

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Розробка мобільного додатку "Visor": комплексний
підхід до здорового харчування, аналіз складу продуктів,
рецепти»

Виконав студент групи ІПЗ-20-3
ІВАНИЦЬКИЙ Франц Анатолійович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програмного
забезпечення

Тетяна Вакалюк

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ІВАНІЦЬКИЙ Франц Анатолійович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Розробка мобільного додатку "Visor": комплексний підхід до
здорового харчування, аналіз складу продуктів, рецепти»,**

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**

Вихідні дані для роботи: **використання мобільно орієнтованих технологій для
розробки, проектування та реалізація мобільного додатку "Visor":**

**комплексний підхід до здорового харчування, аналіз складу продуктів,
рецепти**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що
їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю. М.	27.02.2024р.	27.02.2024р.
2	Єфремов Ю. М.	27.02.2024р.	27.02.2024р.
3	Єфремов Ю. М.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмного комплексу додатка «“Visor”»: комплексний підхід до здорового харчування, аналіз складу продуктів, рецепти» і пояснювальної записки. Пояснювальна записка до випускної роботи містить 69 сторінок, 38 ілюстрацій, 1 таблицю, 2 додатки та 25 літературних джерел.

Метою випускної кваліфікаційної роботи є розробка мобільного додатку, що дозволить полегшити та прискорити пошук інформації про товари та дасть можливість людям звикнути до здорового раціону харчування рахуючи калорійність продуктів. Мобільний додаток буде створено на основі фреймворку React Native, що забезпечить високу продуктивність, масштабованість та безпеку системи. Додаток буде спрямований легкий доступ до інформації харчового продукту.

Інтерфейс мобільного додатку буде інтуїтивно зрозумілим і зручним для користувачів різного рівня технічної підготовки. Він забезпечить швидкий доступ до основних функцій, таких як перегляд статей, можливість задати питання консультантам та відгукам компанії.

КЛЮЧОВІ СЛОВА: ЗДОРОВЕ ХАРЧУВАННЯ, РЕЦЕПТИ, МОБІЛЬНИЙ ДОДАТОК, DJANGO, REACT NATIVE, EXPO.

					ІПЗ.КР.Б-121-24-ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка мобільного додатку "Visor": комплексний підхід до здорового харчування, аналіз складу продуктів, рецепти	Лім.	Арк.	Аркуші	
Розроб.		Іваницький Ф. А.							
Керівник		Єфремов Ю. М.					2		
						Житомирська політехніка, група ІПЗ-20-3			
Н. контр.									
Зав. каф.									

ABSTRACT

The bachelor's thesis consists of a mobile application of «"Visor": an integrated approach to healthy eating, analysis of the composition of products, recipes» and an explanatory note. The explanatory note to the final thesis contains 69 pages, 38 illustrations, 1 table, 2 appendices and 25 literary sources.

The purpose of the final qualification work is to develop a mobile application that will facilitate and speed up the search for information about products and will enable people to get used to a healthy diet considering the calorie content of products. The mobile application will be based on the React Native framework, which will provide high performance, scalability and security of the system. The application will be directed to easy access to food product information.

The interface of the mobile application will be intuitive and convenient for users of various levels of technical training. It will provide quick access to basic functions such as viewing articles, the ability to ask questions to consultants and company reviews.

KEYWORDS: HEALTHY EATING, RECIPES, MOBILE APPLICATION, DJANGO, REACT NATIVE, EXPO.

					ИПЗ.КР.Б-121-24-ИЗ	Арк.
						4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ДОДАТКУ "VISOR"	8
1.1 Постановка задачі	8
1.2 Аналіз аналогів програмного продукту.....	9
1.3 Визначення архітектури мобільного додатку	19
1.4 Обґрунтування вибору інструментальних засобів.....	21
Висновки до першого розділу.....	22
РОЗДІЛ 2. ПРОЕКТУВАННЯ ДОДАТКУ "VISOR"	24
2.1 Визначення варіантів використання та структури системи	24
2.2 Розробка бази даних	30
2.3 Проектування та реалізація алгоритмів роботи додатку	33
2.4 Реалізація функціоналу додатку	35
Висновки до другого розділу	40
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ "VISOR"	42
3.1. Порядок встановлення та налаштування параметрів додатку	42
3.2. Структура інтерфейсу системи мобільного додатку	44
3.3. Тестування роботи мобільного додатку	62
Висновки до третього розділу	65
ВИСНОВКИ	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	68
ДОДАТКИ.....	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

DRF – Django Rest Framework

IDE – Integrated Development Environment

БД – база даних

СУБД – система управління базами даних

ORM – Object Relational Mapping

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						6

ВСТУП

Актуальність теми. Сучасний світ характеризується швидким темпом життя, який часто призводить до нехтування принципами здорового харчування. Серед основних проблем, з якими стикаються люди, є недостатня обізнаність про склад споживаних продуктів, їхню харчову цінність, а також відсутність знань щодо складання збалансованого раціону. Неправильне харчування є однією з головних причин розвитку хронічних захворювань. Тому підвищення обізнаності про здорове харчування стає нагальною потребою.

Мобільний додаток «Visor» пропонує комплексний підхід до вирішення проблеми здорового харчування, надаючи користувачам інструменти для аналізу складу продуктів та можливість складання збалансованого меню безпосередньо у додатку. Це дозволяє користувачам більш усвідомлено підходити до вибору продуктів, враховуючи їхній харчовий склад, калорійність та вміст корисних речовин, усе в одному зручному місці. Такий інтегрований підхід сприяє зменшенню витрат часу та формуванню здорових харчових звичок та зниженню ризику розвитку хронічних захворювань.

Метою кваліфікаційної роботи є розробка та впровадження мобільного додатка «Visor», який забезпечує комплексний підхід до здорового харчування шляхом аналізу складу продуктів та надання користувачам рецептів здорових страв.

Для реалізації поставленої мети потрібно виконати наступні завдання:

1. Провести аналіз вимог до функціональності додатку, враховуючи потреби користувачів.
2. Здійснити вибір засобів проектування.
3. Розробити мобільний додаток.

Об'єктом дослідження є розробка, проектування та реалізація мобільного додатку для здорового харчування та аналізу продуктів.

Предметом дослідження є використання мобільно орієнтованих технологій для створення мобільного додатку для здорового харчування та аналізу продуктів.

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						7

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ДОДАТКУ "VISOR"

1.1 Постановка задачі

Сучасне суспільство все більше усвідомлює важливість здорового харчування, що зумовлює зростання попиту на інформаційні технології, які можуть допомогти людям краще контролювати свій раціон. Одним із перспективних напрямків є розробка мобільних додатків.

Основною метою кваліфікаційної роботи є створення мобільного додатку "Visor", який надаватиме користувачам комплексний інструмент для підтримки раціону харчування. Цей додаток повинен дозволити користувачам легко і зручно отримувати інформацію про склад продуктів, отримувати доступ до великої бази харчових товарів та здорових рецептів мобільного додатку.

Мобільний додаток повинен містити можливості залишення відгуків про товар, перегляду повного опису, корисності, калорійності та інших деталей товару, рецептів та способів використання харчових продуктів, одо його корисності, але й звернути увагу на думку інших людей чи вподобали вони цей товар.

Для втілення поставленої задачі, основними фазами є:

1. Аналіз вимог та функцій мобільного додатку.
2. Проектування та розробка мобільного додатку:
 - Розробка архітектури;
 - Створення впорядкованого та легкого для навігації дизайну;
3. Розробка сторінок та функціоналу мобільного додатку:
 - Модуль завантаження та привітання користувача;
 - Модуль авторизації та реєстрації користувача;
 - Модуль товарів;
 - Модуль сканеру штрихкодів;
 - Модуль рецептів;

- Модуль списку;
- Модуль особистого кабінету;

4. Тестування мобільного додатку на емуляторах.

Результатом виконання поставленого завдання зі створення мобільного додатку "Visor" стане комплексний, функціональний та інтуїтивно зрозумілий інструмент, який забезпечить користувачів необхідними засобами для підтримки здорового харчування. Користувачі додатку зможуть використовувати його не тільки, як інструмент пошуку товарів з великої бази даних, але й штрихкод сканер, базу рецептів та нотатник.

1.2 Аналіз аналогів програмного продукту

Для успішної розробки мобільного додатку важливо було провести детальний аналіз аналогічних програмних продуктів, що вже існують на ринку. Це дозволило визначити сильні та слабкі сторони конкурентів, а також зрозуміти, які функціональні можливості необхідно включити в наш додаток, щоб зробити його конкурентоспроможним.

Користувачам завжди була актуальна тема укомплектованості додатку, щоб отримати більше доступного функціоналу в одному додатку, ніж завантажувати безліч застосунків і перемикатись між ними.

Для аналізу було обрано 3 аналоги:

- Fooducate;
- Lifesum;
- Cronometer;

Першим аналогом є Fooducate[1]. Додаток для аналізу харчових продуктів та надання користувачам інформації щодо їх складу та корисності (рис. 1.1). Основна мета додатку - покращення здоров'я споживачів шляхом надання доступної інформації про продукти харчування. Застосунок має застарілий дизайн, але містить детальну інформацію про харчову цінність продукту, включно з рейтингом продукту

який базується на калорійності, здорових та шкідливих складниках, рівні цукру, що є дуже цінним для людей, котрі слідкують саме за складниками та калорійністю раціону.

Основний екран мобільного додатку демонструє заплановані та прораховані прийоми їжі, а також лічильник кроків.

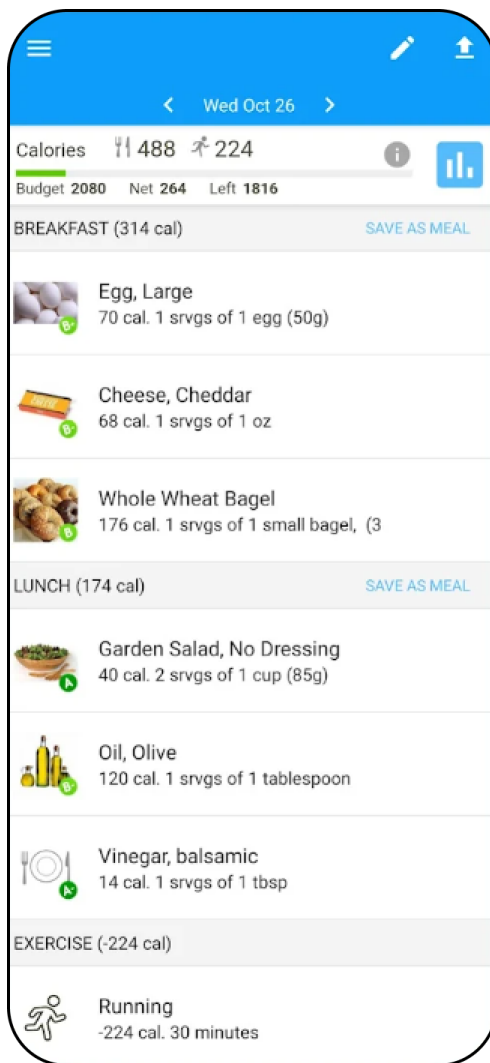


Рисунок 1.1 – Головна сторінка мобільного додатку Fooducate

Наступна сторінка демонструє сканер штрихкодів (рис. 1.2), після вдалого сканування додаток видає картку товару, де в повному обсязі описані всі харчові цінності продукту.

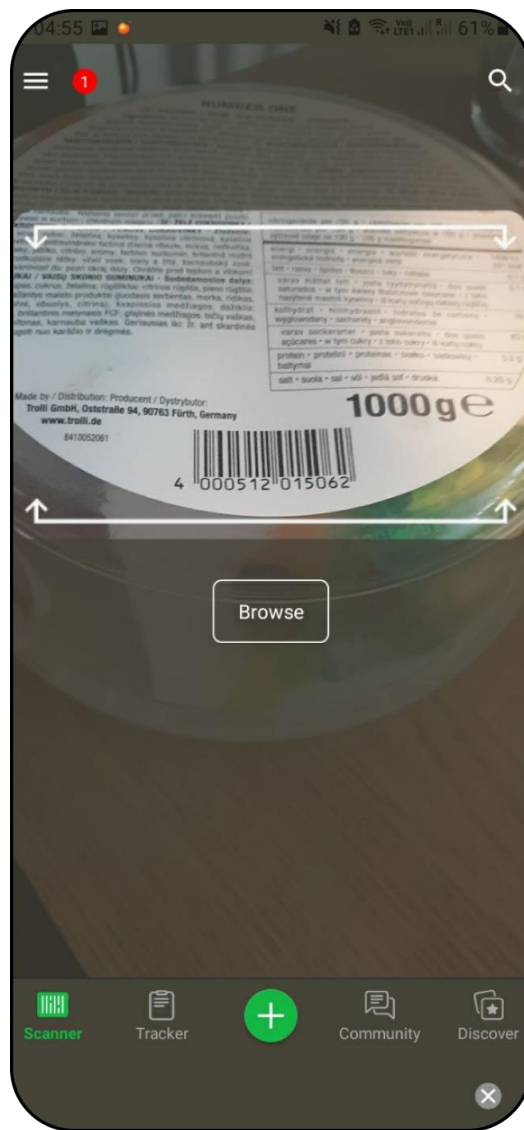


Рисунок 1.2 – Екран сканеру в мобільному додатку Fooducate

У мобільному додатку існує розділ "Community", де регулярно з'являються статті та новини з тем, що є актуальними для користувачів цього додатку. Користувачі можуть завжди отримувати свіжу та цікаву інформацію про оновлення, корисні поради або ділитись своїм життям та фото своєї їжі.

Останнім є екран "Discover" (рис. 1.3). В цьому розділі представлений список порад для приготування салатів та дієтичних страв.

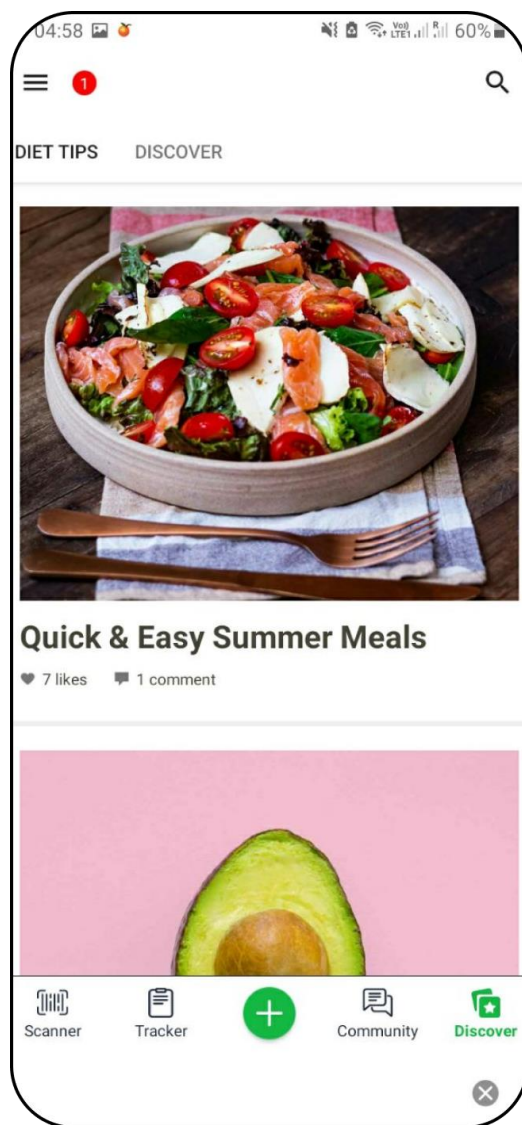


Рисунок 1.3 – Екран "Discover" у мобільному додатку Fooducate

Мобільний застосунок Fooducate має широкий функціонал, складається з чотирьох основних екранів, що наділяє додаток автономністю від інших. Дизайн має застарілий вигляд.

Функціональність:

- Комунікація спільноти між один одним;
- Регулярні публікації рецептів та відкриттів;
- Сканер штрихкодів;
- Лічильники калорій та кроків;
- Планувальник раціону харчування;

Другим аналогом є Lifesum[2] (рис. 1.4). Додаток дозволяє відстежувати споживання їжі та напоїв, рахувати калорії, макроелементи та вітаміни. Застосунок пропонує індивідуальні плани харчування, включаючи рецепти та поради щодо здорового харчування. Інтерфейс додатку є інтуїтивно зрозумілим і візуально привабливим, що робить його зручним для щоденного використання.

На головному екрані представлений лічильник калорій, кількість випитої води, та рекомендований раціон харчування, здебільшого більшість функціоналу є платним з преміум підпискою.

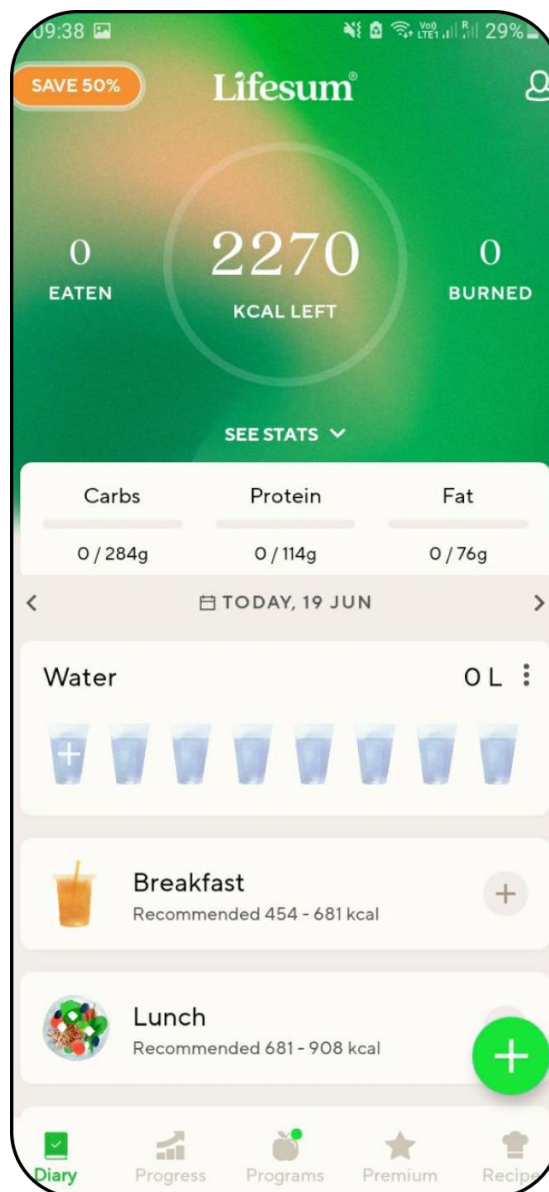


Рисунок 1.4 – Головний екран у мобільному додатку Lifesum

Головною перевагою додатку є розділ з рецептами (рис. 1.5), де зібрано великий обсяг рецептів. Однак повний доступ до цього розділу відкривається лише для користувачів з преміум підпискою.

