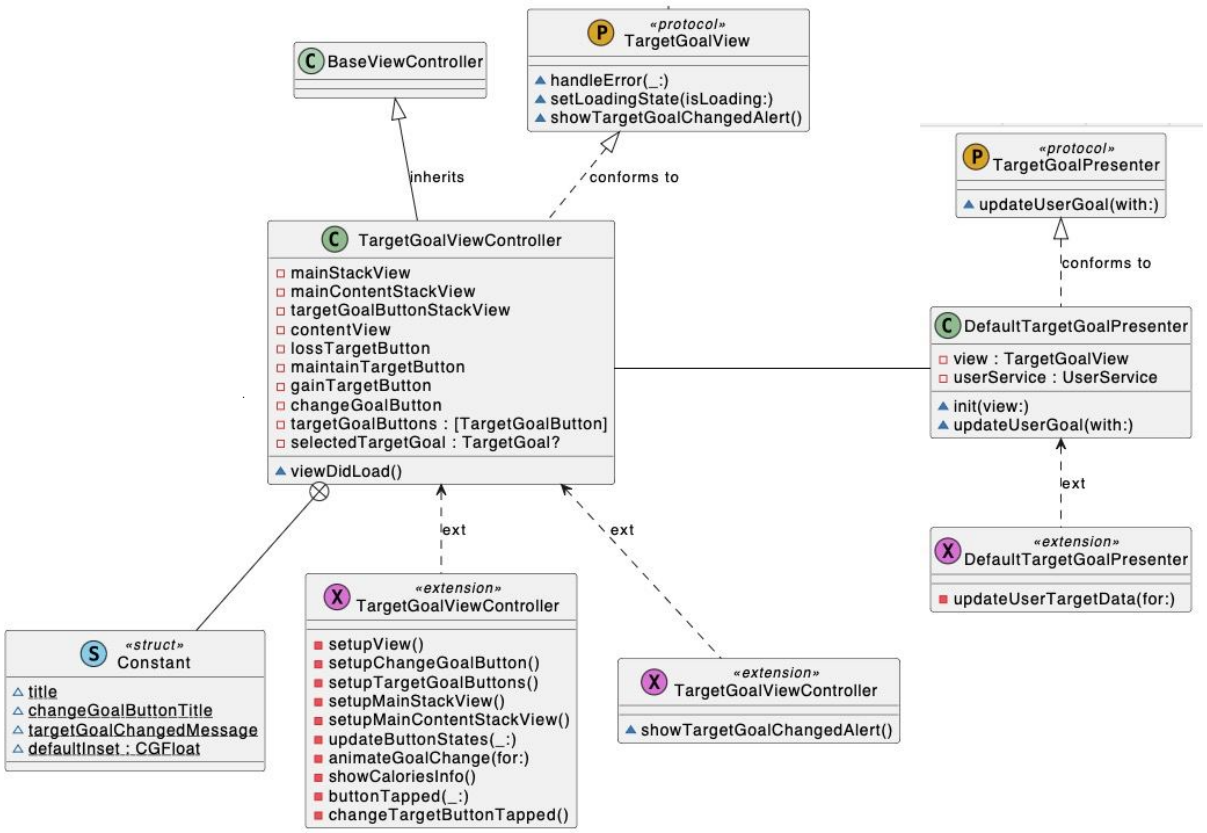

Рисунок 10 — *Діаграма класів модулю TargetGoalScreen*

3.6 Структури даних

База даних Firebase Firestore для цього додатка складається з двох основних таблиць: "usersTarget" та "usersPersonalInfo". Кожна з цих таблиць має спеціальні поля, які забезпечують ефективне зберігання та доступ до потрібної інформації.

Таблиця "usersPersonalInfo" зберігає особисті дані користувачів, а "usersTarget" призначена для зберігання інформації, пов'язаної з цілями та досягненнями користувача щодо харчування.

Таке структуроване зберігання даних у Firebase дозволяє додатку ефективно керувати інформацією, надаючи користувачам зручний доступ до своїх особистих даних та інформації про досягнення їхніх цілей.

Перейдемо до детального огляду кожної з таблиць.

