МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему: «Гібридний мобільний додаток для розпізнавання тексту з зображення»

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

Керівник роботи **Шовкопляс О. А.**

Студентка групи ІН – 62 **Погибелєва Л. В.**

2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую _____
Зав. кафедрою Довбиш А.С.
«_____» _____ 20 р.

Завдання
до випускної роботи

Студентки четвертого курсу, ІН – 62 спеціальності “Інформатика” денної форми навчання Погибелєва Л. В.

Тема: “**Гібридний мобільний додаток для розпізнавання тексту з зображення**”

Затверджена наказом по СумДУ
№_____ від _____ 20___ р.

Зміст пояснювальної записки: 1) аналіз проблеми; 2) методи розробки гібридного мобільного додатка; 3) програмна реалізація.

Дата видачі завдання “_____”_____ 20___ р.
Керівник випускної роботи _____ Шовкопляс О. А.
Завдання прийняв до виконання _____ Погибелєва Л. В.

РЕФЕРАТ

Записка: 69 стор., 33 рис., 1 додаток, 18 джерел.

Об'єкт дослідження – процес розробки та реалізації інформаційного та програмного забезпечення мобільного додатка.

Мета роботи – розробка та програмна реалізація додатка, здатного розпізнавати текст на зображенні та конвертувати його в текст в форматі Txt.

Методи дослідження – методи розробки гібридних систем.

Результати – виконано розробку програмного забезпечення гібридного мобільного додатка. Детально розглянуто методи створення гібридного мобільного додатка мовою JS на фреймворці React. Проведено аналіз існуючих аналогів, визначено їх переваги та недоліки. Проведено порівняння альтернатив з розроблюваним додатком. Було розглянуто обрані для створення додатка компоненти розробки. Для валідації користувача було розглянуто і обрано бібліотеку Formik . Реалізовано авторизацію користувача в системі через Firebase. Реалізовано навігацію по додаткові через react—navigation. Обрано Firestore для збереження нотаток користувача. Розглянуто та обрано документо подібну базу данных NoSQL.

Зміст

ВСТУП 5

1 АНАЛІЗ ПРОБЛЕМИ 6

1.1 Огляд аналогів систем для розпізнавання об’єктів 6

1.2 Середовище розробки 7

1.3 Аналіз і порівняння засобів і мов кросплатформного програмування .. 8

1.4 Постановка задачі 15

2 МЕТОДИ РОЗРОБКИ ГІБРИДНОГО МОБІЛЬНОГО ДОДАТКА 18

2.1 Загальні відомості 18

2.2 Особливості мови та загальні відомості 18

2.3 Особливості фреймворку React для розробки мобільних додатків. 19

3 ПРОГРАМНА РЕАЛІЗАЦІЯ 22

3.1 Функціонал додатка 22

3.1.1 Документо Подібна база даних NoSQL 22

3.1.2 Реалізація валідації через бібліотеку Formik 23

3.1.3 Реалізація авторизації через Firebase 24

3.1.4 Реалізація навігації в застосункові через React Navigation 25

3.1.5 Хмарне сховище Firestore — для збереження нотаток 27

3.2 Тестування додатка 31

ВИСНОВКИ 56

СПИСОК ЛІТЕРАТУРИ 57

ДОДАТОК А 59

ВСТУП

Мова — найвидатніше досягнення людства та основний, найбільш зручний, інструмент комунікації. Можливість передавати інформацію завдяки мовленню — одна з головних відмінностей людей від інших створінь, здатних до комунікації. Навіть у наш час не існує більш досконалого способу комунікації ніж мова.

В сучасному світі, де необхідність вести документацію являється невід'ємною частиною життя, ми нерідко зіштовхуємося з проблемою, коли необхідно швидко переписати (сфотографувати) текст документу, оголошення, а в випадку школярів та студентів і тексти завдань, або тем для рефератів, доповідей, тощо. Подібні методи зберігання та передачі інформації є недостатньо оптимальними чи комфортними, в перспективі величезної кількості не зручностей, пов'язаних із роботою з текстами на фото (наприклад: якість зображення, чи недоліки пристрою з якого необхідно дублювати інформацію).

З кожним роком, питання необхідності зручно обмінюватися та зберігати інформацію стає все більш гостро та актуально. В столітті гаджетів та цифрових файлообмінювачів паперова документація залишається розчаровуючим та гнітючим пережитком минулого, від якого закостенілі в своїх стереотипах бюрократи й досі не можуть відмовитися.

В світлі вище описаних, та ряду менш значущих причин, в мене з’явилася думка, щодо необхідності створення додатка, котрий дозволив би в рази полегшити наше життя.

В даній роботі буде описаний метод створення гібридного мобільного додатка для розпізнавання тексту із зображень та конвертування його в формат Txt для редагування або іншої роботи з текстом.

1 АНАЛІЗ ПРОБЛЕМИ

1.1 Огляд аналогів систем для розпізнавання об’єктів

На даний момент програма “sCAn” має декілька аналогів які, тим не менш, мають ряд відмінностей. Найпопулярнішою серед подібних є “Adobe Scan: сканування PDF, OCR” і не дивлячись на відомий бренд навіть у неї є недоліки [2]:

1. Дані зберігаються в хмаровому сховищі, що автоматично робить їх більш доступними, тож для роботи з деякими конфіденційними документами даний додаток просто не підходить, через високу небезпечність злому сторінки та розповсюдження даних.
2. Також, додаток не завжди розпізнає текст, що шкодить роботі.
3. Для коректної роботи з додатком необхідне підключення до інтернету.

Дещо схоже можна знайти в мобільній версії перекладача Google [1]. В онлайн версії перекладача доступне розпізнавання тексту з зображення з наступним перекладом. Це досить корисно, та даний функціонал має свої недоліки. До них можна віднести те, що:

1. Ця опція не працює в офлайн режимі;
2. З відсканованим текстом надалі складно працювати через недоліки самого додатка та непередбачувані збої роботи пристрою чи мережі;
3. Надається кілька варіантів для розпізнавання та перекладу, що може збентежити не досвідчених користувачів;
4. Деякі з запропонованих варіантів для сканування являються не достатньо зручними у використанні через постійне розмиття чи навіть зміну тексту.

В цілому можна знайти не один приклад подібних систем для розпізнавання об’єктів. До них можна віднести зчитування QR кодів, програми розпізнавання облич та нейронні сіті з розпізнаванням об’єктів.

Досить популярним прикладом подібних програм можна вважати розважальний додаток “Snapchat”. В ньому використовується розпізнавання для

фіксації необхідних точок на обличчі людини для подальшого накладання різноманітних масок.

Наразі, досить популярною темою серед програмістів вважається створення нейросітей здатних розпізнавати об’єкти різних типів та класифікувати їх згідно до параметрів. Машинне навчання становиться все більш відомим та необхідним для різних сфер життя людини. Таким чином в майбутньому можна буде створити роботизовані пристрої для медицини, задля проведення надскладних операцій, що потребують надлюдської точності.

З уже існуючих, до систем подібного типу можна віднести безпілотні автомобілі, що здатні розрізняти дорожні знаки, сигнали та пішоходів.

1.2 Середовище розробки

Майже кожен зараз має смартфон, та зі скількома незручностями зіштовхуються користувачі різних систем? Війна між прихильниками користувачів “Android”та “iOS” не вщухає вже більше ніж десятиліття. Тож на даний момент, розробники по максимуму намагаються створювати саме гібридні мобільні додатки, аби користуватися ними могли з будь якого пристрою.

Гібридний мобільний додаток створюється одночасно і для «Android» і «iOS», це означає, що він є кросплатормним. Зовнішній вигляд та функціонал гібридних мобільних додатків, при цьому, працюють однаково, як і рідні, і за для їх створення застосовуються єдині технології. Наприклад, «Instagram», «Facebook» та «Discord» є крос платформними додатками.

Переваги гібридних мобільних додатків [3]:

1. висока швидкість і функціональність проектування;
2. можуть працювати водночас на «iOS» і «Android»;
3. подібність до нативних, за нижчою ціною;
4. реалізація одразу на ринках “App Store” та “Google Play”, як двох додаткових каналах комунікації;

5. початківці і компанії гіганти IT – індустрії надають перевагу саме гібридній розробці;
6. гнучкість і легка адаптивність щодо користувацьких потреб;
7. оптимізація розтрат на розробку і покриття потреб більшої кількості користувачів.

Потрібно зазначити, що генерація у них легша, а вартість одного кросплатформного рішення значно нижче, ніж окреме створення додатків під «iOS» та «Android».

1.3 Аналіз і порівняння засобів і мов кросплатформного програмування

Кросплатформність – це можливість ПЗ працювати більше ніж лише на одній платформі чи ОС.

Міжплатформність ПЗ здобула значення після завершення ери без роздільного панування платформи Wintel. Та від 2012 процент її користувачів впав до 35% і досі продовжує зменшуватися. Це дало змогу крос платформному програмуванню в галузі прикладного ПЗ стати більш економічно привабливим.

Вважається, що одним з перших випадків міжплатформного прикладного ПЗ стала реалізація можливості запускати програми для CP/M в IBM DOS (MSDOS). Це було пов'язано з тим, що IBM DOS 1.0 був не тонким обчисленням IBM, а операційною системою CP/M, переписаною під шістнадцятибітовий процесор Intel 8088.

Мови програмування, які можна використовувати для розробки кросплатформних додатків, можна розділити на три групи:

1. Міжплатформні мови програмування компіляційного виду – для таких мов програмування для різних платформ існують різні компілятори (Pascal, C++, C, тощо);

2. Міжплатформні інтерпритатори – для таких мов існують тлумачі (PHP, Python, Ruby тощо) під різні платформи;
3. Міжплатформні мови виконавчого рівня (Java і C#) – результатом роботи компіляторів в таких мовах є байт—код, котрий є можливість запустити най різноманітніших платформах, в тому числі і за допомогою віртуальних машин (JavaVM для Java).

Далі необхідно розглянути короткий опис характеристик міжплатформних мов програмування, коли вони знаходяться на стадії компіляції.

Таблиця 1.1 Компіляційний рівень міжплатформних мов програмування

Інструментальна оболонка	**Підтримувані компілятори / кількість мов програмування**	**Підтримувані ОС / їхня кількість**
Екліпс	C/C++, Fortran/3	AIX, Опен Соларіс, Соларис, QNX, FreeBSD, Андроід (ARM)/10 HP—UX, Линукс, Mac OS X, Майкрософт Віндоус
Qt Creator	GCC, ICC, GCCE, Clang, MinGW, MSVC, Linux, RVCT, WINSCW/8	Линукс, Юникс, iOS, Андроід, БлєкБери10, WinRT,OS X, Майкрософт Віндоус, QNX/10, Ембеддед Линукс
Фрі Паскаль	Фрі Паскаль Компілер, Об’єктний Паскаль, частково GNU Паскаль, ISO Extended Pascal/4	Haiku/7, MS DOS, Майкрософт Віндоус

Лазарсус	Фрі Паскаль Компілер /1	Линукс, Майкрософт Віндоус, Андроід/5, FreeBSD, Mac OS X
Code::Blocks	GNU GCC Компілер для павер ПК, AVR GCC, Intel C++, GNU Fortran, GNU ARM, GNU GDC/15, Диджитал C/C++, Диджитал Марс D, SDCC, Майкрософт вижуал C++	Віндоус, Юнікс/4, Лінукс, Mac OS X
NetBeans IDE	C, C++	Віндоус, Лінукс, Соларіс/4, FreeBSD
Embarcadero RAD Studio XE7	Delphi, C, C++/3	Віндоус, Андроід/4, Mac OS X
Майкрософт Віжуал Студіо Код	C, C++, Об'єктний-C/3	Лінукс, Mac OS X/2

В таблиці 1.2 приведено короткі характеристики міжплатформних мов на виконавчого рівня (в зв'язку з великою кількістю інструментальних засобів мови Java їх список неповний).

Таблиця 1.2 Виконавчий рівень між платформних мов програмування

Оболонка Інструменту	**Компілятори що підтримуються**	**Кількість підтримуваних операційних систем**
Екліпс	Java	AIX, Лінукс, Мас, OS X, Опен Соларіс, Соларіс, QNX, Майкрософт Віндоус, Андроід (ARM)/10
NetBeans IDE	Java	Віндоус, Лінукс, FreeBSD, Соларіс/4
IntelliJ IDEA	Java	Віндоус, Лінкус, Мас OS X/3
Майкрософт Віжуал Студіо Код	C#, Java/2	Лінукс, Мас OS X/2
.NET Core 5	C#	Лінукс, Мас OS X/2
Моно	C#	Лінукс, Мас OS X/2

В таблиці 1.3 приведено опис короткої характеристики між платформних інтерпретаторів.

Таблиця 1.3 Кросплатформні інтерпретатори

Оболонка інструменту	**Підтримувані перекладачі**	**Підтримувані операційні системи**
Екліпс	Перл, PHP, JS, Пайтон, Рубі/5	AIX, HP-UX, Лінукс, Мас OS X, ОпенСоларіс, Соларіс, QNX, Майкрософт Віндоус, Андроід (ARM)/10
NetBeans IDE	JavaFX, Java, JavaScript,	Віндоус, Лінукс, FreeBSD,

	HTML5, Пійтон, Groovy /7	Соларіс/4
Embarcadero RAD Studio XE7	HTML5/1	Віндоус, Мас OS X, iOS, Андроід/4
Хojo IDE	Реал Базик/1	OS X, Віндоус, Лінукс, iOS/4
Комодо IDE/ Комодо Edit	Перл, PHP, CSS3, HTML5, XML, Пайтон, Рубі, Tcl. JS, XSLT/10	Віндоус, Мас OS X/3, Лінукс
PyCharm	Пайтон, JS, HTML/3	Віндоус, Мас OS X/3, Лінукс
Intellij IDEA	Java/1	Віндоус, Мас OS X/3, Лінукс
Aptana Studio 3	JS, Рубі, Пайтон/4, PHP	Віндоус, Мас OS X/3, Лінукс
Майкрософт Віжуал Студіо Код	Пайтон, PHP,JS, JSON, XML, HTML, CSS /7	Лінукс/2, Мас OS X

Самими популярними більше ніж десять років є мови програмування С (16,703%) та Java (15,528%). Їх широко використовують в кросплатформному програмуванні. Далі за популярністю йдуть С++ , С# , PHP, Python, Perl, Delphi/Object, Pascal. Іншим чинником відбору є число апаратних платформ і число операційних систем, для котрих доцільно використовувати засоби міжплатформної розробки. Для стандартизованого програмування важливі стандартизовані бібліотеки часу виконання. Бібліотеки мови С стали прийнятим стандартом. Власні стандартизовані бібліотеки наявні і в С ++, Java, Пайтон. Вони оснащені інструментами розробки та доступні на платформах що підтримуються. Стандартизовані бібліотеки часу виконання поділені на бібліотеки з відкритим вихідним кодом і закритим вихідним кодом (таблиця 1.4).

Таблиця 1.4 Стандартні бібліотеки та програмні каркаси з відкритим та закритим кодом

Відкритий код	**Закритий код**
Буст, GIMP ТулКІТ, , OpenCL, OpenCV, GTK+, FLTK, Ківі, OpenGL, SDL, Apache Cordova, Tk	
OpenAL (Більш ранні версії)	OpenAL (Більш пізні версії)
Qt	Qt
SDM L	ЮнітіЗD

GTK+ від GIMP ТулКІТ — це спеціальна програма для створення графічного інтерфейсу користувача. Створений для редактора растрової графіки GIMP у 1997 році Спенсером Кімболом та Пітером Матіс, членами XCF в UC Беркли. В даний час розроблений як частина проекту GNU, це безкоштовний програмний пакет. Код GTK + поширюється за ліцензією LGPL, що дозволяє використовувати GTK + не тільки для розповсюдження безкоштовних PZ, але і для створення фірмової програми, яка не використовує своїх виробників і не вимагає використання будь-якої програми. Спеціально спровоковано використання не C / C ++, а іншого програмування в програмі з візуальним інтерфейсом, за допомогою якого Glade може включити полегшеного користувача та скоротити час написання графічних інтерфейсів.