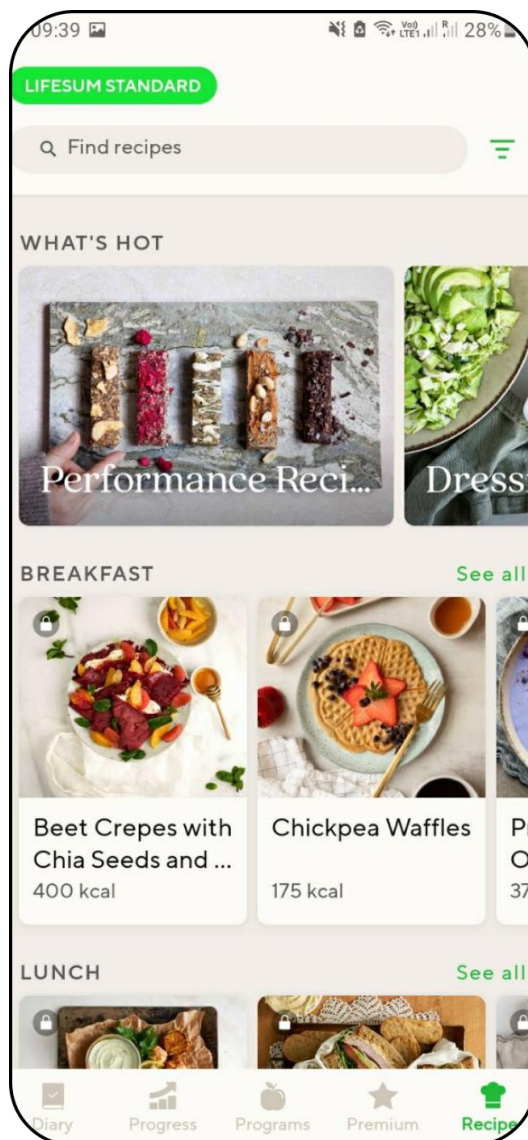


Рисунок 1.5 – Розділ “Recipes” у мобільному додатку Lifesum

Можливість монетизації мобільного застосунку яка наявна в Lifesum (рис. 1.6) не тільки дає змогу далі розвивати проект, але й надає безліч переваг для користувачів. Вона включає доступ до розширеного функціоналу, такого як розширений каталог рецептів з різноманітням дієт, інструменти для відстеження харчування та фізичної активності, персоналізовані рекомендації щодо здорового харчування, а також можливість інтерактивної взаємодії зі спільнотою користувачів. Підписка дозволяє

користувачам максимально використовувати потенціал додатку для досягнення особистих цілей зі здоров'я та фітнесу, забезпечуючи зручний і ефективний інструментарій для покращення їхнього.

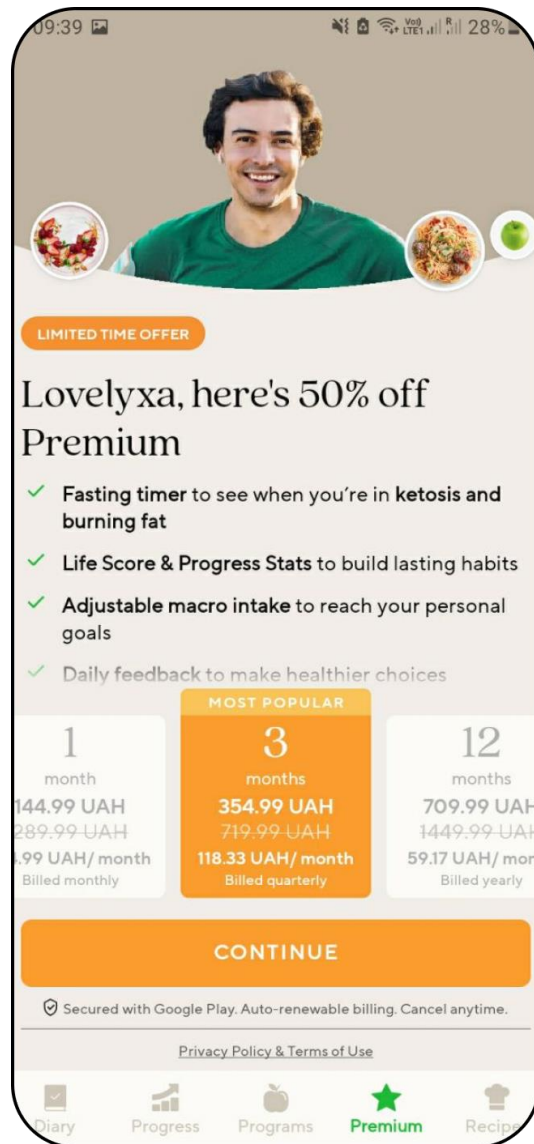


Рисунок 1.6 – Розділ “Premium” у мобільному додатку Lifesum

Функціональність:

- Велика база даних рецептів;
- Платні підписки;

Третім аналогом є Cronometer[3] (рис. 1.7), мобільний додаток, що спеціалізується на відстеженні харчування та наданні користувачам детальної інформації про їхнє харчування. Він володіє широкою базою даних продуктів і

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						15

забезпечує точні дані про калорійність для точного обчислення харчової цінності раціону. Хоча додаток має приємний зовнішній вигляд, недоліки у постановці контрастів елементів і відсутність чіткого фокусу можуть ускладнювати його використання в повсякденному житті.

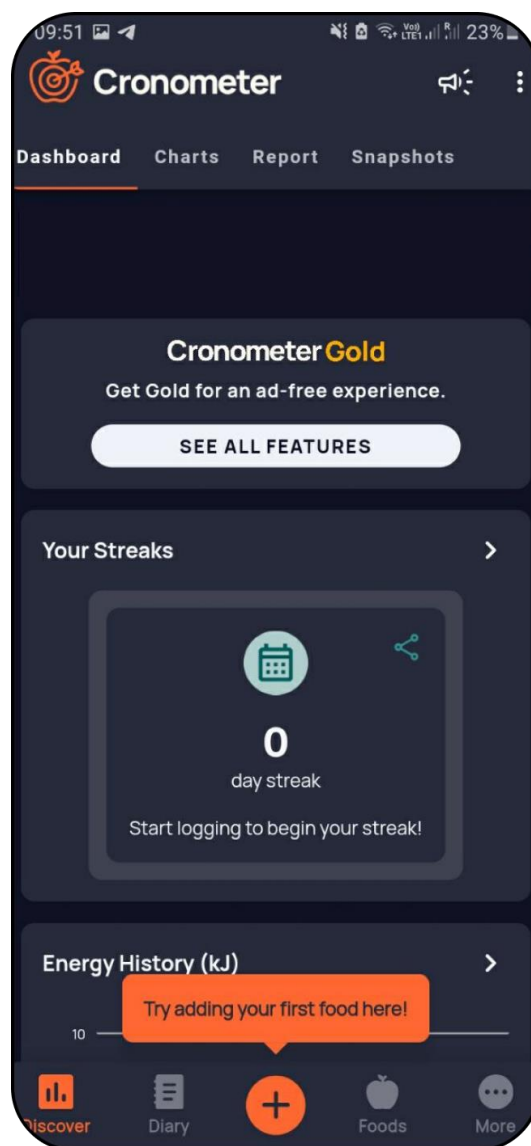


Рисунок 1.7 – Головний екран у мобільному додатку Cronometer

Застосунок має зручний сканер штрихкодів (рис. 1.8), який дозволяє користувачам не лише сканувати штрихкоди, але і власноруч вводити код, якщо це необхідно. Крім того, в сканері є функція активації ліхтаря, що полегшує процес сканування в умовах обмеженого освітлення.

Проте недоліками додатка Cronometer є його нестабільність, яка проявляється у частому аварійному закритті без зрозумілих причин. Це може призводити до непередбачених переривань роботи програми, що ускладнює її надійність і зручність використання.

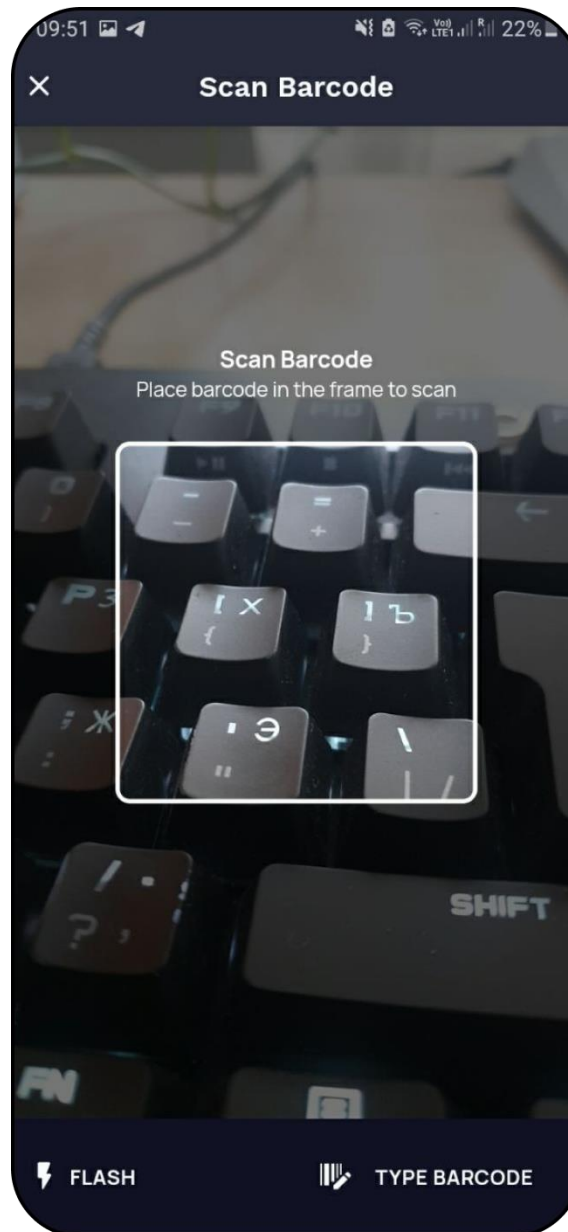


Рисунок 1.8 – Сканер штрихкодів у мобільному додатку Cronometer

Додаток забезпечує комплексну інформацію, необхідну користувачам, але потребує суттєвих оновлень для підвищення стабільності й покращення дизайну елементів.

Функціональність:

- Безкоштовний та функціональний сканер штрихкодів;
- Екран списку товару;

Для порівняння всіх описаних аналогів було створено таблицю (табл. 1.1)

Таблиця 1.1

Порівняння аналогів

Характеристика	Fooducate	Lifesum	Cronometer
Сканер штрихкодів	Зручний	Відсутній	Дуже зручний
Інтерфейс	Зручний	Зручний	Незручний
Екран рецептів	+	+	-
Екран товарів	+	-	+
Можливість написання відгуків	+	-	-
Дизайн	Застарілий та загромаджений	Чудовий	Застарілий
Стабільність роботи	Стабільний	Стабільний	Нестабільний

Проаналізувавши таблицю (табл. 1.1), можна зробити висновок, що кожен з цих мобільних додатків має свої переваги та недоліки. Наприклад, додаток Fooducate пропонує зручний сканер штрихкодів і інтерфейс для рецептів і товарів, але має застарілий дизайн і загромаджений інтерфейс. Lifesum має чудовий дизайн і стабільність, але не має сканера штрихкодів і екрану товарів. Cronometer, незважаючи на зручний сканер і екран товарів, має незручний інтерфейс і відсутність екрану рецептів, а також страждає від застарілого дизайну та нестабільності роботи.

З цього порівняння можна зрозуміти, що жоден з цих додатків не відповідає усім вимогам одночасно. Для створення ідеального додатка для харчування та здоров'я необхідно об'єднати найкращі аспекти кожного з них: сучасний дизайн, інтуїтивно зрозумілий інтерфейс, зручний сканер штрихкодів, екрани для товарів і рецептів, а також можливість написання відгуків. Це забезпечить зручний і сучасний користувацький досвід і задовольнить усі вимоги користувачів.

1.3 Визначення архітектури мобільного додатку

Вибір архітектури мобільного додатку є критично важливим етапом у розробці проекту. Від цього вибору залежить продуктивність, підтримка, безпека, а також легкість розширення та підтримки додатку. Ефективність виконання функцій додатку значною мірою визначається архітектурним підходом.

Серед наявних архітектурних підходів розглянемо

1. Модульна архітектура.
2. Мікросервісна архітектура.
3. Монолітна архітектура.

Модульна архітектура передбачає розбиття додатку на незалежні модулі, кожен з яких відповідає за певну функціональність. Цей підхід полегшує розробку та тестування, оскільки кожен модуль можна розробляти, тестувати та підтримувати окремо. Модулі працюють в одному процесі і спільно використовують ресурси, такі як база даних або файли конфігурації. Основною перевагою цього підходу є швидкість реагування на бізнес-зміни і додавання нових функцій, що досягається за рахунок легкості додавання, видалення або зміни модулів без необхідності переписування всього додатку. Для реалізації проекту, потрібно буде розробити декілька модулів, кожен з яких відповідає за певний функціональний аспект і забезпечує розподіл відповідальностей між різними частинами системи. Це дозволить покращити керованість проектом та забезпечити гнучкість обслуговування.

Мікросервісна архітектура передбачає розбиття додатку на невеликі самостійні сервіси, кожен з яких виконує окрему функцію і може працювати незалежно. Кожен сервіс має свою власну кодову базу, базу даних та інтерфейс для комунікації з іншими сервісами, зазвичай через протоколи обміну повідомленнями. Це дозволяє розробникам одночасно працювати над різними сервісами, використовуючи різні технології, що сприяє швидкому розвитку та тестуванню. Основною перевагою мікросервісної архітектури є можливість масштабування окремих сервісів, що дозволяє ефективно управляти ресурсами і покращувати продуктивність додатку. Саме елементи мікросервісної архітектури будуть потрібні для розгортання та масштабування окремих мікросервісів.

Монолітна архітектура передбачає розробку додатку як єдиного цілого, в якому всі компоненти тісно пов'язані і працюють разом. Цей підхід спрощує процес розробки і розгортання, оскільки всі компоненти розміщені в одному проекті та розгортаються як єдиний додаток. Для невеликих додатків монолітна архітектура може бути доцільним вибором через простоту реалізації та управління. Вона дозволяє швидко впроваджувати нові функції, бо немає необхідності в координації роботи багатьох модулів чи сервісів.

Отже, для мобільного додатку, орієнтованого на допомогу користувачам в їх розумінні та підборі здорового харчування, було вирішено використати модульну архітектуру з елементами мікросервісного підходу для розробки додатку “Visor”. Такий підхід забезпечує гнучкість, масштабованість і ефективність у виконанні функцій додатку. Використання фреймворку Django[12] дозволяє ефективно реалізувати поставлені задачі, забезпечуючи високу продуктивність, легкість у підтримці та розширенні додатку.

Таким чином, обраний підхід поєднує найкращі практики та технології для створення додатку, який задовольнить всі вимоги користувачів і забезпечить зручний, стабільний і ефективний користувацький досвід.

1.4 Обґрунтування вибору інструментальних засобів.

У процесі розробки кваліфікаційної роботи було використано різноманітні інструменти, що сприяли створенню функціональної та ефективної системи. Основний фреймворк для розробки бекенду і бази даних були реалізовані з використанням Django[12] та Python[7]. Android Studio[11] використовувався для зручного написання та відлагодження клієнтської частини, для відлагодження серверної частини використовувався PyCharm. Для створення динамічного інтерфейсу був використаний React Native, що забезпечив відмінну інтерактивність додатку. Також для побудови API було використано додатковий сервіс DRF.

1. Django - це високорівневий веб-фреймворк, написаний на Python, призначений для швидкої розробки веб-додатків. Він пропонує розробникам різноманітні готові інструменти та бібліотеки, що спрощують автоматизацію рутинних задач. Фреймворк Django підтримує створення складних веб-додатків, забезпечуючи швидкість розробки завдяки великому набору вбудованих функцій, таких як адміністративна панель, система маршрутизації, обробка форм і управління базами даних
2. Android Studio - це інтегроване середовище розробки (IDE) для платформи Android, створене компанією Google. Воно базується на IntelliJ IDEA і надає розробникам зручний і потужний інструментарій для створення Android додатків. Android Studio має вбудовану підтримку розгортання Android додатків, що спрощує весь цикл розробки від початку до кінця.
3. PyCharm - це інтегроване середовище розробки (IDE) для мови програмування Python, розроблене компанією JetBrains. Воно надає розширений інструментарій для розробки Python додатків, включаючи підтримку відлагодження, автодоповнення коду, системи контролю версій і управління проектами

4. React Native - це фреймворк для розробки мобільних додатків, створений компанією Facebook. Він базується на популярній бібліотеці React для створення інтерфейсів користувача і дозволяє розробникам писати мобільні додатки, використовуючи JavaScript та React.

Після вибору та обґрунтування інструментальних засобів було сформовано системні вимоги, які визначають мінімальну конфігурацію, при якій система працюватиме стабільно:

1. Вимоги до середовища виконання:

Мінімально необхідна конфігурація, при якій система працюватиме стабільно:

- Операційна система: Windows 7 або вище;
- Процесор: 32- або 64-розрядний з тактовою частотою 2.5 ГГц або вище;
- Оперативна пам'ять: 8 ГБ для 32-розрядних процесорів та 16 ГБ для 64-розрядних процесорів;
- Вільне місце на жорсткому диску: 36 ГБ для 32-розрядних систем або 48 ГБ для 64-розрядних систем;

2. Вимоги до СУБД та доступу до даних:

- У ядрі системи повинна бути представлена реляційна система управління базами даних (СУБД);

3. Вимоги до клієнтського пристрою

- Операційна система Android;
- 2 ГБ ОЗУ;
- Камера;
- Доступ до мережі, для здійснення онлайн запитів та оновлення даних;
- 20 ГБ простору для операційної системи, програм та даних;

Висновки до першого розділу

У першому розділі було проведено аналіз вимог до системи мобільного додатку

Аналіз включав в себе огляд потреб користувачів. Результати цього дослідження виявили потреби у доступі до інформації про склад товарів, корисні рецепти та сканеру штрихкодів, які необхідно врахувати для успішної реалізації додатку.

Порівняльний аналіз існуючих рішень у даній галузі дав змогу зрозуміти оптимальні практики та найкращі підходи, які можна використати при розробці. Цей етап дослідження зібрав важливі висновки щодо того, як покращити доступ користувачів до необхідної інформації, забезпечити зручний інтерфейс і підвищити загальну користь від використання додатку.

Вибір технологій для реалізації проекту, таких як Django для серверної частини системи через його потужність, гнучкість і високу ефективність. React Native було вибрано для мобільної платформи з метою забезпечення швидкості розробки, масштабованості. Цей вибір технологій відповідає потребам у створенні ефективного інтерфейсу.

Обраний підхід до розробки "Visor" забезпечує гнучкість та можливість швидко реагувати на зміни у вимогах і потребах користувачів. Він гарантує стійкість і ефективність роботи системи навіть під час масштабування та розширення функціоналу. Таким чином, "Visor" планується бути не лише інноваційним і корисним інструментом для користувачів, але й технічно надійною платформою для подальшого розвитку і покращення..

РОЗДІЛ 2. ПРОЕКТУВАННЯ ДОДАТКУ "VISOR"

2.1 Визначення варіантів використання та структури системи

Для забезпечення ефективності і надійності мобільної системи, що спрямована на підтримку здорового харчування, визначено ключові вимоги, які гарантують її стабільну роботу і відповідність усім стандартам і специфікаціям. Ці вимоги охоплюють не лише технічні аспекти, але й забезпечують безпеку та ефективність взаємодії з користувачами.

