МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2022 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

МОБІЛЬНИЙ ЗАСТОСУНОК ОБЛІКУ ТА ПОПЕРЕДЖЕННЯ ПРАВОПОРУШЕНЬ ІЗ ПРИВ'ЯЗКОЮ ДО ГЕОЛОКАЦІЇ

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402. 21810219

Виконав студент 4-го курсу, групи 402
_____ ***К. А. Мурадян***
«__» червня 2022 р.

Керівник: канд. техн. наук, доцент (б.в.з.)
_____***М. Л. Дворецький***
«__» червня 2022 р.

Миколаїв – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____20____р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук Мурадяну Карену Арсеновичу

1. Тема кваліфікаційної роботи «Мобільний застосунок обліку та попередження правопорушень із прив'язкою до геолокації».

Керівник роботи Дворецький Михайло Леонідович, канд. техн. наук, доцент (б.в.з.)

Затв. наказом Ректора ЧНУ ім. Петра Могили від «___» ____ 20__ р. № _____

2. Строк представлення кваліфікаційної роботи студентом «__» _____ 20__ р.

3. Очікуваний результат роботи: мобільний застосунок обліку та попередження правопорушень із прив'язкою до геолокації.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):
- аналіз стану злочинності та протидії їй на території України;
- попередження злочинності у високорозвинених зарубіжних країнах;
- вибір інструментальних засобів створення застосунку;
- програмна реалізація мобільного застосунку обліку та попередження правопорушень із прив'язкою до геолокації

5. Перелік графічного матеріалу: сторінок – 92 , таблиць – 2, рисунків – 61, посилань – 24, додатків – 2, презентація.

6. Завдання до спеціальної частини: «Охорона праці при користуванні екранними пристроями».

7. Консультанти:

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Алексєєва А.А., канд. техн. наук, доцент	

Керівник роботи ___ канд. техн. наук, доцент (б.в.з.) Дворецький М. _____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання _Мурадян К.А._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання « 23 » листопада 2021 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: «Мобільний застосунок обліку та попередження правопорушень із прив'язкою до геолокації»

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників БКР	20.11.2021	20.11.2021	виконано
2	Отримання завдання на виконання БКР	01.12.2021	01.12.2021	виконано
3	Складання календарного плану роботи на весь період виконання БКР	02.12.2021	04.12.2021	виконано
4	Отримання завдання на переддипломну практику	23.05.2022	23.05.2022	виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до БКР	24.05.2022	02.06.2022	виконано
6	Розробка звіту з переддипломної практики	01.06.2022	05.06.2022	виконано
7	Виконання БКР: аналіз предметної сфери, вибір інструментальних засобів створення застосунку, розробка застосунку, тестування	27.02.2022	21.05.2022	виконано
8	Попередній захист БКР на засіданні комісії кафедри	30.05.2022	30.05.2022	виконано
9	Доробка та остаточне оформлення БКР	31.05.2022	11.06.2022	виконано
10	Подання БКР рецензенту	17.06.2022	17.06.2022	виконано
11	Подання БКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	24.06.2022	24.06.2022	
12	Захист БКР перед екзаменаційною комісією (ЕК)	29.06.2022	29.06.2022	

Розробив студент _____Мурадян К.А._____ _____
(прізвище та ініціали) (*підпис*)

Керівник роботи канд. техн. наук, доцент (б.в.з.) Дворецький М. Л_____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«____» _____ 202__ р.

АНОТАЦІЯ

бакалаврської кваліфікаційної роботи студента групи 402 ЧНУ ім. Петра Могили

Мурадяна Карена Арсеновича

Тема: «Мобільний застосунок обліку та попередження правопорушень із прив'язкою до геолокації»

Керівник: кандидат технічних наук, доцент (б.в.з.) кафедри Інженерії програмного забезпечення Дворецький Михайло Леонідович.

Об’єкт дослідження – стан злочинності та методи протидії їй на території України.

Предмет дослідження – засоби оперативного обліку та звітності про зафіксовані правопорушення.

Мета – створення передумов для зниження рівня злочинності за рахунок залучення кожної небайдужої людини до фіксації потенційних правопорушень шляхом розробки мобільного застосунку обліку та попередження правопорушень із прив'язкою до геолокації.

У першому розділі розглядається стан злочинності на території України та найкращі методи її протидії, що встигли зарекомендувати себе закордоном.

У другому розділі було досліджену різницю між нативною розробкою застосунків під окрему операційну систему і кросплатформною розробкою.

У третьому розділі було продемонстровано ключові моменти розробки застосунку обліку та попередження правопорушень із прив'язкою до геолокації.

В останньому спеціальному розділі було розглянуто нормативну базу охорони праці при користуванні екранними приладами.

У результаті виконання роботи було зроблено мобільний застосунок обліку та попередження правопорушень із прив'язкою до геолокації.

Сторінок – 92 , таблиць – 2, рисунків – 61, посилань – 24, додатків – 2.

Ключові слова: правопорушення, попередження правопорушень, кросплатформна мобільна розробка, React Native, Redux.

ABSTRACT

for bachelor's qualification work of a student of 402 group at Petro Mohyla Black Sea National University

Muradian Karen Arsenovich

Topic: "Geolocation-related application of accounting and prevention of offenses"

Supervisor: candidate of technical sciences, senior lecturer of the Department of Software Engineering Dvoretskyi Mykhailo Leonydovych.

The object of the research is the state of crime and methods of combating it in Ukraine.

The subject of the research is the means of operational accounting and reporting of recorded offenses.

The goal is to create preconditions for reducing the level of crime by involving every caring person in the recording of potential offenses by developing a geolocation-related mobile application of accounting and prevention of offenses.

The first section examines the state of crime in Ukraine and the best methods of combating it, which have proven themselves abroad.

The second section explored the difference between native application development for a separate operating system and cross-platform development.

The third section demonstrated the key points of developing a mobile application of accounting and prevention of offenses related to geolocation.

In the last section the normative base of labor protection at use of screen devices was considered.

As a result of the work, a geolocation-related application of accounting and prevention of offenses was made.

Pages – 92, tables – 2, figures – 61, links – 24, appendices – 2.

Keywords: offense, crime prevention, cross-platform mobile development, React Native, Redux.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

КК – Кримінальний кодекс України
ЄРДР – Єдиний реєстр досудових розслідувань
КПК – Кримінальний процесуальний кодекс України
ст. – стаття
ст.ст. – статті
ч. – частина
ч.ч. – частини
п. – пункт
пп. – пункти
п.п. – підпункти
RN – React Native

Пояснювальна записка

до кваліфікаційної роботи

на тему:

«МОБІЛЬНИЙ ЗАСТОСУНОК ОБЛІКУ ТА ПОПЕРЕДЖЕННЯ ПРАВОПОРУШЕНЬ ІЗ ПРИВ'ЯЗКОЮ ДО ГЕОЛОКАЦІЇ»

Спеціальність 122 «Комп’ютерні науки»

122 – БКР – 402.21810219

Виконав студент 4-го курсу, групи 402
_____ *К. А. Мурадян*
(підпис, ініціали та прізвище)
«___» _____ 2022 р.

Керівник: *канд. техн. наук, доцент (б.в.з.)*
(наук. ступінь, вчене звання)
_____ *М. Л. Дворецький*
(підпис, ініціали та прізвище)
«___» _____ 2022 р.

Миколаїв – 2022

ВСТУП

Кожна людина хоче жити у спокійному місті, де вона буде впевнена у стабільності завтрашнього дня. Це досить нечітке поняття насправді складається з групи зрозумілих кожному факторів життя таких, як: низький рівень злочинності, висока середня заробітна плата, можливість особистого розвитку майже у будь-якій сфері життя без необхідності зміни місця проживання, наявність достатньої кількості дитячих садків, шкіл, університетів, магазинів – об’єктів громадської інфраструктури, близький доступ до котрих так сильно впливає на наше розуміння комфорту, та інше.

Ще у далекому, як це здається зараз, 2019-му році президент нашої країни Володимир Зеленський відвідав Миколаїв і під час розмови з виконувачем обов'язків голови Миколаївської обласної державної адміністрації В'ячеславом Бонем сказав наступне «Все говорят, что Николаев – бандитский город. Почему? Вы живете в этом городе. Это нормально?» [2]. Одразу після перегляду цієї ситуації я задумався над тим, що раніше я вже чув подібне висловлювання про наше місто. Ще через два роки, вже у жовтні 2021-го року журнал «Фокус» оприлюднив рейтинг комфортності міст України. У ході складання даного списку було вирахувано один дуже цікавий і, що важливо, об’єктивний показник злочинності на душу населення. На першому місці по цьому критерію опинився Миколаїв з показником 130 правопорушень на 10 тисяч жителів.

Саме після аналізу цього рейтингу я почав думати над тим, який інструмент може хоча б трохи змінити ситуацію у кращу сторону. Одним з головних критеріїв для мене була легкість у використанні для усіх груп населення, котрі, в цілому, вміють користуватися смартфоном, а також оперативність оновлення інформації.

Це обумовило **мету роботи**, яка полягає у створенні інструменту, що дозволятиме кожній людини приймати участь у зниженні рівня злочинності.

Об’єкт дослідження – стан злочинності та методи протидії їй на території України.

Предмет дослідження – методи оперативного обліку та звітності про зафіксовані правопорушення.

Отже, для досягнення поставленої мети треба буде вирішити наступні **задачі**:

1) Проаналізувати стан злочинності та протидії їй на території України.

2) Вивчити методи попередження злочинності у високорозвинених зарубіжних країнах.

3) Зробити вибір інструментальних засобів створення застосунку, опрацювавши особливості розробки мобільних застосунків, та визначити вимоги для програмного продукту.

4) Спроектувати структуру та логіку мобільного застосунку.

5) Здійснити програмну реалізацію мобільного застосунку обліку та попередження правопорушень із прив'язкою до геолокації.

1 АНАЛІЗ СТАНУ ЗЛОЧИННОСТІ НА ТЕРИТОРІЇ УКРАЇНИ ТА МЕТОДИ ЇЇ ПРОТИДІЇ

1.1 Навмисне корегування статистичних даних про стан злочинності

Здоров’я і життя людини, її недоторканність, честь і гідність, особисте відчуття та реальне розуміння власної безпеки – усе це складові разом утворюють найвищу соціальну цінність. Утвердження та дотримання прав і свобод людини має бути гарантованим державою і фактично є головним її обов’язком перед кожним громадянином. Саме ці чинники формують свідомість громадян та їхнє ставлення до держави в цілому, а особливо до органів її управління, в той же час вони найбільшим чином впливають на рейтинг цієї країни на міжнародній платформі, тобто у очах країн, що потенційно будуть її союзниками, а наявність союзників у сучасному світі є складовою розвитку та стабільності будь-якої країни, адже окреме існування в ізольованому світі не може навіть серйозно обговорюватись у наших реаліях.

Global Peace Index – це Міжнародний рейтинг найбезпечніших для життя країн, станом на червень 2020 року Україна займає у ньому 148 місце, що є дуже поганим результатом, особливо зважаючи на те, що загалом у ньому 163 позиції. Для ще більш повного розуміння цієї ситуації слід зазначити, що наша держава посідає свою сходинку прямо між такими країнами, як Венесуела і Нігерія. Соціальні опитування показують дуже напружену картинку: все більша кількість українців перестає довіряти органам правопорядку і прокуратури, а також суддям. Така тенденція є дуже негативною, але якщо, знову ж таки, звернутися до Міжнародного рейтингу найбезпечніших для життя країн, то можна побачити, що місце України у ньому дійсно падає, а отже у громадян є реальні підстави для того, щоб так думати. Якщо аналізувати Global Peace Index то можна самостійно переконатися у цьому. Наприклад, ще у 2009-му році Україна займала 118 місце, потім у період з 2010-го по 2013 роки – 120 місце, а у складному 2014 році наша країна знаходилась на 144 сходинці даного рейтингу.

Для підтвердження реальної картини, котру ілюструє даний рейтинг, можна згадати звуковий запис [1], що був викладений у мережу в грудні 2019-го року, на якому жіночий голос, що дуже нагадує голос керівника слідства Львівської області Наталії Харченко, дає вказівки знизити рівень «негативу» в області. Цитата: "Систематичне невиконання, систематична неорганізація роботи виїзду СОГ і документування. Взагалі нема ніякого бажання організовувати працювати. Хлопці, так робота не буде. І ще, запам'ятайте всі, від завтра все внесення в ЄРДР (Єдиний реєстр досудових розслідувань, - ред.) починаючи від 125, закінчуючи трупом природною смертю виключно за моїм погодженням. Всі почули? Щоб завтра не було, що хтось не почув. Все, що ви будете планувати вносити в ЄРДР, все погоджуєте зі мною. Не може бути така негативна картина".

Зважаючи на подібні інциденти, звіт компанії Kantar online TRACK, що на регулярній основі проводить вимірювання рівня довіри до різних інституцій, а також інспектує якість їхньої роботи, вже не здається чимось дивним. Він показав, що більших з тисячі громадян, котрі були опитані у рамках з 16-го по 30-е травня 2020-го року не довіряли поліції (63%), низько оцінювали роботи органів охорони цивільного порядку (61%) і лише 12% респондентів були впевнені у можливостях поліції захистити їх у разі порушення їх прав.

Дані Судової адміністрації України також свідчать про те, що правоохоронні органи все частіше ухиляються від реєстрації всіх заяв і повідомлень про вчинення злочинів. За цими даними видно, що у 2020-му році кількість скарг про бездіяльність слідчого та прокурора стосовно внесення відомостей про кримінальне правопорушення до ЄРДР різко виросла та становила 37 826 заяв, це становить майже половину від усіх одержаних скарг. У тому ж році від усієї маси заяв було розглянуто і задоволено 22853, що становить 60,4% від усіх заяв. Цей показник ще більше наштовхує на негативні думки, якщо порівняти його с аналогічними даними за попередній 2019-ий рік, де розглянуто було 21832 заяви, що становили собою 53%.

Практика закриття прокурорами і слідчими кримінальних проваджень також позначається на об'єктивності статистичних даних. З усієї маси зареєстрованих кримінальних правопорушень за 2020-ий рік більш ніж у половині (54%) випадків були прийняті рішення про закриття проваджень на реабілітуючих підставах, тобто через факт відсутності події і складу злочину. Цей показник набагато вищий за аналогічні дані у попередніх роках: 47,2% у 2019-му та 46,8% у 2018-му році.

Про погіршення ситуації у відношенні розкриття кримінальних злочинів також свідчить і той факт, що із 13199 особливо тяжких злочинів лише у 5691 випадку, тобто у 43,1%, конкретним особам було пред’явлено підозру, а до суду було направлено і того менше обвинувальних актів – 4033, це лише 30,5%. Виходить, що у кінці року нерозслідуваними залишились дві треті усіх злочинів з цієї групи. Упродовж 2020-го року слідчими усіх правоохоронних органів було закрито майже 120 тисяч особливо тяжких кримінальних правопорушення, але дуже важливо звернути увагу саме на причини закриття. Цілих 110 тисяч діл було закрито за ч.1 п.п.1,2,4,6,9-1 ст.284 КПК, усі ці причини – це відсутність події і складу злочину, і відсутність достатньої кількості доказів для доведення винуватості в суді. Також у дану категорію автоматично додаються і 23 провадження, що були закриті за п.10 цієї ж ст.284 КПК, це діла, строки котрих закінчилися, а особи, що вчинили кримінальні правопорушення, так і не були встановлені, або строк закінчився вже після повідомлення підозри конкретній особі.

Кількість облікованих тяжких злочинів, зареєстрованих у 2019-му році, також менша, ніж у попередні роки. Таких за 2019-й рік назбиралось трохи більше 125 тисяч, з них усього 51894 випадки (41,5%) були доведені до повідомлення про підозру конкретним особам, а це менше, якщо проводити порівняння з попереднім 2018-м роком, де таких випадків було 57770. Обвинувальних актів у кримінальних правопорушеннях цієї ж категорії 2019-го року було і того менше – 45169 (34,5%), роком раніше їх також було більше – 50120 актів. Проблема стає ще більш зрозумілою, якщо брати до уваги і факт того, що досить велика кількість злочинів

з цієї суспільно небезпечної категорії залишилась нерозслідуваною, кількість таких випадків становить 78842, що еквівалентно 63%.

Упродовж 2020-го року було провадження у 19539 тяжких кримінальних правопорушеннях, але і тут цілих 19203 – це ті справи, що прийшлись на реабілітуючі обставини, тобто п.п.1,2,4,6,9-1 ч.1 ст.284 КПК. У тому ж році кількість облікованих злочинів середньої тяжкості становила 53141, що також помітно менше, ніж у попередні роки. Але і з цієї кількості справ про підозру повідомлено було лише у половині випадків, якщо бути точним, то це 25864 підозри (47%). Це трохи більше, ніж аналогічний показник у вчиненні тяжких і особливо тяжких злочинів, але, зважаючи на порівняно незначну складність розслідування подібних видів правопорушень, відсоток залишається дуже низьким. Обвинувальних актів і того менше – 21047, що становить 39,6%. Та в цій категорії злочинів закритих справ за пунктами про відсутність події та складу злочину абсолютна більшість – це 16692 загалом закритих проваджень, майже всі з котрих закриті за п.п.1,2, 4,6,9-1 ч.1 ст.284 КПК.

Помічену тенденцію підтверджує і кількість зареєстрованих у 2020-му році злочинів невеликої тяжкості. Їх кількість становила 98282, що є найменшим показним за попередні 6 років. Ситуація про справи, де конкретним особам повідомлено про підозру, теж не є позитивною, адже їх кількість нараховує усього 30038, що становить 30,6%, а до суду із підозрами направлено навіть менше актів – 27924, тобто 28,4%. Тенденція про закриття злочинів за п.п.1,2,4,6, 9-1 ч.1 ст.284 КПК тут також зберіглась – це 140665 справ, що навіть більше кількості облікованих проваджень.

