**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ**

**КАФЕДРА КОМП'ЮТЕРНИХ НАУК**

# ВИПУСКНА РОБОТА

**на тему:**

## «Мобільний додаток для створення користувацького кулінарного довідника»

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

**Керівник роботи** — **Шовкопляс О. А.**

**Студента групи ІН – 62** — **Чернякова А. О.**

**СУМИ 2020**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

**Кафедра комп'ютерних наук**

Затверджую_____

Зав. кафедри Довбиш А.С.

"_____" _____20___ р.

# ЗАВДАННЯ

## до випускної роботи

Студентки четвертого курсу, групи ІН-62 спеціальності "Інформатика" денної форми навчання Чернякової Анни Олександрівни.

**Тема:** "**Мобільний додаток для створення користувацького кулінарного довідника**"

Затверджена наказом по СумДУ

№_____ від _____20___ р.

**Зміст пояснювальної записки:** 1) інформаційний огляд та аналіз; 2) постановка задачі; 3) вибір методу вирішення задачі; 4) програмна реалізація

Дата видачі завдання "_____"_____20___ р.

Керівник випускної роботи _____Шовкопляс О. А.

Завдання прийняв до виконання _____Чернякова А. О.

# РЕФЕРАТ

**Записка:** 59 стор., 16 рис., 1 табл., 1 додаток, 12 джерел.

**Об’єкт дослідження** – методи проектування та технології створення мобільних додатків.

**Мета роботи** – розробка мобільного додатка для створення користувацького кулінарного довідника.

**Методи дослідження –** технології гібридної розробки.

**Результати** – проведено аналіз типів додатків та обрано найбільш підходящий, після чого здійснено огляд фреймворків для гібридної розробки та вибір і проектування бази даних. Було створено прототип мобільного додатка, після чого описано програмну реалізацію. У результаті, розроблено та протестовано мобільний додаток для створення користувацького кулінарного довідника за допомогою фреймворку React Native на мові Java Script.

ГІБРИДНА РОЗРОБКА, ФРЕЙМВОРК REACT NATIVE,
КУЛІНАРНИЙ ДОДАТОК, ДОДАТОК НА ANDROID,
КРОСПЛАТФОРМЕННІСТЬ

.

# ЗМІСТ

# ВСТУП

Мобільні пристрої стрімко розвиваються і сьогоднішній гаджет – це не тільки засіб зв'язку. Вони проникли в усі сфери нашого життя, і їх роль продовжує зростати, і це здебільшого завдяки додаткам. Набір функцій, які пропонують різноманітні додатки може допомогти нам у плануванні подій, організації життя, зберіганні інформації. І все це в одному телефоні, що значно полегшує правильне управління часом. У повсякденному житті використання мобільних додатків можна спостерігати в таких сферах, як спілкування, освіта, кулінарія, соціальні медіа, покупки, бізнес, шлюб та банківська діяльність.

Раніше людям доводилося зберігати різноманітні записи та нотатки у зошитах та, навіть, просто на паперах бумаги. Наприклад, товсті кулінарні книги, щоб записувати цікаві рецепти, поступово відходять у минуле – їх місце займають додатки для смартфонів. Вони завантажуються і встановлюються самим користувачем через мобільні маркетплейси.

У мобільний додаток можна зайти в будь-який час і практично у будь-якому місці. Він є доступним завжди, без перерв і вихідних. Крім того, присутність додатка на пристрої позбавляє користувачів від витрати часу, пов'язаної з відкриттям браузера і пошуком потрібного сайта.

# 1 ІНФОРМАЦІЙНИЙ ОГЛЯД ТА АНАЛІЗ

## 1.1 Аналіз типів додатків

За технологією розробки існує три основні типи мобільних додатків:

- рідні додатки;
- веб-додатки;
- гібридні додатки.

Розглянемо та проаналізуємо їх більш детально.

Нативні (рідні) програми

Дані мобільні додатки розроблені для певної операційної системи, тому вони є рідними для конкретного пристрою чи платформи. Для розробки використовуються інструменти та мова, які підтримує певна платформа. Відповідно, їх не можна використовувати самостійно на будь-якій іншій ОС, наприклад, додатки для Android не можна використовувати на iOS. Рідні програми можна отримати через спеціальні магазини додатків – Google Play для Android, AppStore для iOS тощо.

Рідні програми розроблюються за допомогою різних мов програмування. Деякі приклади включають: Java, Python, Kotlin, Objective-C, C ++, React та Swift.

Переваги:

- швидкі та надійні;
- інтерактивні та інтуїтивно зрозумілі;
- використовують вбудований інтерфейс пристрою;
- легко взаємодіють з будь-якими функціями телефону.

Недоліки:

- створені для одної платформи;
- мають складнощі при підтримці;
- використовуються складні мови програмування.

Веб-додатки

Веб-додатки поводяться аналогічно рідним програмам, але доступ до них здійснюється через веб-браузер на мобільному пристрої. Вони не є окремими програмами, а працюють на веб-сервері. Веб-програми переспрямовують користувачів до URL-адреси і надалі пропонують параметри встановлення, створивши закладку у браузері.

Веб-додатки розроблюються за допомогою HTML5, CSS, JavaScript, Ruby та подібних мов програмування, які використовуються для роботи в Інтернеті.

Переваги:

- не потребують встановлення на пристрій;
- прості у підтримці;
- доступні для платформи з будь-якою ОС;
- не займають пам’ять на телефоні.

Недоліки:

- залежні від інтернету;
- залежні від браузера;
- мають проблеми з безпекою;
- недоступні у магазинах додатків.

Гібридні додатки

Концепція гібридного додатка - це поєднання нативних додатків та веб-програм. Фактично вони створюються так, щоб виглядати і використовуватися як рідні додатки, але вони обов'язково повинні працювати через компонент, за допомогою якого додатки показують веб-вміст. Гібридний тип передбачає використання єдиної бази коду, яка працює в декількох мобільних операційних системах. Гібридні додатки також доступні у маркетплейсах.

Переваги:

- не обмежені однією платформою;
- не потребують браузера для роботи;
- мають доступ до функцій телефону;
- мають високу функціональність;

- можуть працювати без інтернету.

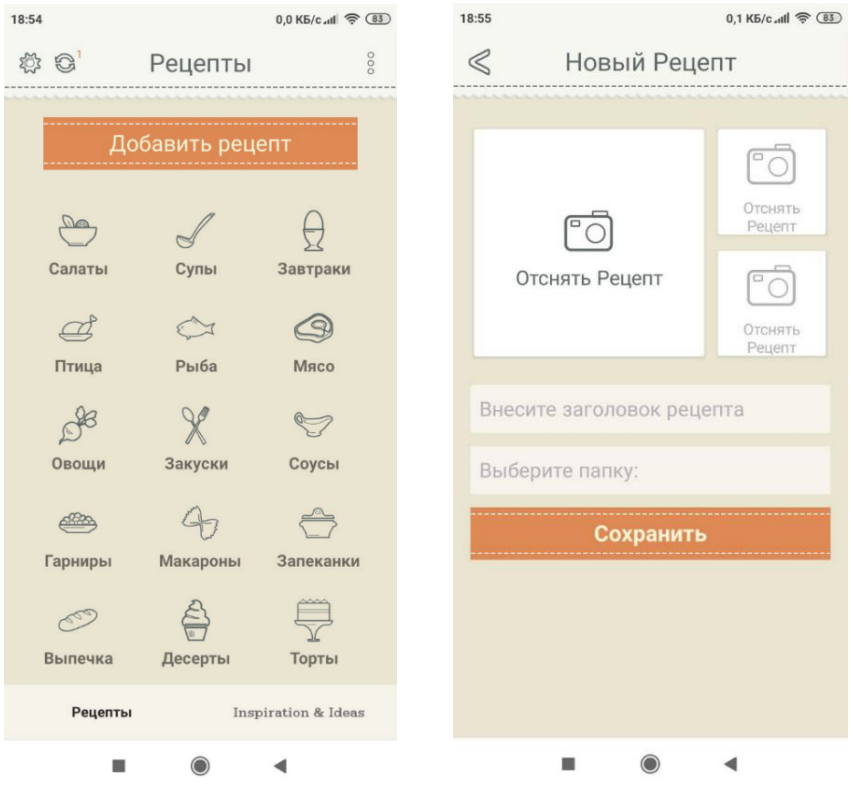
Недоліки:

- не підходять для складного функціоналу;
- можуть працювати злегка повільно;
- менш інтерактивні [1, 2].

### 1.2 Огляд відомих рішень

Розглянемо та проаналізуємо декілька готових мобільних додатків для зберігання рецептів.

OrganizEat – мобільний додаток, який дозволяє зберігати рецепти на пристрої. Має приємний та зрозумілий дизайн. Оформлений у нейтральних кольорах. На головній сторінці є папки із назвами категорій, що дуже зручно, тому що можна одразу перейти до потрібного розділу. Доступний пошук не тільки за назвою рецепту, а й за інгредієнтами (рис.1.1).

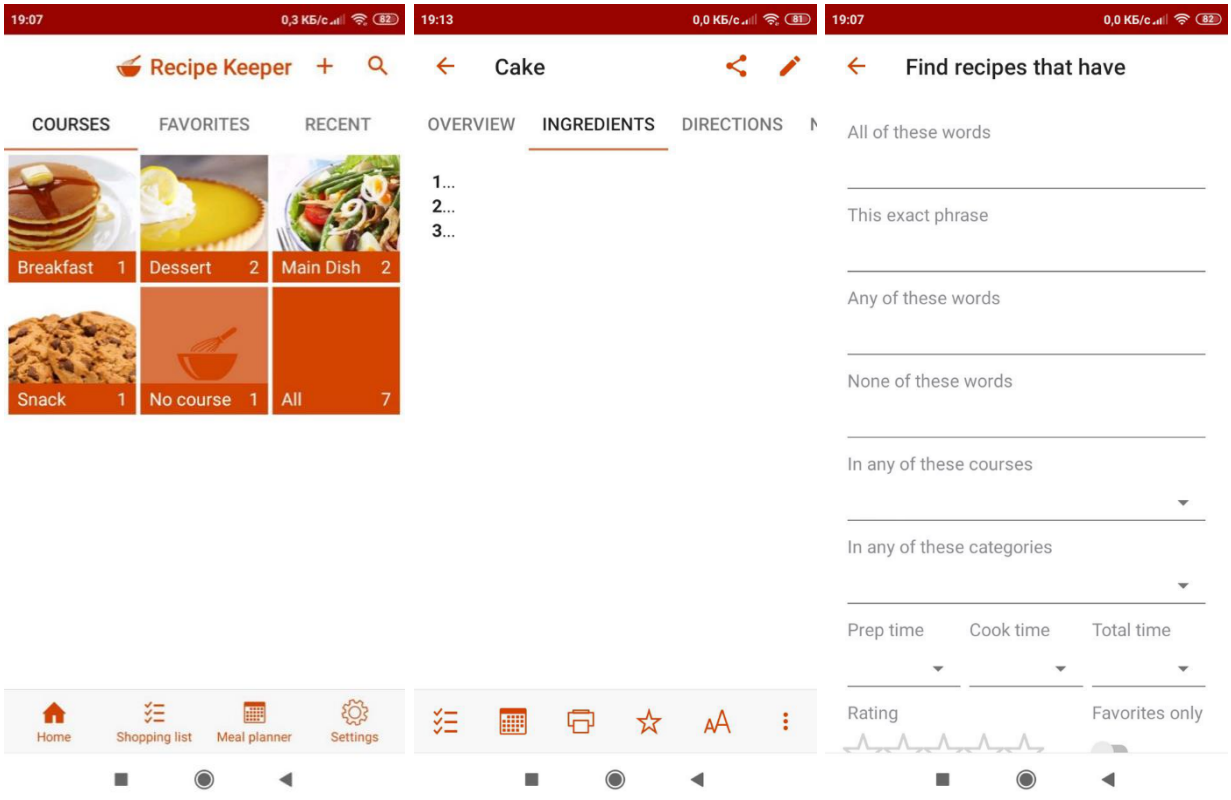



а б

Рисунок 1.1 – Екрани «OrganizEat»: головний екран із рецептами (а), екран створення нового рецепта (б)

Із недоліків можна назвати те, що при створенні рецепта, можливо додати тільки назву, фото та обрати папку-категорію. Щоб наповнити рецепт інформацією, необхідно перейти у потрібну папку, відкрити його та тільки потім змінювати. Ці дії не є зрозумілими на інтуїтивному рівні. Тобто, користувачу потрібно буде витратити час на пошуки щойно створеного рецепта, щоб описати його. Навряд чи, захочется користуватися даним додатком, коли прості функції реалізовані складно і незрозуміло з першого використання.

Recipe Keeper має можливість додавати власні рецепти, імпортувати рецепти з веб-сайтів та опубліковувати фотографії рецептів з книги чи журналу. Має гарний інтерфейс із анімацією. Доступний розширений пошук, що є великою перевагою. Надає широкий функціонал. З першого погляду додаток зацікавлює (рис.1.2).

