Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема Розробка мобільного додатку для організації поштових повідомлень

Спеціальність 121 “Інженерія програмного забезпечення”
код і назва спеціальності

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДП.ІПЗ-18.ПЗ
(позначення)

Рецензент
доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

Студент
ІПЗ-41 Головецький П.І.
(шифр групи) (підпис) (дата) (розшифровка підпису)

Нормоконтролер
доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

Керівник дипломного проекту
доц. Лазарович І.М.
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

2020
(рік)

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет Математики та інформатики Кафедра та інформаційних технологій
Спеціальність 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри Козленко М.І.
_____
„_____”_____ 20___р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Головецькому Петру Івановичу
(прізвище, ім’я, по батькові студента)

1. Тема проекту Розробка мобільного додатку для організації поштових повідомлень
затверджена наказом від „25” жовтня 2020р .№7
2. Термін здачі студентом закінченого проекту
3. Вихідні дані до дипломного проекту існуючі засоби для відстеження поштових відкправлень, теорія та інструментальні засоби моделювання, розробки та тестування ПЗ, методи аналізу економічної доцільності проекту
4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати) “Аналіз вітчизняної та зарубіжної літератури за напрямком дослідження ” , “Аналіз основних характеристик досліджуваного процесу”, “Розробка програмного забезпечення додатку Warehouse Manager”, “Бізнес-план впровадження мобільного додатку Warehouse Manager в роботу служб Нова Пошта та Укрпошта”
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень) “Мета роботи”, “Огляд функціоналу”, “Переваги додатку”, “Принцип роботи додатку”, “Використані технології”, “Порівняння з аналогічними технологіями”, “Архітектура додатку”, “Актуальність додатку”

6. Дата видачі завдання 11.09.2019

Керівник _____ Лазарович І.М.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Головецький П.І.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
1. Аналіз вітчизняної та зарубіжної літератури за напрямком роботи	02.12.2019	Виконав
2. Розробка структурних рішень	15.02.2020	Виконав
3. Розробка програмного забезпечення додатку Warehouse Manager	17.04.2020	Виконав
4. Бізнес-план впровадження мобільного додатку Warehouse Manager в роботу служб Нова Пошта та Укрпошта	11.05.2020	Виконав

Студент _____Головецький Петро Іванович
(підпис) (розшифровка підпису)

Керівник проекту _____Лазарович Ігор Миколайович
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 77 сторінок (без додатків), 32 рисунків, 1 таблиць, 47 джерел, 1 додатків на 33 сторінках.

Ключові слова: МОБІЛЬНИЙ ПРИСТРІЙ, МОБІЛЬНИЙ ДОДАТОК, WAREHOUSE MANAGER, JAVASCRIPT, REACT NATIVE, REDUX, FIREBASE, НОВА ПОШТА, УКРПОШТА,

Об`єктом дослідження є мобільний додаток для організації поштових повідомлень.

Мета роботи: теоретично проаналізувати порядок розробки та практично розробити мобільний додаток для організації поштових повідомлень.

Стислий опис тексту пояснювальної записки:

У роботі перелічено основні види та призначення мобільних додатків для поштових повідомлень. Дано коротку характеристику таких основних видів сучасних мобільних додатків, що працюють з поштовими повідомленнями, як Pushover, Мобільний додаток від Нова Пошта, Мобільний додаток Укрпошта, Addappt, TravelPost, App Store Preview.

Описано головні особливості таких технологій, як JavaScript, React Native, Redux, Firebase, вказано їх переваги та недоліки. Описано етапи створення програмного додатку Warehouse Manager. Проаналізована економічна ефективність розроблюваної системи та розроблена необхідна програмна документація.

ABSTRACT

Explanatory note: 77 pages (without appendix), 32 figures, 1 tables, 47 references, 1 appendix on 33 pages.

Key words: MOBILE DEVICE, MOBILE APPLICATION, МОБІЛЬНИЙ ДОДАТОК, WAREHOUSE MANAGER, JAVASCRIPT, REACT NATIVE, REDUX, FIREBASE, NOVA POSHTA, UKRPOSHTA,

Object of study: mobile application for organizing post messages.

Brief description of the text of the explanatory note:

Theoretically analyzing the order of development and practically develop a mobile application for organizing e-mails.

Brief description of the text explanatory note:

The paper lists and the main types: purposes of mobile applications for e-mail messages. At the brief described the main types of modern mobile applications that work with e-mails, such as Pushover, Mobile application from Nova Poshta, Mobile application Ukrposhta, Addappt, TravelPost, App Store Preview.

The main features of such technologies as JavaScript, React Native, Redux, Firebase are described, their advantages and disadvantages are indicated. The stages of creating the Warehouse Manager software application are described. The economic efficiency of the developed system is analyzed and the necessary software documentation is developed.

ЗМІСТ

ВСТУП

Потреба людини завжди залишатися в курсі подій створює велику потребу в постійному створенні нових мобільних пристроїв і гаджетів. Зважаючи на незручності використання стаціонарних комп'ютерів і ноутбуків на перший план все частіше виходять мобільні телефони та планшети.

Актуальність роботи. Мобільні пристрої є невід'ємною частиною життя сучасної людини, адже вони забезпечують комунікації між людьми, прослуховування улюбленої музики, перегляд аудіовізуальної інформації, можуть бути блокнотом і мають ще безліч додаткових функцій. По своїй суті, мобільний смартфон це копія комп'ютера, яку постійно можна мати при собі.

З урахуванням розвитку інформаційних технологій, з року в рік з'являються все більш потужні мобільні пристрої. Отже, істотно підвищуються вимоги до додатків, призначених для цих пристроїв. Існують різні мобільні платформи, і для кожної є свій інструментарій розробки. Можуть бути розроблені додатки, які будуть працювати на кількох платформах, тобто вони кросплатформенні. Однак потрібно розуміти, що можливості програмного забезпечення, написаного для певної платформи, ширше, тому логічніше буде встановлювати відповідне програмне забезпечення на відповідну платформу.

З моменту появи перших мобільних додатків Java завжди була однією з найпопулярніших мов, що використовується розробки таких додатків. Таким він залишається і донині, незважаючи на появу нових платформ. У наш час, найпоширенішою оперативною системою є ОС Android. Android підтримує велику кількість пристроїв, від різних виробників, а головна причина поширення ОС Android – безкоштовні засоби розробки, в той час як розробка під систему IOS вимагає високих початкових витрат [1].

Мета роботи – теоретично проаналізувати порядок розробки та практично розробити мобільний додаток для організації поштових повідомлень (на прикладі служб Нова Пошта та Укрпошта).

Завдання на проектування можна сформулювати так:

- провести аналіз вітчизняної та зарубіжної літератури за напрямком дослідження, назвати основні види та призначення мобільних додатків для поштових повідомлень;
- дати характеристику операційної системи Android 6 та перелічити її ключові особливості;
- описати загальні поняття розробки додатків для мобільних пристроїв та назвати програми, що необхідні для їх розробки;
- вказати головні структурні елементи для програмування мобільного додатку;
- зробити опис мобільного додатку Warehouse Manager, провести його тестування та вказати шляхи вдосконалення додатку;
- розробити бізнес-план впровадження мобільного додатку Warehouse Manager в практику (Нова Пошта та Укрпошта);
- провести оцінку ефективності від використання розробленого продукту.

Практична цінність роботи полягає в наявності теоретичного матеріалу по дослідженню, відсіяного з-поміж іншого в процесі пошуку інформації по темі, та в систематизації матеріалу напрямку дослідження. Проведене дослідження має більш глибокий ступінь аналізу напрямку дослідження, спираючись на попередні дослідження вчених, дисертантів та дослідників в даній проблематиці.

1 АНАЛІЗ ВІТЧИЗНЯНОЇ ТА ЗАРУБІЖНОЇ ЛІТЕРАТУРИ ЗА НАПРЯМКОМ РОБОТИ

1.1 Основні види та призначення мобільних додатків для поштових повідомлень

Сучасні інформаційні технології з їх стрімко зростаючим потенціалом і швидко зниженими витратами відкривають великі можливості для нових форм організації праці та зайнятості в рамках, як окремих корпорацій, так і суспільства в цілому. Спектр таких можливостей значно розширюється за рахунок нововведень, які впливають на всі сфери життя людей, сім'ю, освіту, роботу, географічні межі людських спільностей [2].

Недарма кажуть, що 21 століття – століття інформаційних технологій. Нинішній час разюче відрізняється від попереднього: зараз світом править техніка, а товаром виступає інформація. Стрімко розвивається ринок електроніки, адже зараз вже існує величезна кількість різних гаджетів. Мобільні телефони, електроприлади всіх типів і призначень, комп'ютери, цифрові фотоапарати і відеокамери, програвачі музичних і відео компакт-дисків, електронні носії інформації – все це частина життя сучасної людини.

На сьогоднішній день мобільні технології охоплюють все більше сфер діяльності людини. Зростання ринку мобільних додатків показує значимість, зручність і актуальність використання мобільних пристроїв. Внаслідок збільшення доступності мобільного інтернету спостерігається тенденція до використання мобільних додатків для здійснення таких дій, як замовлення, бронювання або покупка будь-яких товарів і послуг. Це дозволяє користувачеві уникнути необхідності самостійно здійснювати телефонний дзвінок, який може обернутися тривалим очікуванням, недоступністю абонента і витратою грошових коштів. Крім того, не завжди користувач опиняється в умовах тиші і

можливості говорити по телефону, що робить використання мобільного додатку зручніше, ніж здійснення дзвінка [3].

Мобільний додаток – це спеціально розроблений додаток під конкретну мобільну платформу (iOS, Android, Windows Phone). Багато мобільних додатків встановлені на самому пристрої або можуть бути завантажені на нього з онлайнових магазинів додатків, таких як App Store, Google play market, Windows Phone Store – та інших, безкоштовно або за плату. Розповсюджування мобільних додатків сприяє полегшенню життя користувачів мобільних пристроїв.

На даний момент до компаній в самих різних сферах діяльності приходить усвідомлення того, що в найближчому майбутньому розробка мобільного додатку є вкрай необхідною, адже мобільний додаток має такий ряд переваг в порівнянні з мобільною версією сайту [4]:

– більш зручний і зрозумілий інтерфейс;
– програми лояльності онлайн;
– постійна комунікація з користувачем;
– геолокація;
– найбільш точний збір даних про цільову аудиторію.

Перш ніж розглядати види мобільних додатків необхідно зрозуміти, які програми користуються найбільшим попитом серед користувачів.

Більшість (за статистикою це 53 %) користуються додатками, які можна скачувати безкоштовно. Друга половина (52 %) відвідують сайти за допомогою мобільних телефонів. Третина (а саме – 38 %) використовує можливість відвідувати соціальну мережу через мобільний телефон. Деякі користувачі грають в ігри (близько 34 %). Приблизно 3 / 4 людей користуються спілкуванням через мобільний телефон: це не тільки SMS повідомлення, дзвінки, але також месенджер і додатки соціальних мереж.

Таким чином, мобільний додаток – це свого роду адаптер, що допомагає користувачеві взаємодіяти з різноманітною інформацією. У зв’язки з цим розрізняють такі типи мобільних додатків:

– додатки – події, які призначені для трансляції спортивних або інших подій;
– додатки служби, які є аналогами сайтів, що відображають діяльність організацій;
– ігри, в тому числі розвиваючі та навчальні;
– інтернет магазини, що розробляються для покупок в онлайн режимі;
– промо-програми, що використовуються для реклами різних брендів;
– бізнес-додатки, що дозволяють оптимізувати процес роботи організації, забезпечуючи доступ до ділової інформації та інтеграцію з базами даних;
– системні програми, що використовують додаткові налаштування, опції телефону і його програмного забезпечення [5];
– навігаційні та пошукові сервіси, що застосовують GPS-модуль, який дозволяє використовувати телефон як повноцінний навігатор;
– мультимедійні програми, що розширюють можливості телефону при роботі з відео- та аудіо інформацією;
– соціальні мережі, що представляють собою онлайн-сервіси для спілкування, поширення інформації та організації соціальних взаємин;
– контентні програми.

Для нашого дослідження, вибір платформи ОС Андроїд. пояснюється тим, що по-перше, така платформа найбільш поширена як у нас в Україні так і в світі (за даними дослідницької компанії IDC), по-друге, програмування під ОС Android, завдяки поширенню цієї платформи, дозволяє створювати корисні мобільні додатки практично під будь-які потреби.

Проведемо коротку характеристику основних видів сучасних мобільних додатків, що працюють з поштовими повідомленнями і які мають поширення на території України.

1. Pushover надсилає поштові повідомлення на будь-який смартфон і організовує повідомлення та сповіщення з пристроїв в одному загальному просторі. Ви можете надсилати 7,5 тис повідомлень щомісяця безкоштовно і отримувати необмежену кількість повідомлень на пристроях Android та iOS і настільних комп'ютерах [6].

2. Мобільний додаток від Нова Пошта. Існуючий мобільний додаток компанії «Нова пошта» дозволяє отримати швидкий доступ до інформації, яка допоможе зробити послугу експрес-доставки товарів по Україні ще більш простою і комфортною. Мобільний додаток «Нова пошта» з'явився в 2013 році і тоді він мав такий мінімальний набір функцій: трекінг і список відділень, до яких ще не можна було прокласти маршрут.

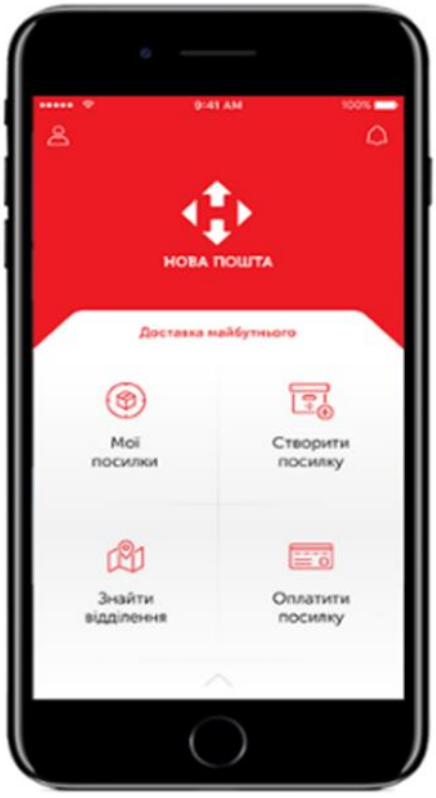

Рисунок 1.1 – Зовнішній вигляд мобільного додатку від Нової Пошти

З часом виявилося, що аудиторія була готова використовувати навіть таке просте мобільне рішення. До 2015 року в додатка з'явилося 200 тисяч активних користувачів, що є значною кількістю для тих часів, коли смартфони

ще не були на піку популярності. Втім, вже тоді мобільні технології перетворювалися на тренд, а телефони – на пріоритетний для клієнта пристрій. У 2015 році такий додаток перетворився на Приват24, який призначався для отримання повідомлень про надходження посилок користувачу від Нова Пошта. Даний додаток постійно вдосконалювався і зараз він має такі функції [7]:

Актуальний каталог всіх відділень «Нової пошти» та їх графіки роботи, а також графіки відправки вантажів у той же день і графіки прибуття відправлень до відділень;

Новини компанії, інформацію про нові послуги та сервіси, про відкриття нових відділень.

За допомогою додатку можна [8]:

– замовити повернення або переадресацію відправлення;
– створити експрес накладну з зворотною доставкою грошового переказу на банківську картку та отримати бонус знижку при відправленні;
– перегляд місця розташування відділення на карті із зазначенням основних орієнтирів для швидкого пошуку;
– прорахунок вартості та строків доставки;
– відстеження стану доставки відправлення за номером ЕН (відстежити посилку);
– відправити заявку для виклику кур’єра за адресою.

Додаток доступний російською та українською мовами.

1. Мобільний додаток Укрпошта.

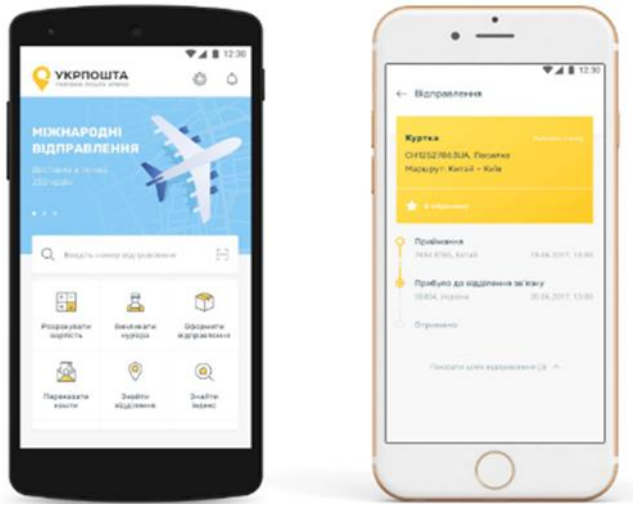

Рисунок 1.2 – Зовнішній вигляд мобільного додатку від Укрпошта

Цей мобільний додаток дозволяє користувачу відстежити відправлення посилки за рахунок отримання СМС повідомлень на телефон, провести оформлення відправлень, зробити розрахунок вартості відправлень, створити власну поштову марку, здійснити пошук відділень у місцевості де живе абонент, провести пошук індексів, здійснити переказ між картками користувача, провести переказ з картки у відділення Укрпошти.

2. Мобільний додаток Addappt спрощує керування контактами. Коли у абонента ділові контакти, друзі та родина оновлюють свою контактну інформацію, зміни автоматично поширюються на телефон користувача, якщо контакти також використовують цей додаток. Використовуючи даний додаток можна отримувати повідомлення через поштові повідомлення про надіслані повідомлення або посилки. За допомогою даного додатку можна було організувати свою книгу контактів, розбивши усіх осіб на групи – та надсилати повідомлення за допомогою програми [9].

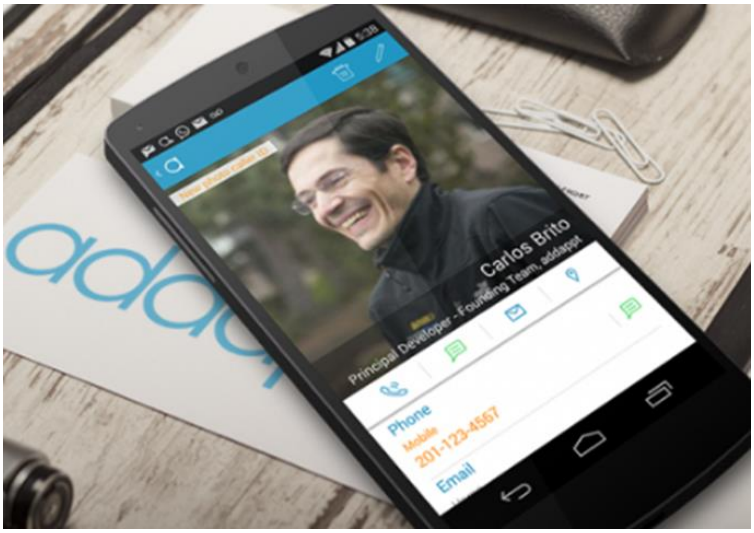

Рисунок 1.3 – Зовнішній вигляд мобільного додатку від Addappt

3. TravelPost. Сервіс для відправки посилок разом з мандрівниками і попутниками. TravelPost – це альтернатива традиційним поштовим перевезенням. У додатку ви вказуєте маршрут і ціну доставки. Далі «кур'єр» знаходить абонента і домовляється з ним про деталі угоди. Сервіс допоможе зробити доставку дешевшою і зручнішою, а мандрівникові частково компенсувати витрати на дорогу.

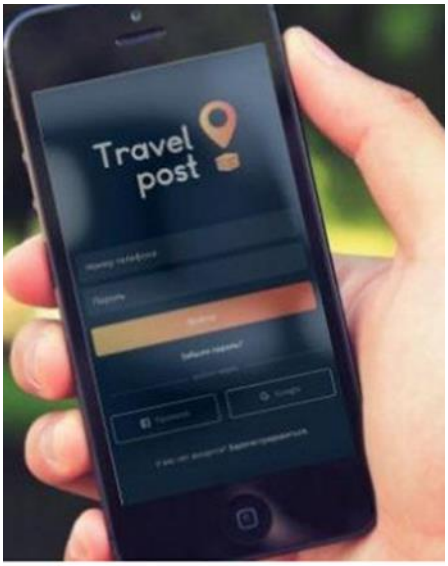

Рисунок 1.4 – Зовнішній вигляд мобільного додатку від TravelPost

У даного мобільного додатку є оцінка / відгуки про кур'єрів і клієнтів. Кожна успішна доставка робить вас надійнішим в очах інших користувачів сервісу.