Підводячи підсумок результативної діяльності органів досудового розслідування і прокуратури за 2020-й рік за даними вищенаведених статистичних даних, можна з упевненістю сказати, що абсолютна більшість усіх закритих кримінальних проваджень була доведена до кінця за реабілітуючими обставинами. Усього таких кримінальних правопорушень налічилося 296266, з яких:

1. 119801 – особливо тяжких.

2. 19203 – тяжких;

3. 16597 – середньої тяжкості;

4. 140665 – невеликої тяжкості;

З таких статистичних даних можна зробити декілька невтішних висновків:

1. Абсолютна більшість кримінальних правопорушень з категорії тяжких та особливо тяжких, що були скоєні упродовж 2020-го року, та тих, котрі перейшли у цей звітній період з минулих років, залишись нерозкритими, а самі злочинці, котрі вдалися до правопорушень, не були покарані, а отже можуть у перспективі нести загрозу для суспільства.

2. Заподіяні матеріальні та моральні збитки за справами, що не були розкриті, не будуть відшкодовані.

3. Величезна кількість закритих проваджень (майже 300 тисяч) була досягнута у результаті їх закриття за реабілітуючими обставинами.

1.2 Попередження злочинності у високорозвинених закордонних країнах

У країнах-членах Європейського союзу виділяють два рівні попередження правопорушень: соціальний і ситуаційний. Соціальне попередження має на меті виправити несприятливі умови формування особистості людини. Наступний рівень попередження базується на тому, що окремі категорії правопорушень відбуваються тільки за певних обставин, у певний час та лише певних місцях. Тобто каталізатором для злочину є сама ситуація, наприклад, вуличні бійки біля кінотеатрів, зґвалтування у скверах і парках тощо. Таке попередження щодо прийнятої у нас термінології відповідає запобіганню та припинення злочинів.

Цікава модель попередження правопорушень є у Германії. Там виділяють первинне, вторинне та третинне попередження. Метою первинного попередження є подолання дефіциту соціальності. Вторинне є зрозумілим і звичним для нас, адже воно передбачає роботу поліцейських органів і пов’язане з правовими засобами утримання від злочинів. У процесі покарання та реабілітації злочинців проводять певні заходи, саме їх і виділяють у третинне попередження правопорушень.

Діяльність щодо запобігання злочинів координується і на міжнародному рівні, для цього створюються відповідні органи, наприклад, Національні ради. У таких органів виділяють основні функції.

1. Збір інформації, планування та виконання програм для попередження злочинів.
2. Координація діяльності поліції.
3. Забезпечення участі населення у подібних заходах.
4. Залучення засобів масової інформації.
5. Налагодження зв’язків та подальша співпраця із законодавчими органами.
6. Науково-дослідницька роботи.
7. Навчальна підготовка.

Якщо ж говорити про практичну діяльність поліції, то у них широко розповсюджений тезис про те, що більшість правопорушень скоюються коли потенційний правопорушник зустрічає слабку або незахищену жертву чи об’єкт. З цього слідує, що превентивні міри мають бути направлені в першу чергу або на правопорушника, або на систему безпеки, або ж на потенційну жертву. Це називається індивідуальною, загальною та віктимологічною профілактикою відповідно. Серед цих мір особливу увагу прийнято приділяти попередженню та цілеспрямованій роботі з населенням, направленій на його самозахист.

У США на всіх рівнях реалізуються програми запобігання злочинів, це досягається завдяки трьом моделям превентивної діяльності: модель громадських установ, модель безпеки індивідуумів, а також модель дії через навколишнє середовище. У деяких штатах головною причиною зниження кількості пограбувань є взаємодія громадян з поліцією для зміцнення правопорядку. Використання практики винагороди за інформацію з оперативно-профілактичним значенням лише спонукає населення до надання допомоги поліції.

У Канаді населення залучають до патрулювання вулиць. Такі випадки не є поодинокими, а скоріше навпаки – вони мають дуже широке використання. Саме у

результаті роботи звичайних людей з поліцією знижується рівень страху перед злочинцями, а також підтримується відчуття особистої безпеки. Не рідкістю також є контрольовані пости, що створюються у зонах підвищеної злочинної активності. Така практика широко використовується, особливо у нічний час.

Великобританія також широко залучає звичайне населення до патрулювання, а також чергування у найбільш кримінальних районах. Практика безкоштовної видачі найбільш відзначеним громадянам поліцейського обмундирування без службової атрибутики, а також рацій, кийків та у деяких випадках справжньої зброї і поліцейських автомобілів для патрулювання також є реальністю. Такий вид діяльності отримує як моральну, так і матеріальну підтримку і зі сторони суспільства, і зі сторони держави.

Для реалізації властивого всім людям прагнення самозахисту існує віктимологічне спрямування попередження злочинів. Статті по типу «Як захистити себе, свою сім'ю, своє житло та свою власність від злочинних посягань» періодично з’являються у країнах Західної Європи та США. Реалізація програм для захисту потерпілих, а також свідків у кримінальних справах, створення асоціацій потерпілих від злочинів також позитивно впливають на дану ситуацію. У всіх навчальних закладах – від початкової школи до вузів – проводяться заняття з прикладної віктимології. Бажаючі можуть не лише прослухати лекції, але й пройти спеціальний тренінг: освоїти прийоми самозахисту та виробити навички оптимальної поведінки в екстремальних ситуаціях.

У США і країнах Західної Європи велика увага приділяється превентивних заходам щодо попередження у суспільстві. Наприклад, Адміністрація Президента США ухвалила федеральну програму морального оздоровлення американського суспільства, яка складається з декількох елементів.

1. Підтримка місцевих громад у запобіганні насильству.
2. Робота щодо припинення расової та культурно-національної ворожнечі.
3. Обмеження поширення зброї у суспільстві.

4. Стимулювання спокійного сімейного способу життя.

5. Проведення наукових досліджень для виявлення умов, у яких попередження злочинів здійснюється найуспішніше.

Висновки до розділу 1

Підсумовуючи проаналізовані дані, можна впевнено сказати, що ситуація у розкритті злочинів різного виду не є позитивною, навіть навпаки, згідно до статистики та опитувань звичайних громадян, вона тільки погіршується з плином часу: кількість нерозслідуваних діл збільшується, а більшість тих, що закриваються, просто не знаходять винуватих та припиняють розслідування через недостачу доказів. Але цю тенденцію можна змінити. Проблеми подібного характеру вже траплялися у деяких країнах, і там вони були або повністю, або частково вирішені, саме тому, на мою думку, найкращим підходом було б впровадження засобів попередження правопорушень схожих на ті, що використовуються у високорозвинених закордонних країнах. У більшості випадків вони потребують щільної співпраці між державою і суспільством.

2 СУЧАСНА РОЗРОБКА МОБІЛЬНИХ ЗАСТОСУНКІВ

2.1 Розробка застосунків для мобільних пристроїв Apple на мові програмування Swift

Swift – це просто надзвичайно прогресивна мова програмування, що дозволяє створювати програми для смартфонів, десктопних комп’ютерів, серверів та інших пристроїв, що входять у ряд тих, котрі розроблюються компанією Apple. Якщо підбирати якісь додаткові синоніми, то можна також зазначити, що Swift – це, безумовно, безпечна, швидка та інтерактивна мова програмування, на основі котрої працюють майже усі сучасні мобільні застосунки, створені для операційної системи iOS. Це не єдина інструмент для створення таких застосунків, але він є найсучаснішим, наповненим найкращими ідеями передових мов, а також тим, що увібрав у себе увесь попередній досвід та мудрість інженерної культури Apple. Його компілятор оптимізований таким чином, що розробник не повинен думати про компроміси з того чи іншого боку.

Однією з найголовніших переваг Swift є дружелюбність та відкритість по відношенню до новачків не лише у сфері мобільних застосунків, але й, в цілому, у програмуванні. Цей інструмент промислової якості є першим прикладом мови програмування, що не є скриптовою, але зберігає у собі зрозумілість та безумовно є захоплюючою для програміста. Можливість написання коду у пісочниці дає змогу побачити результат майже миттєво, і при користуванні цією можливістю потреби компілювати і запускати код просто немає.

Swift також дозволяє виключити величезну кількість поширених серед інших мов програмування помилок через те, що у його набір входять сучасні програмні паттерни, такі як.

1. Усі змінні ініціалізовані ще до того, як вони використовуються.
2. Індекси масивів завжди автоматично перевіряються на out-of-bounds помилки.
3. Цілі числа завжди проходять перевірку на переповнення.

4. Значення nil завжди будуть гарантовано опрацьовані завдяки опціоналам.

5. Автоматичне управління пам’яттю.

6. Обробка помилок дозволяє здійснювати контрольоване поновлення від непередбачених помилок.

Для отримання максимальної віддачі від сучасних машин код на Swift компілюється та оптимізується. Головне правило, котрого дотримувались програмісти, що створювали керівництво щодо синтаксису та стандартної бібліотеки, полягало у тому, що найочевидніший і найпростіший спосіб написання коду є найкращим варіантом. Швидкість роботи цієї мови програмування та безпека середовища дозволяють створювати як найпростіші застосунки рівня "Hello, World!", так і справжні production інструменти, що своєю складністю досягають рівня операційних систем.

У Swift сучасні ідеї можуть бути виражені просто та коротко завдяки тому, що він поєднує у собі виведення типів та pattern-matching із простим синтаксисом. І у якості результату код стає не лише легше писати, але й читати його, навість якщо він був написаний іншим програмістом. Така комбінаціє йде у користь як новачку, що лише починає своє знайомство із цією мовою програмування, так і досвідченому спеціалісту, котрий використовує Swift уже багато років.

Хоч Swift і відносно нова мова програмування, але за її плечима вже маються роки розвитку та прогрес спостерігається і зараз, адже нові можливості з’являються і по сьогодення. Xcode 13 за замовчуванням підтримує Swift 5.5. Тринадцяту версію найпопулярнішого редактору коду можна використовувати для складання таргетів, написаних як на Swift 5.5, Swift 4.2, так і на Swift 4. Використовуючи Xcode 13 для того, щоб зібрати проект написаний на Swift 4 або Swift 4.2, більшість функціоналу Swift 5.5 також буде доступна. Єдиний унікальний функціонал коду, написаного на Swift 5.5 включає у себе.

1. Функції, які повертають непрозорі типи, потребують версії Swift 5.1.

2. Вираз try? не додає додаткового рівня опціональності, якщо вираз є опціоналом.

3. Ініціалізація через великі цілі літерали тепер повертає коректний цілий тип. Наприклад, UInt64(0xffff_ffff_ffff_ffff) тепер визначається вірно, а раніше він викликав переповнення.

Для реалізації паралелізму потрібна Swift 5.5 або ще більш сучасна версія цієї мови програмування, а також потрібна відповідна версія стандартної бібліотеки Swift, котрою надаються типи для реалізації паралелізму. Для можливості використання такого функціоналу треба встановлювати таргетовану версію операційної системи, для котрої ведеться розробка, як мінімум watchOS 8.0 для розумних годинників, macOS 12 для ноутбуків Apple, iOS 15 для iPhone, а також tvOS 15 для телевізорів, що використовують приставку Apple.

За традицією програмістів, перша програма при вивченні нової мови програмування просто повинна виводити на екран, а найчастіше у консоль, фразу «Hello, world». На Swift така проста програма буде виглядати наступним чином (рис. 2.1).

```
1   print("Hello, world!")
```

Рисунок 2.1 – «Hello, world» програма на Swift

Цей синтаксис є знайомим для будь-кого, хто раніше вивчав код на C або Objective-C. На Swift дана команда є повноцінною та після її компілювання розробник вже зможе побачити бажаний текст у консолі. Немає потреби у імпорті якихось бібліотек чи сторонніх пакетів для того, щоб мати змогу вводити або виводити дані у консоль. Потреби у функції main() також немає у даній програмування, адже код, що був написаний у глобальній області, одразу використовується як вхідна точка для програми. Як можна помітити, крапки з комою після кожного рядка також не є потребою у Swift.

У Swift треба використовувати let для створення констант та var для оголошення змінних. Значення константи не обов'язково повинно бути встановлене

на момент компіляції, але воно має привласнюватися строго один раз. Це означає, що використовувати константу треба для позначення змінних, що будуть визначатись лише один раз, не будуть потім перезаписані, але використовуватися можуть, звісно, у багатьох місцях.

```
1   var simpleVariable = 10
2   simpleVariable = 20
3   let simpleContant = 999
```

Рисунок 2.2 – Приклад використання змінних та констант

Явно оголошувати тип змінної чи константи не потрібно завжди, але вони повинні мати такі ж типи даних, які розробник хоче їм привласнити. Якщо створювати змінну чи константу без явного встановлення її типу, компілятор візьме цю частину роботи на себе та автоматично передбачить її тип. У прикладі на рис. 2 компілятор автоматично встановив integer (ціле число) для simpleVariable, адже надане йому значення було цілим числом.

Якщо ж значення ще не присвоєно, або ж воно не дає достатньої інформації, тип треба вказати після назви змінної чи константи, розділеної за допомогою двокрапки.

```
1   let implicitIntegerConst = 100
2   let implicitDoubleConst = 100.0
3   let explicitDoubleConst: Double = 100
```

Рисунок 2.3 – Приклад явного вказання типу константи

Значення ніколи не повинні конвертуватися в інший тип. Для обробки даного прикладу треба конвертувати значення в інший тип, а вже після цього створити екземпляр потрібного типу.

```
1   let ageMessage = "My age is "
2   let ageValue = 94
3   let fullMessage = ageMessage + String(ageValue)
```

Рисунок 2.4 – Приклад конвертування в інший тип

Так як потреба помістити значення в строку виникає досить часто, розробники Swift передбачили ще більш зручний синтаксис. Для цього треба

записати значення у дужках і поставити перед ними зворотний слеш "\", як показано на рис. 2.5.

```
let bananasValue = 3
let strawberriesValue = 5
let bananasMessage = "I have \(bananasValue) bananas."
let finalMessage = "I have \(bananasValue + strawberriesValue) pieces of fruit."
```

Рисунок 2.5 – Скорочений синтаксис для поміщення значення у строку

Для змінних, що займають декілька рядків, треба використовувати три подвійні лапки. Відступи кожного окремого рядка повинні збігатися з відступом закриваючих лапок.

```
let exampleMessage = """
Full line.
Another line.
One more line

I still have \(bananasValue + strawberriesValue) pieces of fruit.
"""
```

Рисунок 2.6 – Константа з декількома рядками

Для створення масивів та словників у мові програмування Swift передбачений синтаксис подвійних квадратних дужок, а для того, щоб отримати доступ до певного елементу по його індексу, треба вказати сам індекс або ж ключ у квадратних дужках. Якщо поставити кому після останнього елементу, то це теж не спровокую помилку, дозволяються обидва варіанти.

```
var shoppingList = ["apple", "banana", "strawberry"]
shoppingList[1] = "cherry"

var programmerLevels = [
    "Junior": "1 year of experience",
    "Middle": "1-3 years of experience",
    "Senior": "over 3 of experience",
]
programmerLevels["Senior"] = "It depends..."
```

Рисунок 2.7 – Звернення до конкретного елементу масиву чи словника

Масив автоматично збільшується при додавання елементу (рис. 2.8), а для створення пустого масиву треба використати вираз ініціалізації (рис. 2.9).

```
11   shoppingList.append("pineapple")
12   print(shoppingList)
```

Рисунок 2.8 – Збільшення розміру масиву при додаванні нового елементу

```
14   let emptyArray = [String]()
15   let emptyDictionary = [String: Float]()
```

Рисунок 2.9 – Створення пустого масиву та словника

Також у Swift є цікавий синтаксис для випадків, коли програміст бажає, щоб інформація про тип змінної або константи, що використовується у масиві або словнику, має бути передбачена компілятором. Для цього треба створити пустий масив за допомогою двох квадратних дужок, а також пустий словник за допомогою квадратних дужок та двокрапки між ними (рис. 2.10).

```
17   shoppingList = []
18   programmerLevels = [:]
```

Рисунок 2.10 – Створення масиву або словника з контентом, тип якого буде передбачений під час компіляції

Для того, щоб створити функцію у мові програмування Swift, треба використати оператор func. Викликається функція дуже лаконічно: для цього всього лише треба вказати її ім’я з дужками та передати у них список аргументів. Для відокремлення імен та типів аргументів треба від типу функції треба використати «->» (рис. 2.11).

```
1   func greet(name: String, day: String) -> String {
2       return "Hello \(name), the day is \(day)."
3   }
4   greet(name: "Karen", day: "Monday")
```

Рисунок 2.11 – Створення і використання функції

За замовчуванням функції використовують імена параметрів як ярлики їх аргументів. Можна написати свій власний ярлик до імені параметра функції, або ж вказати знак підкреслення «_», щоб пропустити ярлик.

```swift
func greet(_ name: String, on day: String) -> String {
    return "Hello \(name), the day is \(day)."
}
greet("Karen", on: "Monday")
```

Рисунок 2.12 – Робота з імена параметрів

Для створення єдиного складового значення потрібно використовувати кортеж. Це може знадобитися, наприклад, для повернення кількох значень з функції. Для посилання на елемент кортежу можна використовувати як імена, так і порядкові номери.

```swift
func calcStat(scores: [Int]) -> (min: Int, max: Int, sum: Int) {
    var min = scores[0]
    var max = scores[0]
    var sum = 0

    for score in scores {
        if score > max {
            max = score
        } else if score < min {
            min = score
        }
        sum += score
    }

    return (min, max, sum)
}

let stats = calcStat(scores: [5, 3, 100, 3, 9])

print(stats) // (min: 3, max: 100, sum: 120)
print(stats.sum) // 120
print(stats.2) // 120
```

Рисунок 2.13 – Використання функції, що повертає кортеж

Swift також дає змогу створювати та використовувати вкладені функції. Вони мають доступ до змінних, оголошених у зовнішній функції. Зазвичай такі функції використовують для того, щоб упорядкувати код у довгих або складних функціях.

```swift
func returnFifteen() -> Int {
    var y = 10
    func add() {
        y += 5
    }
    add()
    return y
}
returnFifteen()
```

