Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка мобільного додатку зчитування фізіологічних даних людини»

Студента 2м курсу, 5 групи,
спеціальності
121 «Інженерія програмного забезпечення»
спеціалізації
«Інженерія програмного забезпечення»
Вишняка Ярослава Олексійовича

_____
підпис студента

Науковий керівник
К.е..н., доцент Палагута К.О.

_____
підпис керівника

Гарант освітньої програми
д.т.н., професор Криворучко О.В.

_____
підпис керівника

КИЇВ – 2019

ТЕХНІЧНЕ ЗАВДАННЯ

«Розробка мобільного додатку зчитування фізіологічних даних людини»

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: «FitBack».

Галузь застосування: інформаційні технології.

Скорочене найменування системи - «iOS-додаток», «Додаток», «Застосунок».

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на ВКР, затверджене кафедрою програмної інженерії та кібербезпеки КНТЕУ.

3. ПРИЗНАЧЕННЯ І ЦІЛІ РОЗРОБКИ

3.1. Призначення інформаційної системи.

Основним призначенням додатку є створення клієнт-серверного прикладного програмного забеспечення, що дозволяє ефективно тренуватися.

3.2. Мета інформаційної системи.

Метою є створення iOS-додатку з функціоналом вимірювання фізіологічнх показників.

3.3. Цільова аудиторія інформаційної системи.

Цільова аудиторія інформаційної системи представлена наступними групами користувачів:

- Люди, котрі займаються фізичними навантаженнями.
- Люди, котрим цікаво покращити свої результати в спорті.
- Тренери.

4. ВИМОГИ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

4.1. Вимоги до стилістичного оформлення додатку

Стилістичне оформлення додатку має відповідати всім сучасним тенденціям дизайну та суворо дотримуватись рекомендацій Apple – Human Interface Guidelines.

4.2. Вимоги до графічного дизайну додатку.

- Дизайн інформаційної системи має бути в першу чергу зрозумілий для користувача, асоціативний.
- Дизайн має використовувати контрастні кольори, щоб додатком могли користуватись навіть люди з вродженими вадами зору.

4.3. Вимоги до шрифтового оформлення інформаційної системи.

Основними шрифтовими гарнітурами стилю є гарнітури Helvetica та San-Francisco.

4.4. Вимоги до операційної системи

- iOS версії 13.0 і вище
- WatchOS 6.0 і вище

5. СТРУКТУРА ІНФОРМАЦІЙНОЇ СИСТЕМИ

Логічна структура додатку представлена на рис. 1.1.

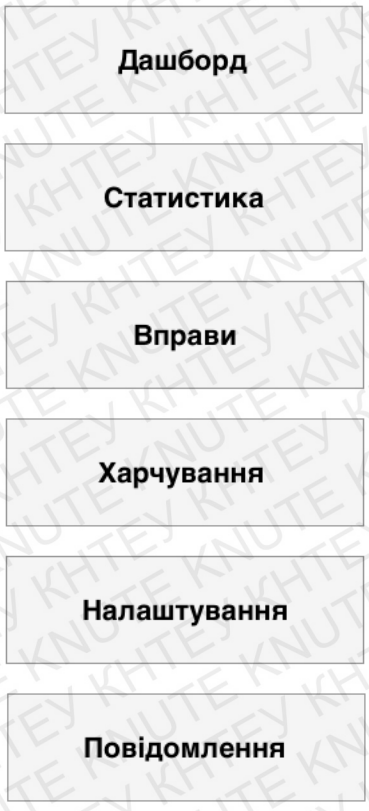


Рис. 1.1. Загальна структура інформаційної системи

АНОТАЦІЯ

До випускної кваліфікаційної роботи Вишняка Ярослава Олексійовича
на тему «Розробка мобільного додатку зчитування
фізіологічних даних людини»

У випускній кваліфікаційній роботі розглядається розробка мобільного додатку для платформи iOS з використанням клієнт-серверної архітектури, що включає мову програмування Swift, серверну БД Firebase і інтерфейси для зчитування даних з датчиків Apple Watch, зокрема HealthKit.

Ключові слова: Swift, iOS, WatchOS, Firebase, HealthKit, Data-Driven, VIPER, клієнт-серверна архітектура, мова програмування, база даних.

ANNOTATION

Prior to the final qualification work of Vyshnyak Yaroslav Alekseevich
on the topic "Development of human physiological data reader"

The final qualification examines the development of a mobile application for the iOS platform using a client-server architecture, including Swift programming language, Firebase server database and interfaces for reading data from Apple Watch sensors, including HealthKit.

Keywords: Swift, iOS, WatchOS, Firebase, HealthKit, Data-Driven, VIPER, client-server architecture, programming language, database.

ЗМІСТ

					КНТЕУ 121– 05-05.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*	*Розробка мобільного додатку зчитування фізіологічних даних людини*	*Стадія*	*Аркуш*	*Аркушів*
Зав. каф.		Криворучко О.В.				СП		
Керівник		Палагута К.О.						
Гарант		Криворучко О.В			*Список використаних літературних джерел*	Факультет обліку, аудиту та інформаційних систем, 2м курс, 5 група		
Розроб		Вишняк Я.О.						

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Sam Daves, Jim Rames: iOS10 by Tutorials, 2016
2. Derek Selander: Advanced Apple debugging by reverse engineering, 2018
3. Eric Kember, Ben Morrow, Steven Van Impe: Swift Apprentice, 2017
4. Alexander Svets: Design Patterns, 2010
5. Jon Hoffman: Swift 4 Protocol-Oriented Programming, 2017
6. Kelvin Lau: Data Structures & Algorithms in Swift, 2017
7. Marin Todorov: iOS Animations, 2016
8. Wallase Wang: Swift 5 development, 2019
9. Mike Katz, Joshua Greene: iOS Test-Driven Development, 2018
10. Scott Grousch: Push Notification by Tutorials, 2018
11. Etnab Amer, Scott Atkinson: WatchOS by Tutorials, 2017

ВСТУП

Смартфони та різна розумна електроніка сьогодні оточує людей і полегшує їм життя. Якщо раніше мобільна електроніка ставила перед собою в основному задачі комунікації та автоматизації, то на даний момент електроніка моле не тільки полегшувати, а й суттєво поліпшувати різні сфери людської життєдіяльності.

Починаючи з появи фітнес-треккерів та різних розумних годинників постало питання в прикладному використанні даних гаджетів. Найпопулярнішим варіантом став «моніторинг здоров'я», котрий дозволяв людині як слідкувати за своїми фізичними показниками так і ефективніше і якісніше жити. На даний момент деякі розумні годинники можуть навіть передбачити серцевий напад чи навіть діагностувати цукровий діабет, і на відміну від медичного персоналу електроніка завжди знаходиться з людиною і здатна до цілодобового моніторингу. Тож вся індустрія мобільних пристроїв рухається в цьому напрямку зараз.

Метою даної роботи є розробка мобільного додатку зчитування фізіологічних даних людини, що дозволить вибудувати персональну стратегію харчування і тренування.

Об'єктом дослідження є фізична діяльність людини і її характерні особливості.

Предмет дослідження - це методи та алгоритми для зчитування, класифікації та обробки фізіологічних даних під час фізичної активності.

РОЗДІЛ 1 ОСНОВНІ ПОНЯТТЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ ТА ДОДАТКІВ-СУПУТНИКІВ ДЛЯ ЗЧИТУВАННЯ БІОЛОГІЧНИХ ДАНИХ

1. Основи прикладного застосування мобільних додатків у цілях моніторингу здоров'я

Сьогоднішні мобільні пристрої розроблені так, щоб ми могли отримувати миттєвий зворотній зв'язок з усіх питань, які нас найбільше хвилюють, тому не дивно, що додатки, пов'язані із здоров'ям та здоров'ям, є одними з найпопулярніших завантажень. Деякі додатки допомагають аналізувати симптоми, деякі програми допомагають отримати найкращу ціну за рецептами, а деякі додатки просто сприяють здоровому способу життя. Хоча медичні та медичні додатки ніколи не повинні покладатися на заміну професійної діагностики чи належного медикаментозного лікування, вони можуть бути корисними інструментами для подолання розриву знань у сфері, яка є життєво важливою для всіх нас.

Починаючи з появи фітнес-трекерів та розумних годинників почався період надточного моніторингу життєвих показників людини за допомогою мобільних пристроїв. Це створило багато можливостей для прогнозування систематичних професійних тренувань та діагностики стану здоров'я. З'явилося багато операційних систем для розумних годинників і прикладного програмного забезпечення у вигляді додатків для них.

Серед них є як і прості додатки для нагадування поповнювати водний баланс так і створити систему тренувань відповідно стану фізичної підготовки власника мобільного пристрою.

Наразі вже є технічна можливість підключатись до зовнішніх медичних приладів через такі інтерфейси як Bluetooth чи мережу Wi-Fi, що дозволяє більше деталізувати результати, створити ширшу вибірку фізіологічних даних. Це дозволяє краще продемонструвати поточний стан потенційного пацієнта та провести більш ефективну діагностику. Також за допомогою відслідковування рухів, спалення калорій, інтенсивності тренувань та інших показників дозволяє мати репрезентативне уявлення про результати своєї спортивної або повсякденної діяльності людини, адже датчики можуть моніторити стан людини двадцять чотири години на добу, включаючи навіть сон і його показники: час сну, тривалість різних фаз сну. Основні показники, котрі може моніторити та вираховувати сучасне програмне забезпечення:

- Серцевий ритм
- Час спокою
- Час збудження
- Біологічні ритми
- Побудова енцефалограми
- Інтенсивність навантажень
- Кількість кроків
- Зміщення в просторі

На основі більшості цих показників можна формувати висновок про поточну критичну ситуацію – падіння чи удар, і відповідно реагувати: додаток викликає швидку чи надсилає повідомлення

довіреному контакту. В той час як енцефалограма дозволяє діагностувати наближення серцевого нападу.

За своєю суттю ЕКГ вимірює активність електричного потенціалу в серце. Під час биття електричний потенціал рухається по серцю, і це видно як сплесків і опуклостей на ЕКГ. У цій ЕКГ використовується 10 електродів: 4 для кожної з кінцівок, і 6, розташованих певним чином навколо серця. Вони дають 12 окремих відведень по всьому серцю. В якомусь сенсі, таку ЕКГ можна уявити собі як тривимірне вимір електричної серцевої активності. У використовуваних в госпіталях кардіомоніторах зазвичай використовується від 3 до 5 електродів, що обмежує кількість вимірюваних відведень, але все одно залишається корисним для діагностичних тестів. ЕКГ в Apple Watch є ЕКГ з одним датчиком, що вимірює відведення I. Це прекрасно підходить для вимірювання швидкості і ритму роботи серця, що може виявитися корисним для обстеження на предмет фібриляції передсердь. У поточному варіанті Apple Watch буде постійно відслідковувати биття серця за допомогою оптичного датчика, і якщо годинник виявлять аритмію, вони видадуть попередження, після якого власник зможе пройти обстеження на ЕКГ. Фібриляція передсердь, або ФП - найбільш поширений вид серцевої аритмії, і 25% людей старше 40 років повинні зіткнутися, по крайней мере, з одним епізодом ФП. Він відбувається в разі переривання нормального шляху електричної активності.

ФП може відбуватися періодично, такі епізоди можуть тривати по кілька хвилин, після чого серце може самостійно відновлювати ритм. Деякі з них можуть відбуватися внаслідок вживання алкоголю, хвороби або інших проблем зі здоров'ям. У деяких людей епізоди ФП з подальшим самовідновлення ритму можуть відбуватися регулярно, і причини цього невідомі - це стан також відомо, як пароксизмальна

форма ФП. Екстремальний варіант такого стану - постійно присутня ФП, при якій буває дуже важко повернутися до нормального ритму.

Саме по собі наявність ФП не обов'язково означає критичну ситуацію. У багатьох людей з ФП повністю відсутні які б то не було симптоми, інші можуть періодично випробовувати почастішання пульсу або його нерегулярність.

ФП стає небезпечною, коли на її тлі відбуваються відхилення життєво важливих показників, коли тиск у людини падає настільки, що він може втратити свідомість, він може відчувати задишку, або небезпечно високий пульс. Якщо ваш пульс перевищує 100-110 ударів в хвилину, ви відчуваєте ФП з тахисистолией. У цей момент лікарі невідкладної допомоги могли б контролювати ваш пульс за допомогою вводяться внутрішньовенно ліків.

І хоча короткі періоди безсимптомною ФП можуть бути безпечними, постійна ФП збільшує ризик інфаркту, тромбів в легенях і зупинки серця. Залежно від чинників ризику пацієнта, деяким може знадобитися розрідження крові. Чим довше ваше серце знаходиться в стані неконтрольованої ФП, тим складніше звернути серцеві зміни.

Також важливим є оцінка і замір біологічних ритмів.

Біологічні ритми - періодично повторювані зміни характеру і інтенсивності біологічних процесів та явищ у живих організмах. Біологічні ритми фізіологічних функцій настільки точні, що їх часто називають «біологічним годинником». Ритми, що задаються внутрішніми «годинами» або водіями ритму, називаються ендогенними, на відміну від екзогенних, які регулюються зовнішніми

факторами. Більшість біологічних ритмів є змішаними, тобто. Е. Частково ендогенними і частково екзогенними. У багатьох випадках головним зовнішнім фактором, що регулює ритмічну активність. Тривалість світлового дня. Це єдиний фактор, який може бути надійним показником часу, і він використовується для установки «годин». Конкретна природа «годин» невідома, але немає сумнівів, що тут діє фізіологічний механізм, який може включати як нервові, так і ендокринні компоненти. Більшість ритмів формуються в процесі індивідуального розвитку (онтогенезу). Так, добові коливання активності різних функцій у дитини спостерігаються до його народження, їх можна зареєструвати вже у другій половині вагітності.

Днем зростає частота серцевих скорочень (ЧСС), вище артеріальний тиск (АТ), частіше дихання. День у день до моменту пробудження, як би передбачаючи зростаючу потребу організму, в крові підвищується вміст адреналіну - речовини, яке збільшує ЧСС, підвищує артеріальний тиск, активізує роботу всього організму; до цього часу в крові накопичуються біологічні стимулятори. Зниження концентрації цих речовин до вечора - неодмінна умова спокійного сну. Недарма порушення сну завжди супроводжуються хвилюванням і тривогою: при цих станах в крові наростає концентрація адреналіну і інших біологічно активних речовин, організм тривалий час знаходиться в стані «бойової готовності». Підкоряючись біологічним ритмам, кожен фізіологічний показник протягом доби може істотно змінювати свій рівень.

За сукупністю усіх цих параметрів можна прогнозувати проблеми зі здоров’ям і можливих загроз пов’язаних з образом життя. Але на даний момент доступні для користувача датчики мають великі обмеження в силу їх ергономічних особливостей, конструктивних параметрів і економічної ситуації на ринку мобільних пристроїв.

Чого не можуть Apple Watch:

- На даний момент Apple Watch з функцією ЕКГ не призначені для визначення інших проблем з серцем, крім ФП.
- Вони також не підходять для людей, яким вже поставили діагноз ФП - їм необхідно регулярно відвідувати лікаря.
- Вони НЕ МОЖУТЬ точно виявити ризик інфаркту. Навіть повна ЕКГ з 12 відведеннями може пропустити певні ознаки інфаркту.
- Вони не вважаються пристроєм, схваленим Управлінням із санітарного нагляду за якістю харчових продуктів і медикаментів США (FDA). FDA просто випускає дозволу, «попередні форми схвалення 510к до виходу продукту на ринок», в яких недвозначно написано, що пристрій не призначений для людей молодше 22 років. Пристрій вважається апаратом для домашнього використання класу II - в цей клас входять презервативи і тести на вагітність. [В минулому році FDA схвалила спеціальний ремінець для Apple Watch, що вимірює роботу серця, зазначивши його як медичний аксесуар / прим. перев.]
- Вони не є пристроєм для постійного відстеження електричної активності серця. Вони можуть відстежувати ЕКГ, тільки коли ви другою рукою торкаєтеся до коліщатка.
- ЕКГ з одним електродом побудувати фізично неможливо. Для вимірювання електричної активності необхідно організувати замкнутий контур, що проходить через серце. З цим не впорається навіть бездротове пристрій, надіте на іншу руку, оскільки воно не буде частиною того ж контура.

В цілому Apple Watch виглядають як прекрасний інструмент, але вони не належать до апаратів медичного класу, і не замінять професійної медичної оцінки в разі появи симптомів. І навіть якщо ЕКГ на ваших Apple Watch виглядає нормально, це не означає, що у вас немає ФП або інших серцевих аномалій.

1.2. Технічне забезпечення для вимірювання фізіологічних даних

Основні апаратні складові розумного годинника Apple Watch, котрі надають дані:

- Акселерометр
- Гіроскоп
- Альтиметр
- Фотодавач
- Оптичний датчик пульсу
- Датчик ЕКГ
- GPS

Акселерометр - це прилад для вимірювання сили реакції індукованої прискоренням або гравітацією. Одно- та багато-вісні моделі можуть визначати величину та напрям прискорення у вигляді векторної величини і тому можуть бути використані для визначення орієнтації, вібрації й ударів. Акселерометр вимірює проекцію повного прискорення. Повне прискорення є рівнодіючою сил негравітаційної природи, що діє на масу, віднесеної до величини цієї маси. Акселерометр можна застосовувати як для того, щоб вимірювати проекції абсолютного лінійного прискорення, так і посередні проекції гравітаційного прискорення. Остання властивість використовується щоб створювати інклінометри. Акселерометр входить до складу інерціальних навігаційних систем, де отримані за їх допомогою виміри інтегрують, отримуючи інерційну швидкість і координати носія. Останнім часом завдяки поширенню знань акселерометри застосовують у манекенах для вимірювання навантаження на органи людини в екстремальних умовах. Також широко застосовується в смартфонах. В першу чергу, саме завдяки акслерометрам зображення

на екрані змінює своє положення в залежності від горизонтальної або вертикальної орієнтації гаджета. Як наслідок, його наявність забезпечує як яскравий ігровий процес, так і деякі службові функції: наприклад, деякі моделі дозволяють прийняти вхідний дзвінок, легко потрусивши апарат.

Основними параметрами акселерометра є:

- Масштабний коефіцієнт - коефіцієнт пропорційності для лінійної залежності між вимірюваним удаваним прискоренням і вихідним сигналом (електричним сигналом, частотою коливань (для струнного акселерометра) або цифровим кодом).
- Робочий діапазон частот.
- Порогова чутливість (дозвіл) - величина мінімального зміни удаваного прискорення, яке здатний визначити прилад.
- Зсув нуля - різниця між показаннями приладу і проекцією гравітаційного прискорення на вісь чутливості при нульовому уявній прискоренні.
- Випадкове блукання - середньоквадратичне відхилення від зсуву нуля.
- Нелінійність - відхилення залежності між вихідним сигналом і позірним прискоренням від лінійної при зміні удаваного прискорення.

На величину вихідного сигналу акселерометра в основному впливають:

- температура навколишнього середовища і місця кріплення акселерометра (температурні похибки);

- зовнішні магнітні поля (похибки від магнітного поля);
- вібрація і кутові коливання підстави (вібраційні похибки);
- частотні характеристики акселерометра (частотні похибки);
- гистерезис показань (одна зі складових нелінійності).

Альтиме́тр, висотомір - інструмент, що використовується для вимірювання висоти підйому або висоти над рівнем моря, зокрема — у літальних апаратах. Поширена форма - барометр-анероїд, що працює за принципом визначення різниці тиску повітря на різній висоті.

GPS, Систе́ма глоба́льного позиціонува́ння (англ. Global Positioning System) — сукупність радіоелектронних засобів, що дозволяє визначати положення та швидкість руху об'єкта на поверхні Землі або в атмосфері. Положення об'єкта обчислюється завдяки використанню розміщеного на ньому GPS-приймача, який приймає та обробляє сигнали супутників космічного сегменту GPS-системи глобального позиціонування. Для визначення точних параметрів орбіт супутників та керування GPS-системою вона в своєму складі має наземні центри управління.

Коли мова йде про GPS, найчастіше мається на увазі система NAVSTAR, розроблена на замовлення військового відомства — Міністерства оборони США, але зараз існують або розробляються також інші системи глобального позиціонування (ГЛОНАСС, Galileo та інші). Станом на сьогодні основою системи NAVSTAR (Navigation Satellite Time and Ranging) є 31 супутник (всього наявно 32), що працюють у єдиній мережі й обертаються на шести різних кругових орбітах, розташованих під кутом 60° одна до одної. На кожній орбіті

розміщено по 4 супутники, висота орбіт приблизно дорівнює 20 200 км а період обертання кожного супутника навколо землі дорівнює 12 годинам. Таким чином, із будь-якої точки земної поверхні зазвичай одночасно видно від чотирьох до дванадцяти таких супутників.

Супутники перебувають під контролем станцій, які розташовані на Землі. Розміщуються такі станції на Колорадо-Спрінгз, Дієго-Гарсія, острові Вознесіння, атолі Квадженлейн і на Гаваях. Вся інформація, що проходить через ці станції, записується ними та передається на головну станцію на військовій базі Falcon (штат Колорадо).