Бізнес вимоги:

1. Основною ціллю є забезпечення користувачам можливості аналізу складу продуктів і рецептів, що сприяє покращенню здоров'я через здорове харчування;
2. Представлення проекту - проект буде реалізовано у вигляді мобільного додатку, що містить усі необхідні функції для цієї задачі;

Вимоги користувачів:

1. Можливість сканування товару;
2. Можливість залишити відгук про товар;
3. Можливість видалення відгуку про товар;
4. Перегляд відгуків про товар;
5. Відстежування статистики;
6. Перегляд списку товарів;
7. Перегляд списку рецептів;
8. Можливість запису нотаток;

Вимоги персоналу мобільного додатку:

1. Перегляд та видалення відгуків про товар;
2. Додавання, редагування, видалення товарів;
3. Додавання, редагування, видалення рецептів;
4. Додавання, редагування, видалення шкали корисності товару чи рецепту;

5. Перегляд, редагування, видалення відгуків про рецепти;

Характеристика об'єкта комп'ютеризації:

Завдяки зручному та легкому у використанні інтерфейсу додатку, користувач отримає швидкий доступ до повної бази даних продуктів і рецептів страв. Це забезпечить не лише зручність, але й ефективність в підтримці здорового харчування.

Функціональні вимоги:

1. Авторизація та реєстрація у мобільному додатку: застосунок повинен забезпечувати можливість користувачам авторизуватися за допомогою існуючого облікового запису або створювати новий шляхом реєстрації;
2. Можливості для аналізу складу харчових товарів: функціонал дозволяє передивлятися інформацію про товар, його фото, харчову цінність та шкалу корисності;
3. Нотатник: функціонал, що дозволяє користувачам створювати і зберігати персональні нотатки, які можуть включати плани харчування, списки покупок, рецепти або будь-яку іншу корисну інформацію
4. Сканування штрихкоду: інструмент для сканування штрихкодів продуктів, що дозволяє швидко отримувати інформацію про товар, включаючи його склад, харчову цінність;
5. Перегляд списку товарів: можливість користувачам переглядати повний список доступних товарів із детальною інформацією про кожен з них;
6. Перегляд список рецептів: функціонал, що дозволяє користувачам переглядати повний список рецептів, включаючи їх інгредієнти, інструкції з приготування та іншу корисну інформацію;
7. Відгуки: функціонал, що дозволяє користувачам залишати свої відгуки про товари та рецепти, з можливістю перегляду цих відгуків іншими користувачами;

Нефункціональні вимоги:

— Сприйняття:

Час, необхідний для освоєння сайту звичайних користувачів – 2 хвилини, а для просунутих користувачів – 1 хвилина;

Час відгуку для типових задач – не більше 3 секунд, для складних завдань – не більше 5 секунд;

— Надійність:

Доступність – час, що витрачається на обслуговування системи не повинно перевищувати 3% від загального часу роботи;

Середній час безвідмовної роботи – 30 календарних днів;

Норма помилок та дефектів в роботі системи – 6 помилок на десять тисяч рядків коду;

— Продуктивність:

Система повинна підтримувати мінімум 200 одночасно працюючих користувачів, пов'язаних із загальною базою даних;

— Масштабування:

Забезпечення стабільної та ефективної роботи системи при збільшенні обсягу даних і кількості користувачів;

Для візуального представлення взаємодії з користувачами була розроблена діаграма варіантів використання (рис. 2.1). Ця діаграма зображує функціональні можливості додатку для двох основних ролей: звичайного користувача та модератора.

Звичайний користувач має доступ до таких функцій: авторизація, перегляд особистих даних, сканування кодів, список товарів, список рецептів та нотатник. Перегляд особистих даних включає можливість зміни цих даних. Сканування кодів надає інформацію про товар, яка також включає перегляд відгуків та написання відгуків. Крім того, інформація про товар дозволяє користувачу створювати власне харчування. У “списку” користувач також може створювати власне харчування.

Адміністратор, у свою чергу, має розширені можливості, включаючи створення, редагування та видалення товарів і рецептів, перегляд та видалення відгуків про товари та рецепти, а також додавання, редагування та видалення шкали корисності товару чи рецепту. Діаграма є інструментом розуміння того, як користувачі взаємодіють з системою і відіграє важливу роль у аналізі та вдосконаленні додатку.

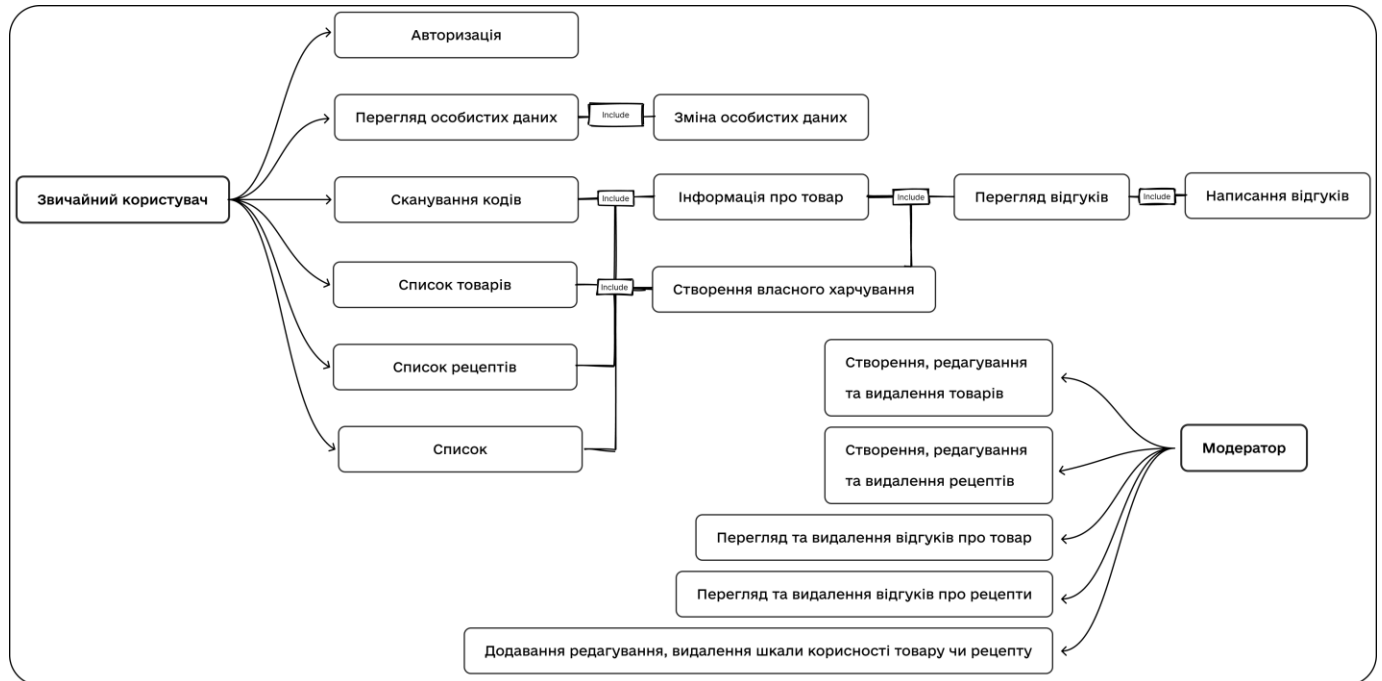


Рисунок 2.1 – Діаграма варіантів використання додатку

Підсумком аналізу та визначення функціональних вимог стала побудова ключових сутностей, які є невід'ємною частиною системи. Цей процес дозволив чітко окреслити основні елементи, з якими буде працювати додаток, та забезпечити їх взаємодію на різних рівнях.

Однією з важливих результатів цього етапу стала розробка діаграми класів сутностей, що представлена на рисунку (рис. 2.2). Діаграма класів сутностей є фундаментальним інструментом для програмістів, оскільки вона надає візуальне уявлення про архітектуру додатку та служить основою для подальшої програмної реалізації. Завдяки цьому, можна створити стабільний та надійний продукт, який відповідатиме всім встановленим вимогам та очікуванням користувачів.



Модуль “Accounts” сутність якого є CustomUser є моделлю Django, що розширює AbstractUser і містить поля унікальний ідентифікатор, ім'я користувача, URL аватара, email, пароль, активність, статус персоналу, статус суперкористувача та список користувачів.

Модуль “Healthfulness” визначає модель Django з полями унікальний ідентифікатор та поле lvl з варіантами вибору, які включають 'A', 'B', 'C', або 'D'. Також прописаний метод який повертає рівень харчової корисності.

Модуль “Products” містить клас “Product” який визначає модель для зберігання інформації про продукти. Кожен продукт має унікальний ідентифікатор для ідентифікації, назву і опис для опису характеристик продукту, штрих-код для ідентифікації в роздрібній торгівлі, зображення для візуального представлення продукту, виробника для вказівки виробника продукту, кількість калорій для харчових характеристик і шкалу харчової корисності за допомогою зовнішнього ключа, що посилається на модель “Healthfulness”.

Модуль “Recipes” сутність представлена класом “Recipe” є орієнтований для зберігання рецептів. Кожен рецепт має унікальний ідентифікатор для ідентифікації, відношення до продуктів через поле з багато-до-багатьох зв'язків, назву для ідентифікації рецепту, зображення для візуального представлення рецепту, опис для опису процесу приготування, дату створення для відстеження часу створення рецепту і оцінку шкали корисності за допомогою зовнішнього ключа, що посилається на модель “Healthfulness”.

Модуль “Reviews” є найбільш обсяжним за вмістовністю класів так функцій. Клас “BaseReviewModel” є абстрактною моделлю Django, яка визначає загальні атрибути для всіх об'єктів відгуків, включаючи унікальний ідентифікатор, вміст відгуку, користувача, який його залишив, та оцінку за вибором. Він використовується як базовий клас для моделей “ProductReview” та “RecipeReview”. Також в модулі представлені функції “update_product_rating” та “update_recipe_rating” .

Клас “ProductReview” наслідує “BaseReviewModel” і додає поле product, яке вказує на продукт, до якого відноситься відгук. Метод який наявний в тілі повертає рядок, що представляє відгук для конкретного продукту. Схожий функціонал виконує клас “RecipeReview”, який вказує на рецепт та повертає відгук для конкретного випадку.

Класи “ProductRating” та “RecipeRating” наслідують від “BaseRatingModel” і додають поля product та recipe відповідно, які є зовнішніми ключами до продуктів та рецептів. Ці моделі також мають метод, який повертає рядок, що описує рейтинг продукту чи рецепту.

2.2 Розробка бази даних

Для побудови СУБД мобільного додатку "Visor" було обрано SQLite через його спрощену впровадження та використання, низькі вимоги до ресурсів і можливість працювати без окремого сервера баз даних. SQLite ідеально підходить для проектів з обмеженим обсягом даних і вимагає мінімальних зусиль для налаштування.

Інтеграція Django з SQLite дозволила ефективно організувати базу даних. За допомогою Django було створено двадцять п'ять таблиць, включаючи сім, що були розроблені власноруч, і вісімнадцять, що були інтегровані через адміністративну панель Django або міграційні команди між таблицями. Django забезпечує ці таблиці структурованість завдяки своєму ORM, що спрощує взаємодію з даними та забезпечує їхню консистентність.

Завдяки цій інтеграції, розробники можуть не лише ефективно використовувати переваги обох технологій, але й забезпечувати високу масштабованість і надійність системи, що є критичними аспектами для успішного впровадження мобільного додатку "Visor". Це дозволяє забезпечити високу продуктивність та швидкість роботи додатку, що задовольняє високі вимоги сучасного ринку програмного забезпечення.

Кожна таблиця бази даних відповідає за зберігання різних типів даних, подій і журналу взаємодії з базою даних. Цей підхід дозволяє ефективно керувати інформацією у додатку "Visor", забезпечуючи необхідну структуру для зберігання та обробки даних користувачів. Така організація бази даних допомагає забезпечити високу швидкість розробки і надійність додатку, що є критичними аспектами для успішного впровадження та експлуатації мобільного додатку.

Таблиці створені власноруч:

1. Accounts_customuser – таблиця, була створена на основі кастомної моделі користувача (CustomUser) в Django. Поля цієї таблиці включають UUID, username, password, email, last_login, is_superuser, first_name, last_name, is_staff, is_active, data_joined та avatar_url. Таблиця містить основні дані для аутентифікації та авторизації користувачів у системі, а також додаткові поля, специфічні для функціональності додатку.
2. healthfulness_healthfulness – зберігає дані про статті, опубліковані на вебсайті. Поля цієї таблиці включають articleID, title, content, image, status та Date. Ця таблиця має зв'язок із таблицею Comments, яка містить коментарі користувачів до статей.
3. products_product – зберігає інформацію про продукти в мобільному додатку. Поля цієї таблиці включають: id - унікальний ідентифікатор продукту (UUID), name - назва продукту, description - опис продукту, barcode – штрихкод продукту, producer – виробник, calories – харчова цінність продукту, abc_score – шкала харчової корисності та picture – фотокартка товару.
4. recipes_recipe – використовується для зберігання рецептів страв. Поля включають id, name, description, create_date та abc_score_id та picture, має зв'язок з таблицею healthfulness_healthfulness.
5. recipes_recipe_products – ця таблиця використовується для зберігання інформації про те, які продукти входять до складу конкретного рецепту, забезпечуючи таким чином зв'язок між рецептами та продуктами. Поля id, recipe_id, product_id.
6. reviews_productreview – таблиця використовується для написання відгуків про продукт. Поля цієї таблиці включають id, content, rating, product_id та user_id.
7. reviews_recipereview – таблиця призначена для створення відгуків на рецепти страв. Вона містить такі поля: id, content, rating, recipe_id і user_id.

2.3 Проектування та реалізація алгоритмів роботи додатку

Процес проектування та реалізації алгоритмів для мобільного додатку є складним і вимагає уваги до багатьох аспектів розробки програмного забезпечення. Насамперед необхідно ретельно проаналізувати вимоги до додатку, щоб зрозуміти, які конкретні завдання повинні бути виконані і які функції реалізовані. На цьому етапі визначаються потреби користувачів, технічні обмеження та інші фактори, які можуть вплинути на архітектуру додатку.

При розробці алгоритмів для додатку, спрямованого на здорове харчування, акцент робиться на інформуванні користувачів та покращенні їхніх знань про раціон. Після реєстрації та авторизації користувач отримує доступ до всіх основних функцій додатку. Користувач може самостійно знаходити товари зі списку або скористатися сканером штрихкодів для швидкого пошуку інформації про продукт. Це дозволяє отримати необхідні дані за лічені секунди та прочитати відгуки інших користувачів.

Щодня модератори додатку оновлюють базу даних товарів і додають нові рецепти. Вони також видаляють неактуальні товари та коментарі, забезпечуючи тим самим актуальність і достовірність інформації. Такий підхід дозволяє користувачам отримувати детальну та якісну інформацію про харчові продукти, а також планувати свій раціон безпосередньо в додатку.

Додаток надає можливість користувачам створювати власні нотатки, списки покупок і планувати меню. Це робить процес організації харчування зручнішим і ефективнішим. Завдяки всім цим функціям, додаток допомагає користувачам дотримуватися здорового харчування, а компаніям — ефективно вирішувати питання клієнтів та покращувати рівень обслуговування.

Алгоритм роботи користувача з додатком зображено на рисунку (рис.2.4). Ця діаграма ілюструє покрокову послідовність дій, які виконує користувач під час взаємодії з додатком, починаючи від входу в систему і закінчуючи завершенням роботи. Така візуалізація алгоритму є важливим інструментом для розробників і

користувачів, оскільки вона допомагає зрозуміти, як саме працює додаток і які кроки необхідно виконати для досягнення бажаного результату.

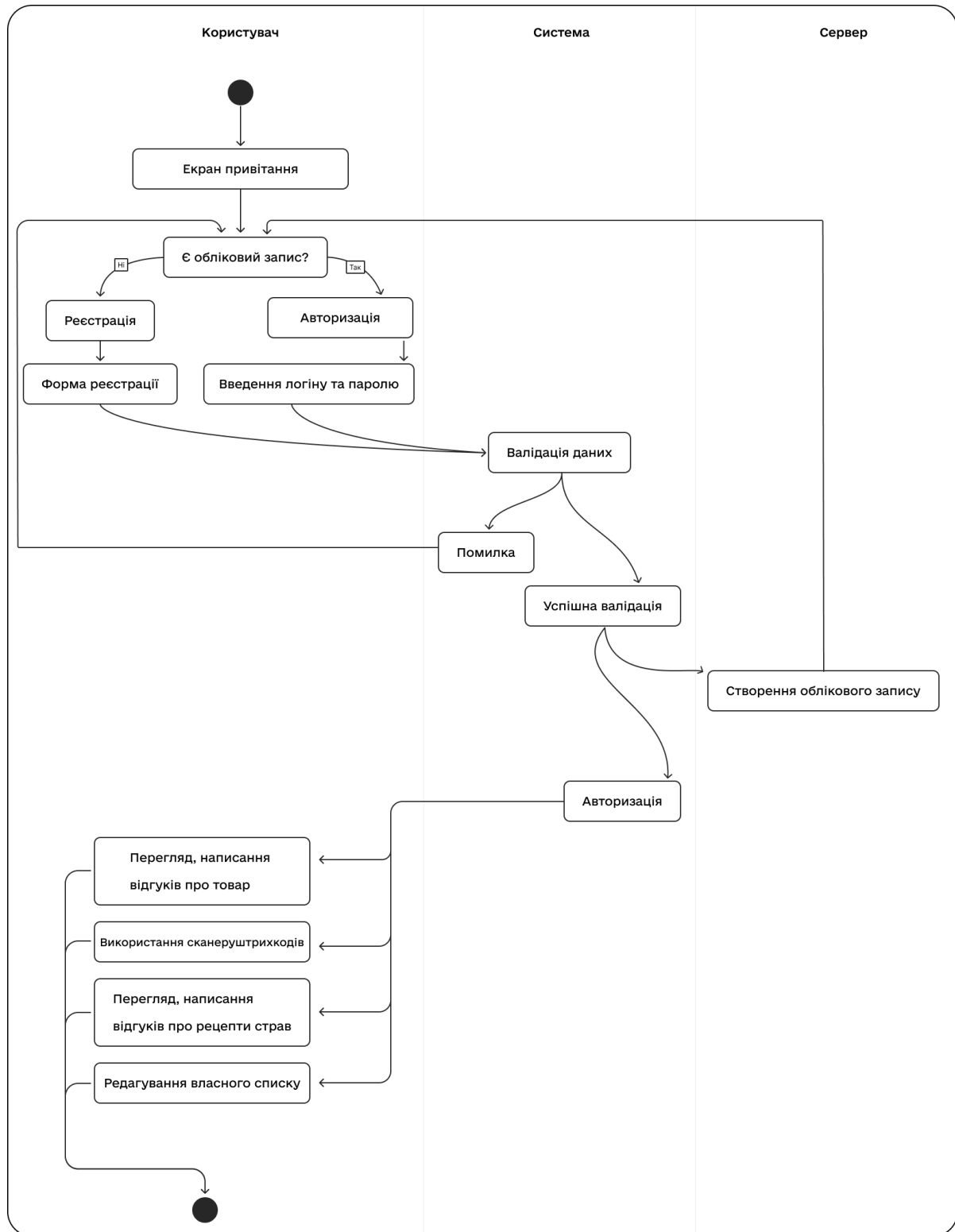


Рисунок 2.4 – Діаграма активності користувача

2.4 Реалізація функціоналу додатку

Розглянемо реалізацію початкових екранів а саме OnBoarding. Екран повинен містити слайд з певної інформацією, яка дає можливість користувачеві познайомитись з мобільним додатком “Visor”. Першим кроком імпортування необхідних залежностей і компонентів, бібліотек, зображень для слайдерів та створення стану для визначення поточного слайду екрану інтерфейсу, які підкажуть користувачам, які вимоги до пароля (мінімальна довжина).