Рисунок 2.14 – Використання вкладених функцій

Результатом функції може бути інша функція, адже функції – це об’єкти першого класу.

```swift
func makeIncrementer() -> ((Int) -> Int) {
    func addOne(number: Int) -> Int {
        return 1 + number
    }
    return addOne
}
var increment = makeIncrementer()
increment(7)
```

Рисунок 2.15 – Приклад функції, що повертає іншу функцію

Ще однією цікавою особливістю, реалізованою у мові програмування Swift, є те, що функція може приймати іншу функцію у якості аргументу.

```swift
func hasAnyMatches(list: [Int], condition: (Int) -> Bool) -> Bool {
    for item in list {
        if condition(item) {
            return true
        }
    }
    return false
}
func lessThanTen(number: Int) -> Bool {
    return number < 10
}
var numbers = [20, 19, 7, 12]
hasAnyMatches(list: numbers, condition: lessThanTen)
```

Рисунок 2.16 – Приклад функції у якості аргументу

Замикання – це блок коду, котрий може бути викликаний пізніше. Функції, у свою чергу, - це окремий випадок замикань. Код усередині замикань має доступ до таких об'єктів, як змінні та функції, котрі були створені в тих же рамках, що й самі замикання. Замикання можна писати навіть без імені, для цього потрібно позначити код фігурними та круглими дужками наступним чином «({})». Для розмежування аргументів і типу, що повертається від тіла замикання, треба використовувати in.

```
15  numbers.map({ (number: Int) -> Int in
16      let result = 3 * number
17      return result
18  })
```

Рисунок 2.17 – Використання замикання у функції

Для того, що написати замикання коротше, Swift надає декілька можливостей:

1. Якщо тип замикання точно відомий, можна пропустити тип його аргументів, тип значення, що повертається, або навість і те, й інше. Цей випадок скороченого синтаксису буде корисний при використанні зворотного делегату, іншими словами callback.

```
20  let mappedNumbers = numbers.map({ number in 3 * number })
21  print(mappedNumbers)
```

Рисунок 2.18 – Перший варіант укороченого синтаксису для замикання

2. До аргументів можна звертатися не лише за іменем, а й за їх порядковим номером. Зазвичай даний підхід використовують у дуже коротких замиканнях. Замикання, що було передано у якості останнього аргументу, може з'явитися безпосередньо після дужок.

```
23  let sortedNumbers = numbers.sorted { $0 > $1 }
24  print(sortedNumbers)
```

Рисунок 2.19 – Другий варіант укороченого синтаксису для замикання

2.2 Розробка застосунків для мобільних пристроїв на операційній системі Android на мові програмування Kotlin

До 2017 року розробники мобільних застосунків для Android були просто зобов’язані використовувати мову програмування Java 6 з підтримкою наступних доповнень для створення програм. Java 6 була представлена ще в далекому 2006 році. Для порівняння слід зазначити, що операційна система Android вийшла лише через два роки після цього – у 2008.

Розробники, що використовують мову програмування Java, навіть ті, що створюють не мобільні застосунки, давно використовують IntelliJ IDEA, що була розроблена компанією JetBrains. Ця IDE за довгі роки свого існування набула неймовірної популярності та користується довірою як новачків, так і досвідчених програмістів, що багато років користуються цим інструментом та навіть не задумуються про його аналоги. У 2011 JetBrains представила нову мову програмування Kotlin, але тоді ця новинка була досить холодно зустріта фахівцями у сфері програмування, досить невелика кількість людей змогла по-справжньому оцінити новий продукт. Kotlin хоч був уже готовий до виробництва, але він не був стабільним, особливо у порівнянні із Java, котра на той момент стала стандартом для мобільних застосунків для операційної системи Android. Через п’ять ситуація різко змінилась, адже компанія JetBrains оновила Kotlin до версії 1.0. Це була дуже сильна зміна в більшості фундаментальних структур даної мови програмування, розробникам навість довелося змінити свою кодову базу.

На Google I/O 2017 компанія Google оголосила, що відтепер Android буде підтримувати Kotlin як першокласну мову програмування для розробки мобільних застосунків. У той же момент стався і випуск Android Studio 3.0, головною новинкою в котрій була підтримка Kotlin з коробки. Наступні мінорні оновлення Android Studio ще більше покращували взаємодію IDE з мовою програмування Kotlin.

Kotlin — це сучасна строго типізована мова програмування, яка працює на віртуальній машині Java (JVM) шляхом компіляції коду Kotlin у байт-код Java. Однією з особливостей цієї мови програмування є те, що її можна скомпілювати до вихідного коду JavaScript та до нативних виконуваних файлів. Kotlin є дуже гнучким і має величезну кількість цікавих функцій.

У момент, коли Android просто штурмом захопив світ, у розробників не було серйозного вибору серед мов програмування. Звісно, Java навіть тоді не була єдиною мовою, котра давала змогу створювати програми для мобільних пристроїв Android, але вона стала таким стандартом, що починати створювати мобільний застосунок не на Java було просто несерйозно. Вона була мовою програмування за допомогою котрої найдосконаліші телефони запускали своїх рідні програми на своїх же власних операційних системах. Наприклад, операційна система Symbian, що була створена компанією Nokia лише для телефонів своєї фірми, мала програми Java ME. Головна проблема Java на той час полягала у тому, що вона мала в собі багато історичного багажу, котрий було просто недоцільно або й навіть неможливо використовувати при розробці для мобільних пристроїв.

Із виходом Java 8 певна кількість проблем цієї мови програмування у контексті розробки для мобільних застосунків була вирішена, і ця тенденція стала ще більш помітною у версіях Java 9 і 10. На жаль, розробнику потрібно встановити мінімальний пакет SDK на Android 24, щоб використовувати всі функції Java 8, аце не є варіантом для абсолютної більшості. Фрагментація екосистеми Android не дає можливості залишити користувачів зі старими пристроями, тому для величезної кількості розробників використання Java 9 і 10 так і залишились мрією.

Kotlin — це сучасне рішення, одна з найтрендовіших мов програмування у цілому та точно найкращий інструмент для програмістів, що створюють програми для операційної системи Android. Є кілька речей, завдяки яким Kotlin чудово підходить для Android.

1. Сумісність: Kotlin сумісний з JDK 6, тому старі пристрої не залишаються без підтримки.

2. Продуктивність на більш високому рівні, ніж у Java.

3. Сумісність: Kotlin на 100% сумісний з Java, включаючи анотації.

4. Мала за розміром runtime бібліотека.

5. Час компіляції: є невеликі витрати на чисті збірки, але це набагато швидше з інкрементальними збірками.

6. Вивчати Kotlin дуже легко, особливо для людей, які звикли до сучасних мов програмування. Конвертер Java в Kotlin в IntelliJ і Android Studio робить це ще простіше.

7. Програміст також може використовувати поєднання Kotlin і Java у справжньому проєкті, тому поспішати вивчати Kotlin і повністю на нього переходити немає сенсу – це можна роботи паралельно із розробкою на Java.

Створювати змінні у Kotlin неймовірно легко і інтуїтивно зрозуміло. На рис. 2. 20 можна побачити найпоширеніший синтаксис для створення змінних.

```
var item = "house"
val points = 35

var car: String = "BMW"
```

Рисунок 2.20 – Створення змінних у Kotlin

Яка різниця між val і var? Змінні, що були оголошені за допомогою ключового слова val у подальшому не можуть бути перезаписані, спроба це зробити одразу спровокує помилку компілятора. Фактично, такі змінні є константами. Змінні ж, що були створені за допомогою ключового слова var можуть спокійно бути перезаписані без ніяких помилок.

```
points = 36
```

Рисунок 2.21 – Перезапис значення змінної val, що призведе до помилки

У випадку, що показаний вище (рис. 2.21) буде одразу помилка, адже ми намагаємось записати нове значення у константу. Цей спосіб оголошення змінних є аналогічним до оголошення за допомогою ключового слова final у Java.

Я і у випадку з мовою програмування Swift, у більшості випадків немає необхідності одразу присвоювати і тип змінної при її оголошенні. У Kotlin автоматичне визначення типу працює досить добре. Саме тому в більшості випадків явно оголошувати тип змінної не треба. Якщо казати про випадки, коли таку маніпуляцію все ж потрібно буде виконати, то слід зазначити, що найпоширенішим таким прикладом є випадок, коли мова йде про необов'язковий тип, який може бути нульовим. Для наведеної у рис. 2.20 змінної car оголошення її типу не є обов'язковим, але, якщо це все ж зробити, як у нашому випадку, де ми явно вказали, що в змінній, котра зараз нічим не заповнена, може знадитись лише string, компілятор помилку не покаже.

```
var highScore = points + 999
```

Рисунок 2.22 – Створення змінної з неявним вказанням типу

На рис. 2.22 показано приклад, де оголошується змінна без явного оголення її типу, але Kotlin автоматично встановить їй integer, через значення, котре їй було присвоєне.

Навіть у деяких сучасних мовах програмування ще досі існує проблема з доступом до null змінних. Нульове посилання виникає, коли програміст оголошуєте змінну об'єкта без надання їй значення, або коли він призначає цій змінній значення null. У такому випадку програма при спробі отримати доступ до цієї змінної не зможе цього зробити, тому що вона просто не знає, де є її шукати. Найпоширенішим результатом виконання такого коду є раптова зупинка програми з подальшим повідомленням про збій. Мабуть, кожна людина, котра користувалась мобільним телефоном на операційній системі Android ще до виходу Kotlin зіштовхувалась з подібним повідомленням (рис. 2.23). Це і є NullPointerException, котрий ніяк не опрацьовується автоматично у Java.

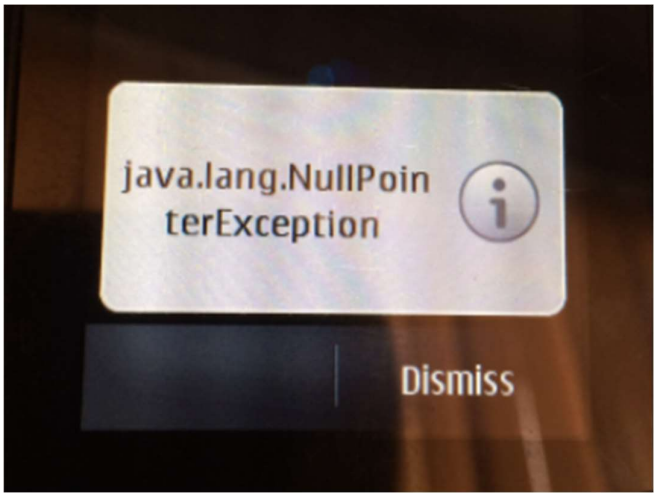

Рисунок 2.23 – NullPointerException у застосунку, створеному на Java

Система типів Kotlin має на меті усунути NullPointerException, ця ціль відома під назвою void safety [6]. У Kotlin є лише декілька причин для NullPointerException, серед них.

1. Зовнішній код Java.
2. Явний виклик NullPointerException.
3. Використання оператору «!!».
4. Змінна lateinit не ініціалізована перед доступом.

Така невелика кількість причин цієї досить розповсюдженої проблеми дає можливість програмісту дуже швидко зорієнтуватись у тому, де саме виникла така помилка, та виправити її.

У Kotlin можна створювати змінні, що допускають у собі збереження null та такі, що забороняють це робити. Якщо програміст оголосив змінну як не нульову, то він не матиме змоги записати у неї null. Компілятор забезпечує неможливість подібної помилки, ускладнюючи ненавмисне аварійне завершення роботи програми.

Ще однією відмінністю Kotlin від Java є те, що у Kotlin абсолютно всі змінні мають бути ініціалізовані в точці оголошення, в конструкторі або в init. Винятком з цього правила є лише lateinit var.

Щоб оголосити змінну нульовою, потрібно додати ? до його типу в точці оголошення, як ви бачите в цій декларації shareActionProvider.

```
private var shareActionProvider: ShareActionProvider? = null
```

Рисунок 2.24 – Створення nullable змінної

Ще однією неймовірно цікавою фішкою Kotlin є late initialization variables або ж змінні пізньої ініціалізації, якщо переводити це на українську мову. Фактично, створення таких змінних – це обіцянка розробника, що він не забуде ініціалізувати змінну перед тим, як отримати до неї доступ. За іронією це і є найпоширенішою причиною NullPointerException у Kotlin, але все ж при виникненні подібної помилки програміст в першу чергу перевірить усі випадки взаємодії зі змінними даного типу, що дозволить дуже швидко знайти місце помили, як це вже зазначалося раніше. Для оголошення такого виду змінною достатньо лише додати lateinit у конструкторі або блоці ініціалізації.

```
class SomeClass {
  private lateinit var myName: String

  fun methodThatAccesses() {
    myName.trim()
  }

  fun methodThatAssigns() {
    myName = "initialize the variables!"
  }
}
```

Рисунок 2.25 – Оголошення змінної без ініціалізації

У рис. 2.25 можна побачити приклад оголошення змінної без ініціалізації myName. У даному класі, окрім змінної, є також два методи: methodThatAccesses і methodThatAssigns. Якщо спочатку викликати methodThatAccesses, то програма покаже помилку, адже ми оголосили змінну без ініціалізації, забули її ініціалізувати і звертаємося до ней – в результаті ми отримаємо NullPointerException. Для уникнення помилки у ситуаціях роботи зі змінними

подібного типу треба на забувати ініціалізувати їх до того, як ми звернемося до них. У даному випадку (рис. 2.25) треба спочатку викликати methodThatAssigns для ініціалізації, а потім вже methodThatAccesses для звернення до змінної, що нам буде потрібна.

Найчастіше потреба у змінній без ініціалізації виникає у випадках, коли програмісту потрібно створити змінну, котру він не може або не хоче ініціалізувати у конструкторі або в блоці ініціалізації. У Android представлення призначаються методом onCreate(). Якщо виникає така ситуація, коли програмісту потрібно, щоб одне з представлень було членом класу, і він не хоче повсюдно мати справу з можливим null, має сенс оголосити це представлення lateinit.

У Java щоб отримати доступ до методу чи властивості для змінної, яка допускає значення null, спочатку потрібно виконати перевірку на ненульове значення (рис. 2.26).

```
if (shareActionProvider != null) {
  shareActionProvider.setShareIntent(shareIntent);
}
```

Рисунок 2.26 – Перевірка на ненульове значення у Java

Kotlin в свою чергу дозволяє значно спростити дану конструкцію, завдяки новому синтаксису, що спеціально був розроблений для таких випадків – це оператор безпечного виклику «?.». У такому випадку властивість або метод викликається лише тоді, коли змінна, яка допускає значення null, не є нульовою. На рис. 2.27 setShareIntent буде викликано лише у тому випадку, коли властивість shareActionProvider не містить у собі значення null.

```
shareActionProvider?.setShareIntent(shareIntent)
```

Рисунок 2.27 – Доступ до методу через змінну, яка допускає значення null, у Kotlin

Оператор «!!» також є однією з можливих причин страшної помилки NullPointerException. Цей оператор має сенс використовувати коді, коли програміст

впевнений у тому, що посилання, що допускає значення null, не є нульовим (рис. 2.28).

```
shareActionProvider = MenuItemCompat.getActionProvider(shareItem!!) as
ShareActionProvider
```

Рисунок 2.28 – Використання оператору «!!»

На рис. 2.28 буде помилка NullPointerException лише у тому випадку, коли змінна shareItem має значення null.

Ще однією особливістю Kotlin у порівнянні з Java є оператор Елвіса «?:». Хоч він і виглядає як тернарний оператор у Java, але працює інакше. Даний оператор отримав таку цікаву назву через те, що він нагадує волосся Елвіса Преслі, якщо нахилити голову на 90 градусів вліво.

```
val len = coverId?.length ?: 0
```

Рисунок 2.29 – Використання оператору Елвіса

Якщо вираз зліва від оператора Елвіса не є нульовим значення, то він повертає результат виразу. В іншому випадку він повертає вираз праворуч. У даного оператора є оптимізація, подібно оператору if-else, тобто він оцінює вираз праворуч, лише якщо вираз зліва дорівнює нулю.

```
val len: Int
if(coverId?.length != null) {
  len = coverId?.length
} else {
  len = 0
}
```

Рисунок 2.30 – Еквівалент оператору Елвіса у Java

Додатковою особливістю Kotlin є підтримка type inference. Це означає, що компілятор може виводити типи змінних з ініціалізатора. Але паралельно із цим програміст завжди може оголосити тип, якщо йому закортить, або ж у випадку, коли компілятор не зможе впоратись із цією задачею. На рис. 2.31 можна побачити, що типи змінних imageUrlBase і imageURL виводяться з їх ініціалізаторів.

```
private val imageUrlBase = "http://covers.openlibrary.org/b/id/"
private var imageURL = ""
```

Рисунок 2.31 – Приклад type inference

Компілятор слідкує за виведеним типом кожної змінної, кожної у якості string, тобто усі значення, присвоєні змінній після цього, також повинні мати строковий тип. Для чисел програміст має зробити тип змінної таким, щоб компілятор зрозумів його (рис. 2.32).

```
var longVar = 0L
var floatVar = 0.0f
var doubleVar = 0.0
var integerVar = 0
```

Рисунок 2.32 – Типи змінних для чисел

2.3 React Native – трендовий мультиплатформний фреймворк, реалізований на мові програмування JavaScript

React Native — це популярний фреймворк для створення мобільних застосунків на основі JavaScript, який дозволяє створювати програми для iOS та Android із нативною формою. Фреймворк дає змогу вести розробку одночасно для різних платформ, використовуючи ту саму кодову базу.

React Native був вперше випущений Facebook як проект з відкритим кодом у 2015 році. Успіху та популярності він досяг за досить короткий період, а через декілька років його вже називали одним із найкращих рішень, що використовуються для мобільної розробки. Розробка на React Native використовується для створення як невеликих мобільних застосунків, так і для величезних програм, аудиторія котрих доходить до десятків та сотень мільйонів користувачів.

Існує кілька причин глобального успіху React Native.