GPS-приймач обчислює власне місцезнаходження, вимірюючи час проходження сигналу від GPS-супутників. Кожен супутник постійно надсилає повідомлення, в якому міститься інформація про час, точку орбіти супутника, з якої було надіслано повідомлення (ефемерида), та загальний стан системи й приблизні дані орбіт усіх інших супутників системи GPS (альманах). Ці сигнали розповсюджуються зі швидкістю світла в космосі (і з трохи меншою швидкістю — в атмосфері). Приймач визначає час затримки в надходженні сигналу та обчислює відстань до супутників, виходячи з якої, застосувавши метод трилатерації, визначає своє місце. Отримані координати перетворюються в наочну форму (широта та довгота чи положення на карті) та відображаються користувачеві.

Теоретично для визначення власних координат достатньо визначити відстань до трьох супутників. Однак для обчислення положення необхідно знати час із високою точністю. Щоб усунути потребу в високоточному годиннику, отримують інформацію з 4-х чи більше супутників, тобто, GPS-приймач використовує чотири параметри для обчислення чотирьох невідомих: x, y, z та t.

У деяких окремих випадках можна обійтися меншою кількістю супутників. Якщо заздалегідь відома одна змінна (наприклад, висота над рівнем моря човна в океані дорівнює 0), приймач може обчислити положення, використовуючи дані з трьох супутників. Також на практиці приймачі використовують різну допоміжну інформацію для обчислення положення з меншою точністю в умовах відсутності одразу чотирьох супутників. Попри те, що проекти побудови GPS-систем впроваджувались військовими відомствами, зараз, окрім приймачів спеціального призначення, випускаються прилади, вмонтовані в різноманітну дрібну техніку: наручні годинники, мобільні телефони, ручні радіостанції, портативні комп'ютери та фотоапарати, за допомогою яких можна орієнтуватися на місцевості або фіксувати місцезнаходження користувача. Їх використовують альпіністи, рятівники, туристи.

Склад комплексу персонального моніторингу

До складу програмно-апаратного комплексу входять персональний трекер, сервер зі спеціальним програмним забезпеченням та пристрої кінцевих користувачів — підключені до мережі Internet персональні комп'ютери та/або мобільні телефони, що здатні виконувати програми певного типу і мають вихід у мережу Internet.

Також до комплексу входять навігаційні супутники системи GPS, мережа стільникового зв'язку GSM і всесвітня інформаційна мережа Internet. Внаслідок загальнодоступності та глобальності цих складових комплекс може бути застосований скрізь, де є:

- можливість для трекерів приймати сигнали навігаційних супутників GPS;
- наявність GSM-покриття;

- вихід в інформаційну мережу Internet.

Користувач може здійснювати моніторинг осіб (тварин, об'єктів), оснащених персональними трекерами, практично на всій території земної кулі, навіть, знаходячись при цьому на значній відстані від свого звичайного місця розташування — за умови, що виконуються вищезгадані три умови. Пристрій записує отриману GPS-інформацію з регулярними інтервалами, а потім може ці дані записувати або передавати їх за допомогою радіозв'язку, GPRS- або GSM-з'єднання чи супутникового модему на сервер підтримки або інший комп'ютер (наприклад, у вигляді SMS або через мережу Internet). У разі використання сервера підтримки, він обробляє отримані дані та реєструє їх у своїй базі даних; потім користувач трекера може зайти на сервер системи з мережі Internet під своїм ім'ям та паролем, і система відображає місцезнаходження і географію переміщення на карті. Пересування трекера можна аналізувати або в режимі реального часу, або пізніше. Функція GPS-трекінгу є в деяких моделях стільникових телефонів.

Фотодавач - пристрій, здатний виявляти електромагнітне випромінювання, та забезпечувати на виході сигнал у вигляді струму або різниці потенціалів, пропорційні інтенсивності виявленого випромінювання. снують різні типи фотодавачів, що відрізняються відповідно до різних ефектів взаємодії між випромінюванням і матеріалом. Зокрема, вони можуть різнитися у частині електромагнітного спектру, який здатні виявляти, або мінімальною силою світла, яку ними може бути виміряно (деякі з давачів, здатні виявляти поодинокі фотони). За способом використання, фотодавачі (фотоелементи) поділяються на: сигнальні пристрої для систем автоматизації (для воріт або дверей, тощо); визначальні давачі для багатьох спортивних дисциплін, пов'язаних з фотоелементами.

Фотоелементи можуть контролювати перемикання ламп відповідно до потрібної яскравості, а в 60-ті роки деякі виробники телевізорів, використовували фотодавачі, для регулювання яскравості чорно-білого зображення, відповідно до освітленості навколишнього середовища. Крім того, вони використовуються у всіх областях, у яких це потрібно, для вимірювання інтенсивності світла, наприклад, у спектроскопії або у фотометрії, чи фотографуванні. Давачі Active-піксельні (APSS) є давачами зображення. Зазвичай, використовуються у камерах мобільних телефонів, веб-камерах та деяких дзеркальних фотоапаратах. Болометри вимірюють потужність електромагнітного випромінювання за допомогою нагрівання матеріалу, з залежним від температури електричним опором. Мікроболометричний - окремий тип болометра, використовується як детектор у теплових камерах.

Зараз це найпоширеніший спосіб вимірювання пульсу з точки зору масового застосування, реалізований в спортивних годинах, трекерах, мобільних телефонах. А перші спроби використання цієї технології робилися ще в 1800-х роках.

Звуження і розширення судини під дією пульсації кровотоку викликають відповідну зміну амплітуди сигналу, одержуваного з виходу фотоприймача.

Спосіб широко використовується в лікарнях, пізніше технологія перейшла і в побутові пристрої - компактні пульсоксиметри, які реєструють пульс і насичення киснем крові в капілярах пальця. Чудово підходить для періодичних вимірювань пульсу, але абсолютно не підходить для постійного носіння.

Ідея вимірювання пульсу з зап'ястя спортсмена за допомогою оптичної плетизмографии без додаткового носіння нагрудних ремінців виглядала дуже заманливо. Першими цю ідею реалізували в годиннику Mio Alpha, які проголосили свій пристрій проривом і

новим витком у вимірі пульсу. Сам модуль вимірювального датчика був розроблений компанією Philips.

Світло, що потрапляє в кровотік, буде досить передбачувано розсіюватися при зміні швидкості кровотоку (наприклад, підвищення серцевого викиду) .Оптическая технологія вимірює пульс за допомогою світлодіодів, які оцінюють кровотік на зап'ясті. Це означає, що ви можете вимірювати пульс без використання нагрудного датчика. На практиці це працює так: оптичний сенсор на зворотному боці годинника випромінює світло на зап'ясті за допомогою світлодіодів, і вимірює кількість розсіяного кровотоком світла. Для вимірювання пульсу важлива область з максимальним поглинанням - це діапазон від 500 до 600 нм. Зазвичай вибирається значення 525 нм (зелений колір). Зелений світлодіод датчика пульсу - найбільш ходовий варіант в смарт-годинах і браслетах.

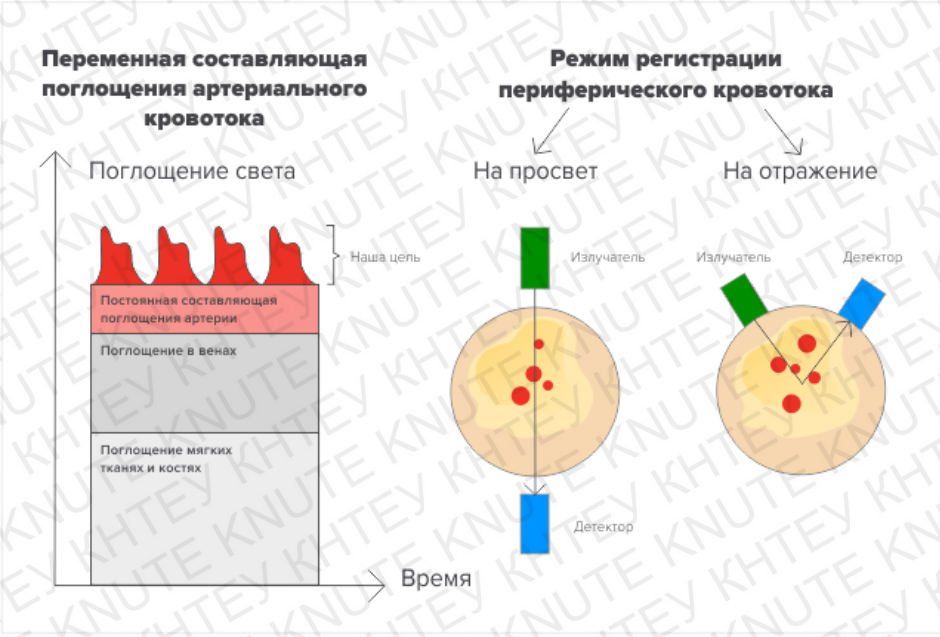


Рис. 1.1. Схема роботи датчика і випромінювача

Зараз ця технологія добре відпрацьована і впроваджена в серійне виробництво. Спектр з'явилися пристроїв з подібною технологією досить широкий (смартфони, браслети-трекери, годинник), а виробники спортивних пристроїв теж не відстають - всі найбільш значущі компанії розширюють лінійку пульсометрів моделями з оптичними датчиками.

Вважається, що оптичні датчики досить точно визначають пульс при ходьбі і бігу. Однак, при підвищенні частоти пульсу, скажімо, до 160 уд / хв, кровотік настільки швидко проходить через область датчика, що вимірювання стають менш точними.

Крім цього, на зап'ясті, де не так багато тканини, але багато кісток, зв'язок і сухожиль, будь-яке зниження кровотоку (наприклад, в холодну погоду) може спотворити роботу оптичного датчика пульсометра.

В одному невеликому дослідженні було проведено порівняльний аналіз точності нагрудних і оптичних датчиків пульсометрів. Випробовуваних розділили на дві групи, в одній групі пульс вимірювався за допомогою нагрудного датчика, а в іншій - за допомогою оптичного. Обидві групи проходили тест на біговій доріжці, де вони спочатку йшли, а потім бігли, в цьому час реєструвалася частота пульсу. У групі з нагрудним кардиодатчики точність вимірювання ЧСС була 91%, тоді як в групі з оптичним датчиком вона склала лише 85%.

На думку глави компанії Mio Global, в даний час жоден з датчиків пульсометра не зрівняється в точності з нагрудним ременем.

Не можна забувати і про специфічні ситуаціях, коли оптичний датчик може не працювати. Надіті поверх біговій куртки годинник, наявність татуювання на зап'ясті, нещільно прилеглі до шкіри

годинник, тренування в спортзалі - все це може привести до похибок у вимірюванні пульсу за допомогою оптичних датчиків.

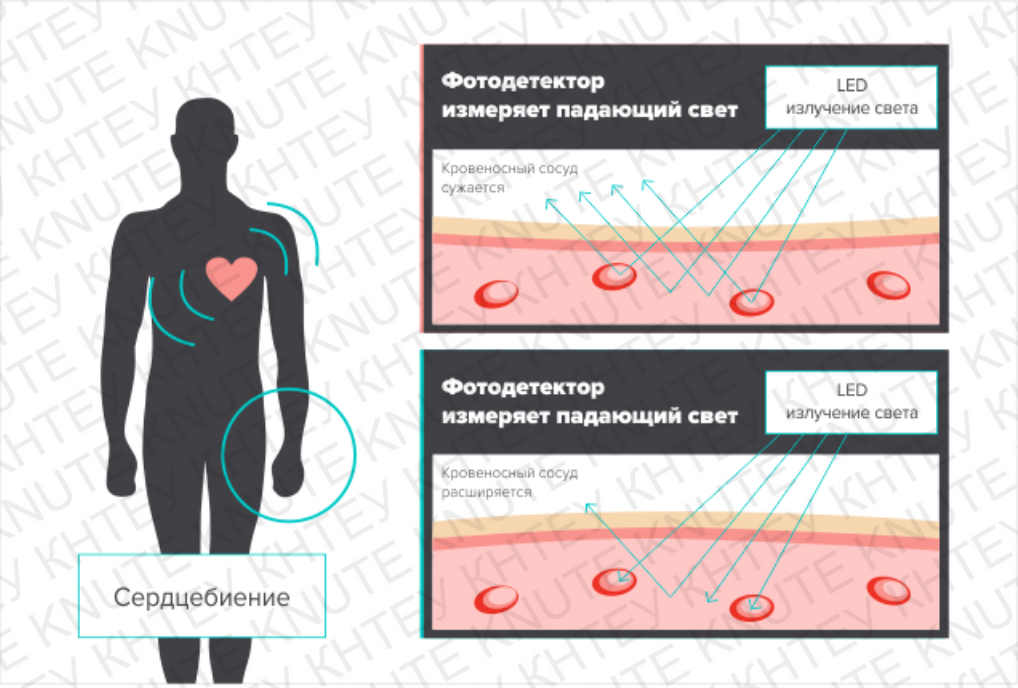


Рис. 1.2. Демонстрація роботи датчика пульсу

Незважаючи на це, технологічний прогрес у вимірі ЧСС привів до появи корисної альтернативи нагрудним ременів, і при усуненні ряду недоліків оптичних датчиків ми отримаємо ще один потужний і точний інструмент спостереження за пульсом під час занять спортом.

Строго говоря, продвинутая беговая динамика измеряется при наличии нагрудного ремня. Внешне обычный, внутри датчик состоит из трансмиттера и акселерометра, благодаря которому и происходит анализ движения бегуна. Те же самые акселерометры есть в телефонах, футподах, браслетах-трекерах.

До просунутим бігових показників відносять три величини: час контакту з землею (ground contact time), вертикальні коливання (vertical oscillation) і частоту кроків, або каденс (cadence).

Час контакту з землею (ground contact time, GCT) показує як довго ваша стопа знаходиться на поверхні землі під час кожного кроку. Вимірюється в мілісекундах. Типовий бігун любитель витрачає на контакт з поверхнею 160-300 мілісекунд. При підвищенні швидкості бігу значення GCT коротшає, при уповільненні - зростає.

Існує взаємозв'язок між часом контакту з землею і частотою розвитку травм, а також м'язовим дисбалансом у бігуна. Зменшення часу контакту з землею знижує частоту травм. Одним з найбільш дієвих способів зменшити цей показник вважається вкорочення кроку (підвищення каденса), зміцнення сідничних м'язів і включення коротких спринтів в програму тренувань.

Вертикальні коливання (vertical oscillation, VO). Подивіться на будь-якого професійного бігуна - ви побачите, що верхня половина їх тулуба робить зовсім незначні рухи, в той час як основну роботу по переміщенню бігуна виконують ноги.

Вертикальні коливання визначають наскільки ваша верхня половина «підстрибує» при бігу. Ці підстрибування вимірюються в сантиметрах щодо якоїсь фіксованої точки (в разі нагрудного ременя - це сенсор, вбудований в нагрудний датчик). Вважається, що найбільш економічна техніка бігу передбачає мінімальні вертикальні коливання, а зменшення вертикальних коливань досягається підвищенням каденса.

Частота кроків або каденс (cadence). Як зрозуміло з назви показника, він демонструє кількість кроків за хвилину. Досить важливий параметр, що оцінює економічність бігу. Чим швидше ви біжите, тим вище каденс. Вважається, що частота близько 180 кроків за хвилину є оптимальною для ефективного і економічного бігу.

Пульсові зони (heart rate zones). Знаючи максимальний пульс, різні моделі бігових годин можуть розбивати вашу тренування по

пульсовим зонам, показуючи, скільки часу в ході тренування ви провели в тій чи іншій зоні.

У різних виробників ці зони позначені по-своєму, але їх можна поділити на такі типи: відновна зона (60% від максимального ЧСС), зона для тренування витривалості (65% -70% від максимального ЧСС), зона тренування аеробного ємності (75 82% від максимальної ЧСС), зона ПАНО (82-89% від максимального ЧСС) і зона максимальної аеробного навантаження (89-94% від максимального ЧСС).

Знання своїх пульсових зон допоможе вам отримати максимум від кожного тренування. Про тренування по пульсу ми детально розповімо в наступній статті рубрики. Крім просунутих бігових характеристик сучасні пульсометри можуть вимірювати і відслідковувати ще кілька цікавих показників:

EPOC (excess post-exercise oxygen consumption). Показник споживання кисню після тренування демонструє, наскільки змінився ваш метаболізм після пробіжки. Ми всі знаємо, що біг призводить до спалювання калорій, але навіть після того, як тренування закінчилася, калорії продовжують згоряти. Безумовно, для їх заповнення потрібно якісно відновитися. Спостереження за показником EPOC допоможе вам зрозуміти, які тренування найбільш енергетично затратні, а також поліпшити процес відновлення.

Підрахована споживання кисню (est. VO2). Показник поточного споживання кисню, розрахований на підставі максимального споживання кисню і максимальної ЧСС.

Максимальне споживання кисню (VO2max). Показник відображає здатність вашого організму споживати кисень. Це важливо, оскільки при підвищенні цього показника ваше тіло може

краще і швидше утилізувати доставляється до працюючих м'язів кисень.

Значення максимального споживання кисню (МПК) збільшується при підвищенні тренованості. Це один з найважливіших бігових показників, безпосередньо пов'язаний з економічністю бігу. Як і у випадку з визначенням максимальної ЧСС, найкращим способом визначення МПК є тестування в лабораторії, але ряд виробників пульсометрів використовує алгоритми розрахунку МПК прийнятної точності. Тренування допомагають поліпшити значення цього показника.

Рис. 1.3. Набір можливих даних для аналізу

Бігова продуктивність (running performance). Показник, який використовує VO2max (глобальний стандарт аеробного тренованості і витривалості) для відстеження прогресу в тренуваннях.

Піковий тренувальний ефект (peak training effect, PTE). Показує вплив тренувальної сесії на загальну витривалість і аеробне продуктивність. Чим ви тренированнее, тим важче ви повинні тренуватися для того, щоб досягти більш високих цифр PTE.

1.3. Прикладний функціонал, заснований на моніторингу життєвих показників

Ринок мобільних додатків вміщає пропонує користувачам тисячі рішень для полегшення ведення побуту користувача. Так само й додатками, котрі базуються на замірі фізіологічних показників: вони частіше за все заміняють користувачу персонального тренера.

Є пропозиції для тих, хто хоче привести себе в форму і скинути зайву вагу. Перед початком роботи з додатком потрібно поставити мету. Наприклад, скинути 5 кг. І у даних додатків часто існує спільнота накшталт соціальних мереж. Вашу мета побачать всі учасники спільноти і зможуть підтримувати вас і давати корисні поради. Після кожного заняття ви зможете записувати свою активність і дивитися, скільки калорій вам вдалося витратити. Статистика ведеться окремо по кардіо і силових тренувань. Особливу увагу розробники такі додатки приділяють правильному харчуванню. Ви зможете отримувати рецепти від дієтологів, відстежувати свій раціон і калорійність їжі. Також кожен користувач може ввести назву продукту і дізнатися все про його харчової цінності.

Також є додатки помічники для домашніх занять. Відрізняються особливим алгоритмом дій для підбірки вправ. Щоразу воно рандомно вибирає різні вправи з усіх доступних. Це працює наступним чином: ви вибираєте вид фізичної активності: кардіо, пілатес, стретчинг і т.д., а потім вказуєте тривалість тренування. І програма генерує комплекс вправ, які потрібно виконати. Таким чином, тренування не будуть набридати, а ви швидше досягнете результату за рахунок розмаїтості фізичних навантажень. Техніку виконання вправ демонструють на відео реальні спортсмени. Тут також можна відстежувати свою

статистику. А ще ви можете розробити власну програму тренувань і ділитися нею зі спільнотою.

Деякі додатки здатні цілком замінити персонального тренера. Ви зможете вибрати різну навантаження: йога, бокс, силовий тренінг, вправи на окремі групи м'язів. Тренування від таких додатків підійдуть навіть для самих зайнятих - тут доступні коротке заняття тривалістю 15 хвилин без спорядження, так що займатися с цим додатком можна в будь-якому місці і в будь-який час. А можна вибрати і тренування тривалістю 45 хвилин і зі спеціальним спорядженням. Також тренування розділені по інтенсивності: низька, середня і висока і за рівнем підготовки. Під час заняття працює голосовий помічник і таймер. Голосовий помічник контролюватиме процес тренування, і підбадьорювати вас мотивуючими фразами. Якщо захочете, функцію голосового помічника можна відключити.Щодня додаток становить список персональних рекомендацій. Ці рекомендації залежать від видів тренувань, які ви вибираєте найчастіше. Так що, якщо будете займатися тривалий час, додаток створить ідеальну програму тренувань конкретно для ваших цілей і переваг. Все тренування фіксуються, і ви зможете відслідковувати свій прогрес і побачити, скільки калорій вам вдалося витратити.

Також завжди присутні стандартні додатки від виробника пристрою, в випадку Apple Watch – це додаток “Тренування”, котрий включає в себе такі опції як трекінг :

- Бігу в залі
- Ходьби надворі
- Ходьби в залі
- Заняття на орбітреку

- Заняття на велотренажері
- Біг надворі
- Їзда на велосипеді
- Веслування
- Підйом по сходинках
- Інтенсивно-інтервальне тренування
- Піший туризм
- Йога

Також має можливість додати своє особливе тренування. В парі з ним працюють додатки “Здоров’я” та “Активність”, перший оперує даними про вас, другий моніторить вашу активність, а саме пройдені дистанції, спалені калорії, вправи, час стану спокою.