Першим кроком є оголошення сталого елемента “SLIDES”, що представляє собою масив об’єктів, кожен з яких є окремий слайд:

```
const SLIDES = [
  {
    img: slideImage1,
    title: 'Привіт, ми Visor',
    description: 'Ti, хто допоможе тобі в організації твого здорового раціону, у нас QR сканер.',
  },
]
```

Подальше створення компонента OnboadringScreen який і буде відповідати за відображення екрану, використовується хуки “useState” який додає локальний стан компоненту, “currentSlide” який відстежує поточний слайд. Були прописані обробники для кнопок для переміщення по слайдах та навігації до наступного екрану:

```
const scrollToSlide = (index) => {
  flatListRef.current.scrollToIndex({ animated: true, index })
}

const handleOnNextSlideClick = () => {
  const nextSlide = currentSlide < SLIDES.length - 1 ? currentSlide + 1 :
currentSlide;
  setCurrentSlide(nextSlide);
  scrollToSlide(nextSlide)
}
const handleOnPrevSlideClick = () => {
  const prevSlide = currentSlide > 0 ? currentSlide - 1 : currentSlide;
  setCurrentSlide(prevSlide);
  scrollToSlide(prevSlide)
}
const handleOnLastSlideClick = () => {
  setCurrentSlide(SLIDES.length - 1);
  scrollToSlide(SLIDES.length - 1)
}
```

Екран `ProductListScreen` в мобільному додатку є досить складним та багатофункціональним компонентом. Основні складнощі полягають у необхідності забезпечити ефективне завантаження та відображення великої кількості продуктів. Це включає інтеграцію з сервісною частиною для отримання даних, обробку відповіді сервера та оновлення стану інтерфейсу користувача. Однією з ключових особливостей є реалізація пошукового функціоналу, що дозволяє користувачам швидко знаходити потрібні товари за назвою. Це вимагає розробки зручного пошукового інтерфейсу та ефективних алгоритмів фільтрації даних.

Крім того, екран повинен бути оптимізованим для роботи з великою кількістю даних, щоб забезпечити плавний скролінг і швидке відображення списку продуктів.

Ініціалізація та реалізація початкових змінних:

```
export const ProductsListScreen = () => {
  const [products, setProducts] = useState([]);
  const [searchQuery, setSearchQuery] = useState('');
}
```

Завантаження даних зі сервісної сторони. Для підключення було використано бібліотеку “`axios`”, яка і виконує HTTP-запити. URL зберігається в константі “`BASE_API_URLS.device`”, а саме:

```
export const BASE_API_URLS = {
  emulator: "http://10.0.2.2:8000/api",
  device: "http://192.168.1.31:8000/api",
}
```

При успішному отриманні даних, вони зберігаються у стані “`products`” використовуючи “`setProducts`”. Для коректного з’єднання з API, сервер був урухомлений на IPv4 адресі комп’ютера командою - `python manage.py runserver 192.168.1.31:8000`. Також набув зміни файл конфігурації серверу `settings.py`, було дозволено нову IP адресу для записку серверу.

Відображення екрану товарів, для цього використані компоненти `ScreenTemplate` та `FlatList` для відображення списку продуктів :

```
<FlatList
  data={products}
  renderItem={({ item }) => (
    <ProductCard
      id={item.id}
```

```

        name={item.name}
        imageUrl={item.picture}
        lvl={item.abc_score}
        style={styles.card}
      />
    ))
    keyExtractor={item => item.id}
    numColumns={2}
    contentContainerStyle={styles.list}
    columnWrapperStyle={styles.columnWrapper}
  />
</ScreenTemplate>
);
}

```

Для естетичного та комфортного відображення плиток продуктів було стилізовано компоненти.

Написання функціоналу екрану штрихкодів є однією з найважливіших вимог для реалізації додатку, оскільки забезпечує критичні можливості для користувачів, які підвищують загальну ефективність та зручність використання додатку.

Сканування штрихкодів дозволяє користувачам швидко отримувати детальну інформацію про продукти, зокрема їхню харчову цінність, інгредієнти, наявність алергенів та інші важливі характеристики.

Це значно прискорює процес вибору продуктів, зменшуючи час, необхідний для їх ідентифікації та порівняння, що є особливо корисним під час покупок у магазині. Крім того, можливість сканування штрихкодів сприяє точності введених даних, мінімізуючи ризик помилок, які можуть виникнути при ручному введенні інформації.

Такий функціонал робить додаток більш привабливим для користувачів, забезпечуючи їм інструмент, який спрощує повсякденні завдання та підвищує загальний рівень задоволеності від використання додатку.

Визначення станів компоненту сканера, за допомогою “useState”. Це включає змінні для перевірки дозволів на доступ до камери, для перевірки чи сканер вже сканував штрихкод

```

export const ScannerScreen = () => {
  const [hasPermission, setHasPermission] = useState(null);
  const [scanned, setScanned] = useState(false);

```

Запит дозволу використання камери:

```
useEffect(() => {  
  const getCameraPermissions = async () => {  
    const { status } = await Camera.requestCameraPermissionsAsync();  
    setHasPermission(status === 'granted');  
  };  
  getCameraPermissions();  
}, []);
```

Замість повідомлення за замовчуванням, що ваш товар було зіскановано було вирішено відображати картку товару:

```
<Image style={styles.image} source={product.img}/>  
<Text>  
  {product.name}  
</Text>  
<View style={styles.actionContainer}>  
  <Button title='Відсканувати спочатку' mode='outlined'  
style={styles.action} onPress={onRescan}/>  
  <Button title='Детальніше' mode='contained' style={styles.action}  
onPress={onDetails}/>  
</View>  
</View>  
)
```

Задля зручності подальшої розробки саму картку винесено окремим компонентом “ScanCard.js”. Додано візуальні зміни для підвищення інтуїтивної зрозумілості.

Також великої уваги було приділенню саме картці товару, при переході до неї, будь то зі списку товарів, чи результатом сканування штрихкоду. Основна складність полягає в інтеграції численних компонентів, забезпеченні відповідного стилю та макета, а також у коректній роботі з даними, отриманими з сервісної частини. картка товару містить детальну інформацію про продукт, включаючи його зображення, назву, штрихкод, опис, калорійність, країну виробництва, рейтинг, відгуки, а також можливість додавання продукту в ваш особистий список.

Налаштування стану компонентів проводимо за допомогою “UserState”, “useNavigation”, “useNavigationState”:

Зв’язок та запит до сервісної частини проекту:

```
useEffect(() => {  
  const getProductById = async () => {  
    try {  
      const response = await  
        axios.get(`${BASE_API_URLS.device}/products/${productId}`);  
  
      setProductData(response.data);  
    }  
  }  
});
```

```

    } catch (error) {
      console.error('Error fetching product:', error);
      throw error;
    }
  }

  getProductById()
}, [productId]);

```

Зображення товару відображається у верхній частині і для його реалізації використовуємо компонент “Image” з бібліотеки “expo-image”:

```
<Image source={productData.picture} style={styles.image}/>
```

Під зображенням розташовується основна інформація про товар. Це включає назву товару, його штрихкод та оцінку харчової корисності “HealthScore”:

```

</View>
<HealthScore score={productData.abc_score}/>
</View>

```

Окрему увагу потрібно приділити компоненту харчової корисності, реалізовано текстові елементи, що представляють різні рівні корисності. Кожен рівень отримує свій стиль, зокрема, перший і останній елементи мають закруглені кути. Поточний рівень “score” виділяється рамкою для інтуїтивної зрозумілості:

```

return (
  <HorizontalView style={style}>
    {Object.keys(HEALTHY_LVL5).map((key, index) => (
      <Text key={key}
        style={
          [
            styles.item,
            { backgroundColor: HEALTHY_LVL5[key].color },
            index === 0 && styles.firstItem,
            index === 3 && styles.lastItem,
            score === key && [styles.currentScore, { borderColor:
theme.colors.background }],
          ]
        }
      >{HEALTHY_LVL5[key].title}</Text>
    ))}
  </HorizontalView>
);
}

```

Більшість навігації було винесено в окремих файлах, для більш зручного використання.

Розглянемо компонент NavigationBar, який виконує роль навігаційної панелі в додатку, яка дозволяє користувачам переходити між основними екранами. Він

відображає іконки та текст для кожного з чотирьох основних розділів: "Товари", "Сканер", "Рецепти" і "Список":

Функція `handleNavigateToScreen` викликає навігацію до зазначеного екрану:

```
const handleNavigateToScreen = (screenName) => {  
  navigation.navigate(screenName);  
}
```

Відстеження видимості подій показу та приховування клавіатури, за допомогою цього ми можемо сховати навігаційну панель, коли додаток розуміє що клавіатура видима:

```
useEffect(() => {  
  const keyboardDidShowListener = Keyboard.addListener(  
    'keyboardDidShow',  
    () => {  
      setKeyboardVisible(true);  
    }  
  );  
  const keyboardDidHideListener = Keyboard.addListener(  
    'keyboardDidHide',  
    () => {  
      setKeyboardVisible(false);  
    }  
  );  
  
  return () => {  
    keyboardDidHideListener.remove();  
    keyboardDidShowListener.remove();  
  };  
}, []);
```

Перевірка чи клавіатура видима, якщо так то навігацій панель не відображається:

```
if (isKeyboardVisible) return null;
```

Зміни та оновлення активного екрану кожного разу, коли змінюється стан навігації, тобто коли переходимо з екрану на екран:

```
useEffect(() => {  
  setActiveScreen(screenName);  
}, [screenName]);
```

Висновки до другого розділу

У другому розділі було визначено ключові бізнес-вимоги, а також вимоги користувачів і персоналу, разом з функціональними та нефункціональними вимогами. Це дало змогу сформулювати конкретні цілі для подальшого проектування та розробки.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						40

Одним з кроків була побудова бази даних за допомогою SQLite, яка включає 25 таблиць, кожна з яких виконує унікальні функції та зберігає важливі дані. Цей метод дозволяє ефективно керувати інформацією, підтримувати цілісність даних та забезпечувати доступність для системи адміністрування на платформі Django, що є ядром проекту “Visor”.

Для кращого розуміння взаємодій між компонентами системи було розроблено діаграму активності, яка наочно демонструє основні процеси.

Далі було реалізовано ключові функціональні можливості додатку. Було розроблено та інтегровано різноманітні модулі, включаючи вітальний екран, авторизацію, каталог товарів, сканер штрихкодів, картку товару, навігацію та інші складові. Це дозволило створити мобільний додаток з дружнім до користувача інтерфейсом, який відповідає всім визначеним вимогам.

Варто відзначити заходи, спрямовані на забезпечення безпеки та масштабованості системи. Використання сучасних технологій і методологій дозволило створити надійний продукт, який може працювати під великим навантаженням і захищений від потенційних загроз. Цей підхід відкриває можливості для подальшого розвитку проекту “Visor” та інтеграції з іншими сервісами, значно розширюючи функціонал додатку.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ "VISOR"

3.1. Порядок встановлення та налаштування параметрів додатку

Реалізація та конфігурація параметрів додатку "Visor" вимагає роботи з кількома частинами, кожна з яких потребує індивідуального розгортання та налаштування. Для зручності уявлення цього процесу була створена графічна діаграма розгортання (рис. 3.1), що ілюструє взаємозв'язки між компонентами.

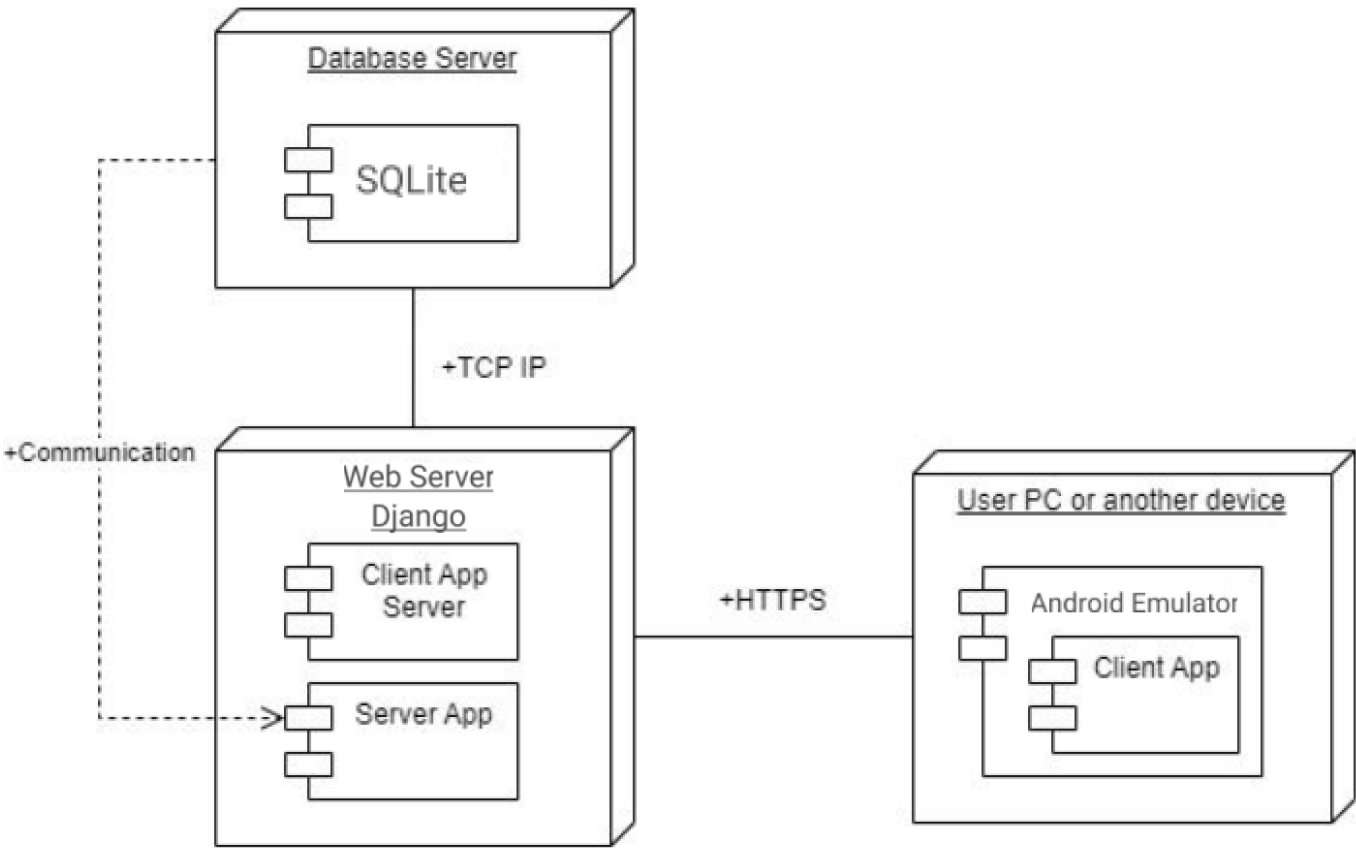


Рисунок 3.1 – Діаграма розгортання

Опис діаграми розгортання:

У діаграмі розгортання системи видно три основні вузли: сервер, сервер баз даних і комп'ютер або інший пристрій користувача. Зв'язок між комп'ютером користувача і веб-сервером забезпечується за допомогою протоколу HTTPS. У свою чергу, взаємодія між клієнтським та серверним додатками всередині сервера здійснюється за допомогою протоколу HTTP. Протоколи HTTP і HTTPS базуються на

стандарті TCP для надійної передачі даних без втрат. Комунікація між сервером баз даних і веб-сервером здійснюється за допомогою протоколу TCP/IP.

Для реалізації та конфігурації додатку необхідно виконати такі кроки:

1. Налаштування серверного середовища:

- i) Встановити і налаштувати веб-сервер Django;
- ii) Забезпечити коректну роботу серверного додатку;
- iii) Встановити всі необхідні залежності та бібліотеки для роботи серверного додатку;

2. Налаштування бази даних:

- i) Встановити і налаштувати сервер баз даних SQLite;
- ii) Налаштувати зв'язок між сервером баз даних і веб-сервером;
- iii) Виконати необхідні міграції бази даних для створення всіх необхідних таблиць і структур даних;

3. Використання Exro для налаштування клієнтського додатку:

- i) Встановити Exro CLI на комп'ютер;
- ii) Створити новий проект Exro або імпортувати існуючий;
- iii) Запустити проект на Android Studio з її власним емулятором або на фізичному пристрої;
- iv) Переконаватися, що додаток успішно взаємодіє з сервером;

Налаштувати моніторинг та підтримку системи для своєчасного виявлення та усунення проблем.

Таким чином, налаштування та конфігурація додатку "Visor" включає в себе комплекс дій, спрямованих на забезпечення коректної взаємодії між клієнтськими та серверними компонентами, забезпечення безпеки передачі даних та підтримку стабільної роботи системи в цілому. Використання Android Studio з її власним емулятором та Exro дозволяє ефективно тестувати та розгортати клієнтський додаток, забезпечуючи його функціональність і надійність.

3.2. Структура інтерфейсу системи мобільного додатку

Взаємодія користувачів з додатком починається з його відкриття та потрапляння на екран привітання (рис. 3.2). Перший слайд, що вони побачать це: Привіт, ми Visor! Далі йде стислий опис для чого саме наш додаток, ще нижче ми бачимо кнопку, за допомогою якої можна перейти до наступного слайду на цьому екрані.