FLTK — це бібліотека біологічних інструментів з відкритим кодом для вбудованого користувацького графічного інтерфейсу (GUI). Працює на UNIX / Линукс X11, Майкрософт Віндоус, Юнікс / Лінукс X11, Mac OS X.

OpenGL — специфікація, яка не залежить від мови програмування міжплатформенного інтерфейсу програмування додатків (API), що

використовує 2D та 3D комп'ютерну графіку. Цей інтерфейс використовує 250 функцій, які можна виконати для малювання складних тривіальних сцен простими нотами. Графічні інтерфейси (Compiz, Clutter) широко використовуються в комп'ютерній індустрії та віртуальній реальності для візуалізації наукових даних, що використовуються в комп'ютерному дизайні.

SDML — це мультимедійна міжплатформна бібліотека, яка вільно поширюється з необробленим кодом. Письмовий рух C, що забезпечує більш простий інтерфейс для графіки, звуку та дизайну, що використовується на різних платформах. Бібліотека SDL надає такі функції, як швидке використання 2D графіки, обробка вхідних даних, аудіо програмування, використання 3D через OpenGL та без великої продуктивності в можливостях крос-платформ незалежно від використання систем. Він намагається створити вживану та ігрову, яка швидко виводить двовимірну графіку, програмуючи звук, використовуючи розширену обробку у користувача. Бібліотека випускається під ліцензією LGPLv2 за допомогою Linux, Windows, Windows CE, BeOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD / OS, Solaris, IRIX та QNX.

OpenAL (Open Audio Library) - це міжплатформенний інтерфейс програмування програм (API) для роботи з аудіоданими. Призначений для ефективної взаємодії з багатоканальним звуком у тривимірному просторі. Раніші версії OpenAL були відкритим кодом, але пізніші версії - це власні програми.

OpenCL (від англ. Open Computing Language) - програмне середовище для створення комп'ютерних програм, пов'язаних з паралельними обчисленнями на різних графічних (GPU) та центральних (CPU) процесорах. Програмне середовище OpenCL включає мову програмування на основі C99 та інтерфейс програмування комп'ютера (API). Платформа OpenCL забезпечує паралелізм на рівні інструкцій та даних та є реалізацією технології GPGPU. Це повністю відкритий стандарт, його використання доступне на основі безкоштовних ліцензій. Мета OpenCL - доповнити OpenGL та OpenAL, які є галузевими

стандартами для 3D-комп’ютерної графіки та звуку, використовуючи можливості GPU. Розроблений та підтримуваний OpenCL некомерційним консорціумом Khronos Group, до якого входять багато великих компаній, серед яких Apple, AMD, ARM, Intel, nVidia, Qualcomm, Sun Microsystems, SCE.

OpenCV — це бібліотека для роботи з комп’ютерним зором, обробкою зображень, комп'ютерним навчанням та числовими алгоритмами загального призначення. Містить понад 2500 оптимізованих алгоритмів. Бібліотека поширюється за ліцензією BSD, вона підтримує C, C ++, Python та Java. Підтримуються Windows, Linux, Mac OS, iOS та Android.

Qt — це програмне забезпечення для створення кросплатформних програм із широким спектром функцій: створення графічного інтерфейсу, робота з мережею, дво- та тривимірна графіка, веб-контент, бази даних тощо. Рамка написана на C + + і дозволяє створювати крос-платформні програми для ряду різних платформ: Windows, Linux / X11, OS X, Android, iOS, Windows 8, QNX / BlackBerry 10, Embedded Linux, VxWorks, Integrity. Він поширюється як під безкоштовним ліцензією (GPLv3, LGPL2 / 3), так і під комерційною ліцензією. Його можна використовувати для розробки в Python (PyQt, PySide).

Також розглянемо деякі відомі інструменти для розробки міжплатформних програм для інтерпретованих мов.

Apache Cordova — це програмне середовище для створення мобільних додатків, яка продовжує розробку платформи PhoneGap після передачі проекту Adobe в Apache Foundation. HTML, JS, CSS використовуються для створення програм. Підтримуються наступні мобільні платформи: Android, bada, BlackBerry 10, iOS, Firefox OS, Tizen, Windows iPhone 7/8, Windows 8.

Ківі — це кроссплатформенная програмне середовище, написана на Python, яка фокусується на створенні новітніх творчих призначених для користувача інтерфейсів і мобільних додатків. Підтримує створення програм

для програмних платформ Linux, Windows, OS X, Android і iOS. Поширюється за ліцензією MIT.

Юніті 3D - це кроссплатформенная платформа і система для створення комп'ютерних ігор. Підтримує ряд програмних, мобільних та ігрових платформ: BlackBerry 10, Windows Phone 8, Windows, OS X, Linux, Android, iOS, Unity Web Player, Adobe Flash, PlayStation, Xbox, Wii. Поширюється в безкоштовній версії (деякі можливості обмежені) і з платної ліцензією. Ви можете використовувати C # (найчастіше), а також мови UnityScript і Boo для програмування.

Tk - це безкоштовний відкритий багатоплатформовий набір віджетів і програмних інструментів для створення графічних інтерфейсів для настільних комп'ютерів. Створено на початку як розширення мови Tcl. Може використовуватися для створення графічних інтерфейсів в Ada, Haskell, Перл, Пайто, Рубі, Rexx і Common Lisp; wxWidgets - набір віджетів і програмного забезпечення для створення графічних інтерфейсів. Підтримує Віндоус, OS X, iOS, Лінукс / Unix і деякі інші платформи. Може використовуватися з Пайтон, Перл, Рубі, C ++.

Таким чином, за популярністю мов програмування серед компіляторів перше місце займає мова програмування C, серед перекладачів - мова Java.

Залежно від кількості операційних систем, в яких можна використовувати засоби розробки, згадані в огляді, вони розташовані в таблиці 1-3. Останній рядок у таблиці 1 з двома підтримуваними ОС - код Microsoft Visual Studio, наведений в таблиці 2 Останні три рядки з двома підтримуваними операційними системами зайняті кодом Microsoft Visual Studio.NET Core 5, Mono, в таблиці 3 Останній рядок з двома підтримуваними ОС зайнята кодом Microsoft Visual Studio.

Як і очікувалося, кількість стандартних бібліотек з відкритим вихідним кодом майже в чотири рази більше ніж кількість стандартних бібліотек з закритим вихідним кодом.

1.4 Постановка задачі

Метою роботи є розробка та програмна реалізація гібридного мобільного додатка для розпізнавання тексту з зображення. Для досягнення поставленої мети необхідно виконати такі завдання.

1. Розробка функціоналу додатка:
   а) створення бази даних;
   б) підключення функціоналу реєстрації;
      - реалізація валідації;
      - реалізація авторизації користувача в системі;
   в) налаштування логіки навігації в додатку.
2. Тестування додатка.

2 МЕТОДИ РОЗРОБКИ ГІБРИДНОГО МОБІЛЬНОГО ДОДАТКА

2.1 Загальні відомості

Додаток “sCAn” створений здебільшого для повсякденного користування. Він може бути корисним в будь якій сфері та для людей будь-якого віку. Відсканований текст додаток зберігає в форматі нотаток з можливістю подальшого редагування. Об’єм нотаток не обмежений.

2.2 Особливості мови та загальні відомості

JavaScript - це проста у використанні об'єктно-орієнтована мова, об'єднана JIT, з функціями першого класу. Вона найбільш широко використовується як мова сценаріїв для веб-сторінок, а також використовується в інших програмних продуктах, таких як node.js або Apache CouchDB. JavaScript - орієнтована на прототип, багатопарадигматична мова, що має динамічну типізацію, яка підтримує об'єктно-орієнтовані, імперативні та декларативні стилі програмування.

Стандартом JS являється ECMAScript. Раніші версії браузерів підтримують принаймні версію 3.ECMAScript а від 2012 року, всі браузери підтримують версію 5.1 ECMAScript. В червні 2015 року вийшла шоста версія ECMAScript. Ця версія носить назву ECMAScript 2015, яку часто називають ECMAScript 2015. Стандарти ECMAScript останнім часом видаються щорічно.

Не слід плутати JavaScript с мовою програмування Java. І "Java", і "JavaScript" є торговими марками або зареєстрованими торговими марками Oracle в США та інших країнах. Однак, в обох мов різний синтаксис, семантика і застосування, хоча вони мають подібні особливості у стандартних бібліотеках та конвенціях про іменування. Синтаксис обох мов "успадковується" від C, але семантика та дизайн JS є результатом впливу Селф і Схеми.

Найчастіше використовується для створення скриптів веб-сторінок, які дозволяють клієнтові взаємодіяти з користувачем, контролювати браузер,

асинхронно спілкуватися з сервером та змінювати зовнішній вигляд та структуру веб-сторінки.

JavaScript класифікується як прототипна скриптова мова програмування з динамічною типізацією. Окрім прототипування, JavaScript також частково підтримує інші парадигми програмування, такі як імперативні та частково функціональні, та деякі архітектурні особливості, зокрема: динамічне та слабке введення тексту, автоматичне управління пам’яттю, успадкування прототипу, функціонує як об’єкти першокласного класу.

JavaScript використовується для:

- створення односторінкових веб-додатків;
- написання сценаріїв веб-сторінок, щоб зробити їх інтерактивними;
- серверне програмування;
- стаціонарні програми;
- мобільні додатки;
- сценарії в прикладному програмному забезпеченні;
- в PDF-документах.

2.3 Особливості фреймворку React для розробки мобільних додатків.

React — бібліотека JavaScript для створення інтерфейсів користувача, яка призначена для вирішення проблем неповного оновлення вмісту на веб-сторінці, які виникають при розробці односторінкових додатків. Його розробляють Facebook, Instagram та спільнота окремих розробників.

React використовують щоб розробляти великі веб-додатки, які використовують дані, які змінюються з плином часу, не перезавантажуючи сторінку. Його мета - бути простим, швидким, масштабованим. React обробляє лише інтерфейс користувача у додатках. Це відповідає шаблону контролера модельного перегляду (MVC) [4] і може використовуватися в поєднанні з іншими бібліотеками JS, або у великих структурах MVC, таких як AngularJS [5]. Його також можна використовувати з доповненнями на основі React для

догляду за частинами без інтерфейсу для створення веб-додатків. Як бібліотека інтерфейсів, користувачі найчастіше використовують React спільно з іншими бібліотеками, на кшталт Redux [6].

В даний час React використовують такі компанії як: Netflix [14], Sony [15], Yahoo [18], атласький [16], Airbnb [17], Академія Хана [13].

Усі властивості передаються компонентам візуалізації як властивість тегу html. Компонент не може змінювати власні передані властивості, але вони можуть використовувати його через функцію зворотного виклику.

Відповідає за те, що реалізує віртуальний DOM [7], не звертаючись до браузера DOM. Можно побачити бібліотеку, яку DOM завжди змінював, порівняно зі складеним віртуальним DOM, і можна бачити, як оновлення браузера DOM є найбільш ефективними. Таким чином програміст, який працює зі сторінкою, вважає що вона оновлюється вся, але бібліотека самостійно вирішує які частини сторінки необхідно оновити.

Компонуючі React фактично написані в JSX [8]. Код, написаний у JSX, компілюється у вибраних методах бібліотеки React. Розробники також можуть писати в чистому JavaScript. JSX запропонував ще один текст, який «Facebook». створив для розширення PHP - XHP.

React, використовується не лише для перегляду HTML у браузері. Наприклад, у «Facebook» є динамічна графіка, яка відображається в тегах «<canvas>», а «Netflix» і «PayPal» використовують лише для завантаження для оновлення HTML-облікових даних на серверах та сторінках клієнтів.

Методи життєвого циклу - це різноманітні методи, що використовуються за допомогою ReactJS. Ці можливості дозволяють розробникам працювати в різних точках в різних циклах програми React. Приклад:

- shouldComponentUpdate - це метод життєвого циклу, який доручає Javascript оновлювати компонент за допомогою логічних змінни.

- компонентWillMount - це метод життєвого циклу, який доручає Javascript конфігурувати певні дані перед монтажем компонентів.
- компонентDidMount - метод життєвого циклу, подібний до компонента WillMount, за винятком того, що він працює після методу візуалізації і може використовуватися для додавання даних JSON [9], а також для визначення властивостей та станів.
- Візуалізація - найважливіший спосіб життєвого циклу, необхідний для будь-якого компонента. Метод візуалізації - це те, що підключається до JSX і відображає власний JSX.

Кілька елементів на одному рівні повинні бути загорнуті в один контейнерний елемент, наприклад , або повернутись як масив.

JSX надає ряд атрибутів, призначених для відображення тих елементів, які є у форматі HTML. Спеціальні атрибути також можуть передаватися компоненту. Усі атрибути будуть отримані компонентом у вигляді реквізиту.

Принципи React Native здебільшого такі самі, як і ReactJS, за винятком того, що він не управляє DOM через VirtualDom.

Він працює у фоновому режимі, який інтерпретує код розробника Javascript безпосередньо на цільовому пристрої та спілкується зі своєю власною платформою.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Функціонал додатка

В наступній частині більш детально буде розглянути які саме методи та компоненти використовувалися при розробці додатка. Так як розробка повноцінного додатка включає в себе більше ніж одну підсистему, кожен компонент буде розглянуто окремо, включаючи і пояснення того, які саме елементи функціоналу було розроблено і з допомогою чого.

3.1.1 Документо Подібна база даних NoSQL

Бази даних NoSQL спеціально розроблені для певних моделей даних і мають гнучкі схеми, які дозволяють розробляти сучасні програми. Бази даних NoSQL набули широкого поширення в зв'язку з простотою розробки, функціональністю і продуктивністю при будь—яких масштабах..

NoSQL використовує різноманітні моделі даних для доступу до даних і керування ними. Бази даних цих типів оптимізовані для додатків, що працюють з великими обсягами даних. Все це є результатом пом'якшення суворих вимог до узгодженості даних, характерних для інших типів баз даних.

Розглянемо приклад схем моделювання для простої бази даних нотаток додатки, розроблений в даній дипломній роботі.

Введення даних в базу даних часто ділиться на кілька частин (або «нормалізується») і зберігається в окремих таблицях. У цьому прикладі таблиця Notes містить стовпці «Ім'я нотаки» та «ISBN». Реляційна модель призначена для забезпечення цілісності довідкових даних. Дані нормалізуються для зменшення надмірності і в цілому оптимізуються для зберігання.

У базі даних NoSQL запис дописів звичайно зберігається як документ JSON. Для кожної замітки значення «Ім'я замітки» і «ISBN» зберігаються як атрибути в одному документі. Ці моделі оптимізовані для інтуїтивного дизайну і горизонтальної масштабованості.

База даних NoSQL підходить для багатьох сучасних додатків, таких як мобільні, як в даному випадку, для ігор та інтернет-додатків, де необхідні гнучкі можливості для масштабування даних і забезпечення високої продуктивності і великої функціональності.

Як правило, бази даних NoSQL надають гнучкі схеми. Завдяки використанню гнучких моделей даних, база даних NoSQL добре підходить для частково структурованих і неструктурованих даних.