4. App Store Preview. Мобільний додаток Mail. Ru для UA дозволить користувачам підключатися до поштових скриньок Yandex, Mail. Ru, Gmail тощо з будь-якої точки світу, коли завгодно, через захищений протокол без використання сторонніх VPN-додатків та інших складних способів спотворення місцезнаходження мережі [10].

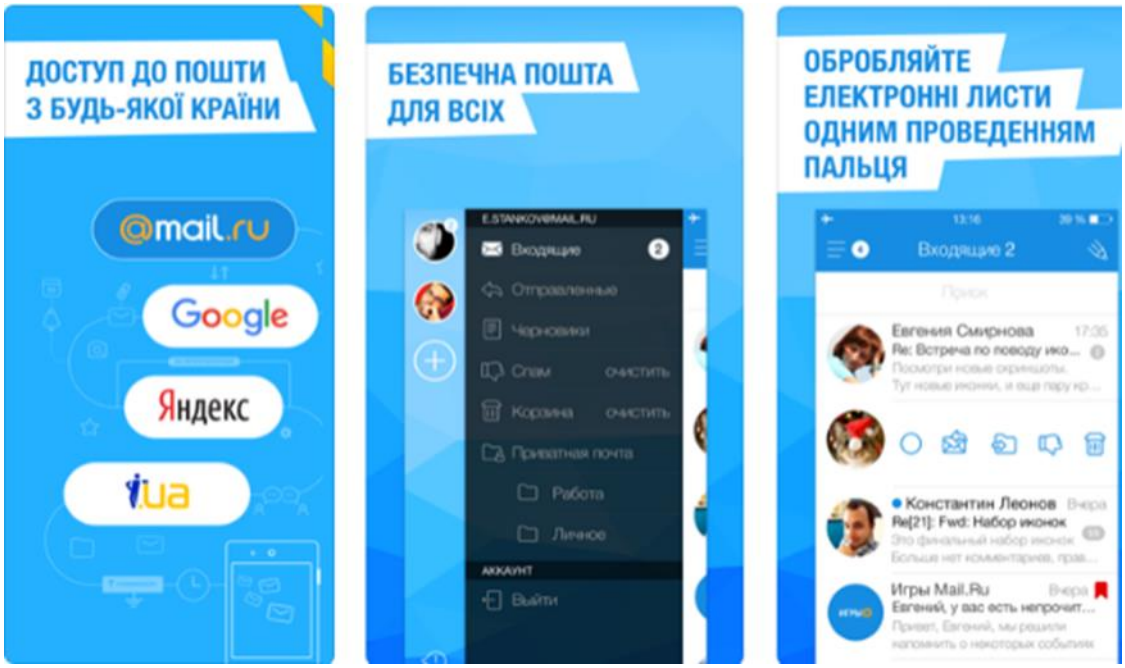

Рисунок 1.5 – Зовнішній вигляд мобільного додатку від Мобільний додаток Mail. Ru для UA

Огляд аналогів показав, що в першу чергу варто звернути увагу на надійність сервісів, використовуваних для реєстрації нових користувачів.

У розроблюваному додатку повинна бути дозволена робота без підключення до Інтернету із заздалегідь завантаженою інформацією.

Крім того, мобільний додаток повинен володіти зручним і інтуїтивно зрозумілим інтерфейсом і дизайном, що забезпечує комфортну роботу з додатком.

1.2 Характеристика операційної системи Android 6 та її ключові особливості

Із року в рік обсяги продажів пристроїв Android і кількість завантажень Android-додатків стрімко зростають. Мобільні телефони Android першого покоління з'явилися на ринку в жовтні 2008 року. За даними звіту IDC, до кінця першого кварталу 2015 року Android належало 78% глобального ринку смартфонів, в порівнянні з 18,3% у Apple, 2,7% у Microsoft і 0,3% у Blackberry. На конференції Google I / O в 2015 році компанія Google оголосила, що за попередні 12 місяців в магазині Google Play – магазині для додатків Android – кількість установок додатків досягла 50 мільярдів. Існуюча конкуренція серед розробників популярних мобільних платформ і мобільних сервісів призвела до швидкого впровадження інновацій і стрімкого обвалу цін. Завдяки суперництву між десятками виробників пристроїв Android відбулося прискорення впровадження апаратних і програмних інновацій в співтоваристві Android [11].

Однією з основних переваг платформи Android є її відкритість. Операційна система Android побудована на основі відкритого вихідного коду і знаходиться у вільному поширенні. Це дозволяє розробникам отримати доступ до вихідного коду і зрозуміти, яким чином реалізовані властивості і функції додатків. Будь-який користувач може взяти участь у вдосконаленні даної операційної системи. Для цього достатньо точно відправити звіт про виявлені помилки або взяти участь в одній з дискусійних груп Open source Project. В Інтернеті доступні різні програми Android з відкритим вихідним кодом, пропоновані компанією Google і рядом інших виробників.

На початку нашого дослідження ми більш детально наголосимо на тому, чому саме була обрана платформа Android для створення мобільних додатків. Справа в тому, що ОС Android досить проста для вивчення, вона надає достатні можливості для розробки додатків, і навіть дає можливість конкурувати з досвідченими програмістами. Причин для цього кілька [12]:

– Як вже було сказано вище, за даними глобального сервісу моніторингу Netmarketshare на травень 2019 року, гаджети на базі цієї операційної системи займають 60,99% ринку;
– Більш лояльна політика по відношенню до розробників робить ринок мобільних додатків для Android менш статичним, а значить і більш відкритим для інновацій;
– Досить широкий спектр напрямків, в яких може вестися розробка мобільних додатків для Android [13];
– Одна і та ж програма Java може бути запущена без будь-яких змін на різних комп'ютерах, наприклад PC, Apple або інших платформах. Аналогічно і мобільний додаток може бути розроблено як для смартфона і планшета, так і для Android Wear, Android TV, Android Auto і навіть Google Glass. Фактично програми, написані на Java, навіть не знають, на якому комп'ютері вони виконуються, адже вони виконуються всередині спеціальної програмної оболонки, яка називається віртуальною машиною JVM (Java Virtual Machine);
– Java дозволяє створювати програмні елементи (класи), які представляють об'єкти з реального світу. Наприклад, можна створити клас Java з ім'ям Car (Автомобіль) і задати властивості цьому класу, такі як двері, колеса, подібно до тих, які є у справжніх автомобілів. Після цього, ґрунтуючись на цьому класі, можна створити інший клас, наприклад, Ford, який буде мати всі властивості класу Car плюс ті властивості, які є тільки у автомобілів марки Ford;

Таким чином, Android – операційна система для смартфонів, планшетних комп'ютерів, електронних книг, цифрових програвачів, «розумних» наручних годинників, ігрових приставок, нетбуків, окулярів Google, телевізорів, систем автоматичного керування автомобілем та інших пристроїв.

ОС заснована на ядрі Linux і власної реалізації віртуальної машини Java від Google. Спочатку розроблялася компанією AndroidInc., яку в 2005 році

купила Google. Згодом Google ініціювала створення альянсу OpenHandsetAlliance (OHA), який зараз займається підтримкою і подальшим розвитком платформи. Android дозволяє створювати Java-додатки, що керують пристроєм через розроблені Google бібліотеки. AndroidNativeDevelopmentKit дозволяє портувати (але не налагоджувати) бібліотеки та компоненти додатків, написані на Сі та інших мовах [14].

Значна гнучкість Android пов'язана з тим, що ця система побудована на ядрі Linux, що має відкритий програмний код, який дає необмежені можливості розробникам. Android може бути запущений на пристроях, що мають обсяг оперативної пам'яті менше 256 Мб. Найбільш нові версії системи вимагають 512 Мб оперативки, що також є невеликим значенням для сучасних апаратів. Система не вимагає наявності високопродуктивного процесора і може працювати на пристроях, оснащених ядром з частотою 600 МГц.

Операційна система дає можливість установки додатків з офіційного репозиторію Google, який надає найбільшу в світі базу програм. Це пов'язано з тим, що кожен розробник може самостійно написати будь-яку програму для апарату і розмістити її в магазині. Варто відзначити, що додатки на пристрої під управлінням Android можуть бути встановлені як безпосередньо з телефону або планшета, так і через комп'ютер шляхом завантаження файлу. apk і його подальшої установки на апараті [15].

Відмінною особливістю Android є його інтегрованість з сервісами Google-Gmail, Hangouts, Voice Search і т. п. на Android офіційно реалізована підтримка Chrome, що дозволяє синхронізувати відкриваються в браузері вкладки на смартфоні з комп'ютерним браузером.

Наприклад, ви можете почати перегляд сторінок з телефону і при бажанні продовжити вивчати інформацію, відкривши цю ж вкладку на комп'ютері, не вдаючись до допомоги повторного пошуку.

Всі потрібні додатки даної програми розміщуються одночасно на головному екрані і в меню апарату, яке викликається натисканням на центральну

сенсорну клавішу або відповідну кнопку на екрані. Всі налаштування розташовуються в секції «налаштування», а кожна дія користувача пояснюється коментарями і підказками при першому запуску апарату. Операційна система швидко реагує на натискання користувача і виробляє установку і скачування потрібних програм і файлів зі швидкістю, яка не програє іншим сучасним мобільним ОС [16].

Розробку додатків можна вести також і в середовищі Eclipse, використовуючи при цьому плагін – Android Development Tools (ADT) або в IntelliJ IDEA. Версія JDK при цьому повинна бути 5.0 або вище. С точки зору програміста, Android-платформа, що абстрагує розробника від ядра і дозволяє йому створювати код на Java. Android володіє кількома такими корисними можливостями:

1. По-перше, це фреймворк, що пропонує великий набір API для створення різних типів додатків і, крім того, забезпечує можливості повторного використання і заміни компонентів, які пропонуються платформою і сторонніми додатками;

2. По-друге, наявність віртуальної машини Dalvik, що відповідає за запуск додатків на Android. Для роботи над додатками є безліч бібліотек:

– Bionic (бібліотека стандартних функцій, несумісна з glibc);

– мультимедійні бібліотеки на базі PacketVideo OpenCORE (підтримують такі формати, як MPEG–4, H. 264, MP3, AAC, AMR, JPEG і PNG);

– SGL (движок двомірної графіки);

– OpenGL ES 1.0 ES 2.0 (движок тривимірної графіки);

– Surface Manager (забезпечує для додатків доступ до 2D / 3D);

– WebKit (готовий движок для веб-браузера; обробляє HTML, JavaScript);

– FreeType (движок обробки шрифтів);

– SQLite (легка СУБД, доступна для всіх додатків);

У порівнянні зі звичайними додатками Linux, додатки Android підпорядковуються додатковим правилам: Content Providers – забезпечує обмін даними між додатками; Resource Manager – дає доступ до таких ресурсів, як файли XML, PNG, JPEG; Notification Manager – надає доступ до рядка стану; Activity Manager – управління активними додатками.

Google пропонує для вільного скачування інструментарій для розробки (Software Development Kit), який призначений для x86-машин під операційними системами Linux, Mac OS X (10.4.8 або вище), Windows XP, Windows Vista і Windows 7. Для розробки потрібно JDK 5 або новіший [17].

Розробку додатків для Android можна вести мовою Java (не нижче Java 1.5). Існує плагін для Eclipse-Android Development Tools (ADT), призначений для Eclipse версій 3.3–3.7. Також існує плагін для IntelliJ IDEA, що полегшує розробку Android-додатків, і для середовища розробки NetBeans IDE, який, починаючи з версії NetBeans 7.0, перестав бути експериментальним, хоч поки і не є офіційним. Крім того, існує Motodev Studio for Android – комплексне середовище розробки на базі Eclipse, що дозволяє працювати безпосередньо з Google SDK.

У 2009 році на додаток до ADT був опублікований Android Native Development Kit (NDK) – пакет інструментаріїв і бібліотек, що дозволяє реалізувати частину програми на мові C / C++. NDK рекомендується використовувати для розробки ділянок коду, критичних до швидкості. У 2013 році Google представила нове середовище розробки Android Studio, заснована на IntelliJ IDEA від JetBrains. У 2013-му році відбувся реліз Embarcadero RAD Studio-XE5. Можливість розробки нативних додатків для платформи Android. Процес створення Android програми не вимагає додаткових пристроїв, крім, власне, Android пристрою (в принципі, можна обійтися і емулятором) [18].

За даними дослідницької компанії IDC, операційна система від Google продовжує лідирувати як на українському (77%), так і на світовому (78%) ринку

смартфонів. Причина – у великій кількості виробників – тут є і недешеві телефони, і потужні смартфони за кілька тисяч доларів.

Втім, значне охоплення породжує і основні недоліки розробки для Android. Якщо орієнтуватися на широкий спектр пристроїв, доведеться враховувати різну продуктивність, нескінченне кількість розмірів екрану і пам’ять. Як наслідок – зростають витрати на проектування кількох інтерфейсів і додаткове тестування. Вартість розробки в такому випадку збільшується пропорційно кількості підтримуваних пристроїв [19].

Зареєструватися в Google Play може будь-який розробник. Для цього потрібно заплатити $25 за допомогою банківської карти, завантажити скріншоти програми, її опис і код. У Google немає таких жорстких вимог до якості програми, як у Apple. Це знижує і ризики витрат на доопрацювання. Крім того, існують альтернативні магазини додатків, такі як Amazon Apps, Samsung Apps, Opera Apps, SlideMe. У кожного своя політика модерації, однак присутність в них дає відчутний приплив користувачів.

1.3 Постановка задачі

Основною задачею дипломної роботи є створення програмного забезпечення для організації поштових повідомлень.

Вхідними даними мають бути повідомлення від операторів поштового зв'язку.

Вихідними даними буде інформація про відправлення.

Для створення програмного забезпечення для організації поштових повідомлень необхідно:

– Розробити дизайн додатку.

– Розробити додаток, бекенд частину та базу даних.

Основними функціями програмного забезпечення є :

– Організація та структуризація поштових повідомлень.
– Синхронізація поштових повідомлень між пристроями.
– Можливість працювати з різних акаунтів на одному пристрою.
– Створення власних поштових повідомлень.

Вимоги до проектованого ПЗ:

– Простота використання.
– Зменшена кількість ризиків пов’язаних з «людським фактором».
– Синхронізація даних між пристроями.
– Можливість використання одного пристрою декількаома користувачів

2 РОЗРОБКА СТРУКТУРНИХ РІШЕНЬ

2.1 Обгрунтування вибору технологій та пакетів для розробки додатку

Для розробки мобільного додатку було вибрано мову JavaScript, фреймворк React Native та бібліотеку Redux, також використано фреймворк Expo. Для розробки бекенд частини додатку було вибрано платформу Firebase. Даний набір інструментів дозволяє розробнику швидко створювати мобільні додатки використовуючи такий самий підхід як і в розробці веб застосунків.
Перелічимо основні переваги JavaScript [20]:

- Швидкий для кінцевого користувача: сценарій JavaScript написаний для клієнтської сторони, для підтримки веб-сервера не потрібна підтримка. Він також не потребує компіляції на стороні клієнта, що дає йому певні переваги швидкості. Оскільки сценарій виконується на комп'ютері користувача, залежно від завдання, результати виконуються майже миттєво. Наприклад, можна перевірити будь-який користувацький ввід перед відправкою запиту на сервер. Це знижує навантаження на сервер.
- Простота: JavaScript відносно простий в освоєнні і реалізації. Він використовує модель DOM, яка забезпечує безліч встановлених функцій для різних об'єктів на сторінках, що робить його легким для розробки сценарію для вирішення користувача мети.
- Універсальність: JavaScript відмінно працює з іншими мовами і може використовуватися в самих різних додатках. В даний час існує безліч способів використання JavaScript через сервери Node. js. Якщо завантажити node. js за допомогою Express, то використовується така база даних документів, як mongodb [21].

React Native дозволяє створювати мобільні додатки, використовуючи при цьому тільки JavaScript з такою ж структурою, що і у React. Це дає можливість складати багатофункціональний мобільний UI із застосуванням декларативних компонентів. Програми, які створюються за допомогою React Native, не є мобільними веб-додатками, тому що React Native використовує ті ж компоненти, що і звичайні програми для iOS і Android. Замість того щоб використовувати мову Swift, Kotlin або Java, можна збирати ці компоненти за допомогою JavaScript і React.

Перевагами React Native є такі можливості

Платформна розробка. Основна мета розробників – надати клієнтам сервіси. Ніхто не хотів би, щоб його користувачі були обмежені тільки однією якоюсь платформою тільки тому, що розробник не може створювати додатки для інших платформ. Отже, і сам розробник не повинен обмежувати свої здібності тільки тому, що йому або їй комфортно працювати з конкретним інструментом розробки. Фреймворк React Native є портативним, тобто його єдина кодова база, написана в JavaScript, створить модулі як для Android, так і для iOS [22].

Розробка гібридних додатків дозволяє зменшити час розробки, який дозволяє відразу вести розробку під дві платформи. Також React Native надає можливість повторно використовувати код з веб додатками, які розроблені за допомогою React.

Якщо порівнювати React Native з фреймворком Ionic, який використовує для відображення PhoneGap/Cordova мостик, для доступу до нативних функціям додатку. Ionic сильно відстає від React Native, тому що використовує візуалізації веб-технології, та не застосовує нативні компоненти.

Така сама ситуація з відомим фреймворком PhoneGap, який використовує теж веб-технології для роботи, що суттєво негативно відображається на швидкодії.

React Native в свою чергу пропонує роботу з нативними бібліотеками. Також він не використовує веб-технології в якості прослойки, що позитивно відображається на роботі програми.

Також React Native пропонує більшу швидкість та простоту при розробці додатків порівняно з нативними мовами Java/Swift, та не сильно програє їм в скорості роботи.

Освоєння React. Освоївши React і JavaScript, можна відкрити для себе новий світ front-end розробки стосовно, наприклад, до веб-сайтів. Фреймворк React Native заснований на тих же компонентах, що і React, тому отримані тут навички не обмежуються тільки розробкою мобільних додатків.

Час збірки швидше, ніж в Android Studio. Якщо раніше витрачалось більше 2–3 хвилин на збірку, то тепер щоб протестувати потрібно менще часу з допомогою React Native. З такою функцією, як «гаряча перезавантаження» (Hot Reloading), розробка і тестування користувальницького інтерфейсу стає значно легшим. Завдяки цій функції додаток перезавантажується кожен раз, коли JS-файл зберігається [21].

JavaScript зручний для передачі даних по мережі. У React Native виклик API, рендеринг зображень по URL і інші процеси дуже прості. Більше не потрібно використовувати Retrofit, OkHttp, Picasso і т.д. набагато менше часу витрачається на налаштування. Коли дані надходять з API на платформі Android, то вони спочатку перетворюються в POJO-модель і лише потім використовуються в Елементах UI. А ось дані JSON, отримані в React Native, зручні для JavaScript і можуть безпосередньо використовуватися для попереднього перегляду UI. Це дозволяє полегшити веб-інтерфейс для GET або POST-запитів від REST API.

Розробка UI. У React Native в якості розмітки UI виступає JSX, серйозний конкурент XML-розмітки на Android. У React Native UI-елементи в основному повинні розроблятися з нуля, тоді як в нативній розробці для Android бібліотека

підтримки Google Design Support Library вже підключена. Це дає розробнику свободу в плані інтерактивного і адаптивного дизайну.

Недоліками React Native є наступне [23]:

– Неприязнь до мови. Багато людей не люблять JavaScript за те, що ця мова не схожа на традиційні мови, такі як Java, C++ та інші;

– Не так вже й багато сторонніх бібліотек. Спільнота React Native все ще перебуває в стадії становлення і підтримує сторонні бібліотеки, не такі популярні, як нативна бібліотека Android.