1.4. Висновки до розділу 1

Отже, у цьому розділі було визначено основні засади, поняття та принципи моніторингу різних фізіологічних показників людини. Було детально розглянуто апаратну складову, а саме акселерометр, гіроскоп, висотомір, оптичний датчик вимірювання пульсу, датчик ЕКГ та фотоприймач. Визначено, що датчик вимірювання пульсу забезпечує достатню точність даних для побудови статистики і аналітики життєвих показників під час спокою та тренувань. А у поєднанні з гіроскопом та акселерометром можна визначити зміщення в просторі та інтенсивність вправи, а також падіння користувача. У поєднанні цих датчиків з GPS можна визначити переміщення користувача на деякі дистанції, що також можна використовувати для порівняльної статистики для багатьох фізичних вправ поза спортивною залою. Базуючись на багатьох вище перерахованих показниках ґрунтуються багато комерційних додатків, котрі допомагають користувачу слідкувати за власною вагою, сном, впорядкувати тренування і режим дня. Також розподілити навантаження і покращити спортивні результати в тренажерному залі.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ДОДАТКУ

2.1 Постановка задачі при проектуванні програмного забезпечення

При проектуванні мобільного додатку FitBack функціонал орієнтується на основні потреби користувачів, повязані з фізичними активностями. Вся бізнес-логіка додатку формується довкола розпорядку користувача і його цілей у менеджменті своїх вправ і харчування. Основними задачами, що стоять при розробці даного застосунку є можливість мати доступ до таких даних:

- Каталог вправ
- Каталог харчування
- Розклад
- Статистика

Також зкомпонувати ці всі дані та категорії у зручному для користувача форматі та дизайні. Також для повноти функціоналу потрібно організувати спілкування з умовним тренером, отже додати зручний менеджер повідомлень.

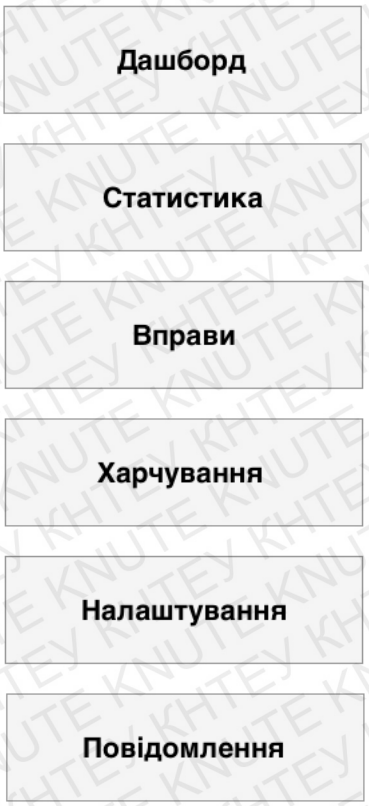


Рис. 2.1. Блок-схема логічних модулів додатку

2.2. Вибір засобів розробки та платформи

Вибір програмно-апаратної платформи

Розроблятись додаток буде для операційної системи iOS для платформи Apple iPhone і Apple Watch з Watch OS.

iPhone - ряд смартфонів, розроблений компанією Apple і випущений 2007 року. iPhone функціонує як камерофон, портативний медіа-плеєр, інтернет-клієнт (з електронною поштою, веб-браузером та Wi-Fi), зокрема з можливостями відправки SMS та візуальної голосової пошти. Головною особливістю смартфону є сенсорний екран з технологією multi-touch, навколо якого побудований інтерфейс користувача з віртуальною клавіатурою замість фізичної. Сторонні програми можна завантажити з App Store, запущеного 2008 року. У США iPhone вперше з'явився у продажу 29 червня 2007 року, в Європі - 9 листопада 2007-го року, в Азії планувалася поява у 2008 році. Торговельна марка «iPhone» належить компанії Cisco Systems, яка у грудні 2006 року випустила апарат з такою назвою для інтернет-телефонії. 20 лютого 2007 р. Apple та Cisco досягли домовленості щодо спільного користування торговельною маркою. 27 липня 2016 року компанія Apple заявила про продаж 1 млрд мобільних телефонів iPhone різних версій. Успіх плеєрів iPod та інших продуктів з приставкою «i», спонукав маркетинговий відділ і керівництво компанії до її використання і в назві телефону — iPhone. Однак це викликало ряд складнощів. Торгова марка «iPhone» зареєстрована 20 березня 1996 року компанією Infogear.

Після оголошення 9 січня 2007 року про випуск Apple мобільного телефону з назвою «iPhone» компанія Cisco подала до суду на Apple за неправомірне використання торгової марки. 21 лютого 2007 року компанії досягли спеціальної угоди про спільне

використання торгової марки iPhone, деталі якого не розголошувалися.

iOS - це власницька мобільна операційна система від Apple. Розроблена спочатку для iPhone, згодом також вдосконалена для використання на iPad (до літа 2019, коли на конференції Apple WWDC було представлено нову OS для iPad — iPadOS), iPod Touch та Apple TV (до 9 вересня 2015, коли на спеціальному заході Apple було представлено tvOS). Apple не дозволяє роботу ОС на мобільних телефонах інших фірм. (Відома як iPhone OS до червня 2010 року). iOS є похідною від OS X, отже, є за своєю природою Unix-подібною операційною системою. Користувацький інтерфейс iOS заснований на концепції прямої маніпуляції з використанням жестів Multi-Touch. Елементи інтерфейсу управління складаються з повзунків, перемикачів і кнопок. Він призначений для безпосереднього контакту користувача з екраном пристрою. Внутрішній акселерометр використовуються деякими програмами для реагування на струшування пристрою, яке є також загальною командою скасування, або обертання пристрою у трьох вимірах, що є загальною командою перемикання між книжковим та альбомним режимами. Станом на 2019 рік інтернет-магазин App Store містить понад 2 мільйони застосунків для iOS, які були завантажені понад 15 мільярдів разів. Станом на травень 2010 року, iOS становив 15,4 % ринку операційних систем для смартфонів, третій після Symbian і Blackberry.

Apple Watch - наручний годинник із додатковими функціями (розумний годинник) створений корпорацією Apple. Він поєднує у собі фітнес-трекер та різноманітні фукнції для стеження за здоров'ям, що інтегруються з iOS та іншими продуктами і сервісами Apple. Для його повноцінної роботи потрібен смартфон сімейства iPhone 5 або більш пізньої модифікації, зокрема для здійснення дзвінків і

надсилання текстових повідомлень. Наявніть wi-fi модулю дозволяє розумному годиннику виконувати певні обмежені функції без прив'язки до iPhone. Apple Watch з'явився у продажу 24 квітня 2015 року[1]і швидко став світовим лідеров з продажу носимих пристроїв. Друге покоління годинників було представлене у вересні 2016. Третє покоління Apple Watch Series 3 було представлене 22 вересня 2017 р. Останнє, четверте покоління годинників, було презентоване 12 вересня 2018 року. Основними змінами у 4-му поколінні стали збільшення розміру циферблату годинника, а також активного дисплею за рахунок зменшення його країв до мінімуму. Також у значним нововведенням стала здатність нового Apple Watch робити електрокардіограму користувача.

Apple Watch працюють на операційній системі Watch OS. Годинники мають поворотне коліщатко для прокрутки або збільшення. Натискання на нього дозволяє повернутися до домашнього екрану. Дисплей має розмір 40x40 або 44×44 мм, і здатний розрізняти натискання і дотик. Апарат не має роз'ємів, підзарядка батареї відбувається за допомогою індуктивного адаптера з магнітною фіксацією. На нижній стороні годинників можуть бути розташовані світлодіоди і фотодіоди для вимірювання пульсу. Також є в наявності інтерфейс NFC, який дозволить робити безконтактну оплату через систему Apple Pay.

WatchOS - операційна система, розроблена компанією Apple для роботи на годиннику Apple Watch. Вперше випущена 24 квітня 2015 року. watchOS підтримує програми, розроблені за допомогою WatchKit. Головне меню системи виконано у вигляді Carousel, що надає зручний доступ до будь-якого додатка.

Swift - багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового коду. Swift була представлена на конференції розробників WWDC 2014. Мова побудована з LLVM компілятором, включеного у Xcode 6 beta. Безкоштовний посібник мови програмування Swift доступний для завантаження у магазині iBooks.

Компілятор Swift побудований з використанням технологій вільного проекту LLVM. Swift успадковує найкращі елементи мов C і Objective-C, тому синтаксис звичний для знайомих з ними розробників, але водночас відрізняється використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду.

При цьому Swift-програми компілюються у машинний код, що дозволяє забезпечити високу швидкодію. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C. Замість збирача сміття Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація.

Мова також пропонує безліч сучасних методів програмування, таких як замикання, узагальнене програмування, лямбда-вирази, кортежі і словникові типи, швидкі операції над колекціями, елементи функційного програмування. Основним застосуванням Swift є розробка користувацьких застосунків для macOS, iOS, tvOS, watchOS з використанням тулкіта Cocoa і Cocoa Touch. При цьому Swift надає об'єктну модель, сумісну з Objective-C. Сирцевий код мовою Swift може змішуватися з кодом на C і Objective-C в одному проекті.

Swift щільно інтегровано до власницького середовища розробки Xcode, проте може бути викликано з терміналу, що уможливлює її

використання на операційний системах, відмінних від macOS, наприклад, на Linux.

Окремо варто відзначити, що Swift від компанії Apple не варто плутати з досить давно розроблюваною скриптовою мовою Swift, націленої на багатонитеве програмування і поставленого під вільною ліцензією Apache.

Старший віце-президент по розробці програмного забезпечення Apple Крейг Федеріго під час анонсу цього продукту заявив, що мова програмування Swift був закладений ще в платформі NeXT (ОС NeXTSTEP випускалася в 1989-1995 роках), яка стала основою для сучасної macOS, а потім і iOS .

Розробка поточного варіанту мови Swift почалася в 2010 році Крісом Латтнером, керівником відділу розробки інструментів для створення програмного забезпечення Apple і одним з основних розробників LLVM. Swift запозичив ідеї з «Objective-C, Rust, Haskell, Ruby, Python, C #, CLU, і ще з стількох багатьох мов, що важко перелічити». Спочатку для нової мови використовували назву Shiny.

- 2 червня 2014 року на конференції WWDC Swift був офіційно представлений разом з безкоштовним керівництвом по використанню мови об'ємом в 500 сторінок, доступним на сервісі «iBook Store».
- 8 червня 2015 року компанія Apple оголосила про випуск нової версії Swift 2.0, яка отримала вищу продуктивність, нове API обробки помилок, поліпшення синтаксису мови, а також функцію перевірки доступності функцій Swift для цільових ОС.

- грудня 2015 року було випущено бета версія Swift 3.0 з підтримкою операційних систем OS X, iOS і Linux і ліцензована під відкритою ліцензією Apache 2.0 license with a Runtime Library Exception. Версія 3.0 назад не сумісна з більш ранніми версіями мови; починаючи з нативної середовища розробки XCode версії 9 більш версії мови Swift-2 і раніше не підтримуються. На початку квітня 2016 неназване джерело ЗМІ в корпорації Google повідомив, що компанія розглядає можливість переведення мови Swift в мову «першого класу» для платформи Android. Раніше вже висувалися прототипи Swift компілятора для Android. 19 вересня 2017 року було випущено версію Swift 4.0. У вересні 2018 року, разом з новою версією iOS 12, була випущена нова стабільна версія мови Swift 4.2, і з'явилася бета-версія Swift 5.0. У версії 5.0 заявлена, нарешті, стабільна робота ABI зі стандартними бібліотеками (Swift Dynamic Library), підтримка регулярних виразів і першокласне рішення для паралельної обробки даних з асинхронним режимом обробки async / await.
- 20 вересня 2019 Swift 5.1 - це наступний великий реліз Swift, і його завдання нарешті забезпечити стабільність ABI.

Xcode - інтегроване середовище розробки (IDE) виробництва Apple. Дозволяє створювати програмне забезпечення з використанням таких технологій як GCC, GDB, Java та ін. На сьогодні є єдиним

засобом написання «універсальних»(Universal Binary) прикладних програм для Mac OS X. Xcode включає в себе більшу частину документації розробника від Apple та Interface Builder — застосунок, який використовується для створення графічних інтерфейсів. Пакет Xcode містить змінену версію вільного набору компіляторів GNU Compiler Collection і підтримує мови C, C++, Objective-C, Swift, Java, AppleScript, Python і Ruby з різними моделями програмування, включаючи (але не обмежуючись) Cocoa, Carbon і Java. Сторонніми розробниками реалізована підтримка GNU Pascal, Free Pascal, Ada, C#, Perl, Haskell і D. Пакет Xcode використовує GDB як back-end для свого налагоджувача.

2.3. Проектування архітектури додатку

З самого початку розглядалась протоколо-орієнтовна архітектура VIPER, котра дуже добре вписується в формат мікромодульної архітектури, де кожен екран є окремим модулем з структурними компонентами: View-Interactor-Presenter-Entity-Router. Це поділ також відповідає принципу Single Responsibility. Interactor відповідальний за бізнес-логіка, Presenter відповідальний за відображення, і View відповідальний за візуальне представлення.

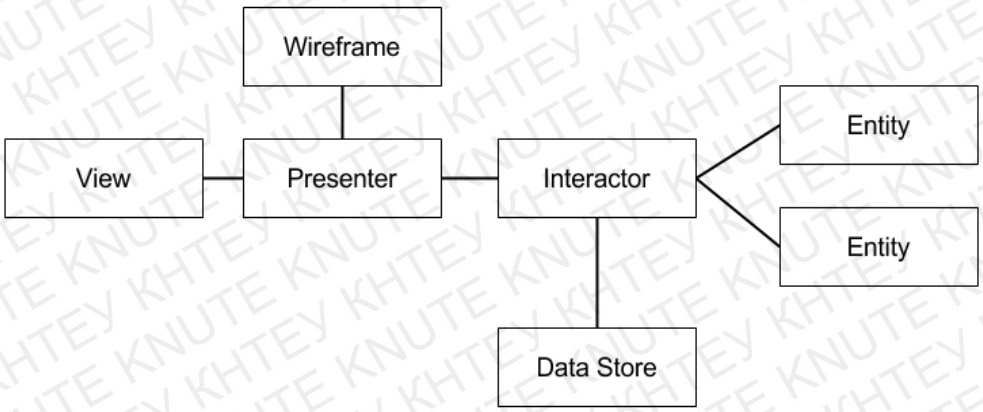


Рис. 2.2. Блок-схема архітектури VIPER

Interactor є простим юз кейсом в додатку. Він містить бізнес-логіку для управління об'єктами (Entity), щоб виконати певне

завдання. Завдання виконується в Interactor'е, незалежно від будь-якого UI. Той же Interactor можна використовувати в iOS додатках або консольних додатках для Mac OS.

Оскільки Interactor є PONSO (Звичайний NSObject), який перш за все містить логіку, і його легко розробити за допомогою TDD (Розробка через тестування).

Entity - це об'єкти, якими управляє Interactor. Entity тільки управляє Interactor. Він ніколи не передає суті рівнем подання (тобто Presenter'у).

Data Store (наприклад, веб-сервіс, база даних) відповідає за надання Entity в Interactor. Оскільки Interactor застосовує свою бізнес-логіку, він буде здійснювати вибірку Entity зі сховища даних, управляти Entity і потім повертати оновлені Entity назад в сховище даних. Сховище даних управляє персистентного Entity. Entity не знають про сховище даних, таким чином, вони не знають, як зберігатися.

При використанні TDD (Розробка через тестування) для розробки Interactor'a, можливо відключити виробниче сховище даних за допомогою double / mock тестів. Чи не звертаючись до віддаленого сервера (для веб-сервісу) або диска (для бази даних) дозволяє вашим тестів бути повторюваними і швидкими.

Presenter - це PONSO, який в основному складається з логіки, щоб керувати UI. Він збирає вхідні дані від взаємодії з користувачем, таким чином, він може відправляти запити Interactor'у. Presenter також отримує результати Interactor'a і перетворювати результати в стан, який є найбільш ефективним для відображення на View.

Entity ніколи не передаються з Interactor'a до Presenter'у. Замість цього прості структури даних, у яких немає поведінки, передаються з Interactor'a до Presenter'у. Це перешкоджає будь

'реальній роботі' в Presenter'е. Presenter може тільки підготувати дані для відображення на View.

View є пасивною. Воно чекає Presenter'а, щоб передати змісту для виведення на екран; вона ніколи не запитує дані у Presenter'а. Методи, визначені для подання (наприклад, LoginView для екрану входу в систему), повинні дозволити Presenter'у спілкуватися на більш високому рівні абстракції, вираженої з точки зору його вмісту, а не те, як це вміст буде відображатися. Presenter не знає про існування UILabel, UIButton, і т.д. Presenter тільки знає про зміст, яке він підтримує і коли його потрібно вивести на екран. Presenter потрібно визначати, як зміст виводитися на екран.

Маршрутизація обробляє навігацію від одного екрана до іншого, як визначено в wireframes, створених проектувальником взаємодії. Wireframe об'єкт несе відповідальність за маршрутизацію. Wireframe об'єкт володіє об'єктами UIWindow, UINavigationController, і т.д. Він відповідальний за створення Interactor, Presenter і View / ViewController і за налаштування ViewController. Так як Presenter містить логіку, щоб реагувати на введення даних користувачем, Presenter знає, коли перейти на інший екран. Wireframe знає, як це зробити. Отже, Presenter - це користувач Wireframa.

Але чиста архітектура VIPER має деякі мінуси, такі як:

- Проблематичне тестування
- Велика кількість файлів
- Різнонаправленість
- Відсутній менеджмент сторонніх ефектів

Тому було прийнято рішення застосувати Data Driven підхід, а його особливістю є те, що наш View має якийсь перелік визначених

станів, котрими оперує додаток залежно від того, які дані надходять чи що користувач на даний момент часу намагається зробити.

Основні переваги такого підходу:

- Однонаправленість
- Легкість тестування
- Відсутність невалідних станів
- Відсутність багатьох зайвих залежностей
- Кращий менеджмент даних
- Очікуваний UX

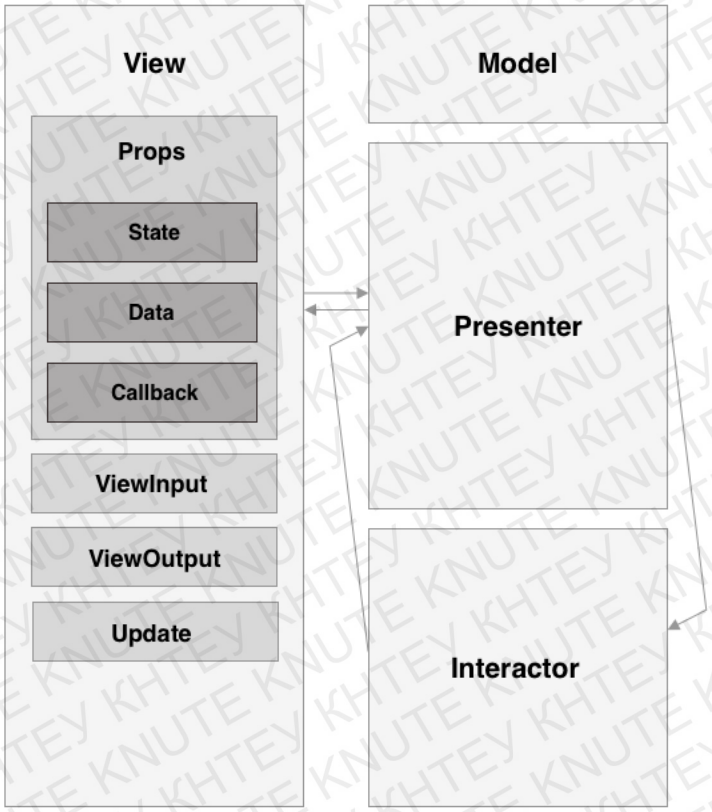


Рис. 2.3. Блок-схема Data-Driven підходу до VIPER

2.4. Методологія розробки

Канбан — це метод управління розробкою програмного забезпечення з наголосом з доставкою якраз вчасно, та униканні перевантаження членів команди. При цьому підході процес від опису задачі до доставки результатів її виконання користувачу, наочно показується учасникам процесу, і члени команди можуть витягувати роботу з черги. Канбан в контексті розробки програмного забезпечення означає систему візуального управління процесом, яка вказує, що виробляти, коли виробляти і як багато виробляти і бере за основу Toyota Production System та Ощадливе виробництво.
Слово «канбан» походить з японської і грубо перекладається як «вивіска» чи «білборд». Девідом Андерсеном він був описаний як підхід до інкрементної, еволюційної зміни процесів і систем в організації. Він використовує обмеження роботи, що знаходиться в процесі виконання, як ключовий механізм для виявлення проблем в роботі системи та стимулює співпрацю для постійного її вдосконалення. Корені знаходяться в чотирьох базових принципах:

- Почніть з того, що ви маєте зараз. Метод Канбан не описує конкретний набір ролей чи кроків процесу. Він стартує з ролями і процесами, що є у вас зараз, і стимулює постійні інкрементні та еволюційні зміни в системі. Канбан — це метод управління змінами.
- Погодьтесь домагатись інкрементних, еволюційних змін.Організація (чи команда) повинна погодитись, що постійні, інкрементні зміни — це спосіб покращити систему, і зробити так, щоб ці покращення прижились. Глобальні зміни можуть виглядати більш ефективними, але мають більший ризик провалу, через опір та страх змін в організації. Канбан заохочує постійні невеликі зміни до поточної системи.
- Поважайте поточний процес, ролі, відповідальності та посади. Дуже ймовірно, що організація має деякі елементи, що працюють задовільно, і їх варто зберегти. Канбан намагається уникнути страхів, домовляючись поважати поточні ролі, відповідальності та посади з метою отримати ширшу підтримку.
- Лідерство на всіх рівнях. Лідерські дії на всіх рівнях - від окремих працівників і аж до старшого менеджменту — схвалюються.