База даних NoSQL розрахована на масштабування з використанням розподілених апаратних кластерів, а не шляхом додавання дорогих, надійних серверів. Більшість постачальників хмарних послуг виконують ці операції в фоновому режимі.

База даних NoSQL оптимізована для конкретних моделей даних і шаблонів доступу, що дозволяє домогтися більш високої продуктивності в порівнянні з даними бази даних.

У даній роботі використовувалась JSON, так як це ефективна і інтуїтивна модель даних. Документи бази даних дозволяють зберігати і запитувати дані в базі даних. Гнучкий, напівструктурований, ієрархічний характер документів дозволяє їм розвиватися відповідно до потреб додатка. Документація добре працює в каталогах, є всі необхідні версії. Amazon DocumentDB (сумісна з MongoDB) і MongoDB — поширені документальні бази даних, які надають функціональні і інтуїтивно зрозумілі API для гнучкої розробки.

3.1.2 Реалізація валідації через бібліотеку Formik

Formik — це зірка, що піднімається над долиною форм. Дана бібліотека базується на техніці надання реквізиту, і перше, що кидається в очі — гасло: "Побудуйте форми в Реакті без сліз".

Початок роботи з Formik насправді дуже простий. Доступні два варіанти використання: через HOC або компонент з рендером. Найпростіше використати

рендери, оскільки всі конфігурації (перевірка з'єднання, обробка відправки) досить зручно організовані в один компонентний файл.

Основні компоненти:

- Formik — Компонент з усією логікою та render prop
- withFormik — High—Order Component
- Field — просунутий компонент <input> (аналогічно до redux—form)

Formik дає повну свободу вибору бібліотеки валідації. Валідація за замовчуванням проводиться на кожну зміну поля і на зміну фокуса, але це вимикається налаштуванням параметрів. В документації використовується бібліотека yup для створення валідаційних схем.

На відміну від Redux—form, можна налаштувати валідаційну схему без інтеграції в сам процес надсилання форми, замість ручної роботи це зробить Formik.

Для ситуації, коли валідація на фронті вдала, але сервер вирішив, що з даними щось не так, можна скористатися вбудованим функціоналом setStatus або setErrors, щоб передати об’єкт помилок з середини функції надсилання форми.

Потрібно написати функцію, яка приймає values та formikBag як аргументи. Якщо з values все зрозуміло — це дані полів форми, то з formikBag складніше. formikBag — набір корисних функцій для маніпулювання станом форми, найголовніші зних: надсилання (setSubmitting), зміни статусу (setStatus) при надходженні помилок.

3.1.3 Реалізація авторизації через Firebase

В застосункові “sCAn” реалізована аутентифікація Firebase, що дозволяє виконувати вхід в програму за допомогою системи, з якої користувачі добре знайомі. Потім додаток може зберігати дані і персональні налаштування

користувача в захищеному хмарному сховищі і забезпечувати доступ до них на всіх його пристроях.

Ідентифікація користувачів дозволяє розділяти доступ, надійно зберігати особисті дані користувачів в хмарі і забезпечити персоналізований досвід на всіх пристроях користувача.

Firebase забезпечує служби серверної частини, прості пакети розробника і готові бібліотеки інтерфейсу для аутентифікації користувачів різних додатків на будь—яких платформах. Аутентифікацію можна виконувати за допомогою паролів і інтегрованих систем ідентифікації — Google, Facebook, Twitter та ін. Це значно спрощує процедуру входу в додаток і забезпечує надійний захист.

Основні можливості :

1. Тісна інтеграція з іншими функціями Firebase.
2. Використання галузевих стандартів, таких як OAuth 2.0 і OpenID Connect, що спрощує інтеграцію з серверним кодом.

Два варіанти для розробки: FirebaseUI – повністю інтегруване універсальне рішення для виконання аутентифікації — або пакет Firebase Authentication SDK, який дозволяє вручну інтегрувати один або кілька методів входу в додаток.

Безпека аутентифікації і зручність забезпечуються за рахунок можливості виконувати вхід через акаунт Google та інші інтегровані системи ідентифікації — Facebook, Twitter і т. Д. Також для входу можна використовувати пароль.

Зручне робоче середовище з миттєвим доступом до додатка на будь—яких пристроях. Це дозволяє утримувати увагу користувачів протягом дня.

3.1.4 Реалізація навігації в застосункові через React Navigation

Кожна програма так чи інакше складається з екранів, різних переходів. Різні бібліотеки реалізують функціонал навігації: NavigatorIOS, wix / react—native—navigation, react—navigation. У всіх одна мета, але кардинально різні

підходи. Для створення додатка розглянутого в даній дипломній роботі обрано саме react—navigation і далі розглянемо чому саме він.

Навігація складається з безлічі частин які визначають систему навігації в цілому. Кожен шматок містить конфігурацію (видимість таба, наявність хедер), параметри для екрану (назва екрану) .Основні елементи навігатора — це:

— StackNavigator (createStackNavigator);

— TabNavigator (createTabNavigator);

— SwitchNavigator (createSwitchNavigator).

Додаток має єдину точку входу, так званий RootNavigator.

CreateStackNavigator — Навігатор який робить стек з роутів. При навігації новий екран штовхається в початок стека, а при виході він же знищується (принцип LIFO, звичайний стек).

У нових версіях навігатора (> 2) кореневий навігатор створюється використовуючи createAppContaner на пряму. Роут це налаштування екранів. Може включати в себе відразу компонент або об'єкт у якого властивість screen є обов'язковим. Так само може містити будь—який інший тип навігатора.

CreateTabNavigator — Якщо потрібен таб навігатор — використовується ця функція. У параметр приймає все так же конфиг роутерів і конфіг для табів. Всередині навігатор використовує react—native—tab—view.

З допомогу конфіга можна змінювати кольорову схему, положення таба, сам таб компонент (для IOS TabBarBottom і для android TabBarTop). Label для таба так само просто можна замінити.

SwitchNavigator — Працює так само як stack, але з тією відмінністю що при переходах екрани не штовхаються в стек і видно один екран в один момент часу (не можливо перейти назад).

У навігатора дуже багато параметрів для кастомізації. Для самих екранів, для анімації переходів.

3.1.5 Хмарне сховище Firestore — для збереження нотаток

Компанія Google нещодавно запустила Cloud Firestore. Cloud Firestore — це хмарна база даних NoSQL, яку Google позиціонує як заміну бази даних в реальному часі.

Cloud Firestore дозволяє зберігати дані на віддаленому сервері, легко отримувати доступ до них та відстежувати зміни в режимі реального часу. Далі ми детальніше розглянемо Cloud Firestore, його переваги перед базою даних у реальному часі та чому саме Cloud Firestore було обрано для розробки програми на прикладі.

Створення та підключення до проекту:

В консолі Firebase обираємо Database і натискаємо на Create database. Далі необхідно обрати налаштування доступу. Для бета версії додатка буде достатньо тестового режиму, але на продакшині краще підійти до цього питання серйозніше [10].

Для налаштування проекту відтворюємо наступні кроки:

Додати Firebase до проекту по інструкції [11]

Додати залежність в app / build.gradle

implementation 'com.google.firebase: firebase—firestore: 18.1.0'

Тепер все готово.

Ознайомимося з базовими прийомами роботи з Cloud Firestore на прикладі додатка розробленому в даній дипломній роботі. Для його роботи необхідно створити проект в консолі Firebase і додати файл google—services.json в проект в Android Studio.

Наступним кроком буде ознайомлення зі структурою зберігання даних.

Firestore використовує колекції та документи для зберігання даних. Документ — це запис, який містить будь—які поля. Документи об'єднуються в колекцію. Документ також може містити вкладені колекції, але це не

підтримується для Android. За аналогією з базою даних SQL колекція — це таблиця, а документ — запис у цій таблиці. Одна колекція може містити документи з різними наборами полів.

Далі розглянемо отримання і запис даних.

Для того щоб отримати всі документи будь—якої колекції досить наступного коду:

```
remoteDB.collection ( "Notes")
        .get ()
        .addOnSuccessListener {querySnapshot ->
            // Успішно отримали дані. Список в querySnapshot.documents
        }
        .addOnFailureListener {exception ->
            // Помилка при отриманні даних
        }
}
```

В даній частині запитуються всі документи з колекції Notes.

Бібліотека дозволяє формувати запити з параметрами. Наступний код показує як отримати документи з колекції за умовою:

```
remoteDB.collection ( "Notes")
        .whereEqualTo ( "title", "Note1")
        .get ()
        .addOnSuccessListener {querySnapshot ->
            // Успішно отримали дані. Список в querySnapshot.documents
        }
        .addOnFailureListener {exception ->
            // Помилка при отриманні даних
        }
}
```

У цій частині ми запитуємо всі документи з колекції Notes, у яких поле title відповідає значенню Note1.

При отриманні документів, їх можна відразу конвертувати в data-класи:

```
remoteDB.collection ( "Notes")
        .get ()
        .addOnSuccessListener {querySnapshot ->
            // Успішно отримали дані. Список в querySnapshot.documents
           val taskList: List <RemoteNote> = querySnapshot.toObjects
(RemoteNote :: class.java)
        }
        .addOnFailureListener {exception ->
            // Помилка при отриманні даних
        }
}
```

Для запису необхідно сформувати Hashmap з даними (де в якості ключа виступає назва поля, а в якості значення — значення цього поля) і передати бібліотеці. Наступний код це демонструє:

```
val taskData = HashMap <String, Any> ()
taskData [ "title"] = task.title
taskData [ "created"] = Timestamp (task.created.time / 1000, 0)
remoteDB.collection ( "Notes")
    .add (taskData)
    .addOnSuccessListener {
        // Успішний запис
    }
    .addOnFailureListener {
        // Помилка під час запису
    }
```

Щоб задати власний id необхідно зробити наступне:

```
val taskData = HashMap <String, Any> ()
taskData [ "title"] = task.title
taskData [ "created"] = Timestamp (task.created.time / 1000, 0)
remoteDB.collection ( "Notes")
    .document ( "New note")
```

```
    .set (taskData)
    .addOnSuccessListener {
        // Успішна запис
    }
    .addOnFailureListener {
        // Помилка під час запису
    }
```

В цьому випадку якщо немає документа з id рівному New note, то він буде створений, а якщо є, то зазначені поля будуть оновлені.

Firestore дозволяє підписатися на зміни даних. Підписатися можна як на зміни колекції, так і на зміни конкретного документа:

```
remoteDB.collection ( "Notes")
        .addSnapshotListener {querySnapshot, error ->
            // querySnapshot - список змін
            // error - помилка
        }
```

querySnapshot.documents— містить оновлений список всіх документів

querySnapshot.documentChanges — містить список змін. Кожен об'єкт містить змінений документ і тип зміни. Можливі 3 типи змін:

ADDED — документ доданий,

MODIFIED — документ змінений,

REMOVED — документ видалений.

Realtime Database надає більш менш зручний механізм завантаження великої кількості даних, який полягає в ручному редагуванні json—файлу і його завантаження. Firestore з коробки нічого такого не надає. Було дуже незручно додавати нові документи, поки я не знайшла спосіб як можна легко завантажити великий обсяг інформації.

Встановити Node.js і npm

Встановити пакет firebase—admin виконавши команду

npm install firebase—admin —save

Сформувати json—файл з даними колекції.

Для завантаження нам знадобитися ключ доступу. [12]

require ('./ firestore_key.json') — файл із ключем доступу.

<NOTE_DATABASE> — назва firestore—бази

"./Json/Notes.json" — шлях до файлу в якому лежать дані

['Created'] — список імен полів з типом Timestamp

Виконати скрипт : node export.js

В скрипті використовуються напрацювання dalenguyen.

3.2 Тестування додатка

Для перевірки працездатності додатка створимо нового користувача, та додамо нотатку. Надалі буде наглядно продемонстровано кожен крок в користуванні додатком.

Рисунок 3.1 — Вікно авторизації

На рисунок 3.1 зображено початковий екран під час запуску додатка. Першим кроком є реєстрація нового користувача, або авторизація в системі. В прикладі який ми розглядаємо, для нас є актуальною саме реєстрація нового користувача. В цьому випадку ми обираємо кнопку “SIGN UP”.

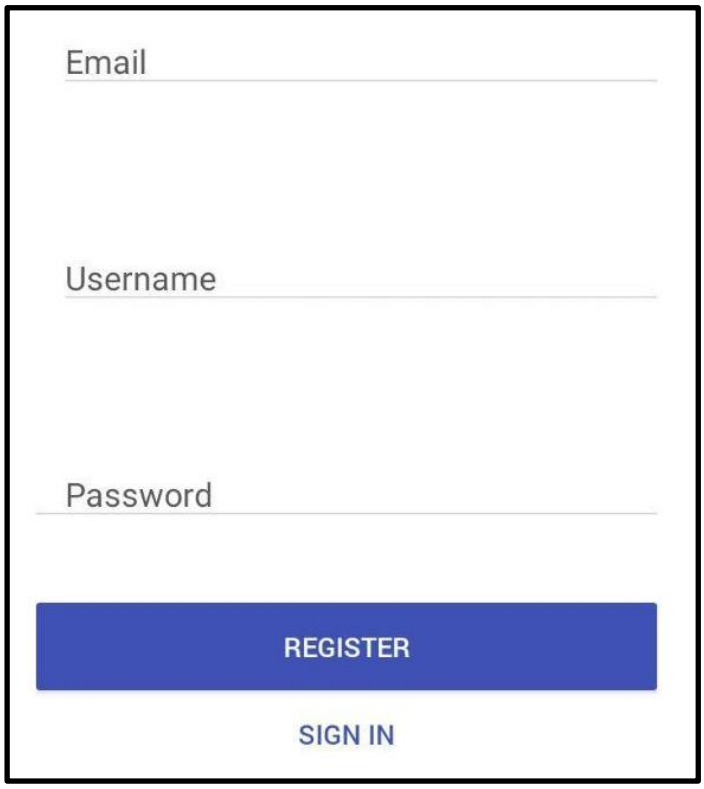

Рисунок 3.2 — Вікно реєстрації користувача з пустими полями

Після того як ми обрали “SIGN UP” відкриється вікно реєстрації користувача, яке зображено на рисунку 3.2. В ньому є три поля для заповнення :

Електронна пошта — “Email”;

Ім’я користувача — “Username”;

Пароль — “Password”.

Електронна пошта необхідна для верифікації користувача, пройти реєстрацію без неї буде неможливо.

Якщо користувач вже має акаунт то він може повернутися до вікна авторизації натиснувши “SIGN IN”

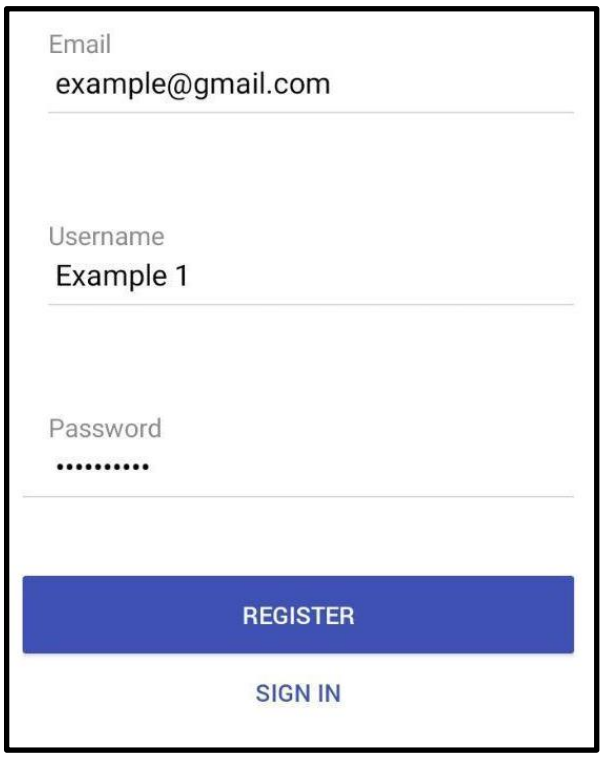

Рисунок 3.3 — Вікно реєстрації користувача з заповненими полями

Для реєстрації заповнюємо поля необхідними даними.