Колекція “usersTarget”:

1. Поле “userId” (string): унікальний ідентифікатор користувача.
2. Поле “dailyCaloriesConsumed” (integer): кількість спожитих калорій за день.
3. Поле “dailyCaloriesTarget” (integer): цільова кількість калорій на день.
4. Поле “dailyFoodDetails” (array): масив, що містить інформацію про спожиті продукти за день.
5. Поле “targetGoal” (string): дієтична ціль.
6. Поле “weeklyCaloriesTarget” (integer): цільова кількість калорій на тиждень.

Колекція "usersPersonalInfo”:

1. Поле “userId” (string): унікальний ідентифікатор користувача.
2. Поле “activity” (integer): рівень активності користувача.
3. Поле “age” (integer): вік користувача.
4. Поле “gender” (string): стать користувача.
5. Поле “height” (integer): зріст користувача в сантиметрах.
6. Поле “weight” (integer): вага користувача в кілограмах.

ER діаграму цих таблиць можна побачити на рисунку 11.

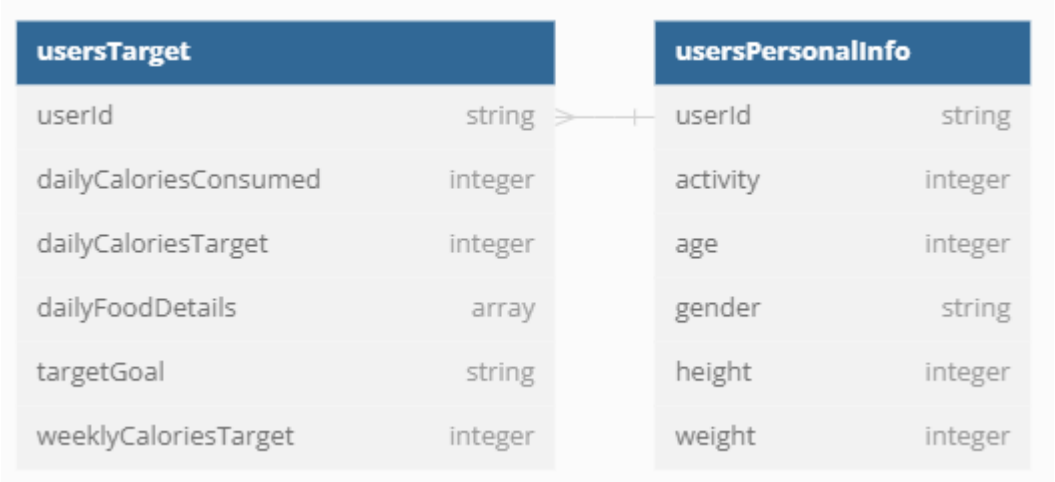

Рисунок 11 — *ER діаграма*

У додатку використовується структура DailyFoodDetails, яка зберігає інформацію про спожиті продукти.

Структуру DailyFoodDetails можна побачити на лістингу 5.

Лістинг 5 *Структура DailyFoodDetails*

```
struct DailyFoodDetails: Codable {
    let foodId: String
    let name: String
    let nutrients: Nutrients
    let image: String?
    var quantity: Int
}
```

Структура складається з таких полів:

1. Поле "foodId" (string): унікальний ідентифікатор продукту.
2. Поле "name" (string): назва продукту.
3. Поле "nutrients" (Nutrients): інформація про поживні речовини продукту.

4. Поле "image" (string?): URL або шлях до зображення продукту (опціонально).
5. Поле "quantity" (integer): кількість спожитого продукту.

У структурі DailyFoodDetails також використовується вкладена структура Nutrients, яка зберігає інформацію про поживні речовини продукту.

Структуру Nutrients можна побачити на лістингу 6.

Лістинг 6 *Структура Nutrients*

```
struct Nutrients: Codable {
    let energy: Double?
    let protein: Double?
    let fat: Double?
    let carbohydrates: Double?
}
```

Структура Nutrients складається з таких полів:

1. Поле "energy" (Double?): енергетична цінність продукту (калорії).
2. Поле "protein" (Double?): вміст білків у продукті.
3. Поле "fat" (Double?): вміст жирів у продукті.
4. Поле "carbohydrates" (Double?): вміст вуглеводів у продукті.
5. Поле "fiber" (Double?): вміст клітковини у продукті.

