Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК 004.41 На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: “Проектування та програмна реалізація мобільного програмного забезпечення для розв’язання типових задач лінійної алгебри”

Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА
ВКБР.ІПЗ - 22.00.00.000 ІПЗ

Студент
ІПЗ-42 **Олексій КОЛЮЧИЙ**

Науковий керівник
к.ф.-м.н., доц. **Оксана КОВТУН**

Консультант з питань нормоконтролю
Тамара ЧАПОВСЬКА

Допускається до захисту
Зав. каф.
д.т.н.,проф. **Олексій БИЧКОВ**

Київ – 2021

Рішенням Екзаменаційної комісії

випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії

професор, доктор техн. наук Бондарчук А.П.

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО
Зав. кафедри програмних систем і технологій
_____ (Олексій БИЧКОВ)

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Колючому Олексію Олександровичу

1. Тема випускної кваліфікаційної бакалаврської роботи “Проектування та програмна реалізація мобільного програмного забезпечення для розв’язання типових задач лінійної алгебри”

керівник проекту (роботи) Ковтун Оксана Іванівна, к.ф.-м.н., доцент

затверджені наказом вищого навчального закладу від „11” листопада 2021 р. № 6

2. Строк здачі студентом закінченої роботи „__” _____ 2021 р.

3. Вхідні дані до проекту (роботи): офіційна документація, статті зарубіжних авторів, інтернет-ресурси

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

- Аналіз предметної області

- Огляд технологій

- Реалізація програмного комплексу

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)

- Рис. 2.2.3.1 (Блок-схема реалізації алгоритму Flux), Ст. 32

- Рис. 3.2.1 (Модульна архітектура додатку)), Ст. 43

- Рис. 3.3.1 (Діаграма обміну даними у додатку)), Ст. 45

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1. Аналіз предметної області	Ковтун О.І.	10.11.2020	20.01.2021
Розділ 2 Огляд технологій	Ковтун О.І.	25.01.2021	02.03.2021
Розділ 3. Реалізація програмного комплексу	Ковтун О.І.	02.03.2021	20.05.2021

7. Дата видачі завдання _____

Керівник Оксана КОВТУН_____ _____

Завдання прийняв до виконання Олексій КОЛЮЧИЙ_____ _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Відмітка про виконання
1.	Уточнення постановки задачі	10.11.2020-02.12.2020	Виконано
2.	Аналіз літератури	02.12.2020 - 11.01.2021	Виконано
3.	Огляд та аналіз існуючих методів, концепцій та алгоритмів вирішення завдання	11.01.2021 -25.01.2021	Виконано
4.	Побудова алгоритмічної моделі основних процесів	25.01.2021 -02.03.2021	Виконано
5.	Розробка програмного забезпечення	02.03.2021 - 05.04.2021	Виконано
6.	Тестування розробленого програмного забезпечення	05.04.2021 -20.05.2021	Виконано
7.	Оформлення і друк пояснювальної записки	20.05.2021 -01.06.2021	Виконано
8.	Отримання рецензії	02.06.2021 -05.06.2021	Виконано
9.	Оформлення презентації	05.06.2021- 09.06.2021	Виконано

10.	Затвердження пояснювальної записки роботи виконувачем обов'язки завідувача кафедри	_____	
11.	Захист дипломної роботи	22.06.2020	

Студент-бакалавр Олексій КОЛЮЧИЙ_____ _____

Керівник роботи Оксана КОВТУН_____ _____

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JS – Javascript

RN – React Native

ПЗ – Програмне Забезпечення

ОС – Операційна система

МД – Мобільний додаток

АНОТАЦІЯ

У рамках завдання на випускну кваліфікаційну бакалаврську роботу проаналізовано сучасні технології, тенденції та архітектурні шаблони що використовуються для реалізації мобільного програмного забезпечення.

Випускна кваліфікаційна робота: 65 сторінки, 3 рисунки, 7 додатків та 8 джерел використаних в роботі.

Тема: Проектування та програмна реалізація мобільного програмного забезпечення для розв'язання типових задач лінійної алгебри.

Об’єкт дослідження: основні проблеми лінійної алгебри, методики їх розв'язання та процес розробки мобільного додатку.

Мета роботи: забезпечення студентів та викладачів закладів вищої освіти інструментарієм для проведення розрахунків в області лінійної алгебри.

Предмет дослідження: типові задачі з лінійної алгебри та мобільне програмне забезпечення для їх розв’язання.

Результати дослідження: досліджено типові задачі лінійної алгебри та способи їх розв’язання. Розглянуто основні методики розробки мобільного програмного забезпечення. Запропоновано алгоритмічно-оптимізований підхід до виконання обчислень розрахований на особливості вимог та ресурсів мобільних пристроїв.

Висновок: програмно реалізовано кросплатформений мобільний додаток для розв’язку типових задач лінійної алгебри.

АНОТАЦИЯ

В рамках задания на выпускную квалификационную бакалавскую работу проанализировано современные технологии, тенденции и архитектурные шаблоны для реализации мобильного програмного обеспечения.

Выпускная квалификационная работа: 65 страниц, 3 рисунка, 7 дополнений и 8 источников использованных в работе.

Тема: Проектирование и программная реализация мобильного программного обеспечения для решения типичных задач линейной алгебры.

Объект исследования: основные проблемы линейной алгебры, методики их решения и процесс разработки мобильного приложения.

Цель работы: обеспечение студентов и преподавателей высших учебных заведений инструментарием для проведения расчетов в области линейной алгебры.

Предмет исследования: типичные задачи по линейной алгебре и мобильное программное обеспечение для их решения.

Результаты исследования: исследованы типичные задачи линейной алгебры и способы их решения. Рассмотрены основные методики разработки мобильного программного обеспечения. Предложено алгоритмически-оптимизированный подход к выполнению вычислений рассчитан на особенности требований и ресурсов мобильных устройств.

Вывод: программно реализовано кроссплатформенных мобильное приложение для решения типичных задач линейной алгебры.

ANNOTATION

As part of the assignment for the final qualifying bachelor's work, modern technologies, trends and architectural patterns for the implementation of mobile software were analyzed.

Final qualifying work: 65 pages, 3 figures, 7 additions and 8 sources used in the work.

Topic: Design and software implementation of mobile software for solving typical linear algebra problems.

Object of research: the main problems of linear algebra, methods of solving them and the process of developing a mobile application.

Purpose of work: providing students and teachers of higher education institutions with tools for calculations in the field of linear algebra.

Research subject: typical linear algebra problems and mobile software for solving them.

Research results: typical problems of linear algebra and methods of their solution are investigated. The main methods of mobile software development are considered. An algorithmic-optimized approach to computing is designed for the specific requirements of the resources of mobile devices.

Conclusion: software implemented cross-platform mobile application for solving typical problems of linear algebra.

ВСТУП

В наші часи, роль мобільного телефону в житті сучасної людини досить складно переоцінити. Завдяки цьому компактному пристрою, можна в будь-який час зв'язатися з друзями, родичами, колегами для того, щоб отримати необхідну інформацію. Крім контактів, багато людей зберігають на своїх мобільних пристроях пам'ятні дати, ідеї, думки, та інші файли різного характеру.

Кілька десятиліть тому, на початку свого становлення, мобільні технології були ще не такими і мобільними. Гордим і вельми забезпеченим власникам була потрібна непогана фізична форма, щоб досить комфортно оперувати важкими і громіздкими гаджетами. Крім того, потрібно було мати доступ до електричної розетки щоб користуватися першими мобільними пристроями, оскільки акумулятори в них ще не вбудовували.

З тих пір технології просунулися неймовірно далеко, причому особливо великий ріст відбувся за останні десять років. Мобільні пристрої стали менше, могутніше і набагато корисніше. Вони проникли в усі сфери нашого життя, і їх роль продовжує зростати. Доступність всіляких смартфонів, планшетів, електронних книг, розумних годинників сприяє їх швидкому поширенню по всьому світу. І, звісно, всі ці мільярди мобільних пристроїв роблять серйозний вплив на якість нашого життя.

Молодим людям вкрай важливо завжди залишатися на зв'язку, в тому числі це стосується і студентів, які за допомогою мобільного телефону спілкуються зі своїми родичами і друзями, а також вирішують питання, пов'язані з навчанням.

Актуальність роботи

Кінцевими споживачами розробленого під час виконання бакалаврської роботи мобільного програмного забезпечення є студенти та викладачі закладів вищої освіти. З розвитком мобільної індустрії студенти все частіше звертаються до використання своїх мобільних пристроїв для спрощення процесу навчання, та перевірки своїх результатів. Особливо це актуально у наш час – час пандемії, коли більшість навчальних процесів проходять за допомогою використання технологій дистанційного навчання. Саме сучасні тенденції і спонукають до створення спецалізованого програмного забезпеення, яке спростить та розширить можливості навчальних процесів.

Створення подібного ПЗ – це крок до покращення навчання та її єффективності. За допомогою розробленого програмного забезпечення студенти та викладачі можуть швидко та єффективно перевіряти результат типових задач з лінійної алгебри, а саме найпопулярніші з них – операції з матрицями.

Лінійна алгебра працює з векторами і матрицями - а точніше, з їх лінійними комбінаціями, які також є векторами і матрицями. Математично вектор можна представити набором дійсних чисел. Лінійна алгебра доволі часто зустрічається у освітніх програмах та дуже активно використовується у Data Science та при розробці штучного інтелекту.

Мета і задачі дослідження

Метою бакалаврської роботи є створення інструменту проведення розрахунків типових задач лінійної алгебри для студентів та викладачів закладів вищої освіти.

Кінцеве програмне забезпечення повинен надавати можливості для вирішення типових задач з лінійної алгебри, бути інтуітивно зрозумілим, мати простий та не перевантаженний інтерфейс, що дасть можливість користувачам швидко отримувати необхідні їм результати за допомогою власного мобільного пристрою. Також ПЗ повинно мати підтримку двох найпопулярніших мобільних

ОС, а саме IOS та Android. Іншою важливою вимогою є можлива розширюванність можливостей розробленого ПЗ.