Redux – бібліотека управління станом для додатків, написаних на JavaScript. Redux – це менеджер станів. Найчастіше її використовують з React, але її можливості не обмежуються однією цією бібліотекою. Хоча в React є власний метод управління станами (почитати про нього можна в керівництві по React для початківців), він погано масштабується. Переміщення стану вгору по дереву працює для простих додатків, але в більш складних архітектурах зміна стану проводиться через властивості (props). Ще краще робити це через зовнішнє глобальне сховище [24].

Redux – це спосіб управління станом програми. Він заснований на кількох концепціях, вивчивши які, можна з легкістю вирішувати проблеми зі станом. Одна зі складнощів в розумінні роботи Redux-це безліч неочевидних термінів типу редюсерів, селекторів і санків. Для більш чіткого розуміння поглянемо на розширений Flux-цикл.

Redux ідеально підходить для середніх і великих додатків. Їм варто користуватися тільки у випадках, коли неможливо управляти станом програми за допомогою стандартного менеджера станів в React або будь-який інший бібліотеці. У Redux загальний стан програми представлено одним об’єктом JavaScript-state (стан) або state tree (дерево станів). Незмінюване дерево станів доступно тільки для читання, змінити нічого безпосередньо не можна. Зміни можливі тільки при відправці action (дії).

Основна перевага Redux – це управління як станом даних, так і станом інтерфейсу. Redux – єдине джерело істини при розробці, адже тут все дуже спрощується при пошуці актуальної інформації. Є безліч способів і підходів при роботі з Redux [25].

Серед недоліків Redux можна виділити наступне:

Повна залежність від порядку виклику. Наприклад, якщо поміняти місцями withUsers і withPresents, то HOC не зможе впоратися із завданням – адже withPresents не знайде списку користувачів, що може бути обов'язковим параметром. Крім того, HOC може сам змінювати дані. І коли ми зіткнемося з такою проблемою, нам потрібно буде спочатку зрозуміти, що у нас з цим є проблема, а в більшості випадків це може бути складно. Для її вирішення, нам потрібно або писати новий HOC (такий же як вихідний з деякими змінами), або правити вже існуючий, що може зламати логіку в справно працюючих місцях. Це основні підходи, які ми використовуємо в нашому додатку Warehouse Manager.

Actions можна розглядати як складні переходи між станами, але в основному вони лише задають одне значення. У додатках, зроблених на Redux, часто накопичується безліч таких простих actions, і це явно нагадує Java з написанням функцій сетерів вручну.

Один і той же фрагмент стану можна було б використовувати по всьому додатку, але в більшості випадків він відноситься до однієї певної частини інтерфейсу. Перенесення стану з компонентів в сховище Redux – це просто додаткове перенаправлення без належного рівня абстракції.

Функції – редюсери могли б впоратися з самими хитромудрими завданнями метапрограмування, але зазвичай зводяться до примітивної диспетчеризації action відповідно до його типу. Це не проблема для таких мов, як Elm або Erlang, які відрізняються лаконічним і виразним синтаксисом pattern matching, але в Javascript доводиться мати справу з громіздкими конструкціями switch [26].

Крім того, стан компонентів в React занадто неповороткий для роботи з наскрізною функціональністю, яка зачіпає багато модулів програми, як, наприклад, інформація про користувача або оповіщення. Якраз для цього в Redux є дерево станів, незалежне від призначеного для користувача інтерфейсу. До того ж, при обробці стану поза інтерфейсом легше підтримувати сталість, адже серіалізація в localStorage або URL проводиться в єдиному місці.

Firebase допомагає швидко створювати якісні додатки, збільшувати аудиторію залучених користувачів і підвищувати доходи. Платформа містить безліч корисних функцій для вашої програми, в тому числі серверний код для мобільних сервісів, статистику, а також інструменти для монетизації і розширення аудиторії. Пакет розробника Firebase об'єднує інтуїтивно зрозумілі API, позбавляючи вас від необхідності керувати окремими пакетами. Вибирайте тільки те, що вам потрібно, і користуйтеся перевагами інтегрованого рішення. Платформа, що використовує інфраструктуру Google, надає необхідні можливості для кожного етапу розробки і зростання [26].

Головними перевагами Firebase є:

– Швидкість роботи. У пакеті розробника Firebase зібрані інтуїтивно зрозумілі API, які спрощують і прискорюють розробку якісних додатків. Також у вашому розпорядженні всі необхідні інструменти для розширення користувальницької бази і підвищення доходів – вам залишається тільки вибрати відповідні для ваших цілей;

– Готова інфраструктура. Не потрібно створювати складну інфраструктуру або працювати з декількома панелями управління. Замість цього ви зможете зосередитися на потребах користувачів;

– Статистика. В основі Firebase лежить безкоштовний аналітичний інструмент, розроблений спеціально для мобільних пристроїв. Google Analytics для Firebase дозволяє отримувати дані про дії ваших користувачів і відразу ж вживати заходів за допомогою додаткових функцій [27];

– Кроссплатформеність. Firebase працює на будь-яких платформах завдяки пакетам розробника для Android, iOS, JavaScript і C++. Ви також можете звертатися до Firebase, використовуючи серверні бібліотеки або REST API;

– Масштабованість. Якщо додаток стане популярним і навантаження на нього зросте, вам не доведеться міняти код сервера або залучати додаткові ресурси – Firebase зробить це за вас. Крім того, більшість функцій Firebase безкоштовні і залишаться такими незалежно від масштабу ваших проектів. Платних функцій чотири. У них передбачений безкоштовний пробний період і два тарифних плани.

Безкоштовна підтримка по електронній пошті. Крім того, команда Firebase і фахівці з розробки Google дадуть відповідь на ваші запитання на ресурсах Stack Overflow і GitHub. Firebase має свої недоліки:

– Його область застосування набагато менше, ніж у NoSQL-рішення;

– Firebase сильно обмежує вас при вибірці даних і при необхідності записати дані в кілька місць одночасно;

– Далеко не з усіма структурами даних зручно працювати в Firebase.

2.2 Проектування архітектури програмної системи

Дослідження архітектури програмної системи намагається визначити як найкраще розбити систему на частини, як ці частини визначають та взаємодіють одна з одною, як між ними передається інформація, як ці частини розвиваються поодинці і як все вищеописане найкраще записати використовуючи формальну чи неформальну нотацію.

Архітектура повинна будуватись щоб найкраще відповідати вимогам до системи що створюється, згідно принципу «форма відповідає функції». Проектування архітектури програмної системи – це процес розробки, що виконується після етапу аналізу і формулювання вимог. Задача такого

проектування – перетворення вимог до системи у вимоги до програмної системи і побудова на їхній основі архітектури системи [28]. Побудова архітектури системи здійснюється шляхом визначення цілей системи, її вхідних і вихідних даних, декомпозиції системи на підсистеми, компоненти або модулі та розроблення її загальної структури. Проектування архітектури програмної системи може проводитися різними методами (стандартизованим, об'єктно-орієнтованим, компонентним), кожний з яких пропонує свій шлях побудови архітектури, а саме, визначення концептуальної, об'єктної й інших моделей за допомогою відповідних конструктивних елементів (блок-схем, графів, структурних діаграм). Проведемо коротку характеристику використаних програм для створення мобільного додатку.

Flux – архітектура додатків, розроблена Facebook для подолання обмежень архітектури MVC. Redux ідеально підходить для середніх і великих додатків. Ним варто користуватися тільки у тих випадках, коли неможливо управляти станом програми за допомогою стандартного менеджера станів в React або будь-який інший бібліотеці.

Архітектура Flux має такі компоненти:

– Дія: піднімається за допомогою представлення, коли користувач взаємодіє з елементами керування інтерфейсом у представленні;

– Відправник: зберігає контекст до сховища даних і передає дію з представлення до сховища. Відправник отримує дію з представлення та сповіщає сховище;

– Сховище: зареєстровано у відправнику. Сховище містить дані. Воно отримує подію оновлення від відправника і відповідає на нього. Відповідь буде іншою подією;

– Представлення: відреагує на подію зміни і внесе відповідні зміни.

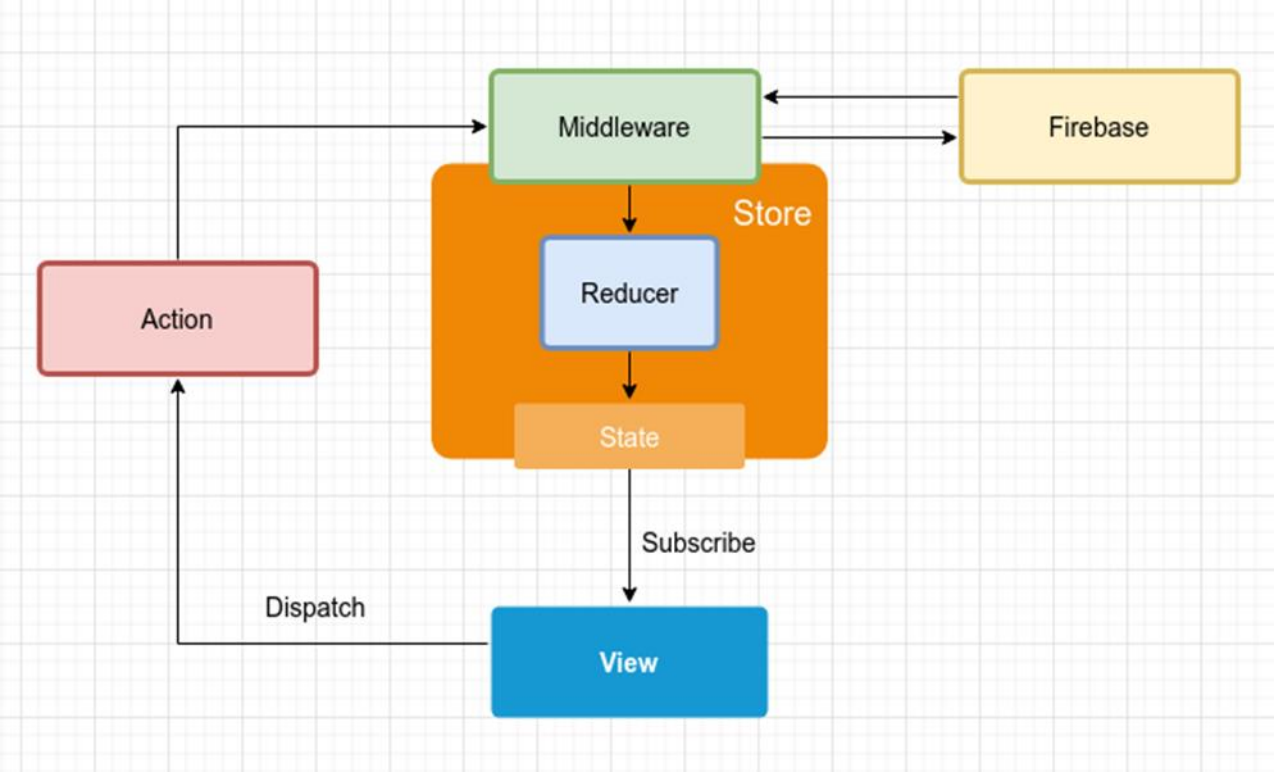

Рисунок 2.1 – Архітектура проекту реалізація Flux-архітектури в вигляді Redux

Redux є передбачуваним контейнером стану для додатків JavaScript. Тобто у Redux загальний стан програми представлено одним об’єктом JavaScript-state (стан) або state tree (дерево станів). Незмінюване дерево станів доступно тільки для читання, змінити нічого безпосередньо не можна. Зміни можливі тільки при відправці action (дії). Redux – це спрощена реалізація архітектури Flux у Facebook, яка є структурою Model-View-Controller. За допомогою редукторів зменшується складність. Redux редуктори є функціями без побічних ефектів, які обчислюють наступний стан програми. Redux заснований на трьох принципах:

– Стан програми зберігається в одному об’єкті. Redux зберігає стан в одному об’єкті JavaScript, щоб полегшити відображення та передачу даних у всій програмі. Централізація стану в одному об’єкті також робить процес тестування і налагодження швидшим;

– Стан програми є незмінним. У Redux, стани не можуть бути змінені. Єдиний спосіб змінити стан – це відправити дію. Дії є незмінними об’єктами JavaScript, які описують зміни стану [29];

– Редуктори вказують, як дія перетворює стан. Редуктори – це функції JavaScript, які створюють новий стан з даним поточним станом і дією. Вони централізуються мутації даних і можуть діяти на всю або тільки частину стану.

Редуктори також можуть бути об’єднані і повторно використані. Ця архітектура значно збільшує масштабованість для великих і складних програм. Вона також дозволяє використовувати дуже потужні інструменти для розробників, оскільки можна простежити кожну мутацію до дії, яка її викликала. Зі станом і дією наступний стан програми можна передбачити з абсолютною впевненістю.

У нашому випадку, використання архітектурного підходу Redux має такі переваги:

– Передбачувані оновлення стану полегшують розуміння того, як працює потік даних у додатку;

– Використання «чистих» редукторних функцій полегшує тестування логіки, а також дає можливість використовувати корисні функції, такі як «подорож у часі»;

– Централізація стану полегшує реалізацію подій, таких як внесення змін до даних або збереження даних між оновленнями сторінок.

На додаток до цих загальних переваг, Redux забезпечує деякі переваги для збереження стану у застосуванні React [30].

Отже, на основі вищезгаданих переваг, для розробки мобільного додатку для організації поштових повідомлень було обрано архітектурних підхід – Redux.

2.3 Розробка структури бази даних

База даних Firebase Realtime Database дозволяє створювати багаті спільні програми, забезпечуючи безпечний доступ до бази даних безпосередньо з клієнтського коду. Дані зберігаються локально, і навіть в автономному режимі

події реального часу продовжують працювати, даючи кінцевому користувачеві можливість реагувати на них. Коли пристрій відновлює з'єднання, база даних реального часу синхронізує локальні зміни даних з віддаленими оновленнями, які відбулися під час роботи клієнта в автономному режимі, автоматично об'єднуючи всі конфлікти [31].

База даних реального часу має гнучку мову правил на основі виразів, що називається Firebase Realtime Database Security Rules, щоб визначити, як дані повинні бути структуровані і коли дані можуть бути прочитані або записані. При інтеграції з перевіркою справжності Firebase розробники можуть визначити, хто має доступ до яких даних і як вони можуть отримати до них доступ.

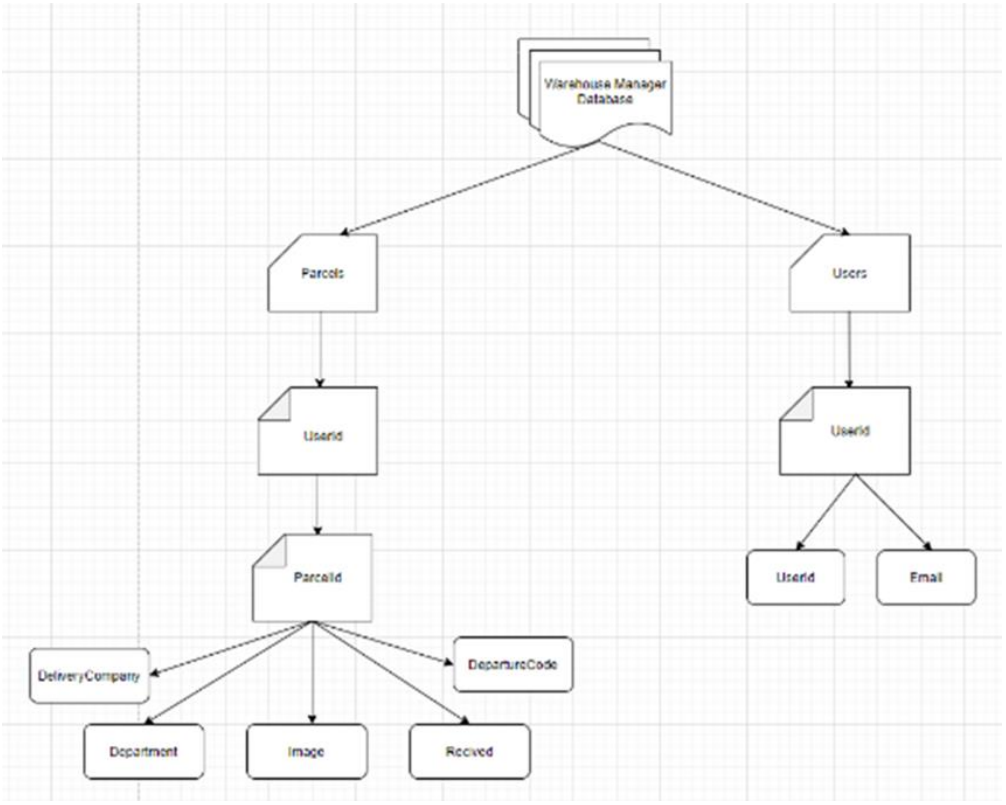

Рисунок 2.2 – База даних платформа Firebase Realtime Database

База даних реального часу є базою даних NoSQL і як така має різні оптимізації і функціональні можливості в порівнянні з реляційною базою даних. API бази даних реального часу призначений тільки для швидкого виконання операцій. Це дозволяє вам створити відмінний досвід роботи в реальному часі,

який може обслуговувати мільйони користувачів без шкоди для оперативності реагування. Тому важливо подумати про те, як користувачі повинні отримати доступ до ваших даних, а потім відповідним чином структурувати їх.

Термін «NoSQL», що розшифровується як «Not Only SQL» або «Not Relational», до кінця не визначений. Зазвичай такі системи задовольняють таким ознакам:

– Наявність засобів розподілу навантаження на безліч серверів;

– Можливість розподіл даних на безліч серверів;

– Простий протокол виклику операцій (у порівнянні з SQL);

– Більш слабка модель паралелізму, ніж acid-транзакції;

– Ефективне використання розподілених індексів і оперативної пам'яті для зберігання даних;

– Відсутність фіксованої схеми даних.

NoSQL – системи зазвичай не задовольняють властивостям acid-транзакцій: допускається узгодженість в кінцевому рахунку. Пропонується модель «BASE» (Basically Available, Soft state, Eventually consistent, узгодженість в кінцевому на противагу ACID. Ідея полягає в тому, що, відмовившись від обмежень ACID, можна домогтися набагато кращої продуктивності і масштабованості. Більшість систем різняться в ступені відмови від ACID [32].

В нашому дослідженні ми розглянули дів основні переваги використання бази даних NoSQL.

1. Ефективність розробки додатків. Більшість зусиль, пов'язаних з розробкою додатків, витрачаються на відображення даних зі структур, зберігаються в пам'яті, в реляційні бази даних. База даних NoSQL може забезпечити модель даних, краще задовольняє потреби програми, спростивши тим самим цю взаємодію і зменшивши кількість коду, який необхідно написати, налагодити і розвинути.

2. Великомасштабні дані. Дана програма дає можливість зберігати більше великих обсягів даних і швидше їх обробляти. Основна причина полягає в тому,

що реляційні бази даних призначені для роботи на одному комп'ютері, в той час як великі обсяги даних і програми для їх обробки економніше зберігати на кластерах, що складаються з численних невеликих і дешевих комп'ютерів. Багато баз даних NoSQL розроблені спеціально для кластерів, тому вони краще вписуються в сценарії обробки великих обсягів даних.

Створення програми за допомогою Redux часто починається з продумування структури даних стану програми (application state). З її допомогою описується, що відбувається в додатку в кожен момент часу. Стан (state) є у будь-якого фреймворку і архітектури. У додатках на базі Ember і Backbone стан зберігається в моделях (Models). У додатках на базі Angular стан найчастіше зберігається в фабриках (Factories) і сервісах (Services). У більшості Flux-додатків стан є сховищем (Stores).

Головна його відмінність в тому, що всі стани додатків зберігаються в єдиній деревоподібній структурі. Таким чином все, що необхідно знати про стан програми, міститься в одній структурі даних з асоціативних (map) і звичайних масивів і у цього рішення є чимало наслідків. Одним з найважливіших є те, що можна відокремити стан програми від його поведінки. Стан – це чисті дані. Вони не містять жодних методів або функцій, і вони не заховані всередину інших об'єктів. Все знаходиться в одному місці. Це може здатися обмеженням, але насправді це прояв більшої свободи, оскільки можна сконцентруватися на одних лише даних.