Андерсон ідентифікував п'ять ключових властивостей, які спостерігаються в кожній успішній реалізації методу Канбан. Пізніше їх було названо практиками і розширено шостою.

1. Візуалізуйте. Візуалізація процесів роботи допомагає в правильному розумінні змін, що плануються і допомагає впроваджувати їх згідно з планом.Типовим способом візуалізувати процес роботи є використання дошки з колонками і картками. Колонки на дошці позначають різні кроки процесу роботи.
2. Обмежуйте задачі в процесі виконання. Обмеження задач в процесі виконання має на увазі те, що використовується система «витягування» на частинах, або всьому процесі роботи. Система «витягування» працює як один з головних стимулів до постійних покращень в системі. Система «витягування» може бути реалізована, як система канбан, CONWIP, DBR чи якийсь інший варіант. Критичним елементом є те, що робота, котра перебуває в стані виконання на кожному кроці робочого процесу, є обмеженою, і що нова робота «витягується» в кожен крок, коли з'являється місце в колонці кроку.
3. Керуйте потоком. Кожен перехід між станами в потоці моніториться, вимірюється і звітується. Активне управління потоком дозволяє оцінити позитивні та негативні ефекти змін у системі.
4. Зробіть політики явними. Поки механізм чи процес не стане явним, часто важко чи неможливо здійснювати обговорення щодо його вдосконалення. Без явного розуміння, як все працює, будь-які обговорення проблем стають емоційними та суб'єктивними. З явним розумінням можливо перейти до більш раціональних, емпіричних та об'єктивних обговорень проблем.
5. Створіть цикли зворотнього зв'язку. Організації що не створили другий рівень зворотнього зв'язку — перегляд операцій, — зазвичай не бачать вдосконалення процесу поза локалізованим рівнем команди.
6. Вдосконалюйте співпрацюючи, розвивайтесь експериментально (використовуючи моделі та науковий метод). Метод Канбан пропагує малі поступові, постійні та еволюційні зміни які приживаються. Коли команди мають спільне розуміння теорій про роботу, процес, ризики, вони більш ймовірно будуть здатними виробити спільне розуміння проблем та запропонувати

вдосконалення які будуть результатом консенсусу.Метод Канбан радить використовувати науковий підхід до втілення змін.

7.

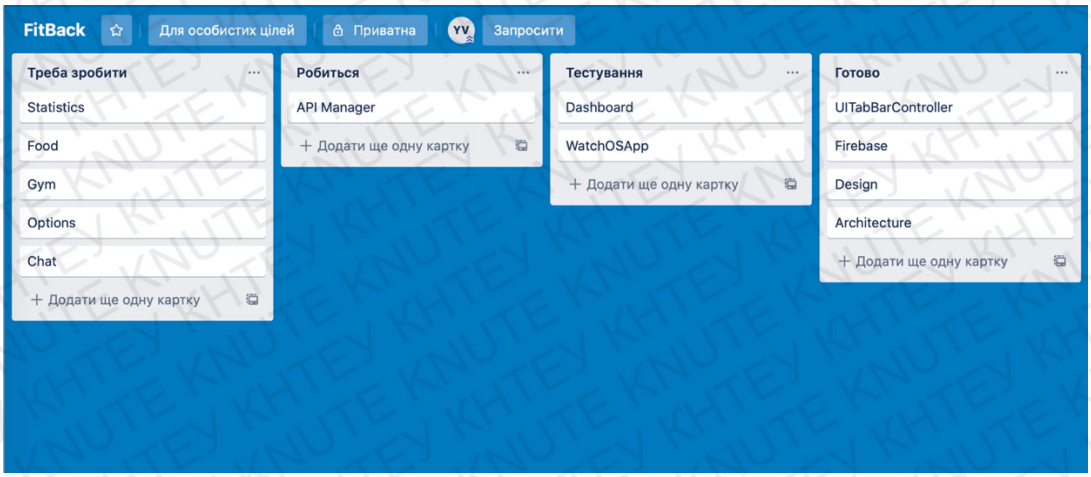


Рис. 2.3. Канбан-дошка з умовним процесом розробки FitBack

2.5. Висновки до розділу 2

Отже, у цьому розділі було визначена і описана платформа для розробки. В даному випадку це Apple iPhone і Apple Watch. Вибір був зумовлений в першу чергу наявністю дуже хорошої екосистеми, котру створила компанія Apple. Також були визначені інструменти розробки, а саме надсучасна мова програмування Swift і IDE Xcode. Саме ця мова була вибрана, оскільки призначена в першу чергу для апаратно-програмних платформ компанії Apple і є сучаснішим і швидшим рішенням за Objective-C, її попередника. А також знаходиться на вищому рівні абстракції на C чи C++. За методологію розробки був взятий “канбан”, а для візуалізації використана канбан-дошка.

РОЗДІЛ 3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1. Прототипування та дизайн додатку

Перед тим як розпочати безпосередньо розробку потрібно створити макет, щоб оцінити розміщення і компонування даних в додатку, що в свою чергу буде відображенням бізнес-логіки і напряму впливатиме на користувацький досвід користувача. Основна складність у роботі зі додатками в тому, що він зазвичай народжується на підставі однієї ідеї, яку на початку розуміє тільки засновник. І навіть якщо він сам добре розуміє свою ідею, не факт, що він може правильно передати та пояснити її іншим учасникам розробки продукту. Сенс ось у чому: коли ви лише починаєте готувати до запуску свій додаток, ви концентруєтеся на чомусь одному, дуже важливому для вас. Ви робите першу версію того як це бачите ви, показуєте продукт своїм друзям і потенційним клієнтам і отримуєте цілу купу зауважень, як це хотіли б бачити вони на місці користувачів. У будь-якому випадку, ви повинні рухатися за класичним UX-процесом. Так що ж таке UX прототип і як почати проектування? Базово процес складається з наступних кроків:

Рис. 3.1. Рух від абстракцій до конкретики

Пройшовши всі ці кроки, ви отримаєте описані та зафіксовані цілі, завдання, персони, цінності, use cases проекту, які стануть вам у нагоді і для отримання інвестицій, і при запуску продукту в світ. Для даного додатку проектування UX робилося в сервісі NinjaMock. Розглянемо декілька екранів:

Рис. 3.2. Прототип екрану дашборду з актуальною поточною інформацією

Прототип показує розміщення основних елементів на даному екрані, а саме UITabBarController знизу, що заміняє головне меню в iOS додатках, з котрого ми можемо перейти в будь-яку іншу частину додатку і визначити поточне положення в ієрархії екранів.

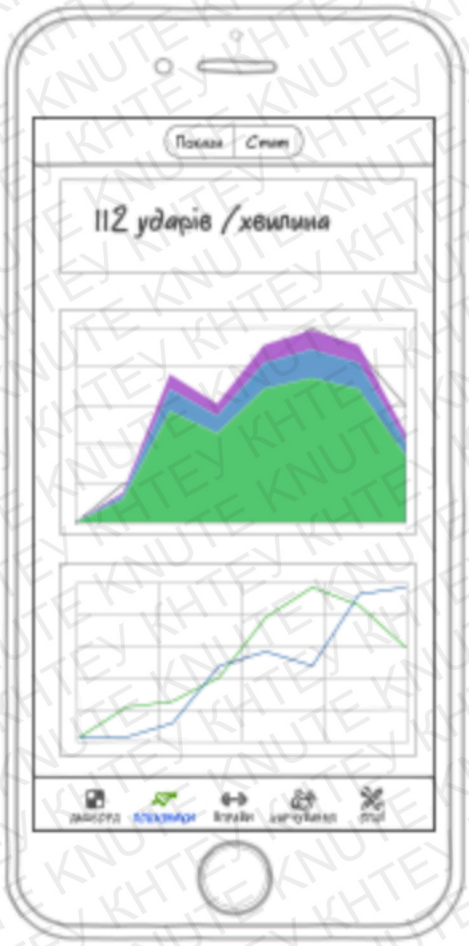


Рис. 3.3. Прототип екрану показників

Даний екран має в собі поточний показник серцебиття, а також всю статистику пов’язану з ним за деякий період. Більше екранів розміщено в додатку И.

Тому UX-прототипування додатку має декілька важливих особливостей, які відрізняють його від корпоративних проектів:

- досить дешево: проектування (прототип, створення специфікації) має коштувати не більше 15% проекту;
- рух від загального до конкретного: ви повинні чітко розуміти кроки, які вам потрібно пройти при проектуванні. Це означає

рухатися від загального – цінності, які несе ваш продукт, персони споживача, до окремого – екранів, інтерфейсів і т.п .;

- раннє тестування: ви повинні якомога раніше тестувати на справжніх клієнтах, а не на друзях і членах команди;
- збір даних: ви повинні зібрати якомога більше даних і на їх підставі покращувати UX;
- міцний UI-kit, який дозволить реалізувати весь функціонал і кардинально міняти його в майбутньому, не роблячи редизайн елементів.

Після прототипування і готового технічного завдання з специфікаціями, час приступати до створення дизайну програмного продукту. Для цього знадобиться Sketch, Figma, Adobe XD або інші графічні редактори. В даному випадку використовувався Sketch.

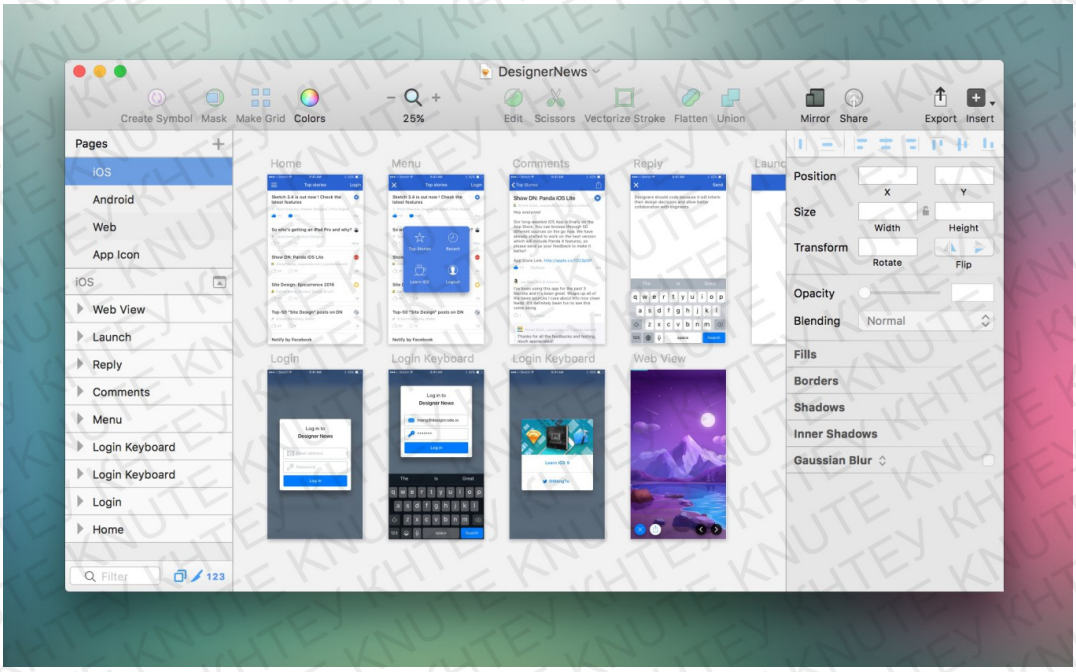


Рис. 3.4. Зовнішній вигляд редактору Sketch

Sketch - це платний векторний графічний редактор інтерфейсів для Apple's macOS, розроблений Нідерландською компанією Bohemian Coding. Отримав нагороду Apple Design Award 2012 року. Sketch був вперше випущений 7 вересня 2010.

Графічний редактор дозволяє побачати нам фінальний варіант нашого користувацького інтерфейсу згідно UX та ТЗ. Роглянемо прототиповані екрани після нанесення UI:

Рис. 3.5. Екран дашборду

На Рис. 3.4 видно, що хоч структура залишилася спільною з прототипом, але всі елементи користувацького інтерфейсу

заповнились даними та набули сенсу, отже основна функція даного екрану керувати поточним розпорядком дня користувача, де чередуються сутності трьох видів:

- Фізичне навантаження
- Прийом їжі
- Відпочинок

Ідея екрану полягає в його контекстності, картки з відповідними сутностями привязані до деякого часу доби, всі картки є клікабельним на даному екрані. При натисканні на тренування ми запустимо його відстежування. Також зверху є кнопка переходу до переписки з тренером, котрий в свою чергу і має формувати нам дієту і набір навантажень.

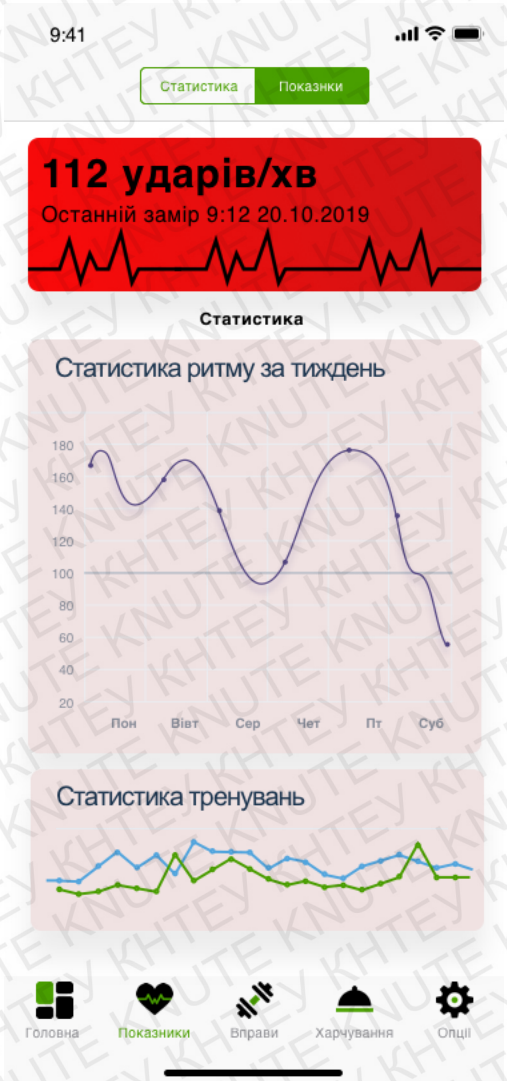


Рис. 3.6. Екран фізіологічних показників

На даному екрані бачимо поточний сердечний ритм користувача та деяка статистика по минулим замірам серцебиття та тренуванням, а саме їх інтенсивність і періодичність. Також можемо перемикатись між сутностями, такими як збірна статистика і показники: в такому випадку наповнення екрану змінює свій контекст.

Рис. 3.7. Екран вправ

На даному екрані бачимо деякий банк даних про вправи користувача, розбитий по категоріям, в кожній відповідно існують сутності, котрі собою являють праву, з інструкцією по виконанню. Також з даного екрану можна перейти в календарний план тренувань. Важливо звернути увагу, що усі вправи позначені відповідним кольором по всьому додатку, щоб не плутати користувача і викликати єдину асоціацію.

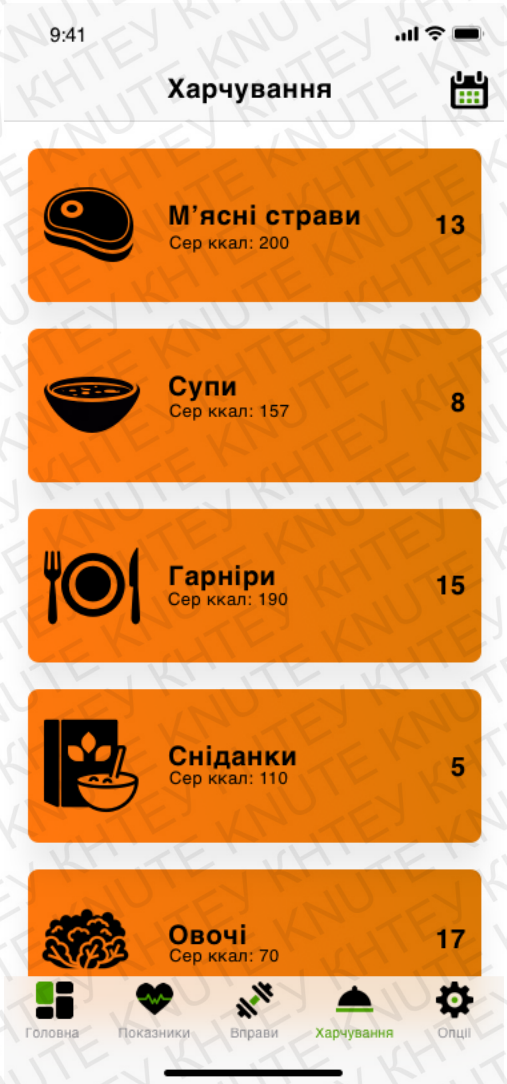


Рис. 3.8. Екран харчування

На даному екрані бачимо деякий банк даних про харчування користувача, розбитий по категоріям. Кожна категорія містить кількість страв в ній та середню їх калорійність. Також з даного екрану можна перейти в календарний план харчування. Важливо звернути увагу, що усі страви позначені відповідним кольором по всьому додатку, щоб не плутати користувача і викликати єдину асоціацію.

3.2. Побудова архітектури та алгоритму роботи

Основна задача перед початком написання програмного коду – це продумати структуру, архітектуру, взаємодію всіх пакетних компонентів та ієрархію всіх екранів і функціоналу додатку.

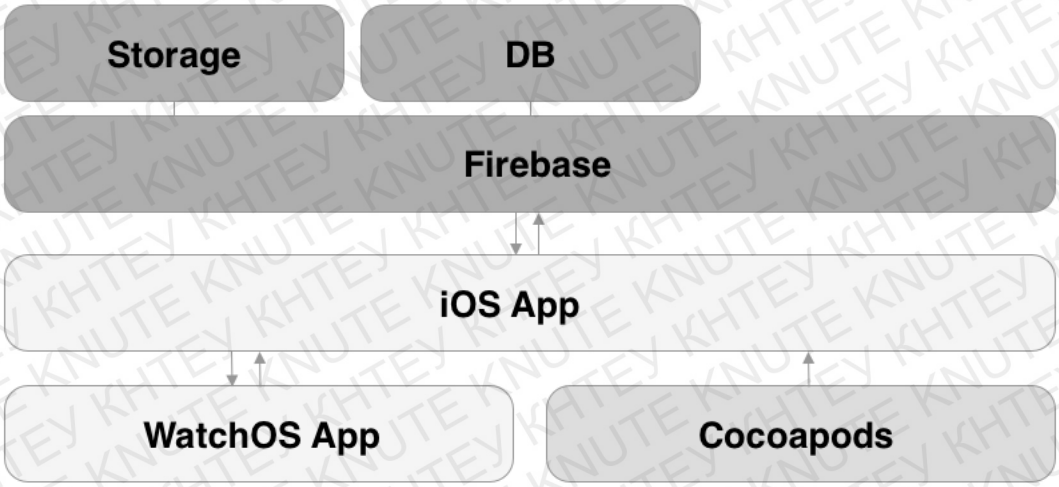


Рис. 3.9. Структура всього функціоналу додатку

Додаток використовує так звану клієнт-серверну взаємодію. Де в ролі серверу виступає Google Firebase. В свою чергу безпосередньо iOS додаток використовує функціонал WatchOS додатку та відповідно функціонал зовнішніх бібліотек, з'єднаних з додатком менеджером залежностей Cocoapods.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;

- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів — клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Структура проекту, зокрема модулю має такий вигляд:

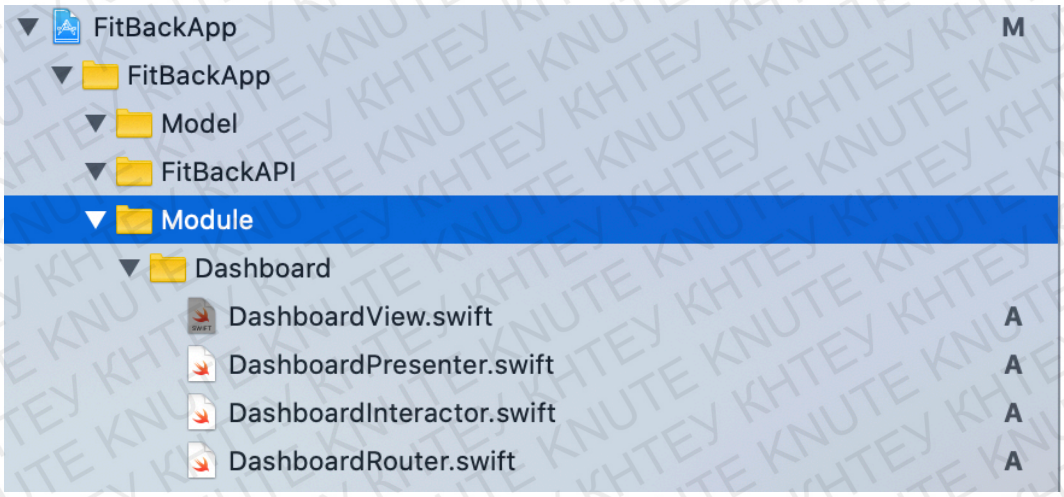


Рис. 3.10. Структура всього функціоналу додатку

Кожен екран є модулем, а значить окремим структурним елементом, а кожен структурний елемент зберігається, відповідно в окремій теці.