Для досягнення мети бакалаврської роботи було виконано наступне:

- Аналіз типових задач з лінійної алгебри;
- Аналіз алгоритмів розв’язання типових задач з лінійної алгебри;
- Дослідження сучасних середовищ розробки мобільного ПЗ;
- Вибір мови програмування для програмної реалізації ПЗ;
- Вибір фреймворків для програмної реалізації.

Об'єктами дослідження є основні проблеми лінійної алгебри та методики їх розв'язання та процес розробки мобільного додатку.

Предметами дослідження є типові задачі з лінійної алгебри та мобільне ПЗ для їх розв’язання.

Методи дослідження

Для реалізації мобільного додатку було використано методи аналізу та експерименту.

Новизна одержаних результатів

Було досліджено використання алгоритмів для розвязання типових задач лінійної алгебри та удосконалено взаємодію студентів і викладачів з вирішенням та перевіркою отриманих результатів задач лінійної алгебри.

Практичне значення одержаних результатів

Розроблений додаток для мобільних платфом може бути застосован для розвязання типових задач лінійної алгебри. Додаток може розпоширюватися серед користувачів на платформах IOS та Android за допомогою наступних сервісів: Play Market та App Store. Застосунок не потребує зєднання з мережею Інтернет, та не вимагає реєстрації для користування. Також має зручний, мінімалістичний та зрозумій інтерфейс.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз видів мобільного програмного забезпечення

Всі мобільні додатки поділяють на 3 види:

1.1.1. Нативні додатки

Нативну (native) розробку можна назвати «рідний» для операційних систем - Android, IOS, Win Phone і т.д. Такі мобільні додатки пишуться на мовах програмування, затверджених розробниками програмного забезпечення під кожну конкретну платформу, а тому органічно вбудовуються в самі операційні системи. Програми завантажуються через магазини додатків (App Store, Google Play і т.д.) і відповідають вимогам цих магазинів.

Головна перевага нативних додатків - то, що вони оптимізовані під конкретні операційні системи, а значить працюють коректно і швидко. Також вони мають доступ до апаратної частини пристроїв, тобто можуть використовувати в своєму функціоналі камеру смартфона, мікрофон, акселерометр, геолокацію, адресну книгу, плеєр і т.д. Можна налаштувати отримання push-повідомлень. Ще один плюс - економну витрату ресурсів телефону (батарея, пам'ять).

Нативні додатки можуть повністю або частково працювати і при відсутньому інтернет-з'єднанні, тому користувачі менш залежать від якості зв'язку та можуть користуватися додатком там і тоді, коли їм це зручно.

Зрозуміло, написання такого продукту вимагає від розробника володіння спеціальними знаннями і вміннями для роботи в конкретному середовищі розробки (xCode для iPhone, eclipse для пристроїв на Android). Як наслідок

вартість таких додатків набагато вище в силу їх трудомісткості і того, що під кожну платформу доводиться писати окремий додаток на іншій мові.

1.1.2. Гібридні додатки

Гібридні мобільні додатки дозволяють створювати кросплатформені додатки наближені по функціоналу і якості до нативним додатків. Це щось середнє між нативними і веб-додатками. Такі додатки встановлюються через офіційні магазини, мають обмежений доступ до апаратної частини смартфонів і планшетів, в них можна налаштовувати push-повідомлення. А також вони, як правило, дешевші за нативні додатків.

Якість і можливості гібридних додатків залежать від самого фреймворка, яким користувався розробник. Є дешевші і дорожчі варіанти.

1.1.3. Веб-додатки

По суті, це мобільна версія сайту тільки з розширеним інтерактивом. Але різниця між веб-додатком і адаптивної верстkою сайту не велика, оскільки і там і там застосовуються стандартні веб-технології, а швидкість роботи обмежена якістю інтернет-з'єднання. При цьому веб-додатки не розміщуються в спеціалізованих магазинах додатків і зазвичай використовують браузер телефону для роботи.

Буває, що такий додаток навіть скачується через офіційні магазини і має свою іконку на екран смартфона. Однак для користувача завжди буде очевидно, що його якість не відповідає повноцінної нативной розробці. Термін життя таких додатків на пристроях користувачів не великий, а відгуки найчастіше негативні. Коли мова йде про імідж компанії, то вже краще не мати додаток зовсім, ніж таке, яке відверне клієнтів від користування послугами даної компанії.

Є ще пара нюансів - веб додатки не завжди безпечні, оскільки не можуть шифрувати файлову систему, а якщо в подальшому будуть потрібні оновлення та розширення функціоналу, то доведеться писати додаток заново.

1.2. Висновки до розділу

Проаналізувавши види сучасних мобільних додатків можна дійти до висновку що гібридне та нативне ПЗ має найбільшу ефективність. Кожен з цих видів має свої переваги та недоліки. Для реалізації програмного забезпечення для вирішення типових задач лінійної алгебри було обрано суміш нативного та гібридного додатку, адже це дасть можливість створити кросплатформене та ефективне програмне забезпечення.

РОЗДІЛ 2

ОГЛЯД ТЕХНОЛОГІЙ

2.1. Аналіз технологій що використовувались

Під час розробки в першу ж чергу було вирішено які технології і мову програмування використовувати для програмної реалізації ПЗ. Розглянемо найпопулярніші:

Java

З моменту появи Java стала основною мовою для розробки мобільних додатків на Android. Він забезпечує крос-платформену підтримку. Крім того, додатки на Java легко перенести на різні операційні системи. Програми Java працюють за принципом «Написано один раз, запускається всюди» (WORA - Write Once Run Anywhere) - вони будуть працювати однаково на будь-якому сумісному з Java пристрої без необхідності зміни коду. І хоча Java - відносно старий мову програмування, він зберігає популярність.

Kotlin

Kotlin - це нова мову програмування, повністю сумісний з Java. Ці дві мови взаємозамінні. У минулому році Google назвав Kotlin «основною мовою для розробки додатків на Android». Серед переваг в порівнянні з Java варто відзначити масштабованість Kotlin.

Swift

Swift - мова програмування, розроблений Apple як сучасна заміна Objective-C, який раніше використовувався для створення додатків на iOS. Спочатку Swift призначався для розробки на iOS, але тепер його можна

використовувати для розробки додатків для macOS, Windows і Linux. Також доступні і неофіційні інструменти для додавання підтримки Android.

Rust

Rust - відносно нова мова, яка вже стала відомою своїми можливостями управління пам'яттю і безпекою. Як і Java, Rust має крос-платформену підтримку і може використовуватися для розробки мобільних додатків на Android, iOS, Windows, macOS, Linux і для ряду різновидів Unix. Rust підходить для розробки нативних і веб-додатків, а також операційних систем, компонентів браузера і ігрових движків.

2.1.1. React Native

Хоча React Native (RN) не є мовою програмування, але саме за допомогою цього, нині дуже стрімко набираючого оберти та популярність, фреймворку до JavaScript зараз існує безліч мобільних кросплатформених додактів, та можливостей до їх нативних реалізацій. [1]

Для розробки ПЗ було проаналізовано сучасні технології для реалізації мобільниого ПЗ та обрано мову JS на фреймворку React Native, адже це молода сучасна технологія, що дозволяє зручно розробляти кроспалформені нативні мобільні додатки.

Розглянемо плюси та мінуси RN.

2.1.2. Плюси React Native:

- Економічна ефективність

React Native надає розробникам недорогий спосіб створення кроссплатформенних додатків за допомогою React Native. Замість того щоб створювати два різних додатки для Android і iOS, розробник може використовувати один і той же код для обох платформ. Це означає скорочення витрат на розробку приблизно на 50%. Ця функція також робить React Native дешевше в обслуговуванні. [9]

- Використання JavaScript

Згідно популярному сайту Statista, JS є самим широко використовуваним мовою програмування в світі. Згідно з дослідженням Statista, 68% розробників пишуть на JavaScript. Це дозволяє розробникам з досвідом на JavaScript легко використовувати React Native, оскільки фреймворк написаний на JavaScript. [5]

- Не потребує великої команди розробників

Стандартний проект розробки додатків потребує двох командах розробників для створення Android і iOS версії. Це часто призводить до нерозуміння і неузгодженості роботи двох команд. У деяких випадках зовнішній вигляд і функції програми в кінцевому вигляді не будуть однаковими на обох платформах. Використання React Native дає перевагу в створенні додатків для Android і IOS, адже для цього потрібна всього одна команда. Було б корисно мати досвідченого нативного розробника серед членів команди. І дві різні команди розробників більше не потрібні. [10]

- Перевага відкритого вихідного коду

Дивлячись на той факт, що React Native- це ліцензована MIT платформа з відкритим вихідним кодом, вона дає розробникам доступ до використання бібліотек і фреймворків безкоштовно. Це накладає певні обмеження на повторне використання програмного забезпечення, але також забезпечує правовий захист для розробників.

- Активне товариство розробників

Сайт Statista показує, що React Native-найпопулярніша на сьогоднішній день платформа для розробки кроссплатформенних додатків з часткою ринку 42% і кількістю завантажень понад 90 тисяч на Github. Це не дивно, тому що платформа забезпечує величезну вигоду для розробників мобільних додатків.

- Відмінна продуктивність

Нативні додатки дійсно мають більш високу продуктивність, але додатки, створені на React Native, також показують вражаючу продуктивність, порівнянну з нативними додатками. Завдяки вбудованим мобільним пристроям і елементам управління в додатках React Native, які використовують власні компоненти ОС, щоб без проблем створювати коди для власного API. [12]

- Модульний дизайн

React Native використовує технологію модульного програмування, в якій функції реалізовані у вигляді окремих блоків, званих модулями. Цей підхід забезпечує гнучку середу розробки додатків, а також покращує взаємодію між розробниками. Він дозволяє легко створювати і інтегрувати оновлення програм за допомогою простих у використанні модулів, які можна повторно використовувати як для WEB, так і для мобільних API. [11]

2.1.3. Мінуси React Native:

- Проблеми налагодження і сумісності

Незважаючи на всі чудові функції, React Native все ще є бета-версією. Ось чому у неї все ще є деякі очевидні проблеми, як, наприклад, складність налагодження додатків, а також обмеження, включаючи проблеми сумісності. Майте на увазі, що робота з React native може бути проблематичною, коли виникає необхідність в налагодженні.