Аналізуючи структуру мобільного додатку можна наголосити, що він складається з двох колекцій Посилки (Parcels) та Користувачі (Users). У своєму складі містить і характеризується такими особливостями:

– Колекції Посилки та Користувачі складаються з Документів. Документи які мають ідентифікатор;

– Документів в базі даних може бути скільки завгодно;

– Документи користувача мають ідентифікатор користувача і не він не має повторюватись;

– Документи посилок складається з документу в якого ід такий самий як і в користувача якому належать посилки в середині якого є документи з id посилок;

– В документі посилки зберігаються повідомлення про стан посилки поле Received (тип Boolean);

– DepartureCode код відправлення (тип String);

– DeliveryCompany назва компанії яка займається відправленням (тип String);

– Department місце прибуття посилку вулиця, місто, відділення (тип String);

– Image поле зберігає url фотографії посилки (тип String);

– В документі Користувачі є два поля;

– UserId (тип Object ID);

– Email (тип String);

Дані зберігаються у форматі JSON і оновлюються в режимі реального часу через механізм pus / sub. Ключова відмінність від інших движків БД полягає в тому, що доступ до неї можна отримати безпосередньо на клієнті (веб-браузер, Android / iOS додаток), а також доступно REST API, щоб контролювати доступ до конфіденційних даних є можливість використовувати декларативні правила безпеки;

У Firebase Realtime database підтримується автономний режим, всі змінені дані, коли клієнт був в автономному режимі будуть автоматично синхронізовані з сервером [33].

2.4 Розробка use case діаграми

У програмній і системної інженерії use case діаграма являє собою список дій або кроків заходи, які зазвичай визначають взаємодію між роллю (відомої на мові уніфікованого моделювання як «актор») і системою для досягнення мети.

«Актор» може бути людської або інший зовнішньою системою. Даний процес діє як метод моделювання програмного забезпечення, що визначає функції, які повинні бути реалізовані, і вирішення будь-яких помилок, які можуть виникнути.

Характеристики поведінки розроблюваної системи фіксуються і документуються засобами моделі, яка відображає функції (варіанти використання – use cases) продукту, представляє оточення системи (безліч активних суб'єктів-actors) і визначає зв'язки між варіантами використання і активними суб'єктами (діаграми варіантів використання – use case diagrams). Найбільш важливою є комунікативна складова моделі, що дозволяє групам розробників, замовників і кінцевих користувачів, які обговорюють властивості системи, говорити на одній мові.

Основи моделі закладаються вже на початковій фазі процесу розробки, коли ідентифікуються основні активні суб'єкти і варіанти використання системи, а пізніше, на етапі планування, модель розвивається і поповнюється за рахунок уточнення існуючих і додавання нових елементів [34].

Існує три основних елементи процесу:

– «Актори» – це тип користувачів, які взаємодіють з системою;

– Система – функціональні вимоги, які визначають передбачуване поведінку елементів;

– Цілі – use case зазвичай ініціюються користувачем для виконання цілей, що описують дії і варіанти, які беруть участь в їх досягненні.

Характеристики методики:

– Організація функціональних вимог;

– Моделювання цілей взаємодії користувачів системи;

– Запис сценаріїв з тригерних подій в кінцеві цілі;

– Опис основного ходу дій і виняткового потоку подій;

– Дозвіл доступу до функцій якої іншої події;

Основні кроки при розробці діаграм:

– Визначити користувачів системи;

– Для кожної категорії створіть профіль користувача. Це включає в себе всі ролі, які мають відношення до системи;

– Визначте важливі цілі, пов’язані з кожною роллю для підтримки системи. Цінова пропозиція системи визначає значну роль;

– Створіть приклади використання для кожної цілі, пов’язаної з шаблоном, і підтримуйте однаковий рівень абстракції в усьому прецедент.

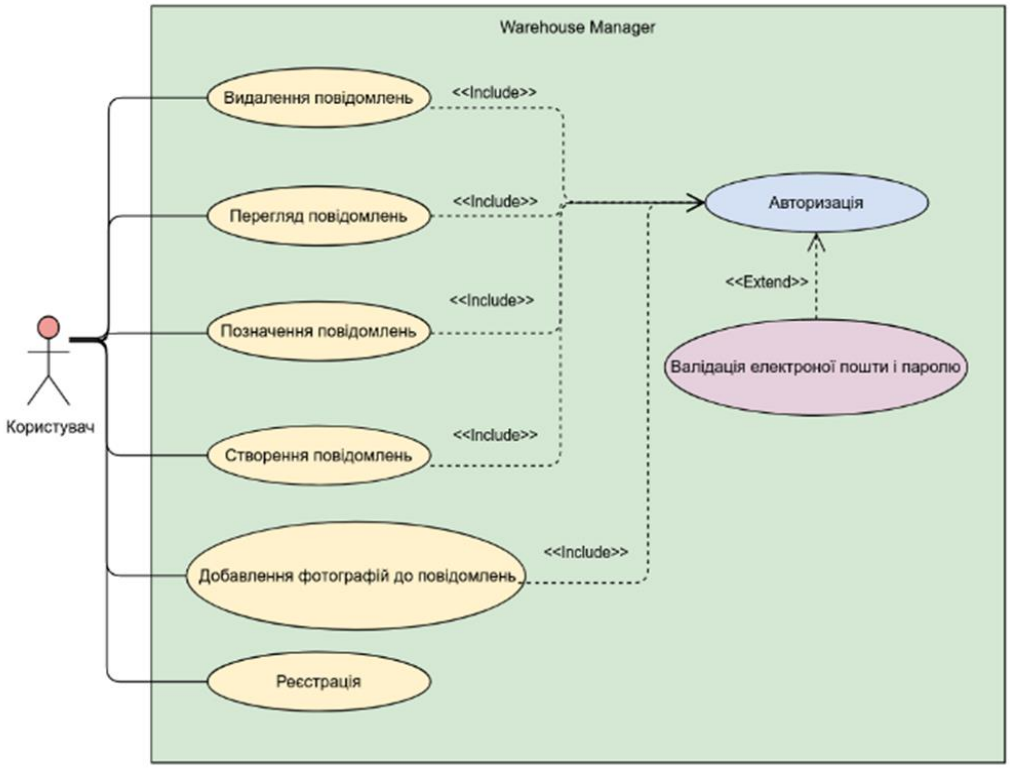

Рисунок 2.3 – Зовнішній вигляд use case діаграми

Use case діаграми використовуються для обліку функціональних вимог системи. Після визначення вищевказаних пунктів ми повинні використовувати такі рекомендації для побудови діаграми ефективного використання:

– Ім’я прецеденту дуже важливо – виберіть його таким чином, щоб воно могло ідентифікувати їх функції;

– Необхідно дати відповідне ім’я для акторів;

– Показати на діаграмі відносини і залежності;

– Не потрібно включати всі типи відносин, оскільки основна мета діаграми полягає у визначенні вимог;

– При необхідності можна використовувати пояснення, щоб прояснити деякі важливі моменти.

Таким чином, use case діаграми використовуються для збору вимог до системи, включаючи внутрішні і зовнішні впливи (як правило, це вимоги до дизайну). Отже, коли система аналізується для збору її функціональних можливостей, розробляються приклади використання і ідентифікуються учасники.

Приклад use case діаграми – графічне зображення взаємодій між елементами системи. Це методологія, яка використовується в системному аналізі для виявлення, уточнення і організації системних вимог. У цьому контексті термін «система» відноситься до того, що розробляється або експлуатується, наприклад до веб-сайту з продажу та обслуговування товарів поштою. Кроки використання більш високого рівня розглядаються як цілі для більш низького рівня. Використання діаграми варіантів use case diagram в середовищі продажів включає в себе упорядкування товарів, оновлення каталогу, обробку платежів і відносини з клієнтами. Діаграма використання виглядає як блок-схема. Інтуїтивні символи представляють собою елементи системи [35].

Діаграма може підсумувати відомості про користувачів вашої системи (також відомих як суб’єкти) та їх взаємодію з системою. Щоб побудувати один об’єкт, буде використовуватися набір спеціалізованих символів і конекторів.

Для того щоб користувач зробив будь яку дію крім реєстрації йому обов’язково потрібно авторизуватись. Авторизація також включає в себе валідацію даних.

Use case діаграма не має великого значення при відсутності чіткого розуміння процесу – вона не буде моделювати порядок виконання кроків, якщо не закладено чіткий алгоритм. Експерти рекомендують використовувати дані діаграми для доповнення текстового варіанта. У діаграмі на високому рівні

демонструється взаємозв’язок між варіантами використання, суб’єктами і системами.

2.5 Розробка мокапів

Для початку, нами цього були розроблені мокапи, що представляють собою фактичний дизайн-макет мобільного додатку для поштових повідомлень. Для розроблення мобільного додатку здійснено такий набір послідовних кроків: додаток написаний мовою JavaScript, оскільки ця мова використовується для крос-платформних додатків, заповнено бриф – опитувальну анкету, що дає виконавцю змогу зрозуміти вихідні дані, такі як маркетинговий складник, графічні та технічні аспекти, узгоджено технічне завдання та дизайн з вимогами[36].

Рисунок 2.4 – Стартовий екран мобільного додатку

Весь макет мобільного додатку будується з почергових елементів, що додаються, кожен з яких має свій набір налаштувань. Варто відзначити, що Mockups – це ретельно продумана та досить зручна у використанні програма. У

сфері свого застосування вона стає все більш популярною, а на просторах мережі можна знайти вже готові макети і використовувати їх як шаблони. Опис головного в мобільному додатку представлено на рис. 2.5.

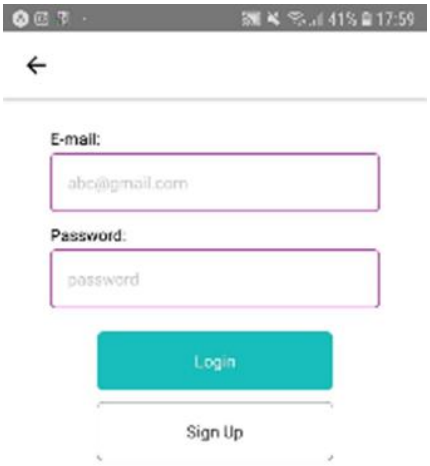

Рисунок 2.5 – Екран авторизації користувача

Трохи докладніше опишемо, як працює розроблений мобільний додаток. Спершу клієнту відкривається головна сторінка: тут є поле реєстрації або ж входу (рис. 2.6). Далі клієнт потрапляє в свій особистий кабінет або «Мій профіль» (рис. 2.7). Після цього відкривається наступна сторінка, за допомогою якої клієнт може створити своє повідомлення чи побачити повідомлення, які зберігаються у особистому кабінеті.

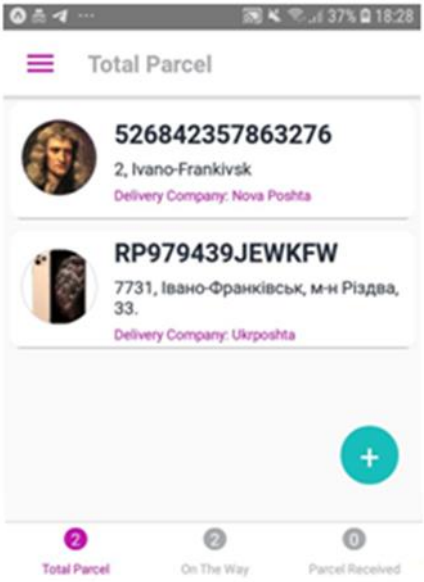

Рисунок 2.6 – Поле екрану із списком повідомлень

Повідомлення складається з фотографії якщо її добавити в ручну нажавши на круглу ділянку де фото, код посилки, місце доставки і компанії яка доставляє.

Надалі клієнт переходить до сторінки повідомлень, де може подивитись супровідну інформацію для прийняття рішення.

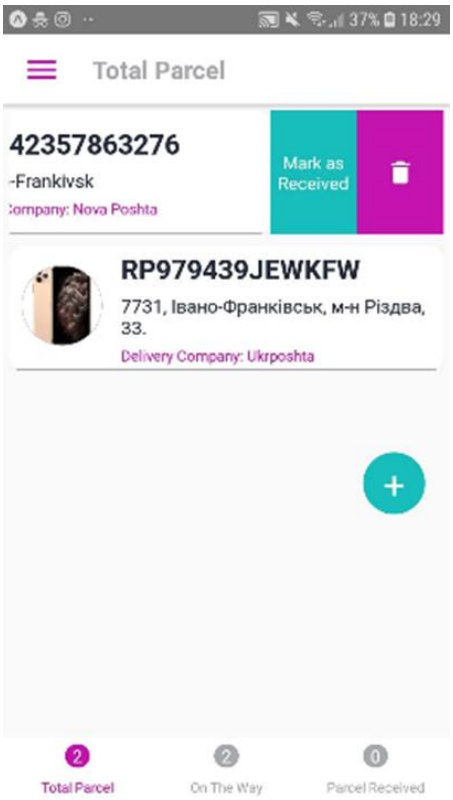

Рисунок 2.7 – Вигляд повідомлення про посилку клієнту від Укрпошти

Якщо потягнути повідомлення появиться дві функції: Позначити як отриману посилку або видалити її. Якщо посилка вже відмічена як отримана появиться з права знак ✓ та якщо потягнути знову в право появиться надпис відмітити як не отримана тоді знак ✓ пропаде.

Знизу є перемикачі які показують який зараз активний список посилок. Можна показувати всі посилки, тільки посилки в дорозі і посилки отримані. При натиску на перемикач з верху показується який активний перемикач.

На сторінці створення повідомлення (рис. 2.9.) розміщена форма для створення свого повідомлення, яке буде добавлено в список повідомлень. У клієнта є можливість відмінити вже обраний товар і розпочати свої дії спочатку.

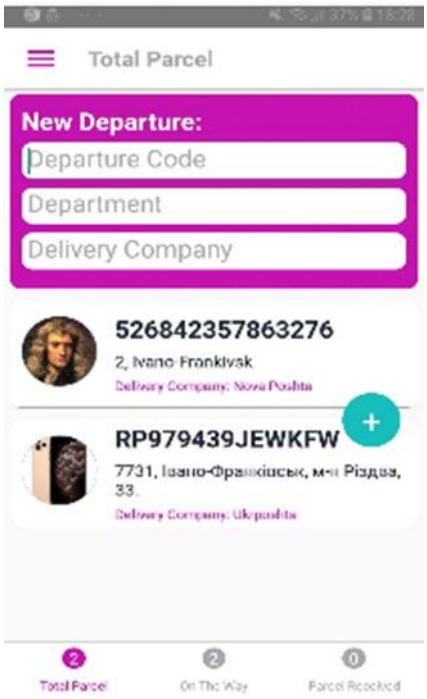

Рисунок 2.8 – Вигляд сторінки мобільного додатку з формою добавлення повідомлення

При натиску кнопки + появляється форма де користувач може добавити свою посилку.

Якщо нічого не вписано в поле Departure Code при повторному натиску на + форма пропаде.

Якщо інформація буде в полі Departure Code знак + поміняється на ✓. При кліку на дану кнопку посилка добавиться в список а форма пропаде.

Для зручності користувач може добавити фотографію, це може бути фото товару який отримує чи відправляє або фото людину якій прямує чи від кого прямує товар. Якщо провести з ліва на право по екрану або натиснути на іконку меню з гори появиться даний сайд бар. Сайд бар – це закріплена бічна панель ресурсу, область навігації або допоміжної інформації, графічно відокремлена від основної області контенту. sidebar може бути розміщений ліворуч або праворуч від контенту.

По ширині бічна панель набагато вже основний області перегляду. Якщо потрібно максимально заощадити місце, можна звузити сайд бари так, щоб в них можна було розташовувати тільки невеликі логотипи або абревіатури. На мобільному додатку може бути використано від одного до чотирьох сайд барів. Зустрічається використання двох бічних блоків, коли важливо відобразити багато інформаційних і навігаційних блоків.

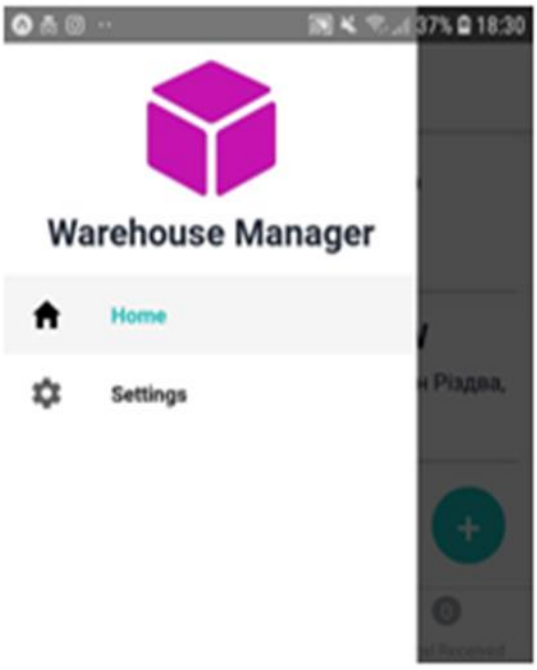

Рисунок 2.9 – Вигляд сайд бару у мобільному додатку

Дані, поміщені в бічну панель, відображаються на кожній сторінці проекту, а це позитивно позначається на взаємодії з користувачем.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДОДАТКУ WAREHOUSE MANAGER

3.1 Реалізація мобільного інтерфейсу

Для реалізації мобільного інтерфейсу було вибрано фреймворк React Native. Даний фреймворк, для побудови інтерфейсу використовує JSX, це розширення синтаксису для JavaScript. JSX зручний тим, що дозволяє легко та ефективно пов'язувати дані, які динамічно змінюються та інтерфейс. React дозволяє розділяти відповідальність між вільно зв’язаними одиницями, що дозволяє поєднувати логіку та розмітку. Таке поєднання називається “компонентами”.

Рисунок 3.1 – Приклад синтакисиу JSX

Додаток Warehouse Manager складається з головного файлу App.js, де підключаються екрани додатку, платформа Firebase та Redux.

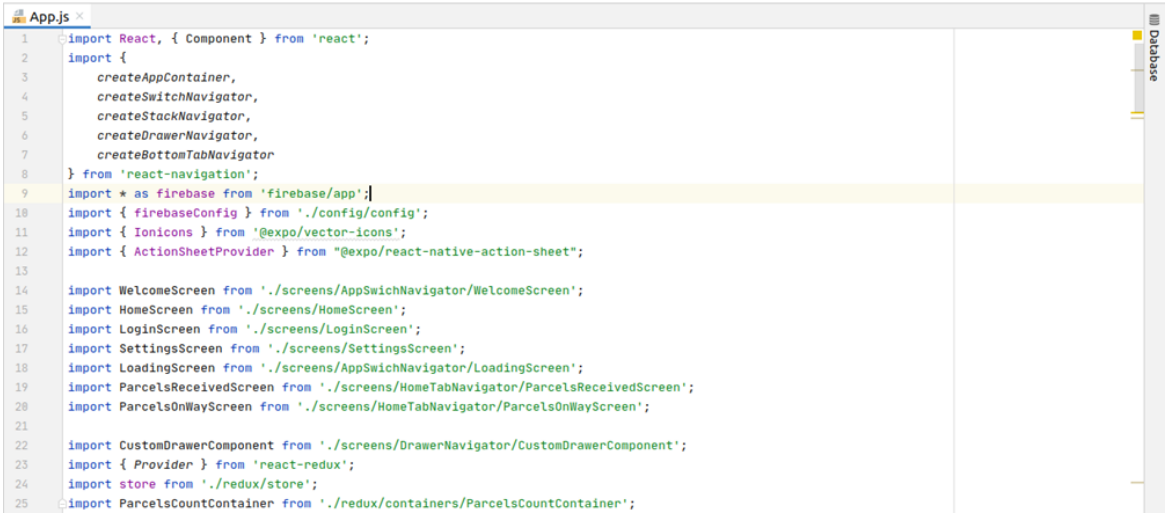

Рисунок 3.2 – Код файлу App.js

Даний підхід дозволяє легко маштабувати додаток та при потребі, добавляти новий функціонал.

При запусту програми відбувається підключення до платформи Firebase та до центрального сховища.

Рисунок 3.3 – Код підключення до сховища та платформи Firebase

Підключення до центрального сховища відбувається за допомогою компонентів “Провайдерів”. Це компоненти, які дозволяють підписуватися на зміну контексту. В даному випадку контекстом виступає платформа Firebase та центральне сховище.