3.7 Проєкт інтерфейсу

Сучасний iOS-додаток для контролю калорійності дієти вимагає ретельно продуманого та зручного інтерфейсу. Чіткий та привабливий дизайн не тільки підвищує ефективність використання додатку, але й значно впливає на задоволеність користувачів, сприяючи їх тривалому та регулярному використанню.

Процес створення інтерфейсу починається з детального аналізу потреб та очікувань користувачів. Це включає збір вимог, розробку початкових прототипів та їх тестування з реальними користувачами. На основі отриманих відгуків інтерфейс постійно покращується.

Використання сучасних технологій, таких як Swift та UIKit, дозволяє створювати ефективні та швидкодіючі елементи інтерфейсу. Архітектурний підхід MVP забезпечує простоту підтримки та масштабованість додатку, що є особливо важливим у контексті постійного розширення функціональності.

Для надійного зберігання даних та швидкого доступу до них використовується Firebase, який забезпечує синхронізацію інформації між різними пристроями користувача. Інтеграція з Edamam API надає детальну інформацію про калорійність та поживну цінність продуктів, що значно спрощує користувачам процес планування дієти.

Результатом комплексного підходу до розробки є інтерфейс, який дозволяє користувачам легко відстежувати калорійність своїх прийомів їжі, встановлювати персональні цілі та отримувати рекомендації щодо здорового харчування. Завдяки зручності та інтуїтивній зрозумілості інтерфейсу, користувачі можуть зосередитися на досягненні своїх цілей без зайвих труднощів у використанні додатку.

Детальніше інтерфейс та його ключові елементи представлені на рисунках 12 — 19.