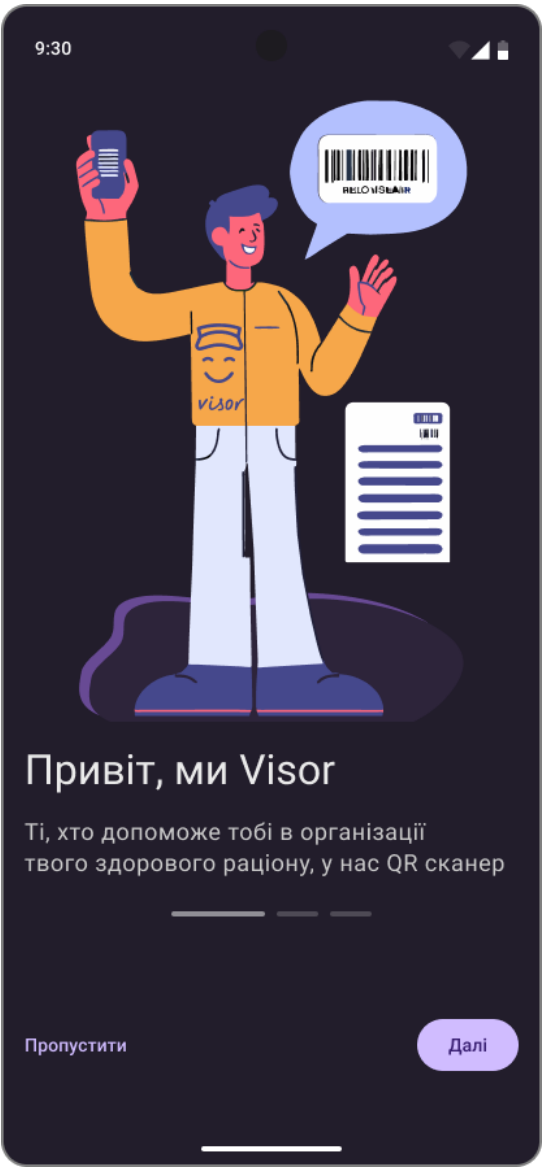


Рисунок 3.2 – Екран привітання (перший слайд)

Натиснувши кнопку далі по побачимо ще два інформаційні слайди (рис. 3.3) про мобільний додаток та велику основну кнопку для переходу до авторизації в застосунку

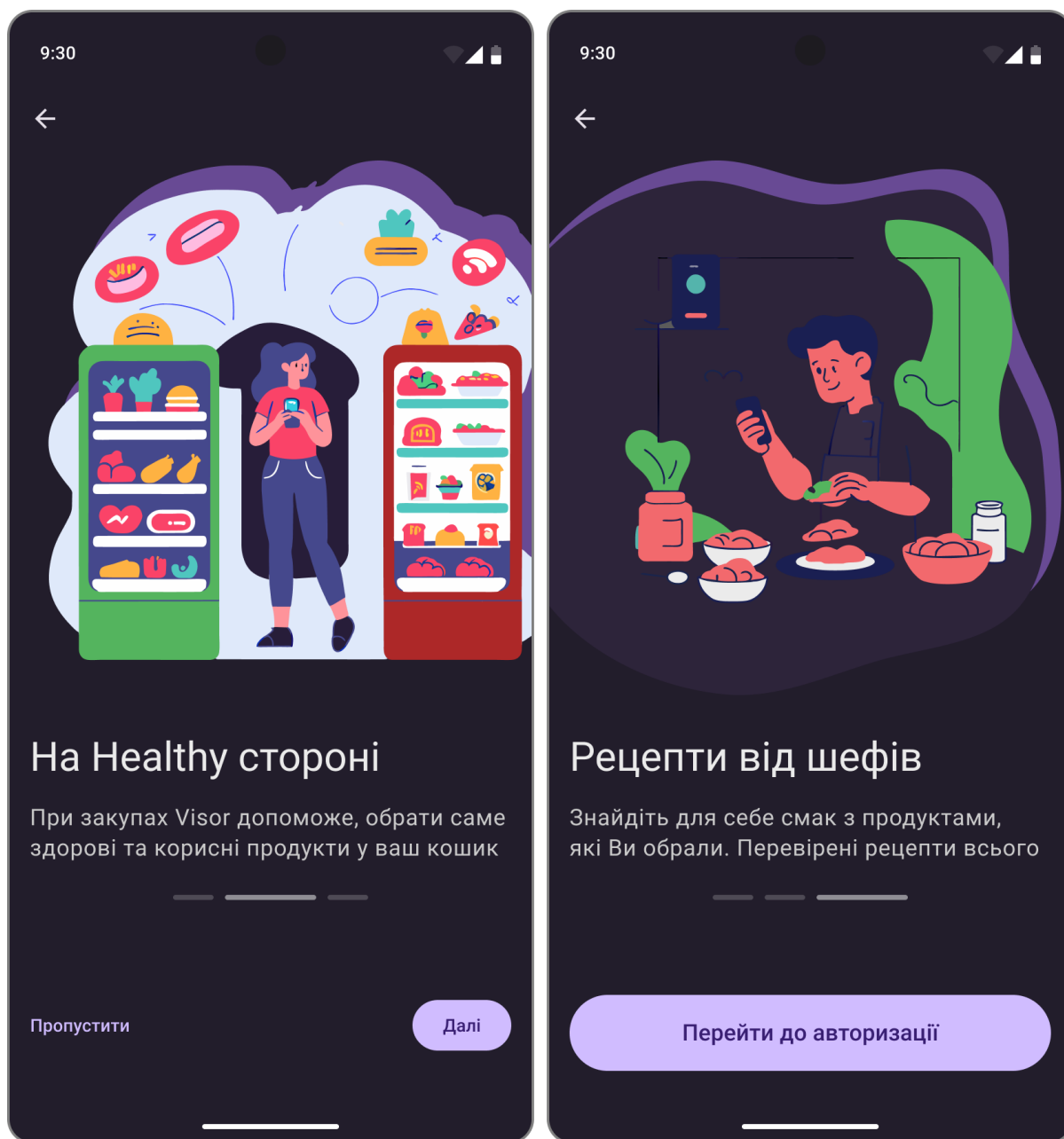


Рисунок 3.3 – Екран привітання (другий та третій слайди)

Натиснувши на кнопку “Перейти до авторизації”, користувач потрапляє на екран авторизації (рис. 3.4), де йому надається можливість пройти кастомну авторизацію. Цей екран має сучасний та стильний дизайн, що сприяє зручності використання. На екрані розташовані поля для введення логіну та пароля, які дозволяють користувачу ввести свої облікові дані. Кожне поле чітко позначене, що мінімізує можливість помилок під час введення інформації.

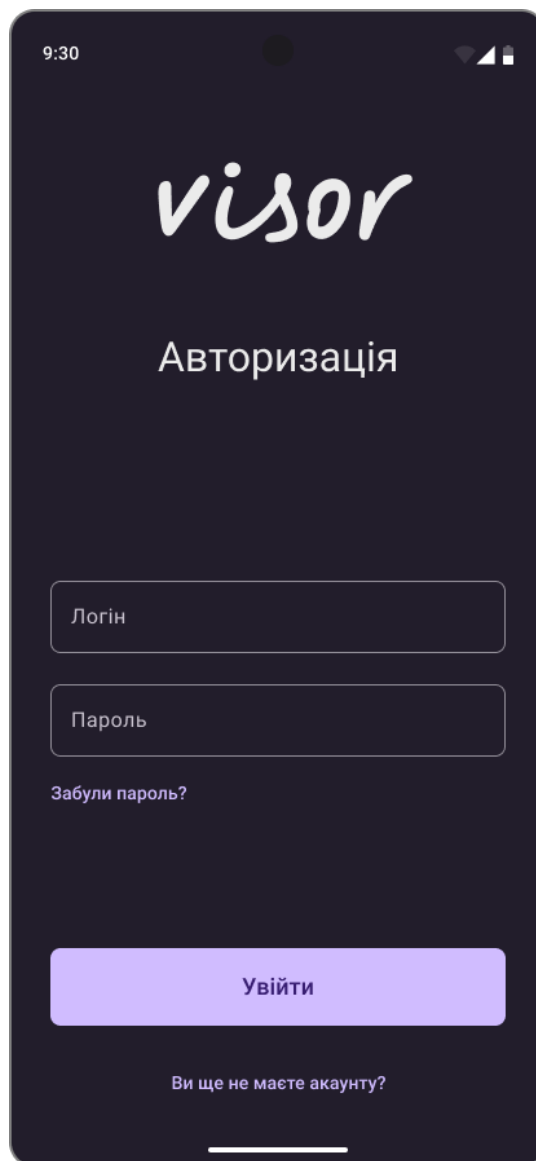


Рисунок 3.4 – Екран авторизації в обліковий запис

При умові, що користувач не має облікового запису, він може створити його, натиснувши на відповідну кнопку і перейшовши на екран реєстрації (рис. 3.5). На екрані реєстрації буде доступно заповнення всіх необхідних полів, таких як ім'я користувача, електронна пошта, пароль та повторення паролю. Після успішного введення та підтвердження цих даних новий користувач буде автоматично створений в базі даних системи і зможе здійснювати вхід у свій обліковий запис для подальших дій, що вимагають авторизації.

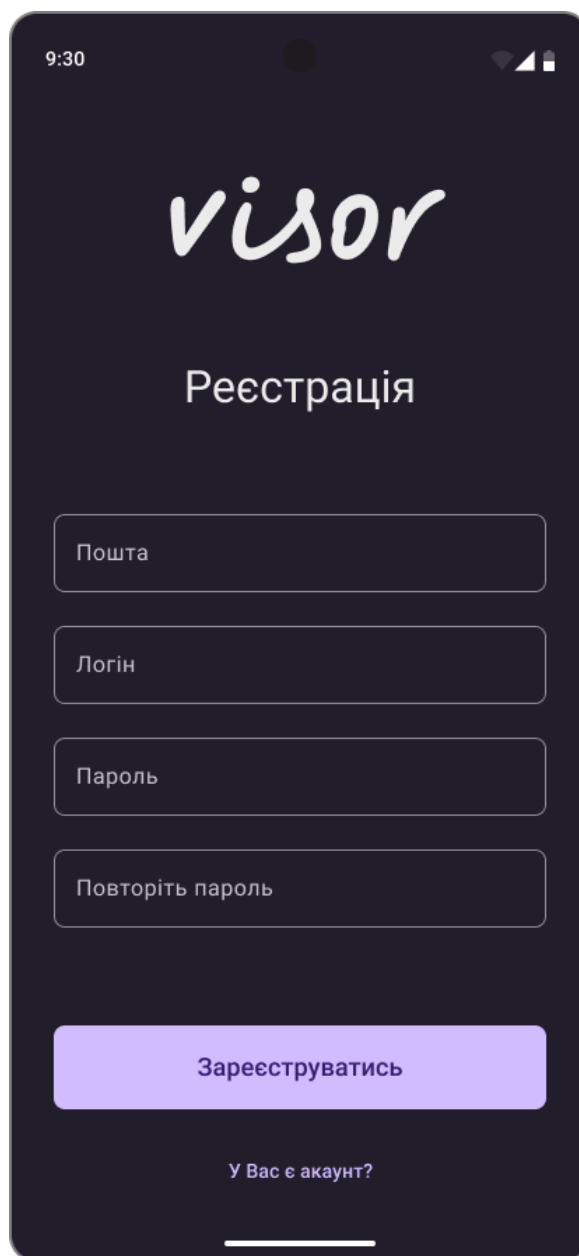


Рисунок 3.5 – Екран реєстрації нового облікового запису

Після успішної авторизації користувача в системі відбувається автоматичне перенаправлення на головний екран застосунку, який представляє собою список товарів (рис. 3.6). На цьому екрані користувач має можливість переглядати широкий асортимент товарів, що зберігаються у базі даних програми. Крім того, він може скористатися рядком пошуку для швидкого знаходження конкретного товару за його назвою або штрихкодом. Це забезпечує зручність і ефективність використання застосунку, дозволяючи швидко знаходити необхідні товари без зайвих зусиль.

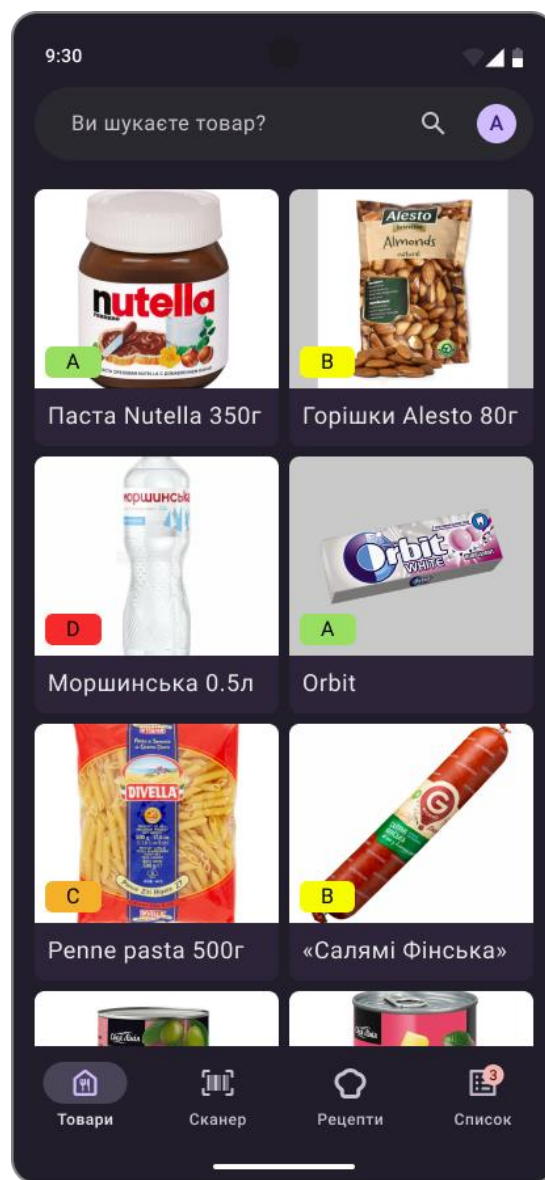


Рисунок 3.6 – Екран списку товарів

Натиснувши на товар, користувач переходить до розгорнутої картки цього товару (рис. 3.7), де він може ознайомитися з детальною інформацією про продукт. На цьому екрані представлено повний опис продукту, включаючи його калорійність та шкалу харчової корисності, яка дозволяє швидко оцінити його поживну цінність. Також користувач може побачити штрихкод товару, що забезпечує легкість його ідентифікації.

Крім цього, на картці товару є можливість залишити свою оцінку та написати відгук про продукт, що сприяє формуванню об'єктивної думки про нього. Кнопка

"Рецепти з цим продуктом" надає користувачеві доступ до рецептів, які включають даний товар, що може бути дуже корисним для тих, хто шукає ідеї для приготування страв.

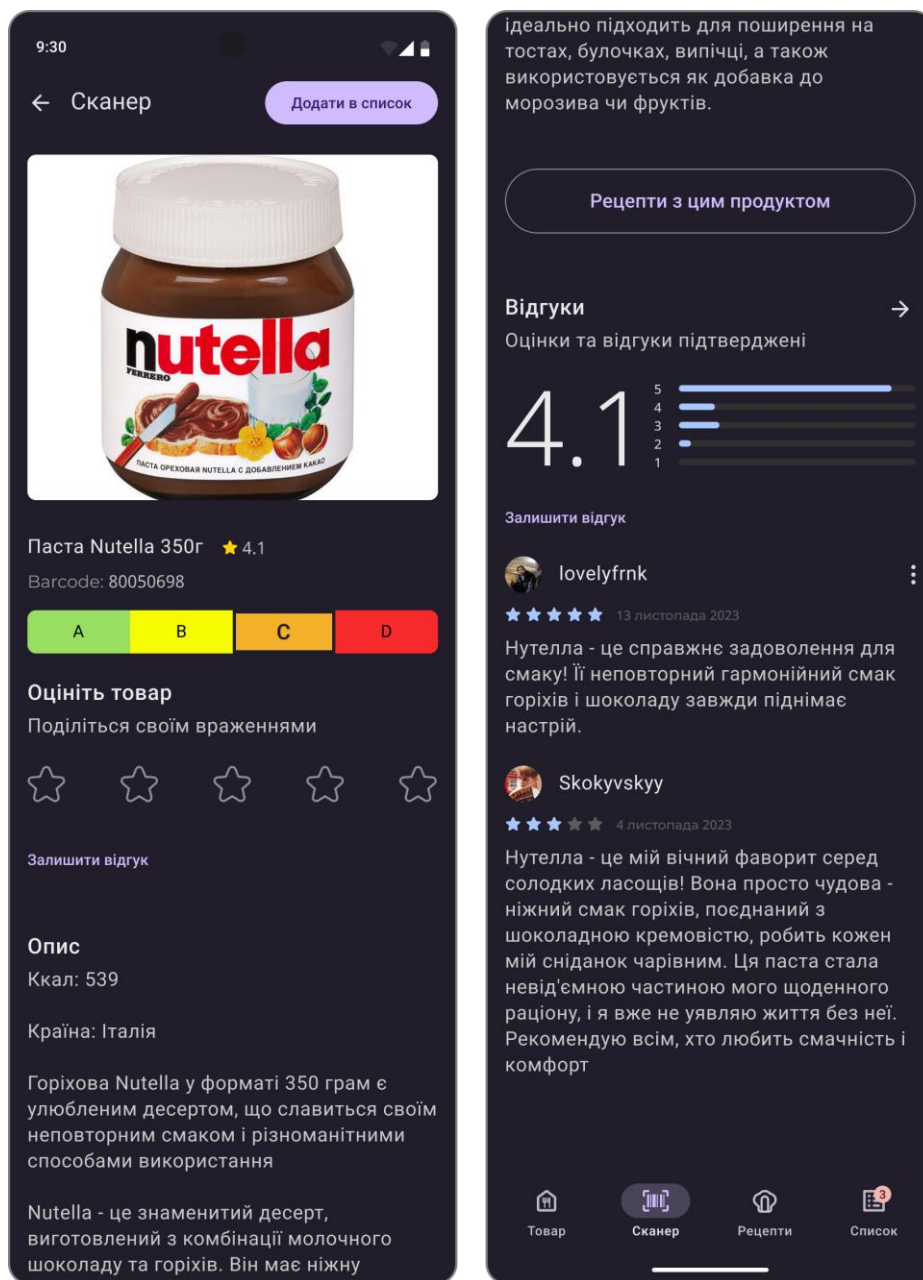


Рисунок 3.7 – Картка товару

Натиснувши на кнопку "Додати в список", назва товару автоматично додається до списку. Після переходу на екран списку (рис. 3.8), користувач отримує можливість виконувати різні дії з доданими товарами. Він може редагувати існуючі записи, змінювати їх параметри, додавати нові нотатки або побажання щодо кожного товару.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						49

Це забезпечує зручне управління списком покупок, що дозволяє користувачеві точно налаштувати свої потреби і вимоги без зайвих зусиль.

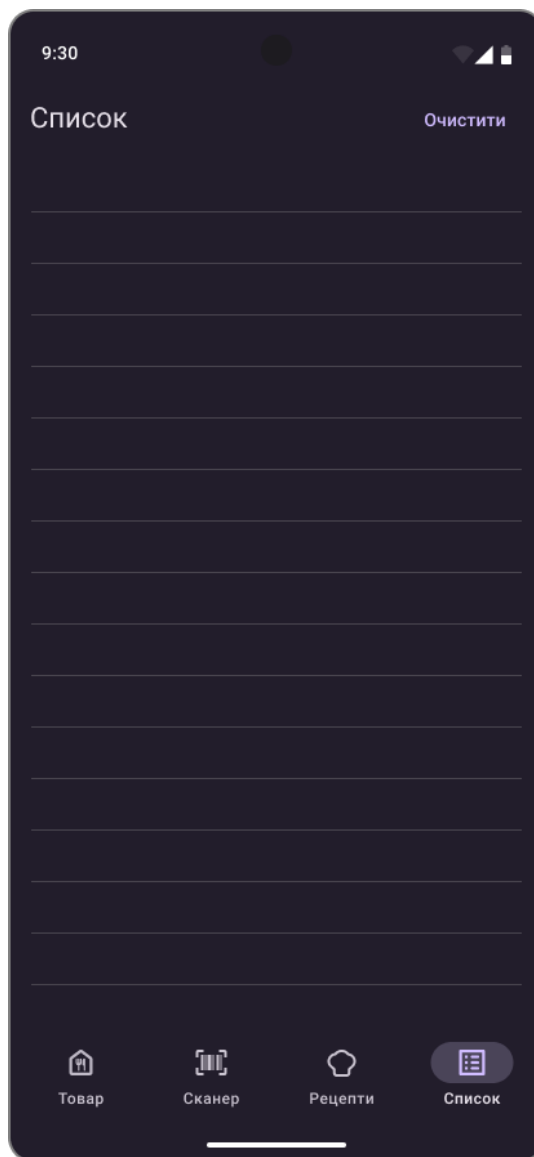


Рисунок 3.8 – Екран списку

Користувач, який переглядає картку товару (рис. 3.7), має можливість залишити свій власний відгук щодо цього продукту. Натиснувши кнопку "Залишити відгук", він переходить на новий екран для написання відгуку (рис. 3.9). На цьому екрані користувач може обрати рейтинг, використовуючи зірочки, і написати докладний опис своїх вражень і вражень щодо товару. Цей процес дозволяє користувачам не лише висловлювати свою думку про продукт, а й надавати іншим споживачам корисну інформацію для прийняття рішення під час покупки. Таким чином, функція

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						50

відгуків стає важливим елементом взаємодії між користувачами і платформою, сприяючи покращенню користувацького досвіду та відкритості в обміні інформацією.