1. Головною причиною популярності та розповсюдження React Native є звісно ж те, що компаніям вигідно як у плані часу, так і плані ресурсів тримати

лише одну команду, що відповідатиме за розробку мобільного застосунку для усіх платформ, ніж забезпечувати дві повноцінні команди, що будуть паралельно розробляти фактично одну й ту ж програму, але з різною кодовою базою через особливості кожної платформи.

2. RN побудований на основі дуже популярного веб-фреймворка React, що за результатами опитування JetBrains [8] використовують майже 50% front-end розробників. Враховуючи кількість конкурентів, така неймовірна популярність може свідчити лише про те, що даний фреймворк майже повністю покриває усі потреби сучасного front-end розробника. Це теж додало додаткові бали до кошика причин популярності React Native.

3. React Native дав змогу величезній кількості front-end розробників почати створювати не тільки веб-застосунки, а й мобільні програми, котрі запускаються як на Android-смартфонах, так і на iPhone.

Коли компанія Facebook вперше вирішила зробити свій сервіс доступним на мобільних пристроях, розробники зробили ставку на мобільну веб-сторінку на основі HTML5 замість того, щоб створювати нативний додаток, як у багатьох провідних технологічних гравців того часу. Як показав час, дане рішення зовсім не виправдало очікувань, залишивши багато місця для покращення інтерфейсу та продуктивності. Більше того, вже у 2012 році Марк Цукерберг визнав, що «найбільшою помилкою, яку ми зробили як компанія, було занадто багато ставок на HTML, а не на native».

Уже через рік після признання головою Facebook невдачі їх експериментів з мобільними застосунками один із передових програмістів з цієї ж компанії по імені Джордан Волке зробив просто неймовірне на той час – він вперше зміг згенерувати нативний елемент користувацького інтерфейсу для мобільного застосунку на iPhone за допомогою JavaScript. Таке новаторське відкриття, звісно ж, спричинило справжню пожежу у IT спільноті. Програмісти з усього світу, що були зацікавлені у розробці на мові програмування JavaScript, котра до цього використовувалась лише у веб-браузерах, мобільних застосунків, почали дуже активно вивчати цю

тему. Майже одразу після цього відкриття був анонсований та незабаром проведений спеціальний хакатон, головною ціллю котрого було з’ясувати, скільки мобільних розробок можна зробити за допомогою (поки що традиційно веб-базованих) рішень JavaScript.

Саме таким чином одна інноваційна ідея та конкретна реалізація Джордана Волке назавжди перевернула світ розробки мобільних застосунків раз і назавжди. З того часу компанія Facebook і почала розробляти React Native, котрий з фреймворку, підтримуючого лише iOS швидко перетворився на інструмент для обох операційних мобільних систем. Фреймворк став відкритим вже у 2015 році і одразу підтримував кросплатформність. Всього через три роки після свого випуску React Native вже став другим за величиною проектом на GitHub, якщо дивитися на кількість учасників.

React vs. React Native. Якщо використовувати найпростіші слова, то можна зазначити, що React Native не є «новішою» версією React, але RN все ж її використовує.

React (також відомий як ReactJS) — це бібліотека JavaScript, яка використовується для створення інтерфейсу веб-застосунку. Як і React Native, його також розробила команда інженерів Facebook.

Тим часом React Native, який працює на базі React, дозволяє розробникам використовувати набір компонентів інтерфейсу користувача для швидкої компіляції та запуску програм iOS і Android.

У React і React Native використовується багато схожих підходів, серед яких є JavaScript для написання логіки застосунку, а також спеціальна мова розмітки JSX.

```
const App = () => {
   return (
     <div>
       <p>Header</p>
       <p>Content</p>
       <p>Footer</p>
     </div>
   );
}
```

Рисунок 2.33 – Приклад JSX коду

JSX означає JavaScript XML. Це просто синтаксичне розширення JavaScript. Це дозволяє розробникам безпосередньо писати HTML у React (у межах коду JavaScript). Створити шаблон за допомогою JSX у React легко, але це не проста мова шаблонів, а натомість вона поставляється з повною потужністю JavaScript. JSX виконує свою роботу швидше, ніж звичайний JavaScript, оскільки він виконує оптимізацію під час перекладу на звичайний JavaScript.

Оскільки React – це веб-бібліотека, тому вона і використовує HTML і CSS, а React Native у свою чергу використовує спеціальні компоненти, котрі були створені розробниками RN. Ці спеціальні компоненти у ході компіляції застосунку перетворюються на нативні елементи тієї операційної системи, на котрій застосунок запускається.

Приклад коду з обговорення Stack Overflow, котрий найкраще демонструє найголовніші стандартні розбіжності React і React Native: «React JSX renders HTML-like components like <h1>, <p>, etc. [Meanwhile] react-native renders native app view components like <View> , <Text>, <Image>, <ScrollView>, so you can't directly reuse your UI component code unless you rework/replace all the elements.».

Кросплатформна розробка – це практика створення програмного забезпечення, сумісного з більш ніж одним типом апаратної платформи. Кросплатформність вважається майже святим Граалем розробки програмного забезпечення, це обумовлюється тим, що програміст має змогу створити кодову базу один раз, а потім запускати її на тій платформі, на котрій забажає, на відміну від програмного забезпечення, створеного для певної платформи. Розробники можуть використовувати інструменти, якими вони володіють, як-от JavaScript або C#, для розробки під ті платформи, які їм потрібні. Кросплатформна розробка цікава не лише програмістам, а й замовникам програмних застосунків, адже вони також економлять на цьому і ресурси, і час. У кросплатформної розробки є декілька характеристик.

1. Більш широка аудиторія. Ще на стадії проектування бізнесу та розробки моделі розповсюдження програми відпадає потреба орієнтуватись лише

на одну аудиторію, наприклад, лише на користувачів iOS або ж тільки на Android юзерів, оскільки кросплатформне програмне забезпечення забезпечить потреби обох груп користувачів, тобто запуститься з одним кодом на обох операційних системах. Саме тому це і дає доступ до більш широкої бази користувачів.

2. Спеціальні стильові розбіжності під окрему платформу. У мобільних застосунків, створених під мобільні телефони iOS та Android, є відмінності у навігації та загалом у дизайні. Це може стати проблемою при паралельному створенні двох програм двома різними командами розробників, що використовують різні мови програмування для своїх цілей. При кросплатформній розробці такі проблеми нівелюються завдяки єдиній кодовій базі. Звісно, для створення послідовної ідентичності бренду на різних платформах розробникам треба навіть кросплатформних застосунків все одно доведеться про це піклуватись, але якісний результат вимагатиме набагато менше зусиль.

3. Код багаторазово використання. Як вже зазначалося раніше, саме можливість мати одну кодову базу як для iOS застосунку, так і для його аналогу на операційній системі Android є неймовірно потужним та найголовнішим бонусом при кросплатформній розробці.

4. Швидша розробка. З попередньої характеристики про єдину кодову базу для мобільного застосунку для операційних систем iOS та Android витікає ще одна риса – це швидкість розробки. Оскільки кодова база у кросплатформних застосунках завжди одна, тобто весь код знаходиться у одному місці, то і пошук причин багів, котрі обов’язково будуть з’являтися з плином часом, а також їх подальше усунення буде проходити набагато швидше. Те саме можна сказати і про імплементацію нових функцій програми. Це досягається завдяки тому, що кросплатформні програми створюються як єдині проекти, а отже весь код може бути використаний повторно, що є однією з найкращих практик не лише для кросплатформних мобільних застосунків, написаних на React Native, а взагалі для будь-якої програми у світі IT.

5. Зниження витрат на розробку кінцевого продукту. Створення кросплатформних застосунків може бути на 30% дешевше, ніж створення нативних програм під кожну мобільну операційну систему, це досягається завдяки можливості використання єдиного коду.

Проаналізувавши всі позитивні сторони кросплатформної розробки можна швидко задатися питанням: «Чому досі існує величезна кількість програмістів, котрі створюють нативні програми під кожну окрему платформу?». Відповідь на це питання з’являється одразу після ознайомлення з недоліками кросплатформної розробки.

1. Для забезпечення високої продуктивності потрібні більш досвідчені програмісти. У IT спільності існує міф, що кросплатформні застосунки працюють не так швидко, як їх нативні аналоги. Насправді, це твердження не є вірним, наприклад, і Flutter, що є ще одним прикладом фреймворку для кросплатформної мобільної розробки і створений на базі мови програмування Dart, і React Native мають на меті працювати зі швидкістю 60 кадрів у секунду. У абсолютній більшості випадків кросплатформні застосунки дійсно можуть працювати за тими ж стандартами, що і нативні програми, але для забезпечення такого рівня продуктивності команда програмістів, що створюють мобільний застосунок повинна мати достатньо навичок та досвіду, тобто дана задача насправді не є дуже легкою.

2. Більш складний код-дизайн. Як вже зазначалося раніше, мобільні застосунки, створені для операційних систем iOS та Android, мають різнитися між собою у деяких моментах, більшість з котрих стосується спеціальних стилів, що використовуються лише на одній с платформ. Це в свою чергу означає, що кросплатформні застосунки мають реагувати на різні пристрої та платформи, де вони запускаються, і це певною мірою ускладнює кодування. Реалізація такої поведінки призводить до більшої роботи, що повинен виконати розробник, котрий просто зобов’язаний мати на увазі цю особливість та завжди враховувати те, що застосунок може бути запущений на будь-якій з платформ.

3. Більш тривалий час випуск нового функціоналу. З випуском нових функцій для Android та iOS потрібен деякий час, щоб оновити обидві програми для підтримки новітніх функцій. Це є недоліком саме кросплатформних застосунків, адже нативні програми оновлюються помітно швидше.

Як працює React Native? Як зазначалося раніше, React Native створений із сумішшю JavaScript та JSX, що є спеціальним кодом розмітки, котрий фактично дозволяє писати HTML розмітку прямо у JavaScript. RN має можливість одночасно спілкуватися як JavaScript, так і з JSX завдяки двом своїм потокам.

Цей фреймворк одночасно взаємодіє з потоками на основі JavaScript та з існуючими потоками нативних програм. Для налагодження такого виду спілкування React Native використовує так званий «міст» (bridge). Незважаючи на те, що JavaScript і Native написані з використанням зовсім різних мов програмування, міст робить можливим двостороннє спілкування. Далі наведено візуалізацію концепції мосту (рис. 2.34)

Рисунок 2.34 – Візуалізація концепції моста

Це означає, що програміст, котрий вже має готовий нативний мобільний застосунок, створений на мові програмування Swift або Kotlin, все ще має можливість використовувати його компоненти або ж повністю перейти до розробки на React Native.

React Native сильно відрізняється від своїх кросплатформних конкурентів у лиці Cordova, PhoneGap і їм подібних тим, що RN не використовує WebView [9] у

своєму коді. Він працює на основі фактичних, рідких представлень і компонент. Це одна з найголовніших причин вражаючого успіху React Native.

На React Native уже створено просто безліч мобільних застосунків, але у приклад можна привести декілька найбільших та найпопулярніших мобільних програм [10] просто для того, щоб зрозуміти, з якими задачами може справлятися цей чудовий фреймворк.

1. Facebook є однозначно найпопулярнішим мобільним застосунком, під час розробки котрого використовувався React Native. Ця програма посідає друге місце у топі найпопулярніших мобільних застосунків [11]. Саме розробники з Facebook створили цей фреймворк і досі займаються його розвитком, тому не дивно бачити саме цей мобільний застосунок у якості головного рекламного ходу RN. Facebook мали мету передати всі переваги веб-розробки для мобільних пристроїв, серед яких є швидкі ітерації та наявність єдиної команди розробників продукту. Ads Manager для iOS та Android був розроблений компанією, а обидві версії були створені однією командою розробників.

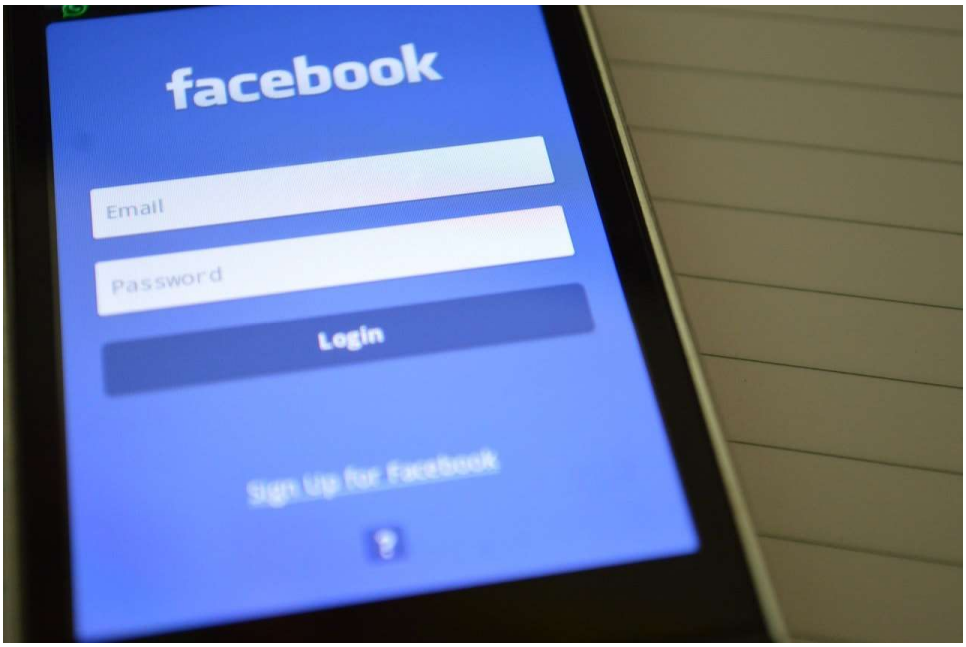

Рисунок 2.35 – Facebook у якості найпопулярнішого мобільного застосунку, створеного на React Native

2. Наступним чудовим прикладом великого React Native продукту є Skype. Ще у 2017 році, рівно через два роки після офіційного релізу React Native Microsoft оголосила, що буде створювати абсолютно новий мобільний застосунок на основі React Native. Користувачі Skype з певною опаскою зустріли цю новину, адже попередня версія мобільного застосунку мала у собі ряд проблем, котрі не були вирішені довгі місяці. Але хвилювання виявились марними, так як новий мобільний клієнт Skype, створений на React Native, виявився повністю переробленим, оновлено було абсолютно все: від значків і нового інтерфейсу до оновленого обміну повідомленнями, котрий з того часу має три розділи: пошук, чат і захоплення.

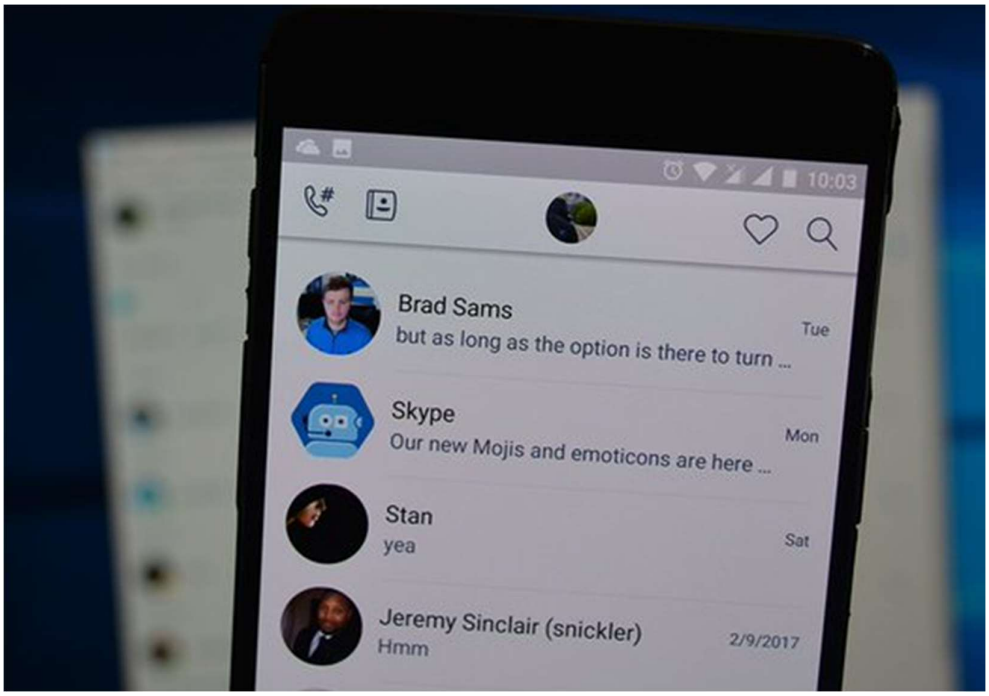

Рисунок 2.36 – Дизайн нової версії Skype, що була повністю створена на RN

3. Instagram почав дуже обережну інтеграцію React Native у свій вже існуючий нативний застосунок [12]. Все почалося з перегляду push-повідомлень, котрі тоді відкривалися у якості WebView. Оскільки інтерфейс був досить зрозумілим та не був перевантаженим зайвими елементами, створювати нову навігаційну інфраструктуру не було потреби. Використання React Native дозволило команді Instagram прискорити розробку своєї програми на 80-85%.