Якщо дані змінюються в центральному сховищі, тоді не потрібно буде їх передавати в кожний компонент вручну, що запобігає виникнення проблеми, де потрібно з батьківських компонентів відправляти властивість в дочірні компоненти.

Ініціалізація підключення до платформи відбувається, за допомогою спеціального файлу конфігурацій.

Рисунок 3.4 – Код конфігурації платформи Firebase

В файлі конфігурації платформи вказані основні налаштування, а саме:

– Ключ який необхідний для доступу до платформи;
– Адреса по, якій відбувається автентифікація користувача;
– Адреса бази даних;
– Ідентифікатор проекту Firebase;
– Адреса місця збереження даних;

Після підключення відбувається перевірка, чи користувач вже залогінений в системі. Якщо користувач вже залогінений, йому не потрібно знову авторизуватися в системі, тоді користувач бачить відразу екран інформації про відправлення. Якщо користувач вже зареєстрований в системі, тоді появиться повідомлення, що даний емайл вже використовується.

Рисунок 3.5 – Код перевірки авторизації користувача

Якщо користувач, ще не авторизований, буде відкрито екран авторизації користувача, де він буде мати змогу залогінитися чи зареєструватися.

Рисунок 3.6 – Код реєстрації користувача

Після входу в систему відривається екран домашньої сторінки. Першим ділом відбувається метод життєвого циклу, який робить снапшот смс-повідомлень, від компаній Нової Пошти та Укр Пошти та перетворює його в масив. Даний масив порівнюється з даними на сервері, якщо є нові повідомлення,

вони відправляються на сервер та появляються в списку повідомлень. Також відбувається загрузка нових повідомлень з сервера.

Рисунок 3.7 – Код методу життєвого циклу

Коли загрузка повідомлень з сервера відбулась, повідомлення добавляються в центральне сховище (Redux).

Рисунок 3.8 – Код добавлення повідомлення в центральне сховище

Після чого повідомлення будуть відображенні в списку відправлень.

Список відправлень можна фільтрувати, за допомогою навігації, яка розташована в низу екрана. Навігація показує, який зараз активний фільтр та кількість повідомлень, які відповідають певному фільтру. Фільтрація відбувається за допомогою перевірки стану відправлення, якщо відправлення,

ще не отримано, тоді воно буде відображено в списку всіх відправлень та списку не отриманих відправлень. Також, якщо відправлення отримано, тоді воно буде відображено в списку отриманих відправлень.

Рисунок 3.9 – Код екрану не отриманих відправлень

Рисунок 3.10 – Код екрану отриманих відправлень

Екрани отриманих повідомлень та не отриманих повідомлень, майже аналогічні між собою. Головною відмінністю їх є тип даних, які вони отримають від центрально сховища. Дані екрани мають дочірний компонент “ActivityIndicator”, який відповідає за анімацію загрузки фотографії з сервера.

Основний функціонал додатку зосереджений на домашньому екрані.

Якщо користувачу потрібно, добавити відправлення вручну, тоді буде потрібно ввести інформацію про відправлення в форму (Рис.2.9), а саме: код відправлення, місце отримання відправлення та назву компанії, яка займається відправленням.

Рисунок 3.11 – Код функції добавлення відправлення

Дана функція приймає інформацію користувача та додає відправлення в список. Після добавлення відправлення в список, інформація відправляється на сервер. Якщо відправлення вже є в списку, тоді появиться сповіщення, що відправлення вже є в списку.

Після отримання відправлення користувач може його відмітити, як отримано (Рис.2.8).

Рисунок 3.12 – Код функції позначення як прочитано

Після відмічення, як отримано відправлення переміщається на екран отриманих відправлень, після чого відбувається запит на сервер з обновленою інформацією. Також можна зняти позначку отримано, тоді дії будуть відбуватися аналогічним чином.

Для зручності знаходження потрібного відправлення, було зроблено можливість прикріпляти зображення до відправлення.

Для добавлення фотографії з галереї, до відправлення потрібно натиснути на круглу область біля інформації про відправлення. Після вибрати пункт добавити фото з галереї.

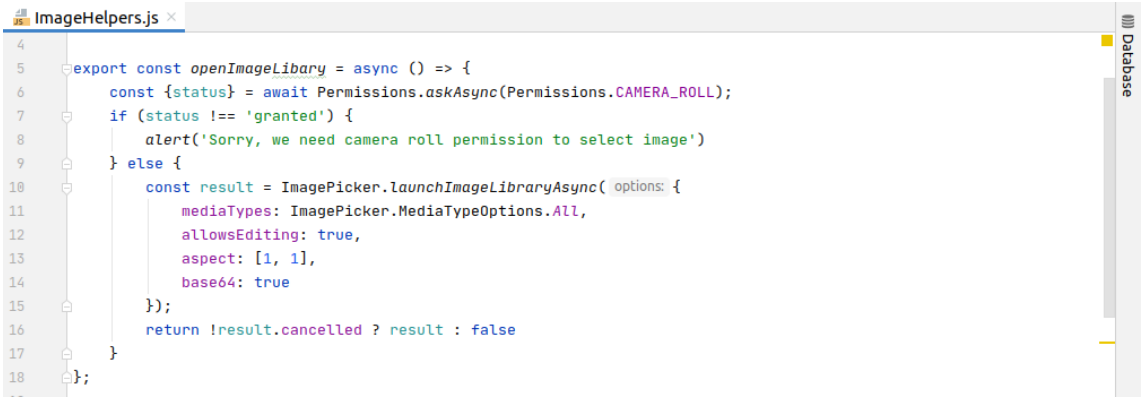

Рисунок 3.13 – Код функції відкриття галереї

Після чого буде запропоновано користувачу надати доступ програми до галереї. Якщо користувач надав доступ, буде відкрита галерея. Після вибору потрібного зображення, буде можливість обрізати його.

Рисунок 3.14 – Код функції відкриття камери

Також, якщо потрібно є можливість добавлення фотографії зробленою камерою. Процес відбувається аналогічним чином.

Рисунок 3.15 – Код функції відкриття камери

Коли користувач вибрав зображення, воно буде відправлено в сховище на сервері.

Розроблена програма Warehouse Manager є мобільним додатком, вона не є громіздкою та ресурсо-затратною. Як вже було сказано вище, в даний момент найбільш популярною операційною системою для смартфонів в Україні є Android тому вона і була нами вибрана для в якості базової. Програма Warehouse Manager призначена для організації повідомлень від Нової Пошти, Укрпошти та інших поштових операторів, які беруться з СМС повідомлень користувача.

Створена за допомогою технологій JavaScript, React Native, Redux, Expo, Firebase.

Додаток актуальний для персонального використовування, малого бізнесу та дропшипінгу для відслідковування поштових повідомлень.

Основна перевага мобільного додатку Warehouse Manager полягає в тому, що він проводить опрацювання поштових повідомлень від служб Нова Пошта та Укрпошта. Розроблене програмне забезпечення Warehouse Manager матиме можливість працювати на будь яких пристроях, які управляються Android та мають можливість виходу в інтернет за допомогою будь якого модуля (WI-FI, GPRS, EGE). Також пристрій, на якому буде встановлено цей додаток повинен мати сенсорний дисплей із розширенням не менше 800x480px.

Таким чином, додаток Warehouse Manager складе пряму конкуренцію для FlapApp, маючи при собі ряд переваг. Крім того, важливою перевагою даного продукту є простота дизайну і швидкодії, в порівнянні з існуючими аналогами (Нова Пошта та Укрпоша). Запити клієнта до сервера, і навпаки, займають в 2 рази менше часу.

3.2 Реалізація керування станом програми

Робота додатку Warehouse Manager повинна бути організована таким чином, щоб забезпечити найбільш можливу швидкодію. Додаток використовуватиме GSM, GPRS, та WI-FI модулі, тому потрібно максимально зменшити час їх використання, для збільшення часу роботи пристрою, оскільки дані модулі потребують досить багато затрат електроенергії.

Після регістрації в мобільному додатку дані електронної пошти і пароль зберігаються в Firebase. Якщо такий вже користувач існує, то появляється вікно з надписом про те, що такий користувач вже є і він може починати працювати в додатку як ідентифікований користувач [38].

Екран авторизації мобільного додатку Warehouse Manager дозволяє можливість роботи у випадку, коли користувач використовує додаток з двох пристроїв або додаток використовують кілька людей на одному пристроїв, чи кілька людей використовують один аккаунт з повідомленнями і хочуть мати їх на всіх пристроях. При реєстрації у Warehouse Manager користувач вказує номер телефону та адресу електронної пошти. При авторизації в додатку вже зареєстрованого користувача процедура аналогічна.

Після авторизації появляється даний екран зі списком повідомлень. Повідомлення складається з фотографії якщо її добавити в ручну нажавши на круглу ділянку де фото, код посилки, місце доставки і компанії яка доставляє. Якщо користувач зареєструвався, то при вході у додаток на мобільному пристрої відбувається підтягування СМС повідомлень від Нової Пошти та Укрпошти (тобто відбувається уніфікація повідомлень від двох служб, а користувачу не потрібно встановлювати окремі додатки цих служб, що вже існують в кожної з поштових служб).

Паролі та коди авторизації не зберігаються в системі у відкритому вигляді. У базі даних зберігаються результати обчислення криптографічної хеш-функції bcrypt від пароля або коду. bcrypt-адаптивна криптографічна функція формування ключа, яка використовується для захищеного зберігання паролів [39]. Для захисту від атак за допомогою таблиць bcrypt використовує довільний рядок даних, що додається до рядка пароля.

Якщо користувач увійшов, то програма перевірить чи немає нових повідомлень від Нової Пошти та Укрпошти. Якщо повідомлення присутні для користувача, то додаток додасть їх у список. Список повідомлень знаходиться в базі даних Firebase.

3.3 Створення інсталяційного файлу

Після того як написаний весь програмний код, потрібно розгорнути додаток. Для розгортання додатку було вибрано сайт https://expo.io .

Одна з переваг фреймворку Expo, це швидкість та простота розгортання додатку. Щоб розгорнути додаток на сервері Expo, потрібно зареєструватися на сайті. Після чого потрібно скористуватися командою “expo build:android”. Далі буде відбуватися компіляція додатку, якщо компіляція пройде без помилок, тоді відбудиться розгортання додатку на сервері.

Спостерігати за процесом розгортання дадатку, можна в персональному кабінеті сайту Expo.

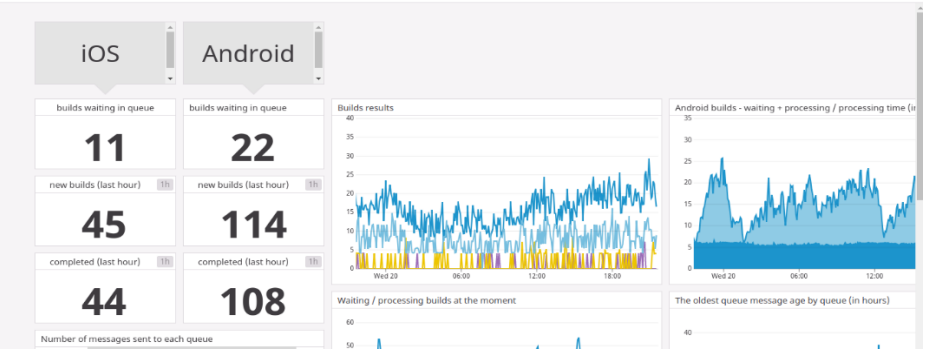

Рисунок 3.16 – Інформація про розгортання додатку

Коли розгортання додатку закінчиться, сам додаток появиться в вкладці Builds, та буде доступний для скачування за наступною ссилкою: https://exp-shell-app-assets.s3.us-west-1.amazonaws.com/android/%40patrokl/warehouse-manager-564af20204a64d18be9a774aa98a7565-signed.apk

3.4 Інструкція по використанню ПЗ

Опишемо порядок використання розробленого мобільного додатку:

1. Скачати додаток та встановити на смартфон.
2. Ввести електронну пошту та придумати пароль, який буде не менше шести символів та зареєструватися в системі.
3. Щоб відмітити відправлення, як отримано потрібно провести, по повідомленню справа наліво, де появляться спеціальні опції, та вибрати Mark as Received.
4. Щоб видалити відправлення з списку, потрібно виконати дії, аналогічні 3 пункту, та вибрати опцію з значком корзини.
5. Для переходу до повідомлень, які ще в дорозі потрібно вибрати фільтр On The Way.
6. Для переходу до отриманих повідомлень потрібно вибрати фільтр Parcel Received.
7. Для добавлення свого повідомлення потрібно: натиснути на кнопку з знаком +, після чого ввести код відправлення, місце відправлення та назву компанії, яка займається відправленням і натиснути кнопку зі значком ✓.
8. Добавлення зображення з галереї потрібно: натиснути на сіру область біля інформації про відправлення та вибрати пункт Select From Photos. Після чого вибрати необхідне зображення та обрізати його.
9. Добавлення зображення з камери телефону відбувається аналогічним чином пункту 8, тільки потрібно вибрати опцію Camera.
10. Для виходу з облікового запису необхідно: натиснути на іконку меню в правому верхньому куті, вибрати настройки та натиснути кнопку Log Out.

4 БІЗНЕС-ПЛАН ВПРОВАДЖЕННЯ МОБІЛЬНОГО ДОДАТКУ WAREHOUSE MANAGER В РОБОТУ СЛУЖБ НОВА ПОШТА ТА УКРПОШТА

4.1 Аналіз українського ринку мобільних додатків для поштових повідомлень

За даними аналітичного бюро Statista, протягом 2017 року було зафіксовано 197 більйонів завантажень додатків на всіх мобільних платформах. Ми вважаємо, що мобільні додатки мають велику кількість переваг для підприємців у порівнянні з веб-сайтами та програмним забезпеченням для персональних комп'ютерів.

Успішний досвід українських компаній, таких як Нова Пошта, Rozetka та Prom. ua свідчить, що мобільні додатки здатні значно збільшити обсяг продажів. Відповідно до даних статистичного бюро Criteo [43], 27% відсотків від усіх платежів у сфері електронної комерції за 2018 рік було проведено за допомогою мобільних додатків. За результатами досліджень аналітичного агентства eMarketer [44], протягом останніх 5 років спостерігається тенденція до збільшення обсягу часу, проведеного протягом дня у додатках, причому обсяг витраченого часу на перегляд мобільних веб-сторінок через браузер практично не змінився.

Аналізуючи особливості ринку мобільних операційних систем, слід зазначити призупинення підтримки ряду з них (Blackberry OS, Windows Phone) та розробки нативних мобільних додатків. Одним з ефективних шляхів вирішення проблемних питань є використання концепції гібридних мобільних додатків та урахування у процесі розробки програмного забезпечення особливостей кросплатформності, яка передбачає можливість автора створювати додатки для кількох платформ одночасно.

Аналізуючи український ринок мобільних додатків для поштових повідомлень можна назвати такі мобільні додатки, що були набільше поширені в 2019 р.

Pushline – це безкоштовна програма, яка синхронізує поштові оповіщення та значки між смартфоном та комп'ютером або комп'ютером Mac, як Continuity для вашого пристрою Android. Для даної програми не потрібна реєстрація або реєстрація, відбувається постійне віддзеркалення push-повідомлень телефону на ПК / Mac. Віддалене управління телефоном прямо з комп'ютера.

AirDroid. Дозволяє безкоштовно отримувати доступ до свого телефону або планшета Android з Windows, Mac або через Інтернет і управляти ними без проводів. Крім того, дозволяє синхронізувати поштові повідомлення між смартфонами та ПК.

Wondershare MobileGo. MobileGo дозволяє вам керувати всіма мобільними додатками в одному місці і синхронізувати всі поштові повідомлення з різних мобільних номерів.

Слід також наголосити, що вартість розробки додатків залежить від великої кількості факторів, серед яких слід виокремити: складність виконання, розцінки розробника, робота з нативними функціями. Для порівняння ціни додатку найбільш популярних операційних систем (табл. 1) було використано онлайн-калькулятор Venturepact [45] з визначеним функціоналом:

– Шаблонний для операційної системи інтерфейс користувача;
– 7–12 робочих екранів додатку;
– Двоетапна автентифікація користувачів;
– Базові заходи захисту;
– Використання системи електронних платежів;
– Створення бази даних для додатку;
– Управління повідомленнями.

Таблиця 4.1 – Порівняння вартості розробки мобільних додатків для різних операційних систем, дол. США

Регіон	Платформи			
	Android	iOS	Android та iOS одночасно	Гібридний додаток
США, Канада, Західна Європа, Австралія	3900	3900	7800	2600
Східна Європа, Середній Схід, Центральна та Південна Америка	1950	1950	3900	1300
Південна Азія, Східна Азія, Південно-Східна Азія, Африка	1200	1200	2400	800

Узагальнюючи показники, доцільно зазначити, що гібридна кросплатформна розробка потребує, у середньому, на 66,7% менше витрат, ніж одночасна нативна розробка окремих додатків для Android та iOS.

Відповідно до опитування, проведеного Ionic, у 2015 році 20% програмістів – прихильники нативних додатків, тоді як у 2017 році їх кількість зменшилася до 2,9% [45].

Структуру ринкових часток мобільних операційних систем наведено на рис. 4.1.

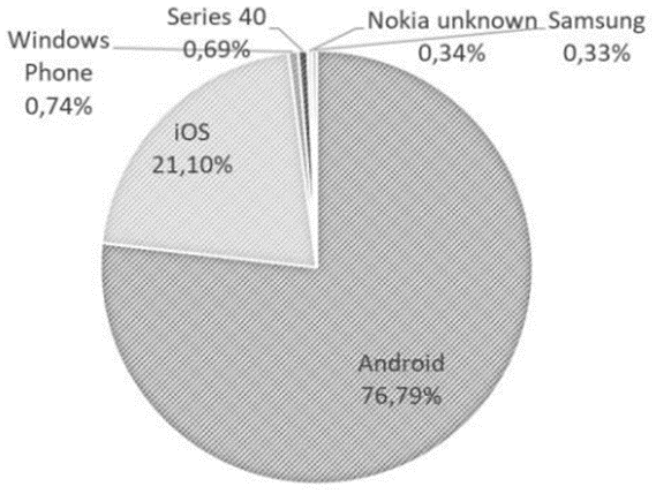

Рисунок 4.1 – Структура ринку мобільних операційних систем України станом на 16.02.2019

Формування попиту на мобільні додатки на регіональних інформаційних ринках у першу чергу пов’язано з їх використанням у компаніях, які успішно

вирішують завдання інформаційної підтримки. У бізнес-сегменті мобільні додатки пропонують великі корпорації та торговельні мережі. Їхні сервіси, як правило, безкоштовні для клієнтів та включають різноманітні інформаційні та операційні можливості.

На початку мобільний додаток Warehouse manager був збитковим, але зараз проект став прибутковим. Компанії Нова Пошта та Укрпошта зберегли більше коштів, ніж витрачали на підтримку особистого мобільного додатку для кожної фірми.

На кожному push-повідомленні (а їх понад 2 мільйони щомісяця) компанія заощаджує від 13 до 22 копійок, на одному оформленні посилки в додатку – 2 гривні і 40 секунд роботи працівника, на кожному виклику кур'єра – 3 хвилини часу оператора контакт-центру, а на оплаті одного відправлення – 55 секунд обслуговування у відділенні [46].

Перспективним є також використання чат-ботів. Щомісяця тестовим ботом у Viber вже користуються близько 1500 клієнтів. Функціонал поки базовий: трекінг посилки, пошук відділення та виклик кур'єра. Доцільним є розвиток цього напрямку, використовуючи підхід machine learning. Йдеться про навчання через нейронні мережі, яке дозволятиме роботам розширювати можливості, сприймати більше семантичних відтінків, відмічати для себе помилки людей. Сьогодні боти переважно використовують тригери, тобто дають стандартні відповіді за ключовими словами. З часом вони матимуть відповіді на найрізноманітніші запитання від користувачів. Теперішні технології machine learning адекватно працюють поки що лише з англійською мовою, та надалі це буде змінюватися.

Формування попиту на мобільні додатки на регіональних інформаційних ринках у першу чергу пов'язано з їх використанням у компаніях, які успішно вирішують завдання інформаційної підтримки. У бізнес-сегменті мобільні додатки пропонують великі корпорації та торговельні мережі. Їхні сервіси, як

правило, безкоштовні для клієнтів та включають різноманітні інформаційні та операційні можливості.