У полі “Email” необхідно ввести електронну пошту користувача. В даному прикладі використаємо “example@gmail.com”.

У полі “Username” необхідно ввести унікальне ім’я користувача. Ім’я повинно містити від чотирьох до десяти символів, без обмежень у використанні спец символів (наприклад: @#$& і подібне, включаючи пропуск). В даному прикладі використаємо ім’я “Example 1”.

У полі “Password” вводимо унікальний пароль, що складається не менше ніж з десяти символів, без обмежень у використанні спец символів (наприклад: @#$& і подібне, включаючи пропуск). Поле паролю ховає введені символи відразу після набору, відображаючи кількість введених символів крапками.

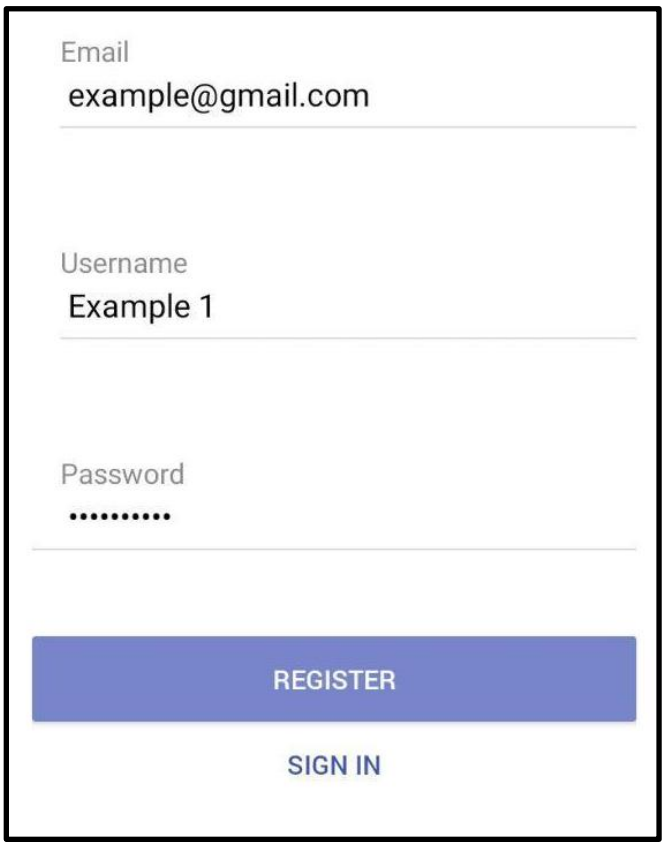

Рисунок 3.4 — Завершення реєстрації

Після заповнення всіх необхідних полів натискаємо кнопку “REGISTER”. На рисунку 3.4 на відміну від рисунка 3.3 ця кнопка блідо синього кольору, що означає, що кнопку активували, а отже процес реєстрації запущено. У разі успішної реєстрації програма повернеться до початкового вікна, зображеного на рисунку 3.1.

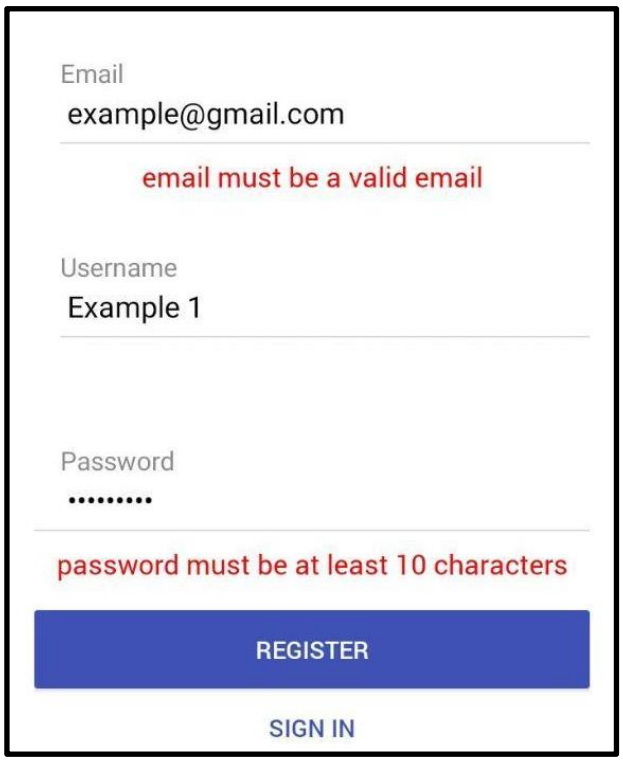

Рисунок 3.5 — Повідомлення про некоректно заповнені поля

Якщо під час заповнення полів “Email” та / або “Password” було допущено помилку, додаток повідомить про це наступними повідомленнями:

1. “Email must be a valid email” — дане повідомлення з'являється якщо було допущено помилку в полі електронної пошти. Вона може вказувати що некоректно введено пошту. Приклади таких помилок зображені на рис 3.6 та рис 3.7
2. В полі “Password” може з'явитися два різних повідомлення: “password is a required field” — яке вказує що поле пароля не заповнено; “password must be at least 10 characters” — яке вказує на те, що введено менше символів ніж допустимий мінімум (десять символів). Приклади цих повідомлень можна розглянути на рис 3.8 та рис 3.9.

Рисунок 3.6 — Приклад помилки в полі введення електронної пошти: “Введено символ пропуску пропуску після пошти”

Рисунок 3.7 — Приклад помилки в полі введення електронної пошти: “Незакінчено введення пошти”

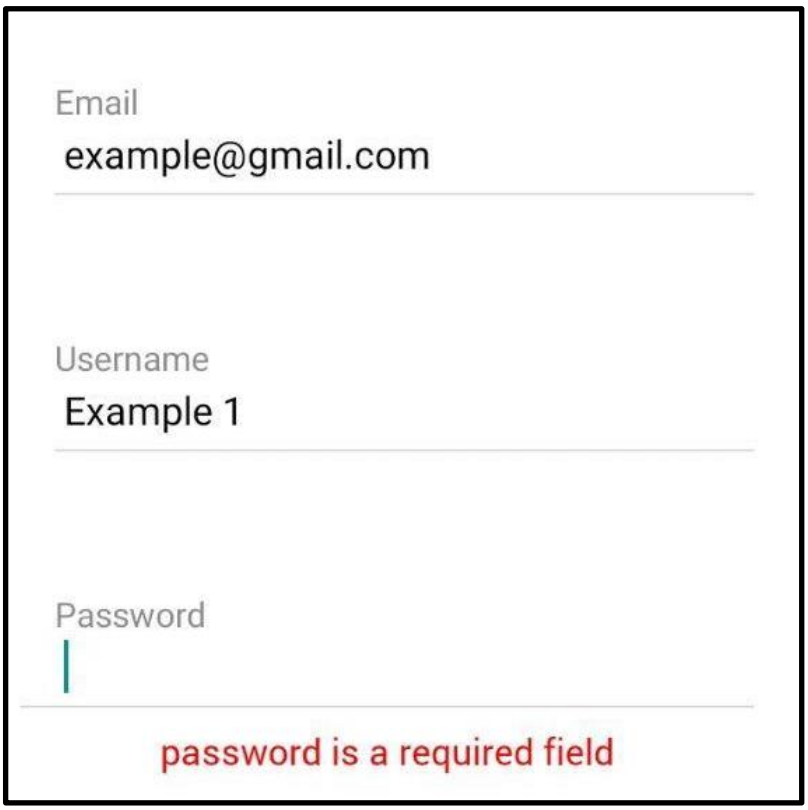

Рисунок 3.8 — Приклад повідомлення про помилку в полі введення паролю: “Поле паролю пусте”

Рисунок 3.9 — Приклад повідомлення про помилку в полі введення паролю: “Пароль повинен складатися що найменш з 10 символів”

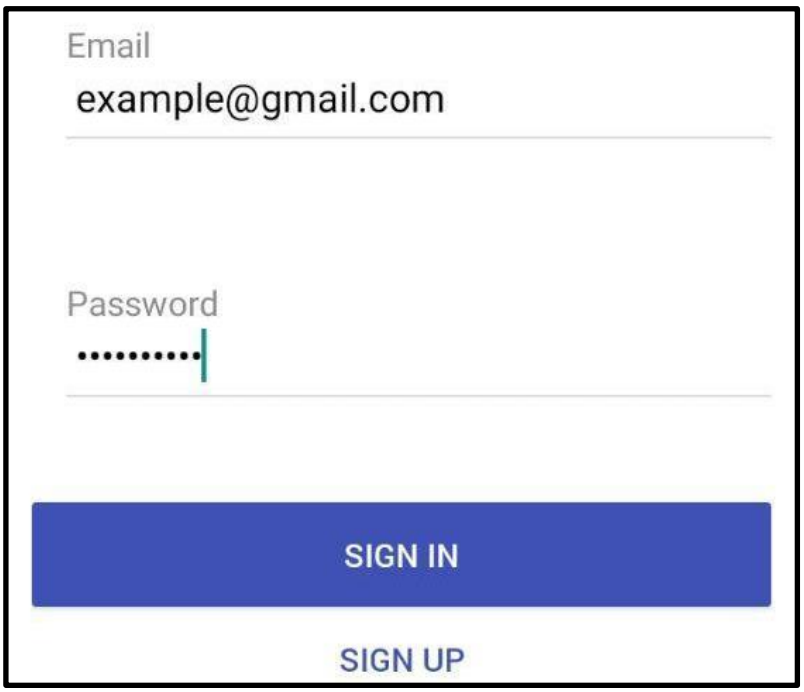

Рисунок 3.10 — Авторизація в додаткові

Після успішного завершення реєстрації повертаємося до вікна авторизації. В поля “Email” та “Password” вписуємо дані, які вказали під час реєстрації. Після введення даних натискаємо кнопку “SIGN IN”.

Рисунок 3.11 — Екран додатка після авторизації. Робоче вікно

В разі успішної авторизації користувач побачить вікно як на рисунку 3.11. На даному зображенні відображений початковий екран при роботі з додатком. Далі ми маємо змогу створити першу нотатку.

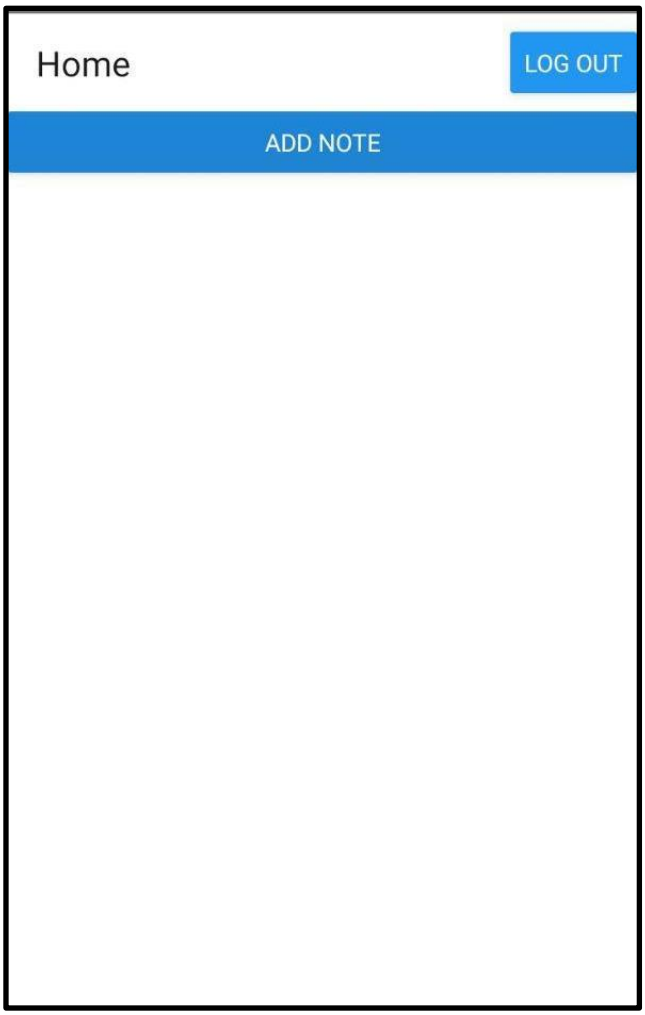

Рисунок 3.12 — Початок роботи з додатком. Створення нотатки

Для створення нотатки необхідно натиснути кнопку “ADD NOTE” , як це зображено на рисунку 3.12. На ньому зафіксовано активацію даної клавіші, що можно помітити по зміні її забарвлення, в порівнянні з цією ж кнопкою на рисунку 3.11. Далі перейдемо до вікна створення нотатки. В цьому розділі будуть відображатися всі збережені користувачем нотатки.

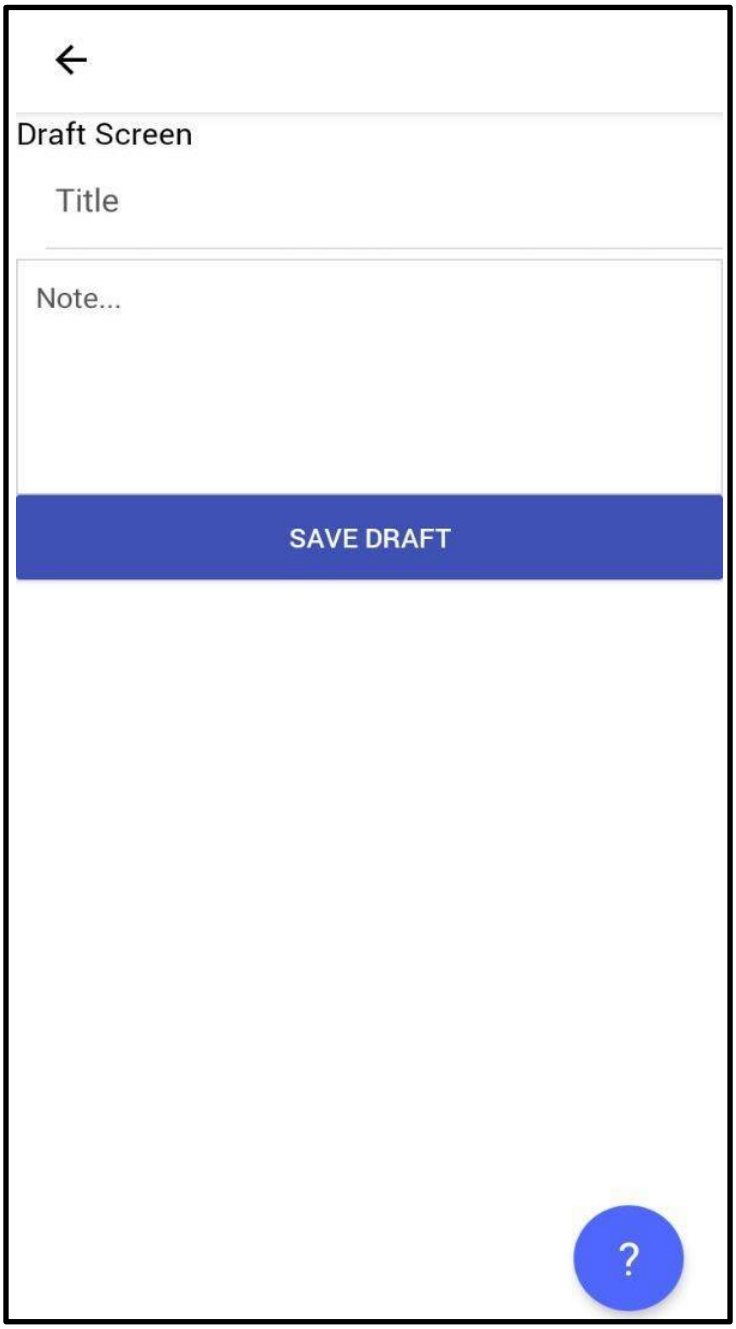

Рисунок 3.13 — Сторінка створення нотатки

На рис. 3.13 зображено сторінку створення нотатки. На ній маємо змогу розглянути наступні елементи:

- Поле “Title”;
- Поле “Note...”;
- Кнопку “SAVE DRAFT”;
- Кнопку функцій (зображена зі знаком питання).