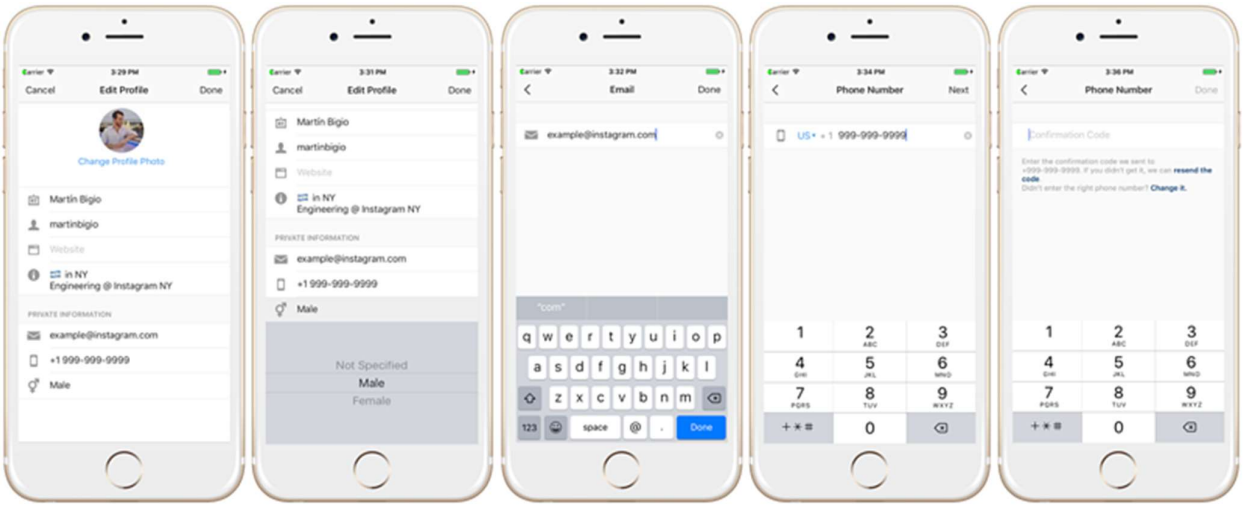

Рисунок 2.37 – Покрокова інтеграція React Native у Instagram

4. Американський бакалійний продавець Walmart завжди славився у IT сфері тим, що не боявся приймати сміливі технологічні рішення. Одним з таких було рішення повністю переписати свої мобільні застосунки на React Native. Раніше деякі частини програми Walmart містили вбудовані веб-представлення, які, як зазначали Walmart Labs, були нижче «стандарту, якого вимагаємо як ми, так і наші клієнти». Після переходу на React Native продуктивність та швидкість роботи мобільних застосунків значно покращилась як на iOS, так і на Android.

Рисунок 2.38 – Інтеграція React Native у Walmart

Результат переходу даного продукту на React Native найкраще за все ілюструють слова Walmart: «Від стартапів до компаній зі списку Fortune 500, якщо ви плануєте взятися за новий мобільний проект, подумайте про використання React Native — ми знаємо, що ви не пошкодуєте.». [13]

Після аналізу найпопулярніших мобільних застосунків, котрі були створені на основі React Native, є сенс перейти до основних переваг цього фреймворку, щоб зрозуміти, чому варто вибрати його у якості рішення для створення мобільного застосунку.

1. Можливість повторного використання коду є найбільшою перевагою React Native. Це означає, що програмістам достатньо створити певний функціонал лише один раз, а можливість використати цей код буде як для iOS, так і для Android. Це є справжньою мрією і для програміста, і для генеральних директорів разом з замовниками.

2. Величезна спільнота розробників. React Native – це проект із відкритим кодом, а це в свою чергу дає змогу абсолютно будь-якому розробнику запропонувати зміни, котрі він вважає таким, що зроблять даний фреймворк кращим. Велика спільнота є просто неймовірно значущою перевагою, якщо говорити про випадки, коли програміст стикається з проблемою і не хоче залишать з нею віч-на-віч. Популярність React Native майже гарантує відсутність подібних випадків, наприклад, на одному лиш Stack Overflow є майже 50 000 активних учасників тегу React Native. Завжди знайдеться програміст, який уже зустрічався з подібною проблем та знайшов для неї найкраще рішення. Окрім очевидних бонусів у якості швидкості вирішення проблем, це також можна розцінювати у якості покращення навичок програмування.

3. Швидке оновлення (fast refresh) дозволяє розробникам запускати програму, оновлюючи її до нових версій та змінюючи інтерфейс користувача. Зміни у такому випадку видні одразу і потреби кожний раз перебудовувати мобільний застосунок вручну немає. Це результує двома важливими перевагами: економія часу і підвищення продуктивності, адже програмісту не треба вручну

займатися створеннями нових білдів програми просто для того, щоб побачити останні зміни.

4. Простий користувацький інтерфейс. React Native використовує React JavaScript для того, щоб створювати інтерфейс програм, а це в свою чергу робить UI більш швидким і покращує швидкість завантаження, що, в цілому, призводить до кращого користувацького досвіду. Реактивний інтерфейс у комбінації з компонентним підходом робить даний фреймворк ідеальним рішенням для створення програмних застосунків як із простим дизайном з мінімальною кількістю елементів, так і з складними layout.

5. Майбутнє React Native виглядає дуже яскраво, особливо враховуючи швидкість його прогресу і той невеликий час, який знадобився даному фреймворку, щоб фактично захопити ринок кросплатформних мобільних застосунків. Розробники справедливо розраховують на довгі роки активної підтримки даного проекту зі сторони розробників з Facebook.

Висновки до розділу 2

У 2022 році значення Kotlin може тільки зрости. Тенденція можна зрозуміти, якщо мати на увазі те, що розробники цієї мови програмування – це команда найкращих програмістів JetBrains, котрі вже багато років розроблюють на підтримують різноманітні IDE для величезної кількості різних мов програмування: починаючи від C# і закінчуючи JavaScript. Маючи можливість проводити аналіз великих масивів даних, вони користуються цим і розвивають Big Data саме для аналізу потреб програмістів, котрі користуються їх програмним забезпеченням. Не дивно, що Kotlin, котрий прийшов на заміну мови програмування Java, що стільки років була майже єдиною опцією для програміста, котрий прагне створювати мобільні застосунки для операційної системи Android, став мовою програмування, яка дійсно виправила абсолютно більшість проблем свого попередника у лиці Java. Рівень переваг, отриманих від роботи на цій платформі, повністю покриває ті незначні слабкі сторони, з якими доведеться зіштовхнутись майбутньому Android

мобільному розробнику. Цю мову програмування дійсно можна розглядати як варіант, котрий зацікавить більше всього досвідчених розробників, котрі прагнуть створити застосунок с найкращим user experience. Але все ж високі витрати на обслуговування, а також дефіцит кваліфікованих розробників роблять Kotlin досить ризиковою інвестицією для малих компаній.

З мовою програмування Swift ситуація ніяк не стає гіршою також. Він прийшов на зміни своєму попереднику у лиці Objective-C також досить невеликий час тому, але вже встиг зарекомендувати себе у якості швидкої, безпечної мови програмування, яка у додатку має високу продуктивність. Та тут справедливо зазначити, що проблема з орієнтованістю лише на операційні системи, котрі випускає компанія Apple, навіть ще більш очевидна, ніж у Kotlin. Підтримка кросплатформності знаходиться на такому початку свого розвитку, що можна сказати, що вона не є пріоритет розробників компанії з Купертіно. Звісно, певні кроки у цьому напрямку можна прослідкувати, особливо звернувши увагу на впровадження SCADE 2.0, але це майбутнє, у яке лише залишається вірити, а реальність поки ніяк не говорить про розвиток кросплатформної розробки у контексті Swift.

React Native, а також його потужний аналог під назвою Flutter нині змагаються за звання найкращого кросплатформного фреймворку. Розробникам подібних рішень дійсно доводиться тримати у голові концепції обох платформ, але результат у якості програмного продукту все ж отримується набагато швидше, через відсутність паралельної розробки двох ідентичних мобільних застосунків, єдиною різницею між якими є операційна система, на якій вони будуть запускатись. Дискусії про якість коду у кросплатформних фреймворках, здається, ніколи на вщухнуть. Вони дійсно мають певну логіку, але зважаючи на кількість переваг таких рішень, кожен розробник має змогу вирішити особисто для себе, чи готовий він прийняти певні компроміси платформи, що дозволить йому створювати кросплатформний застосунок з єдиною кодовою базу, чи він обирає лише один шлях у лиці таких мов програмування як Kotlin чи Swift.

3 АРХІТЕКТУРА МОБІЛЬНОГО ЗАСТОСУНКУ ОБЛІКУ ТА ПОПЕРЕДЖЕННЯ ПРАВОПОРУШЕНЬ ІЗ ПРИВ'ЯЗКОЮ ДО ГЕОЛОКАЦІЇ

3.1 Дерево компонентів

Дерево компонентів мобільного застосунку створеного за допомогою кросплатформного фреймворку для мобільних застосунків React Native – це структура головної папки проекту.

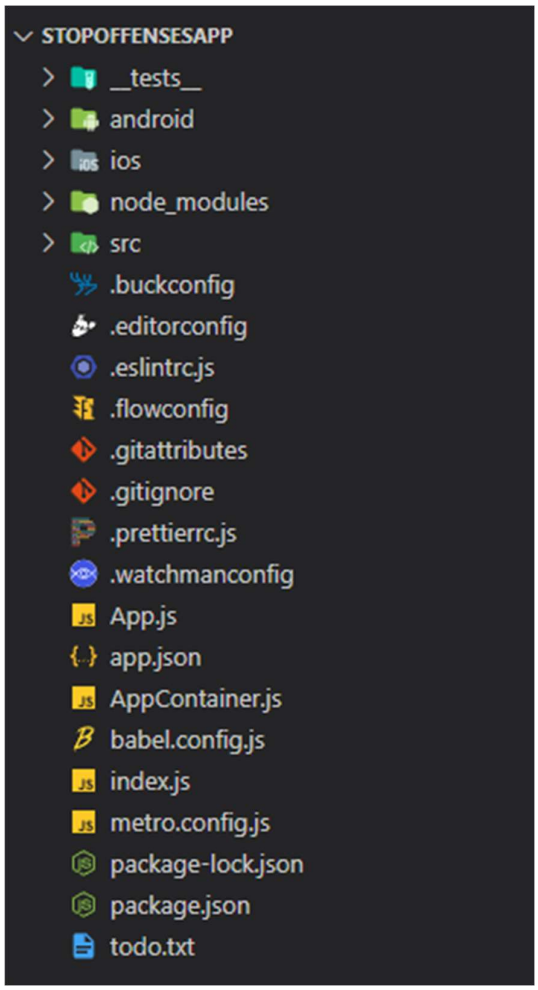

Рисунок 3.1 – Дерево компонентів

У папці __tests__ зазвичай зберігають усі тести, що стосуються даного програмного застосунку: і unit тести, і end-to-end тести. У рамках цього проекту, мною було прийнято рішення не витрачати зайвий час на створення тестів, адже

вони дуже сильно позначаються на швидкості написання коду, а їх реальне значення можна по-справжньому оцінити лише на величезний програмних продуктах, кодова база котрих розвивається протягом довгих років. Саме тому у моєму випадку в цій папці знаходиться лише один файл App-test.js, котрий за замовчування поставляється при ініціалізації проекту React Native.

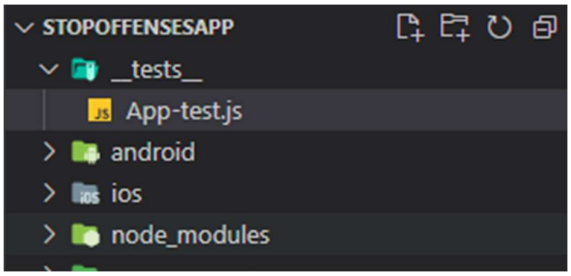

Рисунок 3.2 – Вміст папки __tests__

Наступні дві папка для розгляду – це папка android, а також ios. Вони також одразу поставляються у стандартній збірці проекту React Native, але, на відміну від папки з тестами, вміст даної директорії одразу набагато ширший. Ці дві папки досить часто використовуються програмістами, адже саме у них лежать ті файли, котрі треба редагувати при встановлення додаткових npm модулів.

Npm (node.js package manage) – це найбільший у світі реєстр програмного забезпечення. Він зберігає у собі більше 800 тисяч модулів.

Npm модуль – це, фактично, програма з відкритим кодом, головною метою котрої є вирішення певної вузької проблеми програмного застосунку, з якою часто працюють різні програмісти. Це дозволяє не писати у своєму програмному застосунку логіку абсолютно для всього, а зосередитись лише на тому, що є унікальним саме для того продукту, котрий розроблює програміст.

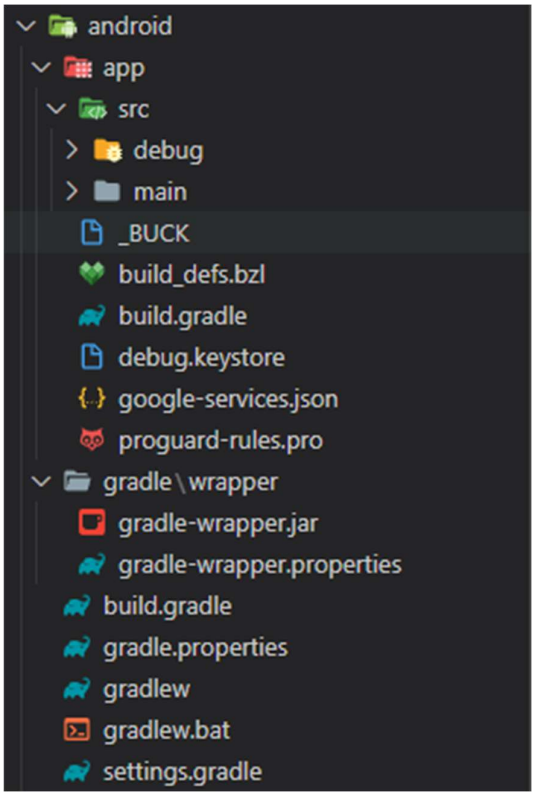

Рисунок 3.3 – Вміст папки android

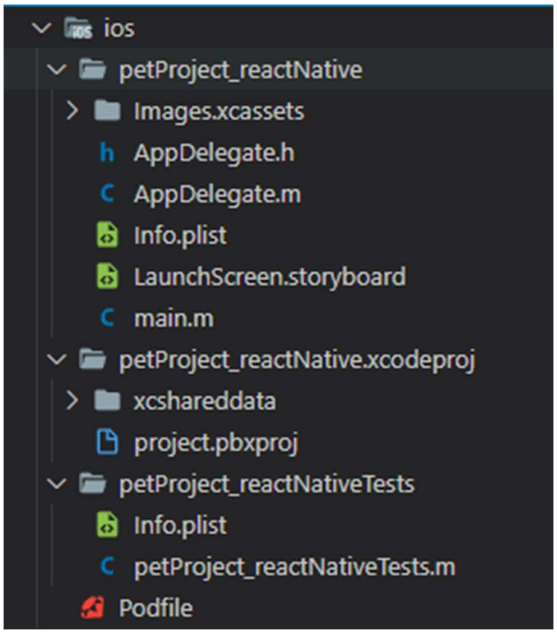

Рисунок 3.4 – Вміст папки ios

Наступна папка у дереві – це node_modules. Саме у цій папці зберігаються усі npm пакети, котрі використовує відкритий застосунок. Директорія node_modules зберігає у собі неймовірно велику кількість файлів. Розмір цієї папки настільки

великий, що сенсу завантажувати її на певні віддалені сервери просто немає. Ця папки завжди залишається поза системою контролю версій git, і кожен розробник після клонування репозиторію, має локально встановити усі пакети, які використовуються у даному програмному застосунку. Звучати ця задача може дуже страшно, але при правильному адмініструванні проекту усі потрібні пакети встановлюються однією маленькою командою npm install.

Далі за списком бачимо папку src (скорочено від слова source). У ній знаходяться майже всі файли, які програміст створює для того, щоб «оживити» програмний застосунок, тобто для написання логіки і зовнішнього вигляду програми.

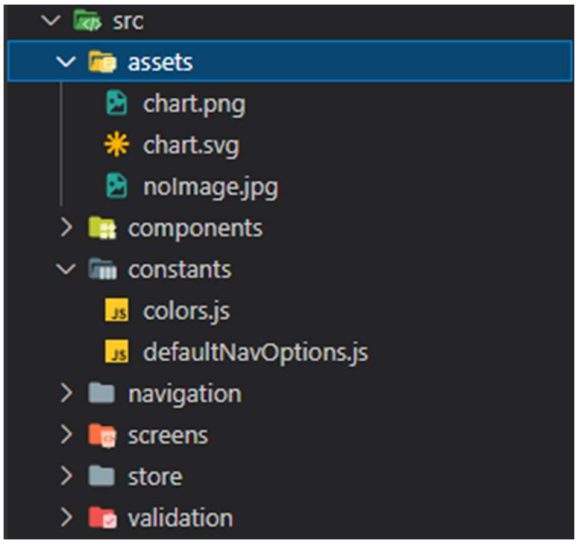

Рисунок 3.5 – Вміст папки src

У папках assets та constants знаходяться статичні файли, але ці дві папки прийнято створювати для того, щоб групувати файли за їх змістом. У assets зберігають звичайні статичні медіа-файли, тобто картинки, відео-файли, pdf-документи тощо. А папка constants слугує для того, щоб групувати усі статичні змінні, котрі можуть використовуватися у будь-якому файлі програми. Такий підхід до збереження файлів дозволяє дуже швидко змінити дані, які використовуються у багатьох місцях. У випадку даного мобільного застосунку, ця папка використовується для збереження усіх hex-кодів кольорів, які

використовуються у мобільному застосунку, а також для стандартних налаштувань навігації.

У папці navigation зберігаються усі навігатори. У даному випадку їх 5. MainNavigator – це головний навігатор для контролю за тим, щоб користувач, котрий ще не увійшов у свій акаунт, не міг потрапити до головного контенту. AuthNavigator відповідає за групування усіх сторінок для логінізації та реєстрації. MainNavigator представляє собою навігатор, котрий створює вкладки та відповідає за навігацію між ними для користувача, котрий увійшов у свій акаунт. ArticlesNavigator і UserNavigator відповідають за навігацію на двох найбільших сторінках у застосунку.

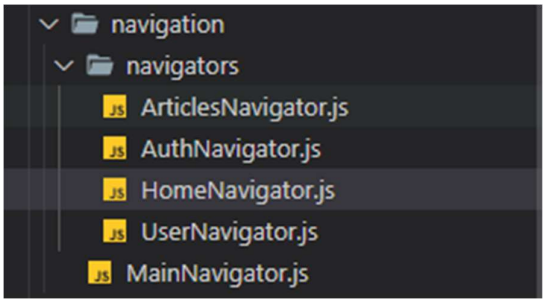

Рисунок 3.6 – Вміст папки navigation

Папка screens зберігає у собі усі найбільші сторінки застосунку. Саме ці сторінки використовуються навігаторами.