4.2 Розрахунок економічної ефективності використання розробленого програмного забезпечення

Капітальні витрати «К» на створення програмного забезпечення (ПЗ) вираховується за формулою.

$$К_З = З_1 + З_2 + З_3 \quad (4.1)$$

Де $З_1$ – затрати праці програмістів-розробників;

$З_2$ – затрати комп'ютерного часу, грн.;

$З_3$ – непрямі (накладні) затрати, грн.

$$К_З = 20389{,}40 + 741{,}5 + 2352{,}58 = 23483.48$$

Затрати праці програмістів-розробників «$З_1$» вираховуються за формулою:

$$З_1 = \sum_{k=1}^{K} N_r \cdot r_k \cdot T_k \cdot K_{зар} \quad (4.2)$$

де N_k – кількість розробників k-й професії, чол;

r_k – годинна розробника k-й професії, грн.;

T_k – складність розробки для k-го розробника (кількість витраченого розробником часу);

$К_{зар}$ – коефіцієнт нарахувань на фонд заробітної плати, (1,2 – 1,45).

$З_1 = 1 * 17,28 * 427 * 1,3 = 9.436$

Годинна зарплата розробника «r_k» визначається за формулою:

$$r_k = \frac{M_k}{F_к^{мес}} \quad (4.3)$$

де M_k – місячна зарплата-го розробника, грн.;

$F_{kміс}$ – місячний час його роботи розробника.

$$r_k = \frac{7000}{23 \cdot 8} = 38,04 \quad (4.4)$$

Розрахунок складністі розробки T_k для кожного розробника здійснюється за формулою:

$$T_k = t_{1k} \quad (4.5)$$

де t_1k– час, витрачений на кожному етапі розробки k-м розробником, год.

$$Tk = 52 \quad (4.7)$$

Витрати для комп'ютерного часу «$З_2$» розраховуються за формулою:

$$З_2 = C_к \cdot F_0 \quad (4.8)$$

де $C_к$ – вартість комп'ютерної години, грн.;

F_0 – витрати комп'ютерного часу на розробку програми, год.

$$З2 = 1,98 * 427 = 819,84$$

$$С_к= С_а + С_є + С_{то}, \quad (4.9)$$

де $С_а$ – амортизаційні відрахування, грн.;

$С_е$ – енерговитрати, грн.;

$С_{то}$ – витрати на технічне обслуговування, грн.

$С_к= 1,52 + 0,25 + 0,21 = 1,98$

Амортизаційні відрахування "$С_а$" розраховуються за формулою:

$$С_А = \sum_{i=1}^{n} = С_і N_{Ai} / F_{годі} \quad (4.10)$$

де $С_і$ – балансова вартість і-го обладнання, яке використовується для створення ПЗ, грн.;

$N_а$ – річна норма амортизації і-го устаткування, частки;

$F_{год}$ – річний фонд часу роботи і-го устаткування, год.

$С_і = 2139 + 1234 = 3,373$

$С_а = 4992 * 0,6 / 2318 = 1,29$

Енерговитрати "$С_е$" розраховуються за формулою:

$$С_е= Р_е \cdot С_{кВт} \quad (4.11)$$

де $Р_е$ – витрата електроенергії, споживаної комп'ютером;

$С_{квт}$ – вартість 1 кВт/г електроенергії, грн.

$С_е = 0,36 * 0,671 = 0,24$

Непрямі витрати "$З_з$" визначаються за формулою:

$$З_з = С_1 + С_2 \quad (4.12)$$

де $С_1$ – витрати на утримання приміщень, грн.;

C_2 – витрати на електроенергію та опалення грн.;

Витрати на утримання приміщень складають 2 – 2,5% від вартості приміщення. Витрати на електроенергію та опалення складають 0,2 – 0,5% від вартості приміщення.

$$C_1 = 11 * 1284 * 2\% = 282$$

$$C_2 = 10 * 2384 * 0{,}6\% = 1430$$

$$З_3 = 282 + 1{,}430 = 1712$$

Розрахунок капітальних витрат на створення програмного забезпечення відбувається наступним чином.

Формула для обчислення розробки проекту “$B_р$”:

$$B_р = \sum_{i=1}^{n} C_p^i \cdot T_p^i \cdot n \qquad (4.11)$$

де $T_р$ – трудомісткість і-го розрахунку, год.;

$C_р$ – годинна вартість і-го розрахунку, грн.;

n – число експериментів, зроблених за рік, шт.

$B_р = 119{,}6 * 0{,}636 * 650 = 49194$

Вартість проектування «$C_р$» вираховується за формулою:

$$C_р = C_р^1 + C_р^2 \qquad (4.12)$$

де $C_{р1}$ – витрати на оплату праці зарплати, грн.;

$C_{р1}$ – непрямі витрати, грн.

$C_р = 55{,}11 + 58{,}22 = 113$

Витрати на оплату зарплати персоналу «$C_{р1}$» вираховується за формулою:

$$C_p^1 = \sum_{k=1}^{n} N_k \cdot R_k \cdot k_{зар,} \quad (4.13)$$

де $N_к$ – кількість працівників k-й професії, які розробляють проект;

R_k – зарплата одного працівника k-й професії в годину, грн.;

$K_{зар}$ – коефіцієнт накопичення на фонд зарплати;

k – кількість використаних професій.

$$C_p^1 = (1 \cdot 14{,}33 + 1 \cdot 12{,}22) \cdot 1{,}3 = 34.51$$

Непрямі витрати «$C_{р2}$» визначаються за формулою:

$$C_{р2} = C_1 + C_2, \quad (4.14)$$

де C_1 – витрати на утримання приміщень за місячний фонд часу роботи, грн.;

C_2 – інші витрати (вартість різних матеріалів, які використовувалися при розробці проекту, послуги сторонніх організацій тощо).

$$C_1 = 282 / 23 * 8 = 1.53$$

$$C_2 = 1430 / 23 * 8 = 497.3$$

$$C_{р2} = 1.53 + 497.3 = 4$$

У разі створення одного ПЗ економічний ефект «Еф» визначається за формулою:

$$E_ф = B_р – E_н \cdot K_{Затр}, \quad (4.15)$$

де $B_р$ – поточні витрати, грн.;

$K_{затр}$ – капітальні витрати на створення програмного виробу, грн;

$E_н$ – нормативний коефіцієнт економічної ефективності капіталовкладень, 0,15 частки.

$E_ф = 31047{,}06 – 18155{,}2 \cdot 0{,}12 = 28858.436$

Коефіцієнт економічної ефективності капіталовкладень «ЕР» показує обсяг річного приросту прибутку або зменшення вартості в результаті використання програмного забезпечення на одну гривню витрат (капіталовкладень), розраховується за формулою:

$$E_p = \frac{B_p}{K_{затр}} \quad (4.16)$$

де $B_р$ – поточні витрати, грн.;

$K_{затр}$ – капітальні витрати на створення програмного забезпечення, грн;

$$E_р = 21047 / 18155 = 1.71$$

Проект вважається ефективним, якщо справджується наступна нерівність.

$$E_р > E_н \quad (4.17)$$

$$1,71 > 0,15$$

Термін окупності капіталовкладень «ТР» – для періоду часу, протягом якого окупаються витрати на ПЗ:

$$T_p = \frac{1}{E_p} \quad (4.18)$$

де ЕР – коефіцієнт економічної ефективності капіталовкладень.

$$T_р = 1/ 171 = 0.58$$

Даний економічний розрахунок показує, що розробка та запровадження додатку Warehouse Manager є економічно виправданою і доцільною. Про це свідчать наступні цифри

1. Економічний ефект складає 28858.436 грн.
2. Період окупності капіталовкладень в ПК складе півтора року.

ВИСНОВКИ

Для того, щоб додаток максимально відповідав поставленим вимогам, був зручним та багатофункціональним, перед його проектуванням та розробкою були розглянуті та проаналізовані існуючі методології та літературні джерела.

Перелічено основні види та призначення мобільних додатків для поштових повідомлень. Проведено коротку характеристику таких основних видів сучасних мобільних додатків, що працюють з поштовими повідомленнями, як Pushover, Мобільний додаток від Нова Пошта, Мобільний додаток Укрпошта, Addappt, Windows Phone (Windows Mobile), TravelPost, App Store Preview.

Зроблено характеристику операційної системи Android 6 та названо її ключові особливості. Описано головні особливості таких технологій, як JavaScript, React Native, Redux, Firebase, вказано їх переваги та недоліки. Деякі з рішень, що були знайдені в цих методологіях, були використанні під час проектування та розробки мобільного додатка. Проектування та розробка програми проводилась з використанням сучасних технологій та підходів до вирішення цих питань, також була спроектована база даних серверної частини.

Деталізовано основні етапи проектування архітектури програмної системи та наведено розробку структури бази даних (Firebase Realtime Database).

Використання шаблонів проектування та окремі дизайнерські рішення розробника дозволили зробити систему гнучкою, що легко піддається модифікації, зрозумілою та надійною. Використання сучасних гнучких технологій розробки дозволить і надалі розширювати функціональність системи або вносити зміни до її поточної функціональності. Інтерфейс програми було спроектовано з урахуванням потреб цільового користувача. Було досягнуто основної мети зробити інтерфейс зручним та зрозумілим. Інтерфейс мав бути спроектований так, щоб будь які дії займали у користувача мінімальну кількість часу, адже планується велика інтенсивність використання додатка.

Була проаналізована економічна ефективність розроблюваної системи. Проведені розрахунки свідчать про економічну доцільність розробки та провадження нового програмного продукту. А також було проведено визначення трудомісткості розробки програмного забезпечення та розраховані витрати на створення програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бордюг В. Л. Выбор инструментов для разработки мобильного приложения методом анализа иерархии. Уфа, 2014. 14–22.
2. Альтернатива SMS від Google. URL: http: // www. mobilemarketing. com. ua / 2016 / 03 / 11 / alternativa-sms-vid-google/#more – 16416. (дата звернення: 18.02.2020)
3. Брайн Харді, Білл Філліпс Android. Програмування для професіоналів. Пітер, 2016. 640 с.
4. Блох Д. Java Эффективное программирование. Санкт-Петербург, 2002. 240 с.
5. Андрєєва Т. Є. Проектний менеджмент як засіб досягнення мети підприємства. Санкт-Петербург, 2011. С. 364–370.
6. Амелин К. С., Граничин О. Н., Кияев В. И., Корявко А. В. Введение в разработку приложений для мобильных платформ. ВВМ, 2011. 518 с.
7. Бродська А. О. Використання інформаційних технологій в управлінні проектами підприємств URL: http://www.irbis-nbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?C21COM=2&I21DBN=UJRN&P21DBN=UJRN&IMAGE_FILE_DOWNLOAD=1&Image_file_name=PDF/Urss_2013_13_4.pdf (дата звернення: 22.03.2020)
8. Голощапов А. Л. Создание приложений для смартфонов и планшетных. БХВ-Петербург, 2013. 832 с.
9. Варакин М. В. Разработка мобильных приложений под Android. УЦ «Специалист» при МГТУ им. Н. Э. Баумана, 2012. С. 40–48.
10. Бурнет Эд. Разработка мобильных приложений. Санкт-Петербург, 2012. 256 с.

11. Вендров А. М. Проектирование программного обеспечения экономических информационных систем. URL: http://www.immsp.kiev.ua/postgraduate/Biblioteka_trudy/ProektirovProgrVendrov2005.pdf (дата звернення: 02.02.2020)
12. Майер, Рето Android 4. Программирование приложений для планшетных компьютеров и смартфонов. Санкт-Петербург, 2013. 816 с.
13. Офіційна документація фреймворку React Native: https://reactnative.dev/. (дата звернення – 10.04.2020)
14. Разработка мобильных приложений: с чего начать. URL: https: // habrahabr. ru / company / mailru / blog / 179113. (дата звернення: 06.04.2020)
15. Триус Ю. В., Франчук В. М., Франчук Н. П. Організаційні й технічні аспекти використання систем мобільного навчання навч.-метод. посіб. НПУ ім. М. П. Драгоманова, 2011. 53–62 с.
16. Дейтел П. Android для программистов: создаем приложения. Санкт-Петербург, 2012. 154 с.
17. Гарнаев А. WEB-программирование на Java и JavaScript. Санкт-Петербург, 2012. 179 с.
18. Пол Дейтел. Android для разработчиков. Санкт-Петербург, 2016. 512 с.
19. Голощапов А. Л. Google Android. Создание приложений для смартфонов и планшетных ПК. Санкт-Петербург, 2012. 230 с.
20. Мельникова О. М.: Смартфоны на Android. Санкт-Петербург, 2013.103–109 с.
21. Коматинэни С. Android для профессионалов. Санкт-Петербург, 2017. 880 с.
22. Майер, Рето Android 2. Программирование приложений для планшетных компьютеров и смартфонов. М.: Санкт-Петербург, 2011. 672 с.

23. Васильев А. Н. Java. Объектно-ориентированное программирование. Учебное пособие. Санкт-Петербург, 2011. 400 с.
24. Офіційна документація фреймворку Redux: https://redux.js.org/. (дата звернення: 02.05.2020)
25. Васильев А. Н. Java. Объектно-ориентированное программирование: навч.-метод. посіб. Санкт-Петербург, 2011. 400 с.
26. Роджерс Р., Ломбардо Д. Android. Разработка приложений. ЭКОМ Санкт-Петербург, 2010. 400 с.
27. Боголюбов Л. Global WebIndex исследовал распространение мобильных устройств в мире. URL: http: // apptractor. ru / info / devices / globalwebindex-issledoval- rasprostranenie-mobilnyih-ustrovstv-v-mire. html. (дата звернення: 15.03.2020)
28. Шелестун А. Сучасні інструменти мобільного маркетингу. Київ : Інститут журналістики КНУ ім. Т. Шевченка. № 3. 2014. 53 с.
29. Яценков В. С. Java за неделю: Вводный курс. Санкт-Петербург, 2018. 312 с.
30. Харди Б. Android приграмирование для профессионалов. Санкт-Петербург, 2016. 636 с.
31. Хашими С. Разработка приложений для Android Санкт-Петербург, 2011. 736 с.
32. Innovations in Science and Technology: the XVI All-Ukrainian R&D Students Conference Proceeding. Київ, 2016. 116 p.
33. Android Studio and SDK Tools. URL: https://developer.android.com/. (дата звернення – 10.12.2019)
34. Фелкер Д. Android. Разработка приложений для чайников.: Санкт-Петербург, 2012. 336 с.
35. Філліпс Б., Стюарт К., Марсикано К. Android. Санкт-Петербург, 2017. 688 с.

36. Харди Б., Филлипс Б. Программирование под Android. Санкт-Петербург, 2014. 592 с.
37. Шматко О. В. Аналіз методів і технологій розробки мобільних додатків для платформи Android. URL: http://repository.kpi.kharkov.ua/bitstream/KhPI-Press/37731/1/Book_2018_Shmatko_Analiz_metodiv_Ch_2.pdf (дата звернення: 28.03.2020)
38. Розробка веб-додатків, мобільних додатків та порталів URL: http: // ittel. com. ua / informacijni-texnologiyi / rozrobka-mobilnih-dodatkiv / (дата звернення: 28.03.2020)
39. Кирпичников А. П., Ляшева С. А. Автоматизированная система взаимодействия пользователей с базой данных посредством приложений для мобильных устройст. Санкт-Петербург, 235–238 с.
40. Технології створення мобільних додатків. URL: http: // group-global. org / ru / publication / 63343-tehnologiisozdaniya-mobilnyh-prilozheniy (дата звернення: 02.04.2020)
41. Пастернак І. В. Розробка мобільного додатку для біржі. Тернопіль: ТНТУ, 2018. 8 с.
42. Офіційна документація фреймворку React Native: URL: https://docs.expo.io/ (дата звернення: 20.03.2020)
43. Орлов С. А. Теория и практика языков программирования. Санкт-Петербург, 2017. 688 с.
44. Кирпичников А. П., Ляшева С. А. Автоматизированная система взаимодействия пользователей с базой данных посредством приложений для мобильных устройств. Санкт-Петербург, 2015. 235–238 с.
45. Кармен Делессіо, Лорен Дерсі, Шейн Кондер Створення додатків для Android за 24 години. Санкт-Петербург, 2015. 528 с.

46. Гриффитс Д. Head First. Программирование для Android. М.: Санкт-Петербург, 2016. 704 с.
47. Елисеев Р. Разработка приложений для ОС Android. URL: http: // www. intuit. ru / studies / courses / 3703 / 945 / info.
(дата звернення: 20.03.2020)
48. CIT 2:2018. Стандарт кафедри інформаційних технологій. Дипломний проект. Вимоги до змісту та оформлення [Чинний від 2018-09-03]. Вид. офіц. Івано-Франківськ, 2018. 43 с.

ДОДАТОК А

Код мобільного додатку Warehouse manager

Лістинг файлу App.js

```
import React, { Component } from 'react';
import {
    createAppContainer,
    createSwitchNavigator,
    createStackNavigator,
    createDrawerNavigator,
    createBottomTabNavigator
} from 'react-navigation';
import * as firebase from 'firebase/app';
import { firebaseConfig } from './config/config';
import { Ionicons } from '@expo/vector-icons';
import { ActionSheetProvider } from "@expo/react-native-action-sheet";

import WelcomeScreen from './screens/AppSwichNavigator/WelcomeScreen';
import HomeScreen from './screens/HomeScreen';
import LoginScreen from './screens/LoginScreen';
import SettingsScreen from './screens/SettingsScreen';
import LoadingScreen from './screens/AppSwichNavigator/LoadingScreen';
import ParcelsReceivedScreen from './screens/HomeTabNavigator/ParcelsReceivedScreen';
import ParcelsOnWayScreen from './screens/HomeTabNavigator/ParcelsOnWayScreen';

import CustomDrawerComponent from './screens/DrawerNavigator/CustomDrawerComponent';
import { Provider } from 'react-redux';
import store from './redux/store';
import ParcelsCountContainer from './redux/containers/ParcelsCountContainer';

export default class App extends Component {
    constructor(){
        super();
        this.initializeFirebase()
    }
    initializeFirebase = () => {
        firebase.initializeApp(firebaseConfig);
    };
    render() {
        return(
            <Provider store={store}>
                <ActionSheetProvider>
                    <AppContainer/>
                </ActionSheetProvider>
```

Продовження додатку А

```
</Provider>
        )
    }
}

const HomeTabNavigator = createBottomTabNavigator({
    HomeScreen: {
        screen: HomeScreen,
        navigationOptions: {
            tabBarLabel: 'Total Parcel',
            tabBarIcon: ({tintColor}) => (
                <ParcelsCountContainer            color={tintColor}
type='parcels' />
            )
        }
    },
    ParcelsOnWayScreen: {
        screen: ParcelsOnWayScreen,
        navigationOptions: {
            tabBarLabel: 'On The Way',
            tabBarIcon: ({tintColor}) => (
                <ParcelsCountContainer            color={tintColor}
type='parcelsOnWay' />
            )
        }
    },
    ParcelsReceivedScreen: {
        screen: ParcelsReceivedScreen,
        navigationOptions: {
            tabBarLabel: 'Parcel Received',
            tabBarIcon: ({tintColor}) => (
                <ParcelsCountContainer            color={tintColor}
type='parcelsReceived' />
            )
        }
    }
},{
    tabBarOptions: {
        style: {
            backgroundColor: '#fff'
        },
        activeTintColor: '#c513af',
        inactiveTintColor: '#a7a9ac'
    }
});

const HomeStackNavigator = createStackNavigator({
    HomeTabNavigator: {
        screen:HomeTabNavigator,
        navigationOptions: ({navigation}) => {
```

Продовження додатку А

```
return {
                headerLeft:(
                    <Ionicons
                        name='md-menu'
                        size={30}
                        color={'#c513af'}
                        onPress={() => { navigation.openDrawer()}}
                        style={{marginLeft: 20}}
                    />
                ),
            }
        }
    }
},{
    defaultNavigationOptions: {
        headerStyle: {
            backgroundColor: '#fff',
            borderBottomWidth: 0
        },
        headerTintColor: '#a7a9ac'
    }
});

HomeTabNavigator.navigationOptions = ({navigation}) => {
    const                        {routeName}                        =
navigation.state.routes[navigation.state.index];

    switch(routeName) {
        case 'HomeScreen':
            return{
                headerTitle: 'Total Parcel'
            };
        case 'ParcelsOnWayScreen':
            return{
                headerTitle: 'On The Way'
            };
        case 'ParcelsReceivedScreen':
            return{
                headerTitle: 'Parcel Received'
            };
        default:
            return{
                headerTitle: 'Warehouse Manager'
            };
    }
};

const AppCreateDrawerNavigator = createDrawerNavigator(
    {
```