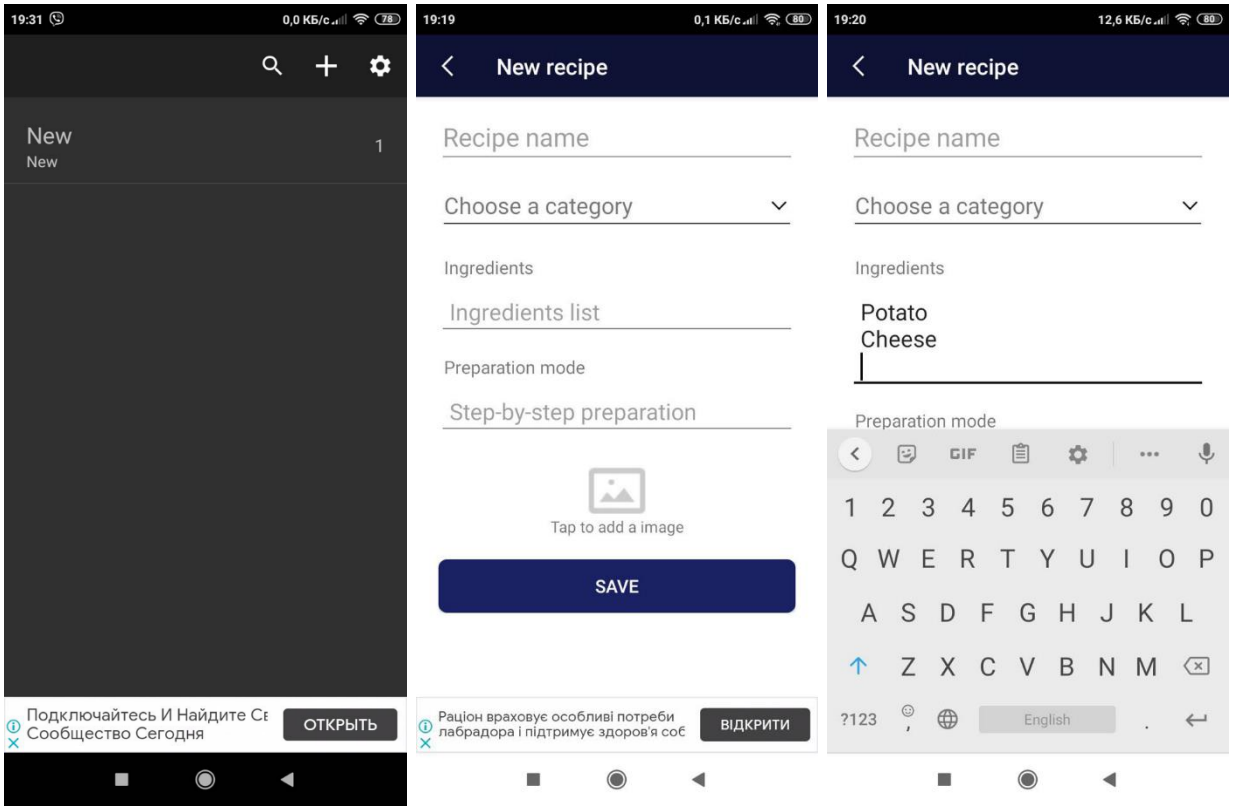



а б в

Рисунок 1.2 – Екрани «Recipe Keeper»: головний екран із рецептами (а), екран створення нового рецепту (б), екран пошуку рецептів (в)

Із мінусів те, що інформація знаходиться у різних вкладинках, що не є зручним. Користувачу потрібно буде постійно переключатися між ними, щоб побачити потрібну інформацію. Також, через те, що додаток багатофункціональний, щоб зробити якусь просту дію, треба витратити час на пошуки відповідної функції. Тобто, не завжди зрозуміло, де і що шукати. Головним мінусом є те, що додаток не працює без інтернету. Якщо користувач не буде мати доступу до мережі і захоче приготувати якусь страву, то це буде неможливо, при тому, що це стосується не тільки пошуку нових рецептів, а й використання своїх власних записаних рецептів.

My Cookbook – додаток для зберігання рецептів. Має простий та зручний дизайн. Без зайвих деталей. Є можливість переключення у темний режим та вибір кольорового оформлення інтерфейсу (рис.1.3).




а б в

Рисунок 1.3 – Екрани «My Cookbook»: головний екран із рецептами (а), екран створення нового рецепта (б), (в)

Серед недоліків можна виділити те, що додаток має дуже мало полів для опису, наприклад, відсутній час приготування, короткий опис. Інгредієнти записуються не кожен окремо, а в одне текстове поле, що не зручно для редагування в подальшому. Немає можливості обрізати фото. У загальному списку на головному екрані рецепти виглядають непривабливо, бо не мають зображення. Тобто, у підсумку, цей додаток не може зацікавити користувача.

### 1.3 Постановка задачі

Проаналізувавши та розібравши різні типи мобільних додатків та розглянувши приклади готових рішень було поставлено мету даної роботи – розробити та програмно реалізувати гібридний мобільний додаток для створення користувацького кулінарного довідника.

Додаток повинен вміти працювати без підключення до інтернету та виконувати наступні задачі:

- створення рецептів;
- редагування та видалення рецептів;
- пошук та сортування рецептів;
- можливість поділитися рецептом через соц. мережі та через телефон.

Для досягнення поставленої мети необхідно виконати такі завдання:

1) Обрати фреймворк та мову програмування для розробки.
2) Підібрати та спроектувати базу даних.
3) Створити прототип мобільного додатка.
4) Програмно реалізувати мобільний додаток.
5) Виконати тестування готового продукту.

# 2 ВИБІР МЕТОДУ ВИРІШЕННЯ ЗАДАЧІ

## 2.1 Аналіз гібридних фреймворків

У першому розділі був зроблений огляд та аналіз технологій розробки мобільних додатків. На основі отриманої інформації щодо переваг та недоліків кожного типу, для поставленої мети було обрано гібридний тип розробки, який якнайкраще підходить для реалізаціїї простого функціоналу. Він не обмежений однією платформою, що дозволить запускати додаток на різних операційних системах та підтримує роботу додатка без інтернету.

Перейдемо до вибору гібридних фреймворків для надійної та потужної розробки мобільного додатка.

Гібридні фреймворки – це набір класів і бібліотек, який використовують для створення динамічних додатків для різних мобільних пристроїв.

Розглянемо декілька найбільш популярних фреймворків:

Apache Cordova – це програма для розробки мобільних додатків з відкритим кодом. Це означає, що для розробки платформ можна використовувати стандартні веб-технології, а саме HTML5, CSS3 та JavaScript. Програми виконуються в обгортках, орієнтованих окремо на кожну платформу, і розраховують на відповідність стандартам прив'язки API для доступу до можливостей кожного пристрою, таких як датчики, дані, стан мережі тощо.

Ось деякі приклади додатків, розроблених на Apache Cordova: Hockey Community, Healthtap, TripCase, My Heart Camera та інші.

Переваги:

- можна використовувати для створення міжплатформних мобільних додатків;
- проста у вивченні;
- має доступ до рідного функціоналу пристрою.

Недоліки:

- погана документація;
- не має власних фреймворків для створення користувацького інтерфейсу;
- проблеми з плагінами [3, 5].

Ionic – це платформа, яка заснована на Apache Cordova та Angular, дозволяє створювати повністю функціональні та просунуті мобільні додатки з використанням веб-технологій. Дана платформа заснована на CSS і досягає свого максимального потенціалу при спільному використанні з AngularJS(фреймворк JavaScript), яка надає платформі елементи інтерфейсу через бібліотеку вбудованих компонентів для iOS та Android. Іншими словами, Ionic - це поєднання кількох технологій, які працюють разом, щоб зробити мобільні додатки швидшими та простішими. Верхнім шаром цього стека є сам Ionic Framework, що забезпечує рівень користувацького інтерфейсу програми, якраз під ним знаходиться Angular. І вже потім ці фреймворки розташовуються на вершині Apache Cordova, що дозволяє веб-додатку використовувати власні можливості пристрою і перетворитись в нативну програму.

Бібліотека пропонує компоненти, які поводяться і виглядають як рідні елементи всіх підтримуваних платформ.

Відомі додатки, що розроблені на основі Ionic: McDonald’s Türkiye, ChefSteps, Sworkit, Justwatch та інші.

Переваги:

- єдина база коду для різних платформ;
- простий у вивченні;
- надає широкий спектр можливостей інтеграції та плагінів;
- має великий вибір елементів інтерфейсу;
- доступна і детальна документація.

Недоліки:

- слабка продуктивність;

- плагін-залежна система;
- можливі проблем з безпекою;
- великий розмір програми [3, 6].

Xamarin – фреймворк, написаний на мові C # з використанням .NET – платформа з відкритим кодом, яка була створена корпорацією Майкрософт для розробки достатньо великої кількості різних типів додатків. Дана платформа надає можливість використання декількох мов, бібліотек та редакторів для створення програм для великої кількості різних платформ з єдиною кодовою базою. У Xamarin реалізовані компоненти виклику нативних елементів, таким чином скомпільований додаток працює по суті як рідний. Він також містить окремі бібліотеки для C ++, Java та Objective C, які роблять Xamarin унікальним серед усіх.

Перелік додатків, розроблених за допомогою Xamarin: CA Mobile, Picturex, Fareboom, ORO та інші.

Переваги:

- має якісну документацію;
- має підтримку Microsoft;
- має високу продуктивність;
- має доступ до системних програм пристрою.

Недоліки:

- має обмежений доступ до бібліотек з відкритим кодом;
- займає багато пам’яті на пристрої;
- має проблеми сумісності з сторонніми бібліотеками;
- багато небезкоштовних інструментів [3, 7, 11].

React Native надає можливість створення мобільних додатків для декількох платформ, використовуючи лише одну мову програмування – JavaScript. Додаток будується з компонентів платформи iOS чи Android – це нативні модулі, загорнуті в React-компоненти. Іншими словами React Native здатний приймати вбудовані компоненти платформи, такі як основні панелі, перемикачі, полоси

прокрутки та загортати їх у аналоги компонентів React. Саме тому немає ніяких проблем із вкладинками і прокруткою екрану, а інтерфейс поводиться і виглядає так само, як і в рідному додатку.

Приклади відомих додатків, розроблених з використанням фреймворку React Native: Instagram, Facebook Messenger, Snapchat, Wix, UberEATS тощо.

Переваги:

- має багато готових до використання компонентів для прискорення процесу розробки;
- має функцію перезавантаження у реальному часі;
- забезпечує автоматичне виявлення помилок;
- має загальну кодову базу;
- має хорошу продуктивність;
- має якісну документацію.

Недоліки:

- не вистачає деяких компонентів;
- не підходить для складних проектів [3, 4].

На основі проведеного аналізу популярних фреймворків для гібридної розробки, визначення сильних та слабких сторін кожної з них, для роботи було обрано фреймворк React Native. Враховуючи можливості, які надає даний фремворк, створені додатки автоматично стають кросплатформенними, без зайвих зусиль та витрати часу на розробку окремої версії для іншої ОС.

### 2.2 Аналіз та огляд баз даних

Після визначення технології та інструментів для розробки додатку, необхідно перейти до вибору бази даних.

Спосіб управління даними відіграє важливу роль у наданні позитивної роботи користувачу. Програма повинна вміти швидко отримувати, обробляти та надавати необхідну інформацію. Крім того, всі ці дані повинні бути захищені.

Цього можна досягти за допомогою розумно вибраної системи управління базами даних.

База даних – це організований набір структурованої інформації або даних, зберігається в електронному вигляді в комп'ютерній системі. База даних зазвичай контролюється системою управління базами даних (СУБД).

Система управління базами даних (СУБД) – це програмний комплекс, який відповідає за функціонування бази даних. Він призначений для визначення, маніпулювання, отримання та управління даними в базі даних. СУБД, як правило, маніпулює самими даними, форматом даних, іменами полів, структурою запису та структурою файлів. Вона також визначає правила перевірки та маніпулювання цими даними [9].

Розглянемо класифікацію СУБД.

За моделлю даних:

1. Ієрархічні – модель бази даних, яка впорядковує дані в деревоподібну структуру з одним коренем, до якого прив'язані всі інші дані. Ієрархія починається від даних Root і розширюється як дерево, додаючи дочірні вузли до батьківських вузлів. У цій моделі одному дочірньому вузлу відповідає лише один батьківський вузол. Серед основних операцій маніпулювання даними можна виділити перехід від одного дерева до другого, перехід від одного запису до іншого в межах одного дерева, пошук запису, вставка запису , видалення поточного запису та інше.
2. Мережеві – модель даних, що є розширенням ієрархічного підходу. Дані організовані більше як графік і дозволяють мати більше одного батьківського вузла. У цій моделі бази даних встановлено багато взаємозв'язків, тому її доцільно використовувати для відображення співвідношень даних багато-до-багатьох. Основні операції маніпулювання даними: створення нового запису, пошук, редагування, видалення поточного запису, перехід від батька до першого дочірнього вузла, перехід

від дочірнього вузла до іншого дочірнього вузла та перехід від дочірнього вузла до батька.

3. Реляційні – модель бази даних, яка впорядковує дані в двовимірних таблицях, а взаємозв'язок між ними підтримується шляхом зберігання загального поля. Основна структура даних у реляційній моделі - таблиці. У таблиці кожен стовбчик має атрибут, кожен рядок містить дані про один тип об’єкту або усю інформацію про зв’язок між декількома об’єктами. Таблиця та стовбчик обов’язково мають унікальне ім’я, а рядки ніяк не називаються.
4. Модель відносин особи – модель даних, де відносини створюються шляхом поділу об'єкта на сутність, а його характеристики на атрибути. Різні сутності пов'язані з використанням відносин.

За ступенем розподіленості:

1. Централізовані (локальні) – база даних зберігається локально, це означає, що всі дані бази знаходяться в одному місці. Під терміном "єдине розташування" мається на увазі єдиний центральний комп'ютер. Щодо доступу до локальної бази даних, це можна зробити з кількох різних точок, як правило, через комп'ютерну мережу.
2. Розподілені – сукупність безлічі логічно взаємопов'язаних баз даних, розподілених по комп'ютерній мережі. Керується системою управління розподіленою базою даних. Ці дані знаходяться на декількох машинах, а не на одній.

За способом доступу до БД:

1. Файл-серверні – база даних, яка знаходиться на файловому сервері. Вона підключена до декількох робочих станцій по мережі. СУБД та програми працюють на кожній робочій станції (клієнтському комп’ютері).
2. Клієнт-серверні – це коли у середовищі файлового сервера обробка даних розподіляється по мережі, як правило, по локальній мережі (LAN).