В полі “Title” користувач вказує ім’я нотатки котру створює. Якщо це поле не буде заповнено то нотатка не збережеться і дані будуть втрачено.

В полі “Note...” буде знаходитись сам текст нотатки.

Кнопка “SAVE DRAFT” необхідна для збереження нотатки, змін редагування проведених над нотаткою. Якщо її не натиснути то нотатку, або зміни, не буде збережено, що призведе до втрати даних.

Кнопка функцій являє собою згорнуте меню функцій. Для перегляду доступних функцій необхідно натиснути на кнопку, як це зображено на рис. 3.14. Під час активації кнопка змінює колір на блідо — блакитний.

Рисунок 3.14 — Вигляд активованої кнопки функцій

Після активація кнопки буде розгорнуто меню функцій зі всіма доступними опціями додатка. В версії додатка котру розглядаємо в даному прикладі наявна лише одна функція: сканувати зображення.

Рисунок 3.15 — Вигляд розгорнутого меню

Кнопка функції сканування зображення представлена у вигляді помаранчевої кнопки зі схематичним силуетом фотокамери що відносить нас до інструменту опціоналу смартфону який буде використовуватися даною функцією — камери.

Рисунок 3.16 — Активація функції сканування

Щоб перейти безпосередньо до сканування необхідно натиснути на кнопку сканування зображення. На рис. 3.16 зображено процес активації даної функції, про що свідчить зміна кольору кнопки на блідо—помаранчевий.

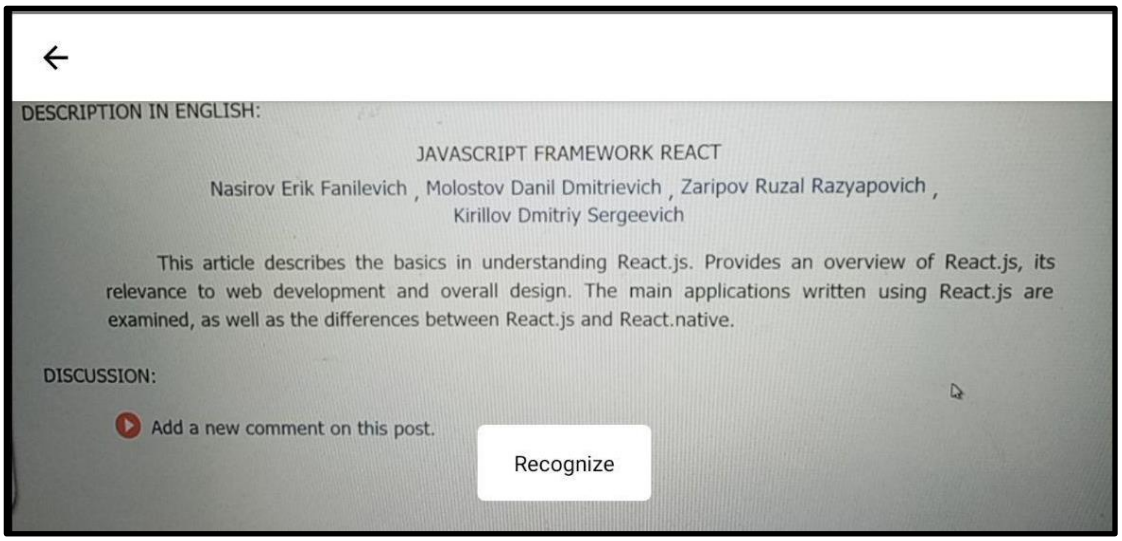

Рисунок 3.17 — Екран під час сканування тексту

На рис. 3.17 представлений скріншот екрану пристрою безпосередньо під час наведення камері на текст. Текст зображений на екрані і буде розпізнано та збережено далі.

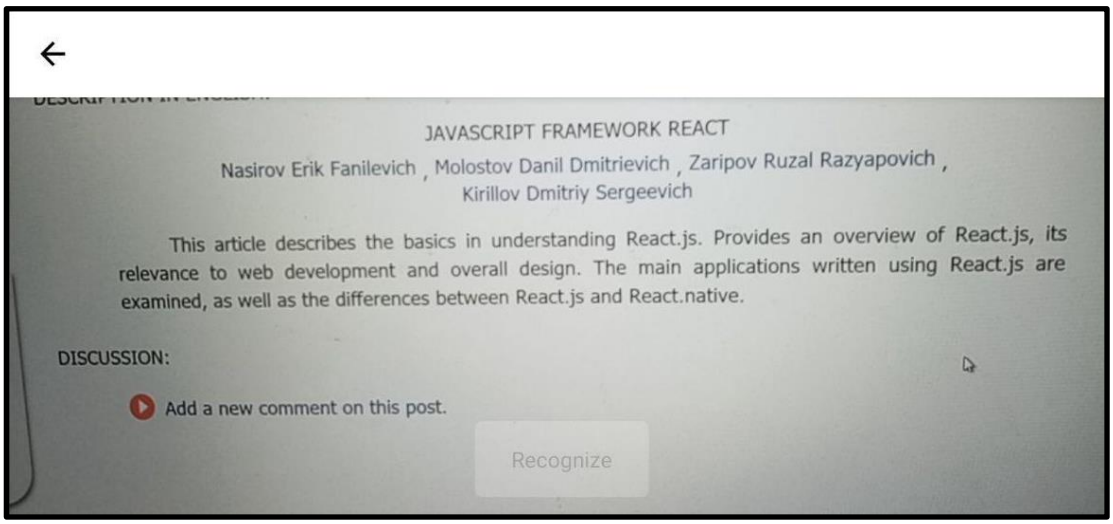

Рисунок 3.18 — Активація функції розпізнавання

Після того як камеру було наведено на необхідний нам текст або ділянку тексту необхідно натиснути на кнопку “Recognize” для запуску сканування тексту з зображення. Активацію функції можно розпізнати по зміні забарвлення кнопки функції — кнопка “Recognize” стає напівпрозорою. Після завершення сканування кнопка “Recognize” повертається до первинного вигляду, що дає нам зрозуміти, що сканування було завершено. Аби повернутися до самої нотатки необхідно натиснути стрілку “назад” в верхньому правому куті (рис. 3.19).

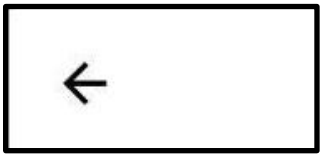

Рисунок 3.19 — Стрілка для повернення до роботи з нотаткою

Про активацію функції повернення до нотатки буде свідчити підсвітка стрілки повернення сірим колом, як це показано на рис. 3.20.

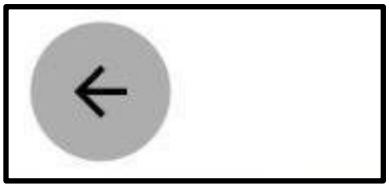

Рисунок 3.20 — Стрілка для повернення до роботи з нотаткою в активованому стані

Після повернення до роботи з нотаткою користувач побачить наступне: в полі “Note…” з'явиться відсканований в попередньому кроці текст, як показано на рис. 3.21. Текст буде максимально можливо зберігати параметри відсканованого тексту: співвідношення розмірів літер, абзаци, тощо.

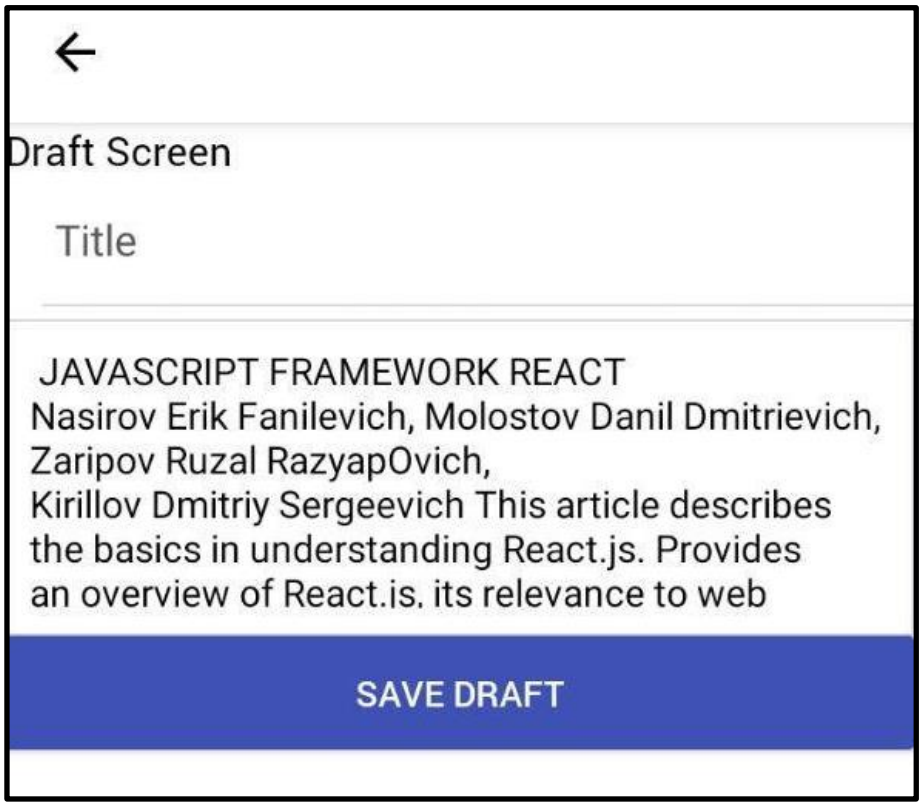

Рисунок 3.21 — Відсканований текст в полі “Note…”

Необхідно зазначити, що за абзац программа приймає будь який текст, якщо він починається з нового рядку і при цьому виділений більшою відстанню від початку абзацу ніж решта тексту, що можна спостерігати на рис. 3.21, де формально список прізвищ являються одним реченням, але на забраженні, з якого було проведено сканування, частина прізвищ була перенесена на рядок нижче, що програма і розпізнала як абзац.

Наступним кроком буде тестування функцій редагування нотатки. Для цього треба обрати зону в тексті, бажану до редагування. В прикладі на рис. 3.22 нові символи вводяться після основного тексту. Як можна побачити на рис. 3.22 в полі нотатки немає обмежень на формат символів що вводяться (літери, цифри, спецсимволи).

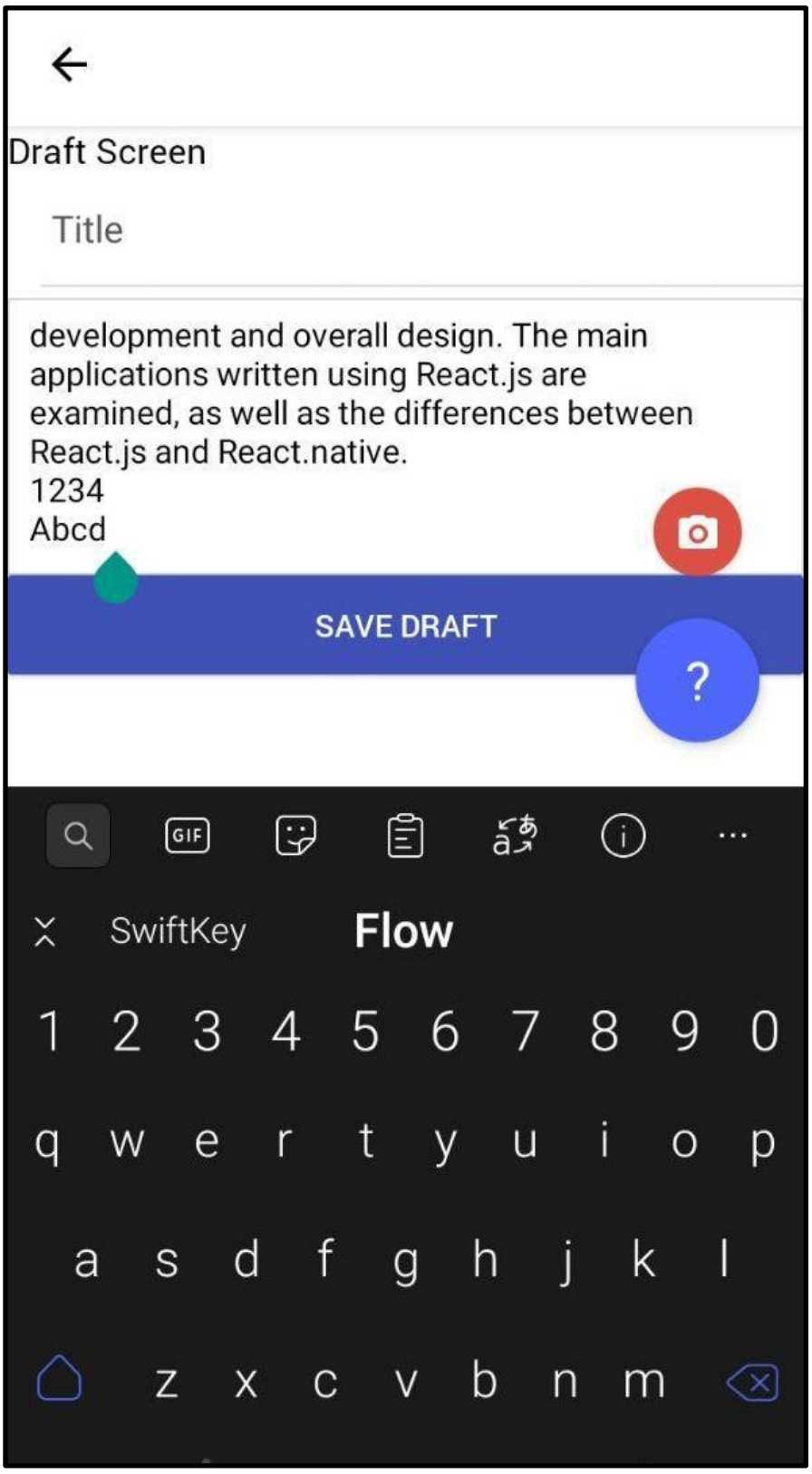

Рисунок 3.22 — Редагування нотатки, доповнення

Перевіривши можливість редагування тексту, наступним кроком буде збереження нотатки. Як вже зазначалося раніше, нотатку не буде збережено,

якщо їй не буде присвоєно ім’я, тож Спочатку необхідно ввести ім’я нотатки. На рисунку 3.23 продемонстровано поле “Title” після заповнення.