Продовження додатку А

```
HomeStackNavigator: {
            screen: HomeStackNavigator,
            navigationOptions: {
                title: 'Home',
                drawerIcon: () => <Ionicons name='md-home' size={24}
/>
            }
        },
        SettingsScreen: {
            screen: SettingsScreen,
            navigationOptions: {
                title: 'Settings',
                drawerIcon:  ()  =>  <Ionicons  name='md-settings'
size={24} />
            }
        }
    },
    {
        contentComponent: CustomDrawerComponent,
        contentOptions: {
            activeTintColor: '#17bebb'
        }
    }
);

const LoginStackNavigator = createStackNavigator({
    WelcomeScreen: {
        screen: WelcomeScreen,
        navigationOptions: {
            header: null,
        },
    },
    LoginScreen: LoginScreen
},{
    mode:'model',
    defaultNavigationOptions: {
        headerStyle: {
            backgroundColor: '#fff'
        }
    }
});

const AppSwitchNavigator = createSwitchNavigator({
    LoadingScreen,
    LoginStackNavigator,
    AppCreateDrawerNavigator
});

const AppContainer = createAppContainer(AppSwitchNavigator);
```

Продовження додатку А

Лістинг файлу SettingsScreen.js

```
import React, { Component } from 'react';
import {View, Text} from 'react-native';
import CustomActionButton from '../components/CustomActionButton';
import * as firebase from 'firebase/app';
import 'firebase/auth';

export default class SettingsScreen extends Component {

    signOut = async () => {

        try {
            await firebase.auth().signOut();
            this.props.navigation.navigate('WelcomeScreen')
        } catch (err) {
            alert('Unable to sign out right now')
        }
    };

    render() {
        return (
            <View style={{
                flex:1,
                alignItems: 'center',
                justifyContent: 'center'
            }}>
                <CustomActionButton
                    onPress={ this.signOut }
                    style={{
                        width: 200,
                        backgroundColor: '#17bebb'
                    }}
                    title='Log Out'>
                    <Text style={{color: '#fff'}}> Log Out </Text>
                </CustomActionButton>
            </View>
        );
    }
}
```

Лістинг файлу LoginScreen.js

```
import React, {Component} from 'react';
import {View, Text, TextInput, ActivityIndicator} from 'react-
native';
import CustomActionButton from '../components/CustomActionButton';
import * as firebase from 'firebase/app';
import 'firebase/auth'
```

Продовження додатку А

```
import 'firebase/database'

export default class LoginScreen extends Component {
    constructor() {
        super();
        this.state = {
            email: '',
            password: '',
            isLoading: false
        }
    }

    onSignIn = async () => {
        if (this.state.email && this.state.password) {
            this.setState({ isLoading: true });
            try {
                const response = await firebase.auth()
                    .signInWithEmailAndPassword(this.state.email,
                        this.state.password);
                if (response) {
                    this.setState({isLoading: false})

this.props.navigation.navigate('LoadingScreen');
                }
            } catch (err) {
                this.setState({isLoading: false})
                switch (err.code) {
                    case 'auth/user-not-found':
                        alert('A user with this email does not
exist. Try signing Up');
                        break;
                    case 'auth/invalid-email':
                        alert('Please Enter an email address');
                        break;
                }
            }
        } else {
            alert('Enter email and password');
        }
    };

    onSignUp = async () => {
        if (this.state.email && this.state.password) {
            this.setState({ isLoading: true });
            try {
                const response = await firebase.auth()

.createUserWithEmailAndPassword(this.state.email,
                        this.state.password);
```

Продовження додатку А

```
if (response) {
                    this.setState({isLoading: false});
                    // sign in user
                    const          user          =          await
firebase.database().ref('users/').child(response.user.uid)
                                          .set({email:
response.user.email, uid: response.user.uid});


this.props.navigation.navigate('LoadingScreen');
                }
            } catch (error) {
                if (error.code === 'auth/email-already-in-use') {
                    alert('User already Exists. Try Loggin in');
                }
            }
        } else {
            alert('Please enter email & password')
        }
    };

    render() {
        return (
            <View style={{
                flex: 1,
                alignItems: 'center',
                backgroundColor: '#fff'
            }}>
                <View style={{marginTop: 25, fontSize:21}}>
                    <Text>E-mail:</Text>
                    <TextInput
                        style={{
                            borderWidth: 1,
                            backgroundColor: '#fff',
                            borderColor: '#c513af',
                            width: 280,
                            height: 50,
                            borderRadius: 5,
                            paddingLeft: 15,
                            marginTop: 3
                        }}
                        placeholder="abc@gmail.com"
                        keyboardType="email-address"
                        onChangeText={email                          =>
this.setState({email})}
                    />
                </View>
                <View style={{marginTop: 10, fontSize:21}}>
                    <Text>Password:</Text>
```

Продовження додатку А

```
<TextInput
                        style={{
                            borderWidth: 1,
                            backgroundColor: '#fff',
                            borderColor: '#c513af',
                            width: 280,
                            height: 50,
                            borderRadius: 5,
                            paddingLeft: 15,
                            marginTop: 3
                        }}
                        placeholder="password"
                        secureTextEntry
                        onChangeText={password                    =>
this.setState({password})}
                    />
                </View>
                <View style={{marginTop: 20}}>
                    <CustomActionButton
                        onPress={this.onSignIn}
                        style={{
                            width: 200,
                            backgroundColor: '#17bebb',
                            marginBottom: 10,
                        }}
                        title='Login'>
                        <Text style={{color: '#fff'}}>Login</Text>
                    </CustomActionButton>
                    <CustomActionButton
                        onPress={this.onSignUp}
                        style={{
                            width: 200,
                            backgroundColor: '#fff',
                            marginBottom: 10,
                            borderColor: '#3a3a3a',
                            borderWidth: 0.5
                        }}
                        title='Sign Up'>
                        <Text    style={{color:    '#3a3a3a'}}>Sign
Up</Text>
                    </CustomActionButton>
                </View>
                {
                    this.state.isLoading ?
                        <View style={{
                            justifyContent: 'center',
                            alignItems: 'center',
                            zIndex: 5,
                            elevation: 1000,
```

Продовження додатку А

```
marginTop: 15
                        }}>
                            <ActivityIndicator         size='large'
color='#5dba16'/>
                        </View>
                        : null
                }
            </View>
        );
    }
}
```

Лістинг файлу HomeScreen.js

```
import React, {Component} from 'react';
import {
    StyleSheet,
    Text,
    View,
    SafeAreaView,
    TextInput,
    FlatList,
    Image,
    ActivityIndicator
} from 'react-native';

import CustomActionButton from '../components/CustomActionButton';
import ListItem from '../components/ListItem';

import * as firebase from 'firebase/app';
import { snapshotToArray } from '../helpers/firebaseHelpers';
import * as Animatable from 'react-native-animatable';
import { connect } from 'react-redux';
import {compose} from 'redux';
import {connectActionSheet} from '@expo/react-native-action-sheet';
import { Ionicons } from '@expo/vector-icons';
import ListEmptyComponent from '../components/ListEmptyComponent';
import Swipeout from 'react-native-swipeout';
import * as ImageHelpers from '../helpers/ImageHelpers';
import 'firebase/storage';

class HomeScreen extends Component {
    constructor() {
        super();
        this.state = {
            isAddNewDepartureVisible: false,
            departureCode:'',
            department:'',
            deliveryCompany: '',
```

Продовження додатку А

```
            parcels: [],
            currentUser: {},
            parcelsOnWay: [],
            parcelsReceived: []
        };
        this.textInputRef = null;
    }

    componentDidMount = async() => {
        try {
            const { navigation } = this.props;
            const user = navigation.getParam('user');

            const currentUserData = await firebase
                .database()
                .ref('users')
                .child(user.uid)
                .once('value');

            const parcels = await
firebase.database().ref('parcels').child(user.uid).once('value');
            const parcelsArray = snapshotToArray(parcels);

            this.setState({
                currentUser: currentUserData.val(),
            });

            this.props.loadParcels(parcelsArray.reverse());
            this.props.toggleIsLoadingParcel(false);
        } catch (err) {
            console.log(err);
        }
    };

    addParcel = async (departureCode, deliveryCompany, department)
=> {
        this.setState({
            departureCode: '',
            department: '',
            deliveryCompany: '',
            isAddNewDepartureVisible: false
        });
        this.textInputRef.setNativeProps({ text: '' });
        this.props.toggleIsLoadingParcel(true);
        try {
            const snapshot = await firebase
                .database()
                .ref('parcels')
                .child(this.state.currentUser.uid)
```

Продовження додатку А

```
.orderByChild('departureCode')
                .equalTo(departureCode)
                .once('value');

            if(snapshot.exists()) {
                alert('Unable to add Parcel already exist')
            } else {
                const key = await firebase
                    .database()
                    .ref('parcels')
                    .child(this.state.currentUser.uid)
                    .push().key;

                const response = await firebase
                    .database()
                    .ref('parcels')
                    .child(this.state.currentUser.uid)
                    .child(key)
                    .set({
                        departureCode: departureCode,
                        deliveryCompany: deliveryCompany,
                        department: department,
                        received: false
                    });

                this.props.addParcel({
                    departureCode: departureCode,
                    deliveryCompany: deliveryCompany,
                    department: department,
                    received: false,
                    key:key
                });
                this.props.toggleIsLoadingParcel(false);
            }
        } catch (error) {
            console.log(error);
            this.props.toggleIsLoadingParcel(false);
        }
    };

    markAsReceived = async (selectedParcel, index) => {
        try {
            this.props.toggleIsLoadingParcel(true);

            await firebase.database().ref('parcels')

.child(this.state.currentUser.uid).child(selectedParcel.key)
                .update({received: true});
```

Продовження додатку А

```
let parcels = this.state.parcels.map(parcel => {
                if            (parcel.departureCode            ===
selectedParcel.departureCode) {
                    return {...parcel, received: true}
                }
                return parcel
            });
            let parcelsOnWay = this.state.parcelsOnWay.filter(
                parcel      =>      parcel.departureCode      !==
selectedParcel.departureCode
            );

            this.props.markParcelAsReceived(selectedParcel);
            this.props.toggleIsLoadingParcel(false);

        } catch(err) {
            console.log(err);
            this.props.toggleIsLoadingParcel(false);
        }
    };

    markAsNotReceived = async (selectedParcel, index) => {
        try {
            this.props.toggleIsLoadingParcel(true);

            await firebase
                .database()
                .ref('parcels')

.child(this.state.currentUser.uid).child(selectedParcel.key)
                .update({received: false});

            this.props.markParcelAsNotReceived(selectedParcel);
            this.props.toggleIsLoadingParcel(false);
        } catch (err) {
            console.log(err);
            this.props.toggleIsLoadingParcel(false);
        }
    };

    uploadImage = async (image, selectedParcel) => {
        const ref = firebase
                .storage()
                .ref('parcels')
                .child(this.state.currentUser.uid)
                .child(selectedParcel.key);

        try {
            const blob = await ImageHelpers.prepareBlob(image.uri);
```

Продовження додатку А

```
const snapshot = await ref.put(blob);

            let dowloadUrl = await ref.getDownloadURL();

            await firebase
                .database()
                .ref('parcels')
                .child(this.state.currentUser.uid)
                .child(selectedParcel.key)
                .update({image: dowloadUrl});

            blob.close();

            return dowloadUrl

        } catch (err) {
            console.log(err);
        }
    };

    openImageLibary = async (selectedParcel) => {
        const result = await ImageHelpers.openImageLibary();

        if (result) {
            this.props.toggleIsLoadingParcel(true);
            const  downloadUrl  =  await  this.uploadImage(result,
selectedParcel);

            this.props.updateParcelImage({...selectedParcel,   uri:
downloadUrl});
            this.props.toggleIsLoadingParcel(false);
        }
    };

    openCamera = async (selectedParcel) => {
        const result = await ImageHelpers.openCamera();

        if (result) {
            this.props.toggleIsLoadingParcel(true);
            const  downloadUrl  =  await  this.uploadImage(result,
selectedParcel);

            this.props.updateParcelImage({...selectedParcel,   uri:
downloadUrl});
            this.props.toggleIsLoadingParcel(false);

        }
    };
```

Продовження додатку А

```
addParcelImage = (selectedParcel) => {
        const options = ['Select from Photos', 'Camera', 'Cancel'];
        const cancelButtonIndex = 2;

        this.props.showActionSheetWithOptions(
            {
            options,
            cancelButtonIndex
            },
            buttonIndex => {
                if (buttonIndex === 0) {
                    this.openImageLibary(selectedParcel)
                }
                else if (buttonIndex === 1) {
                    this.openCamera(selectedParcel)
                }
            }
        )

    };

    deleteParcel = async (selectedParcel, index) => {
        try {
            this.props.toggleIsLoadingParcel(true);

            await firebase
                .database()
                .ref('parcels')

.child(this.state.currentUser.uid).child(selectedParcel.key)
                .remove();

            this.props.deleteParcel(selectedParcel);
            this.props.toggleIsLoadingParcel(false);

        } catch(err) {
            console.log(err);
            this.props.toggleIsLoadingParcel(false);
        }
    };

    renderItem = (item, index) => {

        let swipeoutButtons = [
            {
                text: 'Delete',
                component: (
                    <View style={{flex:1 , alignItems: 'center',
justifyContent: 'center'}}>
```

Продовження додатку А

```
 <Ionicons name='md-trash' size={24} color='#fff'/>
                    </View>
                ),
                backgroundColor: '#c513af',
                onPress:() => this.deleteParcel(item, index)
            }
        ];

        if (!item.received) {
            swipeoutButtons.unshift({
                text: 'Mark Sent',
                component: (
                    <View style={{flex:1,
                        alignItems: 'center',
                        justifyContent: 'center'}}>
                        <Text  style={{color:  '#fff',  textAlign:
'center'}}>Mark as Received</Text>
                    </View>
                ),
                backgroundColor: '#17bebb',
                onPress:() => this.markAsReceived(item,index)
            })
        } else {
            swipeoutButtons.unshift({
                text: 'Mark Not Sent',
                component: (
                    <View  style={{flex:1  ,  alignItems:  'center',
justifyContent: 'center'}}>
                        <Text  style={{color:  '#fff',  textAlign:
'center'}}>Mark Not Received</Text>
                    </View>
                ),
                backgroundColor: '#17bebb',
                onPress:() => this.markAsNotReceived(item,index)
            })
        }

        return (
            <Swipeout
                autoClose={true}
                style={{marginHorizontal:  5,  marginVertical:  5,
backgroundColor: '#f7f7f7'}}
                right={swipeoutButtons}
            >
                <ListItem
                    editable={true}
                    item={item}
                    marginVertical={0}
                    onPress={() => this.addParcelImage(item)}
```

Продовження додатку А

```
>
                    {item.received && (
                        <Ionicons name='md-checkmark'
                                  size={32}
                                  style={{
                                      color: '#17bebb',
                                      marginVertical: 30,
                                      right: 5
                                  }}/>
                    )}
                </ListItem>
            </Swipeout>
        )
    };

    render() {
        return (
            <View style={{
                flex: 1,
                backgroundColor: '#fafafa'
            }}>
                <SafeAreaView/>
                <View style={styles.container}>
                    {this.props.parcels.isLoadingParcels && (
                        <View style={{
                            ...StyleSheet.absoluteFill,
                            alignItems: 'center',
                            justifyContent: 'center',
                            zIndex: 2,
                            evaluate: 1000,
                            position: 'absolute'
                        }}>
                            <ActivityIndicator         size='large'
color='#c513af'/>
                        </View>
                    )}

                    { this.state.isAddNewDepartureVisible && (
                        <View style={styles.textInputContainer}>
                            <Text style={{
                                fontWeight: '800',
                                fontSize: 22,
                                color: '#fff',
                                marginBottom: 5
                            }}>New Departure:</Text>
                            <TextInput
                                style={styles.textInput}
                                placeholder='Departure Code'
                                placeholderTextColor='#a7a9ac'
```

Продовження додатку А

```
 onChangeText={(text) => this.setState({departureCode: text})}
                                ref={component                    =>
{this.textInputRef = component}}>
                            </TextInput>
                            <TextInput
                                style={styles.textInput}
                                placeholder='Department'
                                placeholderTextColor='#a7a9ac'
                                onChangeText={(text)              =>
this.setState({department: text})}
                                ref={component                    =>
{this.textInputRef = component}}>
                            </TextInput>
                            <TextInput
                                style={styles.textInput}
                                placeholder='Delivery Company'
                                placeholderTextColor='#a7a9ac'
                                onChangeText={(text)              =>
this.setState({deliveryCompany: text})}
                                ref={component                    =>
{this.textInputRef = component}}>
                            </TextInput>
                        </View>
                    )}

                    <FlatList
                        data={this.props.parcels.parcels}
                        renderItem={({item},        index)        =>
this.renderItem(item, index)}
                        keyExtractor={(item,        index)        =>
index.toString()}
                        ListEmptyComponent={
                            !this.props.parcels.isLoadingParcels &&
(
                                <ListEmptyComponent        text='No
Parcels Received'/>
                            )
                        }
                    />

                    {this.state.departureCode.length > 0 ? (
                        <CustomActionButton
                            position='right'
                            onPress={()=>
this.addParcel(this.state.departureCode,
this.state.deliveryCompany, this.state.department)}
                            style={styles.addNewParcelButton}>
                            <Text style={{color: 'white', fontSize:
30}}>✓</Text>
```

Продовження додатку А

```
 </CustomActionButton>
                    ) :
                    (
                        <CustomActionButton
                            position='right'
                            onPress={()=>           this.setState({
isAddNewDepartureVisible: !this.state.isAddNewDepartureVisible})}
                            style={styles.addNewParcelButton}>
                            <Text style={{color: 'white', fontSize:
30}}>+</Text>
                        </CustomActionButton>
                    )
                    }

                </View>
                <SafeAreaView/>
            </View>
        );
    }
};

const mapStateToProps = state => {
    return {
        parcels: state.parcels
    }
};

const mapDispatchToProps = dispatch => {
    return {
        loadParcels: parcel => dispatch({
            type: 'LOAD_PARCELS_FROM_SERVER',
            payload: parcel
        }),
        addParcel: parcel =>
            dispatch({type: 'ADD_PARCEL', payload: parcel}),
        markParcelAsReceived: parcel =>
            dispatch({type:   'MARK_PARCEL_AS_RECEIVED',   payload:
parcel}),
        markParcelAsNotReceived: parcel =>
            dispatch({type: 'MARK_PARCEL_AS_NOT_RECEIVED', payload:
parcel}),
        toggleIsLoadingParcel: parcel =>
            dispatch({type:   'TOOGLE_IS_LOADING_PARCEL',   payload:
parcel}),
        deleteParcel: parcel =>
            dispatch({type: 'DELETE_PARCEL', payload: parcel}),
        updateParcelImage: parcel =>
            dispatch({type:     'UPDATE_PARCEL_IMAGE',     payload:
parcel})
```

Продовження додатку А

```
}
};

const wrapper = compose(
    connect(
    mapStateToProps,
    mapDispatchToProps
    ),
    connectActionSheet
);

export default wrapper(HomeScreen);

const styles = StyleSheet.create({
    container: {
        flex: 1,
        position: 'relative'
    },
    header: {
        height: 70,
        borderBottomWidth: 0.5,
        borderBottomColor: '#E9E9E9',
        alignItems: 'center',
        justifyContent: 'center',
        marginTop: 20
    },
    headerTitle: { fontSize: 24 },
    textInputContainer: {
        height: 180,
        flexDirection: 'column',
        margin: 5,
        backgroundColor: '#c513af',
        padding: 10,
        borderRadius:10,
    },
    textInput: {
        flex: 1,
        backgroundColor: '#fff',
        padding: 5,
        fontSize: 22,
        fontWeight: '200',
        color: '#000',
        marginBottom: 8,
        borderRadius:10
    },
    checkmarkButton: {
        backgroundColor: '#38e26f'
    },
    listItemContainer: {
```