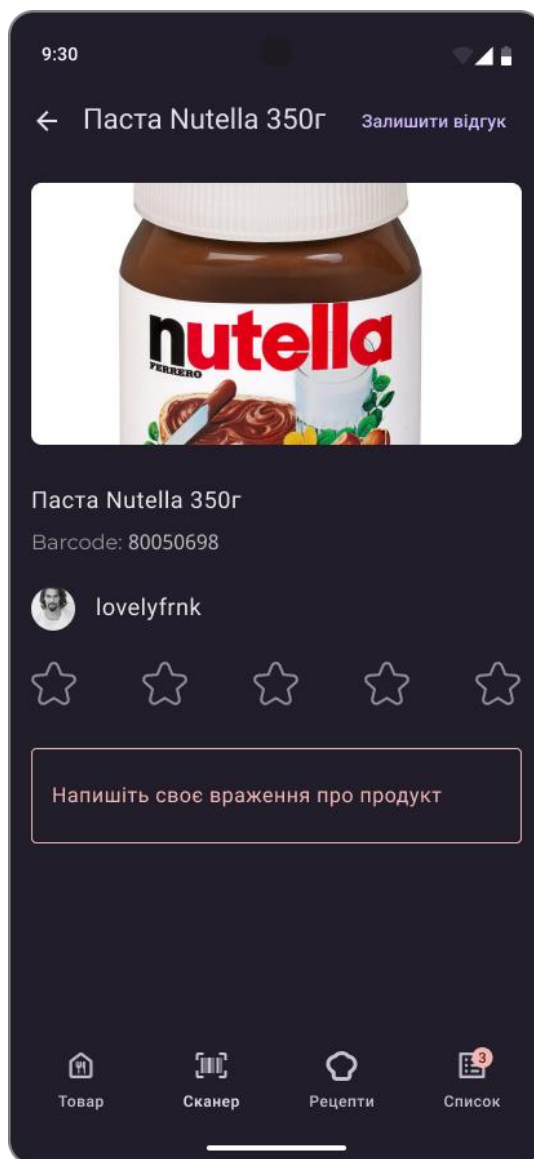


Рисунок 3.9 – Екран написання відгуку

Повернувшись до картки товару (рис. 3.7), користувачеві надана ще одна можливість, за допомогою кнопки “Подивитись рецепти з цим продуктом”, він перейде на екран рецептів (рис. 3.10), де він побачить рецепти з цим продуктом, або якщо їх немає буде виведено просто увесь список рецептів.



Рисунок 3.10 – Екран рецептів

Натиснувши на рецепт, користувач переходить до розгорнутої картки цього рецепту страви (рис. 3.11), де доступна детальна інформація про страву. На цій сторінці він може залишити свій відгук, ознайомитися з інгредієнтами, оцінити харчову корисність, ознайомитися зі способом приготування та переглянути середній бал рецепту на основі відгуків і два останніх коментарі. Цей інтерфейс дозволяє користувачам не лише вивчати рецепти, а й активно взаємодіяти з ними, ділитися своїми враженнями та думками, а також отримувати цінні відгуки від інших користувачів.

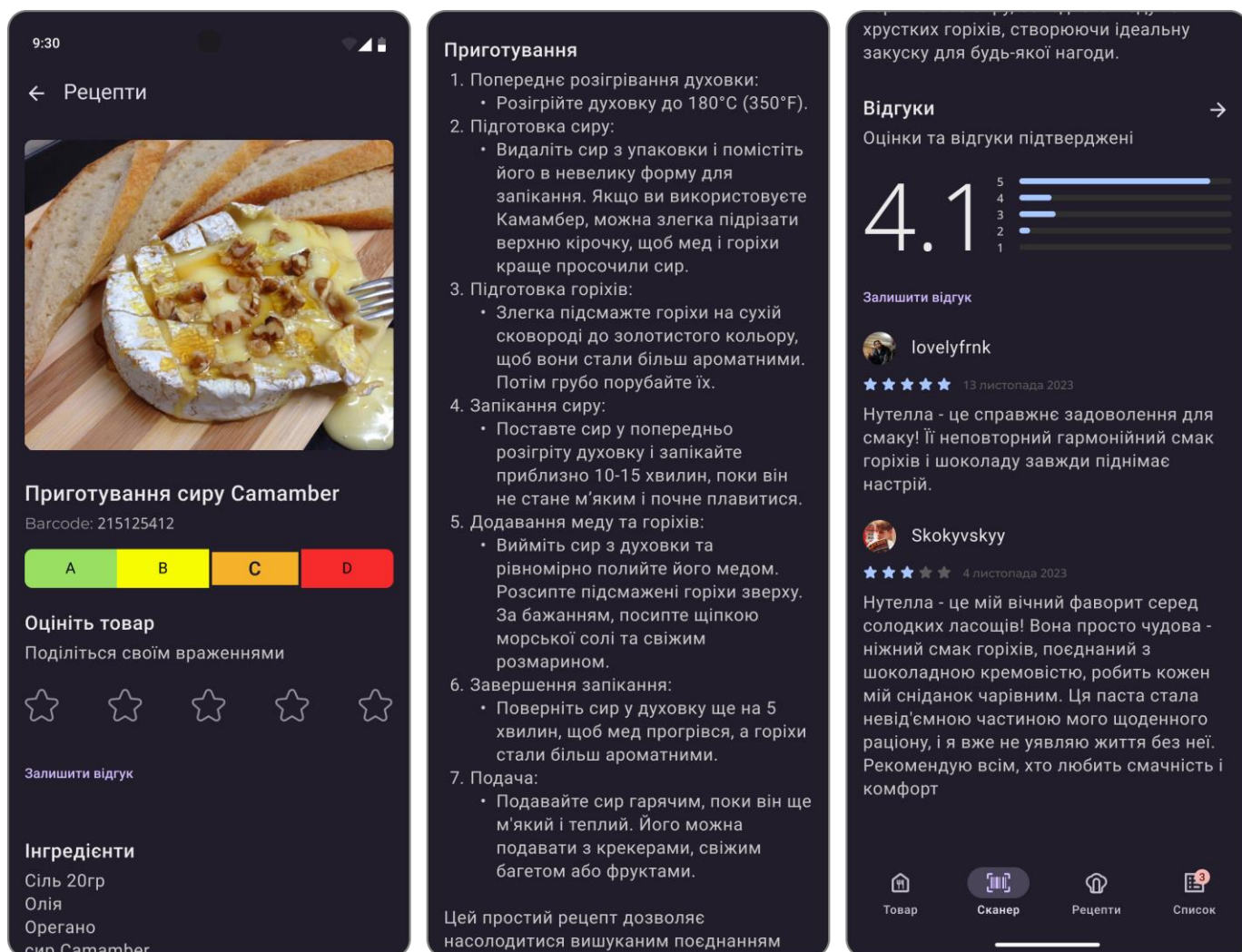


Рисунок 3.11 – Картка рецепту

За допомогою навігаційної панелі, яка розташована в додатку, користувач має можливість швидко перейти на екран сканера штрихкодів (рис. 3.12). Цей інструмент стає вельми корисним в повсякденному використанні, оскільки дозволяє миттєво здійснити пошук будь-якого товару, просто скануючи його штрихкод за допомогою камери смартфона чи іншого пристрою. Використання сканера штрихкодів значно заощаджує час користувача, оскільки не вимагає введення назви товару або пошукового запиту вручну.



Рисунок 3.12 – Екран сканеру штрихкодів

Після успішного сканування (рис. 3.13), користувач переходить на сторінку з деталізованою інформацією про товар (рис. 3.7), де відображається не лише назва і зображення товару, але і інші важливі дані, такі як опис, рейтинг, відгуки користувачів та інше. Таким чином, користувач може швидко отримати повну інформацію про товар і при необхідності продовжити інтерактивну взаємодію з додатком. Цей підхід до пошуку і отримання інформації про товари робить використання додатку більш зручним і ефективним, що позитивно впливає на користувацький досвід та задоволення від використання.

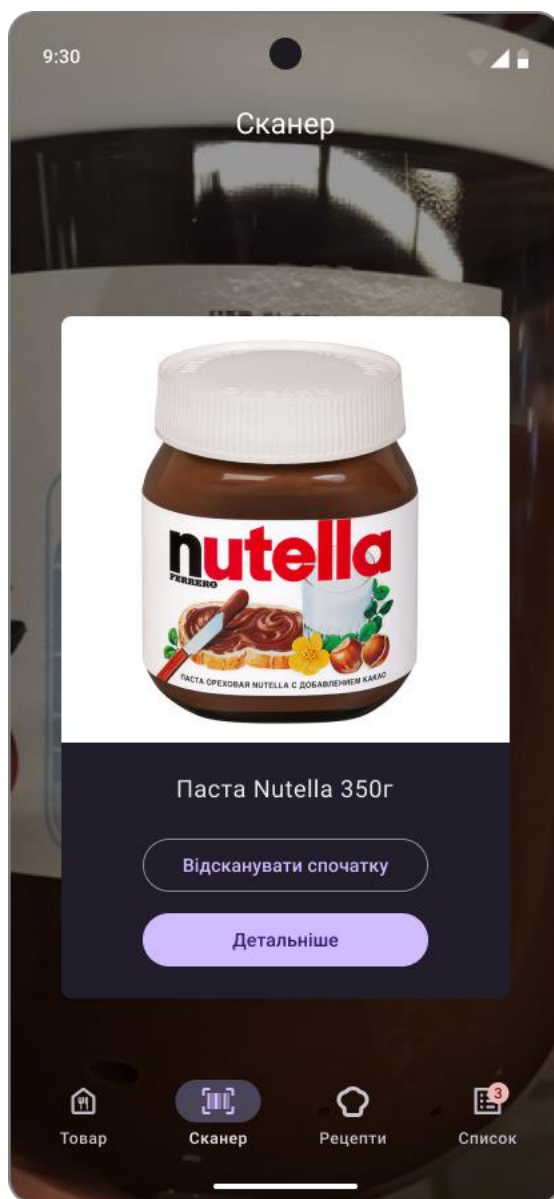


Рисунок 3.13 – Успішне сканування та відображення поп-ап картки

Адміністративна панель є вбудована в сервер Django (рис. 3.14). Після успішної авторизації у системі ми отримуємо повний доступ до всіх таблиць і даних, що дозволяє здійснювати редагування та управління інформацією.

Для надання ролі модератора необхідно звернутися до вже існуючого суперюзера. Суперюзер має право надавати додаткові повноваження існуючим користувачам через адміністративну панель Django, що робить процес управління доступом до функцій та ресурсів додатку "Visor" зручним і ефективним.

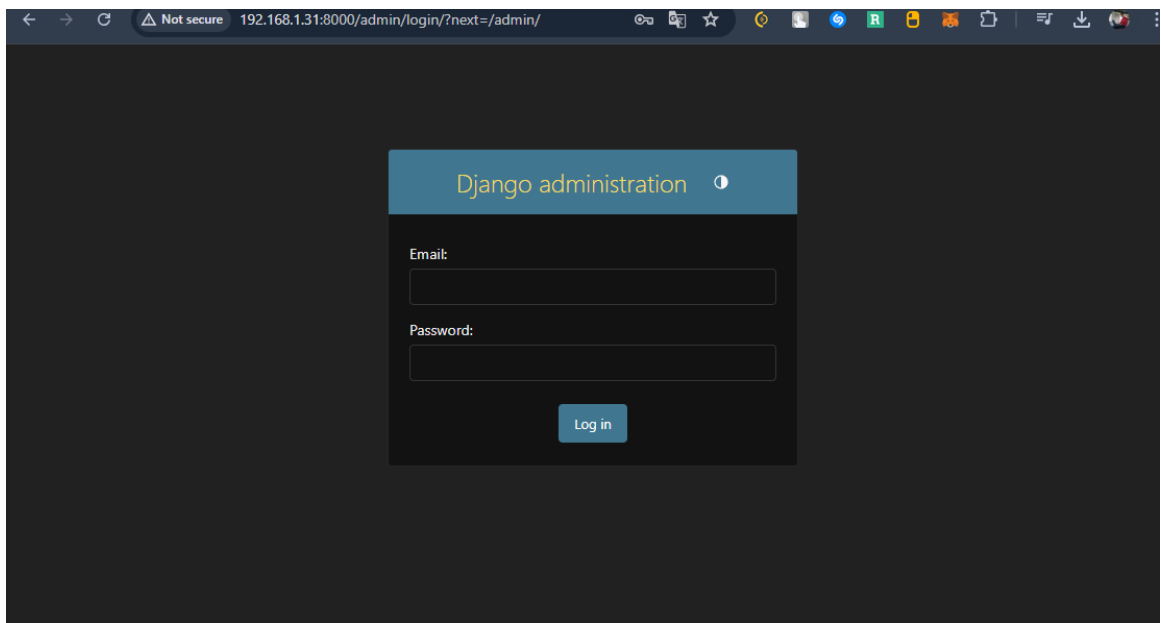


Рисунок 3.14 – Авторизація в Django

При отриманні ролі модератора, надається доступ до адміністративної панелі бази (рис. 3.15). Панель надає можливість керувати різними таблицями проекту, такими як управління користувачами, продуктами, рецептами, шкалою корисності, відгуки товарів, відгуки рецептів та інші параметрами, які впливають на функціонування та безпеку системи. Також є можливим подивитись усі останні дії з базою даних.

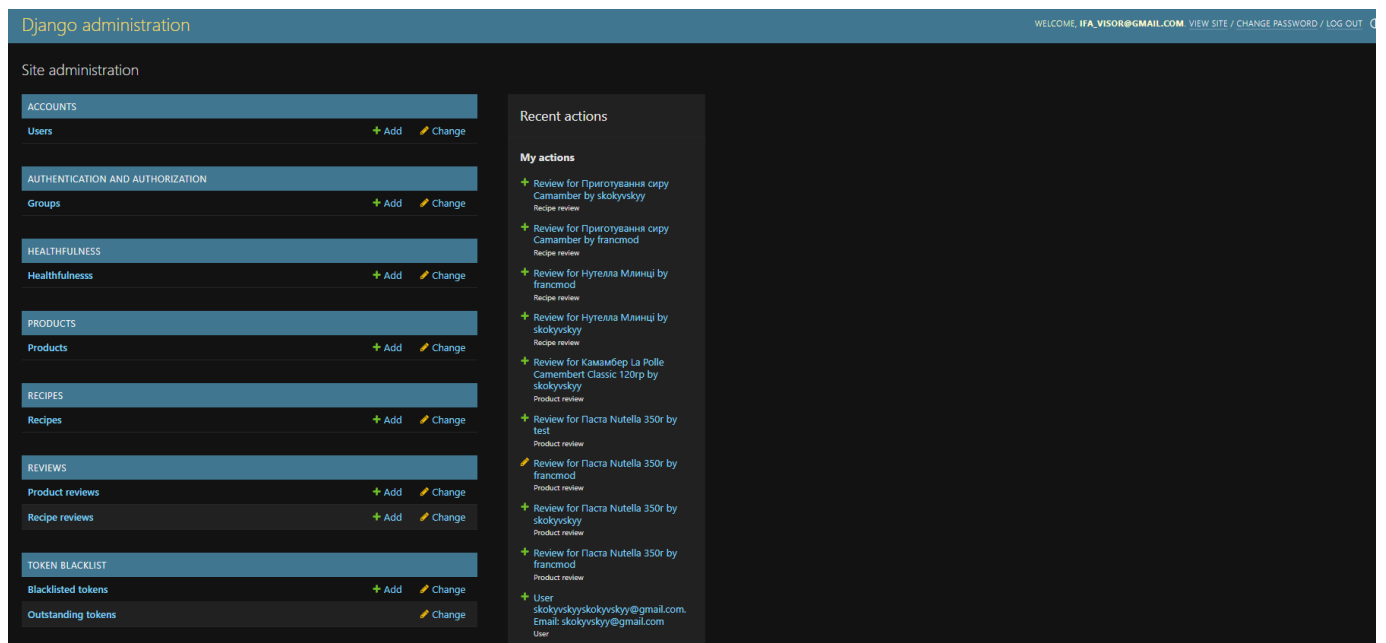


Рисунок 3.15 – Адміністративна панель Django

Новий модератор отримує повний доступ до всіх продуктів у системі (рис. 3.16). Це означає, що він має можливість переглядати, редагувати та видаляти будь-який вміст, пов'язаний з продуктами, представленими в базі даних додатку. Роль модератора наділяє його широкими повноваженнями для забезпечення точності та актуальності інформації, яка доступна іншим користувачам.