3. Вбудована база даних – це технологія бази даних, в якій рішення для управління базами даних вбудовуються в додаток, а не надаються як окремі інструменти. Дані зберігаються локально, в процесі або в пам'яті в межах однієї машини, не передаючи інформацію по мережі. Вбудована база даних включає менше функцій, ніж повноцінна система управління базами даних, що встановлена окремо, саме тому вона більш компактна та ефективна [8, 12].

Розглянувши класифікацію баз даних за різними ознаками, проаналізувавши дану інформацію, для розроблюваного додатка підходить вбудована реляційна локальна база даних. Вона досить проста та незалежна від даних. Обрана система управління базами даних цілком задовольняє вимогу можливості використання додатка без інтернету. Таким чином, дані будуть зберігатися на пристрої і їх можна буде змінити у будь-який момент.

Розглянемо декілька вбудованих систем управління базами даних та оберемо найбільш підходящу для проекту.

Firebase підтримує базу даних No-SQL у режимі реального часу. Вона широко використовується у послугах з розробки додатків на React Native. Ця база даних дозволяє змінювати дані в режимі офлайн та чудово управляється з синхронізацією даних. Також, вона надає міжплатформенний API, який вимагає мінімальної настройки при використанні з додатком, до того ж не потрібен сервер додатків для доступу до даних, оскільки їх можна отримати безпосередньо з мобільного пристрою. Цілком безпечна, але доцільно не використовувати Firebase як базу даних для додатків, які потребують обробки конфіденційних даних, тому що є загроза безпеки.

SQLite була розроблена для забезпечення локального зберігання даних у мобільних додатках. Головною перевагою цієї системи управління базами даних є те, що її можна використовувати безпосередньо в мобільному додатку для прямого доступу до бази даних. Вона має просту бібліотеку, а також вимагає невеликих зусиль для її налаштування. Архітектура бібліотеки проста і всі типи

даних можна обробляти легко. Щоб зробити керуючу сторону простою, SQLite використовує один файл для зберігання всіх даних.

PouchDB дозволяє програмам локально зберігати дані в режимі офлайн, а потім синхронізувати їх із CouchDB та сумісними серверами, коли програма знову в мережі, зберігаючи дані користувачів у синхронізації. Може працювати як власний веб-браузер. PouchDB надає можливість створювати додатки, які можуть працювати як з інтернетом, так і без. Вона безкоштовна, з відкритим кодом. Також ця база даних є досить безпечною [10].

На основі огляду та аналізу СУБД, для роботи було обрано SQLite. Окрім усіх описаних вище переваг цієї СУБД, необхідно додати, що вона ACID-сумісна, тобто реалізує більшість стандартів, заснованих на SQL. Дана база даних чудово взаємодіє з React Native, наприклад, для забезпечення офлайн-зберігання даних за допомогою SQLite, можна використовувати спеціальний плагін для управління даними в додатку. В довершення необхідно зазначити, що ця база даних є загальнодоступною, з відкритим кодом та безкоштовною у користуванні.

# 3 ПОГРАМНА РЕАЛІЗАЦІЯ

## 3.1 Проектування бази даних

Розглянемо схему бази даних. На ній зображені основні таблиці для додатка, в яких буде зберігатися інформація про рецепти – це таблиці recipes, ingredients та tags. Також є дві додаткові окремі таблиці, які утворюються самостійно при створенні бази даних – sqlite_master та sqlite_sequence (рис.3.1).

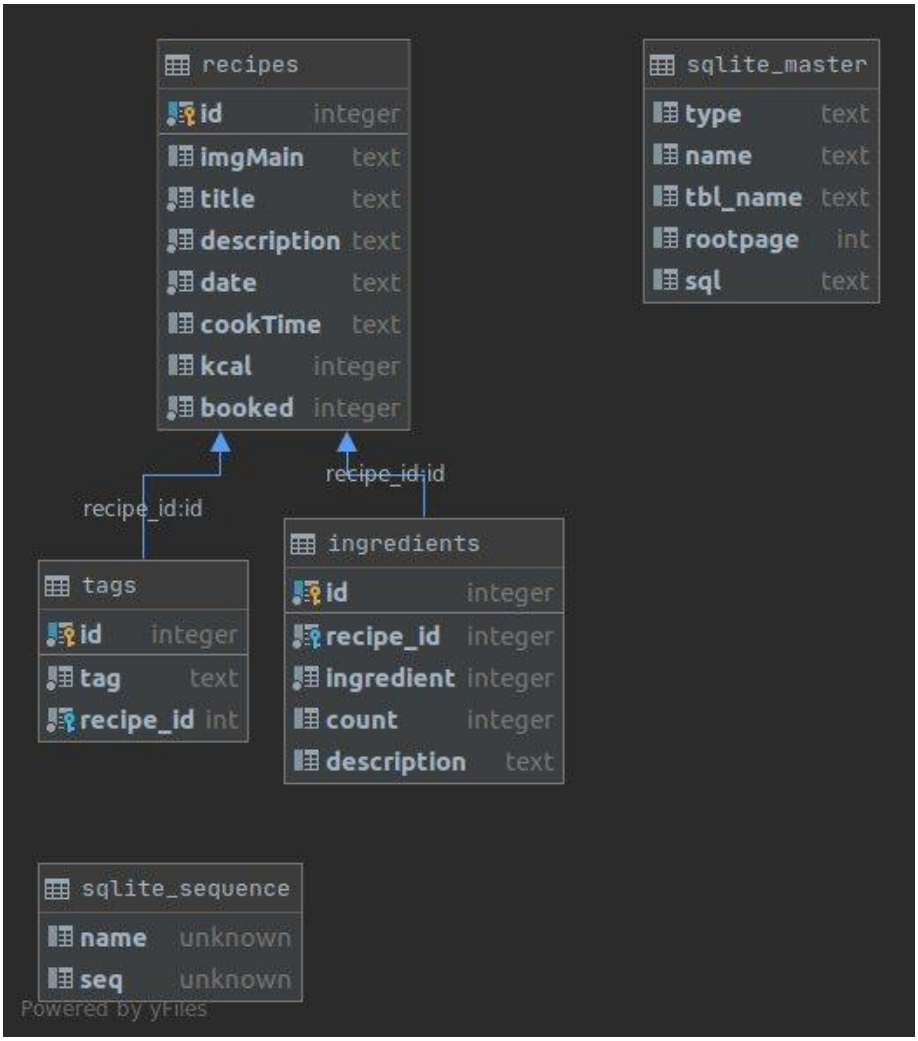



Рисунок 3.1 – ER-діаграма бази даних

В таблиці «recipes» представлена уся інформація про рецепти. У таблиці «ingredients» знаходиться інформація про інгредієнти. Таблиця «tags» містить інформацію про теги. На даний момент ця таблиця створена, але не реалізована. Вона потрібна для того, щоб до кожного рецепта можна було додати один або декілька тегів. Завдяки чому, можна буде швидко та легко знайти необхідні

страви, правильно вказавши ключові слова(теги). При пошуку за тегом будуть показуватися усі рецепти, які його мають. «Sqlite_sequence» – внутрішня таблиця, необхідна для реалізації AUTOINCREMENT – ключове слово, яке використовується, щоб не допустити повторного використання SQLite значення, яке не було використано, або значення з раніше видаленого рядка. Для кожної користувацької таблиці, що використовує інкремент, відповідає рядок таблиці «sqlite_sequence». Перша сторінка файлу бази даних – це коренева сторінка «table B - tree», що містить спеціальну таблицю «sqlite_master» (табл.3.1).

Таблиця 3.1 – Структура бази даних

| **Таблиця** | **Поле** | **Зміст** | **Тип** | **Ключі** |
|---|---|---|---|---|
| recipes | id | Ідентифікатор рецепту | integer | PK |
| | imgMain | Зображення рецепту | text | |
| | title | Назва рецепту | text | |
| | description | Опис рецепту | text | |
| | date | Дата створення | text | |
| | cookTime | Час приготування | text | |
| | kcal | Кількість kcal | integer | |
| | booked | Рецепт, який додано у вибране | integer | |
| tags | id | Ідентифікатор тегу | integer | PK |
| | tag | Назва тегу | text | |
| | recipe_id | Ідентифікатор рецепту | text | PFK |
| ingredients | id | Ідентифікатор інгрідієнту | integer | PK |
| | recipe_id | Ідентифікатор рецепту | integer | PFK |
| | ingredient | Назва інгрідієнту | integer | |
| | count | Кількість | integer | |
| | description | Опис інгрідієнту | text | |

| | | | | |
|---|---|---|---|---|
| sqlite_sequence | name | Ім'я таблиці, що використовує інкремент | – | |
| | seq | Значення інкремента | – | |
| sqlite_master | type | Вказується в залежності від типу об’єкту: table, index, view, trigger. | text | |
| | name | Ім’я об’єкту | text | |
| | tbl_name | Ім’я таблиці | text | |
| | rootpage | Номер кореневої сторінки для таблиць та індексів | int | |
| | sql | Sql запрос, який описує об’єкт | text | |

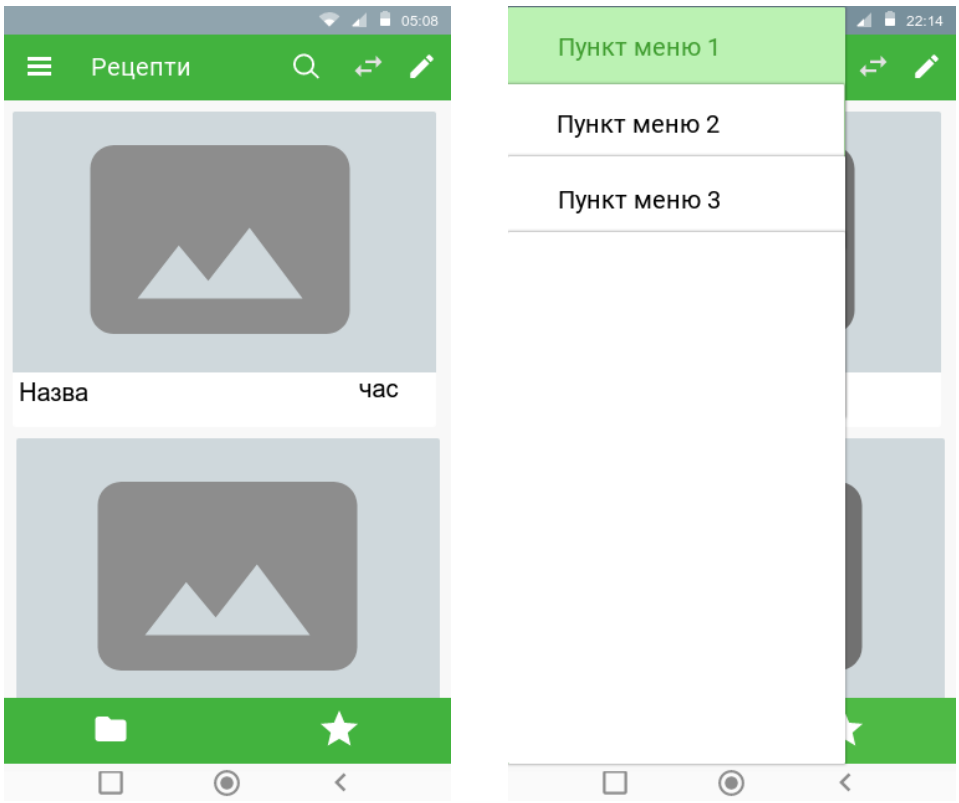
### 3.2 Створення прототипу мобільного додатка

Перед тим, як починати розробку будь-якого програмного продукту, необхідно приділити увагу вибору його оформлення та дизайну. Для цього потрібно ретельно продумати кожну деталь та перебрати безліч варіантів оформлення інтерфейсу. Зібрати усі думки та ідеї до купи допоможе створення прототипу програмного продукту. Простими словами, прототип – це візуалізація того, як програма буде виглядати та функціонувати. Він може бути кольоровим або чорно-білим, другий варіант використовують, коли хочуть зосередитись на розумному розташуванні елементів та зручності користування додатком, в цілому.

Узагалі, створення прототипів є дуже важливим етапом у розробці мобільних додатків. Перш за все, це зручно для розробників, тому що вони мають макети дизайну, що сприяє швидкому розумінню вимог та запобігає витраті часу на пошуки відповідей та пояснень. Створення макету додатка допомагає

побачити, яким буде кінцевий продукт. Часто трапляється так, що початкове уявлення про вигляд програми, відрізняється від реалізованого, і все це тому що, ідея не була візуалізувана у вигляді прототипу. Зазвичай, це помічають уже в кінці розробки, і щоб виправити ситуацію необхідно починати спочатку, що займає велику кількість часу. Також, у ході створення прототипу можливі якісь правки, які зможуть вдосконалити мобільний додаток. Має сенс перед тим, як реалізовувати новий функціонал, зобразити його на макеті, це надає цілісну картину того, як новий елемент виглядає разом з іншими кнопками та іконками. Прототипування допомагає визначити правильне розташування усіх елементів, щоб додатком було зручно користуватися, підібрати кольори, які будуть підходити до тематики розроблюваного продукту та обрати найбільш підходящі та приємні шрифти.

Отже, визначивши необхідність та важливість етапу візуалізації ідеї перед початком розробки, було створено прототип усіх головних екранів мобільного кулінарного додатка, що значно полегшить процес розробки. Створений макет не є повністю деталізованим, він служить в якості надання загального враження того, як повинен виглядати додаток (рис. 3.2–3.4).