Початкова структура вікон має такий вигляд в Interface Builder:

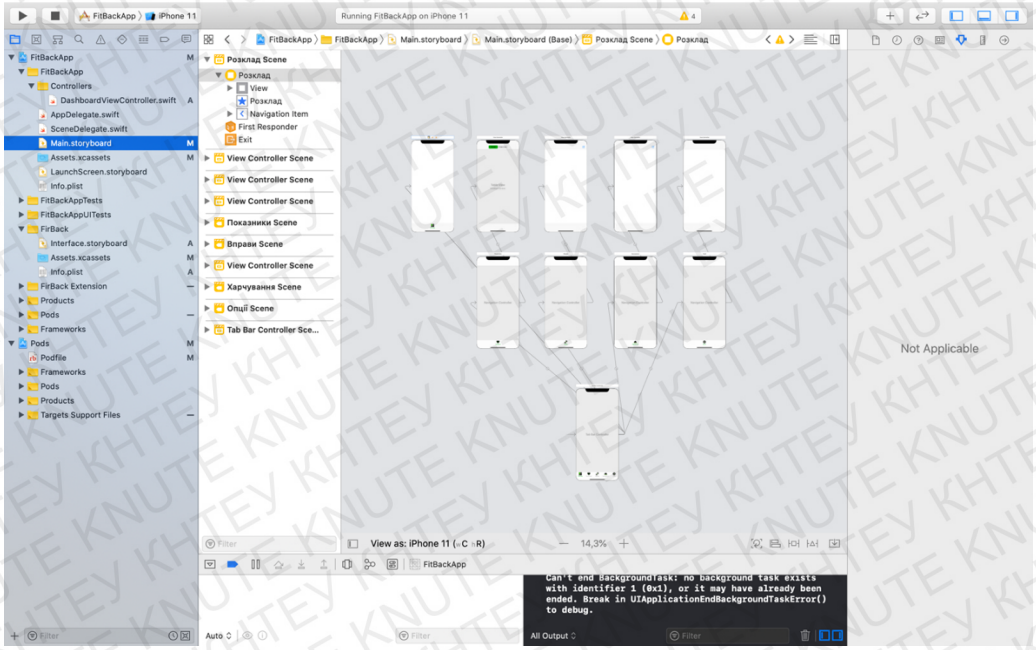


Рис. 3.11. Структура екранів додатку

Сутності Interface Builder являють собою верстку у вигляді XML-коду, приклад:

```
<scene sceneID="fgb-Ba-dym">
    <objects>
        <viewController id="Iur-7g-wm5" customClass="DashboardViewController" customModule="FitBackApp"
            customModuleProvider="target" sceneMemberID="viewController">
            <view key="view" contentMode="scaleToFill" id="SXK-dp-KVD">
                <rect key="frame" x="0.0" y="0.0" width="414" height="896"/>
                <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                <subviews>
                    <imageView clipsSubviews="YES" userInteractionEnabled="NO" contentMode="scaleAspectFit"
                        horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES"
                        image="chat-speech-bubbles" translatesAutoresizingMaskIntoConstraints="NO" id="uo8-rq-6fd">
                        <rect key="frame" x="328" y="44" width="66" height="35"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                    </imageView>
                    <tableView clipsSubviews="YES" contentMode="scaleToFill" fixedFrame="YES" alwaysBounceVertical="YES"
                        dataMode="prototypes" style="plain" separatorStyle="default" rowHeight="-1"
                        estimatedRowHeight="-1" sectionHeaderHeight="28" sectionFooterHeight="28"
                        translatesAutoresizingMaskIntoConstraints="NO" id="Y1c-fn-96F">
                        <rect key="frame" x="0.0" y="87" width="414" height="726"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <color key="backgroundColor" systemColor="systemBackgroundColor"
                            cocoaTouchSystemColor="whiteColor"/>
                        <connections>
                            <outlet property="dataSource" destination="Iur-7g-wm5" id="ZNQ-I2-0hV"/>
                            <outlet property="delegate" destination="Iur-7g-wm5" id="7bi-1O-anf"/>
                        </connections>
                    </tableView>
                </subviews>
                <color key="backgroundColor" systemColor="systemBackgroundColor" cocoaTouchSystemColor="whiteColor"/>
                <viewLayoutGuide key="safeArea" id="MQn-zh-AwX"/>
            </view>
            <tabBarItem key="tabBarItem" title="Розклад" image="dashboard" id="gHJ-2Q-yl8"/>
            <navigationItem key="navigationItem" id="XDp-7e-0c0">
                <barButtonItem key="rightBarButtonItem" title="Item" id="isM-eK-H4d"/>
            </navigationItem>
            <connections>
                <outlet property="chatImageButtonView" destination="uo8-rq-6fd" id="boZ-Ke-r7Y"/>
                <outlet property="dashboardTableView" destination="Y1c-fn-96F" id="xLK-WX-uv3"/>
            </connections>
        </viewController>
```

Рис. 3.12. UIViewController в XML

Побудова ієрархії екранів дозволяє виробити алгоритм роботи, котрий вже лягає на технічне завдання, дизайн UX, UI та архітектурне рішення. Наприклад, алгоритм запуску тренування виглядає наступним чином:

Рис. 3.13. Алгоритм запуску тренування

1. З поточного розкладу користувач вибирає тренування
2. Переходить на екран з вправами конкретного тренування
3. Коли натискає на “прапорець” то починає тренування
4. Секундомір починає відлік, а датчик серцебиття зчитує поточний ритм

Також на етапі 2 можна запустити конкретну вправу з Apple Watch:

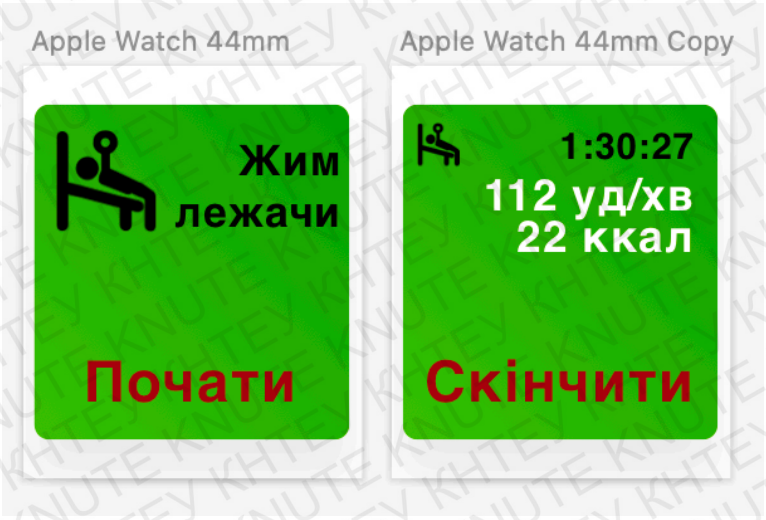


Рис. 3.14. Алгоритм запуску вправи на Apple Watch

3.3. Розробка серверної частини і БД

Firebase надає в режимі реального часу базу даних та бекенд як службу. Ця служба надає розробникам застосунків API, який дозволяє синхронізувати дані застосунків між клієнтами та зберігати їх у хмарі Firebase. Компанія також надає клієнтські бібліотеки, які дозволяють інтеграцію із застосунками Android, iOS, JavaScript / Node.js, Java, Objective-C, Swift. База даних також доступна через REST API та прив'язки до декількох сценаріїв JavaScript, таких як AngularJS, React, Ember.js та Backbone.js. REST API використовує протокол подій із сервером, який є інтерфейсом для створення HTTP-з'єднань для отримання push-повідомлень від сервера. Розробники, які використовують Realtime Database, можуть захищати свої дані за допомогою правил безпеки, що застосовуються на сервері. Cloud Fiestore, яка є наступною генерацією Firebase Realtime Database, була випущена у бета-версії. Firebase Storage забезпечує надійне завантаження та вивантаження файлів для застосунків Firebase незалежно від якості мережі. Розробник може використовувати його для зберігання зображень, аудіо-, відео- чи іншого вмісту, створеного

користувачами. Зберігання Firebase підтримується Google Cloud Storage.

Для того щоб почати працювати з Firebase потрібно:

1. Зареєструвати додаток
2. Вибрати розташування серверу
3. Прописати правила використання
4. Підключити автентифікацію
5. Підключити бібліотеку через менеджер залежностей
6. Внести дозволи в сам додаток
7. Створити БД і Сховище

```
{
  "rules": {
    "chats": {
      "$chatID": {
        "messages": {
          ".read": "data.parent().child('members').child(auth.uid).exists()",
          ".write": "data.parent().child('members').child(auth.uid).val() == 'owner' ||
                     data.parent().child('members').child(auth.uid).val() == 'chatter'"
        },
        "members": {
          ".read": "data.child(auth.uid).val() == 'owner'",
          ".write": "data.child(auth.uid).val() == 'owner' ||
                    (!data.exists() && newData.child(auth.uid).val() == 'owner')"
        },
        "pending": {
          ".read": "data.parent().child('members').child(auth.uid).val() === 'owner'",
          ".write": "data.parent().child('members').child(auth.uid).val() === 'owner'",
          "$uid": {
            ".write": "$uid === auth.uid && !data.exists() &&
                       !(data.parent().parent().child('members').child($uid).exists())"
          }
        }
      }
    }
  }
}
```

Рис. 3.15. Набір правил доступу та використання Firebase

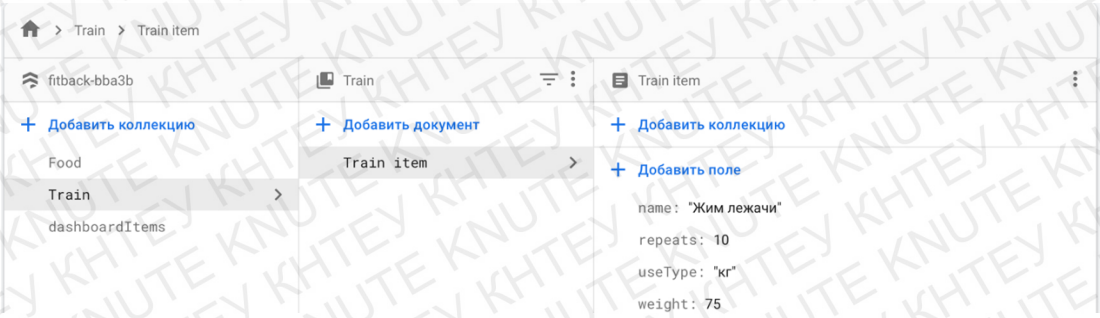


Рис. 3.16. Візуалізація бази даних на сервері

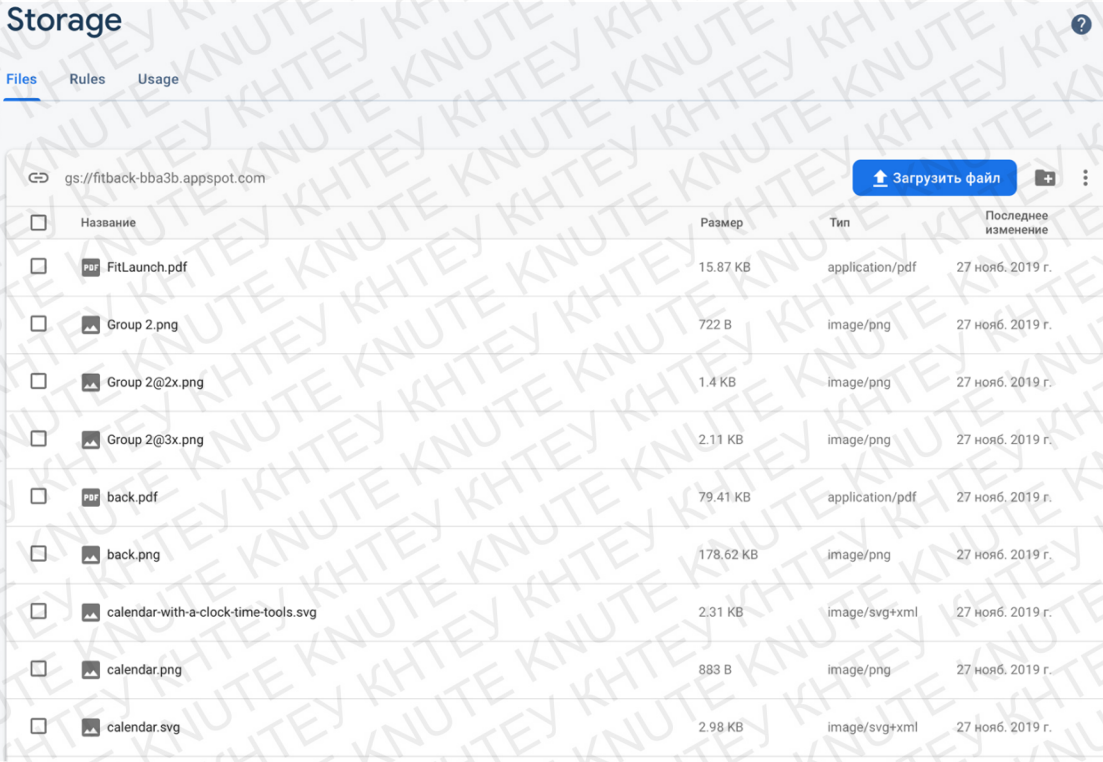


Рис. 3.17. Набір зображень, котрі використовує додаток у Сховищі Firebase

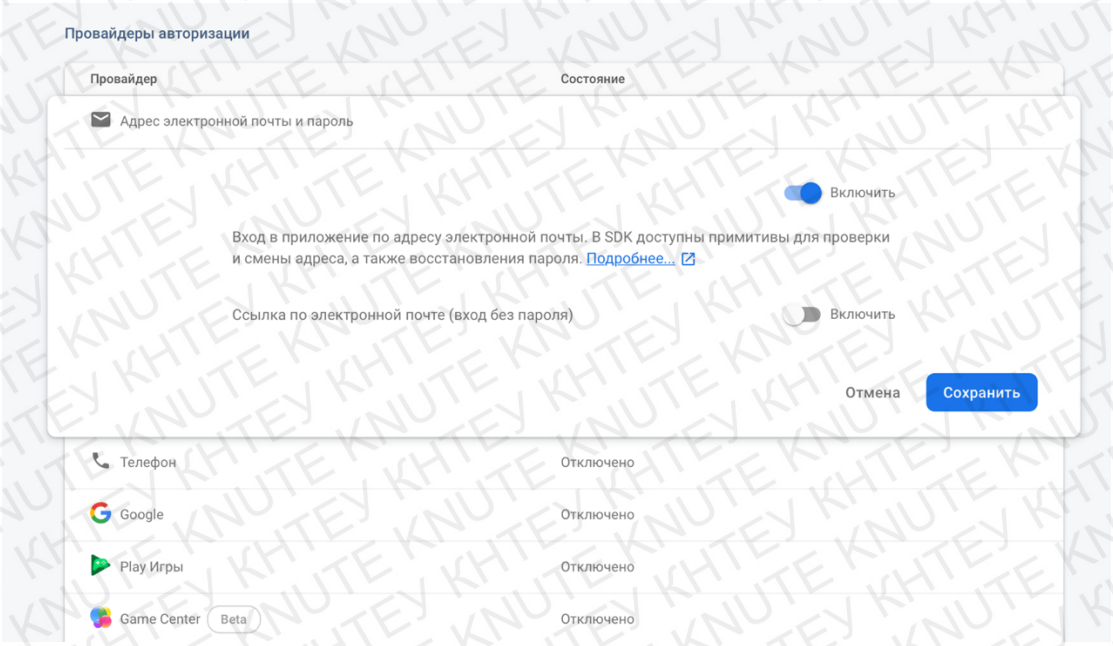


Рис. 3.18. Автентифікація користувача зі сторони серверу

3.4. Розробка iOS додатку

Створення та конфігурація проекту відбуваються в автоматизованому режимі завдяки надсучасному середовищу розробки Xcode. Розробник має лише створити ідентифікатор, назву, задати параметри тестування та вибору режиму верстки, в даному випадку буде використаний Storyboard. Також тут можна вибрати поміж нативних мов програмування Objective-C та Swift.

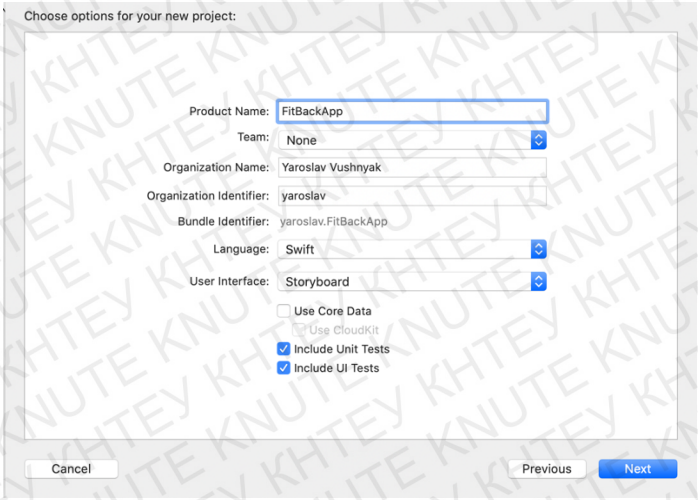


Рис. 3.19. Початкову налаштування проекту

Key	Type	Value
▼ Information Property List	Dictionary	(15 items)
Localization native development re...	String	$(DEVELOPMENT_LANGUAGE)
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
▶ Application Scene Manifest	Dictionary	(2 items)
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)
▶ Supported interface orientations (i...	Array	(4 items)

Рис. 3.20. Дозвільно-опційний файл plist з налаштуваннями додатку

Для запитів по API буде використовуватись обгортка для обробки нетворкінгу Moya, оскільки ні Swif ні iOS-проект не мають окремих прошарків для роботи с сервером, в нашому випадку Firebase.

```
typealias PendingRequestPromise = (promise: Promise<Moya.Response>, cancellable: Cancellable)

func requestCodable<T: Decodable>(target: Target,
                                  queue: DispatchQueue? = nil,
                                  progress: Moya.ProgressBlock? = nil) -> Promise<T> {
    let promise = requestPromise(target, queue: queue, progress: progress)
    let promisetResult = Promise<T>.init { resolver in
        promise.map{ response in
            let response = try response.filterSuccessfulStatusCodes()
            do {
                let successResult = try response.map(T.self)
                resolver.fulfill(successResult)
            } catch {
                resolver.reject(BadResponseError(code: 666))
            }
        }
        .catch(resolver.reject)
    }
    return promisetResult
}

func requestPromise(_ target: Target,
                    queue: DispatchQueue? = nil,
                    progress: Moya.ProgressBlock? = nil) -> Promise<Moya.Response> {
    return requestCancellable(target: target, queue: queue, progress: progress).promise
}
```

Рис. 3.21. Приклад виконання запиту на сервер за допомогою Moya

```
private func requestCancellable(target: Target,
                                queue: DispatchQueue?,
                                progress: Moya.ProgressBlock? = nil) -> PendingRequestPromise {
    let pending = Promise<Moya.Response>.pending()
    let completion = promiseCompletion(resolver: pending.resolver)
    let cancellable: Cancellable = request(target, callbackQueue: queue, progress: progress, completion: completion)

    return (pending.promise, cancellable)
}

private func promiseCompletion(resolver: Resolver<Moya.Response>) -> Moya.Completion {
    return { result in
        switch result {
        case let .success(response):
            resolver.fulfill(response)
        case let .failure(error):
            resolver.reject(error)
        }
    }
}
```

Рис. 3.22. Приклад валідації відповіді з сервера за допомогою Moya

```
enum DahsboardError: String, Error {
    case noData
    case unknown
}

enum DahboardModel {
    case initial(() -> ())
    case loading
    case users([DashboardItem])
    case error(Error)

    struct DashboardItem {
        let name: String
        let icon: Icons
        let type: DashboardType
        let items:[DashboardItem]
        let time: String
        let onSelect: () -> ()
    }

    struct Error {
        let description: String
        let action: () -> ()
    }
}

protocol ViewOutput {
    var viewModel: CurrentValueSubject<DahboardModel,Error> { get }
}
```

Рис. 3.23. Модель для екрану розкладу

Модель на Рис. 3.18 є основним елементом в Data-Driven підході, оскільки архітектура – це VIPER то модель не є власністю View і по суті самостійною сутністю, котра вміщає в собі ймовірні варіанти наповнення вікна Розкладу, а саме:

- Відсутність даних
- Присутність даних
- Завантаження даних
- Помилка

Оновлення моделі на стороні презентера відбувається лише по відклику з View, коли користувач здійснює дію, а Presenter в свою чергу утримує й бізнес-логіку.

```
func populateViewModelWithError() {
    viewModelObserver.send(.loading)
    DispatchQueue.main.asyncAfter(deadline: .now() + .seconds(3)) { [weak self] in
        self?.viewModelObserver.send(
            .error(
                DahboardModel.Error(
                    description: "no data",
                    action: { self?.populateViewModelWithUsers() }
                )
            )
        )
    }
}

func populateViewModelWithData() {

    viewModelObserver.send(.loading)
    interactor.getDahsboardData { (data, error)
        if let networkData = data {
            self?.viewModelObserver.send(.items(networkData))
        } else {
            populateViewModelWithError()
        }
    }
}
```

Рис. 3.24. Оновлення моделі в презентері

Кожен раз коли модель оновлюється виконується стан “завантаження”, поки йде запит даних на сервер, в результаті Presenter віддає в модель дані чи помилку. Саме така логіка актуальна, бо всі запити на сервер асинхронні по своїй суті й розробник не може знати час його виконання.