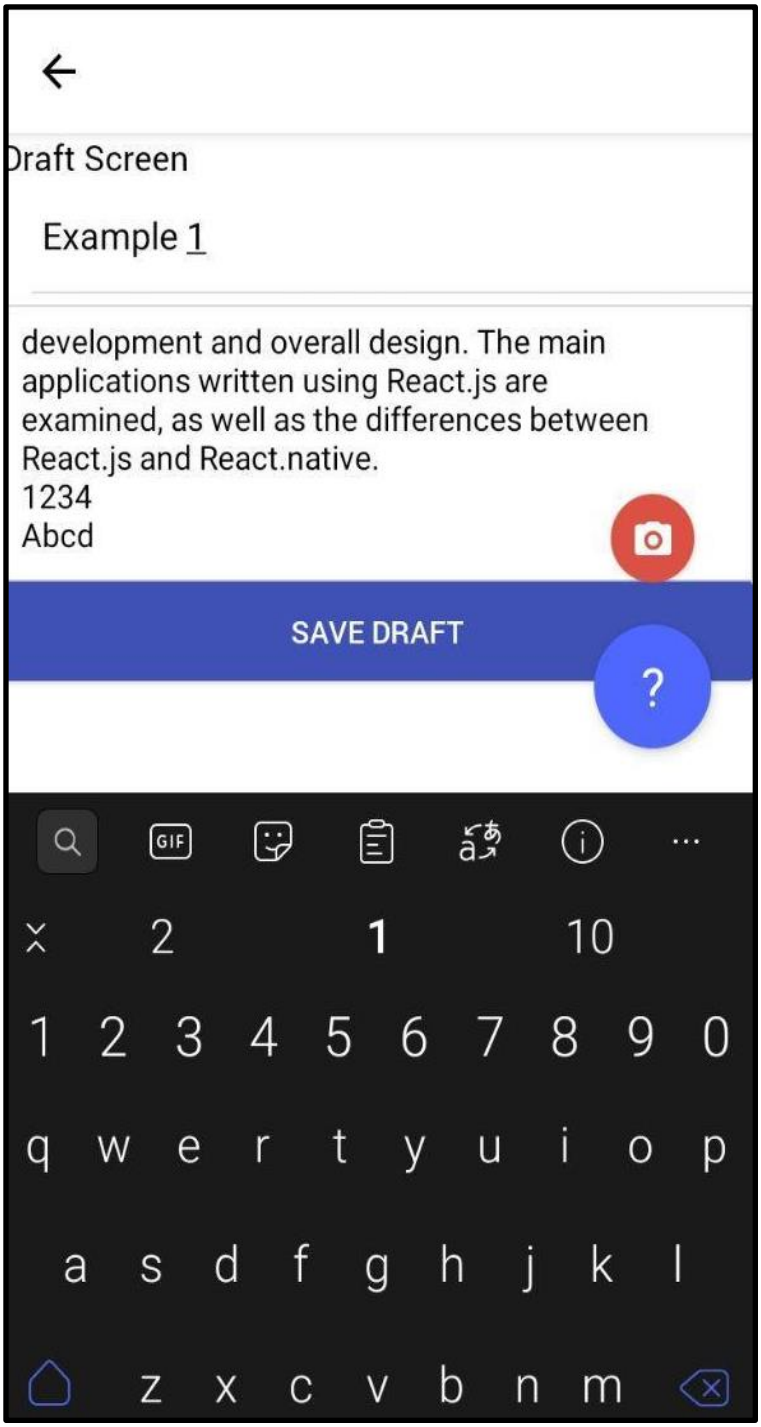

Рисунок 3.23 — Привласнення імені нотатці

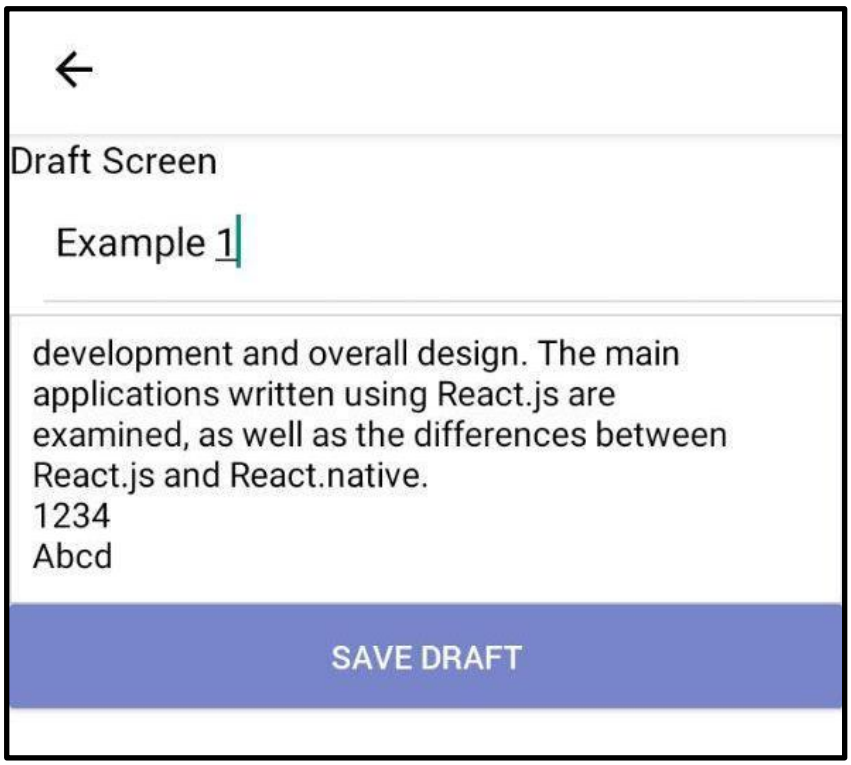

Рисунок 3.24 — Збереження нотатки

Після того як нотатку було успішно поіменовано можно перейти до збереження. Для збереження необхідно натиснути кнопку “SAVE DRAFT”. Про активацію функції збереження буде свідчити зміна кольору кнопки “SAVE DRAFT” з синього на блідо—блакитний.

В разі, якщо користувач не ввів ім’я то після натискання кнопки “SAVE DRAFT” він повернеться на початковий екран, на рис. 3.12 і створення нотатки необхідно буде почати з нуля.

Після успішного збереження, в нотатці з’явиться нова кнопка : “DELETE DRAFT”, як це зображено на рисунку 3.25. Кнопка “DELETE DRAFT” відповідає за функцію видалення нотатки.

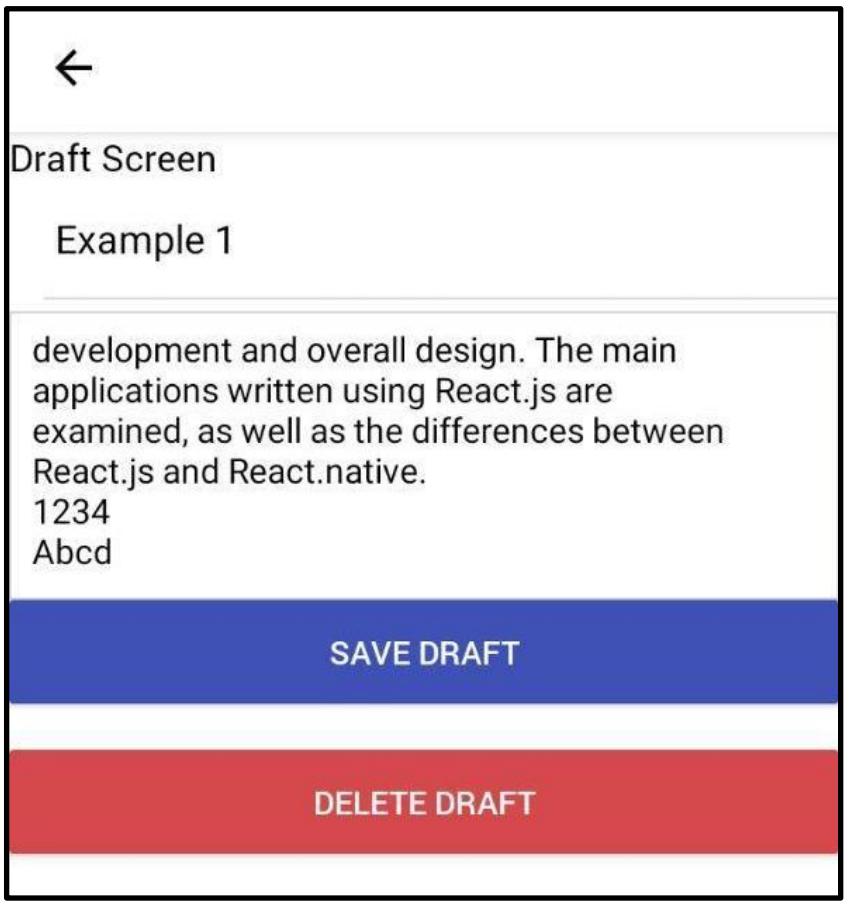

Рисунок 3.25 — Вигляд нотатки після збереження

Якщо нотатку було успішно збережено, користувач може повернутися на головний екран, без остраху що створена нотатка зникне. Для повернення на головний екран необхідно натиснути стрілку “ назад” в правому верхньому куті. Про активацію функції повернення буде свідчити підсвітка в вигляді сірого кола, на тлі стрілки, як це зображено на рисунку 3.26.

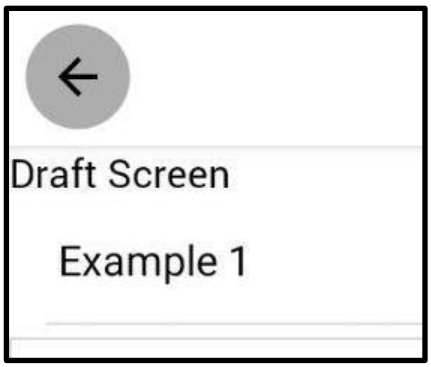

Рисунок 3.26 — Повернення на головний екран

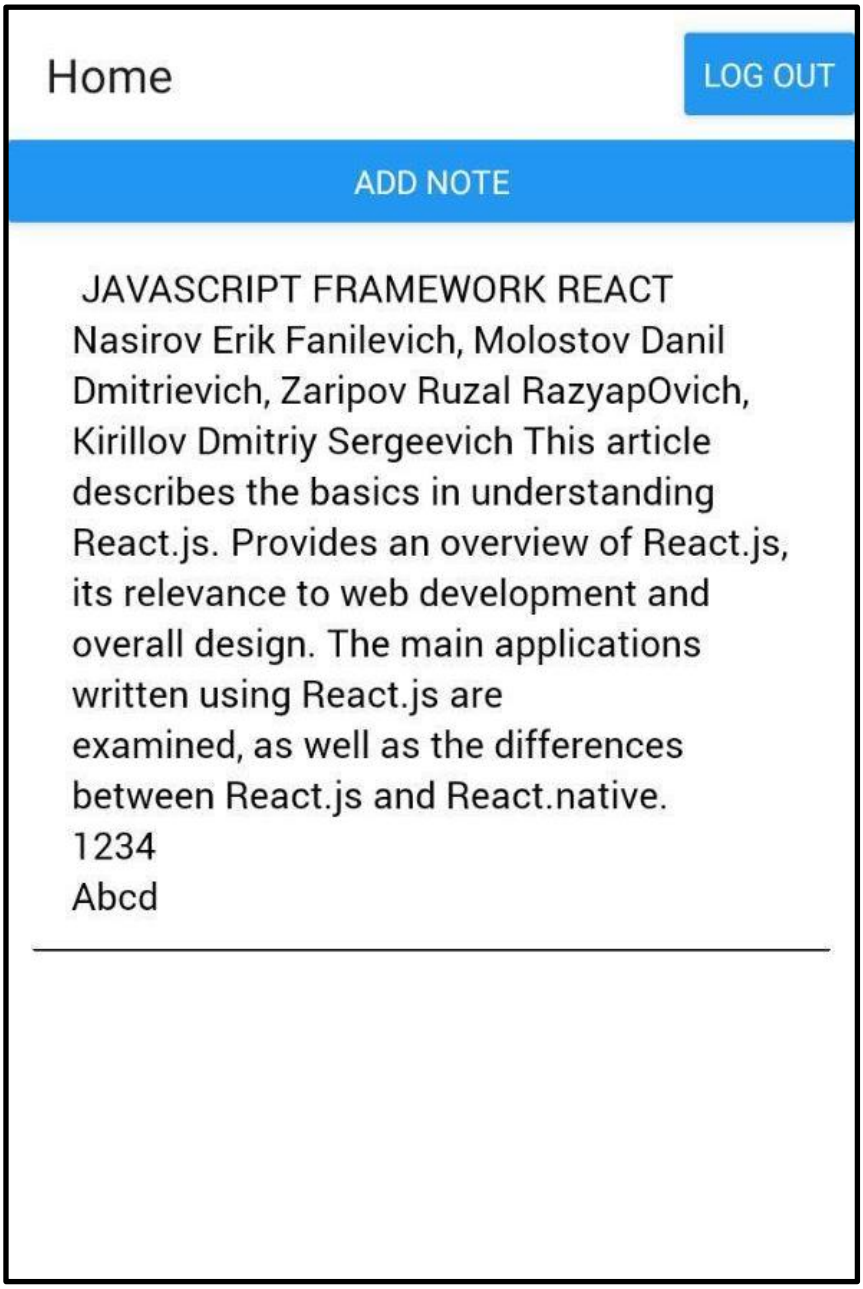

Рисунок 3.27 — Головний екран після збереження нотатки

Повернувшись на головний екран, користувач зможе побачити збережену нотатку. Нотатка відображається на головному екрані повністю. На малюнку 3.27 можна побачити і сам текст, і внесені під час редагування символи, що ще раз підтверджує успішну роботу основних функцій додатка.

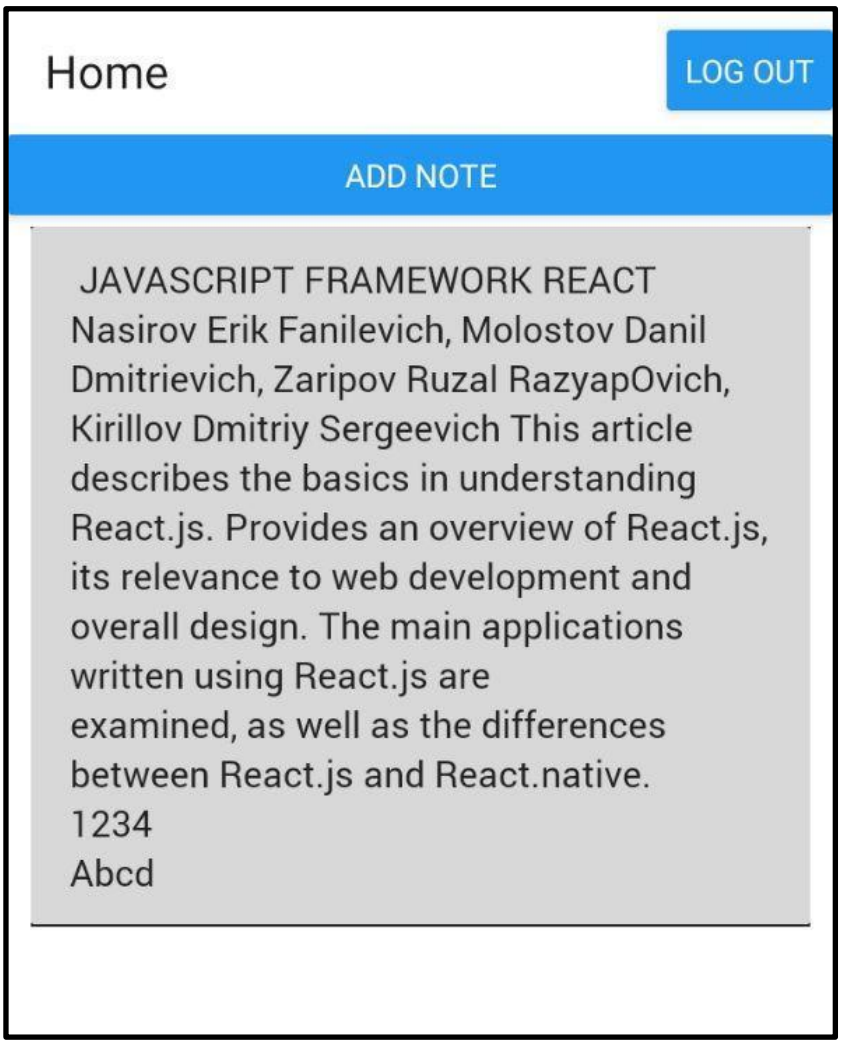

Рисунок 3.28 — Повернення до роботи з нотаткою

Щоб повернутися до роботи з нотаткою необхідно натиснути на неї на головній сторінці. Активація вибору нотатки виділить обрану нотатку сірим тлом, що буде свідчити про успішний початок процесу переходу до нотатки. Після завершення відкриття нотатки, користувач повинен побачити теж саме, що й під час попереднього завершення роботи з нотаткою. Результат відкриття обраної нотатки можна спостерігати на рисунку 3.29. Користувачу знову доступні до редагування поля “Title” та “Note”, і далі буде продемонстровано можливість повторного внесення змін в нотатку.