Продовження додатку А

```
minHeight: 100,
        flexDirection:'row',
        backgroundColor:'#354D58DA',
        textAlign: 'center',
        marginVertical: 5
    },
    listItemTitleContainer: {
        flex: 1,
        justifyContent: 'center'
    },
    listEmptyComponent: {
        marginTop:50,
        alignItems: 'center'
    },
    addNewParcelButton: {
        backgroundColor: '#17bebb',
        borderRadius: 25
    },
    footer: {
        height: 70,
        flexDirection: 'row',
    },
    listEmptyComponentText: {
        fontWeight:'bold'
    },
    markReadButton: {
        width: 100,
        backgroundColor: '#38e26f'
    }
});
```

Лістинг файлу HomeScreen.js

```
import React, {Component} from 'react';
import { View, ActivityIndicator } from 'react-native';
import * as firebase from 'firebase';
import 'firebase/auth';

export default class LoadingScreen extends Component {
    componentDidMount() {
        this.checkIfLoggedIn()
    }

    checkIfLoggedIn = () => {
        this.unsubscribe                                    =
firebase.auth().onAuthStateChanged((user) => {
            if (user) {
                // navigate to the home page
```

Продовження додатку А

```
this.props.navigation.navigate('HomeScreen', {user});
            } else {
                // navigate user to the login screen

this.props.navigation.navigate('LoginStackNavigator');
            }
        })
    };

    componentWillUnmount() {
        this.unsubscribe();
    }

    render() {
        return (
            <View style={{
                position: 'absolute',
                left: 0,
                right: 0,
                top: 0,
                bottom: 0,
                alignItems: 'center',
                justifyContent: 'center'
            }}>
                <ActivityIndicator color='#c513af' size='large'
                />
            </View>
        );
    }
}
```

Лістинг файлу WelcomeScreen.js

```
import React, { Component } from 'react';
import {View, Text} from 'react-native';
import { Ionicons } from '@expo/vector-icons';

import                    CustomActionButton                    from
'../../components/CustomActionButton';

export default class WelcomeScreen extends Component {
    render() {
        return (
            <View style={{
                flex:1,
                backgroundColor: '#ffffff'
            }}>
                <View style={{
                    flex:1,
```

Продовження додатку А

```
                    alignItems:'center',
                     justifyContent: 'center'
                }}>
                    <Ionicons
                        name='md-cube'
                        size={150}
                        style={{color:'#c513af'}}
                    />
                    <Text style={{
                        fontWeight: '100',
                        fontSize: 28,
                        color: '#232a38'
                    }}>
                        Warehouse Manager
                    </Text>
                </View>
                <View style={{
                    flex:1,
                    alignItems: 'center'
                }}>
                    <CustomActionButton
                        onPress={()                                    =>
this.props.navigation.navigate('LoginScreen')}
                        style={{
                            width: 200,
                            backgroundColor: '#17bebb',
                            marginBottom: 10,
                        }}
                        title='Login in'>
                        <Text   style={{color:   '#fff'}}>   Login
</Text>
                    </CustomActionButton>
                </View>
            </View>
        );
    }
}
```

Лістинг файлу CustomDrawerComponent.js

```
import React, { Component } from 'react';
import {
    View,
    ScrollView,
    SafeAreaView,
    Text,
    Platform
} from 'react-native';
import { DrawerItems } from 'react-navigation';
```

Продовження додатку А

```
import { Ionicons } from '@expo/vector-icons';

export default class CustomDrawerComponent extends Component {
    render() {
        return (
            <ScrollView>
                <SafeAreaView style={{backgroundColor: '#fff'}}/>
                <View style={{
                    height: 170,
                    backgroundColor: '#fff',
                    alignItems:'center',
                    justifyContent:'center',
                    paddingTop: Platform.OS === 'android' ? 20 : 0
                }}>
                    <Ionicons
                        name='md-cube'
                        size={100}
                        color='#c513af'/>
                    <Text style={{
                        fontSize:24,
                        fontWeight: '500',
                        color: '#232a38'
                    }}>
                        Warehouse Manager
                    </Text>
                </View>
                <DrawerItems {...this.props}/>
            </ScrollView>
        );
    }
}
```

Лістинг файлу ParcelsOnWayScreen.js

```
import React, {Component} from 'react';
import {View, Text, FlatList, StyleSheet, ActivityIndicator} from
'react-native';
import ListItem from '../../components/ListItem';
import { connect } from 'react-redux';
import                    ListEmptyComponent                    from
'../../components/ListEmptyComponent';

class ParcelsOnWayScreen extends Component {

    renderItem = (item) => {
        return <ListItem item={item}/>
    };
```

Продовження додатку А

```
render() {
        return (
            <View style={{
                flex: 1,
                backgroundColor: '#f7f7f7',
                paddingLeft: 5,
                paddingRight: 5
            }}>
                {this.props.parcels.isLoadingParcels && (
                    <View style={{
                        ...StyleSheet.absoluteFill,
                        alignItems: 'center',
                        justifyContent: 'center',
                        zIndex: 2,
                        evaluate: 1000
                    }}>
                        <ActivityIndicator            size='large'
color='#c513af'/>
                    </View>
                )}
                <FlatList
                    data={this.props.parcels.parcelsOnWay}
                    renderItem={({item},         index)         =>
this.renderItem(item, index)}
                    keyExtractor={(item,         index)         =>
index.toString()}
                    ListEmptyComponent={
                        !this.props.parcels.isLoadingParcels && (
                            <ListEmptyComponent text='No Parcel On
Way'/>
                        )
                    }
                />
            </View>
        );
    }
}

const mapStateToProps = (state) => {
    return{
        parcels: state.parcels
    }
};

export default connect(mapStateToProps)(ParcelsOnWayScreen);
```

Лістинг файлу HomeScreen.js

```
import React, {Component} from 'react';
```

Продовження додатку А

```
import {ActivityIndicator, FlatList, StyleSheet, Text, View} from
'react-native';
import ListItem from "../../components/ListItem";
import {connect} from "react-redux";
import                    ListEmptyComponent                    from
'../../components/ListEmptyComponent';

class ParcelsReceivedScreen extends Component {

    renderItem = (item) => {
        return <ListItem item={item}/>
    };

    render() {
        return (
            <View style={{
                flex: 1,
                backgroundColor: '#f7f7f7',
                paddingLeft: 5,
                paddingRight: 5
            }}>
                {this.props.parcels.isLoadingParcels && (
                    <View style={{
                        ...StyleSheet.absoluteFill,
                        alignItems: 'center',
                        justifyContent: 'center',
                        zIndex: 2,
                        evaluate: 1000
                    }}>
                        <ActivityIndicator            size='large'
color='#c513af'/>
                    </View>
                )}
                <FlatList
                    data={this.props.parcels.parcelsReceived}
                    renderItem={({item},          index)          =>
this.renderItem(item, index)}
                    keyExtractor={(item,          index)          =>
index.toString()}
                    ListEmptyComponent={
                        !this.props.parcels.isLoadingParcels && (
                            <ListEmptyComponent  text='No  Parcels
Received'/>
                        )
                    }
                />
            </View>
        );
    }
```

Продовження додатку А

```
}

const mapStateToProps = (state) => {
    return{
        parcels: state.parcels
    }
};

export default connect(mapStateToProps)(ParcelsReceivedScreen);
```

Лістинг файлу index.js

```
import {combineReducers, createStore} from 'redux';
import ParcelsReducer from '../reducers/ParcelsReducer';

const store = createStore(
    combineReducers({
        parcels: ParcelsReducer
    })
);

export default store;
```

Лістинг файлу ParcelsCountContainer.js

```
import React from 'react';
import {View, Text} from 'react-native';
import { connect } from 'react-redux';
import PropTypes from 'prop-types';

const ParcelsCountContainer = ({color, type, ...props}) => {
    return (
        <View style={{
            flex: 1,
            paddingTop: 5
        }}>
            <View style={{
                backgroundColor: color,
                borderRadius: 10,
                width: 20,
                justifyContent: 'center',
                alignItems: 'center'
            }}>
                <Text style={{color: '#fff'}}>
                    {props.parcels[type].length || 0}
                </Text>
            </View>
        </View>
    );
```

Продовження додатку А

```
};

const mapStateToProps = (state) => {
    return{
        parcels: state.parcels
    }
};

ParcelsCountContainer.defaultProps = {
    color: '#001fff'
};

ParcelsCountContainer.propTypes = {
    color: PropTypes.string,
    type: PropTypes.string.isRequired
};

export default connect(mapStateToProps)(ParcelsCountContainer);
```

Лістинг файлу ParcelsReducer.js

```
const initialState = {
    parcels: [],
    parcelsOnWay: [],
    parcelsReceived: [],
    isLoading: true,
    image: null
};

const parcels = (state=initialState, action) => {
    switch(action.type) {
        case 'LOAD_PARCELS_FROM_SERVER':
            return {
                ...state,
                parcels: action.payload,
                parcelsOnWay:    action.payload.filter(parcel    =>
!parcel.received),
                parcelsReceived:  action.payload.filter(parcel  =>
parcel.received)
            };

        case 'ADD_PARCEL':
            return {
                ...state,
                parcels: [action.payload, ...state.parcels],
                parcelsOnWay:                        [action.payload,
...state.parcelsOnWay],
            };
```

Продовження додатку А

```
      case 'MARK_PARCEL_AS_RECEIVED':
            return {
                ...state,
                parcels: state.parcels.map(parcel => {
                    if (parcel.key === action.payload.key) {
                        return { ...parcel, received: true};
                    }
                    return parcel
                }),
                parcelsOnWay: state.parcelsOnWay.filter(
                    parcel => parcel.key !== action.payload.key
                ),
                parcelsReceived:         [...state.parcelsReceived,
action.payload]
            };

        case 'TOOGLE_IS_LOADING_PARCEL':
            return {
                ...state,
                isLoadingParcels: action.payload
            };

        case 'MARK_PARCEL_AS_NOT_RECEIVED':
            return {
                ...state,
                parcels: state.parcels.map(parcel => {
                    if (parcel.key === action.payload.key) {
                        return { ...parcel, received: false};
                    }
                    return parcel
                }),
                parcelsOnWay:               [...state.parcelsOnWay,
action.payload],
                parcelsReceived: state.parcelsReceived.filter(
                    parcel => parcel.key !== action.payload.key
                )
            };

        case 'DELETE_PARCEL':
            return {
                parcels: state.parcels.filter(parcel => parcel.key
!== action.payload.key),
                parcelsOnWay:  state.parcelsOnWay.filter(parcel  =>
parcel.key !== action.payload.key),
                parcelsReceived:
state.parcelsReceived.filter(parcel     =>     parcel.key     !==
action.payload.key),
            };
```

Продовження додатку А

```
      case 'UPDATE_PARCEL_IMAGE':
            return {
                ...state,
                parcels: state.parcels.map(parcel => {
                    if (parcel.key === action.payload.key) {
                        return      {      ...parcel,      image:
action.payload.uri};
                    }
                    return parcel
                }),
                parcelsOnWay: state.parcelsOnWay.map(parcel => {
                    if (parcel.key === action.payload.key) {
                        return      {      ...parcel,      image:
action.payload.uri};
                    }
                    return parcel
                }),
                parcelsReceived:   state.parcelsReceived.map(parcel
=> {
                    if (parcel.key === action.payload.key) {
                        return      {      ...parcel,      image:
action.payload.uri};
                    }
                    return parcel
                })
            };

        default:
            return state
    }
};

export default parcels;
```

Лістинг файлу config.js

```
export const firebaseConfig = {
    apiKey: "AIzaSyCbWx-1h3s5YdXLJJFqO9fvSCVhQ1mtjcU",
    authDomain: "react-native-1-9c6d2.firebaseapp.com",
    databaseURL: "https://react-native-1-9c6d2.firebaseio.com",
    projectId: "react-native-1-9c6d2",
    storageBucket: "react-native-1-9c6d2.appspot.com",
    messagingSenderId: "651264416948",
    appId: "1:651264416948:web:f9740b23048bf6c62c8be1",
};
```

Лістинг файлу CustomActionButton.js

```
import React from 'react';
import {
```

Продовження додатку А

```
      View,
    Text,
    TouchableOpacity,
    StyleSheet
} from "react-native";
import PropTypes from 'prop-types';

function getPosition(position) {
    switch (position) {
        case 'left':
            return {position: 'absolute', left: 20, bottom: 200};
        default:
            return {position: 'absolute', right: 20, bottom: 180};
    }
}

const CustomActionButton = ({children, onPress, style, position}) =>
{

    const floatingActionButton = position ? getPosition(position):
[];

    return(
        <TouchableOpacity                          onPress={onPress}
style={floatingActionButton}>
            <View style={[styles.button, style]}>{children}</View>
        </TouchableOpacity>
    );
};

CustomActionButton.propTypes = {
    onPress: PropTypes.func.isRequired,
    children: PropTypes.element.isRequired,
    styles: PropTypes.object
};

CustomActionButton.defaultProps = {
    styles: {}
};

export default CustomActionButton;

const styles = StyleSheet.create({
    button: {
        width: 50,
        height: 50,
        backgroundColor: '#e2001b',
        alignItems: 'center',
        justifyContent: 'center',
```

Продовження додатку А

```
 borderRadius: 5
    }
});
```

Лістинг файлу ListEmptyComponent.js

```
import React, {Component} from 'react';
import {Text, View} from 'react-native';
import PropTypes from 'prop-types';

const ListEmptyComponent = ({text}) => {
    return (
        <View style={{
            marginTop: 50,
            alignItems: 'center'
        }}>
            <Text style={{fontWeight: 'bold'}}>{text}</Text>
        </View>
    );
};

ListEmptyComponent.propTypes = {
    text: PropTypes.string.isRequired
};

export default ListEmptyComponent;
```

Лістинг файлу ListItem.js

```
import React from 'react';
import { Image, Text, View, TouchableOpacity } from "react-native";

import NetworkImage from 'react-native-image-progress';
import ProgressPie from 'react-native-progress/Pie';

const ListItem = ({ item, children, marginVertical, editable,
onPress }) => {
    return(
        <View style={{
            minHeight: 100,
            flexDirection:'row',
            backgroundColor:'#fff',
            textAlign: 'center',
            marginVertical: marginVertical,
            justifyContent: 'center',
            borderRadius: 10,
            borderBottomColor: '#a7a9ac',
            borderBottomWidth: 1,
        }}>
```

Продовження додатку А

```
<View style={{
                height: 65,
                width: 65,
                marginLeft: 10,
                marginTop: 15,
            }}>
                <TouchableOpacity
                    style={{
                        flex: 1
                    }}
                    onPress={() => onPress(item)}
                    disabled={!editable}
                >
                    {item.image ? (
                    <NetworkImage
                        source={{ uri: item.image }}
                        style={{
                            flex: 1,
                            height: null,
                            width: null,
                            borderRadius: 35,
                            borderColor: '#a7a9ac',
                            borderWidth: 0.5
                        }}
                        indicator={ProgressPie}
                        indicatorProps={{
                            size: 40,
                            borderWidth: 0,
                            color: '#c513af',
                            unfilledColor:  'rgba(200,  200,  200,
0.2)'
                        }}
                        imageStyle={{borderRadius: 35}}
                    />
                    ) : (
                    <Image
                        style={{
                            flex: 1,
                            height: null,
                            width: null,
                            borderRadius: 35,
                            backgroundColor: '#a7a9ac'
                        }}
                    />
                    )}
                </TouchableOpacity>
            </View>
            <View style={{
                flex: 1,
```

Продовження додатку А

```
                justifyContent: 'center',
                    marginLeft: 15
                }}>
                    <Text style={{
                        fontWeight: '800',
                        fontSize: 22,
                        color: '#232a38'
                    }}>
                        {item.departureCode}
                    </Text>
                    <Text style={{
                        fontWeight: '100',
                        fontSize: 15,
                        color: '#232a38',
                        marginTop: 5
                    }}>
                        {item.department}
                    </Text>
                    <Text style={{
                        fontWeight: '100',
                        fontSize: 12,
                        color: '#c513af',
                        marginTop: 4
                    }}>
                        Delivery Company: {item.deliveryCompany}
                    </Text>
                </View>
                {children}
            </View>
        )
};

ListItem.defaultProps = {
    marginVertical: 5,
    editable: false
};

export default ListItem;
```

Лістинг файлу ParcelCount.js

```
import React from 'react';
import { Text, View } from "react-native";
import PropTypes from 'prop-types';

const ParcelCount = ({ title, count }) => {
    return(
        <View style={{
            flex: 1,
```

Продовження додатку А

```
            alignItems: 'center',
            justifyContent: 'center'
        }}>
            <Text style={{fontSize: 20}}>{title}</Text>
            <Text>{count}</Text>
        </View>
    )
};

ParcelCount.propTypes = {
    count: PropTypes.number.isRequired,
    title: PropTypes.string
};

ParcelCount.defaultProps = {
    title: 'Title'
};

export default ParcelCount;
```

Лістинг файлу package.json

```
{
  "main": "node_modules/expo/AppEntry.js",
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web",
    "eject": "expo eject"
  },
  "dependencies": {
    "@expo/react-native-action-sheet": "^2.1.0",
    "expo": "^35.0.0",
    "expo-image-picker": "~7.0.0",
    "expo-permissions": "~7.0.0",
    "firebase": "^7.2.2",
    "firebase-admin": "^8.8.0",
    "firebase-functions": "^3.3.0",
    "prop-types": "^15.7.2",
    "react": "16.8.3",
    "react-dom": "16.8.3",
    "react-native":                 "https://github.com/expo/react-
native/archive/sdk-35.0.0.tar.gz",
    "react-native-animatable": "^1.3.3",
    "react-native-gesture-handler": "~1.3.0",
    "react-native-get-sms-android": "^2.1.0",
    "react-native-image-progress": "^1.1.1",
    "react-native-progress": "^4.0.2",
```

Продовження додатку А

```
      "react-native-sms": "^1.11.0",
    "react-native-swipeout": "^2.3.6",
    "react-native-web": "^0.11.7",
    "react-navigation": "^3.13.0",
    "react-navigation-stack": "^1.10.2",
    "react-redux": "^7.1.3",
    "redux": "^4.0.4"
  },
  "devDependencies": {
    "babel-preset-expo": "^7.1.0"
  },
  "private": true
}
```

Лістинг файлу ImageHelpers.js

```
import * as Permissions from 'expo-permissions';
import * as ImagePicker from 'expo-image-picker';
import {Platform} from 'react-native';

export const openImageLibary = async () => {
    const {status} = await
Permissions.askAsync(Permissions.CAMERA_ROLL);
    if (status !== 'granted') {
        alert('Sorry, we need camera roll permission to select
image')
    } else {
        const result = ImagePicker.launchImageLibraryAsync({
            mediaTypes: ImagePicker.MediaTypeOptions.All,
            allowsEditing: true,
            aspect: [1, 1],
            base64: true
        });
        return !result.cancelled ? result : false
    }
};

export const openCamera = async () => {
    const {status} = await Permissions.askAsync(
        Permissions.CAMERA_ROLL,
        Permissions.CAMERA
    );

    if (status !== 'granted') {
        alert('Sorry, we need camera roll permission to select
image')
        return false
    } else {
        const result = ImagePicker.launchCameraAsync({
```

Продовження додатку А

```
            quality: 0.1,
            base64: true,
            allowsEditing: Platform.OS === 'ios' ? false : true,
            aspect: [4, 3]
        });

        return !result.cancelled ? result : false
    }
};

export const prepareBlob = async imageUri => {

    const blob = await new Promise((resolve, reject) => {

        const xml = new XMLHttpRequest();

        xml.onload = function () {
            resolve(xml.response)
        };

        xml.onerror = function (e) {
            console.log(e);
            reject(new TypeError('Image Upload failed'));
        };

        xml.responseType = 'blob';
        xml.open('GET', imageUri, true);
        xml.send();
    });

    return blob;
};
```

Лістинг файлу firebaseHelpers.js

```
export const snapshotToArray = snapshot => {
    let returnArray = [];

    snapshot.forEach((childSnapshot => {
        let item = childSnapshot.val();
        item.key = childSnapshot.key;

        returnArray.push(item);
    }));

    return returnArray;
};
```