Варто зауважити, що весь callback відбувається за допомогою анонімних функцій – лямбд, котрі нативно підтримує Swift. Вони є суворо однонаправленими у зв’язку з особливостями архітектурного підходу.

```
extension DashboardViewController: UITableViewDelegate, UITableViewDataSource {

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
         tableView.deselectRow(at: indexPath, animated: true)
        if case .users(let items) = output {
            let data = items[indexPath.row]
            data.onSelect()
        }
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        if case .items(let data) = output {
                return data.count
            }
            return 0
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        if case .items(let items) = output {
                let data = users[indexPath.row]
            return tableView.dequeueReusableCell(withIdentifier: "cell") ?? UITableViewCell(style: .value1,
                reuseIdentifier: "cell").then {
                    $0.textLabel?.font = UIFont.italicSystemFont(ofSize: 35)
                    $0.textLabel?.text = data.name
                    $0.detailTextLabel?.font = UIFont.italicSystemFont(ofSize: 35)
                    $0.detailTextLabel?.text = data.icon
                }
            }
            return UITableViewCell()
    }
}
```

Рис. 3.25. Приклад оновлення таблиці з даними розкладу після оновлення стану вікна

Кожен раз коли стан вікна змінюється – все вікно перемальовується. Це кращий підхід до менеджменту ресурсів, ніж коли вікно перемальовує окремі елементи, так і з точки зору структури класу UIViewController, оскільки все View перемальовується в одному місці. Все відбувається в момент життєвого циклу viewWillLayoutSubview.

```
override func viewWillLayoutSubviews() {
        super.viewWillLayoutSubviews()

    switch output {
     case .initial:
            tableView.isHidden = true
            loadingView.stopAnimating()
            errorLabel.isHidden = true
            reloadButton.isHidden = true
      case .loading:
            tableView.isHidden = true
            loadingView.startAnimating()
            errorLabel.isHidden = true
            reloadButton.isHidden = true
      case .items:
            tableView.isHidden = false
            tableView.reloadData()
            loadingView.stopAnimating()
            errorLabel.isHidden = true
            reloadButton.isHidden = true
          dashboardTableView.reloadData()
      case .error(let errorData):
            tableView.isHidden = true
            loadingView.stopAnimating()
            errorLabel.isHidden = false
            errorLabel.text = errorData.description
            reloadButton.isHidden = false
    }
}
```

Рис. 3.26. Приклад оновлення View з даними розкладу після оновлення стану вікна

Взаємодія з Apple Watch, а саме запит поточного серцевого ритму відбувається всередині iOS додатку через системний інтерфейс HealthKit.

```swift
public func fetchLatestHeartRateSample(
  completion: @escaping (_ sample: HKQuantitySample?) -> Void) {

  /// Create sample type for the heart rate
  guard let sampleType = HKObjectType
    .quantityType(forIdentifier: .heartRate) else {
      completion(nil)
    return
  }

  /// Predicate for specifiying start and end dates for the query
  let predicate = HKQuery
    .predicateForSamples(
      withStart: Date.distantPast,
      end: Date(),
      options: .strictEndDate)

  /// Set sorting by date.
  let sortDescriptor = NSSortDescriptor(
    key: HKSampleSortIdentifierStartDate,
    ascending: false)

  /// Create the query
  let query = HKSampleQuery(
    sampleType: sampleType,
    predicate: predicate,
    limit: Int(HKObjectQueryNoLimit),
    sortDescriptors: [sortDescriptor]) { (_, results, error) in

      guard error == nil else {
        print("Error: \(error!.localizedDescription)")
        return
      }

      completion(results?[0] as? HKQuantitySample)
  }

  self.healthStore.execute(query)
}
```

Рис. 3.27. Запит поточного серцевого ритму

3.5. Розробка watchOS додатку-супутнику

Додаток для WatchOS є розширенням базового додатку для iOS, що в свою чергу існує як “додаток в додатку”.

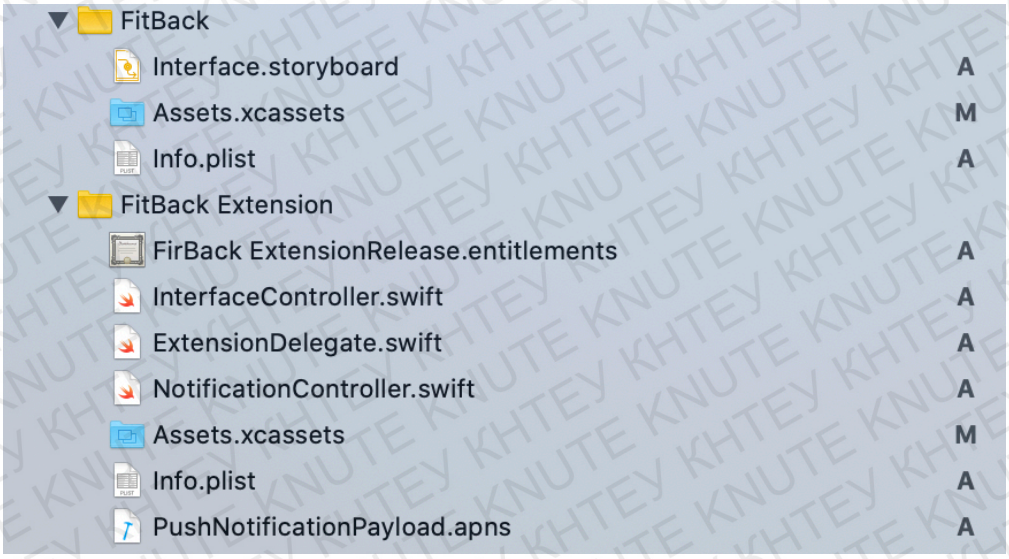


Рис. 3.27. Тека розширення для WatchOS

Розширення має власні налаштування та дозвільний файл, власний цільовий пристрій і версію системи, але вони не мають конфліктувати з параметрами заданими для основного додатку.

Рис. 3.28. Екосистема двох додатків

Варто розглянути логіку роботи тренування, з підрахунком часу і калорій
на WatchOS. Система на себе бере частину низькорівневих обчислень даних датчиків. Для опитування Apple Watch буде використовуватись HKHealthStore.

```
struct PrancerciseWorkout {
  var start: Date
  var end: Date

  init(start: Date, end: Date) {
    self.start = start
    self.end = end
  }

  var duration: TimeInterval {
    return end.timeIntervalSince(start)
  }

  var totalEnergyBurned: Double {
    let prancerciseCaloriesPerHour: Double = 450
    let hours: Double = duration / 3600
    let totalCalories = prancerciseCaloriesPerHour * hours
    return totalCalories
  }
}
```

Рис. 3.29. Модель для тренування

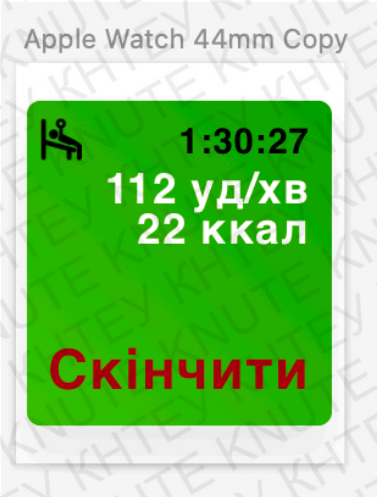


Рис. 3.30. Екран виконання вправи.

```
guard let quantityType = HKQuantityType.quantityType(forIdentifier:
  .activeEnergyBurned) else {
    completion(false, nil)
    return
}

let unit = HKUnit.kilocalorie()
let totalEnergyBurned = prancerciseWorkout.totalEnergyBurned
let quantity = HKQuantity(unit: unit,
                          doubleValue: totalEnergyBurned)
let sample = HKCumulativeQuantitySeriesSample(type: quantityType,
                                              quantity: quantity,
                                              start: prancerciseWorkout.start,
                                              end: prancerciseWorkout.end)

//1. Add the sample to the workout builder
builder.add([sample]) { (success, error) in
  guard success else {
    completion(false, error)
    return
  }

  //2. Finish collection workout data and set the workout end date
  builder.endCollection(withEnd: prancerciseWorkout.end) { (success, error) in
    guard success else {
      completion(false, error)
      return
    }

    //3. Create the workout with the samples added
    builder.finishWorkout { (_, error) in
      let success = error == nil
      completion(success, error)
    }
  }
}
```

Рис. 3.31. Логіка опитування датчиків в черзі.

Опитування Apple Watch проводилось на предмет спалених калорій. Датчики віддають в реальному часі всі заміри по поточній активності, котрі дають змогу реалізувати екран тренування на годиннику.

3.6. Висновки до розділу 3

Отже, у цьому розділі була розглянута структура роботи програмного забезпечення. Були продемонстровані всі етапи розробки програмного продукту: від прототипування до реального опитування апаратної платформи. Був розглянутий. Data-Driven підхід, котрий передбачає лише валідні варіанти роботи екранів додатку. Для нашого користувацького інтерфейсу кожен раз коли стан вікна змінюється – все вікно перемальовується. Це кращий підхід до менеджменту ресурсів, ніж коли вікно перемальовує окремі елементи, так і з точки зору структури класу UIViewController, оскільки все View перемальовується в одному місці. Все відбувається в момент життєвого циклу viewWillLayoutSubview. Кожен раз коли модель оновлюється виконується стан "завантаження", поки йде запит даних на сервер, в результаті Presenter віддає в модель дані чи помилку. Саме така логіка актуальна, бо всі запити на сервер асинхронні по своїй суті й розробник не може знати час його виконання. Даний підхід здатний забезпечити легке впровадження нового функціоналу, написання тестів та підтримки старого коду, що є фундаментальними речами у такому комплексному програмному продукті.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Була проведена велика робота над збором усієї актуальної інформації щодо моніторингу життєвих показників людини в сучасній індустрії мобільних пристроїв. Були розглянуті як теоретичні засади, так і апаратно-програмний комплекс засобів для втілення бізнес-задач пов'язаних з моніторингом життєвих показників, розглято конструкцію і принцип роботи більшості датчиків, котрі беруть участь у створення і обробці статистичних даних для подальшої експлуатації їх в додатку та аналізу.

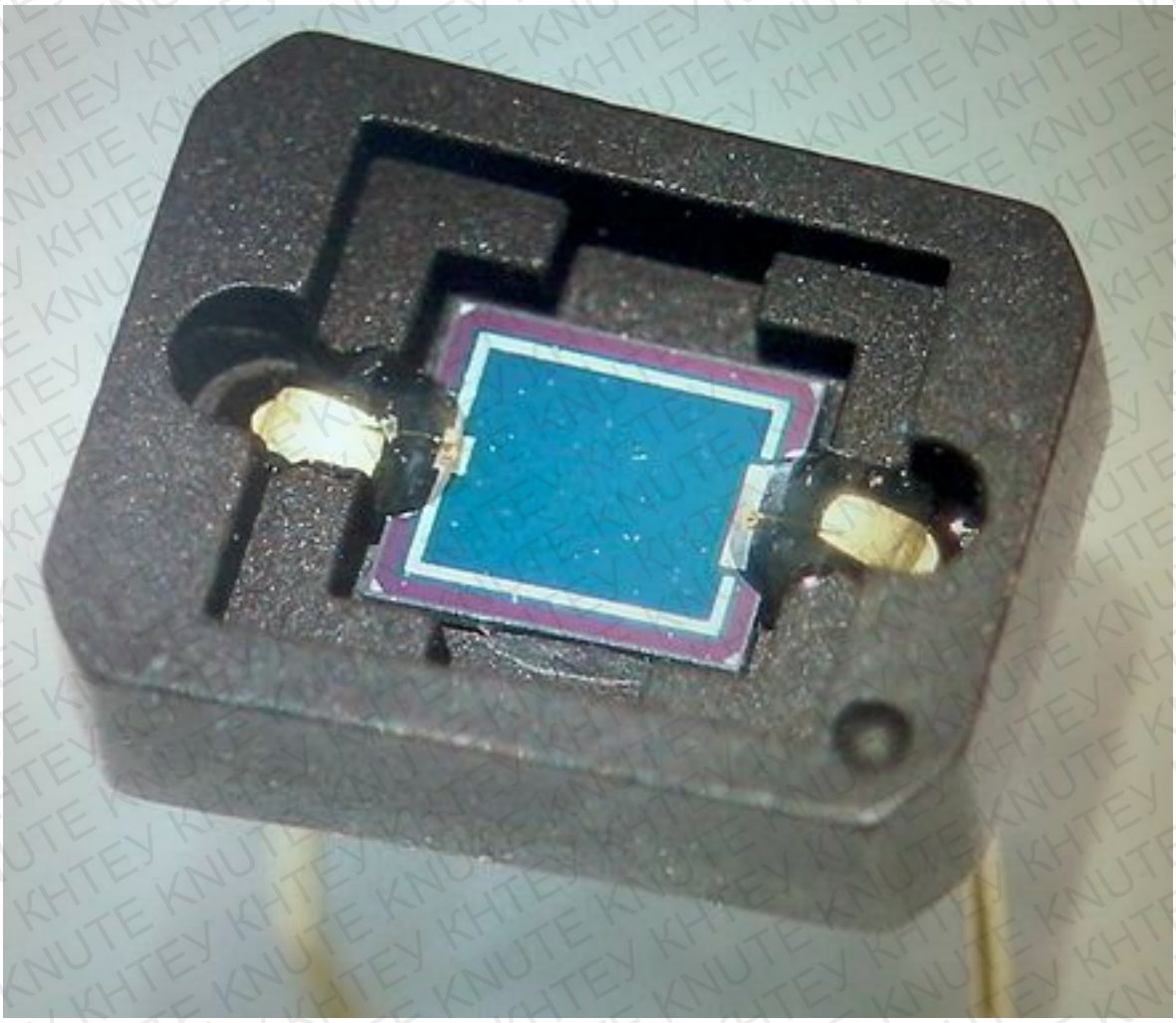
Були обрані програмно-апаратні платформи для розробки, а саме Apple iPhone на базі iOS 13 та Apple Watch на базі WatchOS 6. Такий вибір забезпечила чудова екоситсема Apple, що дозволила втілити багато корисної логіки для обох платформ. Після огляду сучасних підходів до архітектури додатків було обрано протокольно-орієнтовну архітектуру VIPER, з підходом Data-Driven.

Додаток спочатку був спрототипований, а потім прототипи були перетворені в реальний користувацький інтерфейс, після чого почалася розробка. Створення БД на базі Firebase дозволило почати працювати уже з реальними даними в самому додатку. Все розроблялось суворо згідно архітектурного паттерну та код-стилю мови програмування Swift, котра була створена для роботи з апаратними платформами Apple.

					КНТЕУ 121– 05-05.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*	*Розробка мобільного додатку зчитування фізіологічних даних людини*	*Стадія*	*Аркуш*	*Аркушів*
Зав. каф.		Криворучко О.В.				СП	*47*	*48*
Керівник		Палагута К.О.						
Гарант		Криворучко О.В			*Список використаних літературних джерел*	Факультет обліку, аудиту та інформаційних систем, 2м курс, 5 група		
Розроб		Вишняк Я.О.						

Після чого були розроблені iOS-додаток і додаток-супутник. Для них була написана логіка заміру серцевого ритму і підрахунок спалених калорій. Дані, отримані за допомогою пульсометра, допоможуть зрозуміти, як ваш організм реагує на навантаження. Тим не менш, не потрібно ставати заручником цифр і гаджетів. Треба вчитися слухати свій організм, оцінюйте відчуття від кожного тренування, ну а цифри стануть важливим додатковим джерелом інформації.

Фотоелемент ***Додаток А***



Оптичний датчик пульсу ***Додаток Б***

Алгоритм роботи датчику пульсу ***Додаток В***

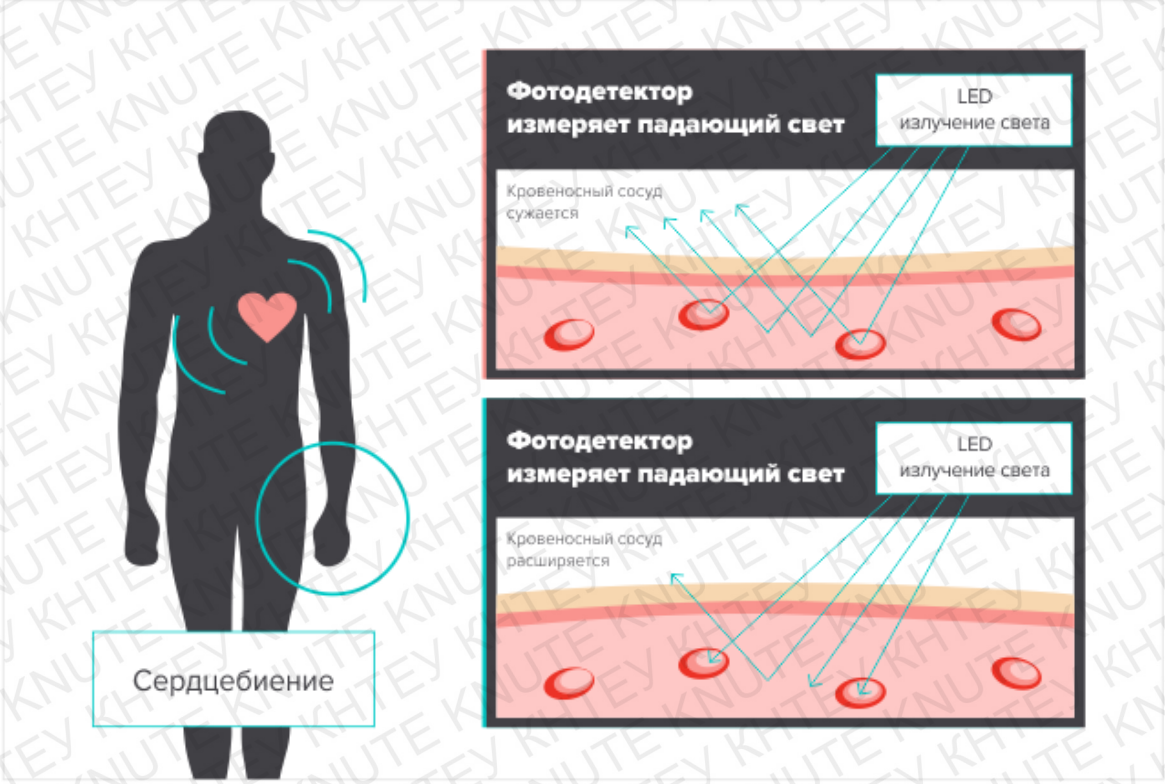
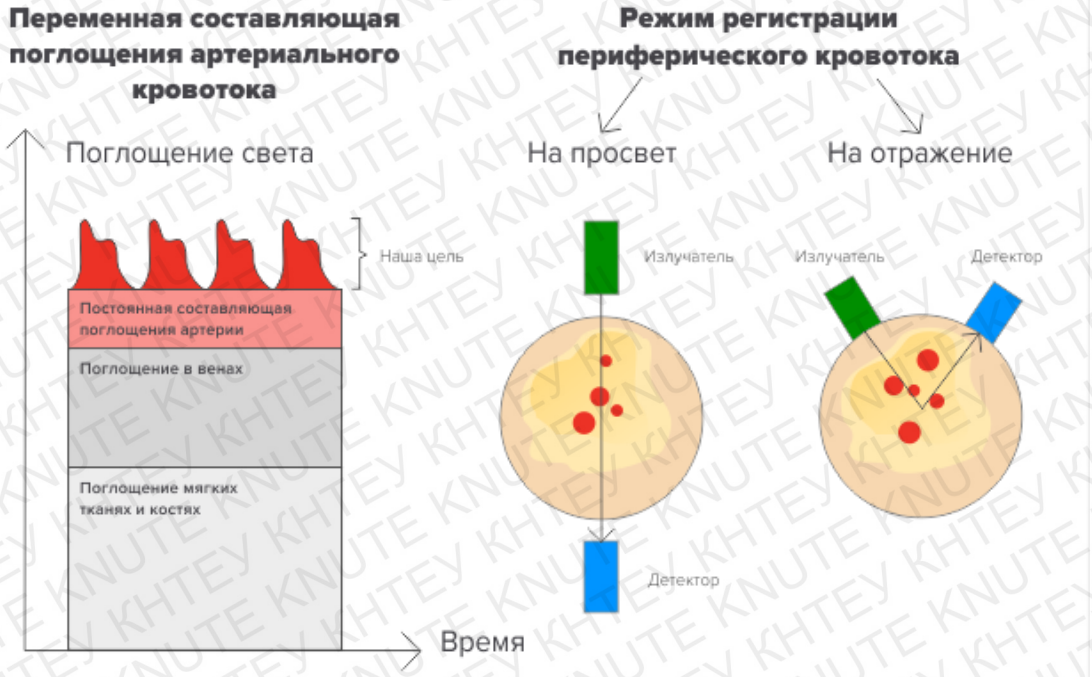
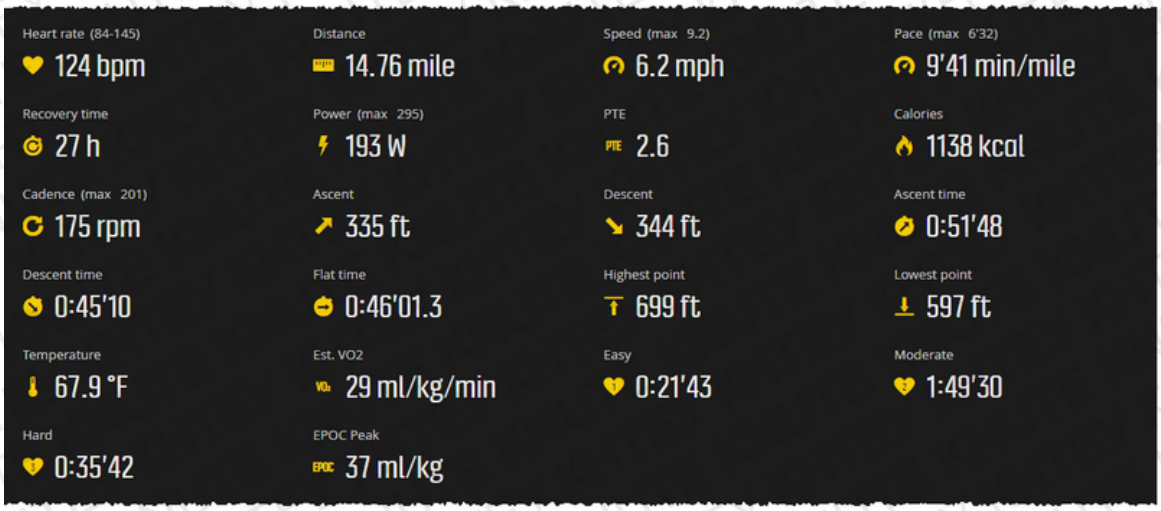