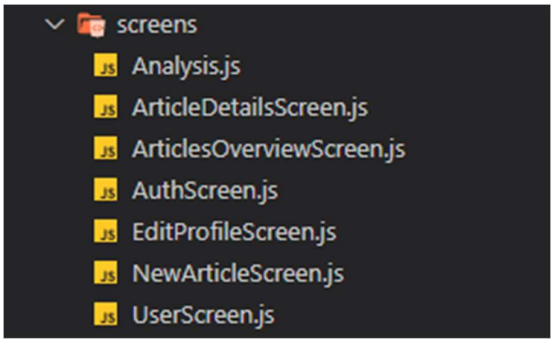

Рисунок 3.7 – Вміст файлу screens

І нарешті папка components. Ця директорія відповідає за групування компонент, які можуть використовуватися у будь-якому місті проєкту і не прив’язані хоч до якоїсь певної сторінки.

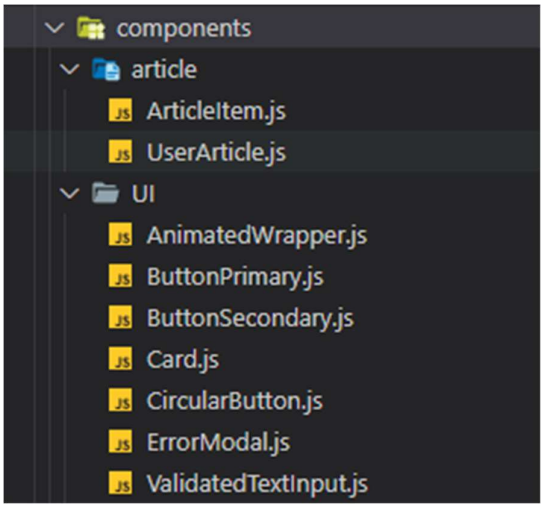

Рисунок 3.8 – Вміст папки component

Директорія validation слугує для того, щоб тримати у одному місці всі файли валідації, які є у проєкті.

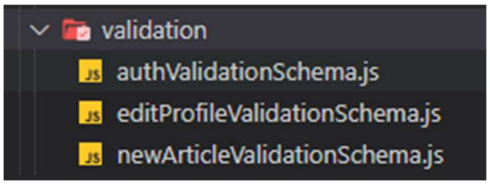

Рисунок 3.9 – Вміст папки validation

3.2 Redux у якості інструменту для збереження глобальних даних у застосунку

Єдина папка, котра ще не була розглянута – це store.

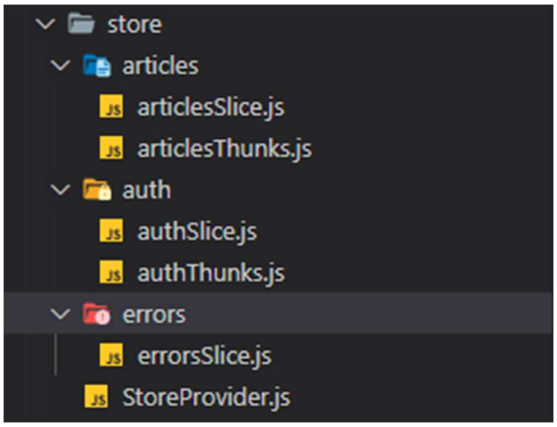

Рисунок 3.10 – Вміст папки store

Саме у цій папці створюється store – глобальний іммутабельний дерево об’єктів у Redux. Redux – це контейнер із передбачуваним станом. Він слугує для

створення даних, котрі є глобальними в усьому проекті, а також являються single source of truth. Тобто саме дані із об’єкта store є найголовнішими даними, тому при певних логічних конфліктах, дані з redux мають слугувати першоджерелом. Незважаючи на те, що Redux найчастіше використовують саме у React і React Native застосунках у якості інструменту для керування станом, він не є унікальним лише для цих двох фреймворків і може використовуватись, наприклад, з Angular, View чи звичайним JavaScript веб-застосунком, який не використовує жодного ферймворку.

Redux дозволяє запобігти такій проблемі, як props drilling [14] завдяки тому, що при використанні цього інструменту стан програми зберігається у єдиному сховищі і кожний компонент може отримати до нього доступ. У Redux існують два головних концепти, на базі котрих цей інструмент і працює.

1. Actions – це події (events). Це єдиний спосіб надсилати дані із компонентів у Redux. Тобто програміст не може витягнути дані із redux і просто перезаписати потрібні йому змінні новими значеннями. Зміна даних має проходити через dispatch actions. Насправді, за цією концепцію немає нічого складного. Dispatch – це один із методів глобального об’єкту store, котрий дозволяє змінювати лише потрібні поля іммутабельним способом, а з action ситуація ще легше, бо action – це звичайний javascript об’єкт, котрий повинен мати одне обов’язкове поле type для того, щоб reducer розумів, яке саме поле і як змінити в залежності від цього поля.

2. Reducer – це чиста функція, котра приймає у якості параметрів стан і action, котрий передається йому завдяки функції dispatch. В залежності від того, яким було поле type у action, буде змінена потрібна частина глобального стану.

3. Store зберігає стан програми, і він має бути унікальним на весь застосунок. Програміст може отримати доступ до стану, оновити стан, зареєструвати чи видалити події за допомогою методів. Дії, які будуть виконуватись над станом, зобов’язані повертати новий стан. Це і є концепція іммутабельності, тобто програміст не може змінити об’єкт, правильний підхід при

такому підході – це створення копії потрібного об’єкту, її зміна, а також подальше повернення саме зміненої копії стану.

4. Усі actions, котрі функція dispatch передає через себе одразу потрапляють у reducer, але це не єдина можлива поведінка, а лише те, що відбувається за замовчуванням. Насправді ж при використання middleware можна розробник може перехопити будь-який action ще до того, як він потрапить у reducer. Фактично, проміжне програмне забезпечення (middleware) – це функції, які викликають наступний метод, отриманий в аргументі після обробки поточної дії. І ці функції викликаються після кожного dispatch. При створенні даного мобільного застосунку, мною було використано thunk middleware.

5. Thunk – це загальний програмний термін, котрий фактично розшифровується як «деякий шматок коду, котрий виконуватиме певну відкладену роботу». Замість того, щоб виконувати щось прямо зараз, тобто при виклику функцій, а thunk – це також функція, програміст може написати тіло функції, яка буде використана після чогось. Thunk є стандартом для написання асинхронної логіки, тобто взаємодії з backend, і подальшими маніпуляціями зі станом, базуючись на відповіді від backend.

3.3 Користувацький інтерфейс

Заходячи у мобільний застосунок вперше, користувач повинен буде зареєструватися, якщо у нього ще немає акаунте, або ж увійти в уже існуючий, якщо такий у нього є. При реєстрації у користувача є три різних опції.

1. Створення нового акаунте через email і пароль.
2. Реєстрація через google пошту.
3. Реєстрація через акаунт facebook.

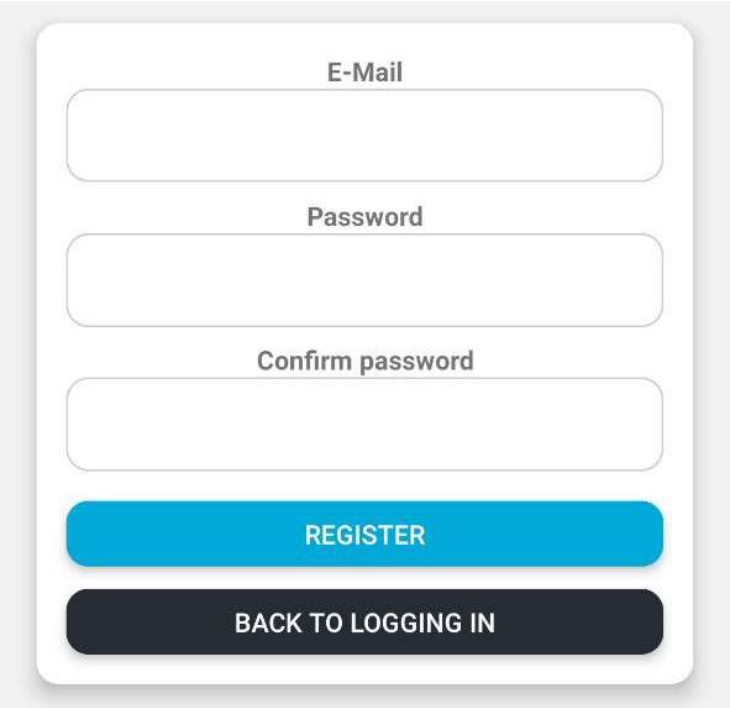

Рисунок 3.11 – Форма реєстрації

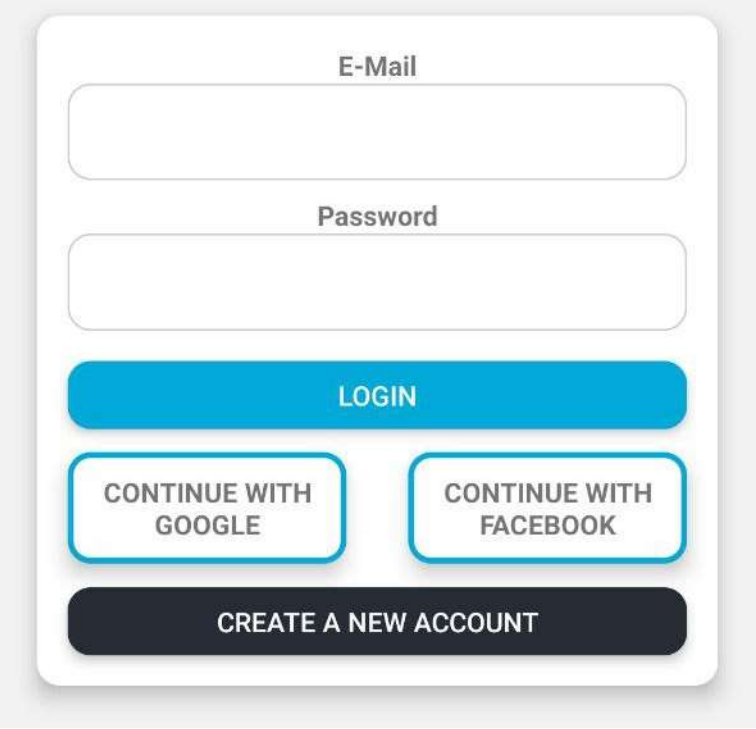

Рисунок 3.12 – Форма для входу у свій акаунт

Після входу у свій акаунт користувач буде автоматично направлений до головної сторінки даного мобільного застосунку, де він побачить список скарг, котрі були створені такими ж користувачами, як і він.

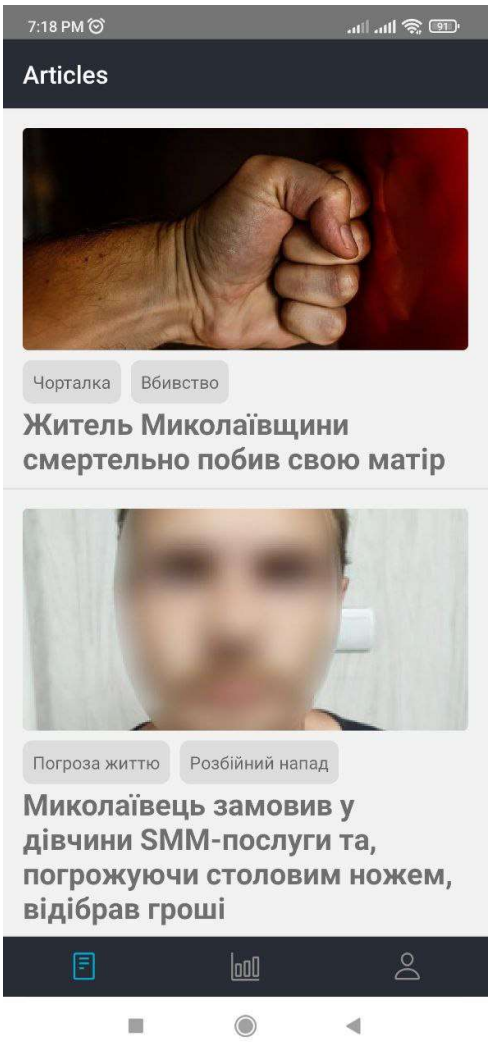

Рисунок 3.13 – Головна сторінка з усіма створеними скаргами

Для того, щоб побачити деталі певного випадку, користувачу буде достатньо натиснути на будь-яке місце у межах тієї скарги, котру він бажає переглянути (заголовок, картинка чи теги). Після цього він буде направлений на сторінку з детальним описом.

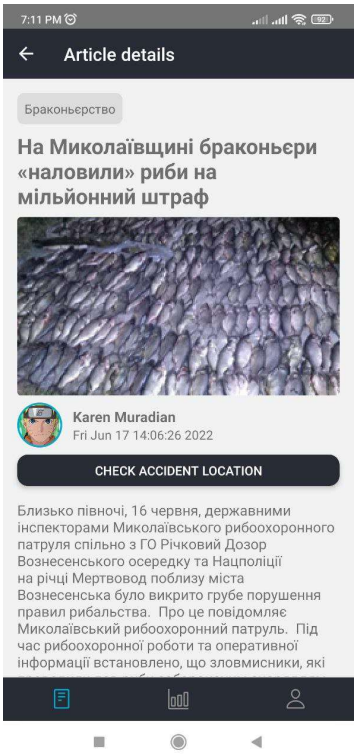

Рисунок 3.14 – Детальна інформація про скаргу (частина 1)

Рисунок 3.15 – Детальна інформація про скаргу (частина 2)

На цій сторінці користувач може ознайомитися з повним текстом скарги, переглянути усі фотографії, що були прикріплені, а також натиснути на кнопку «Check accident location», щоб відкрити мапу з відміченим місцем, де трапилася ця подія.

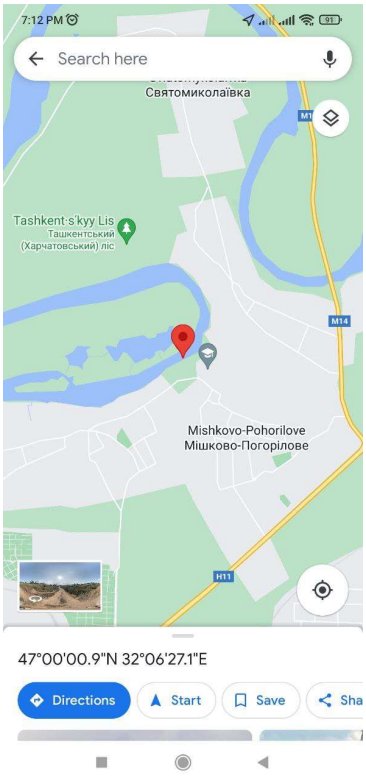

Рисунок 3.16 – Мапа з відміченим місцем події

Для навігації по застосунку користувач може використовувати кнопки знизу, вони переключають між собою вкладки. За замовчування відкривається головна вкладка, що має назву Articles. Після натискання на кнопку посередині нижньої частини екрану відкривається вкладка аналізу даних.

Рисунок 3.17 – Вкладка аналізу даних

Після натискання на останню кнопку знизу користувач буде направлений на сторінку з інформацією про свій акаунт. Тут він може вийти з акаунту, змінити особисту інформацію і аватар, переглянути створені ним скарги, редагувати та видалити їх, а також створити нові.