а б

Рисунок 3.2 – Прототип екрана: з рецептами (а), з відкритим боковим меню (б)

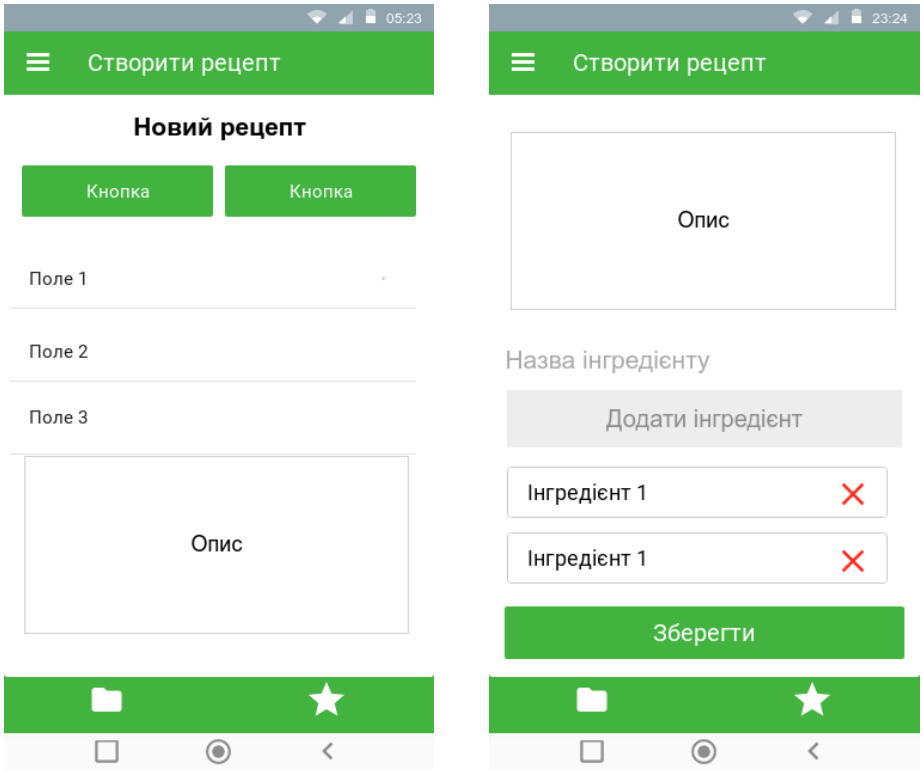
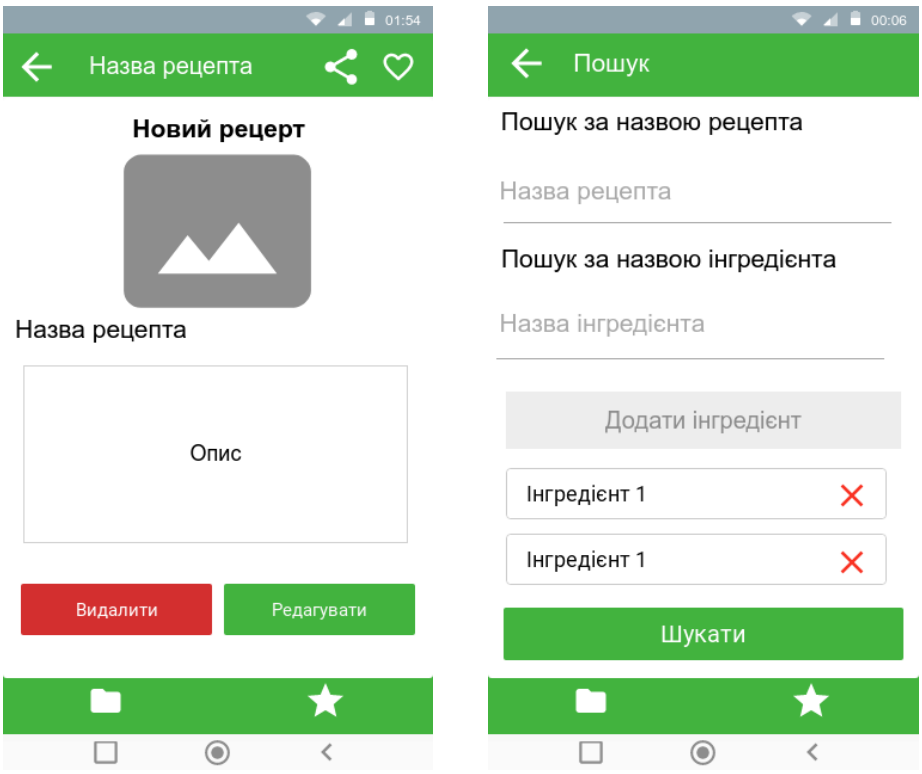



Рисунок 3.3 – Прототип екрана створення нового рецепта




а б

Рисунок 3.4 – Прототип екрана: редагування рецепта (а), пошуку рецепта (б)

### 3.3 Короткий опис програмної реалізації

Основною платформою для розробки кросплатформи було обрано і встановлено Node.js – серверна платформа, побудована на JavaScript для розробки серверних та мережевих додатків. Вона є відкритим джерелом, що значно допомагає при розробці. Окрім фреймворку React Native, який було розглянуто у розділі 1.1, у роботі було використано ще один допоміжний фреймоворк – Expo. Він пропонує багато компонентів, а бібліотеки вже включені, щоб полегшити розробку мобільних додатків за допомогою React Native. Також серед переваг можна виділити те, що даний фреймворк простий у використанні, має якісну зрозумілу документацію, є можливість тестувати програму безпосередньо на пристрої.

Отже, перед тим, як перейти до розробки додатка, встановимо Expo, написавши у командному рядку – `install -g expo-cli`. Перевіривши, що

установка пройшла успішно, пишемо наступну команду – `expo init cookbook`, щоб створити новий проект, де «cookbook» – назва проекту. Далі введемо `cd cookbook`, щоб перейти до папки проекту та `npm start`, щоб запустити локальний сервер розробки Expo CLI. Перейдемо до програмної реалізації.

У якості середовища для розробки служить програма Visual Studio Code – зручна та потужна для редагування вихідного коду. Даний редактор працює на робочому столі та доступний для завантаження на macOS, Windows та Linux. Він оснащений вбудованою підтримкою JavaScript, TypeScript і Node.js.

Відкриємо наш проект «cookbook» у Visual Studio Code. На рисунку можна побачити код файла app.json, який містить усю інформацію про додаток (рис. 3.5).

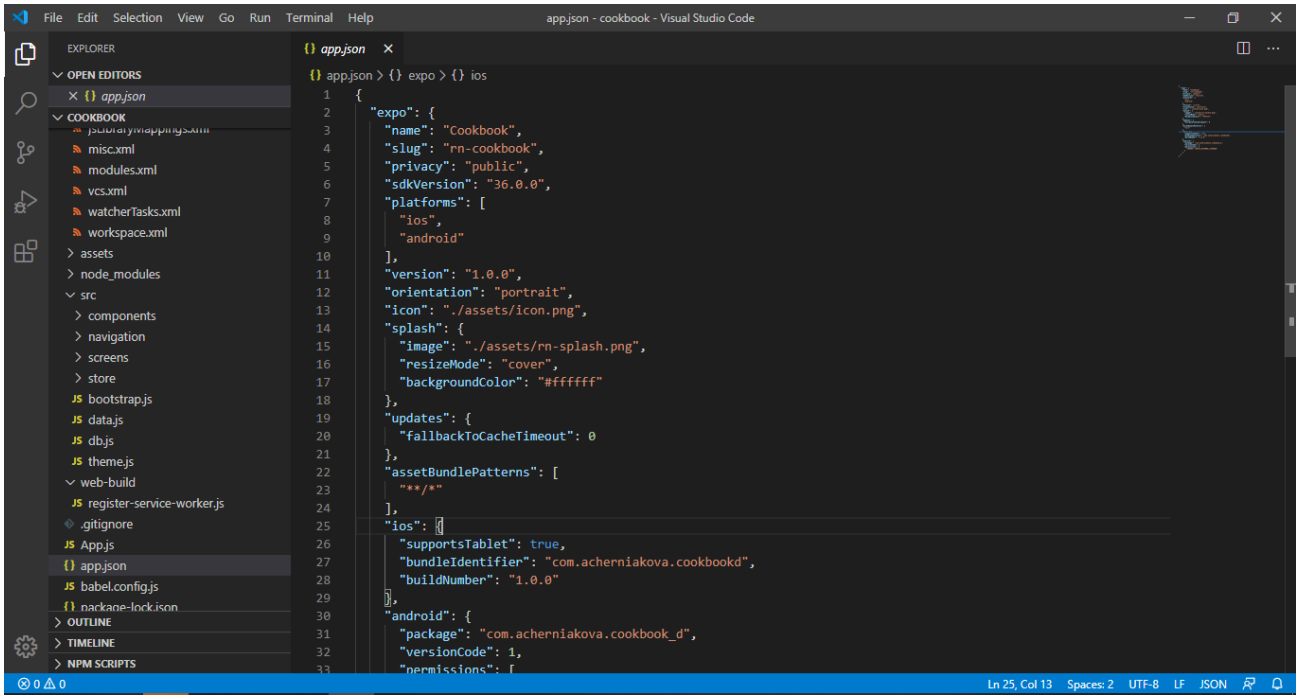



Рисунок 3.5 – Код з усією інформацією про додаток у Visual Studio Code

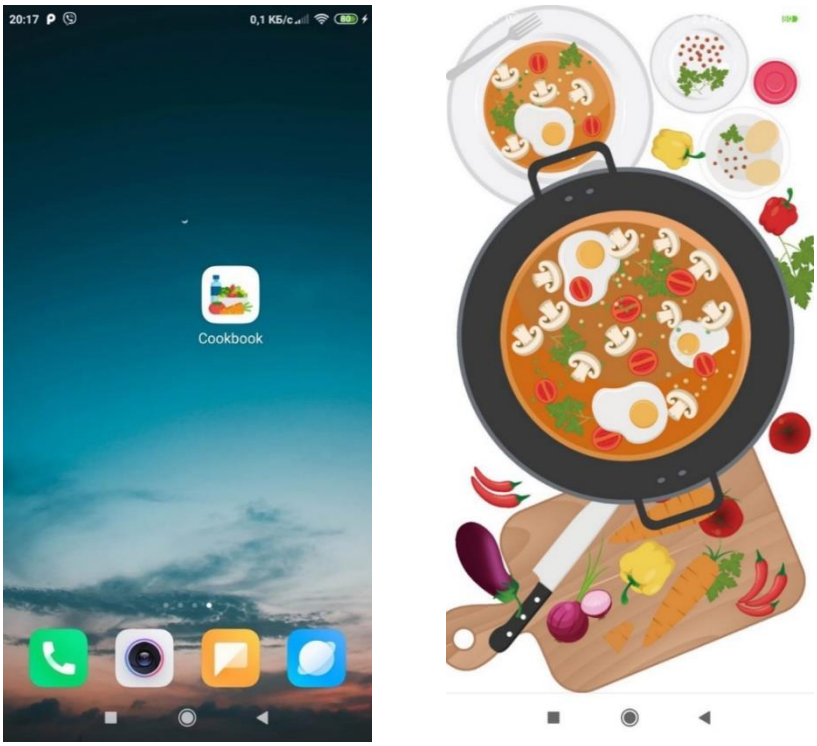
### 3.3 Тестування готового додатка

Щоб почати користуватися готовим додатком на своєму пристрої, необхідно зібрати весь проект у інсталяційний файл з розширенням .apk за допомогою команди `expo build:android`, потім завантажити його на мобільний телефон та запустити.

Для перевірки функціонування та працездатності, додаток було встановлено та протестовано на двох пристроях: Samsung Galaxy A71 (android 10), Xiaomi Redmi 7A (android 9). Результати тестування на різних телефонах наведено нижче (рис. 3.6–3.15).

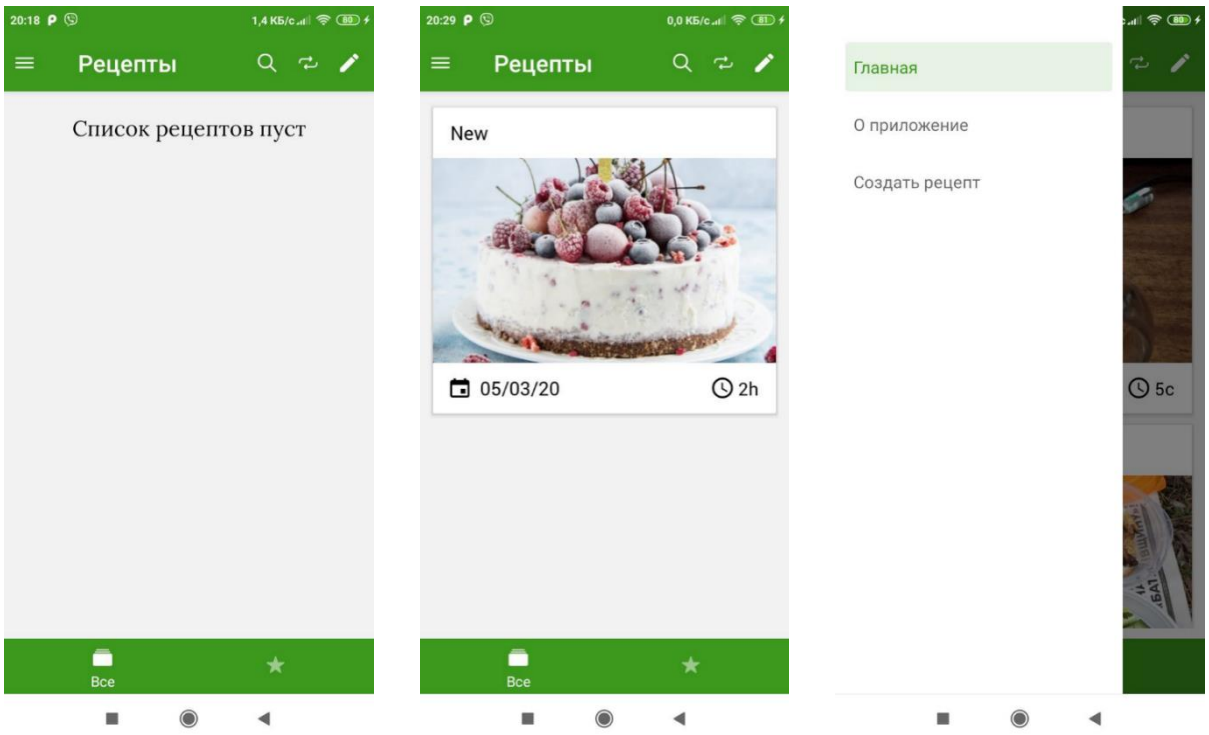
Коли додаток запускається, на екрані з’являється зображення з їжею (рис.3.6).