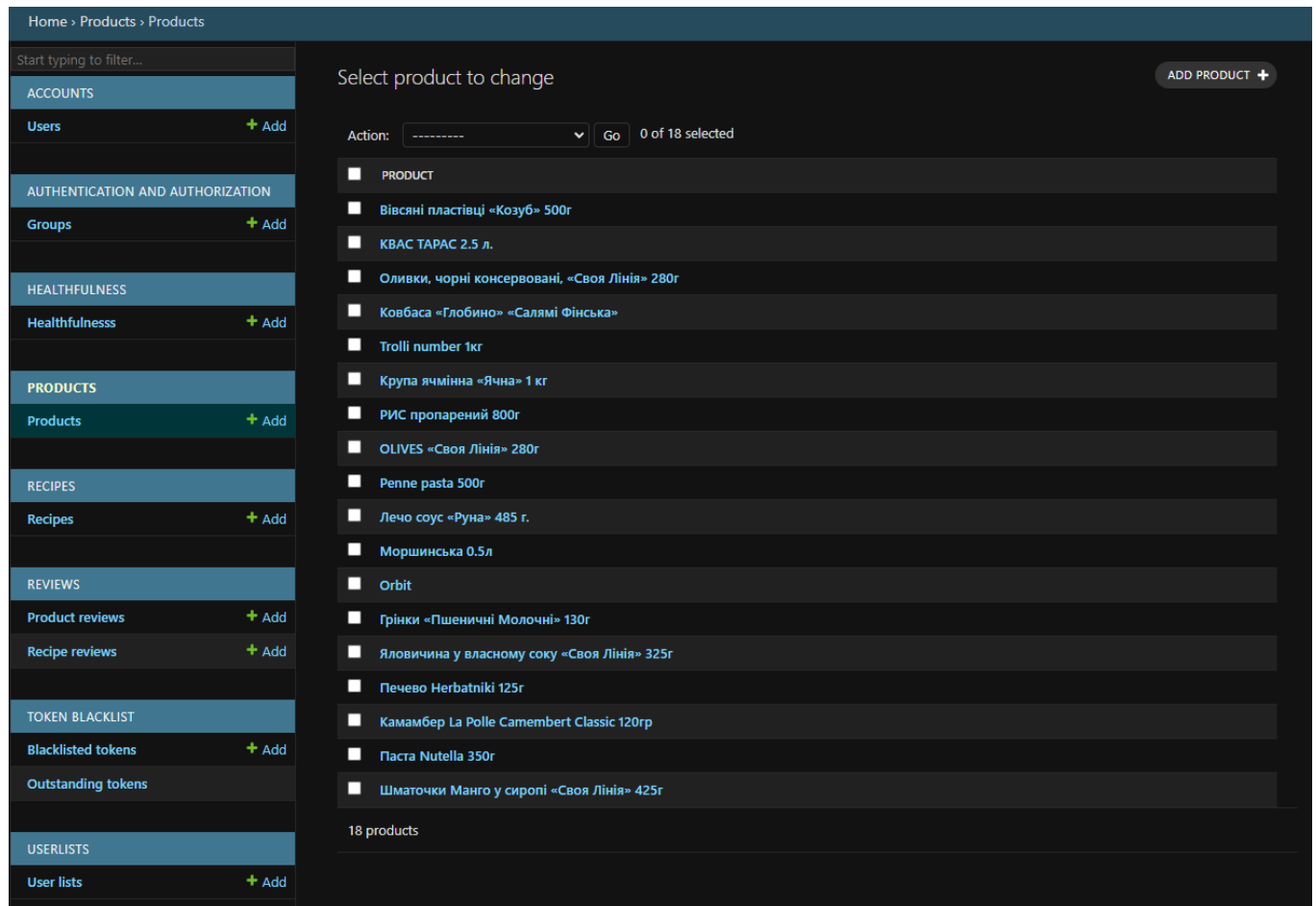


Рисунок 3.16 – Розділ продуктів

На екрані редагування (рис. 3.17) модератор може вносити зміни до описів продуктів, оновлювати їхні характеристики, додавати або видаляти зображення, а також керувати іншими аспектами вмісту, пов'язаного з продуктами. Крім того, модератор має право видаляти продукти, які більше не актуальні або не відповідають стандартам якості додатку. Це дозволяє підтримувати базу даних в належному стані та забезпечувати користувачів достовірною і корисною інформацією. Завдяки цим можливостям модератор може блискавично реагувати на будь-які зміни та потреби,

що виникають у процесі управління продуктами. Він відіграє ключову роль у підтримці бази даних в актуальному стані, що сприяє поліпшенню загального користувацького досвіду та підвищенню рівня задоволеності користувачів додатку.

Home > Products > Products > Яловичина у власному соку «Своя Лінія» 325г

Start typing to filter...

ACCOUNTS

UsersAdd

AUTHENTICATION AND AUTHORIZATION

GroupsAdd

HEALTHFULNESS

HealthfulnessAdd

PRODUCTS

ProductsAdd

RECIPES

RecipesAdd

REVIEWS

Product reviewsAdd

Recipe reviewsAdd

TOKEN BLACKLIST

Blacklisted tokensAdd

Outstanding tokens

USERLISTS

User listsAdd

Change product

Яловичина у власному соку «Своя Лінія» 325г

History

NAME

Яловичина у власному соку «Своя Лінія» 32

DESCRIPTION

Яловичина у власному соку "Своя Лінія" — це високоякісний м'ясний продукт, виготовлений із доброї яловичини, без додавання штучних консервантів і барвників. Яловичина консервується у власному соку, що зберігає її натуральний смак, соковитість та корисні властивості. Продукт готовий до вживання. Його можна використовувати як самостійну страву, додавати до супів, каш, салатів або використовувати для приготування інших кулінарних шедеврів. Це зручний варіант для швидкого приготування ситних та поживних страв.

BARCODE

2999300061225

PICTURE

Currently: [products/images/1f6c6b6-f3a8-44d2-8a84-fb2796f2fb0/catalog_product_main_1228.png](#)
Change: [Choose File](#) No file chosen

PRODUCER

Своя Лінія

CALORIES

3005

ABC SCORE

D

SAVE

Save and add another

Save and continue editing

Delete

Рисунок 3.17 – Картка певного продукту та можливості дій з ним

Усі дії які виконуються над картою, записуються в журнал історії цієї картки (рис. 3.18), що дає змогу детально відслідковувати усі дії модераторі.

Home > Products > Products > Яловичина у власному соку «Своя Лінія» 325г > History

Start typing to filter...

ACCOUNTS

Users

+ Add

AUTHENTICATION AND AUTHORIZATION

Groups

+ Add

HEALTHFULNESS

Healthfulness

+ Add

PRODUCTS

Products

+ Add

Change history: Яловичина у власному соку «Своя Лінія» 325г

DATE/TIME	USER	ACTION
June 21, 2024, 8:26 a.m.	ifa_visor@gmail.com	Added.
June 21, 2024, 9:09 a.m.	ifa_visor@gmail.com	Changed Calories.

2 entries

Рисунок 3.18 – Історія змінень картки товару

Модератор має функціонал для управління товарами, включаючи додавання та видалення (рис. 3.19). У картці товару розміщені чотири кнопки, які забезпечують зручний доступ до редагування, додавання та видалення поточних товарів (рис. 3.20).

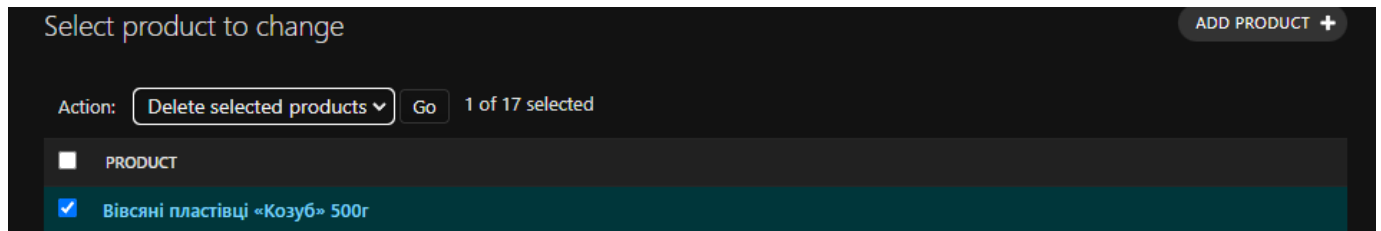


Рисунок 3.19 – Кнопка додавання та видалення товару

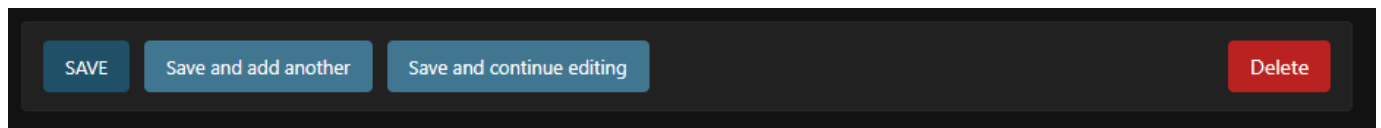


Рисунок 3.20 – Кнопки адміністрування в картці товару

Після ініціювання видалення товару модератор повинен підтвердити свої дії, повторно натиснувши кнопку "видалити" або скасувати дію (рис. 3.21). Після цього він отримує повідомлення про успішне видалення товару (рис. 3.22). Ця функція дозволяє ефективно керувати вмістом додатку, швидко оновлюючи застарілу або неправдиву інформацію.

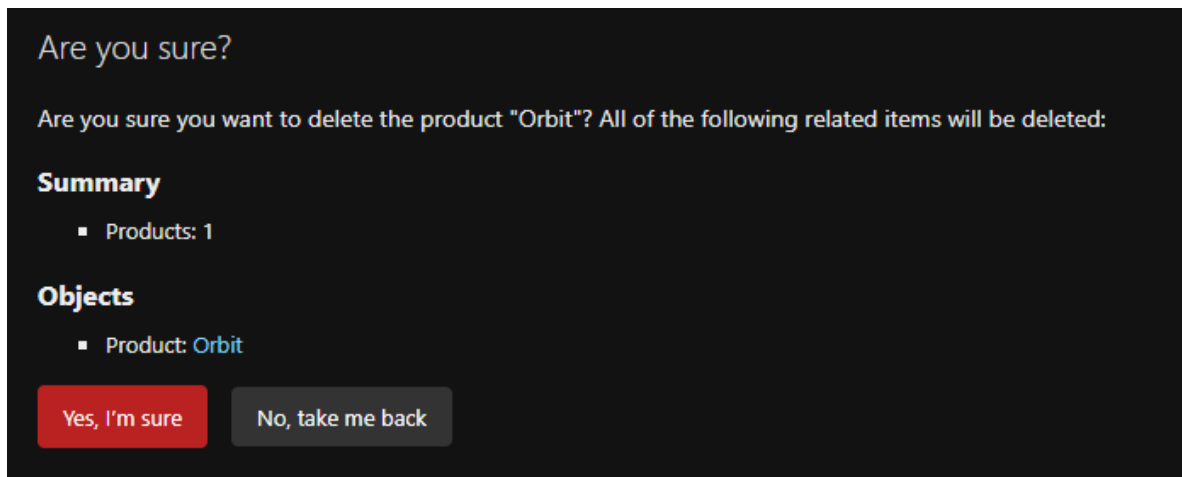


Рисунок 3.21 – Підтвердження видалення товару

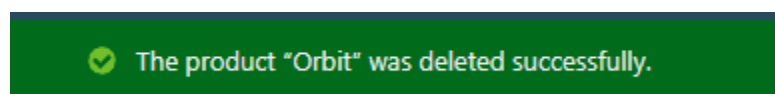


Рисунок 3.22 – Повідомлення про успішне видалення товару

Такий самий функціонал притаманний для таблиці рецептів. Модератор має можливість повного контролю та наповнення контентом застосунку (рис. 3.23).

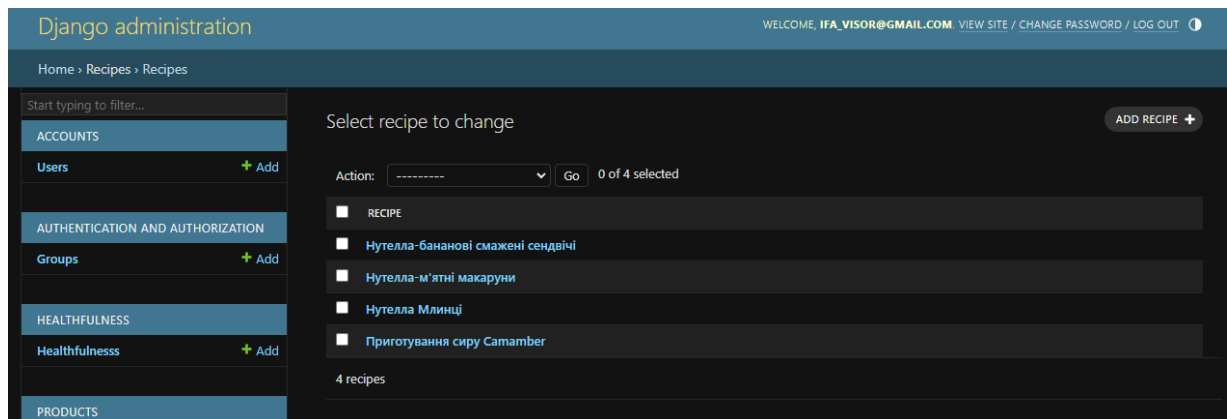


Рисунок 3.23 – Таблиця рецептів

Через адміністративну панель модератор має можливість переглядати всі відгуки, а також визначати, до якого продукту або рецепту вони належать (рис. 3.24). Це дозволяє йому фільтрувати та модерувати невідповідні або порушуючі правила платформи відгуки. Завдяки цьому модератор може ефективно управляти відгуками і видаляти їх за необхідності.

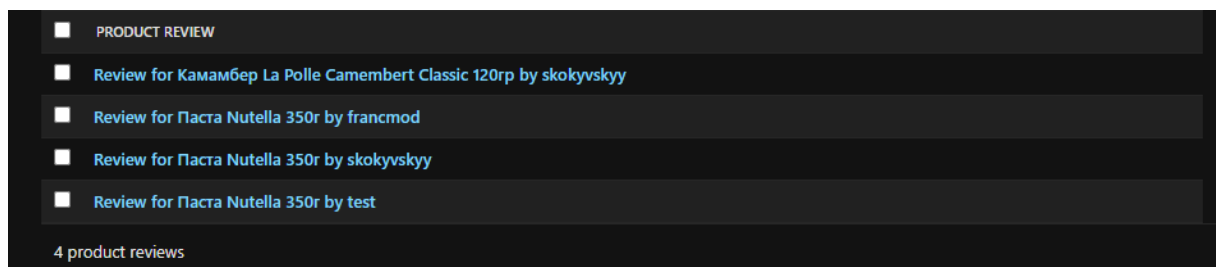


Рисунок 3.24 – Таблиця відгуків на продукт

Супер користувач та модератор мають розширені привілеї в системі, що надають їм доступ до списку всіх зареєстрованих користувачів платформи (рис. 3.25). Це дозволяє їм не лише переглядати інформацію про облікові записи, але й виконувати різноманітні дії з управління ними.

Такий розширений доступ сприяє підтримці ефективної роботи додатку, оскільки супер користувачі та модератори можуть швидко вносити необхідні зміни та забезпечувати дотримання встановлених стандартів і політик. Завдяки цим привілеям, вони можуть оперативно реагувати на будь-які проблеми або запити від користувачів,

що значно покращує загальну якість обслуговування та задоволення від використання платформи.

Select user to change

ADD USER +

Action:

Go

 0 of 4 selected

☐

 USER

☐

 User francmodfrancwanicki@gmail.com. Email: francwanicki@gmail.com

☐

 User skokyvskyskokysky@gmail.com. Email: skokyvsky@gmail.com

☐

 User ifaifa_visor@gmail.com. Email: ifa_visor@gmail.com

☐

 User testtest1@gmail.com. Email: test1@gmail.com

4 USERS

Рисунок 3.25 – Список користувачів

За необхідності, адміністратор має можливість вносити зміни до особистих даних користувачів, що дозволяє коригувати інформацію, таку як ім’я, електронна адреса, контактні дані та інше (рис. 3.26). Таким чином, роль супер користувача та модератора є ключовою для забезпечення належного управління користувачами, а їхні розширені можливості сприяють підтриманню високих стандартів якості та безпеки в роботі додатку.

Change user

HISTORY

User testtest1@gmail.com. Email: test1@gmail.com

Last login:

Date: Today

Time: Now

Note: You are 2 hours ahead of server time.

Groups:

+

The groups this user belongs to. A user will get all permissions granted to each of their groups. Hold down "Control", or "Command" on a Mac, to select more than one.

User permissions:

Accounts | user | Can add user

Accounts | user | Can change user

Accounts | user | Can delete user

Accounts | user | Can view user

Administration | log entry | Can add log entry

Administration | log entry | Can change log entry

Administration | log entry | Can delete log entry

Administration | log entry | Can view log entry

Specific permissions for this user. Hold down "Control", or "Command" on a Mac, to select more than one.

First name:

Last name:

Date joined:

Date:

2024-06-21

 Today

Time:

08:15:57

 Now

Note: You are 2 hours ahead of server time.

Username:

test

Avatar url:

Email:

test1@gmail.com

Password:

pbkdf2_sha1ohcp/U3OMXplgw=25687200005

☒ Is active

☐ Is staff

☐ Is superuser

User list:

+

✕

SAVE

Save and add another

Save and continue editing

Delete

Рисунок 3.26 – Форма для редагування даних користувачів

3.3. Тестування роботи мобільного додатку

Тестування роботи мобільного додатку є важливим етапом в процесі його розробки, спрямованим на забезпечення високої якості функціонування та надійності. Тестування допомагає виявити та виправити потенційні помилки, забезпечуючи відповідність вимогам та очікуванням користувачів.

Важливо проводити тестування під час розробки додатку, щоб виявляти та виправляти помилки на ранніх етапах, забезпечуючи стабільність роботи. Крім того, після завершення розробки необхідно провести повне тестування всього функціоналу, щоб переконатися, що всі компоненти працюють коректно та відповідають вимогам користувачів. Після виявлення та виправлення помилок додаток піддається повторному тестуванню, що гарантує надійність та відсутність нових проблем.

Для тестування мобільного додатку “Visor” було обрано мануальне тестування, оскільки цей додаток є не дуже масштабним, то така техніка дозволить швидко виявити та виправити більшість проблем, а автоматизація може потребувати більше часу та ресурсів. Мануальне тестування передбачає ручне виконання тестових сценаріїв без використання автоматизованих інструментів. Це дає змогу адаптувати тестування до змін, що виникають під час розробки та ретельно перевірити кожну функцію додатку.

В результаті тестування виявлено недоліки в додатку та сервері:

Передумови: Модератор бажає додати новий товар в застосунок.

Кроки:

1. Модератор входить до адміністративної панелі Django;
2. Модератор натискає на кнопку “Add product”;
3. Модератор вводить назву продукту у відповідне поле;
4. Модератор вводить опис продукту у відповідне поле;
5. Модератор вводить відповідний штрихкод продукту;
6. Модератор додає фото продукту;

7. Модератор вводить виробника продукту;
8. Модератор вводить кількість калорій поточного продукту;
9. Модератор вибирає категорію шкали харчової корисності;
10. Модератор натискає кнопку “Save”;

Очікуваний результат: Продукт успішно додається та відображається на екрані списку товарів з мобільного телефону.

Фактичний результат: Продукт доданий та потрапив до бази даних. На емуляторі продукт відображається на екрані списку товарів. На мобільному телефоні екран списку товарів пустий і не відображає цей продукт.

У процесі розробки функціоналу додатку було створено набір тест-кейсів, призначених для перевірки правильності функціонування його основних можливостей.

1. Реєстрація нового користувача:

Передумови: Користувач бажає створити новий обліковий запис.

Кроки:

- Користувач переходить до екрану з реєстрацією;
- Користувач вводить свій логін, електронну пошту, пароль та повторює введений пароль;
- Користувач натискає кнопку “Зареєструватись”;

Очікуваний результат: Користувач успішно реєструється в системі та зберігається у сесії зі своїм унікальним токеном.

Фактичний результат: Користувач успішно реєструється в системі та зберігається у сесії зі своїм унікальним токеном.

2. Сканування штрихкода товару:

Передумови: Користувач бажає відсканувати штрихкод товару, швидко побачити основну інформацію про нього та відгуки інших користувачів.

Кроки:

- Користувач переходить до екрану сканеру штрихкодів;

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						63

— Користувач наводить камеру на штрихкод продукту;

Очікуваний результат: Штрих код успішно зчитується, даний товар є в базі застосунку “Visor”, з’являється поп-ап вікно з демонстрацією продукту, де в подальшому користувач може перейти до повної картки товару, або відсканувати інший продукт.