- Залежність від Facebook

Той факт, що React Native був розроблений Facebook, вражає, але одночасно є одним з недоліків платформи. Припустимо, що Facebook перестане надавати резервну копію платформи, тоді ви вже не зможете її більше використовувати. Але зараз багато інших платформи пропонують ті ж функції, що і React Native.

- Управління Пам'яттю знаходиться на низькому рівні

React Native створює додатки з відмінними функціями. Однак React Native може виявитися не найкращою платформою для створення додатків, ефективно керуючих апаратними ресурсами.

Оскільки управління пам'яттю в React Native не знаходиться на відстані, то не варто вибирати цю платформу для створення високопродуктивних додатків. Якщо вам потрібно створити додаток, яке повинно робити складні обчислення, вам краще шукати альтернативну платформу розробки.

- Проблеми безпеки JavaScript

JavaScript добре відомий своєю низькою безпекою. Саме тому JavaScript не використовується для створення додатків, які повинні забезпечувати першокласну безпеку. Незважаючи на це, React Native використовує JavaScript для основних задач розробки. У світлі цього React Native слід використовувати для розробки проектів, де безпека не має першорядного значення. В якості альтернативи слід самим забезпечувати захист першочергово важливих даних, таких як платіжна інформація. [6]

- Сторонні Компоненти Розробки

Компоненти, необхідні для створення деяких видів додатків, відсутні в ReactNative. Таким чином, ви повинні використовувати сторонні ресурси для додавання цих компонентів в вашу програму. У той час як використання сторонніх компонентів є можливим, ви можете втратити на цьому багато часу і сил. У таких випадках вам доведеться знайти спосіб вирішити проблему самостійно. Ось чому складно використовувати сторонні компоненти в проектах розробки додатків.

React Native використовує JavaScript фреймворк React для роботи, детальніше концепцію та основи React ми розглянемо трохи пізніше.

Після вибору фрейморку та мови програмування слід розгянути основні допоміжні фрейморки що допоможуть реалізувати потрібний функціонал та архітектуру мобільного додатку. Розглянемо ж їх.

- React
- Redux
- React-redux

Давайте проглянемо кожну з них більше детально.

2.1.3.1. React

React JS - це бібліотека JavaScript, яка використовується в веб-розробці клієнтської сторони для створення інтерактивних елементів на веб-сайтах.

Якщо ви раніше не малу справу з бібліотеками JavaScript, то давайте розглянемо що таке JS бібліотека трішки детальніше.

JavaScript відіграє вирішальну роль у розробці веб-сайтів та веб-додатків. Але бувають випадки, коли для виконання функцій, що повторюються, вам потрібні такі речі, як ефекти анімації або функції автозаповнення рядка пошуку.

Звісно, кодувати ці функції кожного разу за потребою стає ситуацією "винахід колеса". Тут на допомогу розробникам і приходять бібліотеки JavaScript.

React - це бібліотека JavaScript, яка спеціалізується на допомозі розробникам у створенні інтерфейсів користувача. Що стосується веб-сайтів та веб-додатків, то користувальницькі інтерфейси - це набір екранних меню, панелей пошуку, кнопок і всього іншого, з ким користується веб-сайт чи додаток. [2]

Давайте приведемо декілька основних переваг React JS, які говорять в користь використання цієї бібліотеки в сучасних веб-додатках та сайтах.

JSX – основою будь-якого базового веб-сайту є HTML-документи. Веб-браузери читають ці документи та відображають їх на комп’ютері, смартфоні чи інших платформах у вигляді веб-сторінок. Під час цього процесу браузери

створюють DOM (Document Object Model), що являє собою дерево з інформацією про те, як влаштовано веб-сторінку.

JSX (JavaScript extension) - розширення React, яке дозволяє веб-розробникам змінювати свій DOM за допомогою простого коду в HTML стилі. Оскільки підтримка браузером React поширюється на всі сучасні веб-браузери, JSX сумісний з будь-яким браузером та його платформою.

Хоча використання JSX є дуже зручним, таке розширення також має важливу роль для оптимізації оновлення DOM. Давайте розглянемо як саме JSX впливає на оптимізацію DOM – Virtual DOM.

Virtual DOM - якщо ви не використовуєте React JS (та JSX) або декілька інших бібліотек/фреймворків з інтегрованим Virtual DOM, ваш веб-сайт використовує HTML, щоб оновити DOM. Це чудово працює для простих, статичних веб-сайтів, але для динамічних веб-сайтів, які передбачають важку взаємодію з користувачем, це може стати проблемою (оскільки весь DOM потребує перезавантаження кожного разу, коли користувач натискає функцію, яка вимагає оновлення сторінки).

Однак якщо розробник використовує JSX для маніпулювання та оновлення своєї DOM, React JS створює Virtual DOM. Virtual DOM - це копія DOM сайту, і React JS використовує цю копію, щоб побачити, які частини фактичного DOM потрібно змінити, коли подія відбувається (наприклад, користувач натискає кнопку).

Отже, Virtual DOM у відмінності від DOM, не змінюється повністю після кожної дії, а створює віртуальну копію нового DOM, та змінює тільки ті частини, які були дійсно були змінені.

2.1.4. Redux

Redux – це бібліотека для JavaScript яка створена для реалізації Flux-архітектури у веб-програмах.

Redux - передбачуваний контейнер стану для додатків JavaScript. Це допомагає писати програми, які ведуть себе послідовно та запускаються в різних середовищах (клієнт, сервер та рідне місце). [3]

Простіше кажучи, Redux - це інструмент управління даними та станом у додатку. Хоча він використовується в основному з React, він може використовуватися з будь-якою бібліотекою в рамках мови JavaScript. Також слід відмітити, що розробники серйозно віднеслись до оптимізації під час розробки, так як Redux важить всього 2 КБ разом зі всіма залежностями. Тому турбуватися про збільшення розміру додатку не варто.

За допомогою Redux стан вашої програми зберігається в глобальному сховищі, і кожен компонент може отримати доступ до будь-якого стану, який йому потрібен з цього сховища. Давайте зануримось трохи глибше, щоб зрозуміти, чому вам може знадобитися інструмент управління станом та даними.

Більшість бібліотек, таких як React, Angular, побудовані так, щоб компоненти могли керувати своїм станом внутрішньо, не потребуючи зовнішньої бібліотеки чи інструменту. Це добре для додатків з малою кількістю компонентів, але в міру того, як програма збільшується, управління станами, що поділяються між компонентами, стає нелегкою справою.

У додатку, де дані діляться між компонентами, не завжди зрозуміло, де саме стан модулів повинен зберігатися. В ідеалі дані в компоненті повинні жити лише в одному модулі, тому обмін даними між компонентами інколи стає важким. [7]

Наприклад, у React, для обміну даними між батьківськими та дочірніми компонентами, стан повинен бути в батьківському компоненті. Метод оновлення цього стану забезпечується батьківським компонентом і передається як властивості компонентам нижчим по ієрархії.

Те, як працює Redux - просто. Є центральне сховище, яке вміщує весь стан програми. Кожен компонент може отримати доступ до збереженого стану без необхідності надсилати запити зі станом з одного компонента в інший.

Є три складові частини: дії (actions), сховище (store), та редюсери (Reducers). Це основні принципи Redux та Flux, які ми вже розглянули у минулих пунктах.

Redux являє собою програмні утиліти для полегшення реалізації Flux-архітектури практично на у проектах.

Actions в Redux

Кажучи простими словами – це JavaScript об’єкти.

Це події та єдиний спосіб, яким ви можете надсилати дані зі своєї програми у ваше сховище Redux. Дані можуть надходити з взаємодії користувачів, викликів API або навіть з подання форми.

Дії надсилаються методом store.dispatch (метод сховища). Як вже було відмічено, Actions – це об’єкти JavaScript. Вони обов’язково повинні мати тип дії, який передає інструкції до редюсерів, та можуть мати навантаження – це дані, що можуть передаватися в об’єкт для майбутньої обробки цієї інформації.

Приклад програмної реалізації за допомогою Redux можна побачити в Додатку А.

Reducers в Redux

Редюсери - це чисті функції, які приймають поточний стан програми, виконують дію та повертають новий стан. Ці стани зберігаються як об'єкти, і вони визначають, як змінюється стан програми у відповідь на дію, надіслану до сховища.

Редюсер заснований на функції reduce в JavaScript, де одне значення обчислюється з декількох значень після того, як була проведена функція зворотного виклику.

Приклад програмної реалізації редюсера можна побачити в Додатку В.

Store в Redux

Сховище – це і є центральна особливість Fux-архітектури.

У сховищі зберігається стан всієї програми. У будь-якому додатку що використовує Redux є сховище. Ви можете отримати доступ до збереженого стану, оновити стан, а також зареєструвати або скасувати реєстрацію підписників за допомогою допоміжних методів.

Дії, що виконуються над сховищем, завжди повертають новий стан. Тобто зміни відбуваються не напряму з активним і актуальним сховищем, а створюється та змінюється його копія, після чого нею заміняється актуальне сховище. Таким чином, стан дуже легкий і передбачуваний.

В Redux в сховищі є один загальний стан, і кожен компонент має доступ до цього стану. Це виключає необхідність постійного переходу стану від одного компонента до іншого.

Використовуючи Redux з React, стан більше не потрібно буде піднімати батьківським компонентам; таким чином, вам простіше простежити, яка дія викликає які зміни. Як було сказано вище, компонент більше не потребує надання будь-якого стану або методів для своїх дочірніх компонентів для обміну даними між собою. Все обробляє Redux. Це значно спрощує додаток і полегшує його обслуговування, та читабельність.

2.1.5. React-Redux

Ми розглянули та розібралися з тим що уявляють собою React та Redux, але в цьому зв’язку повинна бути з’єднуюча ланка.