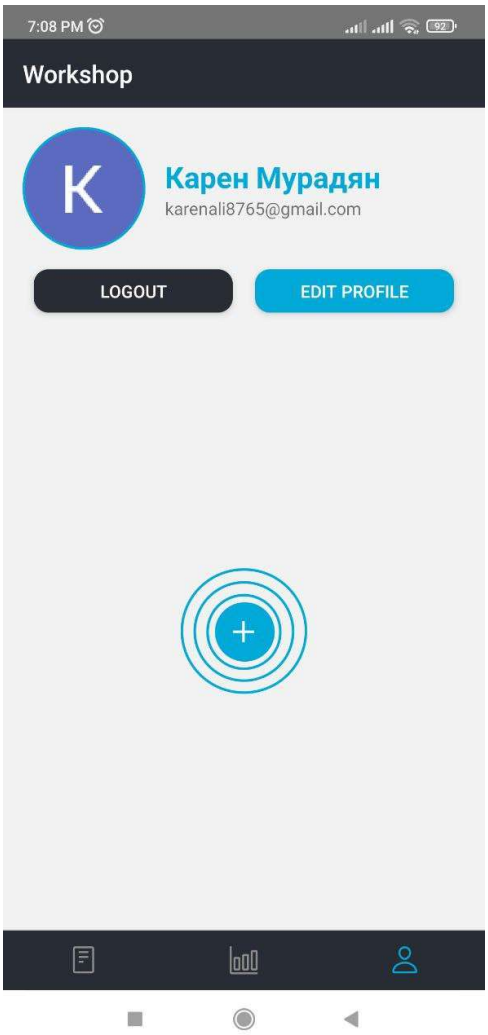

Рисунок 3.18 – Вкладка «Workshop»

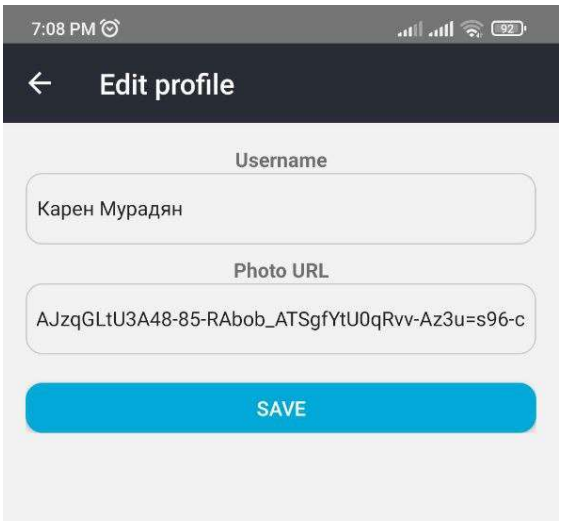

Рисунок 3.19 – Редагування інформації акаунту

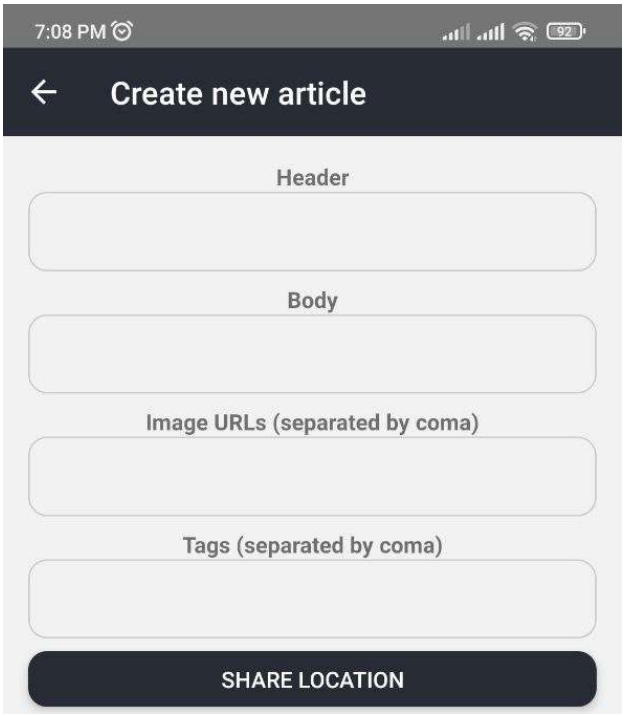

Рисунок 3.20 – Створення нової скарги

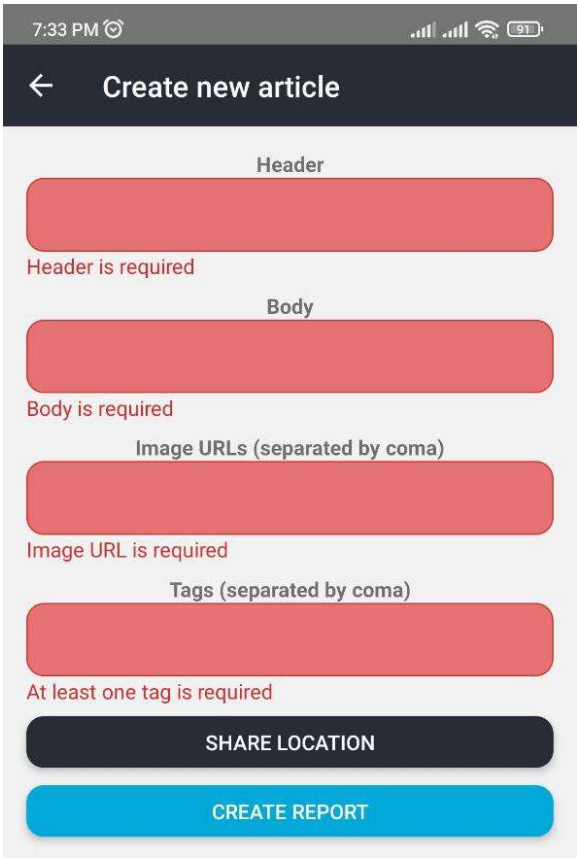

Рисунок 3.21 – Валідація форми створення нової скарги

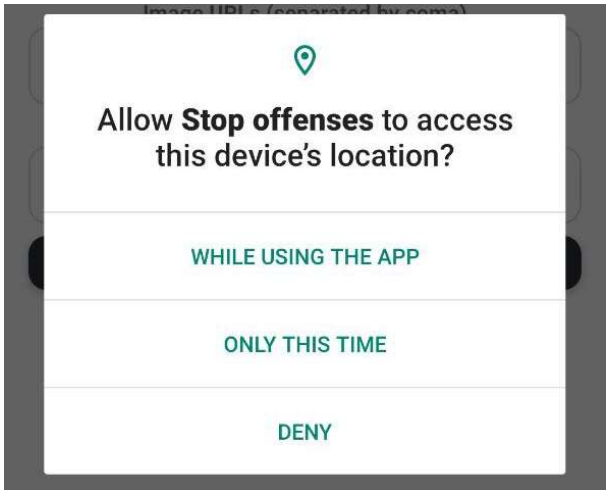

Рисунок 3.22 – Діалогове вікно про доступ до геолокації

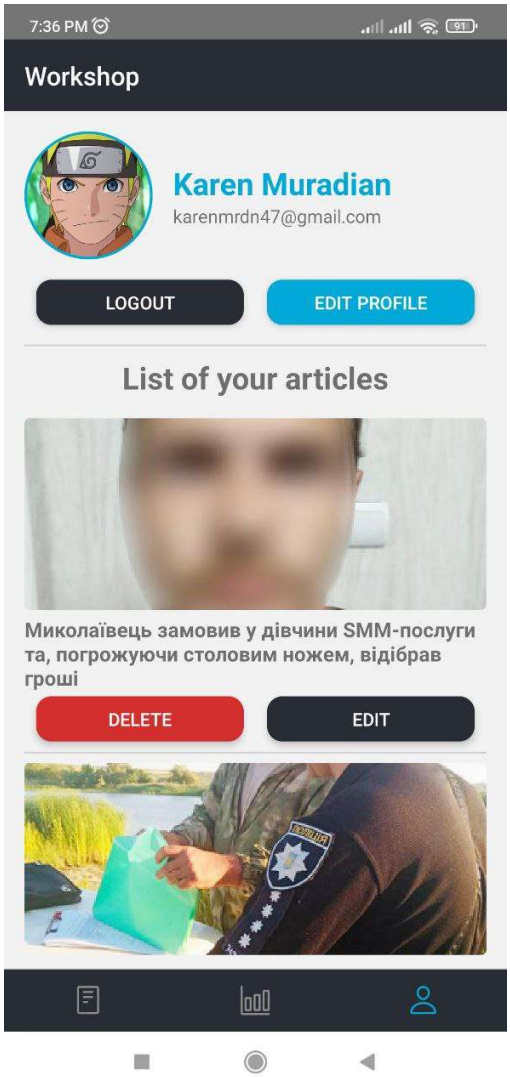

Рисунок 3.23 – Вкладка «Workshop» у користувача, що створив декілька скарг

Після створення скарги користувачу буде запропоновано повідомити про це у службу підтримки, після чого професіонали зможуть оперативно передати цю інформацію до органів виконавчої влади.

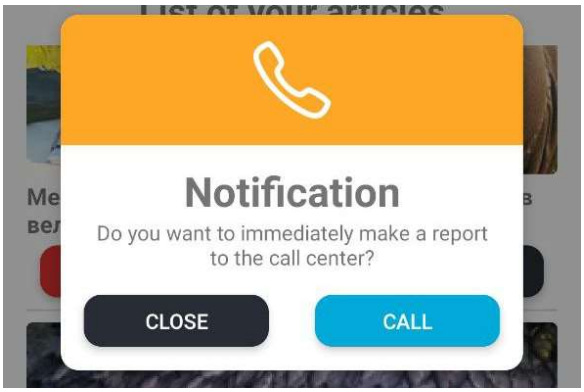

Рисунок 3.24 – Діалогове вікно з пропозицією звернутися у службу підтримки

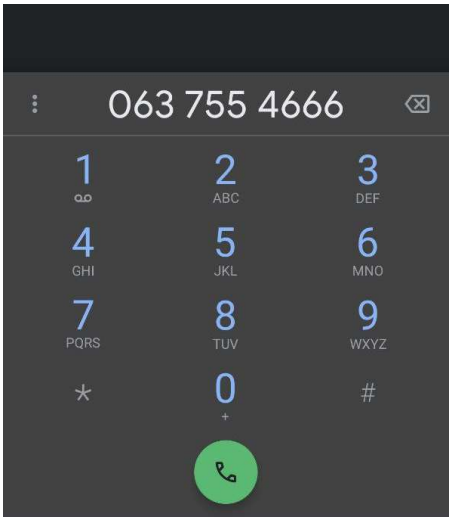

Рисунок 3.25 – Номер служби підтримки, на котрий направляється користувач

Висновки до розділу 3

У даному розділі було представлено весь функціонал мобільного застосунку обліку та попередження правопорушень із прив'язкою до геолокації, продемонстровано дерево компонентів та загальну структуру проекту. Також було виконано опис ключових концепцій Redux у якості інструменту для керування станом.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп’ютерних наук
Кафедра інтелектуальних інформаційних систем

Спеціальний розділ

ОХОРОНА ПРАЦІ

до кваліфікаційної роботи

на тему:

«МОБІЛЬНИЙ ЗАСТОСУНОК ОБЛІКУ ТА ПОПЕРЕДЖЕННЯ ПРАВОПОРУШЕНЬ ІЗ ПРИВ'ЯЗКОЮ ДО ГЕОЛОКАЦІЇ»

Спеціальність 122 «Комп’ютерні науки»

122 – БКР – 402.21810219

Виконав студент 4-го курсу, групи 402
_____ *К. А. Мурадян*

«___» _____ 2022 р.

Консультант к.т.н., доцент
_____*А. О. Алексєєва*

«___» _____ 2022 р.

Миколаїв – 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ВДТ	–	Візуальний дисплейний термінал
ЕОМ	–	Електронна обчислювальна машина
ПЕОМ	–	Персональні електронні обчислювальні машини

4. ОХОРОНА ПРАЦІ

Охорона праці при користуванні екранними пристроями

ВСТУП

Пандемія COVID-19 стала стрес-тестом для урядів всіх країн світу, при цьому інтернет, веб-ресурси та додатки відіграють вирішальну роль у забезпечення взаємопов'язаності та доступності критично важливої інфраструктури та ресурсів. Так, надійний високошвидкісний інтернет є ключовим фактором у забезпеченні доступу лікарень та медичних установ до глобальних інформаційних мереж та ресурсів, необхідних для боротьби з вірусом. Широкополосні з'єднання зараз мають вирішальне значення і для навчальних закладів та підприємств, дозволяючи їм продовжувати надання основних послуг. У зв'язку з безпрецедентною всесвітньою надзвичайною ситуацією у сфері охорони здоров'я, мережі та платформи навантажуються до краю – деякі оператори повідомляють про стрибки навантаження до 800 відсотків.

У результаті потреби більшості бізнесів та критично важливих інфраструктур забезпечення подальшої роботи на «доковідному» рівні, кількість програмістів, що потрібні для такого виду діджиталізації різко виросла. Програмування – робота, від посидючості у якій часто залежить результат. Не завжди такий спосіб життя позитивно позначається на здоров'ї.

Здавалося б, які можуть бути проблеми у людини, яка працює мозком, насамперед, а не руками – у другу. Але програмісти, на жаль, найчастіше стикаються з такими захворюваннями, як: зниження зору, тунельний синдром, гіподинамія, захворювання опорно-рухового апарату, алергія.

Звичайно, це не весь перелік ризиків для здоров'я програміста, а основні захворювання, які можуть зненацька застати його, якщо не робити нічого спрямованого на їх запобігання та не звертати увагу на перші симптоми.

4.1 Нормативна база в галузі охорони праці при роботі з екранними пристроями

Екранні пристрої – електронні засоби для відтворення будь-якої графічної або алфавітно-цифрової інформації (на основі електронно-променевої трубки, рідкокристалічні, плазмові, проекційні, органічні світлодіодні монітори та інші новітні розробки у сфері інформаційних технологій) [15].

Мінімальні вимоги безпеки та захисту здоров’я під час роботи з екранними пристроями всіх типів і моделей установлюють Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями, затверджені наказом Міністерства соціальної політики від 14.02.2018 № 207, який поширюється на всіх суб’єктів господарювання незалежно від форм власності, організаційно-правової форми і видів діяльності та встановлює мінімальні вимоги безпеки та захисту здоров’я під час здійснення роботи, пов’язаної з використанням екранних пристроїв незалежно від їхнього типу та моделі [15].

Ще одним нормативно-правовим актом, яким має керуватися роботодавець з метою створення належних умов праці при роботі з екранними пристроями є Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, затвердженими постановою Головного державного санітарного лікаря України 10.12.1998 р. № 7, що застосовуються на підприємствах, організаціях, установах незалежно від форми власності та поширюються на умови й організацію праці при роботі з візуальними дисплейними терміналами усіх типів вітчизняного та закордонного виробництва на основі електронно-променевих трубок, якими укомплектовані електронно-обчислювальні машини колективного використання та персональні ЕОМ [16].

4.2. Загальні вимоги до роботодавців

Задля забезпечення безпеки та захисту здоров’я під час здійснення роботи, пов’язаної з використанням екранних пристроїв роботодавець повинен виконувати дотримуватися певних вимог. Однак за певних умов, а саме – якщо це не суперечить чинному законодавству, то роботодавець має право встановлювати більш жорсткі та/або спеціальні вимоги безпеки і захисту здоров’я та життя працівників під час роботи з екранними пристроями. Головне, що потрібно пам’ятати під час облаштування робочого місця працівника з екранними пристроями необхідно обирати таке устаткування, яке не створює зайвого шуму та не виділяє надлишкового тепла.

Нижче наведено частину вимог з документу про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями:

– роботодавець повинен поінформувати працівників про умови праці та наявність на їх робочих місцях небезпечних та шкідливих виробничих факторів ;

– роботодавець повинен вжити відповідних заходів, щоб забезпечити відповідність робочого місця працівника;

– під час облаштування робочого місця працівника з екранними пристроями необхідно обирати таке устаткування, яке не створює зайвого шуму та не виділяє надлишкового тепла;

– роботодавець повинен за рахунок тривалості робочої зміни організувати внутрішні регламентовані перерви для відпочинку відповідно до ДСанПІН 3.3.2.007-98 [17];

– роботодавець зобов’язаний за необхідності проводити лабораторні дослідження умов праці працівників з метою виявлення шкідливих і небезпечних факторів виробничого середовища, важкості та напруженості трудового процесу.

4.3 Вимоги до приміщення з використанням екранних пристроїв

Приміщення, у яких робітники використовують електронно-обчислювальні машини, повинні відповідати певному ряду вимог, які перевіряються спеціальними службами, призначеними для нагляду за дотриманням вимог охорони праці на підприємствах.

Нижче наведено частину вимог з Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин:

– розміщення робочих місць з ВДТ ЕОМ і ПЕОМ у підвальних приміщеннях, на цокольних поверхах заборонено;

– площа на одне робоче місце має становити не менше ніж 6,0 $м^2$, а об'єм не менше ніж 20,0 $м^3$;

– приміщення для роботи з ВДТ повинні мати природне та штучне освітлення відповідно до СНиП II-4-79 [18];

– природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи північний схід і зебезпечувати коефіцієнт природною освітленості не нижче ніж 1,5%. Розраховується цей коефіцієнт за методикою, викладеною в СНиП II-4-79 [18];

– у приміщеннях з ЕОМ слід щоденно робити вологе прибирання;

– приміщення з ЕОМ мають бути оснащені аптечками першої медичної допомоги;

– при приміщеннях з ВДТ мають бути обладнані побутові приміщення для відпочинку під час роботи, кімната психологічного розвантаження. В кімнаті психологічного розвантаження слід передбачити встановлення пристроїв для приготування й роздачі тонізуючих напоїв, а також місця для занять фізичною культурою (СНиП 2.09.04.-87 [19]);

Нижче наведено частину вимог з документу наказу про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями від 14.02.2018:

– робочі місця працівників з екранними пристроями мають бути спроектовані так і мати такі розміри, щоб працівники мали простір для зміни робочого положення та рухів;

– усе випромінювання від екранних пристроїв має бути зведене до гранично допустимого рівня;

– освітлення робочого місця працівника з екранними пристроями має створювати відповідний контраст між екраном і навколишнім середовищем (з урахуванням виду роботи) та відповідати вимогам ДСанПІН 3.3.2.007-98 [19];

– мікроклімат виробничих приміщень з робочими місцями працівників з екранними пристроями має підтримуватись на постійному рівні та відповідати вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [20];

– робочий стіл або робоча поверхня повинні бути достатнього розміру та мати поверхню з низькою відбивною здатністю, допускати гнучкість під час розміщення екрана, клавіатури, документів і відповідного устаткування;

– робоче крісло має бути стійким і дозволяти працівнику з екранними пристроями легко рухатися та займати зручне положення. Сидіння має регулюватися по висоті, спинка сидіння – як по висоті, так і по нахилу. Слід передбачати підніжку для тих, кому це необхідно для зручності.

Крім цього для зменшення впливу негативних факторів на працівника, виробниче приміщення повинно відповідати нормам щодо параметрів мікроклімату, освітлення, шуму та вібрації, рівні електромагнітного та іонізуючого випромінювання.

По-перше, на усіх робочих місцях із електронно-обчислювальними машинами обов’язково має бути забезпечено оптимальні значення параметрів температури відносної вологості й рухливості повітря (ГОСТ 12.1.005-88 [21], СН 4088-86 [22]) (табл. 3.1).

Таблиця 4.1 – Нормовані величини температури, відносної вологості та швидкості руху в робочій зоні виробничих приміщень з ВДТ

Пора року	Категорія робіт	Оптимальна температура повітря, °С, не більше	Оптимальна відносна вологість повітря, %	Оптимальна швидкість руху повітря, м/с
Холодна	Легка – Іа	22-24	40-60	0,1
	Легка – Іб	21-23	40-60	0,1
Тепла	Легка – Іа	23-25	40-60	0,1
	Легка – Іб	22-24	40-60	0,2

До категорії робіт Іа належать, що виконується робота сидячі і не потребує фізичного напруження людини, до категорії робіт Іб належать, що робота виконується сидячі, стоячи або пов’язані з ходінням та потребує деякого фізичного напруження.