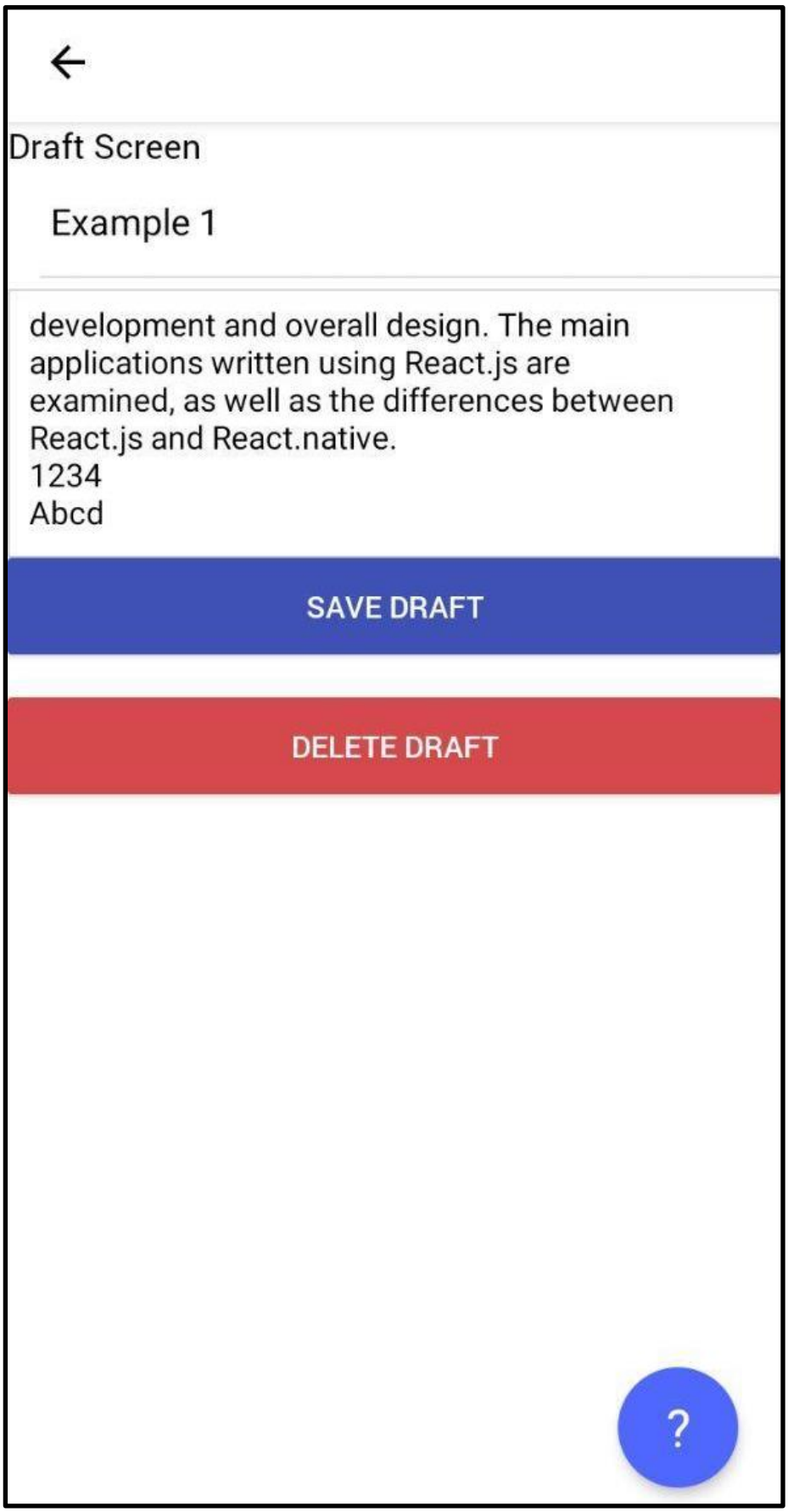

Рисунок 3.29 — Повторне відкриття нотатки для нового редагування

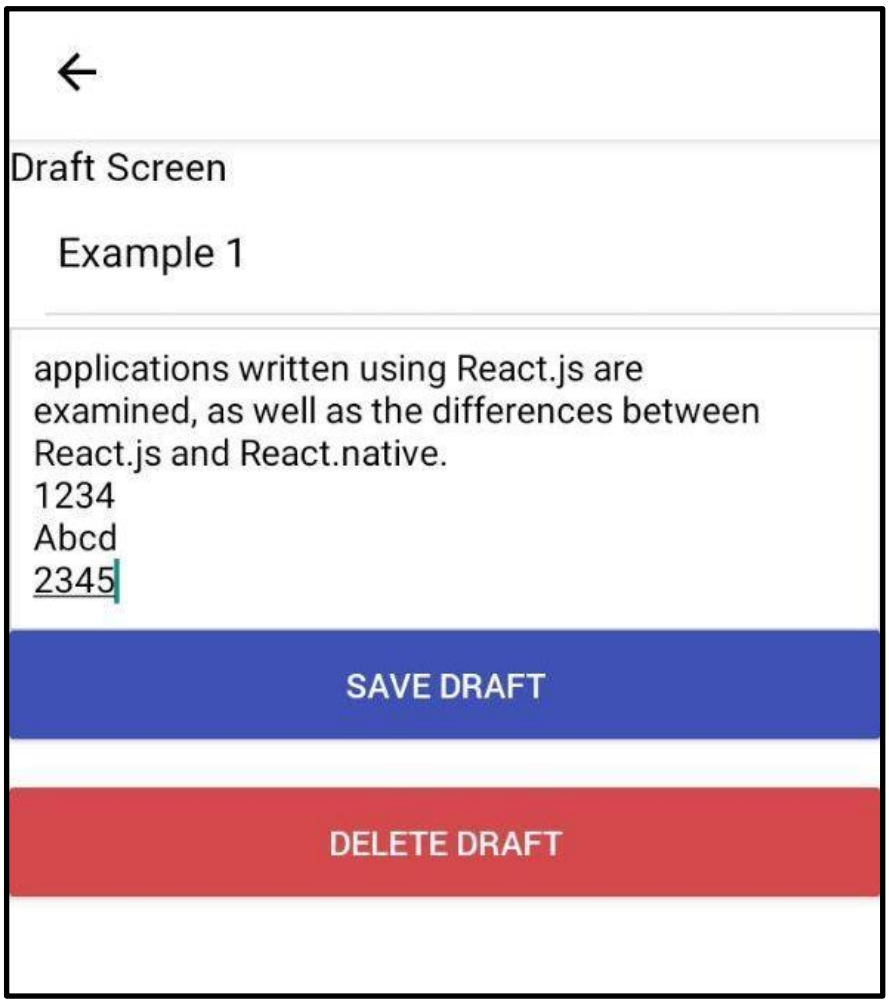

Рисунок 3.30 — Внесення нових змін

Для більшої наочності додаємо новий рядок з новим вмістом. Після, повторюємо процес збереження нових змін натиснувши кнопку “SAVE DRAFT”. Результат успішного збереження змін можна побачити на рисунках 3.31 та 3.32.

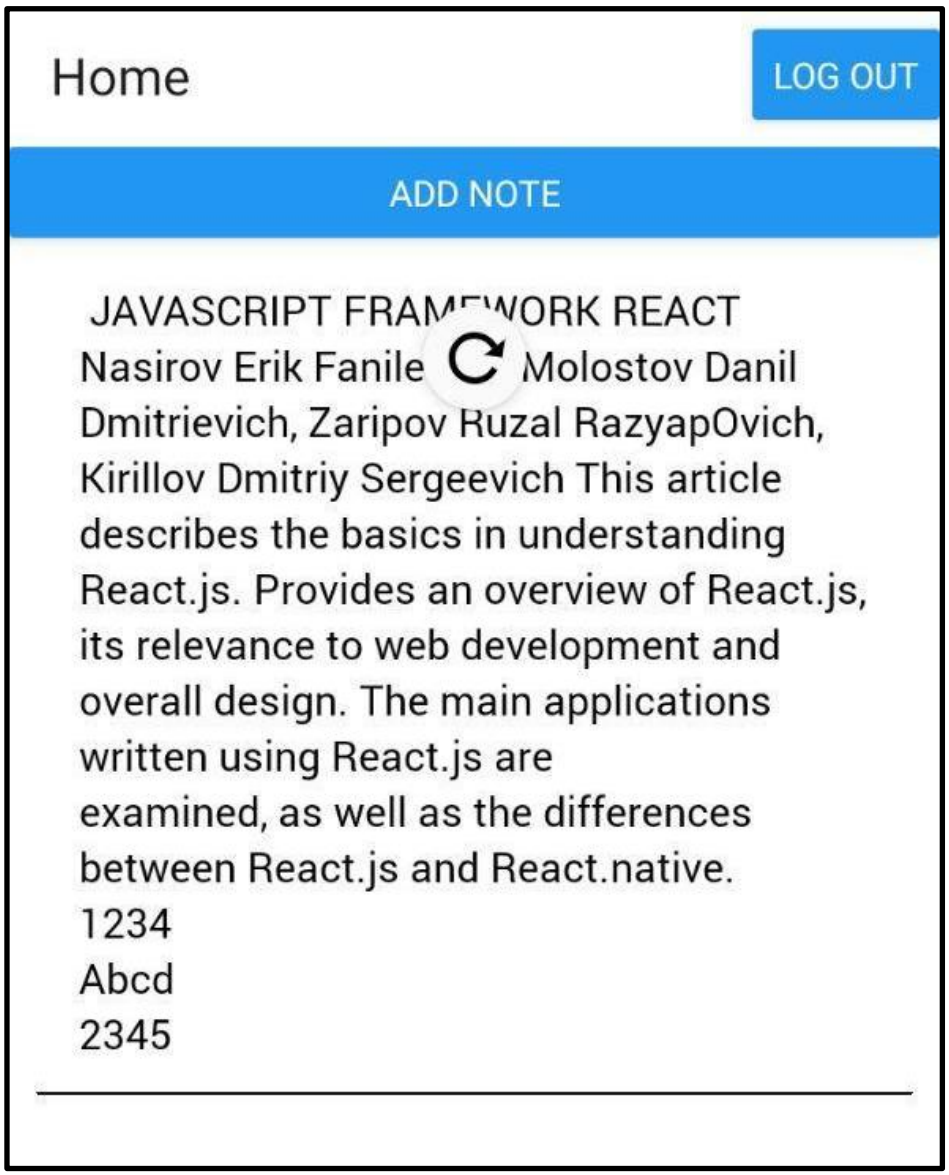

Рисунок 3.31 — Перевірка функції оновлення головної сторінки

В розробленому додаткові також наявна функція оновлення головного екрану. Вона необхідна для того щоб перезавантажити головний екран в разі збоїв роботи (на кшталт заторможення чи “підвисання”). Для активації функції треба всього лише провести пальцем по екрану вниз. Про успішну активацію функції оновлення буде свідчити поява кружечка з характерною круговою стрілкою, загально прийнятим символом загрузки даних. Положення стрілки на екрані залежить від дотику користувача під час проведення пальцем по екрану вниз: чим нижче опускає палець користувач, тим нижче опускається і стрілка. Відмінність положень можна розглянути на прикладі рисунків 3.31 та 3.32.

Рисунок 3.32 — Зміна положення символу перезавантаження

Останньою функцією котру ще не було розглянуто залишилась функція “DELETE DRAFT”. Для активації цієї функції необхідно перейти до самої нотатки, обравши її на головному екрані. Після того як нотатка відкриється користувач матиме змогу побачити функцію “DELETE DRAFT” під нотаткою на функцією “SAVE DRAFT”. Для активації функції видалення треба натиснути на кнопку “DELETE DRAFT”. Про те що функція активна буде свідчити зміна кольору кнопки з червоного на блідо—червоний, як це продемонстровано на рисунку 3.33.

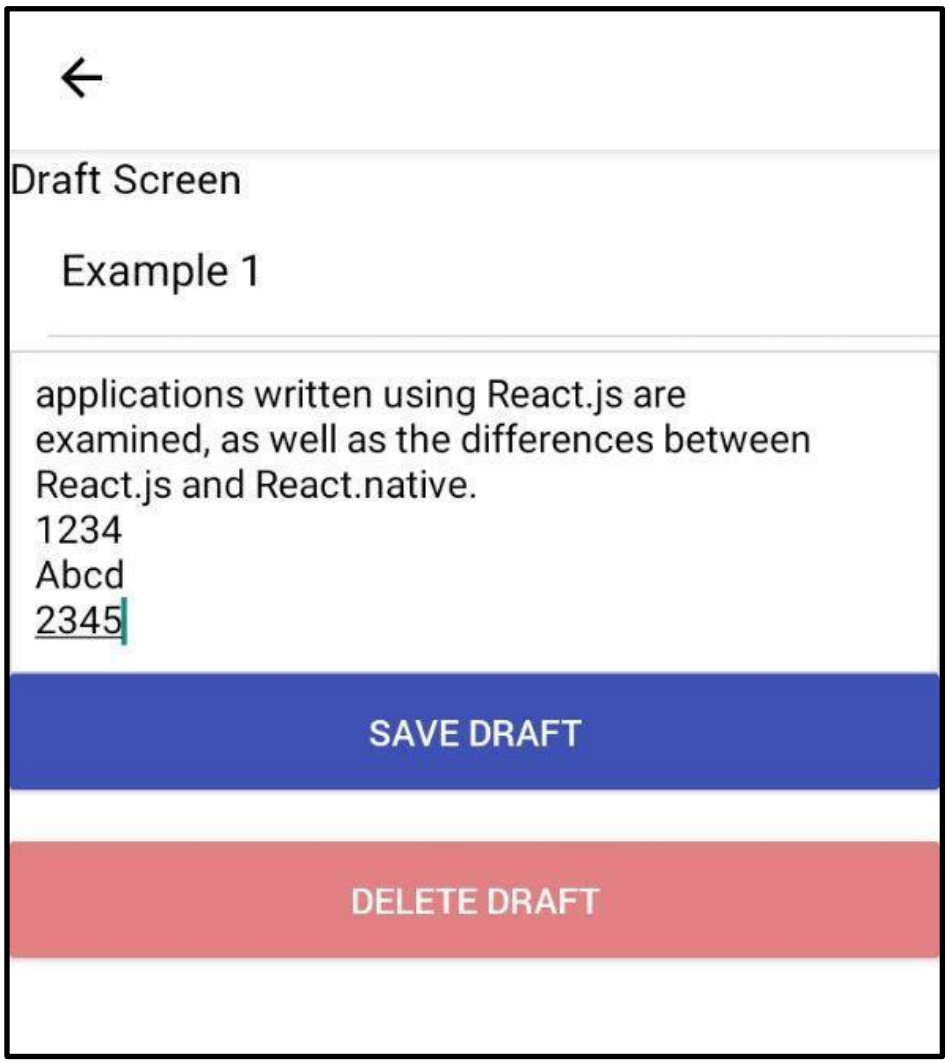

Рисунок 3.33 — активація функції видалення нотатки

Після успішного завершення видалення нотатки додаток автоматично повертає користувача на головний екран, як на рисунку 3.12.