Сам Redux - це окрема бібліотека, яку можна використовувати з будь-яким інтерфейсом користувача, включаючи JS React, React Native, Angular, Vue, Ember та інші. Хоча Redux і React зазвичай використовуються разом, вони не залежать один від одного.

Якщо ви використовуєте Redux з будь-якою реалізацією UI-інтерфейсу, ви зазвичай використовуєте бібліотеку "прив'язки інтерфейсу", щоб зв'язати Redux

разом із інтерфейсом, а не безпосередньо взаємодіяти зі сховищем зі свого коду інтерфейсу. [4]

React-Redux є офіційною бібліотекою прив’язки Redux та UI для React. Якщо ви використовуєте Redux та React разом, вам також слід використовувати React-Redux для зв’язування цих двох технологій.

Звісно, інженер може сам написати та реалізувати логіку підписки на сховище Redux, але це також стане затратним по часу винайденням велосипеду. Крім того, для оптимізації продуктивності інтерфейсу потрібна складна логіка.

Процес підписки на сховище, перевірка та оновлених даних, запуск повторного візуалізації можуть бути більш загальними та багаторазовими. Бібліотека прив'язки інтерфейсу користувача, як React-Redux, обробляє логіку взаємодії сховища, тому вам не доведеться писати цю реалізацію самостійно.

В цілому, React-Redux заохочує хорошу архітектуру Redux та впроваджує складні оптимізації за розробника. Він також оновлюється останніми змінами в різних API від Redux та React. [8]

2.2. Архітектура мобільного додатку

Добре пророблена архітектура потрібна всім додаткам, і складним, і шаблонним. З її допомогою заощаджується час, зусилля і гроші.

Архітектура мобільних додатків - сукупність рішень, як організувати програму. У неї входять: структурні елементи і інтерфейси, зв'язку між обраними елементами, загальний стиль програми.

Гарна архітектура означає вигоду: простота і ефективність. Програму з такою архітектурою легше змінювати, тестувати і налагоджувати. Як зрозуміти, чи гарна архітектура у вашого застосування? [13]

2.2.1. Вимоги до архітектури:

- ***Ефективність*:** додаток виконує поставлені завдання і виконує функції в будь-яких умовах. Система продуктивна, надійна і справляється з усіма навантаженнями;
- ***Гнучкість*:** вибране рішення легко змінювати, і помилок стає менше. Можна змінити один елемент, і це не стане фатальним для інших складових.
- ***Можливість розширення*:** в додаток можна додавати скільки завгодно функцій, якщо буде потрібно.
- ***Масштабованість*:** час на розробку і доповнення зменшується. Гарна архітектура дозволяє направити розробку в декілька паралельних потоків.
- ***Тестування*:** додаток легко тестується, а значить, зменшується кількість помилок і збільшується його надійність.
- ***Повторне використання*:** елементи і структуру можна використовувати в інших проектах.
- ***Зрозумілість*:** код повинен бути зрозумілий як можна більшій кількості людей. [14]

2.2.2. Передумови

В сучасному світі, технології для розробки МД досить сильно відрізняються один від одного, з чого випливає що різні технології, як і архітектурні шаблони, залежать від інших ‘сусідніх’ технологій що ми використовуємо в проектах.

Основою для реалізації архітектури програми став Redux. Redux – це JavaScript реалізація Flux-архітектури, що була обрана для розробки.

2.2.3. Про Flux

Проаналізувавши сучасні архітектурні шаблони, було обрано Flux-архітектуру.

Flux - це архітектура, яку компанія Facebook використовує під час роботи з JS бібліотекою React. Але Flux це не бібліотека або утиліта. Це новий вид архітектури, який доповнює React та концепцію односпрямованого потоку даних.

При цьому Facebook надає репозиторії, які включають бібліотеку диспетчерів. Диспетчер - це свого роду глобальний сигнал, який транслює корисну програмі інформацію про зміни та дії користувача, або описує поведінку програми.

Типова архітектура Flux використовує цю диспетчерську бібліотеку разом з модулем NodeJS для створення системи подій, яка допомагає керувати станом додатків.

Flux, краще пояснюється, пояснюючи його окремі компоненти:

Actions - Допоміжні методи, що полегшують передачу даних диспетчерам. Можуть вміщувати та не вміщувати дані для передачі їх до редюсерів. Це сукупність методів, які викликаються у View-модулях для надсилання дій Диспетчеру. Це фактичні корисні навантаження, які доставляються через диспетчер. Мають свій тип та можуть мати навантаження.

Dispatcher - Отримує дані та передає їх до редюсерів. По суті, диспетчер є керівником всього цього процесу. Це центр для вашої програми. Диспетчер отримує дії та розсилає дії та дані до зареєстрованих зворотніх викликів (callback function).

Розглянувши інформацію вище, можна задати закономірне питання – так не “видавець” чи “підписник” це часом?

Не зовсім. Диспетчер передає корисне навантаження (набір даних) всім своїм зареєстрованим зворотним викликам і включає функціональність, яка дозволяє викликати зворотні виклики у визначеному порядку, навіть чекаючи оновлень, перш ніж продовжувати.

Facebook використовує їх для визначення того, яка дія має відбуватися. Всередині зареєстрованих зворотних викликів ці дії тепер можна обробляти відповідно до їх типів.

Reducers - контейнери для стану та логіки додатків, які відповідають за прийняття сигналу (дії) та обробку інформації що вони передають до глобального сховища. Це чисті функції, тобто функції які на один й той самий набір вхідних даних завжди дадуть один й той самий набір вихідних даних. Редюсери відповідають за отримання сигналу від Action та в залежності від нього змінюють дані в сховищі.

View Controller - React компонент, що за запитами бере інформацію зі сховища, та надсилає цю інформацію до компонентів. Вони слухають зміни подій та отримують стан зі сховища. Потім вони передають ці дані своїм дочірнім компонентам через їх властивості.

Store (глобальне сховище) – у Flux сховища керують станом програми у вашій програмі. З високого рівня це в основному означає, що в сховищі зберігаються керовані дані, методи пошуку даних та зворотні виклики диспетчера.

Тепер, коли ми переглянули кожну окрему частину архітектури Flux, ми повинні мати набагато краще уявлення про те, як ця архітектура працює насправді

На рис. 2.2.3.1 приведена блок-схема реалізації алгоритму Flux.

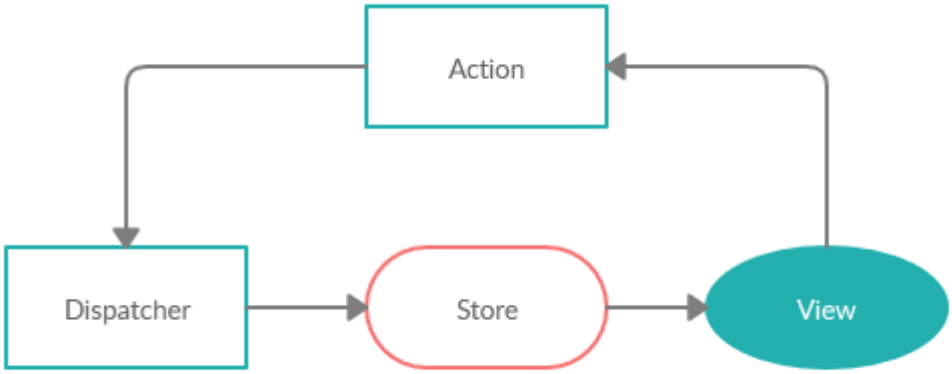


Рис. 2.2.3.1

2.2.4. Переваги такого підходу

1. Легка та зручна передача стану між компонентами

Ми зберігаємо всі дані в одному місці, відомому як сховище в Redux. Коли ми дивимось на проблему переміщення стану з нижнього рівня модулів до вищого, це стає дуже гарним рішенням, оскільки стан буде оновлений, а нижній рівень може отримати доступ до сховища.

2. Передбачувані стани

Стан завжди передбачений у Redux. Якщо один і той же стан і дія передаються редюсеру, то генерується один й той самий результат, оскільки функції редюсерів чисті. Стан також є незмінним і ніколи не зміниться. Це дозволяє виконувати складні завдання, такі як скасування та обробку інформації. Також ми отримуємо доступ до попередніх станів,, що ми можемо

використовувати для різних операцій таких як порівняння даних та якихось дій з ними.

3. Легко знайти помилку

Redux спрощує лагодження програм. За допомогою історії дій та станів, легко зрозуміти яка помилка сталась і в якому саме місці. Також легко можна відстежувати помилки не тільки в коді програми, а і в взаємодії з мережею та інші.

4. Легкість обслуговування

Redux – це архітектурний шаблон про те, як слід організувати код, щоб полегшити розуміння структури будь-якого додатка Redux тим, хто має знання Redux. Навіть у випадку, якщо людина не знайома з таким підходом, опанувавши базовими знаннями ключових аспектів вона може підтримувати, обслуговувати та розробляти програмне забезпечення.

5. Легко тестувати

Реалізації Flux-архітектури, такі як Redux можна легко оцінити як функції для зміни стану чистих функцій. Ви можете зберегти сховище та відновити деякі статуси програм після оновлення. Це може бути по-справжньому корисним.

6. Візуалізація з сервера (Server Side Rendering)

Redux також може використовуватися для відображення контенту на стороні сервера. Це означає, що ви можете використовувати його для управління початковою візуалізацією програми, надіславши статус програми на сервер разом із запитом. Після цього запитувані компоненти повертаються клієнтам у виді HTML.

Це особливо корисно для поліпшення користувальницької роботи або оптимізації системи. Все ще робить користувач – переходить до сховища не стороні клієнта.

7. Велика та активна підтримка

Flux, а разом с ним і Redux, дуже активно підтримується як розробниками, так і людьми що його використовують. Майже кожен день з’являються якісь рішення, для поліпшення та оптимізації процесів у програмах. На даний момент доступна велика кількість рішень, і з часом їх стає все більше.

8. Повторне використовування

Концепція Redux заснована на функціональному програмуванні. Для розуміння принципів функціонального програмування важливим є розуміння того, як працюють чисті функції. Чистий і модульний код можна записати за допомогою функціонального програмування. Ми можемо зробити код набагато простішим для тестування, обслуговування та лагодження, записуючи невеликі та прості функції, які відокремлюються за обсягом та логікою, після чого ці функції ми можемо використовувати повторно для різних цілей.