На головному екрані знаходяться рецепти. Користувач бачить назву, зображення, дату та час приготування. З головного екрану є можливість перейти на екран «Вибране». Також можна відкрити бокове меню (рис. 3.7).




а б

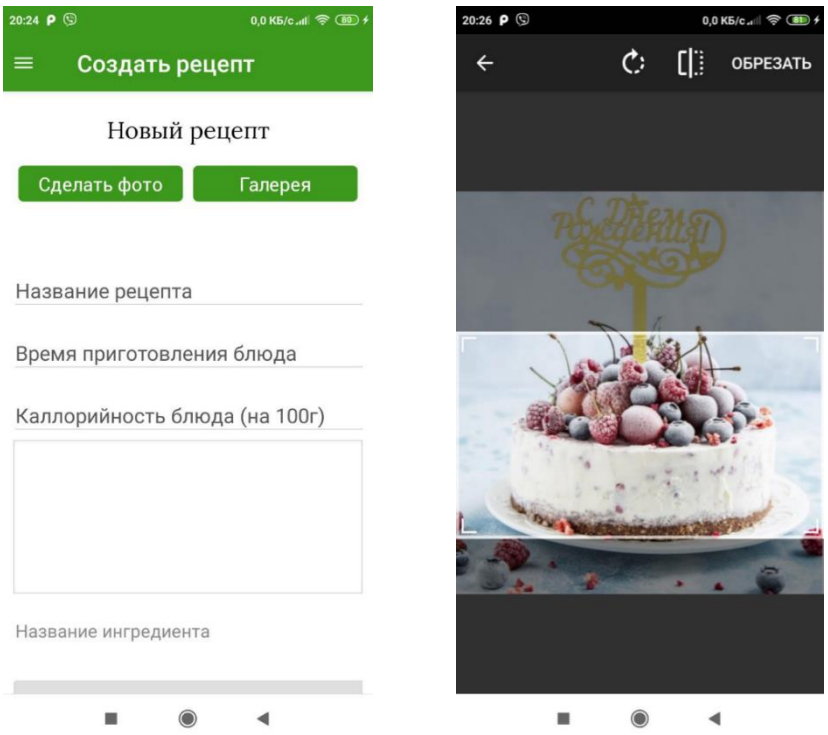
Рисунок 3.6 – Фото з пристрою Xiaomi: Іконка мобільного додатка (а), зображення під час запуску (б)




а б в

Рисунок 3.7 – Фото з пристрою Xiaomi: Головний екран без рецептів (а), головний екран із рецептами (б), праве бокове меню (в)

Додати рецепт можна двома способами, із правого бокового меню, натиснувши на кнопку «Создать рецепт» або, натиснувши на «олівець» на головному екрані. Під час створення рецепту, можна додати готове фото з галереї або зробити фото за допомогою телефону. Обидва варіанти фото можна редагувати. Функції редагування виглядають так само як рідні функції на пристрої, доступний поворот, функція віддзеркалення та обрізка зображення (рис. 3.8).

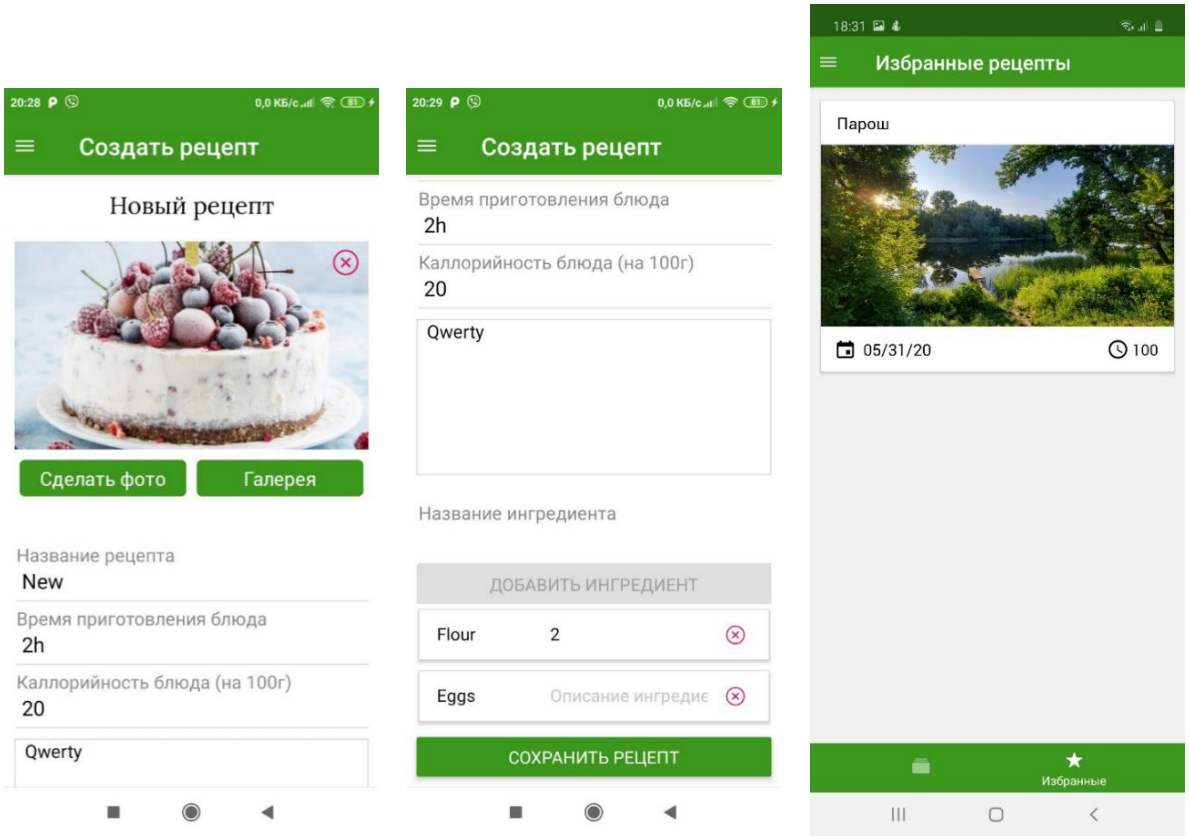



а б

Рисунок 3.8 – Фото з пристрою Xiaomi: екран створення рецепту (а), екран редагування фото (б)

Можна написати назву, час приготування, калорійність, інструкцію по приготуванню, додати інгредієнти та опис до кожного з них (рис. 3.9 а, б).

Будь-який рецепт можна додати у вибране, натиснувши на «серце» тобто виділити якісь улюблені страви. Користувач у будь-який момент може відкрити та приготувати свій улюблений рецепт, не витрачаючи час на його пошуки, а просто натиснувши на «Избранные» (рис. 3.9 в).




а б в

Рисунок 3.9 – Фото з пристроїв: Xiaomi – екрани створення рецепта з заповненими полями (а, б), Samsung – екран із улюбленими рецептами (в)

Рецепт можна змінити та видалити. Для редагування доступні абсолютно всі поля, в тому числі зображення рецепту (рис. 3.10).

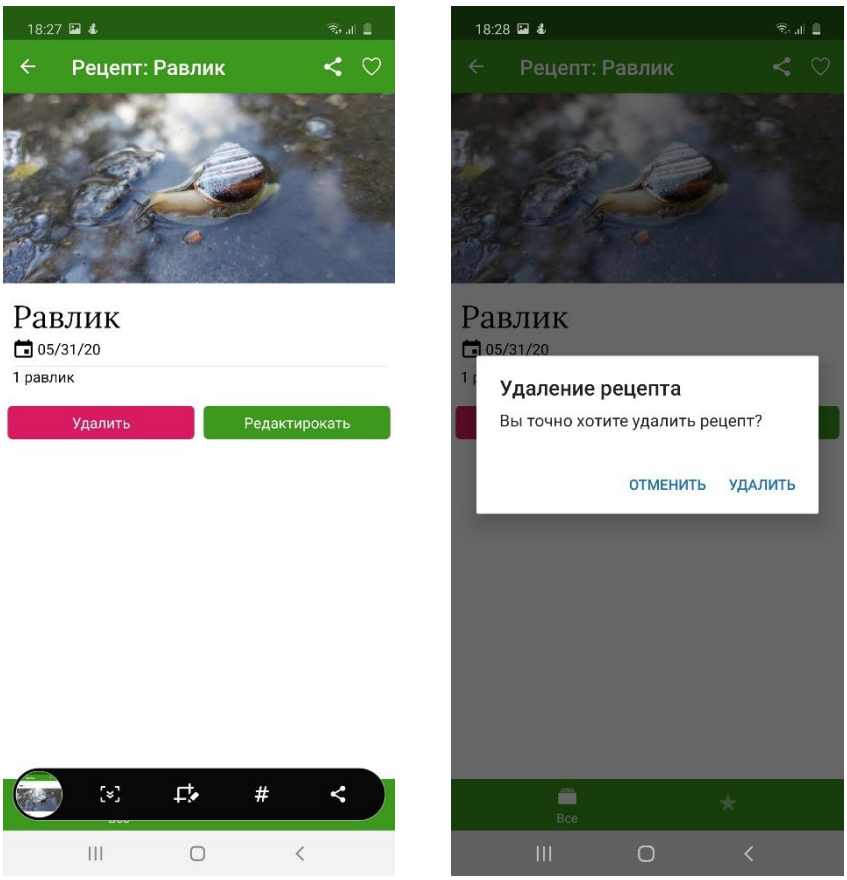



Рисунок 3.10 – Фото з пристрою Samsung: екран редагування рецепта

Також доступна можливість сортувати рецепти за датою, натиснувши на іконку сортування на головному екрані. За замовчуванням стоїть сортування від найбільш нової дати до найбільш старої (рис.3.11).

З головного екрану можна переключитись на екран пошуку, натиснувши на «лупу». Доступний пошук по назві рецепту та по інгредієнтах або по назві і інгредієнтах одночасно. Якщо є результати пошуку, то додаток перенаправляє користувача на екран із знайденими рецептами, з якого є можливість повернутися назад до пошуку. Якщо результатів, задовольняючих пошук немає, то користувач побачить повідомлення про це на екрані (рис.3.12).

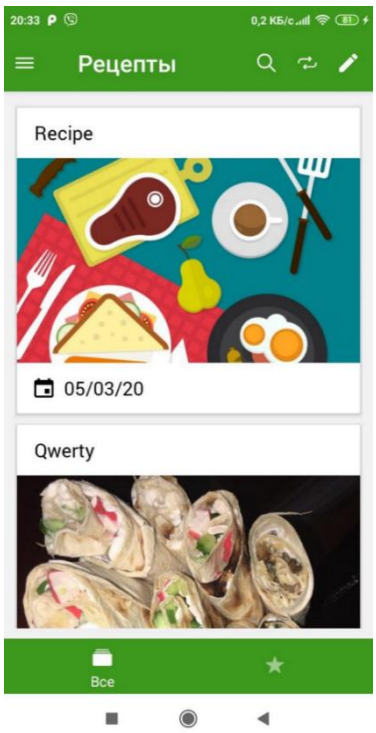
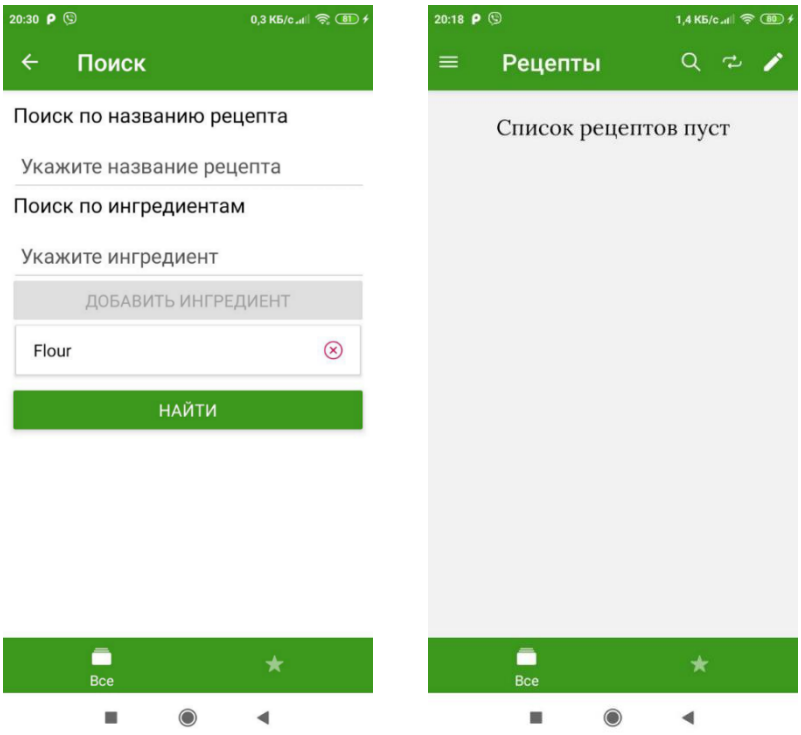