Фактичний результат: Штрих код успішно зчитується, даний товар є в базі застосунку “Visor”, з’являється поп-ап вікно з демонстрацією продукту, де в подальшому користувач може перейти до повної картки товару, або відсканувати інший продукт.

3. Додавання нового відгуку:

Передумови: Користувач бажає залишити відгук про товар.

Кроки:

- Користувач гортає екран списку товарів;
- Користувач заходить до повної картки певного товару;
- Користувач натискає на зірки або на кнопку “Залишити відгук”;
- Користувач переходить на екран написання відгуків;
- Користувач вводить текст відгуку, вибирає кількість зірок та натискає кнопку “Залишити відгук”;

Очікуваний результат: Відгук успішно додається до системи та відображається.

Фактичний результат: Відгук успішно додається до системи та відображається.

4. Перегляд списку рецептів:

Передумови: Користувач бажає переглянути рецепти страв представлених застосунком.

Кроки:

- Користувач переходить на екран рецептів;
- Користувач переглядає список рецептів;

Очікуваний результат: Користувач бачить список рецептів, з можливістю перейти до повної картки цього рецепту.

Фактичний результат: Користувач бачить список рецептів, з можливістю перейти до повної картки цього рецепту.

5. Пошук товару за допомогою пошукового рядка

Передумови: Користувач бажає швидко знайти товар за його назвою.

Кроки:

- Користувач натискає на пошуковий рядок;
- Відкривається клавіатура зникає навігаційна панель;
- Користувач вводить назву товару;

Очікуваний результат: Формується живий список товарів з назвою яку ввів користувач.

Фактичний результат: Формується живий список товарів з назвою яку ввів користувач.

Висновки до третього розділу

У третьому розділі було ретельно проаналізовано всі аспекти розробки мобільного додатку, спрямованого на здорове харчування. Була створена ілюстративна схема, що відображає архітектурні рішення системи. Далі було детально розглянуто інтерфейс користувача, включаючи елементи навігації, різноманітні екрани та функції для користувачів та адміністраторів. Особливий акцент був зроблений на можливості користувачів переглядати відгуки, оглядати список продуктів і рецептів, використовувати сканер штрихкодів та взаємодіяти з різними функціональними модулями платформи.

Крім того, розглянуто панель адміністратора, що включає управлінський функціонал для керування продуктами, рецептами, відгуками та користувачами.

Після цього проведено тестування додатку для виявлення та виправлення можливих недоліків, що дозволило покращити загальну працездатність системи.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						65

Також було представлено кілька прикладів тест-кейсів для перевірки основних функціональних можливостей, включаючи авторизацію, перегляд товарів, роботу сканера штрихкодів, додавання нових продуктів і пошук товарів за допомогою пошукової функції.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						66

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було досліджено та реалізовано мобільний додаток «“Visor”: комплексний підхід до здорового харчування, аналіз складу продуктів, рецепти». У рамках кваліфікаційної роботи було проведено глибокий аналіз вимог до програмного забезпечення та предметної галузі. Визначені завдання і етапи реалізації, здійснено порівняльний аналіз схожих додатків. На основі цього аналізу була обрана оптимальна архітектура системи, що забезпечила її стабільність та ефективність.

Під час детального аналізу та розробки різних аспектів системи, включаючи вимоги користувачів та персоналу, було створено відповідну структуру системи. Особлива увага приділялася розробці інтерфейсу користувача, який має бути зручним та інтуїтивно зрозумілим. Для кращого розуміння процесів була створена діаграма взаємодій між компонентами системи, що відображає можливості для різних типів користувачів, включаючи авторизованих користувачів, не авторизованих користувачів та адміністраторів.

Основний акцент під час дослідження було зроблено на розгляді різних аспектів розробки мобільного додатку, описано інтерфейс користувача та ролі користувачів та адміністраторів. В результаті було успішно спроектовано та реалізовано систему мобільного додатку "Visor", яка відповідає всім вимогам продукту. Система забезпечує не лише зручний інтерфейс для кінцевих користувачів, але й стабільну та ефективну роботу, з можливістю майбутнього розширення та розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fooducate [Електронний ресурс] - Режим доступу до ресурсу:
<https://www.fooducate.com>
2. Lifesum [Електронний ресурс] - Режим доступу до ресурсу:
<https://lifesum.com>
3. Cronometer [Електронний ресурс] - Режим доступу до ресурсу:
<https://cronometer.com>
4. Expo [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.expo.dev>
5. React Native [Електронний ресурс] - Режим доступу до ресурсу:
<https://reactnative.dev/docs/getting-started>
6. React Native Paper [Електронний ресурс] - Режим доступу до ресурсу:
<https://callstack.github.io/react-native-paper/docs/components/ActivityIndicator>
7. Python [Електронний ресурс] - Режим доступу до ресурсу:
<https://www.python.org/doc>
8. Python / Setup [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.python.org/release/3.12.4/using/index.html>
9. Python / Modules [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.python.org/release/3.12.4/installing/index.html>
10. Python / Api [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.python.org/release/3.12.4/c-api/apiabiversion.html>
11. Android Studio [Електронний ресурс] - Режим доступу до ресурсу:
<https://developer.android.com/build>
12. Django / Documentation [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.djangoproject.com/en/5.0>
13. Django / Forms [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.djangoproject.com/en/5.0/topics/forms/>

14. Django / Admin site [Электронный ресурс] - Режим доступа до ресурсу:
<https://docs.djangoproject.com/en/5.0/ref/contrib/admin/>
15. Django / Validators [Электронный ресурс] - Режим доступа до ресурсу:
<https://docs.djangoproject.com/en/5.0/ref/validators/>
16. Django / Customizing authentication [Электронный ресурс] - Режим доступа до ресурсу:
<https://docs.djangoproject.com/en/5.0/topics/auth/customizing/>
17. PyCharm [Электронный ресурс] - Режим доступа до ресурсу:
<https://www.jetbrains.com/help/pycharm/getting-started.html>
18. SQLite [Электронный ресурс] - Режим доступа до ресурсу:
<https://sqlite.org/docs.html>
19. Django Rest Framework [Электронный ресурс] - Режим доступа до ресурсу:
<https://www.django-rest-framework.org/tutorial/quickstart>
20. Django Rest Framework / Serializers [Электронный ресурс] - Режим доступа до ресурсу:
<https://www.django-rest-framework.org/api-guide/fields>
21. Akshat Paul, Abhishek Nalwaya. React Native for Mobile Development, USA, 2019. 237 p.
22. Samuel Dauzon, Aidas Bendoraitis, Arun Ravindran. Django: Web Development with Python. Birmingham, United Kingdom, 2016. 717 p.
23. Neil Smyth. Android Studio 3.3 Development Essentials - Android 9 Edition. Cary, USA, 2019. 787p.
24. Cybellium Ltd. Mastering React Native. Frankfurt, Germany, 2023. 207 p.
25. Sunny Kumar Aditya, Vikash Kumar Karn. Android SQLite Essentials. Birmingham, United Kingdom, 2014. 98 p.

ДОДАТКИ

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						70

```

import uuid

from django.contrib.auth.models import AbstractUser
from django.db import models

from .managers import CustomUserManager
from .validators import validate_username, validate_password_strength

class CustomUser(AbstractUser):
    id = models.UUIDField(default=uuid.uuid4, editable=False, unique=True,
primary_key=True)
    username = models.CharField(max_length=50, unique=True,
validators=[validate_username])
    avatar_url = models.URLField(max_length=255, blank=True)
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=128,
validators=[validate_password_strength])
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)
    user_list = models.OneToOneField(
        'userlists.UserList',
        related_name='account',
        blank=True, null=True,
        on_delete=models.SET_NULL
    )

    objects = CustomUserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['username']

    def __str__(self):
        return f'User {self.username}{self.email}. Email: {self.email}'

import uuid

from django.db import models

class Healthfulness(models.Model):
    LEVEL_CHOICES = [
        ('A', 'A'),
        ('B', 'B'),
        ('C', 'C'),
        ('D', 'D'),
    ]

    id = models.UUIDField(default=uuid.uuid4, editable=False, unique=True,
primary_key=True)
    lvl = models.CharField(max_length=1, choices=LEVEL_CHOICES)

    def __str__(self):
        return self.lvl

import uuid

```

```

from django.db import models

def products_image_path(instance: models.Model, filename: str) -> str:
    return f'products/images/{instance.id}/{filename}'

class Product(models.Model):
    id = models.UUIDField(default=uuid.uuid4, editable=False, unique=True,
primary_key=True)
    name = models.CharField(max_length=255)
    description = models.TextField()
    barcode = models.CharField(max_length=255)
    picture = models.ImageField(upload_to=products_image_path)
    producer = models.CharField(max_length=255)
    calories = models.IntegerField()
    abc_score = models.ForeignKey(
        'healthfulness.Healthfulness',
        on_delete=models.PROTECT,
        blank=False,
        related_name='products'
    )

    def __str__(self):
        return self.name

import uuid

from django.db import models

def recipes_image_path(instance: models.Model, filename: str) -> str:
    return f'recipes/images/{instance.id}/{filename}'

class Recipe(models.Model):
    id = models.UUIDField(default=uuid.uuid4, editable=False, unique=True,
primary_key=True)
    products = models.ManyToManyField('products.Product', related_name='recipes')
    name = models.CharField(max_length=255)
    picture = models.ImageField(upload_to=recipes_image_path)
    description = models.TextField()
    create_date = models.DateTimeField(auto_now_add=True)
    abc_score = models.ForeignKey(
        'healthfulness.Healthfulness',
        on_delete=models.PROTECT,
        blank=False,
        related_name='recipes'
    )

    def __str__(self):
        return self.name

import uuid

from django.db import models
from django.db.models import F

```

```

from django.db.models.signals import post_save
from django.dispatch import receiver

class BaseReviewModel(models.Model):
    RATING_CHOICES = [
        (1, '1'),
        (2, '2'),
        (3, '3'),
        (4, '4'),
        (5, '5'),
    ]

    id = models.UUIDField(default=uuid.uuid4, editable=False, unique=True,
primary_key=True)
    content = models.TextField()
    user = models.ForeignKey('accounts.CustomUser', on_delete=models.CASCADE)
    rating = models.IntegerField(choices=RATING_CHOICES)

    class Meta:
        abstract = True

class ProductReview(BaseReviewModel):
    product = models.ForeignKey('products.Product', on_delete=models.CASCADE,
related_name='reviews')

    def __str__(self):
        return f'Review for {self.product.name} by {self.user.username}'

class RecipeReview(BaseReviewModel):
    recipe = models.ForeignKey('recipes.Recipe', on_delete=models.CASCADE,
related_name='reviews')

    def __str__(self):
        return f'Review for {self.recipe.name} by {self.user.username}'

@receiver(post_save, sender=ProductReview)
def update_product_rating(sender, instance, created, **kwargs):
    if created:
        product = instance.product
        try:
            rating_instance = ProductRating.objects.get(product=product)
        except ProductRating.DoesNotExist:
            rating_instance = ProductRating.objects.create(product=product)
        rating_instance.update_ratings(instance.rating)
        rating_instance.update_average_rating()

@receiver(post_save, sender=RecipeReview)
def update_recipe_rating(sender, instance, created, **kwargs):
    if created:
        recipe = instance.recipe
        try:
            rating_instance = RecipeRating.objects.get(recipe=recipe)
        except RecipeRating.DoesNotExist:
            rating_instance = RecipeRating.objects.create(recipe=recipe)

```

```

        rating_instance.update_ratings(instance.rating)
        rating_instance.update_average_rating()

class BaseRatingModel(models.Model):
    id = models.UUIDField(default=uuid.uuid4, editable=False, unique=True,
primary_key=True)
    rating_1_count = models.IntegerField(default=0)
    rating_2_count = models.IntegerField(default=0)
    rating_3_count = models.IntegerField(default=0)
    rating_4_count = models.IntegerField(default=0)
    rating_5_count = models.IntegerField(default=0)
    average_rating = models.DecimalField(max_digits=3, decimal_places=2, default=0)

    def update_ratings(self, rating_value):
        if rating_value == 1:
            self.rating_1_count = F('rating_1_count') + 1
        elif rating_value == 2:
            self.rating_2_count = F('rating_2_count') + 1
        elif rating_value == 3:
            self.rating_3_count = F('rating_3_count') + 1
        elif rating_value == 4:
            self.rating_4_count = F('rating_4_count') + 1
        elif rating_value == 5:
            self.rating_5_count = F('rating_5_count') + 1
        self.save()

    def update_average_rating(self):
        rating_counts = self.__class__.objects.values(
            'rating_1_count', 'rating_2_count', 'rating_3_count',
            'rating_4_count', 'rating_5_count'
        ).get(pk=self.pk)

        total_ratings = sum(rating_counts.values())

        total_score = (
            rating_counts['rating_1_count'] * 1 +
            rating_counts['rating_2_count'] * 2 +
            rating_counts['rating_3_count'] * 3 +
            rating_counts['rating_4_count'] * 4 +
            rating_counts['rating_5_count'] * 5
        )

        if total_ratings > 0:
            self.average_rating = total_score / total_ratings
        else:
            self.average_rating = 0

        self.save(update_fields=['average_rating'])

class Meta:
    abstract = True

class ProductRating(BaseRatingModel):
    product = models.OneToOneField('products.Product', on_delete=models.CASCADE,
related_name='ratings')

    def __str__(self):

```

```

        return f"Rating for {self.product.name}"

class RecipeRating(BaseRatingModel):
    recipe = models.OneToOneField('recipes.Recipe', on_delete=models.CASCADE,
related_name='ratings')

    def __str__(self):
        return f"Rating for {self.recipe.name}"

import uuid

from django.db import models
from django.db.models import F
from django.db.models.signals import post_save
from django.dispatch import receiver

class BaseReviewModel(models.Model):
    RATING_CHOICES = [
        (1, '1'),
        (2, '2'),
        (3, '3'),
        (4, '4'),
        (5, '5'),
    ]

    id = models.UUIDField(default=uuid.uuid4, editable=False, unique=True,
primary_key=True)
    content = models.TextField()
    user = models.ForeignKey('accounts.CustomUser', on_delete=models.CASCADE)
    rating = models.IntegerField(choices=RATING_CHOICES)

    class Meta:
        abstract = True

class ProductReview(BaseReviewModel):
    product = models.ForeignKey('products.Product', on_delete=models.CASCADE,
related_name='reviews')

    def __str__(self):
        return f'Review for {self.product.name} by {self.user.username}'

class RecipeReview(BaseReviewModel):
    recipe = models.ForeignKey('recipes.Recipe', on_delete=models.CASCADE,
related_name='reviews')

    def __str__(self):
        return f'Review for {self.recipe.name} by {self.user.username}'

@receiver(post_save, sender=ProductReview)
def update_product_rating(sender, instance, created, **kwargs):
    if created:
        product = instance.product
        try:

```

```

        rating_instance = ProductRating.objects.get(product=product)
    except ProductRating.DoesNotExist:
        rating_instance = ProductRating.objects.create(product=product)
    rating_instance.update_ratings(instance.rating)
    rating_instance.update_average_rating()

@receiver(post_save, sender=RecipeReview)
def update_recipe_rating(sender, instance, created, **kwargs):
    if created:
        recipe = instance.recipe
        try:
            rating_instance = RecipeRating.objects.get(recipe=recipe)
        except RecipeRating.DoesNotExist:
            rating_instance = RecipeRating.objects.create(recipe=recipe)
        rating_instance.update_ratings(instance.rating)
        rating_instance.update_average_rating()

class BaseRatingModel(models.Model):
    id = models.UUIDField(default=uuid.uuid4, editable=False, unique=True,
primary_key=True)
    rating_1_count = models.IntegerField(default=0)
    rating_2_count = models.IntegerField(default=0)
    rating_3_count = models.IntegerField(default=0)
    rating_4_count = models.IntegerField(default=0)
    rating_5_count = models.IntegerField(default=0)
    average_rating = models.DecimalField(max_digits=3, decimal_places=2, default=0)

    def update_ratings(self, rating_value):
        if rating_value == 1:
            self.rating_1_count = F('rating_1_count') + 1
        elif rating_value == 2:
            self.rating_2_count = F('rating_2_count') + 1
        elif rating_value == 3:
            self.rating_3_count = F('rating_3_count') + 1
        elif rating_value == 4:
            self.rating_4_count = F('rating_4_count') + 1
        elif rating_value == 5:
            self.rating_5_count = F('rating_5_count') + 1
        self.save()

    def update_average_rating(self):
        rating_counts = self.__class__.objects.values(
            'rating_1_count', 'rating_2_count', 'rating_3_count',
            'rating_4_count', 'rating_5_count'
        ).get(pk=self.pk)

        total_ratings = sum(rating_counts.values())

        total_score = (
            rating_counts['rating_1_count'] * 1 +
            rating_counts['rating_2_count'] * 2 +
            rating_counts['rating_3_count'] * 3 +
            rating_counts['rating_4_count'] * 4 +
            rating_counts['rating_5_count'] * 5
        )

        if total_ratings > 0:

```

```

        self.average_rating = total_score / total_ratings
    else:
        self.average_rating = 0

    self.save(update_fields=['average_rating'])

class Meta:
    abstract = True

class ProductRating(BaseRatingModel):
    product = models.OneToOneField('products.Product', on_delete=models.CASCADE,
related_name='ratings')

    def __str__(self):
        return f"Rating for {self.product.name}"

class RecipeRating(BaseRatingModel):
    recipe = models.OneToOneField('recipes.Recipe', on_delete=models.CASCADE,
related_name='ratings')

    def __str__(self):
        return f"Rating for {self.recipe.name}"

```

Додаток Б

```

import { FlatList, StyleSheet } from 'react-native';
import { useState, useEffect } from 'react';
import axios from "axios";

import { ScreenTemplate } from "../components/templates";
import { ProductCard } from "../components";
import { BASE_API_URLS } from "../consts/apis";

export const ProductsListScreen = () => {
    const [products, setProducts] = useState([]);
    const [searchQuery, setSearchQuery] = useState('');

    const handleOnSearch = () => {
        console.log(searchQuery);
    }

    useEffect(() => {
        const getProducts = async () => {
            try {
                const response = await axios.get(`${BASE_API_URLS.device}/products`)

                setProducts(response.data)
            } catch (error) {
                console.error('Error fetching products:', error);
                throw error;
            }
        }

        getProducts()
    }, []);

    return (
        <ScreenTemplate

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						77

```

navigation
topBar
topBarType='search'
notScrollable
searchBarProps={ {
  searchQuery: searchQuery,
  setSearchQuery: setSearchQuery,
  handleOnSearch: handleOnSearch,
  placeholder: 'Ви шукаєте товар?',
}}
>
<FlatList
  data={products}
  renderItem={({ item }) => (
    <ProductCard
      id={item.id}
      name={item.name}
      imageUrl={item.picture}
      lvl={item.abc_score}
      style={styles.card}
    />
  )}
  keyExtractor={item => item.id}
  numColumns={2}
  contentContainerStyle={styles.list}
  columnWrapperStyle={styles.columnWrapper}
/>
</ScreenTemplate>
)
}

const styles = StyleSheet.create({
  list: {
    gap: 8,
  },
  columnWrapper: {
    gap: 8,
    justifyContent: "space-between",
  },
  card: {
    width: '48%',
  }
})

```