9. Чисті функції

Це функції які на один й той самий набір вхідних даних, завжди повертають одні й ті самі вихідні дані, тобто не мають випадкових операцій з непередбачуваною поведінкою,, наприклад генерування випадкових чисел або створення поточного часу. За допомогою них ви не змінюєте поточні об’єкти, а змінює його та повертаєте новий об’єкт. Ці функції не залежать від стану, з якого вони викликаються, і для заданих аргументів вони повертають лише один результат. Це робить їх дуже передбачуваними. Оскільки чисті функції не змінюють значень, вони не впливають на контекст або будь-які побічні ефекти, з

чого випливає що програміст може зосередитись лише на значеннях, повернених чистою функцією.

Що ми отримаємо використовуючи Flux-архітектуру за допомогою Redux?

З кожним днем Flux та Redux набирає більшої популярності. Велика кількість підприємств використовують такий підхід для створення сучасних веб-додатків. Інколи, розробники можуть скаржитися, що існує витрата часу на створення додаткового коду щоб використовувати можливості Flux, наприклад на таких простих діях, таких як відстеження даних значень полів вводу користувачем.

Дійсно, Flux-архітектура не завжди є найкращим рішенням. В більшості, це залежить від обсягу та ієрархії компонентів у веб-додатках. Складно сперечатися з тим, що для програми з легкою та зрозумілою логікою, невеликою за масштабністю, Flux підхід може тільки заплутати. При такому сценарії, зазвичай, буде достатньо звичайної ієрархії з двосторонньою передачею даних між модулями. Але, коли йде мова про веб-програми, які містять в собі більше логіки, сценаріїв та взаємодій, Flux стає дуже зручним рішенням.

Проте слід зауважити, що зв’язуючи компонент зі сховищем, він стає менш гнучким. Це означає те, що за потребою змінити якусь логіку в передачі даних, або обробці цих даних потрібно лізти в Redux сховище, що не завжди зручно. Також, інколи, з’являється питання про повторне використання компонентів, якщо вони зв’язані зі сховищем. Розглянемо це твердження на прикладі форми веб-додатку: ми маємо компонент Form який відповідає за відображення якихось даних з зовнішнього ресурсу. Уявимо, що цей компонент нам потрібен у двох різних частинах проекту. У випадку, якщо Form буде зв’язаний зі сховищем, ми не зможемо зручно змінювати контент форми за нашими потребами, адже від бере дані зі сховища. Проте якщо ми зв’яжемо 2 компоненти вищі за ієрархією з Redux, тоді ми зможемо передавати в модуль

Form різні дані, та відображати саме ту форму, що нам треба використовуючи при цьому один й той самий модуль.

Тому варто розглядати різні варіанти використання Redux, та комбінувати підходи під ваші потреби.

У ході розробки мобільного ПЗ, дотримувалась наступні правила модифікації Flux-архітектури:

- якщо компоненту потрібен обмін даними та станами з іншими компонентами, то їх варто винести до сховища;
- якщо компонент високого рівня ієрархії відносно проекту, його стан і слід винести до сховища. Навіть якщо це суперечить першому пункту, дуже ймовірно що в майбутньому виникне потреба в цих станах іншим модулям;
- якщо компонент використовується в декількох місцях програми, його стан слід залишити локальним для компоненту, та використовувати його класичними React методами;
- якщо компонент є дочірнім до компонентів, що знаходяться відносно низько ієрархії модулів, цей компонент не слід зв’язувати зі сховищем, тому що, як правило, такі компоненти найчастіше вразливі до змін, при такому сценарію кожного разу змінювати логіку Redux не є зручним.

2.3. Взаємодія з алгоритмами лінійної алгебри

Для зручної взаємодії, підтримки, повторювального використання та для чистоти коду було вирішено винести всі функції які реалізують розрахунки лінійної алгебри у JavaScript бібліотеку.

Для зручного користування бібліотекою, її було винесено в менеджер пакетів для можливого її використання у будь якій JavaScript програмі. Розглянемо що таке менеджер пакетів, та які сучасні менеджери використовуються з JavaScript.

Менеджер пакетів (package manager) - спеціалізоване допоміжне програмне забезпечення для установки, настройки, оновлення та видалення компонентів програмного забезпечення.

2.3.1. NPM

Npm - це менеджер пакетів, який входить до складу Node.js. Протягом багатьох років Node широко використовувався розробниками JavaScript для обміну інструментами, установки різних модулів і управління їх залежностями.

Це широко використовуваний репозиторій для публікації проектів Node.js з відкритим вихідним кодом. Це означає, що це онлайн-платформа, де кожен може публікувати та ділитися інструментами, написаними на JavaScript.

Також це інструмент командного рядка, який допомагає взаємодіяти з онлайн-платформами, такими як браузери і сервери. Ця утиліта допомагає в установці і видаленні пакетів, управлінні версіями і залежностями, необхідними для запуску проекту.

2.3.2. YARN

Yarn це новий менеджер пакетів, спільно створений Facebook, Google, Exponent і Tilde. Як можна прочитати в офіційній документації, його метою є

вирішення цілого ряду проблем, з якими зіткнулися розробники при використанні npm, а саме:

- установка пакетів не була досить швидкою і послідовної;
- існували проблеми з безпекою, так як npm дозволяє пакетам запускати код при установці.

Це не спроба повністю замінити npm. Yarn це новий клієнт командного рядка, що викачує модулі з реєстру npm. У самому реєстрі нічого не змінюється - ви можете завантажувати і публікувати модулі також, як і раніше.

Для роботи обох цих менеджерів, як і розробки програми на JavaScript (RN) також обовязково потрібен Node.js.

Node.js - це середовище для серверної розробки на мові JavaScript. Вона кроссплатформенная і має відкритий вихідний код. Використовується здебільшого для написання веб-серверів, але має і масу інших можливостей.

З'явившись в далекому 2009 році, коли JavaScript вже почав вважатися серйозним мовою, Node завоювала величезну популярність і фактично стала лідером в сфері веб-розробки.

Розберемо навіщо потрібно використовувати Node.js.

- Він ассинхронний

JavaScript працює в одному потоці, використовуючи події і функції зворотного виклику для його розвантаження. Це було зручно на фронтенді, тепер це і зручно на сервері.

Практично всі об'єкти в Node.js успадковують від класу EventEmitter, тобто здатні працювати з подіями.

- Величезна кількість модулів і бібліотек

Пакетний менеджер npm забезпечує стрімкий розвиток екосистеми Node.js. Зараз в ньому понад 500 тисяч опенсорсний пакетів, і кожен день з'являються нові.

- JavaScript

Використовує той же самий, який вже використовують мільйони розробників на фронтенді. Тепер вони можуть сміливо переходити на server side, не вивчаючи принципово інший інструмент.

Відмінності, звичайно, є. Наприклад, в Node.js немає DOM, cookie і інших браузерних API. Зате є безліч власних корисних методів і повний контроль над середовищем виконання коду.

- Рушій V8

Це опенсорсний проект, написаний на C ++, який активно розвивається і вдосконалюється зусиллями тисяч розробників.

Хоча JavaScript вважається інтерпретується мовою, на ділі процес його обробки не так вже простий. Це вже давно дорослий серйозний мову, який може працювати на протязі декількох годин поспіль, тому має сенс створювати готовий відкомпільований код. Сучасні рушії поєднують інтерпретацію і компіляцію, що робить їх дуже швидкими.

React Native використовує Node.js, середу виконання JavaScript, щоб побудувати ваш Код JavaScript.

2.4. Висновки до розділу

Проаналізувавши сучасні технології, що використовуються для розробки мобільного програмно забезпечення можна дійти до наступних висновків: сучасний вибір технологій дуже великий. Існує велика безліч бібліотек, фреймворків, архітектурних шаблонів та інших засобів для реалізації, спрощення та покращення проектів. На основі проаналізованої інформації була обрана мова програмування – JavaScript та фреймворк React Native. Також обрано сучасний архітектурний шаблон – Flux, який дає можливості легкого повторного використовування коду та підтримки його чистоті. Обраний набір технологій зараз дуже популярний та простий сам по собі, що також надає змогу розробникам розширювати фунціонал розробленого ПЗ.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ

3.1. Вимоги до програмного забезпечення

3.1.1. Функціональні вимоги

Перед створенням мобільного додатку для нього були визначені наступні функціональні вимоги:

- Програма повинна надавати можливість користувачу обрати бажану кількість матриць для операцій з ними;
- Програма повинна надавати можливість користувачу зручно обирати бажаний розмір матриці (матриць);
- Програма повинна надавати можливість користувачу заповнювати та редагувати її елементи;
- Програма повинна надавати можливість детермінування, множення, транспонування та інверсування одної матриці;
- Програма повинна надавати можливість користувачу додавати, віднімати та множити дві обрані матриці;
- Програма повинна працювати на ОС IOS та Android всіх сучасних версіях.

3.1.2. Нефункціональні вимоги

- Ефективність: програма повинна дуже швидко оброблювати всі операції без затримок;
- Безпека: програма повинна працювати окремо від всіх інших програм, не уповільнювати їх роботу;
- Повторне використання: програма повинна мати якомога більше частин коду, які можливо використовувати в інших місцях;

- Масштабування: програма повинна легко доповнюватись новим функціоналом за потребою;
- Доступність: програма повинна мати відкритий вихідний код, бути кросплатформеною та безкоштовною.

3.2. Архітектура модулів мобільного додатку

Архітектура модулів мобільного ПЗ - це структура, що складається з взаємозв'язків та взаємодій між компонентами(модулями) програми.

Як було сказано раніше, архітектура МД складається з декількох компонентів, які допомагають побудувати її цифровий склад.

Ці компоненти можна класифікувати на дві області: компоненти програмного інтерфейсу користувача та структурні компоненти.