Рисунок 12 — *Головний екран для неавторизованого статусу*

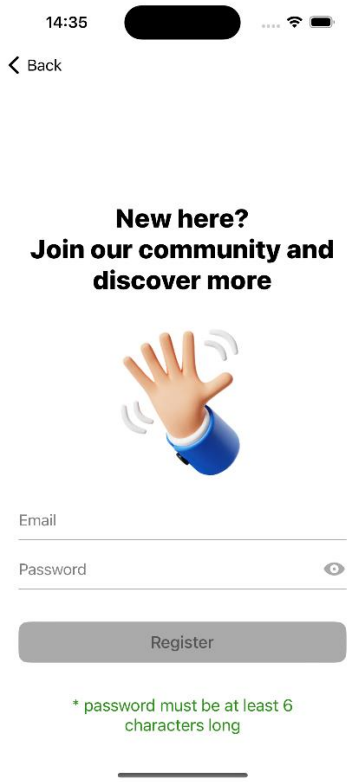

Рисунок 13 — *Екран реєстрації нового акаунту*

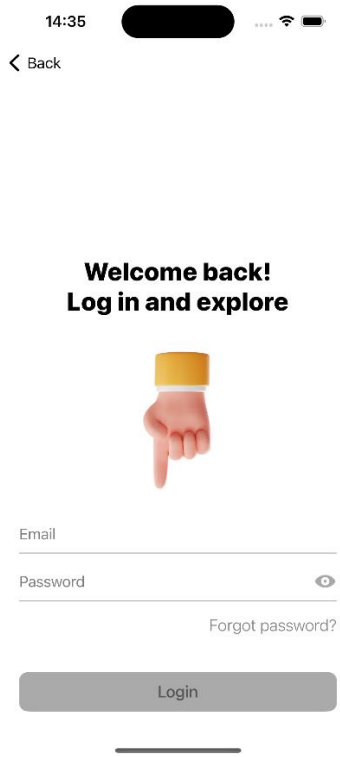

Рисунок 14 — *Екран входу існуючого акаунту*

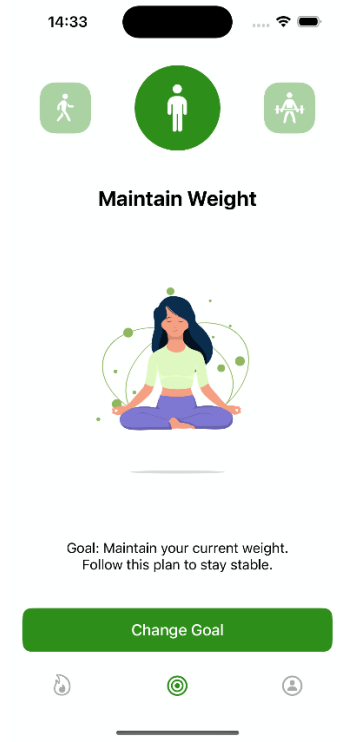

Рисунок 15 — *Екран вибору дієтичної цілі*

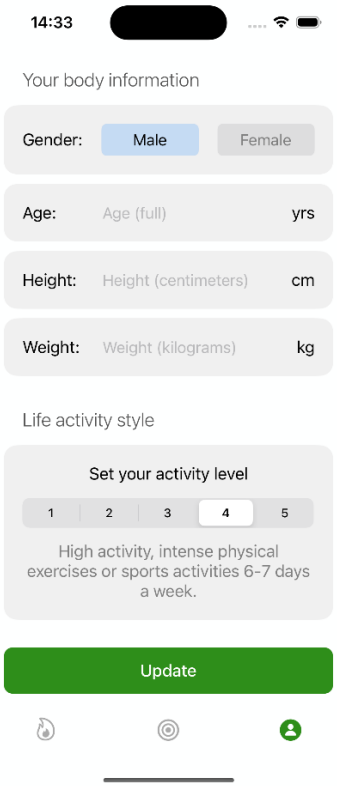

Рисунок 16 — *Екран заповнення персональних даних*

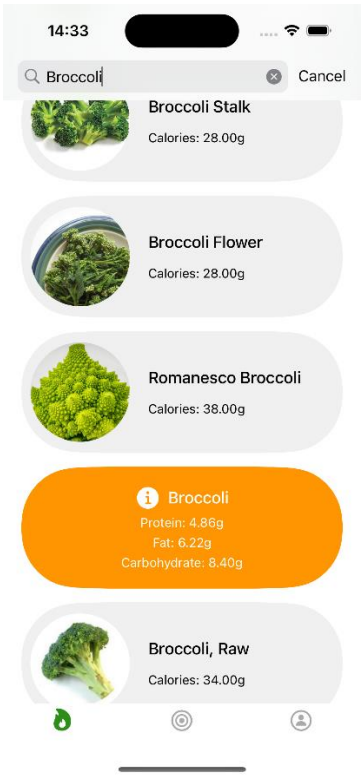

Рисунок 17 — *Екран пошуку продуктів*

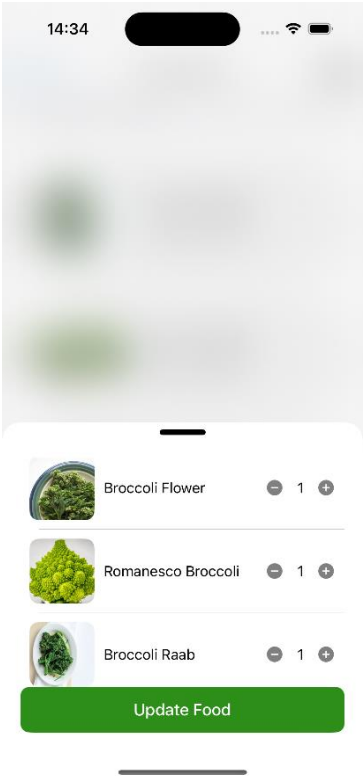

Рисунок 18 — *Корзина для вибору продуктів*

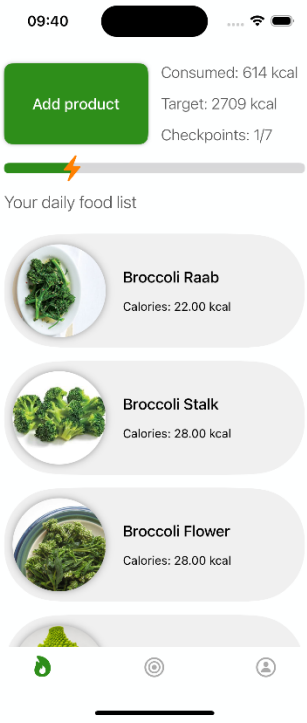

Рисунок 19 — *Головний екран для авторизованого статусу*

3.8 Реалізація і тестування

Реалізація

Фізичні характеристики поточної версії системи:

- Об'єм коду: приблизно 5 600 рядків.
- Розмір: 26 мегабайт.
- Кількість повних модулів: 7.
- Продуктивність: завдяки оптимізації алгоритмів та використанню ефективних бібліотек, завантаження основного екрану додатку займає менше ніж півтори секунди на сучасних пристроях, а саме iPhone 15 і новіші версії.

Тестування

Для забезпечення високої якості та стабільності додатку було проведено тестування на різних пристроях з версіями iOS від 15 до 17. Включало перевірку основних сценаріїв взаємодії користувача, таких як:

1. Реєстрація та авторизація.
2. Перегляд списку продуктів.
3. Налаштування персональних даних.
4. Вибір дієтичної цілі.
5. Додавання та видалення продуктів з корзини, зміна їх кількості.

Додаток пройшов ручне тестування для перевірки всіх функціональних можливостей та інтерфейсу, що підтвердило його високу стабільність та продуктивність, забезпечуючи надійний інструмент для контролю калорійності дієти користувачів.

ВИСНОВКИ

1. Під час виконання кваліфікаційної роботи було детально вивчено літературу, присвячену дослідженням у сфері мобільних додатків для контролю дієти, а також проведено аналіз існуючих аналогів на ринку.
2. У процесі розробки використовувалась мова програмування Swift, яка є ефективним інструментом для створення iOS-додатків. Використання фреймворку UIKit дозволило створити інтерактивний та зручний інтерфейс для користувачів.
3. Під час проєктування архітектури було застосовано архітектурний шаблон MVP, використання якого дало змогу розділити функціональність додатку на незалежні модулі. Це значно полегшило процес розробки, тестування та подальшої підтримки.
4. Реалізація функціональності включала створення модулів для інтеграції з Edamam API, обробки даних користувачів та збереження інформації про продукти та їх калорійність. Для бекенд-інфраструктури було використано Firebase, що забезпечує надійне збереження та обробку даних.
5. У результаті роботи було успішно створено iOS-додаток для контролю калорійності дієти. Розроблений додаток надає користувачам виконувати аутентифікацію, заповнювати персональні дані для подальших розрахунків, встановлювати особисті калорійні цілі, переглядати список продуктів, додавати продукти до кошика та щоденника, а також переглядати свій прогрес у досягненні бажаного калорійного результату.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Smartphone Ownership Is Growing Rapidly Around the World, but Not Always Equally. Pew research site. URL: https://www.pewresearch.org/global/2019/02/05/smartphone-ownership-is-growing-rapidly-around-the-world-but-not-always-equally (дата звернення: 14.01.2024).
2. Worldwide obesity rates see 'startling' increase over past 3 decades. Medical news today site. URL: https://www.medicalnewstoday.com/articles/277450 (дата звернення: 14.01.2024).
3. iPhone vs Android User Stats (2024 Data). Exploding topics site. URL: https://explodingtopics.com/blog/iphone-android-users (дата звернення: 14.01.2024).
4. Нєстєров Д. Р., Полякова Н. П., доцент, к.т.н.. Розробка iOS-додатку для контролю калорійності дієти. Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених «Молода наука-2024» / Запорізький національний університет. Запоріжжя: ЗНУ, 2024. Т.5. С. 125-126.
5. Obesity and overweight. WHO site. URL: https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight (дата звернення: 15.01.2024)
6. Weight Loss and Weight Management. Nutrition site. URL: https://nutrition.org/weight-loss-and-weight-management (дата звернення: 15.01.2024)
7. Weight-Wise Pilot Study. Center for Health Promotion and Disease Prevention site. URL: https://hpdp.unc.edu/completed-research/weight-wise-pilot-study (дата звернення: 15.01.2024).
8. A Smartphone App-Based Lifestyle Change Program for Prediabetes (D'LITE Study) in a Multiethnic Asian Population: A Randomized Controlled Trial. Frontiersin site. URL:

https://www.frontiersin.org/articles/10.3389/fnut.2021.780567/full (дата звернення: 15.01.2024)