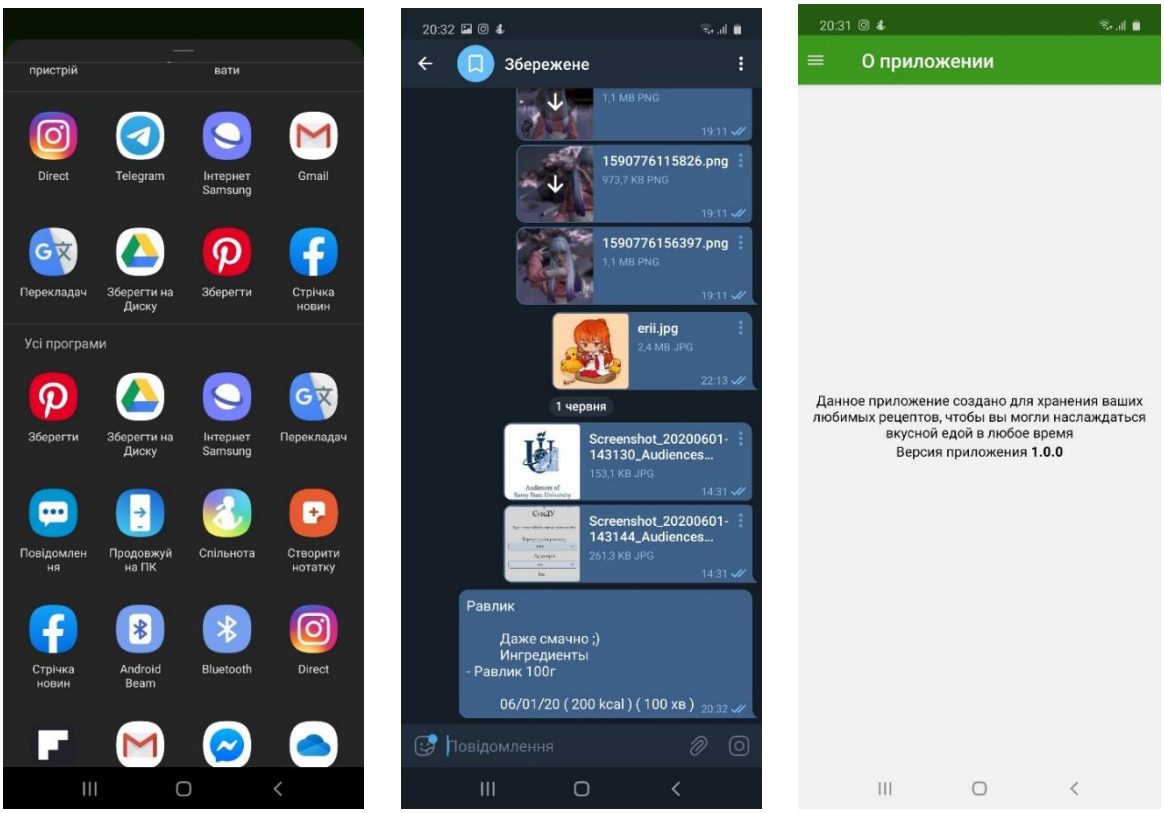

Рисунок 3.11 – Фото з пристрою Xiaomi: рецепти, відсортовані за датою




а б

Рисунок 3.12 – Фото з пристрою Xiaomi: Екран пошуку рецептів (а), екран із відсутніми результатами пошуку (б)

Рецептом можна поділитися через соц.мережі або через телефон(наприклад, через смс). Іншими словами, використовуються стандартні функції телефону і їх вигляд зберігається. Рецепт, яким поділилися, відображається у вигляді тексту, що включає назву, опис, інгредієнти та їх кількість, дату створеня або останнього оновлення рецепта, кількість кілокалорій та час приготування. (рис. 3.13 а, б). Також, можна прочитати про додаток на екрані «О приложении», перейшовши на нього з правого бокового меню. Даний екран містить інформацію про призначення додатка та вказана його поточна версія (рис. 3.13 в).




а б в

Рисунок 3.13 – Фото з пристрою Samsung: екран із функцією «Поділитися» (а), вигляд рецепту, яким поділилися (б), екран із інформацією про додаток (в)

# ВИСНОВКИ

У випускній роботі у першому розділі було розглянуто технології розробки мобільних додатків та приклади готових рішень, на основі чого було сформовано постановку задачі. Також, було проведено аналіз фреймворків для розробки, розглянуто особливості їх застосування.

У другому розділі було розглянуто класифікацію баз даних, після чого обрано певний тип бази даних, проведено огляд та аналіз існуючих систем управління базою даних, виявлено їх слабкі та сильні сторони, розглянуто особливості їх застосування. На основі аналізу було обрано та спроектовано базу даних для проекту.

У третьому розділі було створено прототип мобільного додатка, зроблено короткий опис програмної реалізації, де було вказано обрану платформу та середовище для розробки, розписано основні етапи підготовки інструментів та програми для початку роботи.

Результатом виконання роботи є розроблений мобільний додаток для створенная користувацького кулінарного довідника, працездатність якого було успішно протестовано на двох пристроях. Додаток є надійним, простим та зручним у використанні, має приємний зрозумілий інтерфейс. Також він може працювати як в режимі онлайн, так і в режимі офлайн. Додаток реалізує наступні функції:

- створення рецептів;
- редагування та видалення рецептів;
- пошук та сортування рецептів;
- можливість поділитися рецептом через соц. мережі та через телефон.

У роботі розміщено фото функціонуючого додатку, які було зроблено на пристрої з ОС Android, але так як додаток є кросплатформенним, його можна легко зібрати і для ОС iOS.

# СПИСОК ЛІТЕРАТУРИ


1. Suyash Dubey, “Types of Mobile Apps: Native, Hybrid, Web and Progressive Web Apps” – https://www.pcloudy.com/types-of-mobile-apps/
2. Annie Dossey, “[Infografic] A Guide to Mobile App Development: Web vs. Native vs. Hybrid” – https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/
3. Nishta Maheshwari, “Top 15 Cross-platform Mobile Development in 2020” – https://nmgtechnologies.com/blog/frameworks-hybrid-app-development.html#react-native
4. Dotan Nahum, “Getting Started” in *Programming React Native*, 2015, pp.6-17.
5. John M. Wargo, “Introduction to Apache Cordova” in *Apache Cordova API Cookbook*, USA: Pearson Education, Inc., 2015, pp. 1-17.
6. Chris Griffith, “Hybrid Mobile Apps” in *Mobile App Development with Ionic*, Meg Foley and Justin Billing, USA: O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2017, pp. 1-4.
7. “Xamarin” – https://dotnet.microsoft.com/apps/xamarin
8. Serguei Tarassov, “Classification of of data models in DBMS” – https://www.researchgate.net/publication/316608591_Classification_of_data_models_in_DBMS
9. “Database” – https://www.oracle.com/database/what-is-database.html
10. Hardik Shah, “React Native Database – Choosing the right database for your React Native app” – https://www.simform.com/react-native-database-selection-guide/#3
11. Umair Hassan, “Introduction To Xamarin For Begginers”. Available: https://www.c-sharpcorner.com/article/introduction-to-xamarin-for-beginners/
12. Исаченко А. Н. Модели данных и системы управления базами данных : пособие / А.Н. Исаченко, С.П. Бондаренко. – Минск : БГУ, 2007. – с. 5-16.

# ДОДАТОК А

## Програмна реалізація

AppNavigation.js

```
const RecipeStack = createStackNavigator();

const AppNavigation = () => (
  <RecipeStack.Navigator
    initialRouteName="Home"
    screenOptions={{
      headerStyle: {
        ...Platform.select({
          ios: {
            backgroundColor: THEME.WHITE,
          },
          android: {
            backgroundColor: THEME.MAIN_COLOR,
          },
          default: {
            backgroundColor: THEME.MAIN_COLOR,
          },
        }),
      },
      headerTintColor: THEME.WHITE,
    }}
  >
    <RecipeStack.Screen
      name="Home"
      options={{ title: 'Рецепты' }}
      component={MainScreen}
    />
    <RecipeStack.Screen
      name="Recipe"
      component={RecipeScreen}
      options={{ title: 'Рецепт номер 1' }}
    />
    <RecipeStack.Screen
      name="Search"
      component={SearchScreen}
      options={{ title: 'Поиск' }}
    />
    <RecipeStack.Screen
      name="Edit"
      component={EditScreen}
      options={{ title: 'Редактировать' }}
    />
  </RecipeStack.Navigator>
);

const Booked = createStackNavigator();

const BookedNavigation = () => (
  <Booked.Navigator
    initialRouteName="Booked"
    screenOptions={{
      headerStyle: {
```

```
            ...Platform.select({
              ios: {
                backgroundColor: THEME.WHITE,
              },
              android: {
                backgroundColor: THEME.MAIN_COLOR,
              },
              default: {
                backgroundColor: THEME.MAIN_COLOR,
              },
            }),
          },
          headerTintColor: THEME.WHITE,
        }}
      >
        <Booked.Screen
          name="Booked"
          options={{ title: 'Рецепты' }}
          component={BookmarkedScreen}
        />
        <Booked.Screen
          name="Recipe"
          component={RecipeScreen}
          options={{ title: 'Рецепт номер 1' }}
        />
        <RecipeStack.Screen
          name="Search"
          component={SearchScreen}
          options={{ title: 'Поиск' }}
        />
        <RecipeStack.Screen
          name="Edit"
          component={EditScreen}
          options={{ title: 'Редактировать' }}
        />
      </Booked.Navigator>
);

const Tab =
  Platform.OS === 'android'
    ? createMaterialBottomTabNavigator()
    : createBottomTabNavigator();

const NavigationTabs = () => (
  <Tab.Navigator
    screenOptions={({ route }) => ({
      tabBarIcon: ({ focused, color, size }) => {
        let iconName;

        if (route.name === 'Recipes') {
          iconName = focused ? 'ios-albums' : 'ios-albums';
        } else if (route.name === 'Booked') {
          iconName = focused ? 'ios-star' : 'ios-star';
        }

        return <Ionicons name={iconName} size={20} color={color} />;
      },
    })}
    {...(Platform.OS === 'android'
      ? {
          activeTintColor: THEME.MAIN_COLOR,
          shifting: true,
```

```
            barStyle: { backgroundColor: THEME.MAIN_COLOR },
          }
          : {
            tabBarOptions: {
              activeTintColor: THEME.MAIN_COLOR,
              inactiveTintColor: THEME.GRAY,
            },
          })}
    >
      <Tab.Screen
        name="Recipes"
        component={AppNavigation}
        options={{ title: 'Все' }}
      />
      <Tab.Screen
        name="Booked"
        component={BookedNavigation}
        options={{ title: 'Избранные' }}
      />
    </Tab.Navigator>
);

const About = createStackNavigator();

const AboutNavigation = () => (
    <About.Navigator
      screenOptions={{
        headerStyle: {
          ...Platform.select({
            ios: {
              backgroundColor: THEME.WHITE,
            },
            android: {
              backgroundColor: THEME.MAIN_COLOR,
            },
            default: {
              backgroundColor: THEME.MAIN_COLOR,
            },
          }),
        },
        headerTintColor: THEME.WHITE,
      }}
    >
      <About.Screen
        name="About"
        options={{ title: 'О приложении' }}
        component={AboutScreen}
      />
    </About.Navigator>
);

const Create = createStackNavigator();

const CreateNavigation = () => (
    <Create.Navigator
      screenOptions={{
        headerStyle: {
          ...Platform.select({
            ios: {
              backgroundColor: THEME.WHITE,
            },
            android: {
```

```
                backgroundColor: THEME.MAIN_COLOR,
              },
              default: {
                backgroundColor: THEME.MAIN_COLOR,
              },
            }),
          },
          headerTintColor: THEME.WHITE,
        }}
    >
      <Create.Screen
        name="CreateScreen"
        options={{ title: 'Создать рецепт' }}
        component={CreateScreen}
      />
    </Create.Navigator>
);

const Drawer = createDrawerNavigator();

export const NavigationDrawer = () => (
    <NavigationContainer>
      <Drawer.Navigator
            drawerContentOptions={{
                  activeTintColor: THEME.MAIN_COLOR,
            }}
      >
        <Drawer.Screen
          name="Main"
          component={NavigationTabs}
          options={{
            title: 'Главная',
          }}
        />
        <Drawer.Screen
          name="About"
          component={AboutNavigation}
          options={{
            title: 'О приложение',
          }}
        />
        <Drawer.Screen
          name="Create"
          component={CreateNavigation}
          options={{
            title: 'Создать рецепт',
          }}
        />
      </Drawer.Navigator>
    </NavigationContainer>
);
```