ВИСНОВКИ

У випускній роботі було виконано розробку програмного забезпечення гібридного мобільного додатка. Було детально розглянуто методи створення гібридного мобільного додатка мовою JS на фреймворці React. Було оцінено актуальність і необхідність створення додатка, а також розглянуто існуючі альтернативи. Проведено аналіз існуючих аналогів, визначено їх переваги та недоліки. Проведено порівняння альтернатив з розроблюваним додатком. Для розробки додатка було проведено аналіз мови JS, її засобів та бібліотек. Обрано та розглянуто для розробки фреймворк React, розглянуто та оцінено його методи та переваги. Розглянуто версії та деякі історичні відомості мови JS для її більш детальної та точної оцінки. Розглянуто представників відомих та впливових компаній та сервісів, що використовують саме JS та його методи і бібліотеки.

Було розглянуто обрані для створення додатка компоненти розробки. Для валідації користувача було обрано і розглянуто бібліотеку Formik . Авторизація користувача в системі відбувається через Firebase. Реалізовано навігацію по додаткові через react—navigation. Обрано Firestore для збереження нотаток користувача. Розглянуто та обрано документо подібну базу данных NoSQL.

Для перевірки працездатності додатка було проведено її бетта—тестування.

СПИСОК ЛІТЕРАТУРИ

1. “ Google translator ” — https://support.google.com/translate/answer/6142483?co=GENIE.Platform%3DAndroid&hl=ua
2. “Adobe Scan: сканування PDF, OCR” — https://play.google.com/store/apps/details?id=com.adobe.scan.android&hl
3. “Розробка гібридних мобільних додатків React native” — https://artjoker.ua/ua/uslugi/razrabotka-gibridnykh-mobilnykh-prilozheniy/
4. “Model-view-controller, MVC” — https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C-%D0%B2%D0%B8%D0%B4-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80
5. “ AngularJS ” — https://docs.angularjs.org/guide/overview
6. “ Redux ” — http://getinstance.info/articles/react/learning-react-redux/
7. “ ReactDOM ” — https://ru.reactjs.org/docs/react-dom.html
8. “ JSX ” — https://ru.reactjs.org/docs/introducing-jsx.html
9. “ JSON (JavaScript Object Notation)” — https://www.json.org/json-ru.html
10. “Get started with Cloud Firestore Security Rules” — https://firebase.google.com/docs/firestore/security/get-started
11. “Add Firebase to your Android project” — https://firebase.google.com/docs/android/setup
12. “How to import data to Cloud Firestore?” —https://hackernoon.com/filling-cloud-firestore-with-data-3f67d26bd66e
13. “React Components reusable components by Khan Academy” — https://khan.github.io/react-components/
14. “Netflix React Clone!” https://www.reddit.com/r/reactjs/comments/a1jicn/netflix_react_clone/

15.“React JS розробники для вашого проекту” — https://drupal-coder.ru/react
16.“React і Atlaskit в серверних і датацентрових плагінах Atlassian” — https://habr.com/ru/post/501300/
17. “React Native в Airbnb” — https://medium.com/devschacht/sunsetting-react-native-cae979e08eb2
18.“Authenticate React with Yahoo! using Auth0 ” — https://auth0.com/authenticate/react/yahoo/

ДОДАТОК А

Код авторизації в застосункові:

```
import React, { Component } from 'react';
import { StyleSheet } from 'react-native';
import { Container, Content, Form, Item, Input, Label, Button, Text } from
'native-base';
const styles = StyleSheet.create({
    center: {
        flex: 1,
        textAlign: 'center',
        justifyContent: 'center',
        alignItems: 'center',
    },
    validationText: {
        marginTop: 8,
        marginBottom: 16,
        color: 'red',
        alignSelf: 'center'
    },
});
export default class SignIn extends Component{
    toSignUp = () => {
        const { signUpStatus } = this.props;
        signUpStatus(true);
    };
    render() {
        const { handleSubmit, setFieldValue, errors, signUp } = this.props;
        return (
            <Container>
                <Content>
                    <Form>
                        <Item floatingLabel>
                            <Label>Email</Label>
                            <Input
         onChangeText={text => setFieldValue('email', text)}
                            />
                        </Item>
     <Text style={styles.validationText}>{errors.email}</Text>
                        <Item floatingLabel>
                            <Label>Password</Label>
```

```
                            <Input
                                secureTextEntry={true}
        onChangeText={text => setFieldValue('password', text)}
                            />
                        </Item>
        <Text style={styles.validationText}>{errors.password}</Text>
                        <Button
                            onPress={() => {
                                setFieldValue('signUp', signUp);
                                setTimeout(() => handleSubmit(),0);
                            }}
                            style={styles.center}
                            primary
                        >
                            <Text>Sign in</Text>
                        </Button>
                        <Button   style={styles.center}   onPress={this.toSignUp}
transparent>
                            <Text>Sign up</Text>
                        </Button>
                    </Form>
                </Content>
            </Container>
        );
    }
}
```

Код реєстрації в застосункові:

```
import React, { Component } from 'react';
import { StyleSheet } from 'react-native';
import { Container, Content, Form, Item, Input, Label, Button, Text } from
'native-base';
const styles = StyleSheet.create({
    center: {
        flex: 1,
        textAlign: 'center',
        justifyContent: 'center',
        alignItems: 'center',
    },
    validationText: {
        marginTop: 8,
        marginBottom: 16,
```

```
        color: 'red',
        alignSelf: 'center'
    },
});
export default class SignUp extends Component{
    toSignIn = () => {
        const { signUpStatus } = this.props;
        signUpStatus(false);
    };
    render() {
        const { handleSubmit, setFieldValue, errors, signUp } = this.props;
        return (
            <Container>
                <Content>
                    <Form>
                        <Item floatingLabel>
                            <Label>Email</Label>
                            <Input
      onChangeText={text => setFieldValue('email', text)}
                            />
                        </Item>
      <Text style={styles.validationText}>{errors.email}</Text>
                        <Item floatingLabel>
                            <Label>Username</Label>
                            <Input
  onChangeText={text => setFieldValue('displayName', text)}
                            />
                        </Item>
   <Text style={styles.validationText}>{errors.displayName}</Text>
                        <Item floatingLabel last>
                            <Label>Password</Label>
                            <Input
                                secureTextEntry={true}
          onChangeText={text => setFieldValue('password', text)}
                            />
                        </Item>
          <Text style={styles.validationText}>{errors.password}</Text>
                        <Button
                            onPress={() => {
                                setFieldValue('signUp', signUp);
                                setTimeout(() => handleSubmit(),0);
                            }}
```

```
                            style={styles.center}
                            primary
                        >
                            <Text>Register</Text>
                        </Button>
                        <Button   style={styles.center}   onPress={this.toSignIn}
transparent>
                            <Text>Sign In</Text>
                        </Button>
                    </Form>
                </Content>
            </Container>
        );
    }
}
```

Код навігації по застосунку:

```
import { createSwitchNavigator, createAppContainer, createStackNavigator } from
'react-navigation'

// import TextDetector from 'App/Containers/TextDetection'

/**
 * The root screen contains the application's navigation.
 *         *         @see         https://reactnavigation.org/docs/en/hello-react-
navigation.html#creating-a-stack-navigator
 */
// const StackNavigator = createStackNavigator(
//   {
//      // Create the application routes here (the key is the route name, the
value is the target screen)
//                        //     See     https://reactnavigation.org/docs/en/stack-
navigator.html#routeconfigs
//       // The main application screen is our "ExampleScreen". Feel free to
replace it with your
//     // own screen and remove the example.
//     MainScreen: TextDetector,
//     TextDetector: TextDetector,
//   },
//   {
//     // By default the application will show the splash screen
//     initialRouteName: 'TextDetector',
```

```
//                      //    See    https://reactnavigation.org/docs/en/stack-
navigator.html#stacknavigatorconfig
//     headerMode: 'none',
//   }
// )
import HomeScreen from 'App/Containers/HomeScreen'
import DraftScreen from 'App/Containers/DraftScreen'
import AuthScreen from 'App/Containers/AuthScreen'
import TextDetection from 'App/Containers/TextDetection'
const AuthStack = createStackNavigator({
      MainScreen: AuthScreen,
})
const AppStack = createStackNavigator({
      MainScreen: HomeScreen,
      Draft: DraftScreen,
      Recognizer: TextDetection,
})
export default createAppContainer(
      createSwitchNavigator(
            {
                  App: AppStack,
                  Auth: AuthStack,
            },
            {
                  initialRouteName: 'Auth',
            }
      )
)
```

Код головного екрану:

```
import React from 'react';
import    {Button,    View,    SafeAreaView,    ScrollView,    StyleSheet,
TouchableHighlight, RefreshControl} from "react-native";
import {Input, Item, Label, List, ListItem, Text, CardItem} from "native-
base";
import { SwipeListView } from 'react-native-swipe-list-view';
import {addNote, getNotes} from '../api/notes';
import { signOut } from "../api/auth";
function wait(timeout, callback) {
    return new Promise(resolve => {
        setTimeout(async () => {
```

```
            await getNotes(callback);
            resolve();
        }, timeout);
    });
}
export default class HomeScreen extends React.Component {
    static navigationOptions({ navigation }) {
        const onSignedOut = () => {
            console.log('signed out');
            navigation.navigate('Auth');
        };
        return {
          title: 'Home',
          headerRight: (
            <Button
                title="Log out"
                onPress={() => signOut(onSignedOut)}
            />
          ),
        };
    };
    state = {
        notes: [],
        currentNote: {},
        refreshing: false,
    };
    onNoteAdded = note => {
        this.setState(state => ({notes: [...state.notes, note]}));
    };
    onNotesReceived = notesList => {
        this.setState(state => ({notes: notesList}));
    };
    onDraftSubmit = draft => {
      addNote({
          title: draft.title,
          note: draft.note,
      }, this.onNoteAdded);
    };
```

```
    componentDidMount() {
        getNotes(this.onNotesReceived);
    }

    onEditNote = data => {
        const { item: { noteTitle, note, noteId } } = data;
        this.props.navigation.navigate('Draft', {
            draftSubmit: this.onDraftSubmit,
            editSubmit: this.onNotesReceived,
            edit: true,
            noteTitle,
            noteId,
            note,
        });
    };

    onRefresh = () => {
        this.setState(state => {
            const { refreshing } = state;
            return { refreshing: !refreshing };
        }, () => {
            wait(2000,  this.onNotesReceived).then(()  =>  this.setState({
refreshing: false }));
        })
    };

    render() {
        const {notes, currentNote, refreshing} = this.state;
        console.log(notes, currentNote);
        return (
            <>
                <View style={{ flex: 2 }}>
                    <View>
                        <Button title={'Add note'} onPress={() => {
                            this.props.navigation.navigate('Draft', {
                                draftSubmit: this.onDraftSubmit,
                            });
```

```
                        }} />
                    </View>
                    <SafeAreaView>
                        <ScrollView refreshControl={
                            <RefreshControl       refreshing={refreshing}
onRefresh={this.onRefresh} />
                        }>
                            <SwipeListView
                                style={{ paddingTop: 5, paddingRight: 10,
paddingBottom: 10, paddingLeft: 10 }}
                                data={notes}
                                renderItem={ (data) => (
                                    <TouchableHighlight   onPress={()   =>
this.onEditNote(data)}>
                                        <CardItem                style={{
borderBottomWidth: 1 }}>
                                            <Text>{data.item.note}</Text>
                                        </CardItem>
                                    </TouchableHighlight>
                                )}
                            />
                        </ScrollView>
                    </SafeAreaView>
                </View>
            </>
        );
    }
}

const styles = StyleSheet.create({
    rowFront: {
        // alignItems: 'center',
        // backgroundColor: '#CCC',
        // borderBottomColor: 'black',
        // borderBottomWidth: 1,
        // justifyContent: 'center',
        // height: 50,
    },
```

```
    rowBack: {
        // alignItems: 'center',
        // backgroundColor: '#DDD',
        // flex: 1,
        // flexDirection: 'row',
        // justifyContent: 'space-between',
        // paddingLeft: 15,
    },
});
```

Код розпізнавання тексту:

```
import React, { PureComponent } from 'react';
import { AppRegistry, StyleSheet, Text, TouchableOpacity, View } from 'react-native';
import { RNCamera } from 'react-native-camera';
import RNTextDetector from "react-native-text-detector";

export default class ExampleApp extends PureComponent {
    render() {
        return (
            <View style={styles.container}>
                <RNCamera
                    ref={ref => {
                        this.camera = ref;
                    }}
                    style={styles.preview}
                    type={RNCamera.Constants.Type.back}
                    flashMode={RNCamera.Constants.FlashMode.on}
                    androidCameraPermissionOptions={{
                        title: 'Permission to use camera',
                        message: 'We need your permission to use your camera',
                        buttonPositive: 'Ok',
                        buttonNegative: 'Cancel',
                    }}
                    androidRecordAudioPermissionOptions={{
                        title: 'Permission to use audio recording',
                        message: 'We need your permission to use your audio',
                        buttonPositive: 'Ok',
                        buttonNegative: 'Cancel',
                    }}
                />
```

```
                <View style={{ flex: 0, flexDirection: 'row', justifyContent:
'center' }}>
                    <TouchableOpacity      onPress={this.takePicture.bind(this)}
style={styles.capture}>
                        <Text style={{ fontSize: 14 }}> Recognize </Text>
                    </TouchableOpacity>
                </View>
            </View>
        );
    }
    takePicture = async() => {
        if (this.camera) {
            try {
                const options = {
                    quality: 0.8,
                    base64: true,
                    skipProcessing: true,
                };
                const { uri } = await this.camera.takePictureAsync(options);
    const onRecognized = this.props.navigation.getParam('onRecognized');
                const visionResp = await RNTextDetector.detectFromUri(uri);
                let textImage = '';
                visionResp.forEach(item => {
                    textImage += item.text + ' ';
                });
                onRecognized(textImage);
                // console.log('visionResp', textImage);
            } catch (e) {
                console.warn(e);
            }
        }
    };
}
const styles = StyleSheet.create({
    container: {
        flex: 1,
        flexDirection: 'column',
        backgroundColor: 'black',
    },
    preview: {
        flex: 1,
        justifyContent: 'flex-end',
```

```
        alignItems: 'center',
    },
    capture: {
        flex: 0,
        backgroundColor: '#fff',
        borderRadius: 5,
        padding: 15,
        paddingHorizontal: 20,
        alignSelf: 'center',
        margin: 20,
    },
})
```