Компоненти інтерфейсу користувача стосуються мобільних додатків, на яких відображаються інформаційні панелі, журнали, сповіщення, налаштування конфігурації тощо. Вони не мають значення для структурної розробки програми та орієнтуються більше на користувальницький інтерфейс / досвід.

У процесі проектування проекту, була розроблена модульна архітектура, яку можна побачити на рис. 3.2.1.

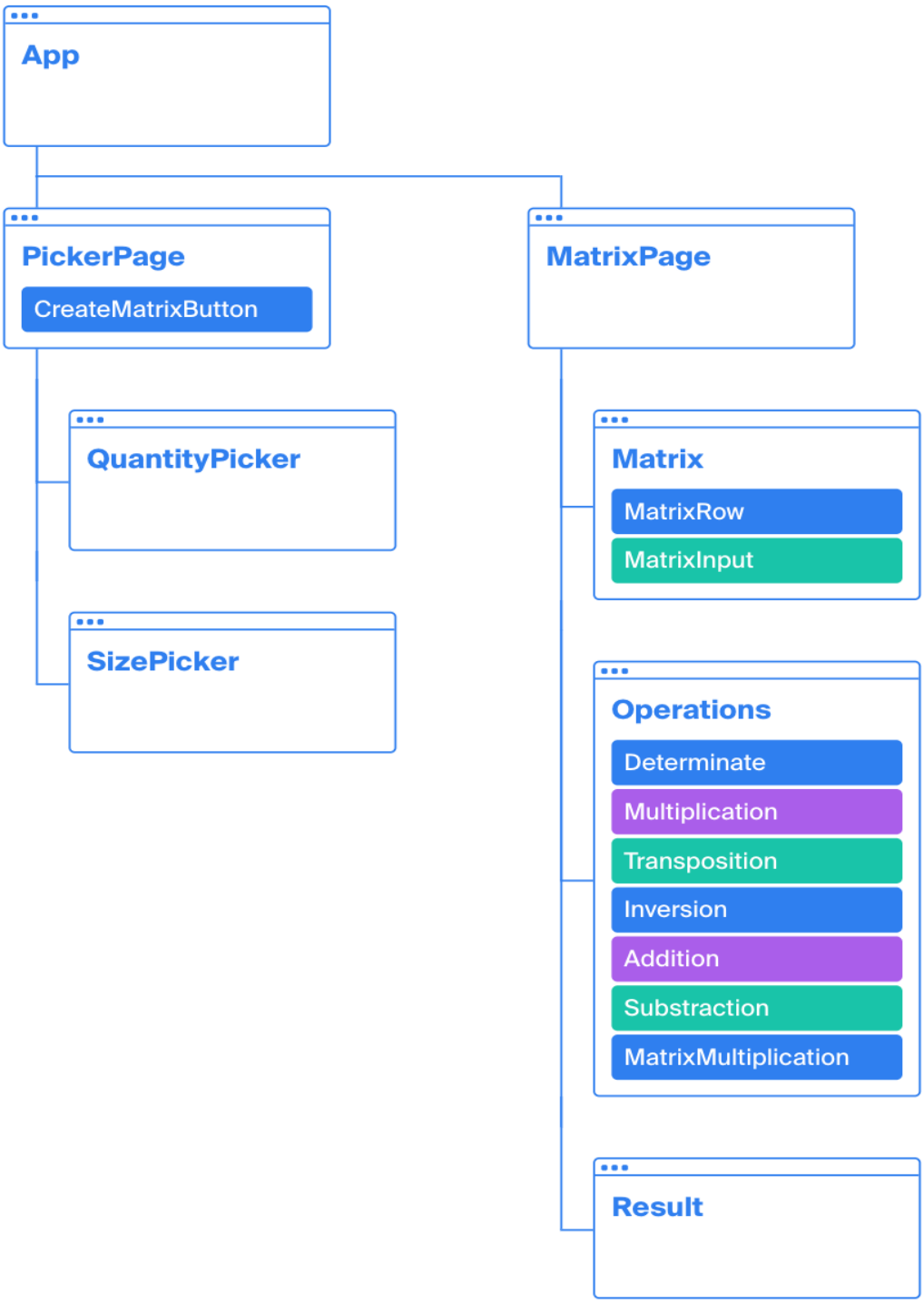

App
PickerPage
CreateMatrixButton
MatrixPage
QuantityPicker
SizePicker
Matrix
MatrixRow
MatrixInput
Operations
Determinate
Multiplication
Transposition
Inversion
Addition
Substraction
MatrixMultiplication
Result

Рис. 3.2.1

3.3. Управління даними в мобільному додатку

Управління даними та станами програми – наряду з архітектурою компонентів, є одним з ключових питань під час розробки мобільних додатків.

Проблеми з традиційною архітектурою MVC:

Перш ніж зануритися в поняття управління даними у Flux, давайте спочатку зосередимось на проблемах, які він вирішує. Модель Model-View-Controller (MVC) в наш час знайома більшості мобільних розробників. Ця модель описує поділ між даними (Model), візуалізацією (View) та логікою програми (Controller). Це гарантує, що ваша програма побудована структуровано, і ви зустрінете мінімальної кількості можливих проблем.

Однак недоліком є те, що ви втрачаєте контроль над потоками даних. Загалом потоки даних є двонаправленими. Доступ користувача користувача до одного копоненту може впливати на інші компоненти і навпаки. Контроль потоку даних та переконання в тому, що всі компоненти інтерфейсу користувача оновлюються відповідно - це завдання, яке дуже часто призводить до помилок. Також по мірі розширення проекту з такою архітектурою не довго чекати моменту, коли аналізувати та підтримувати програмний код буде дуже складно.

Використовуючи Flux, ми вирішуємо цю проблему, вводячи в наш додаток центральне сховище даних. Сховище містить стан програми та є джерелом даних для компонентів. Використовуючи концепцію Flux, вам не потрібно синхронізувати стан між компонентами вручну. Натомість ви можете повністю покластися на сховище у будь-який час.

Мобільний додаток для вирішення типових задач лінійної алгебри реалізовується за допомогою немалої кількості модулів. За принципами описаними у минулих пунктах, деякі компоненти зв’язані зі сховищем,а деякі мають логіку опрацювання даних та станів у межах цих компонентів. Кількість

станів у програмі є дуже великою, тож охопити абсолютно всі стані та логіку, що використовується у проекті займе дуже великий та схожий між собою простір.

Пропоную ознайомитись з ключовими аспектами реалізації Flux-архітектури у програмі на рис. 3.3.1.

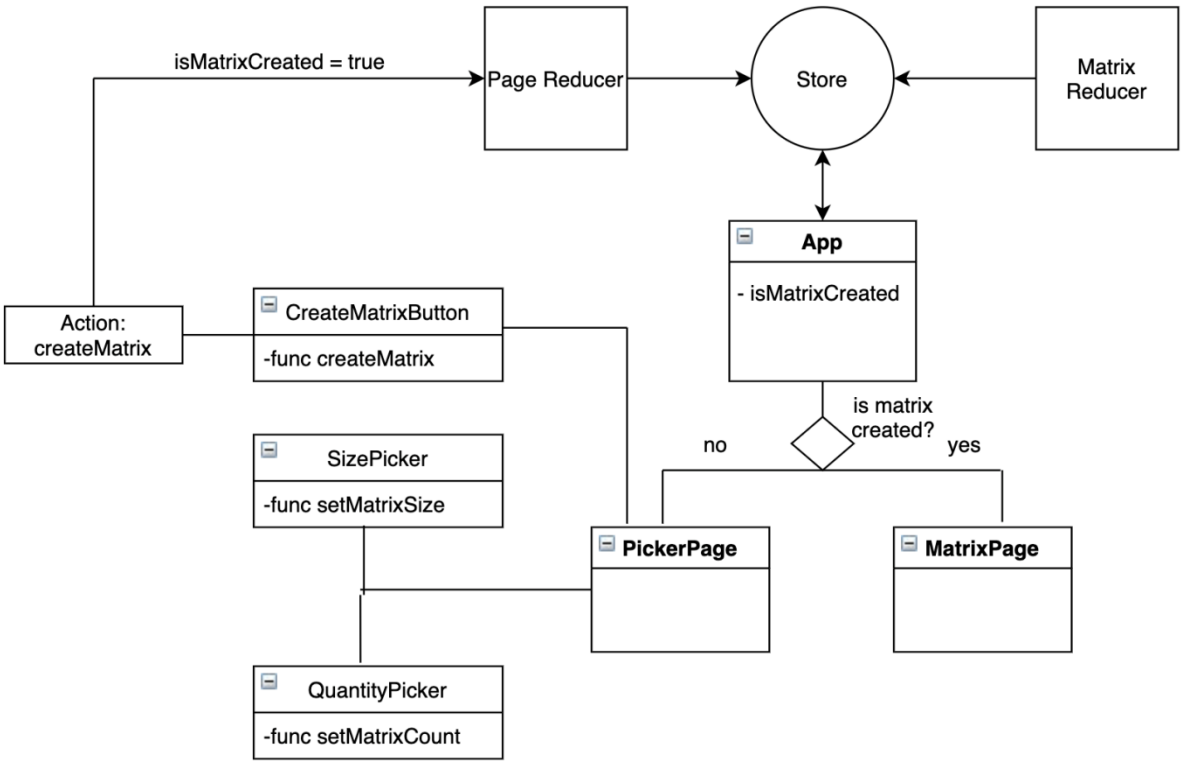


Рис. 3.3.1

Приведена вище діаграма характеризує обмін даними у додатку.

У наведеному прикладі, в нас є глобальне сховище даних (store), та два редюсери: Page Reducer який зберігає інформацію про загальний стан програми, та MatrixReducer, що вміщує дані про елементи матриць.

Три основні компоненти зв’язані зі сховищем: App, PickerPage та MatrixPage. Програмну реалізацію компонента PickerPage можна побачити у Додатку С.

Головний компонент по ієрархії - App отримує зі сховища дані про стан програми та на основі цієї інформації вирішує яку сторінку відображати користувачу. При сценарії, що користувач ще не створив матрицю він буде перенаправлений на сторінку її створення.