## CreateScreen.js

```
export const CreateScreen = ({ navigation }) => {
  React.useLayoutEffect(() => {
    navigation.setOptions({
      headerLeft: () => (
        <HeaderButtons
          HeaderButtonComponent={AppHeaderIcon}
          title="Main icon left"
        >
```

```
            <Item
              title="Take recipe"
              iconName="ios-menu"
              onPress={() => navigation.toggleDrawer()}
            />
          </HeaderButtons>
      ),
    });
}, []);

const [newRecipe, setRecipe] = useState({
  title: '',
  description: '',
  ingredients: [],
  cookTime: '',
  kcal: '',
  tags: [],
  booked: false,
});
const [img, imgSet] = useState(null);
const dispatch = useDispatch();

const saveHandler = () => {
  const recipe = {
    ...newRecipe,
    imgMain: img,
    date: new Date().toJSON(),
  };
  imgSet(null);
  dispatch(addRecipe(recipe));
  navigation.navigate('Home');
  setRecipe({
    title: '',
    description: '',
    ingredients: [],
    cookTime: '',
    kcal: '',
    tags: [],
    booked: false,
  });
};

const photoPickHandler = (uri) => {
  imgSet(uri);
};

const setTitle = (value) => {
  setRecipe({ ...newRecipe, title: value });
};

const setDescription = (value) => {
  setRecipe({ ...newRecipe, description: value });
};

const setCookTime = (value) => {
  setRecipe({ ...newRecipe, cookTime: value });
};

const setKcal = (value) => {
  setRecipe({ ...newRecipe, kcal: value });
};
```

```
const addIngredient = (value) => {
  setRecipe({
    ...newRecipe,
    ingredients: [
      {
        id: new Date().getTime(),
        name: value,
        description: '',
      },
      ...newRecipe.ingredients,
    ],
  });
};

const addIngredientDescription = (value, id) => {
  const ingredientIndex = newRecipe.ingredients.findIndex(
    (item) => item.id === id
  );

  if (ingredientIndex !== -1) {
    newRecipe.ingredients[ingredientIndex].description = value;
    setRecipe({ ...newRecipe });
  }
};

const changeIngredientName = (value, id) => {
  const ingredientIndex = newRecipe.ingredients.findIndex(
    (item) => item.id === id
  );

  if (ingredientIndex !== -1) {
    newRecipe.ingredients[ingredientIndex].name = value;
    setRecipe({ ...newRecipe });
  }
};

const removeIngredientFromRecipe = (id) => {
  setRecipe({
    ...newRecipe,
    ingredients: newRecipe.ingredients.filter((item) => item.id !== id),
  });
};

return (
  <KeyboardShift>
    {() => (
      <ScrollView style={{ backgroundColor: '#ffffff' }}>
        <View>
          <TouchableWithoutFeedback onPress={Keyboard.dismiss}>
            <View style={styles.wrapper}>
              <Text style={styles.title}>Новый рецепт</Text>
              <PhotoPicker
                onPick={photoPickHandler}
                img={img}
                imgSet={imgSet}
              />
              <View style={styles.info}>
                <BaseItem style={styles.item} floatingLabel>
                  <Label>Название рецепта</Label>
                  <Input
                    value={newRecipe.title}
                    onChangeText={setTitle}
```

```
                        />
                      </BaseItem>
                      <BaseItem style={styles.item} floatingLabel>
                        <Label>Время приготовления блюда</Label>
                        <Input
                          value={newRecipe.cookTime}
                          onChangeText={setCookTime}
                        />
                      </BaseItem>
                      <BaseItem style={styles.item} floatingLabel>
                        <Label>
                          Каллорийность блюда (на 100г)
                        </Label>
                        <Input
                          value={newRecipe.kcal}
                          onChangeText={setKcal}
                        />
                      </BaseItem>
                      <Textarea
                        style={styles.item}
                        rowSpan={6}
                        value={newRecipe.description}
                        onChangeText={setDescription}
                        bordered
                        placeholder="Опишите рецепт"
                      />
                    </View>
                    <IngredientsList
                      onAdd={addIngredient}
                      addDescription={addIngredientDescription}
                      changeTitle={changeIngredientName}
                      removeIngredient={
                        removeIngredientFromRecipe
                      }
                      data={newRecipe.ingredients}
                    />
                    <Button
                      title="Сохранить рецепт"
                      color={THEME.MAIN_COLOR}
                      onPress={saveHandler}
                      disabled={
                        !newRecipe.title ||
                        !newRecipe.description
                      }
                    />
                  </View>
                </TouchableWithoutFeedback>
              </View>
            </ScrollView>
          )}
        </KeyboardShift>
    );
};

const styles = StyleSheet.create({
    wrapper: {
      padding: 10,
    },
    info: {
      marginVertical: 25,
    },
    item: {
```

```
            marginVertical: 5,
        },
        title: {
            fontSize: 20,
            textAlign: 'center',
            fontFamily: 'loraRegular',
            marginVertical: 10,
        },
        textarea: {
            padding: 10,
            marginBottom: 10,
        },
    });
```

## EditScreen.js

```
export const EditScreen = ({ navigation, route }) => {
  const recipeId = route.params?.recipeId;

  const [newRecipe, setRecipe] = useState(
    useSelector((state) =>
      state.recipe.allRecipes.find((r) => r.id === recipeId)
    )
  );

  const [img, imgSet] = useState(newRecipe.imgMain);
  const dispatch = useDispatch();

  const saveHandler = () => {
    const recipe = {
      ...newRecipe,
      imgMain: img,
      date: new Date().toJSON(),
    };
    imgSet(null);
    dispatch(editRecipe(recipe));
    navigation.navigate('Recipe', {
      recipeId: recipe.id,
    });
  };

  const photoPickHandler = (uri) => {
    imgSet(uri);
  };

  const setTitle = (value) => {
    setRecipe({ ...newRecipe, title: value });
  };

  const setDescription = (value) => {
    setRecipe({ ...newRecipe, description: value });
  };

  const setCookTime = (value) => {
    setRecipe({ ...newRecipe, cookTime: value });
  };

  const setKcal = (value) => {
    setRecipe({ ...newRecipe, kcal: value });
  };
```

```
const addIngredient = (value) => {
  setRecipe({
    ...newRecipe,
    ingredients: [
      {
        id: new Date().getTime(),
        name: value,
        description: '',
      },
      ...newRecipe.ingredients,
    ],
  });
};

const addIngredientDescription = (value, id) => {
  const ingredientIndex = newRecipe.ingredients.findIndex(
    (item) => item.id === id
  );

  if (ingredientIndex !== -1) {
    newRecipe.ingredients[ingredientIndex].description = value;
    setRecipe({ ...newRecipe });
  }
};

const changeIngredientName = (value, id) => {
  const ingredientIndex = newRecipe.ingredients.findIndex(
    (item) => item.id === id
  );

  if (ingredientIndex !== -1) {
    newRecipe.ingredients[ingredientIndex].name = value;
    setRecipe({ ...newRecipe });
  }
};

const removeIngredientFromRecipe = (id) => {
  setRecipe({
    ...newRecipe,
    ingredients: newRecipe.ingredients.filter((item) => item.id !== id),
  });
};

if (!newRecipe) {
  return (
    <View style={styles.center}>
      <ActivityIndicator color={THEME.MAIN_COLOR} size="large" />
    </View>
  );
}

return (
  <KeyboardShift>
    {() => (
      <ScrollView style={{ backgroundColor: '#ffffff' }}>
        <View style={{ flex: 1 }}>
          <TouchableWithoutFeedback onPress={Keyboard.dismiss}>
            <View style={styles.wrapper}>
              <Text style={styles.title}>
                Изменить рецепт
              </Text>
```

```
<PhotoPicker
  onPick={photoPickHandler}
  img={img}
  imgSet={imgSet}
/>
<View style={styles.info}>
  <BaseItem style={styles.item} floatingLabel>
    <Label>Название рецепта</Label>
    <Input
      value={
        newRecipe?.title.toString() ??
        ''
      }
      onChangeText={setTitle}
    />
  </BaseItem>
  <BaseItem style={styles.item} floatingLabel>
    <Label>Время приготовления блюда</Label>
    <Input
      value={
        newRecipe?.cookTime.toString() ??
        ''
      }
      onChangeText={setCookTime}
    />
  </BaseItem>
  <BaseItem style={styles.item} floatingLabel>
    <Label>
      Каллорийность блюда (на 100г)
    </Label>
    <Input
      value={
        newRecipe?.kcal.toString() ?? ''
      }
      onChangeText={setKcal}
    />
  </BaseItem>
  <Textarea
    style={styles.item}
    rowSpan={6}
    value={
      newRecipe?.description.toString() ??
      ''
    }
    onChangeText={setDescription}
    bordered
    placeholder="Опишите рецепт"
  />
</View>
<IngredientsList
  onAdd={addIngredient}
  addDescription={addIngredientDescription}
  changeTitle={changeIngredientName}
  removeIngredient={
    removeIngredientFromRecipe
  }
  data={newRecipe.ingredients}
/>
<Button
  title="Сохранить рецепт"
  color={THEME.MAIN_COLOR}
  onPress={saveHandler}
```

```
                        disabled={
                            !newRecipe?.title ||
                            !newRecipe?.description
                        }
                    />
                </View>
            </TouchableWithoutFeedback>
        </View>
    </ScrollView>
  )}
 </KeyboardShift>
 );
};

const styles = StyleSheet.create({
 wrapper: {
  padding: 10,
 },
 info: {
  marginVertical: 25,
 },
 item: {
  marginVertical: 5,
 },
 title: {
  fontSize: 20,
  textAlign: 'center',
  fontFamily: 'loraRegular',
  marginVertical: 10,
 },
 textarea: {
  padding: 10,
  marginBottom: 10,
 },
 center: {
  flex: 1,
  justifyContent: 'center',
  alignItems: 'center',
 },
});
```

## RecipeScreen.js

```
export const RecipeScreen = ({ navigation, route }) => {
 const recipeId = route.params?.recipeId;
 const dispatch = useDispatch();

 const recipe = useSelector((state) =>
  state.recipe.allRecipes.find((r) => r.id === recipeId)
 );
 const toggleHandler = useCallback(() => {
  dispatch(toggleBooked(recipe));
 }, [dispatch, recipe]);

 const booked = useSelector((state) =>
  state.recipe.bookedRecipes.some((r) => r.id === recipeId)
 );

 const removeHandler = () => {
  Alert.alert(
   'Удаление рецепта',
```

```
      'Вы точно хотите удалить рецепт?',
      [
        {
          text: 'Отменить',
          style: 'cancel',
        },
        {
          text: 'Удалить',
          style: 'destructive',
          onPress: () => {
            navigation.goBack();
            dispatch(removeRecipe(recipeId));
          },
        },
      ],
      { cancelable: false }
    );
};

const editHandler = () => {
    navigation.navigate('Edit', {
      recipeId: recipe.id,
    });
};

const shareRecipe = async () => {
    try {
      let ingredients = 'Ингредиенты\n';
      recipe.ingredients.forEach((item) => {
        ingredients += `- ${item?.name ?? item?.ingredient} ${
          item?.description
        }\n`;
      });
      const result = await Share.share({
        message: `${recipe.title}\n
                                ${recipe.description}
                                ${ingredients.length > 12 ? ingredients : ''}
                                ${new Date(recipe.date).toLocaleDateString()} ${
          recipe?.kcal.toString().length > 0
            ? '( ' + recipe?.kcal.toString() + ' kcal )'
            : ''
        } ${
          recipe?.cookTime.toString().length > 0
            ? '( ' + recipe?.cookTime.toString() + ' )'
            : ''
        }`.trim(),
      });
    } catch (error) {
      alert(error.message);
    }
};

React.useLayoutEffect(() => {
    const iconName = booked ? 'ios-heart' : 'ios-heart-empty';

    navigation.setOptions({
      title: recipe
        ? `Рецепт: ${
            recipe?.title.length > 10
              ? recipe?.title.substring(0, 10).trim() + '...'
              : recipe?.title
          }`
```

```
          : '',
        headerRight: () => (
          <HeaderButtons
            HeaderButtonComponent={AppHeaderIcon}
            title="Recipe favourite"
          >
            <Item
              title="Share recipe"
              iconName="md-share"
              onPress={shareRecipe}
            />
            <Item
              title="Take recipe"
              iconName={iconName}
              onPress={toggleHandler}
            />
          </HeaderButtons>
        ),
      });
    }, [booked, recipe]);

    if (!recipe) {
      return null;
    }

    return (
      <ScrollView style={{backgroundColor: "#ffffff"}}>
        <Image
          source={
            recipe.imgMain
              ? { uri: recipe.imgMain }
              : require('../../assets/img/def_recipe.jpg')
          }
          style={styles.image}
        />
        <View style={styles.textWrap}>
          <Text style={styles.title}>{recipe.title}</Text>
          <BaseItem style={{ flex: 1, paddingBottom: 5 }}>
            <Left style={styles.recipeDate}>
              <Ionicons name="md-calendar" size={25} />
              <Text style={{ marginLeft: 5 }}>
                {new Date(recipe.date).toLocaleDateString()}
              </Text>
            </Left>
            {recipe?.kcal.toString().length > 0 && (
              <Body style={styles.recipeKcal}>
                <Ionicons name="ios-restaurant" size={25} />
                <Text style={{ marginLeft: 5 }}>
                  {recipe.kcal.toString()} kcal
                </Text>
              </Body>
            )}
            {recipe?.cookTime.length > 0 && (
              <Right style={styles.recipeTime}>
                <Ionicons name="md-time" size={25} />
                <Text style={{ marginLeft: 5 }}>
                  {recipe.cookTime}
                </Text>
              </Right>
            )}
          </BaseItem>
          <Text style={{ textAlign: 'justify' }} multiline>
```

```
            {recipe.description}
          </Text>
          {recipe.ingredients.length > 0 && (
            <View>
              <Text style={styles.subTitle}>Ингредиенты</Text>
              <FlatList
                data={recipe.ingredients}
                keyExtractor={(item) => item.id.toString()}
                renderItem={({ item }) => (
                  <Card>
                    <CardItem>
                      <Body style={styles.ingItem}>
                        <Text style={{ width: '35%' }}>
                          {item.name ?? item.ingredient}
                        </Text>
                        <Text style={{ width: '55%' }}>
                          {item.description}
                        </Text>
                      </Body>
                    </CardItem>
                  </Card>
                )}
              />
            </View>
          )}
        </View>
        <View style={styles.btnGroup}>
          <TouchableOpacity
            onPress={removeHandler}
            style={{
              ...styles.button,
              backgroundColor: THEME.DANGER_COLOR,
            }}
          >
            <Text style={styles.buttonText}>Удалить</Text>
          </TouchableOpacity>
          <TouchableOpacity onPress={editHandler} style={styles.button}>
            <Text style={styles.buttonText}>Редактировать</Text>
          </TouchableOpacity>
        </View>
      </ScrollView>
    );
};

const styles = StyleSheet.create({
  center: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  image: {
    width: '100%',
    height: 200,
  },
  textWrap: {
    padding: 10,
  },
  title: {
    fontSize: 30,
    fontFamily: 'loraRegular',
  },
  ingItem: {
```

```
      alignItems: 'center',
      height: 20,
      flexDirection: 'row',
      justifyContent: 'space-between',
      marginVertical: 2,
    },
    recipeDate: { flexDirection: 'row', alignItems: 'center' },
    recipeTime: {
      flexDirection: 'row',
      alignItems: 'center',
      justifyContent: 'flex-end',
    },
    recipeKcal: {
      flexDirection: 'row',
      alignItems: 'center',
      justifyContent: 'flex-end',
    },
    subTitle: {
      fontSize: 20,
      textAlign: 'center',
      fontFamily: 'loraRegular',
      marginBottom: 10,
    },
    wrapper: {
      marginBottom: 10,
    },
    btnGroup: {
      flex: 1,
      marginVertical: 10,
      flexDirection: 'row',
      justifyContent: 'center',
    },
    button: {
      flex: 1,
      alignItems: 'center',
      flexDirection: 'row',
      backgroundColor: THEME.MAIN_COLOR,
      marginHorizontal: 5,
      height: 35,
      borderRadius: 5,
    },
    buttonText: {
      color: THEME.WHITE,
      textAlign: 'center',
      width: '100%',
    },
});
```