По-друге, показники рівню позитивних та негативних іонів в повітрі приміщень з ВДТ мають відповідати санітарно – гігієнічним нормам №2152-80 [23] (табл. 3.2).

Таблиця 4.2 – Рівні іонізації повітря приміщень при роботі на ВДТ

Рівні	Кількість іонів в 1 $см^3$ повітря	
	n+	n-
Мінімально необхідний	400	600

Оптимальний	1500-3000	3000-5000
Максимально допустимий	50000	50000

Вимірювання кількості іонів та його полярності порядку поточного нагляду проводиться 1 разів у квартал. Вимірювання проводяться також у випадках:

– встановлення нових або відремонтованих іонізаторів;

– організації нових робочих місць;

– впровадження нових технологічних процесів, що потенційно можуть змінити іонний режим у зоні дихання персоналу.

Що до штучного освітлення в приміщеннях з робочими місцями, обладнаними ВДТ ЕОМ та ПЕОМ, то воно має здійснюватись системою загального івномірного освітлення. Зазначення освітлення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300-500 лк. Якщо ці значення освітленості неможливо забезпечити системою загального освітлення, допускається використовувати місцева освітлення. При цьому світильники місцевого освітлення слід встановлювати таким чином, щоб не створювати бліків на поверхні екрана, а освітленість екрана має не перевищувати 300 лк.

У якості джерела світла у приміщеннях для штучного освітлення мають використовуватися люмінесцентні лампи типу ЛБ. При використання відбитого освітлення у виробничих та адміністративних приміщеннях допускається використання метало-галогенних ламп, які мають потужність 250Вт.

Допускається використання ламп розжарювання у світильниках місцевого освітлення.

Інтенсивність потоків інфрачервоного випромінювання має не перевищувати допустимих значень, які зазначені у ДСН 3.3.6.042-99 [20].

Інтенсивність потоків ультрафіолетового випромінювання має не перевищувати допустимих значень, які зазначені у СН 4557-88 [24].

Потужність експозиційної дози рентгенівського випромінювання на відстані 0,05м від екрану та корпусу комп’ютера не повинна перевищувати 0,1мбер/рік (100мкР/рік).

4.4 Вимоги до роботи з екранними пристроями

В обов’язковому порядку працівники, основним місцем роботи яких є місце за екранними пристроями, повинні у свою чергу також дотримуватись певних вимог. Нижче наведено частину вимог з наказу Міністерством соціальної політики України «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями»

На сьогодні існує ряд вимог, які повинні використовувати працівники на підприємствах, основним місцем роботи яких є місце за персональним комп’ютером. Нижче наведено частину вимог з наказу Міністерства соціальної політики щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями від 14.02.2018 [15].

Мінімальні вимоги безпеки під час роботи з екранними пристроями:

– щодня перед початком роботи необхідно очищати екранні пристрої від пилу та інших забруднень;

– після закінчення роботи екранні пристрої слід відключати від електричної мережі;

– у разі виникнення аварійної ситуації необхідно негайно відключити екранний пристрій від електричної мережі.

Не допускається:

– виконувати технічне обслуговування, ремонт і налагодження екранних пристроїв безпосередньо на робочому місці працівника під час роботи з екранними пристроями;

– відключати захисні пристрої, самочинно проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;

– працювати з екранними пристроями, у яких під час роботи виникають нехарактерні сигнали, нестабільне зображення на екрані та інші несправності.

Мінімальні вимоги безпеки до екранних пристроїв:

– екранні пристрої не мають бути джерелом ризику для працівників;

– усе випромінювання, за винятком видимої частини електромагнітного спектра, має бути зведене до незначного рівня з погляду безпеки і охорони здоров’я працівників;

– символи на екранних пристроях мають бути чіткими, відповідного розміру, між символами і рядками символів має бути належна відстань;

– зображення на екрані має бути стабільним, без миготінь або інших видів нестабільності;

– яскравість та/або контрастність символів має легко регулюватися працівником під час роботи з екранними пристроями, а також швидко адаптуватися до навколишніх умов;

– вибираючи екрани, слід надавати перевагу таким екранам, які легко та вільно повертаються і нахиляються відповідно до потреби працівника;

– за необхідності може використовуватись окрема підставка або регульований стіл для розміщення екрана;

– екран не має відблискувати або відбивати світло, щоб не викликати дискомфорту у працівника під час роботи з екранними пристроями.

– обираючи клавіатуру, слід надавати перевагу такій, яка відкидається і є відокремленою від екрана, щоб працівник міг вибрати зручну робочу позу й уникнути втоми рук;

– поверхня клавіатури має бути матовою, щоб уникнути віддзеркалювання. Розташування клавіш і самі клавіші мають полегшувати роботу із клавіатурою. Позначення клавіш повинно бути достатньо контрастним і розбірливим;

– устаткування, яке входить до робочої станції, не має виділяти надлишкового тепла, яке може спричинити незручності працівникам під час роботи з екранними пристроями.

– під час розробки, вибору, замовлення та модифікації програмного забезпечення, а також під час розробки завдань, що передбачають використання

устаткування з екранними пристроями, роботодавець має керуватися таким програмним забезпеченням, яке відповідає розв’язуваним завданням і є простим у використанні, а де необхідно – адаптованим до рівня знань і досвіду працівника.

4.5. Вимоги щодо режиму відпочинку та праці на підприємствах з екранними пристроями

Не менш важливим аспектом при організації роботи, яка пов’язана із електронно-обчислювальними машинами, є внутрішньо змінні регламентовані перерви на відпочинок. Вони повинні містити додаткові короткочасні перерви, які запобігають появі ознак стомлення та зниження продуктивності працівників.

При виконанні протягом дня робіт, що належать до різних видів трудової діяльності, за основну роботу з ВДТ ЕОМ і ПЕОМ слід вважати таку, що займає не менше 50% часу впродовж робочої зміни мають передбачатися [16]:

– перерви для відпочинку і вживання їжі (обідні перерви)

– перерви для відпочинку і особистих потреб (згідно з трудовими нормами);

– додаткові перерви, що вводяться для окремих професій з урахуванням особливостей трудової діяльності.

Встановлюються такі внутрішньозмінні режими праці та відпочинку при роботі з ЕОМ при 8-годинній денній робочій зміні в залежності від характеру праці:

– для розробників програм із застосуванням ЕОМ, слід призначати регламентовану перерву для відпочинку тривалістю 15 хвилин через кожну годину роботи за ВДТ;

– для операторів із застосування ЕОМ, слід призначати регламентовані перерви для відпочинку тривалістю 15 хвилин через кожні дві години;

– для операторів комп'ютерного набора слід призначати регламентовані перерви для відпочинку тривалістю 10 хвилин після кожною години роботи за ВДТ.

У всіх випадках, коли виробничі обставини не дозволяють застосувати регламентовані перерви, тривалість безперервної роботи з ВДТ не повинна перевищувати 4 години.

При 12-годинній робочій зміні регламентовані перерви повинні встановлюватися в перші 8 годин роботи аналогічно перервам при 8-годинній робочій зміні, а протягом останніх 4-х годин роботи, незалежно від характеру трудової діяльності, через кожну годину тривалістю 15 хвилин.

Також із метою зменшення негативного впливу монотонності є доцільним застосовувати чергування операцій усвідомленого тексту і числових даних (зміна змісту роботи). Чередування вводу даних та редагування текстів.

Активний відпочинок має полягати у виконанні комплексу гімнастичних вправ, спрямованих на зняття нервового напруження, м'язове розслаблення, відновлення функцій фізіологічних систем, що порушуються протягом трудового процесу, зняття втоми очей, поліпшення мозкового кровообігу і працездатності

За умови високого рівня напруженості робіт з ВДТ показане психологічне розвантаження у спеціально обладнаних приміщеннях (в кімнатах психологічного розвантаження) під час регламентованих перерв або в кінці робочого дня.

ВИСНОВКИ

Під час виконання спеціальної частини з охорони праці було досліджено умови організації роботи працівників на підприємствах та установах, а також правила роботи при користуванні електронно-обчислювальними пристроями.

ІТ-спеціальності з кожним роком стають все більш актуальними: програмісти, дизайнери, тестувальники, розробники баз даних, менеджери проектів – це лише невеликий список тих спеціалістів, котрих нині як ніколи потребують бізнеси. Тенденція наростання попиту на таких фахівців була відчутною і до глобальної пандемії коронавірусу, але життя у реаліях ковіду фактично викликало дефіцит людей, що можуть на якісному рівні виконувати таку роботу. З однієї сторони робота у ІТ-сфері – це просто мрія для кожної людини, але, якщо придивитись на це більш ретельно, то можна поміти і другу сторону – велику кількість проблем, в першу чергу пов’язаних із погіршення здоров’я, котрі сидячий образ життя та постійна робота з монітором провокують. Це зниження зору, тунельний синдром, гіподинамія, захворювання опорно-рухового апарату, алергія.

Якщо не робити нічого спрямованого на запобігання перелічених проблем, не звертати увагу на перші їх симптоми та нехтувати правилами і вимогами, що прописані у нормативних документах, то можна дуже швидко відчути симптоми на собі у повній мірі.

Як самі робітники, так і роботодавці мають досконало знати та дотримуватися нормативних вимог щодо охорони праці під час трудової діяльності. Якщо говорити про ІТ спеціальності, то слідкувати, насамперед, треба за температурою повітря у приміщенні, вологістю повітря, освітленням, шумом та вібрацією, що оточуюсь фахівців під час робочого процесу.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було проведено аналіз стану злочинності та протидії їй на території України. Встановлено, що величезна кількість закритих проваджень (майже 300 тисяч) була досягнута у результаті їх закриття за реабілітуючими обставинами, тобто покарано нікого не було, а порушники закону залишились на свободі. Також було проведено аналіз попередження злочинності у високорозвинених зарубіжних країнах та встановлено різні моделі превентивної діяльності, що могли б дійсно покращити ситуацію з попередженням правопорушень в Україні.

Після цього було створено мобільний застосунок обліку та попередження правопорушень із прив'язкою до геолокації, який у перспективі може понизити рівень злочинності у різних містах завдяки тому, що абсолютно кожна людина, яка користується даною програмою, може зафіксувати акт порушення правопорядку, одразу створити і опублікувати скаргу, а також направити заявку до консультаційного центру, після чого професіонали зможуть оперативно передати цю інформацію до органів виконавчої влади. Функціональність розробленої програми дозволяє користувачу швидко створити скаргу, додати до неї текстовий опис, фотографії, а також прикріпити геолокацію, щоб надати повну картинку про подію.

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Незаконні вказівки на колегії щодо не внесення відомостей в ЄРДР URL: https://www.youtube.com/watch?v=STyLdRofRcE (дата звернення: 10.03.22).

2. Візит президента України Володимира Зеленського до Миколаєва URL: https://news.pn/ru/public/220324 (дата звернення: 11.03.22).

3. Стан злочинності на території України за 2020 рік URL: http://www.baganets.com/blogs-baganets/zagalnii-stan-zlochinnost-ta-protid-i-na-123.html (дата звернення: 11.03.22).

4. Стан організованої злочинності на території України за 2020 рік URL: http://www.baganets.com/blogs-baganets/stan-organ-zovano-zlochinnost-ta-protid-.html (дата звернення: 11.03.22).

5. Проблеми протидії економічній злочинності на території України за 2020 рік URL: http://www.baganets.com/blogs-baganets/stan-ta-problemi-protid-ekonom-chn-i-zlo.html (дата звернення: 11.03.22).

6. Void safety URL: https://en.wikipedia.org/wiki/Void_safety (дата звернення: 14.03.22).

7. Front-end framework popularity URL: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190 (дата звернення: 16.03.22).

8. Survey: What JavaScript frameworks do you regularly use? URL: https://www.jetbrains.com/lp/devecosystem-2021/javascript/ (дата звернення: 16.03.22).

9. Why you should migrate from Ionic, Cordova, or PhoneGap to React Native (Updated) URL: https://www.netguru.com/blog/react-native-comparison (дата звернення: 16.03.22).

10. 13 Great Examples of React Native Apps in 2021 [Updated] URL: https://www.netguru.com/blog/react-native-apps (дата звернення: 16.03.22).

11. Top 10 Most Downloaded Apps of 2021 So Far URL: https://www.cyberclick.net/numericalblogen/top-10-most-downloaded-apps-of-2020-so-far (дата звернення: 16.03.22).

12. React Native at Instagram URL: https://instagram-engineering.com/react-native-at-instagram-dd828a9a90c7#.3h4wir4zr (дата звернення: 16.03.22).

13. React Native at WalmartLabs URL: https://medium.com/walmartglobaltech/react-native-at-walmartlabs-cdd140589560#.ueonqqloc (дата звернення: 16.03.22).

14. What is prop drilling and how to avoid it? URL: https://www.geeksforgeeks.org/what-is-prop-drilling-and-how-to-avoid-it/ (дата звернення: 17.03.22).

15. Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями URL: http://zakon3.rada.gov.ua/laws/show/z0508-18 (дата звернення: 20.05.2022).

16. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПІН 3.3.2.007-98 URL: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text (дата звернення: 20.05.2022).

17. Державні санітарні правила і норми роботи з ВДТ ЕОМ ДСанПІН 3.3.2.007-98 URL: http://mozdocs.kiev.ua/view.php?id=2445 (дата звернення: 20.05.2022).

18. СНиП II-4-79. Природне і штучне освітлення URL: https://dnaop.com/html/45036/doc- СНиП_II-4-79_ (дата звернення: 20.05.2022).

19. СНиП 2.09.04.-87. Адміністративні і побутові будівлі URL: https://dnaop.com/html/54074/doc -СНиП_2.09.04-87_ (дата звернення: 20.05.2022).

20. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень URL: https://dnaop.com/html/34094/doc -ДСН_3.3.6.042-99 (дата звернення: 22.05.2022).

21. ГОСТ 12.1.005-88.ССБП URL: http://docs.cntd.ru/document/1200003608 (дата звернення: 22.05.2022).

22. СН 4088-86. Санітарні норми мікроклімату виробничих приміщень URL: http://docs.cntd.ru/document/901710059 (дата звернення: 22.05.2022).

23. Санітарно – гігієнічним нормам №2152-80 URL: https://dnaop.com/html/2296/doc -ГН_2152-80 (дата звернення: 22.05.2022).

24. СН 4557-88. Санітарні норми ультрафіолетового випромінювання URL: https://dnaop.com/html/2299/doc -СН_4557-88 (дата звернення: 22.05.2022).

ДОДАТОК А

Асинхронні методи (thunk) для взаємодії з базою даних

```
import { errorsActions } from "../errors/errorsSlice";
import firestore from "@react-native-firebase/firestore";
import { articlesActions } from "../articles/articlesSlice";

export const fetchArticles = () => dispatch => {
  const onError = error => {
    dispatch(errorsActions.setError(error?.message ?? error));
  };

  const onResult = querySnapshot => {
    const articlesArr = [];

    querySnapshot.forEach(documentSnapshot => {
      const data = documentSnapshot.data();
      articlesArr.push({
        id: documentSnapshot.id,
        authorId: data.authorId,
        imageUrl: data.imageUrls[0],
        imageUrls: data.imageUrls.filter((_, index) => index !== 0) || [],
        header: data.header,
        body: data.body,
        tags: data.tags,
        timestamp: data.timestamp,
        location: data.location,
      });
    });
```

```
    dispatch(articlesActions.setArticles(articlesArr));
  };

  firestore()
    .collection("reports")
    .orderBy("timestamp", "desc")
    .onSnapshot(onResult, onError);
};

export const createArticleAsync =
  (userId, header, body, imageUrls, tags, timestamp, location) =>
  async dispatch => {
    dispatch(articlesActions.toggleIsArticlesLoading());

    try {
      await firestore().collection("reports").add({
        authorId: userId,
        header,
        body,
        imageUrls,
        tags,
        timestamp,
        location,
      });
    } catch (error) {
      dispatch(errorsActions.setError(error?.message ?? error));
    }
```

```
  dispatch(articlesActions.toggleIsArticlesLoading());
 };

export const deleteArticleAsync = articleId => async dispatch => {
 dispatch(articlesActions.toggleIsArticlesLoading());

 try {
  await firestore().collection("reports").doc(articleId).delete();
 } catch (error) {
  dispatch(errorsActions.setError(error?.message ?? error));
 }

 dispatch(articlesActions.toggleIsArticlesLoading());
};
```

ДОДАТОК В

Головний навігатор (MainNavigator.js)

```
import React, { useState, useEffect } from "react";
import { View, StyleSheet, ActivityIndicator } from "react-native";
import { NavigationContainer } from "@react-navigation/native";
import { useSelector } from "react-redux";
import HomeNavigator from "./navigators/HomeNavigator";
import AuthNavigator from "./navigators/AuthNavigator";
import ErrorModal from "../components/UI/ErrorModal";
import auth from "@react-native-firebase/auth";
import colors from "../constants/colors";
import { useDispatch } from "react-redux";
import { setUserDataAsync } from "../store/auth/authThunks";

const MainNavigator = props => {
  const dispatch = useDispatch();
  const { error, notification } = useSelector(state => state.errors);
  const [initializing, setInitializing] = useState(true);
  const [user, setUser] = useState({});
  const handleAuthStateChanged = async user => {
    setUser(user);
    if (user) {
      await dispatch(
        setUserDataAsync(user.uid, user.displayName, user.email, user.photoURL),
      );
    }
    if (initializing) {
      setInitializing(false);
```

```
    }
  };
  useEffect(() => {
    const subscriber = auth().onAuthStateChanged(handleAuthStateChanged);
    return subscriber;
  }, []);
  if (initializing)
    return (
      <View style={styles.centered}>
        <ActivityIndicator size="large" color={colors.secondary.main} />
      </View>
    );
  return (
    <NavigationContainer>
      {!user && <AuthNavigator />}
      {!!user && <HomeNavigator />}
      {(!!error || !!notification) && <ErrorModal />}
    </NavigationContainer>
  );
};
const styles = StyleSheet.create({
  centered: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
  },
});
export default MainNavigator;
```