Схема датчику пульсу ***Додаток Г***



Можливі вимірювання ***Додаток Г***



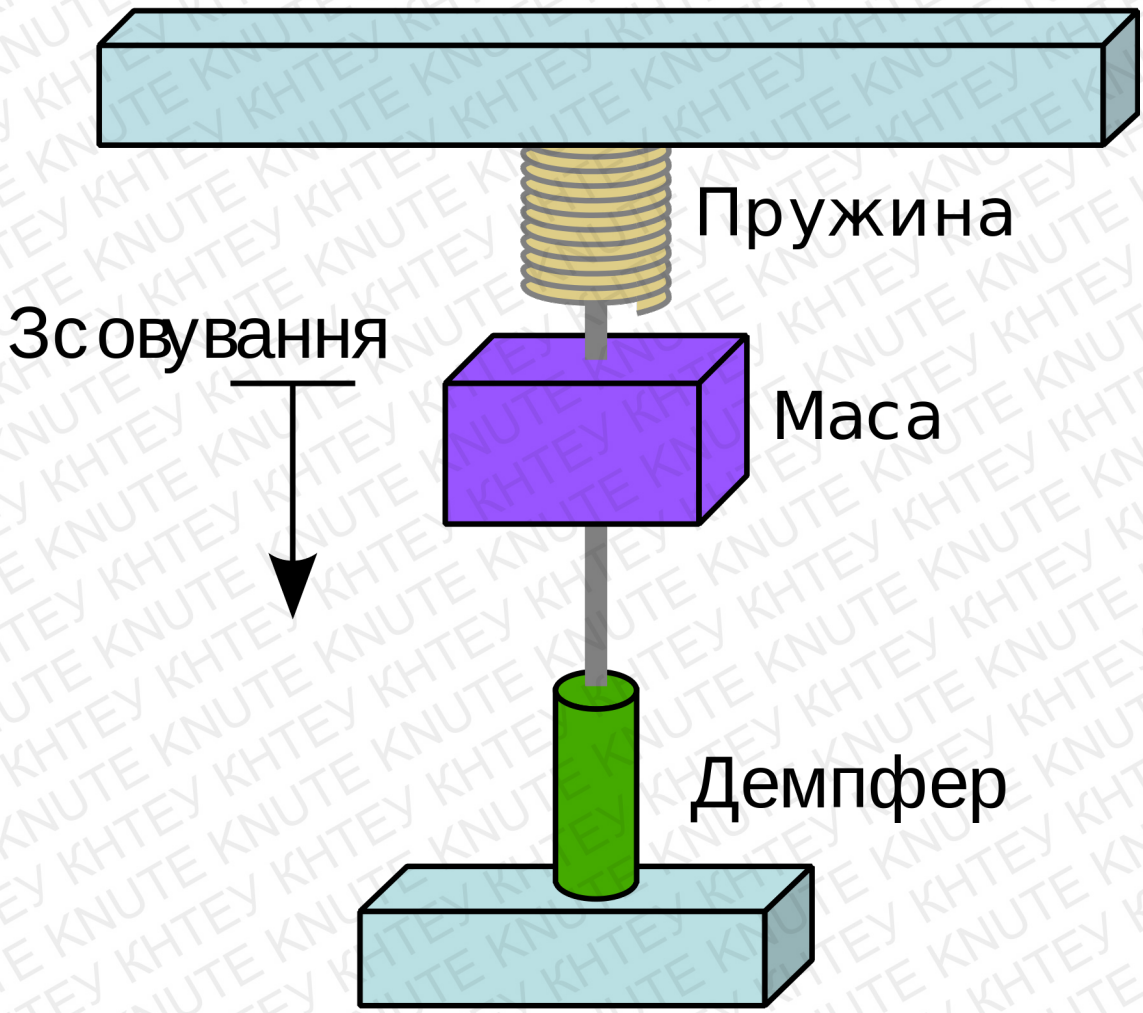
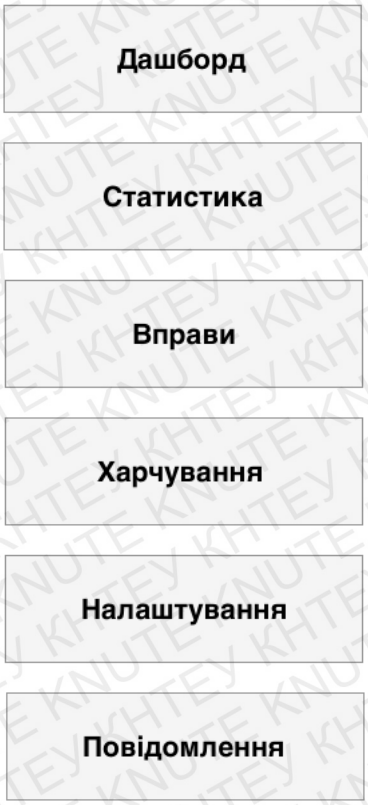
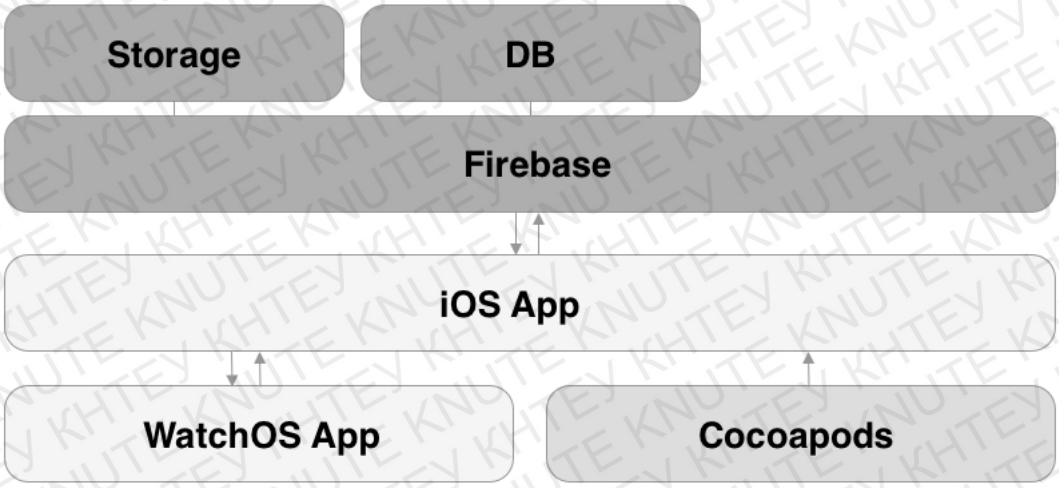

Пружина
Зсовування
Маса
Демпфер

Схема логічних елементів додатку ***Додаток Е***



Струкутурадодатку ***Додаток Є***



Архітектура додатку ***Додаток Ж***

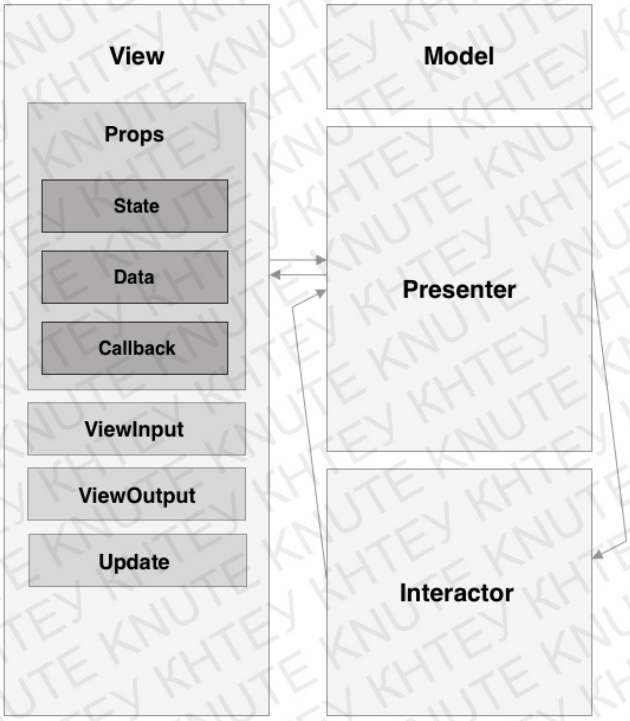
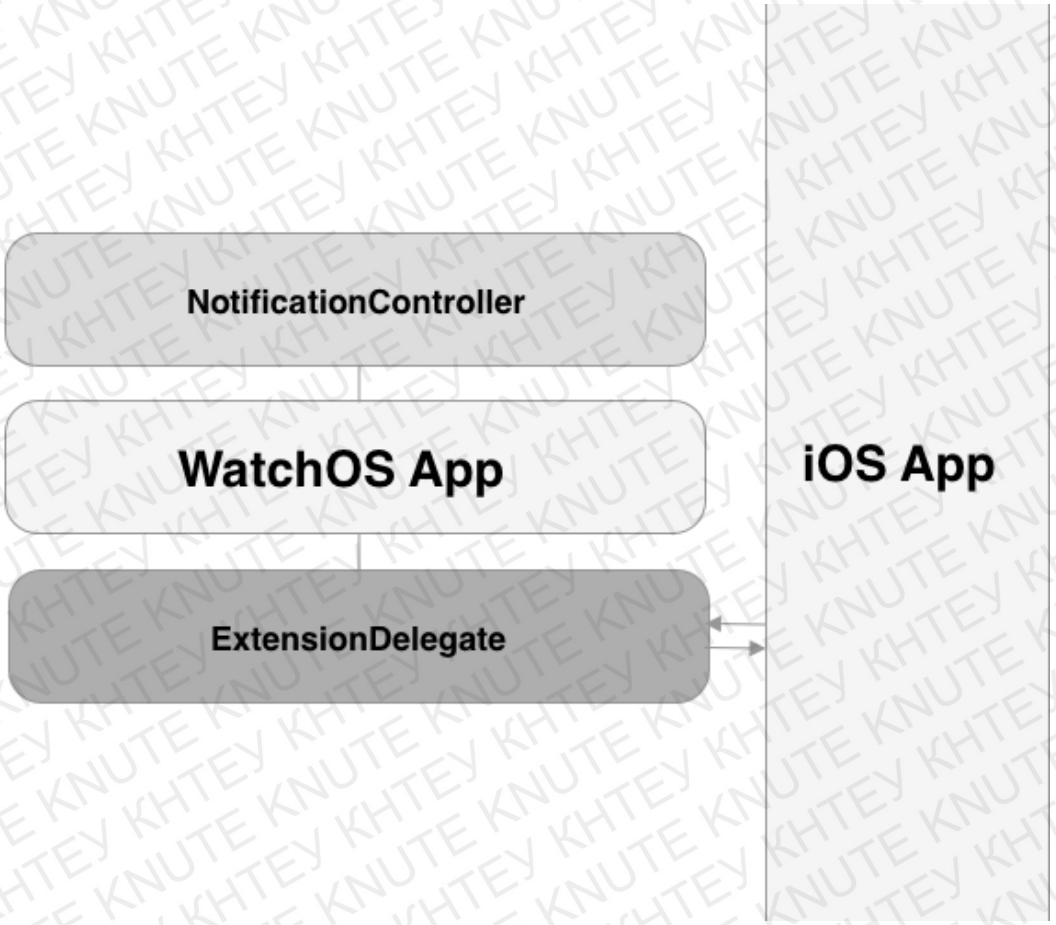


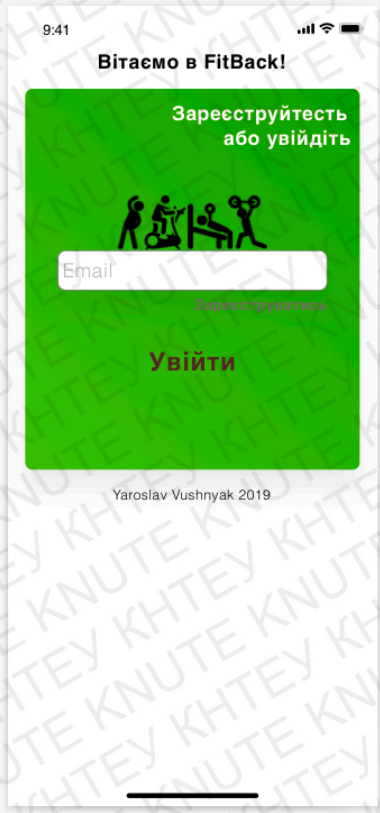
Схема взаємодії додатків ***Додаток З***

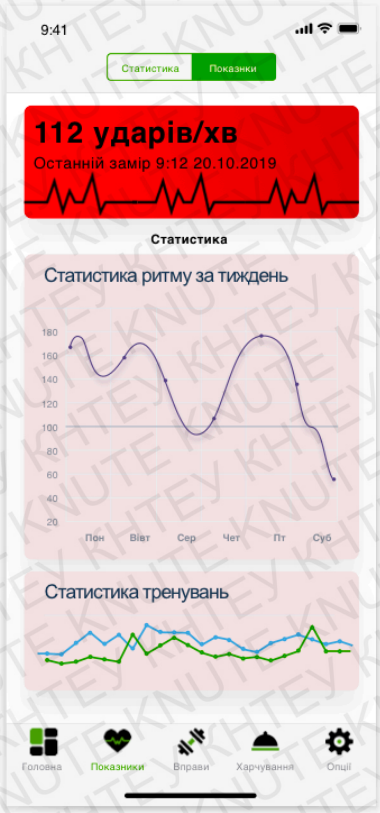


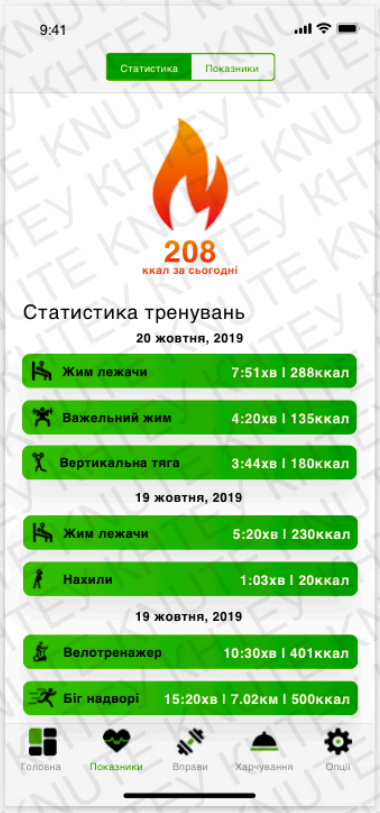

9:41
На сьогодні, 20 жовтня, 2019
Сніданок 9:30
Рис 140г | 200ккал
Руккола 50г | 70ккал
Яйце куряче 70г | 113ккал
Розминка 11:00
Розминка шиї 10 | 3 підх
Розминка ніг 13 | 2 підх
Оберти тулуба 14 | 2 підх
Кардіо 15:00
Бігова доріжка 9км/г | 20 хв
Велотренажер 12км/г | 15 хв
Відпочинок 15:45 - 18:00
18:15
Курячий бульйон 200г | 220ккал
Шпинат 50г | 30ккал
Гречнева каша 150г | 150ккал
Відпочинок 18:30-20:00
Силове тренування 20:00
Жим лежачи 80кг | 12 | 3 підх
Важельний жим 50кг | 10 | 3 підх
Верт. тяга 30кг | 15 | 3 підх
Головна
Показники
Вправи
Харчування
Опції


9:41
Силове тренування
20:00
Жим лежачи
80кг | 12 повторень
75кг | 10 повторень
60кг | 10 повторень
Відпочинок: 1-3хв
Важельний жим
50кг | 10 повторень
45кг | 10 повторень
40кг | 10 повторень
Відпочинок: 1хв
Вертикальна тяга
50кг | 10 повторень
45кг | 10 повторень
40кг | 10 повторень
Відпочинок: 1хв
Орієнтовна тривалість 45хв
Орієнтовне спалення 320ккал
Головна
Показники
Вправи
Харчування
Опції


9:41
Силове тренування
20:00
Жим лежачи
80кг | 12 повторень
75кг | 10 повторень
60кг | 10 повторень
Відпочинок: 1-3хв
1:30:27
112 уд/хв
Завершити вправу
Важельний жим
50кг | 10 повторень
45кг | 10 повторень
40кг | 10 повторень
Відпочинок: 1хв
Вертикальна тяга
50кг | 10 повторень
45кг | 10 повторень
40кг | 10 повторень
Відпочинок: 1хв
Орієнтовна тривалість 45хв
Орієнтовне спалення 320ккал
Головна
Показники
Вправи
Харчування
Опції

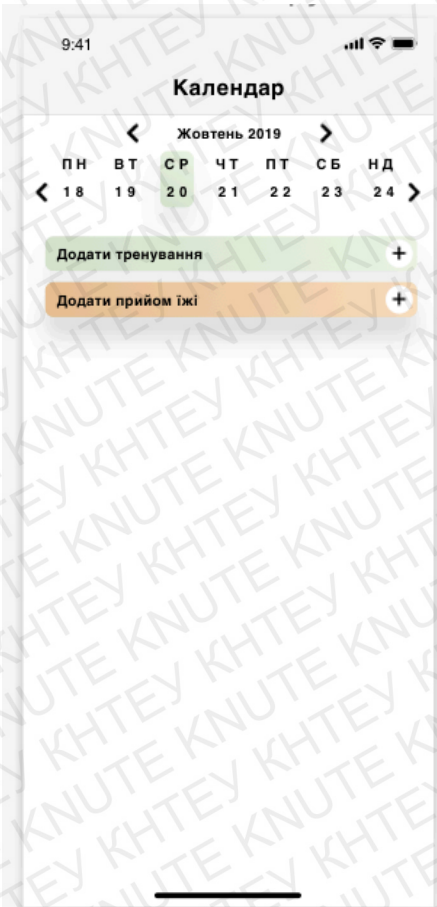

9:41
Вітаємо в FitBack!
Зареєструйтесь або увійдіть
Email
Зареєструватись
Увійти
Yaroslav Vushnyak 2019