На сторінці створення матриці користувач обирає кількість матриць та їх розмір, після чого при створенні викликається дія (action) – createMatrix, яка в свою чергу замінить дані в Page Reducer. Як тільки програма дізнається що користувач створив матрицю – програма змінить сторінку на ту, що відображає матрицю та можливі операції над ними (Matrix).

Програмний код компоненту Matrix можна побачити у Додатку D.

3.4. Алгоритми вирішення типових задач лінійної алгебри

Розроблене програмне забезпечення підтримує наступний набір операцій для вирішення задач:

1 матриця

- Знаходження детермінанту (Determinant)
- Множення матриці на число
- Транспонування матриці
- Знаходження оберненої матриці

2 матриці

- Додавання матриць
- Віднімання матриць
- Множення матриць

3.4.1. Алгоритм знаходження визначника матриці:

Для матриць порядку n = 2 визначник обчислюється за формулою:

$$\Delta = a_{11} \times a_{22} - a_{12} \times a_{21}$$

(3.4.1.1)

Для матриць порядку n = 3 визначник обчислюється через алгебраїчні доповнення.

Матриця, що має розмірність більше трьох, розкладається на алгебраїчні доповнення, для яких обчислюються свої визначники (мінори). Наприклад, визначник матриці 4 порядку знаходиться через розкладання по рядках або стовпцях. [15]

$$\det A = \sum_{j=1}^{n} a_{ij} A_{ij} = \sum_{i=1}^{n} a_{ij}\, A_{ij}$$

(3.4.1.2)

Програмну реалізацію функції детермінування можна побачити у Додатку E.

3.4.2. Алгоритм множення матриці на число

Матрицею A помноженою на число k називається матриця B = k×A того ж розміру, отримана з вихідної множенням на задане число всіх її елементів. [16]

$$b_{ij} = k \times a_{ij}$$

(3.4.2.1)

$$B = M \times n = \begin{bmatrix} M_{11} \times n & M_{12} \times n & \cdots & M_{1n} \times n \\ M_{21} \times n & M_{22} \times n & \dots & M_{2n} \times n \\ \vdots & \vdots & \ddots & \vdots \\ M_{m1} \times n & M_{m2} \times n & \cdots & M_{mn} \times n \end{bmatrix}$$

(3.4.2.2)

Програмну реалізацію множення матриці на число можна побачити у Додатку F.

3.4.3. Алгоритм транспонування матриці

Транспонування матриці - це операція над матрицею, при якій її рядки і стовпці міняються місцями: [17]

$$a_{ij}^{T} = a_{ji}$$

(3.4.3.1)

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \qquad M^{T} = \begin{bmatrix} M_{11} & M_{21} \\ M_{12} & M_{22} \end{bmatrix}$$

(3.4.3.2)

3.4.4. Алгоритм знаходження оберненої матриці

Для знаходження оберненої матриці використовується метод алгебраїчних доповнень.

Для квадратної матриці A зворотної є матриця

$$M^{-1} = \frac{1}{|M|} \times \bar{M}$$

(3.4.4.1)

де $|M| \neq 0$ - визначник матриці M, а $\bar{M}$ - матриця, союзна з матрицею M. [18]

3.4.5. Алгоритм множення матриць

Для множення матриць використовується алгоритм “розділюй та володарюй”, який спирається на блокове розбиття.

Алгоритм працює для всіх квадратних матриць, чиї розміри є степенями двійки. [19]

$$M = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}, A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

(3.4.5.1)

3.4.6. Алгоритм додавання матриць

Додавання матриць A і B – це операція знаходження матриці C всі елементи якої дорівнюють попарній сумі всіх відповідних елементів матриць A і B [20], тобто кожен елемент матриці C обчислюється за формулою:

$$c_{ij} = a_{ij} + b_{ij}$$

(3.4.6.1)

$$C = A + B = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix} + \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1n} \\ B_{21} & B_{22} & \cdots & B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mn} \end{bmatrix} = \begin{bmatrix} A_{11} + B_{11} & A_{12} + B_{12} & \cdots & A_{1n} + B_{1n} \\ A_{21} + B_{21} & A_{22} + B_{22} & \cdots & A_{2n} + B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} + B_{m1} & A_{m2} + B_{m2} & \cdots & A_{mn} + B_{mn} \end{bmatrix}$$

(3.4.6.2)

3.4.7. Алгоритм віднімання матриць

Віднімання матриць (різниця матриць) A - B є операція обчислення матриці C, все елементи якої рівні попарній різниці всіх відповідних елементів матриць A і B, тобто кожен елемент матриці C дорівнює:

$$c_{ij} = a_{ij} - b_{ij}$$

(3.4.7.1)

$$C = A - B = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix} - \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1n} \\ B_{21} & B_{22} & \cdots & B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mn} \end{bmatrix} = \begin{bmatrix} A_{11} - B_{11} & A_{12} - B_{12} & \cdots & A_{1n} - B_{1n} \\ A_{21} - B_{21} & A_{22} - B_{22} & \cdots & A_{2n} - B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} - B_{m1} & A_{m2} - B_{m2} & \cdots & A_{mn} - B_{mn} \end{bmatrix}$$

(3.4.7.2)

3.5. Висновки до розділу

Програмну реалізацію будь-якої програми слід починати з робробки архітектури. Було розроблено архітектуру компонентів додатку, на основі якої за допомогою обраних технологій її було програмно реалізовано. Встановлено залежності, завантажено частину коду, яка відповідає за підрахунки типових задач лінійної алгебри у npm-пакет. Це надасть можливість зручно використовувати цей код у будь-якому проекті на мові JavaScript.

Реалізація мобільного додатку проходила у середовищі для розробки XCode, на операційній системі macOS за допомогою вбудованих емуляторів мобільного застосунку. Реалізована програма має відкритий вихідний код, що надає змогу доповнювати її та використовувати її компоненти.

ВИСНОВКИ

Під час виконання дипломної бакалаврської роботи було:

1. Досліджено типові задачі лінійної алгебри
2. Проведено аналіз видів мобільного програмно забезпечення
3. Проведено аналіз алгоритмів для вирішення типових задач лінійної алгебри
4. Проаналізовано та обрано мову програмування та фреймворк для програмної реалізації
5. Проаналізовано архітектурні шаблони сучасних мобільних додатків
6. Розроблено архітектуру мобільного додатку та архітектуру управління даними
7. Програмно реалізовано алгоритми для вирішення задач лінійної алгебри
8. Програмно реалізовано кросплатформене мобільне програмне забезпечення

Отримана реалізація кросплатформеного ПЗ відповідає сучасним стандартам взаємодії користувача з додатком, таким як: ефективінсть, надійність, розширюваність, зручність використання та безпека.

Обраний набір архітектурних рішень має дуже простий вигляд, що сильно спростить задачу підтримки програмного забезпечення іншими людьми, у разі такої потреби.

Розроблене програмне забезпечення, отримане в результаті виконання кваліфікаційної бакалаврської роботи забезпечить студентів та викладачів вищих навчальних закладів інструментом, який спростить та прискорить процес навчання та перевірки отриманих результатів. Додаток може розповсюджуватись серед користувачів за допомогою онлайн-сервісів App Store та Play Market. Відкритий вихідний код надає змогу повторно використовувати компоненти та

розширювати функціональність розробленого мобільного додатку, за такою потребою.

Ознайомитися з розробленим ПЗ можна у Додатку G.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Оффіційний сайт React Native - https://reactnative.dev/
2. Офіційний сайт React - https://en.reactjs.org/
3. Офіційний сайт Redux - https://redux.js.org/
4. Офіційний сайт React-Redux - https://react-redux.js.org/
5. Сервіс глобальнох статистики - https://www.statista.com/
6. Kyle Simpson, You Don't Know JS: Scope & Closures, 2014
7. Will Faurot, Redux in action, 2018
8. Alex Banks & Eve Porcello, Learning React: Functional Web Development with React and Redux, 2017
9. Eric Masiello & Javob Friedmann, Mastering React Native, 2017
10. Vladimir Novick, React Native – Building Mobile App with JavaScript, 2015
11. Marc Thomas, React In Action, 2018
12. Stan Bershadskiy, React Native Cookbook, 2016
13. Matrin Roberth, Clear Architecture, 2018
14. Martin Roberth, Clear Code, 2008
15. Шафаревич І.Р., Лінійна алгебра і геометрія, 2009
16. Козак А.В. , Лінійна алгебра, 2005
17. Єгоров В.В., Лінійна алгебра в питаннях і завданнях: Навчальний посібник , 2008
18. Бутузов В.Ф., Лінійна алгебра в питаннях і завданнях, 2008
19. Антонов В.І., Лінійна алгебра та аналітична геометрія. Опорний конспект, 2011
20. Кадомцев, С., Аналітична геометрія і лінійна алгебра: Навчальний посібник для вузів, 2011

ДОДАТКИ

Додаток А

Приклад програмної реалізації дії (action)

```
export const CREATE_MATRIX = 'app/CREATE_MATRIX';
export const SET_MATRIX_COUNT = 'app/SET_MATRIX_COUNT';

export const createMatrix = () => ({
  type: CREATE_MATRIX,
});

export const setMatrixCount = (count) => ({
  type: SET_MATRIX_COUNT,
  payload: { count },
});
```

Додаток В

Приклад програмної реалізації редюсера (reducer)

```
const CREATE_MATRIX = 'app/createMatrix';


const initialState = {
  isMatrixCreated: false,
  theme: 'dark',
  matrixCountChoosed: 1,
};

export default (state = initialState, action) => {
  const { type } = action;

  switch (type) {
    case CREATE_MATRIX:
      return { ...state, isMatrixCreated: true };
    default:
      return state;
  }
};
```