9. Effect of Low-Calorie Diet and Regular Exercise on Weight Loss. JAMA Network site. URL: https://jamanetwork.com/journals/jama/fullarticle/2673150 (дата звернення: 15.01.2024).
10. Mediterranean Diet and Physical Activity Improve Cardiovascular Health and Promote Weight Loss. The Lancet site. URL: https://www.thelancet.com/pdfs/journals/landia/PIIS2213-8587(16)30085-7.pdf (дата звернення: 15.01.2024).
11. MyFitnessPal. MyFitnessPal site. URL: https://www.myfitnesspal.com (дата звернення 16.01.2024)
12. Lose it!. Lose it site. URL: https://www.loseit.com (дата звернення: 16.01.2024)
13. Lifesum. App store site. URL: https://apps.apple.com/us/app/lifesum-healthy-diet-plan/id286906691 (дата звернення: 16.01.2024)
14. Objective-C. Wikipedia site. URL: https://uk.wikipedia.org/wiki/Objective-C (дата звернення: 31.01.2024)
15. Swift. Wikipedia site. URL: https://uk.wikipedia.org/wiki/Swift (дата звернення: 31.01.2024)
16. UIKit. Apple Developer site. URL: https://developer.apple.com/documentation/uikit (дата звернення: 02.02.2024)
17. SwiftUI. Apple Developer site. URL: https://developer.apple.com/xcode/swiftui/ (дата звернення: 02.02.2024)
18. iOS Swift: MVP Architecture. Medium site. URL: https://saad-elaoulladi.medium.com/ios-swift-mvp-architecture-pattern-a2b0c2d310a3 (дата звернення: 02.02.2024)

19. MVVM in iOS Swift. Medium site. URL: h https://medium.com/@zebayasmeen76/mvvm-in-ios-swift-6afb150458fd (дата звернення: 02.02.2024)
20. VIPER Architecture in Swift: Building Robust iOS Apps. Medium site. URL: https://medium.com/@dickareynaldisiahaan/viper-architecture-in-swift-building-robust-ios-apps-ed30d673d459 (дата звернення: 02.02.2024)
21. Package Manager. Swift site. URL: https://www.swift.org/documentation/package-manager (дата звернення: 02.02.2024)
22. WHAT IS COCOAPODS. Cocoapods site. URL: https://cocoapods.org/ (дата звернення: 02.02.2024)
23. Carthage: A simple, decentralized dependency. Github site. URL: https://github.com/Carthage/Carthage (дата звернення: 02.02.2024)
24. Make your app the best it can be with Firebase and generative AI. Google Firebase database site. URL: https://firebase.google.com/ (дата звернення: 05.03.2024)
25. PostgreSQL: The world's most advanced open source database. PostgreSQL site. URL: https://www.postgresql.org/ (дата звернення: 05.03.2024)
26. MongoDB: The Developer Data Platform. MongoDB site. URL: https://www.mongodb.com/ (дата звернення: 05.03.2024)
27. Edamam - Food Database API, Nutrition API and Recipe API. Edamam API site. URL: https://www.edamam.com/ (дата звернення: 06.03.2024)
28. Nutritionix. Nutritionix API site. URL: https://www.nutritionix.com/ (дата звернення 06.03.2024)
29. Spoonacular. Spoonacular API site. URL: https://spoonacular.com/food-api (дата звернення: 08.03.2024)
30. Harris-Benedict Equation. Bmicalc site. URL: https://bmicalc.org/resources/harris-benedict-equation (дата звернення: 08.03.2024)

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Нєстєров Денис Романович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz20bd-211@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «Розробка iOS-додатку для контролю калорійності дієти» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 14.06.2024 _____ Нєстєров Денис Романович
(підпис) (прізвище та ініціали) (студент)

Дата 15.06.2024 _____ Полякова Наталія Петрівна
(підпис) (прізвище та ініціали) (керівник)