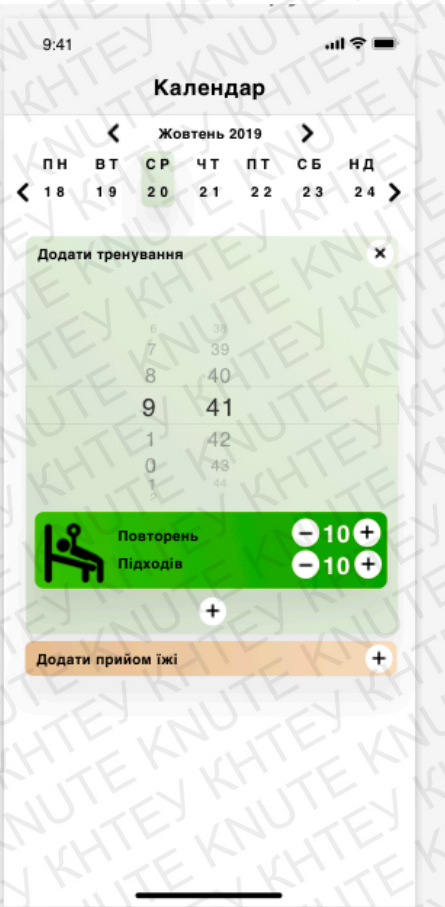

9:41
Статистика
Показники
112 ударів/хв
Останній замір 9:12 20.10.2019
Статистика
Статистика ритму за тиждень
180
160
140
120
100
80
60
40
20
Пон
Вівт
Сер
Чет
Пт
Суб
Статистика тренувань
Головна
Показники
Вправи
Харчування
Опції

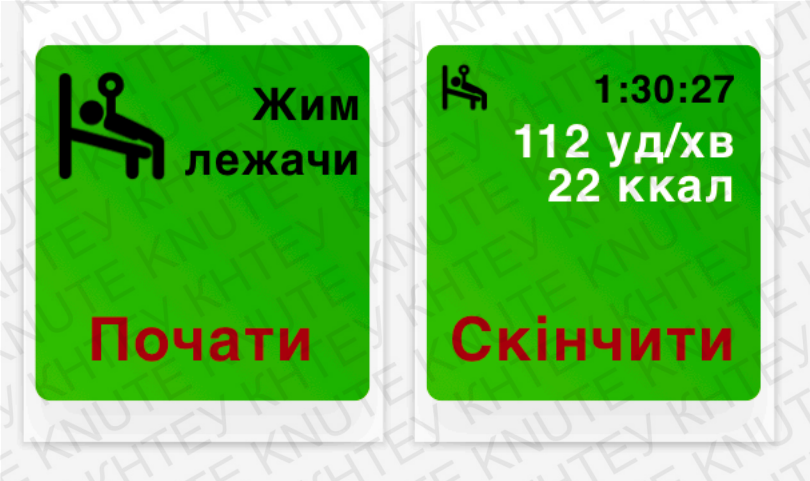

9:41
Статистика
Показники
208
ккал за сьогодні
Статистика тренувань
20 жовтня, 2019
Жим лежачи 7:51хв | 288ккал
Важельний жим 4:20хв | 135ккал
Вертикальна тяга 3:44хв | 180ккал
19 жовтня, 2019
Жим лежачи 5:20хв | 230ккал
Нахили 1:03хв | 20ккал
19 жовтня, 2019
Велотренажер 10:30хв | 401ккал
Біг надворі 15:20хв | 7.02км | 500ккал
Головна
Показники
Вправи
Харчування
Опції


9:41
Вправи
Головна
Показники
Вправи
Харчування
Опції


9:41
Харчування
М'ясні страви
Сер ккал: 200
13
Супи
Сер ккал: 157
8
Гарніри
Сер ккал: 190
15
Сніданки
Сер ккал: 110
5
Овочі
Сер ккал: 70
17
Головна
Показники
Вправи
Харчування
Опції


9:41
Календар
Жовтень 2019
ПН ВТ СР ЧТ ПТ СБ НД
18 19 20 21 22 23 24
Додати тренування
Додати прийом їжі


9:41
Календар
Жовтень 2019
ПН ВТ СР ЧТ ПТ СБ НД
18 19 20 21 22 23 24
Додати тренування
Повторень
10
Підходів
10
Додати прийом їжі



Серверна взаємодія лістинг ***Додаток І***

```
import
PromiseKit
        import Moya


        public extension MoyaProvider {
            typealias PendingRequestPromise = (promise: Promise<Moya.Response>,
        cancellable: Cancellable)

            func requestCodable<T: Decodable>(target: Target,
                                              queue: DispatchQueue? = nil,
                                              progress: Moya.ProgressBlock? =
        nil) -> Promise<T> {
                let promise = requestPromise(target, queue: queue, progress: progress)
                let promisetResult = Promise<T>.init { resolver in
                    promise.map{ response in
                        let response = try response.filterSuccessfulStatusCodes()
                        do {
                            let successResult = try response.map(T.self)
                            resolver.fulfill(successResult)
                        } catch {
                            resolver.reject(BadResponseError(code: 666))
                        }
                    }
                    .catch(resolver.reject)
                }
                return promisetResult
            }

            func requestPromise(_ target: Target,
                                queue: DispatchQueue? = nil,
```

```
                        progress: Moya.ProgressBlock? = nil) ->
Promise<Moya.Response> {
        return requestCancellable(target: target, queue: queue, progress:
progress).promise
    }

    private func requestCancellable(target: Target,
                                    queue: DispatchQueue?,
                                    progress: Moya.ProgressBlock? = nil) ->
PendingRequestPromise {
        let pending = Promise<Moya.Response>.pending()
        let completion = promiseCompletion(resolver: pending.resolver)
        let cancellable: Cancellable = request(target, callbackQueue: queue,
progress: progress, completion: completion)

        return (pending.promise, cancellable)
    }

    private func promiseCompletion(resolver: Resolver<Moya.Response>) ->
Moya.Completion {
        return { result in
            switch result {
            case let .success(response):
                resolver.fulfill(response)
            case let .failure(error):
                resolver.reject(error)
            }
        }
    }
}
```

```
enum UseType {
    case train
    case food
}

struct DashboardItem {
    let name:String
    let useType:UseType
    let repeats:Int
    let weight:Int
}

enum DashboardType {
    case train
    case food
}
enum Icons {
    case trainVerticalPush
    case trainRun
    case trainVelo

    func getImage() -> UIImage {
        return UIImage()
    }
}


enum DahsboardError: String, Error {
    case noData
    case unknown
}

enum DahboardModel {
    case initial(() -> ())
    case loading
    case items([DashboardItem])
    case error(Error)

    struct DashboardItem {
        let name: String
        let icon: Icons
        let type: DashboardType
        let items:[DashboardItem]
        let time: String
        let onSelect: () -> ()
    }

    struct Error {
        let description: String
```

```
        let action: () -> ()
    }
}

protocol ViewOutput {
    var viewModel: CurrentValueSubject<DahboardModel,Error> {
get }
}
```

```
public func subscribeToHeartBeatChanges() {

  // Creating the sample for the heart rate
  guard let sampleType: HKSampleType =
    HKObjectType.quantityType(forIdentifier: .heartRate) else {
      return
  }

  /// Creating an observer, so updates are received whenever HealthKit's
  // heart rate data changes.
  self.heartRateQuery = HKObserverQuery.init(
    sampleType: sampleType,
    predicate: nil) { [weak self] _, _, error in
      guard error == nil else {
        log.warn(error!)
        return
      }

      /// When the completion is called, an other query is executed
      /// to fetch the latest heart rate
      self.fetchLatestHeartRateSample(completion: { sample in
        guard let sample = sample else {
          return
        }

        /// The completion in called on a background thread, but we
        /// need to update the UI on the main.
        DispatchQueue.main.async {

          /// Converting the heart rate to bpm
          let heartRateUnit = HKUnit(from: "count/min")
          let heartRate = sample
            .quantity
            .doubleValue(for: heartRateUnit)

          /// Updating the UI with the retrieved value
          self?.heartRateLabel.setText("\(Int(heartRate))")
        }
      })
  }
}

public func fetchLatestHeartRateSample(
  completion: @escaping (_ sample: HKQuantitySample?) -> Void) {

  /// Create sample type for the heart rate
  guard let sampleType = HKObjectType
    .quantityType(forIdentifier: .heartRate) else {
```

```
      completion(nil)
    return
  }

  /// Predicate for specifiying start and end dates for the query
  let predicate = HKQuery
    .predicateForSamples(
      withStart: Date.distantPast,
      end: Date(),
      options: .strictEndDate)

  /// Set sorting by date.
  let sortDescriptor = NSSortDescriptor(
    key: HKSampleSortIdentifierStartDate,
    ascending: false)

  /// Create the query
  let query = HKSampleQuery(
    sampleType: sampleType,
    predicate: predicate,
    limit: Int(HKObjectQueryNoLimit),
    sortDescriptors: [sortDescriptor]) { (_, results, error) in

      guard error == nil else {
        print("Error: \(error!.localizedDescription)")
        return
      }

      completion(results?[0] as? HKQuantitySample)
  }

  self.healthStore.execute(query)
}

self.fetchLatestHeartRateSample(completion: { sample in
  guard let sample = sample else {
    return
  }

  /// The completion in called on a background thread, but we
  /// need to update the UI on the main.
  DispatchQueue.main.async {

    /// Converting the heart rate to bpm
    let heartRateUnit = HKUnit(from: "count/min")
    let heartRate = sample
      .quantity
      .doubleValue(for: heartRateUnit)

    /// Updating the UI with the retrieved value
    self?.heartRateLabel.setText("\(Int(heartRate))")}})
```

```
struct PrancerciseWorkout {
  var start: Date
  var end: Date

  init(start: Date, end: Date) {
    self.start = start
    self.end = end
  }

  var duration: TimeInterval {
    return end.timeIntervalSince(start)
  }

  var totalEnergyBurned: Double {
    let prancerciseCaloriesPerHour: Double = 450
    let hours: Double = duration / 3600
    let totalCalories = prancerciseCaloriesPerHour * hours
    return totalCalories
  }
}

//
//  DashboardViewController.swift
//  FitBackApp
//
//  Created by Yaroslav Vushnyak on 26.11.2019.
//  Copyright © 2019 Yaroslav Vushnyak. All rights reserved.
//

import UIKit
import Then
import Combine

let healthStore = HKHealthStore()
let workoutConfiguration = HKWorkoutConfiguration()
workoutConfiguration.activityType = .other

let builder = HKWorkoutBuilder(healthStore: healthStore,
                               configuration:
workoutConfiguration,
                               device: .local())

builder.beginCollection(withStart: prancerciseWorkout.start) {
(success, error) in
  guard success else {
    completion(false, error)
    return
  }
}
```

```
guard let quantityType = HKQuantityType.quantityType(
  forIdentifier: .activeEnergyBurned) else {
  completion(false, nil)
  return
}

let unit = HKUnit.kilocalorie()
let totalEnergyBurned = prancerciseWorkout.totalEnergyBurned
let quantity = HKQuantity(unit: unit,
                          doubleValue: totalEnergyBurned)

let sample = HKCumulativeQuantitySeriesSample(type:
quantityType,
quantity: quantity,
start: prancerciseWorkout.start,
end: prancerciseWorkout.end)

builder.add([sample]) { (success, error) in
  guard success else {
    completion(false, error)
    return
  }

  //2. Finish collection workout data and set the workout end
date
  builder.endCollection(withEnd: prancerciseWorkout.end) {
(success, error) in
    guard success else {
      completion(false, error)
      return
    }

    //3. Create the workout with the samples added
    builder.finishWorkout { (_, error) in
      let success = error == nil
      completion(success, error)
    }
  }
}

class func loadPrancerciseWorkouts(completion:
  @escaping ([HKWorkout]?, Error?) -> Void) {
  //1. Get all workouts with the "Other" activity type.
  let workoutPredicate = HKQuery.predicateForWorkouts(with:
.other)

  //2. Get all workouts that only came from this app.
  let sourcePredicate = HKQuery.predicateForObjects(from:
.default())

  //3. Combine the predicates into a single predicate.
```

```
  let compound =
NSCompoundPredicate(andPredicateWithSubpredicates:
    [workoutPredicate, sourcePredicate])

  let sortDescriptor = NSSortDescriptor(key:
HKSampleSortIdentifierEndDate,
                                        ascending: true)
}

let query = HKSampleQuery(
  sampleType: .workoutType(),
  predicate: compound,
  limit: 0,
  sortDescriptors: [sortDescriptor]) { (query, samples, error)
in
    DispatchQueue.main.async {
      guard
        let samples = samples as? [HKWorkout],
        error == nil
        else {
          completion(nil, error)
          return
      }

      completion(samples, nil)
    }
  }

HKHealthStore().execute(query)

WorkoutDataStore.loadPrancerciseWorkouts { (workouts, error) in
  self.workouts = workouts
  self.tableView.reloadData()
}

extension WorkoutsTableViewController {
  override func tableView(_ tableView: UITableView,
                          numberOfRowsInSection section: Int) ->
Int {
    return workouts?.count ?? 0
  }
}

override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
  guard let workouts = workouts else {
    fatalError("""
      CellForRowAtIndexPath should \
      not get called if there are no workouts
      """)
  }
```

```
    //1. Get a cell to display the workout in
    let cell = tableView.dequeueReusableCell(withIdentifier:
      prancerciseWorkoutCellID, for: indexPath)

    //2. Get the workout corresponding to this row
    let workout = workouts[indexPath.row]

    //3. Show the workout's start date in the label
    cell.textLabel?.text = dateFormatter.string(from:
workout.startDate)

    //4. Show the Calorie burn in the lower label
    if let caloriesBurned =
        workout.totalEnergyBurned?.doubleValue(for:
.kilocalorie()) {
      let formattedCalories = String(format: "CaloriesBurned:
%.2f",
                                     caloriesBurned)

      cell.detailTextLabel?.text = formattedCalories
    } else {
      cell.detailTextLabel?.text = nil
    }

    return cell
}

struct PrancerciseWorkoutInterval {
    var start: Date
    var end: Date

    var duration: TimeInterval {
      return end.timeIntervalSince(start)
    }

    var totalEnergyBurned: Double {
      let prancerciseCaloriesPerHour: Double = 450
      let hours: Double = duration / 3600
      let totalCalories = prancerciseCaloriesPerHour * hours
      return totalCalories
    }
}

struct PrancerciseWorkout {
    var start: Date
    var end: Date
    var intervals: [PrancerciseWorkoutInterval]

    init(with intervals: [PrancerciseWorkoutInterval]) {
      self.start = intervals.first!.start
      self.end = intervals.last!.end
```

```
    self.intervals = intervals
  }
}

var totalEnergyBurned: Double {
  return intervals.reduce(0) { (result, interval) in
    result + interval.totalEnergyBurned
  }
}

var duration: TimeInterval {
  return intervals.reduce(0) { (result, interval) in
    result + interval.duration
  }
}

var state: WorkoutSessionState = .notStarted

var intervals: [PrancerciseWorkoutInterval] = []

private func addNewInterval() {
  let interval = PrancerciseWorkoutInterval(start: startDate,
                                            end: endDate)
  intervals.append(interval)
}

func end() {
  endDate = Date()
  addNewInterval()
  state = .finished
}

intervals.removeAll()

var completeWorkout: PrancerciseWorkout? {
  guard state == .finished, intervals.count > 0 else {
    return nil
  }

  return PrancerciseWorkout(with: intervals)
}

private class func samples(for workout: PrancerciseWorkout) ->
[HKSample] {
  //1. Verify that the energy quantity type is still available
to HealthKit.
  guard let energyQuantityType = HKSampleType.quantityType(
    forIdentifier: .activeEnergyBurned) else {
      fatalError("*** Energy Burned Type Not Available ***")
  }

  //2. Create a sample for each PrancerciseWorkoutInterval
```

```
    let samples: [HKSample] = workout.intervals.map { interval in
      let calorieQuantity = HKQuantity(unit: .kilocalorie(),
                                       doubleValue:
interval.totalEnergyBurned)

      return HKCumulativeQuantitySeriesSample(type:
energyQuantityType,
                                              quantity:
calorieQuantity,
                                              start:
interval.start,
                                              end:
interval.end)
    }

    return samples
}

guard let quantityType =
HKQuantityType.quantityType(forIdentifier:
    .activeEnergyBurned) else {
      completion(false, nil)
      return
}

let unit = HKUnit.kilocalorie()
let totalEnergyBurned = prancerciseWorkout.totalEnergyBurned
let quantity = HKQuantity(unit: unit,
                          doubleValue: totalEnergyBurned)
let sample = HKCumulativeQuantitySeriesSample(type:
quantityType,
                                              quantity:
quantity,
                                              start:
prancerciseWorkout.start,
                                              end:
prancerciseWorkout.end)

//1. Add the sample to the workout builder
builder.add([sample]) { (success, error) in
    guard success else {
      completion(false, error)
      return
    }

    //2. Finish collection workout data and set the workout end
date
    builder.endCollection(withEnd: prancerciseWorkout.end) {
(success, error) in
      guard success else {
        completion(false, error)
        return
```

```
    }

    //3. Create the workout with the samples added
    builder.finishWorkout { (_, error) in
      let success = error == nil
      completion(success, error)
    }
  }
}

et samples = self.samples(for: prancerciseWorkout)

builder.add(samples) { (success, error) in
  guard success else {
    completion(false, error)
    return
  }

  builder.endCollection(withEnd: prancerciseWorkout.end) {
(success, error) in
    guard success else {
      completion(false, error)
      return
    }

    builder.finishWorkout { (workout, error) in
      let success = error == nil
      completion(success, error)
    }
  }
}
```

Київський національний торговельно-економічний університет

Факультет_____ Кафедра _____
_____
Освітній ступінь_____
Спеціальність _____
Спеціалізація / освітня програма _____

Затверджую

Зав. кафедри _____
_____
«___» _____20__ р.

Завдання
на випускну кваліфікаційну роботу студентові

Вишняку Ярославу Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи *розробка мобільного додатку зчитування фізіологічних даних людини.* Затверджена наказом ректора від "7" листопада 2018 р. № 41813

2. Строк здачі студентом закінченого роботи (проекту)_____

3. Цільова установка та вихідні дані до роботи (проекту)

Мета роботи розробка мобільного додатку зчитування фізіологічних даних людини

Об'єкт дослідження фізична діяльність людини.

Предмет дослідження методи та алгоритми для зчитування, класифікації та обробки фізіологічних даних під час фізичної активності.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1

ОСНОВНІ ПОНЯТТЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ ТА ДОДАТКІВ-СУПУТНИКІВ ДЛЯ ЗЧИТУВАННЯ БІОЛОГІЧНИХ ДАНИХ

1.1. Основи прикладного застосування мобільних додатків у цілях моніторингу здоров'я

1.2. Технічне забезпечення для вимірювання фізіологічних показників

1.3. Прикладний функціонал, заснований на моніторингу життєвих показників

1.4. Висновки та пропозиції до Розділу 1

РОЗДІЛ 2

ПРОЕКТУВАННЯ ДОДАТКУ

2.1. Постановка задачі при проектуванні програмного забезпечення

2.2. Вибір засобів розробки

2.3. Проектування архітектури додатку

2.4. Методологія розробки

2.5 Висновки та пропозиції до Розділу 2

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Прототипування та дизайн додатку

3.2.Побудова архітектури та алгоритму роботи

3.3. Розробка серверної частини та БД

3.4. Розробка iOS додатку

3.5. Розробка WatchOS додатку-супутнику

3.6. Висновки до Розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*25.09.2018*	
2.	*Розробка та затвердження завдання на проект магістра (Наказ №185 від 07.11.18)*	*07.11.2018*	
3.	*Вступ та перелік літературних джерел*	*28.02.2019*	
4.	*Розробка технічного завдання*	*22.03.2019*	
5.	*Розділ 1. Основні поняття про розробку мобільних додатків та додатків-супутників для зчитування біологічних даних*	*18.04.2019*	
6.	*Розділ 2. Проектування додатку*	*24.05.2019*	
7.	*Розділ 3. Розробка програмного забезпечення*	*21.06.2019*	
8.	*Розробка програми та методики тестування*	*18.10.2019*	
9.	*Написання наукової статті*	*27.09.2019*	
10.	*Керівництво користувача*	*23.10.2019*	
11.	*Висновки та пропозиції*	*01.11.2019*	
12.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*	*13.11.2019*	
13.	*Підготовка автореферату та презентації доповіді*	*14.11.2019*	
14.	*Попередній захист випускної кваліфікаційної роботи*	*14.11.2019 – 15.11.2019*	
15.	*Здача зброшурованої випускної кваліфікаційної роботи*	*25.11.2019*	
16.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*26.11.2019*	
17.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*02.12.2019-04.12.2019*	

7. Дата видачі завдання **«26» вересня 2018 р.**

8. Науковий керівник випускної кваліфікаційної роботи

_____

(прізвище, ініціали, підпис)

9. Гарант освітньої програми _____

(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент_____

(прізвище, ініціали, підпис)

11. Відгук керівника випускної кваліфікаційної роботи

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Науковий керівник випускної кваліфікаційної роботи_____
(підпис, дата)

Відмітка про попередній захист _____
(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____
(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____
(прізвище, ініціали, підпис)

Завідувач кафедри _____
(підпис, прізвище, ініціали)