SearchScreen.js

```
export const SearchScreen = ({ navigation }) => {
    const [search, setSearch] = useState({
      searchField: '',
      ingredientField: '',
      ingredients: [],
    });
    const dispatch = useDispatch();
    const searchedRules = useSelector((state) => state.recipe.searchRules);

    useEffect(() => {
      setSearch({ ...search, ...searchedRules });
    }, []);
```

```
const addIngredient = () => {
  setSearch({
    ...search,
    ingredientField: '',
    ingredients: [
      { id: new Date().getTime(), name: search.ingredientField },
      ...search.ingredients,
    ],
  });
};

const setSearchField = (text) => {
  setSearch({ ...search, searchField: text.trim() });
};

const setIngredientFiled = (text) => {
  setSearch({ ...search, ingredientField: text.trim() });
};

const removeIngredient = ({ id }) => {
  setSearch({
    ...search,
    ingredients: search.ingredients.filter((item) => item.id !== id),
  });
};

const saveFilter = () => {
  dispatch(
    searchRecipes({
      searchField: search.searchField,
      ingredients: search.ingredients,
    })
  );
  navigation.navigate('Home');
};

return (
  <KeyboardShift>
    {() => (
      <ScrollView style={{ backgroundColor: '#ffffff' }}>
        <View style={styles.wrapper}>
          <Label>Поиск по названию рецепта</Label>
          <BaseItem style={styles.item}>
            <Input
              value={search.searchField}
              onChangeText={setSearchField}
              placeholder="Укажите название рецепта"
            />
          </BaseItem>
          <View style={styles.list}>
            <View>
              <Label>Поиск по ингредиентам</Label>
              <BaseItem style={styles.item}>
                <Input
                  value={search.ingredientField}
                  onChangeText={setIngredientFiled}
                  placeholder="Укажите ингредиент"
                />
              </BaseItem>
              <Button
                title="Добавить ингредиент"
                color={THEME.MAIN_COLOR}
```

```
                        onPress={() => {
                          addIngredient();
                        }}
                        disabled={
                          !search.ingredientField.trim().length ||
                          search.ingredientField.trim().length >
                            17
                        }
                      />
                    </View>
                    <View>
                      <FlatList
                        data={search.ingredients}
                        keyExtractor={(item) => item.id.toString()}
                        renderItem={({ item }) => (
                          <Card>
                            <CardItem>
                              <Body style={styles.ingItem}>
                                <Text style={{}}>
                                  {item.name}
                                </Text>
                                <TouchableOpacity
                                  onPress={() => {
                                    removeIngredient(
                                      item
                                    );
                                  }}
                                >
                                  <Ionicons
                                    name="ios-close-circle-outline"
                                    size={22}
                                    color={
                                      THEME.DANGER_COLOR
                                    }
                                  />
                                </TouchableOpacity>
                              </Body>
                            </CardItem>
                          </Card>
                        )}
                      />
                    </View>
                  </View>
                  <Button
                    title="Найти"
                    color={THEME.MAIN_COLOR}
                    onPress={saveFilter}
                  />
                </View>
              </ScrollView>
            )}
          </KeyboardShift>
        );
      };

      const styles = StyleSheet.create({
        wrapper: {
          padding: 10,
          flex: 1,
        },
        mainWrapper: { paddingHorizontal: 10 },
        title: {
```

```
    fontSize: 20,
    textAlign: 'center',
    fontFamily: 'loraRegular',
    marginVertical: 10,
  },
  textarea: {
    padding: 10,
    marginBottom: 10,
  },
  input: {
    padding: 5,
    marginBottom: 5,
  },
  ingItem: {
    alignItems: 'center',
    height: 20,
    flexDirection: 'row',
    justifyContent: 'space-between',
    marginVertical: 2,
  },
  info: {
    marginVertical: 25,
  },
  item: {
    marginVertical: 5,
  },
  list: {
    marginBottom: 10,
  },
});
```

## BookmarkedScreen.js

```
export const BookmarkedScreen = ({ navigation }) => {
  React.useLayoutEffect(() => {
    navigation.setOptions({
      title: 'Избранные рецепты',
      headerLeft: () => (
        <HeaderButtons
          HeaderButtonComponent={AppHeaderIcon}
          title="Booked main icon left"
        >
          <Item
            title="Take recipe"
            iconName="ios-menu"
            onPress={() => navigation.toggleDrawer()}
          />
        </HeaderButtons>
      ),
    });
  }, []);

  const openRecipeHandler = (recipe) => {
    navigation.navigate('Recipe', {
      recipeId: recipe.id,
      recipeTitle: recipe.title,
    });
  };

  const bookedRecipes = useSelector(state => state.recipe.bookedRecipes);
```

```
  return <RecipeList data={bookedRecipes} onOpen={openRecipeHandler} />;
};
```

## MainScreen.js

```
export const MainScreen = ({ navigation }) => {
  const dispatch = useDispatch();
  const allRecipes = useSelector((state) => state.recipe.allRecipes);
  const loading = useSelector((state) => state.recipe.loading);
  const searchedRules = useSelector((state) => state.recipe.searchRules);
  const filteredRules = useSelector((state) => state.recipe.searchedRecipes);
  const searching = useSelector((state) => state.recipe.searching);

  React.useLayoutEffect(() => {
    navigation.setOptions({
      title: 'Рецепты',
      headerRight: () => (
        <HeaderButtons
          HeaderButtonComponent={AppHeaderIcon}
          title="Main icon right"
        >
          <Item
            title="Search"
            iconName="ios-search"
            onPress={() => navigation.navigate('Search')}
          />
          <Item
            title="Date filter"
            iconName="ios-repeat"
            onPress={() => dispatch(toggleDateFilter())}
          />
          <Item
            title="Take recipe"
            iconName="md-create"
            onPress={() => navigation.navigate('Create')}
          />
        </HeaderButtons>
      ),
      headerLeft: () => (
        <HeaderButtons
          HeaderButtonComponent={AppHeaderIcon}
          title="Main icon left"
        >
          <Item
            title="Main menu"
            iconName="ios-menu"
            onPress={() => navigation.toggleDrawer()}
          />
        </HeaderButtons>
      ),
    });
  }, []);

  useEffect(() => {
    dispatch(loadRecipes());
  }, [dispatch]);

  const openRecipeHandler = (recipe) => {
    navigation.navigate('Recipe', {
      recipeId: recipe.id,
    });
  };
```

```
  if (loading) {
    return (
      <View style={styles.center}>
        <ActivityIndicator color={THEME.MAIN_COLOR} size="large" />
      </View>
    );
  }

  return (
    <RecipeList
      data={searching ? filteredRules : allRecipes}
      onOpen={openRecipeHandler}
    />
  );
};

const styles = StyleSheet.create({
  center: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
});
```

db.js

```
import * as SQLite from 'expo-sqlite';

const db = SQLite.openDatabase('test2.db');

export class DB {
  static init() {
    return new Promise((res, rej) => {
      db.exec(
        [{ sql: 'PRAGMA foreign_keys = ON;', args: [] }],
        false,
        () => {
          db.transaction((tx) => {
            tx.executeSql(
              `
                create table if not exists recipes
                (
                  id          integer not null
                    constraint recipes_pk
                      primary key autoincrement,
                  imgMain     text default '',
                  title       text    not null,
                  description text    not null,
                  date        text    not null,
                  cookTime    text,
                  kcal        integer,
                  booked      integer not null
                )`,
              [],
              () => {
                tx.executeSql(
                  `create table if not exists ingredients
                    (
                      id          integer not null
                        constraint ingredients_pk
                          primary key autoincrement,
                      recipe_id integer not null
```

```
                                                                    references recipes
                                                                        on delete cascade,
                                                                    ingredient  text not null,
                                                                    description text
                                                                )`,
                        [],
                        () => {
                            tx.executeSql(
                                `create table if not exists tags
                                                                                    (
                                                                                        id      integer not null
                                                                                            constraint tags_pk
                                                                                    primary key autoincrement,
                                                                                        tag     text    not null,
                                                                                            recipe_id integer
                                                                                                    not null
                                                                                            references recipes
                                                                                    on delete cascade
                                                                                    )`,
                                [],
                                (_, result) => res(result),
                                (_, error) => rej(error)
                            );
                        },
                        (_, error) => rej(error)
                    );
                },
                (_, error) => rej(error)
            );
        });
    }
  );
 });
}

static getRecipes() {
  return new Promise((res, rej) => {
    const allData = [];
    db.transaction((tx) => {
      tx.executeSql(
        'select * from recipes',
        [],
        (_, result) => {
          result.rows._array.forEach((item) => {
            allData.push(
              new Promise((resolve, reject) => {
                tx.executeSql(
                  `select * from ingredients where recipe_id = ?`,
                  [item.id],
                  (_ingredients, ingredientsResult) => {
                    resolve({
                      ...item,
                      ingredients:
                        ingredientsResult.rows
                          ._array,
                    });
                  },
                  (_, error) => reject(error)
                );
              })
            );
          });
```

```
                res(Promise.all(allData));
              },
              (_, error) => rej(error)
            );
          });
        });
      }

      static getRecipe(id) {
        return new Promise((res, rej) => {
          const allData = [];
          db.transaction((tx) => {
            tx.executeSql(
              'select * from recipes where id = ?',
              [id],
              (_, result) => {
                result.rows._array.forEach((item) => {
                  allData.push(
                    new Promise((resolve, reject) => {
                      tx.executeSql(
                        `select * from ingredients where recipe_id = ?`,
                        [id],
                        (_ingredients, ingredientsResult) => {
                          resolve({
                            ...item,
                            ingredients:
                              ingredientsResult.rows
                                ._array,
                          });
                        },
                        (_, error) => reject(error)
                      );
                    })
                  );
                });
                res(Promise.all(allData));
              },
              (_, error) => rej(error)
            );
          });
        });
      }

      static createPost(recipe) {
        const {
          title,
          description,
          imgMain,
          date,
          ingredients,
          cookTime,
          kcal,
        } = recipe;
        return new Promise((res, rej) => {
          db.transaction((tx) => {
            tx.executeSql(
              `insert into recipes (title, description, date, booked, imgMain, cookTime, kcal) values (?, ?, ?, ?, ?, ?, ?)`,
              [title, description, date, 0, imgMain, cookTime, kcal],
              (_, result) => res(result.insertId),
              (_, err) => rej(err)
            );
          });
```

```
    });
  }

  static createIngredients({ ingredients, id }) {
    const insertes = [];
    ingredients.forEach((ingredItem) => {
      const { name = ingredItem.ingredient, description } = ingredItem;
      insertes.push(
        new Promise((res, rej) => {
          db.transaction((tx) => {
            tx.executeSql(
              `insert into ingredients (recipe_id, ingredient, description) values (?, ?, ?)`,
              [id, name, description],
              (_, result) => {
                tx.executeSql(
                  `select * from ingredients where id = ?`,
                  [result.insertId],
                  (_, resultSelect) => {
                    res(resultSelect.rows._array[0]);
                  },
                  (_, err) => rej(err)
                );
              },
              (_, err) => rej(err)
            );
          });
        })
      );
    });

    return Promise.all(insertes);
  }

  static updateRecipe(recipe) {
    return new Promise((res, rej) => {
      db.transaction((tx) => {
        tx.executeSql(
          `update recipes set booked = ? where id = ?`,
          [recipe.booked ? 0 : 1, recipe.id],
          res,
          (_, error) => rej(error)
        );
      });
    });
  }

  static updateFullRecipe({
    cookTime,
    date,
    description,
    id,
    imgMain,
    ingredients = [],
    kcal,
    title,
  }) {
    return new Promise((res, rej) => {
      db.transaction((tx) => {
        tx.executeSql(
          `update recipes set imgMain = ?, title = ?, description = ?, date = ?, cookTime = ?, kcal = ? where id = ? `,
          [
            imgMain,
```

```
                title,
                description,
                date,
                cookTime,
                kcal,
                id,
            ],
            (_, result) => {
                tx.executeSql(
                    `delete from ingredients where recipe_id = ?`,
                    [id],
                    (_, result) => {
                        res(
                            DB.createIngredients({
                                ingredients,
                                id,
                            }).then(() => DB.getRecipe(id))
                        );
                    },
                    (_, error) => rej(error)
                );
            },
            (_, error) => rej(error)
        );
    });
  });
}

static removeRecipe(id) {
  return new Promise((res, rej) => {
        db.exec(
            [{ sql: 'PRAGMA foreign_keys = ON;', args: [] }],
            false,
            () => {
                db.transaction((tx) => {
                    tx.executeSql(
                        `delete from recipes where id = ?`,
                        [id],
                        res,
                        (_, error) => rej(error)
                    );
                })
            });
  });
}
}
```