Додаток С

Програмна реалізація компонента PickerPage

```
const PickerPage = ({ isMatrixCreated }) => {
  const [singleMatrixColumns, setSingleMatrixColumns] = useState(2);
  const [singleMatrixRows, setSingleMatrixRows] = useState(2);
  const [shouldRender, setIsShouldRender] = useState(false);
  const [quantity, setQuantity] = useState(1);
  const [singleMatrixResults, setSingleMatrixResults] = useState([]);

  const handleChangeSingleMatrix = (rowIndex, columnIndex, value) => {…
  };

  const handleDeterminant = () => {…
  };

  return (
    <View>
      <View style={{ marginTop: 20 }}>
        <QuantityPicker onChange={setQuantity} value={quantity} max={2} />
      </View>
      <SizePicker
        onChange={setSingleMatrixColumns}
        value={singleMatrixColumns}
        max={8}
      />
      <Button
        title="Створити матриці" onPress={() => setIsShouldRender(!shouldRender)} />
      {isMatrixCreated && quantity === 1 && (
        <View style={{ marginLeft: 'auto', marginRight: 'auto' }}>
          <Matrix
            onMatrixChange={handleChangeSingleMatrix}
            columns={singleMatrixColumns}
            rows={singleMatrixRows}
          />
        </View>
      )}
      <Operations />
    </View>
  );
};
```

Додаток D

Програмна реалізація компоненту Matrix

```
const Matrix = ({ columns = 2, rows = 2, onMatrixChange = () => {} }) => {
  let lastN = rows - 1;
  let rowIndex = 0;
  const renderMatrix = () => {
    let rowsArray = [];
    while (lastN > 0) {
      rowIndex++;
      rowsArray.push(
        <MatrixRow
          onChange={onMatrixChange}
          count={columns}
          rowIndex={rowIndex}
        />,
      );
      lastN--;
    }
    return rowsArray;
  };

  return (
    <View>
      <MatrixRow
        onChange={onMatrixChange}
        count={columns}
        rowIndex={rowIndex}
      />
      {renderMatrix()}
    </View>
  );
};
```

Додаток Е

Програмна реалізація алгоритму знаходження визначника матриці

```
const determinant = A => {
  const n = A.length;
  const subA = [];
  let detA = 0;

  switch (n) {
    case 1: {
      return A[0][0];
    }
    case 2: {
      return (A[0][0] * A[1][1] - A[0][1] * A[1][0]);
    }
    case 3: {
      return ((A[0][0] * A[1][1] * A[2][2] + A[0][1] * A[1][2] * A[2][0] + A[0][2] * A[1][0] * A[2][1]) - (A[0][0] * A[1][
    }
    default: {
      for (var i = 0; i < n; i++) {
        for (var h = 0; h < n - 1; h++) {
          subA[h] = [];
        }
        for (var a = 1; a < n; a++) {
          for (var b = 0; b < n; b++) {
            if (b < i) {
              subA[a - 1][b] = A[a][b];
            } else if (b > i) {
              subA[a - 1][b - 1] = A[a][b];
            }
          }
        }
        var sign = (i % 2 === 0) ? 1 : -1;
        detA += sign * A[0][i] * determinant(subA);
      }
      return detA;
    }
  }
};
```

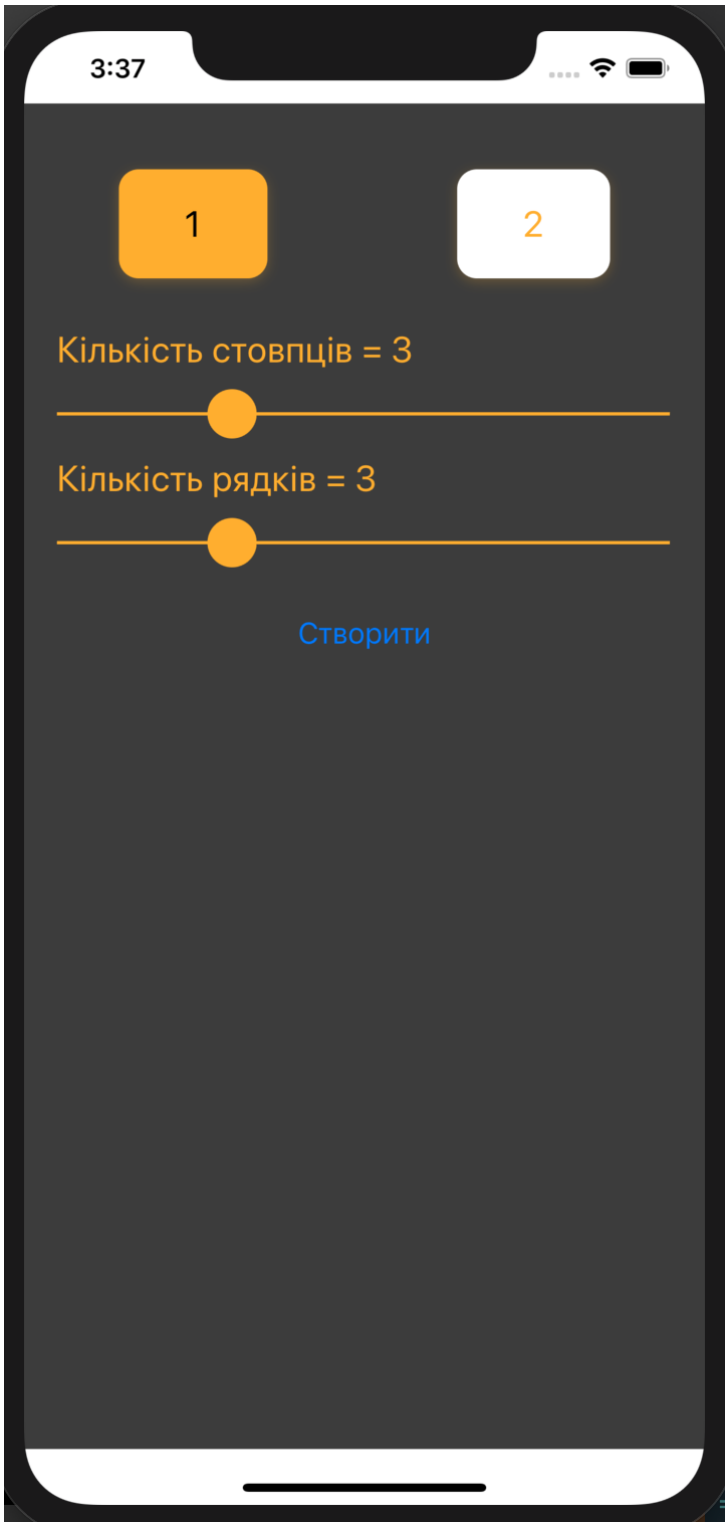
Додаток F

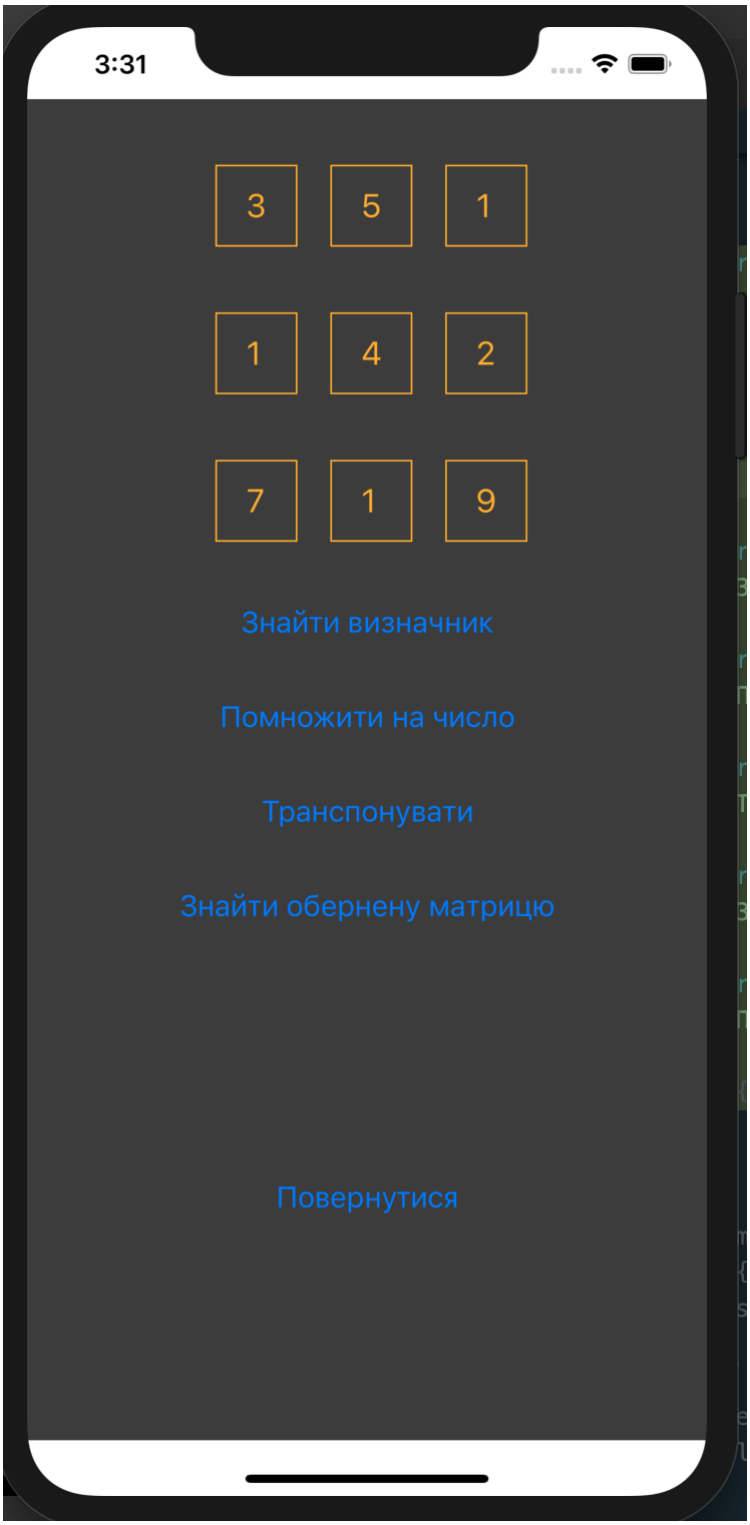
Програмна реалізація алгоритму множення матриці на число

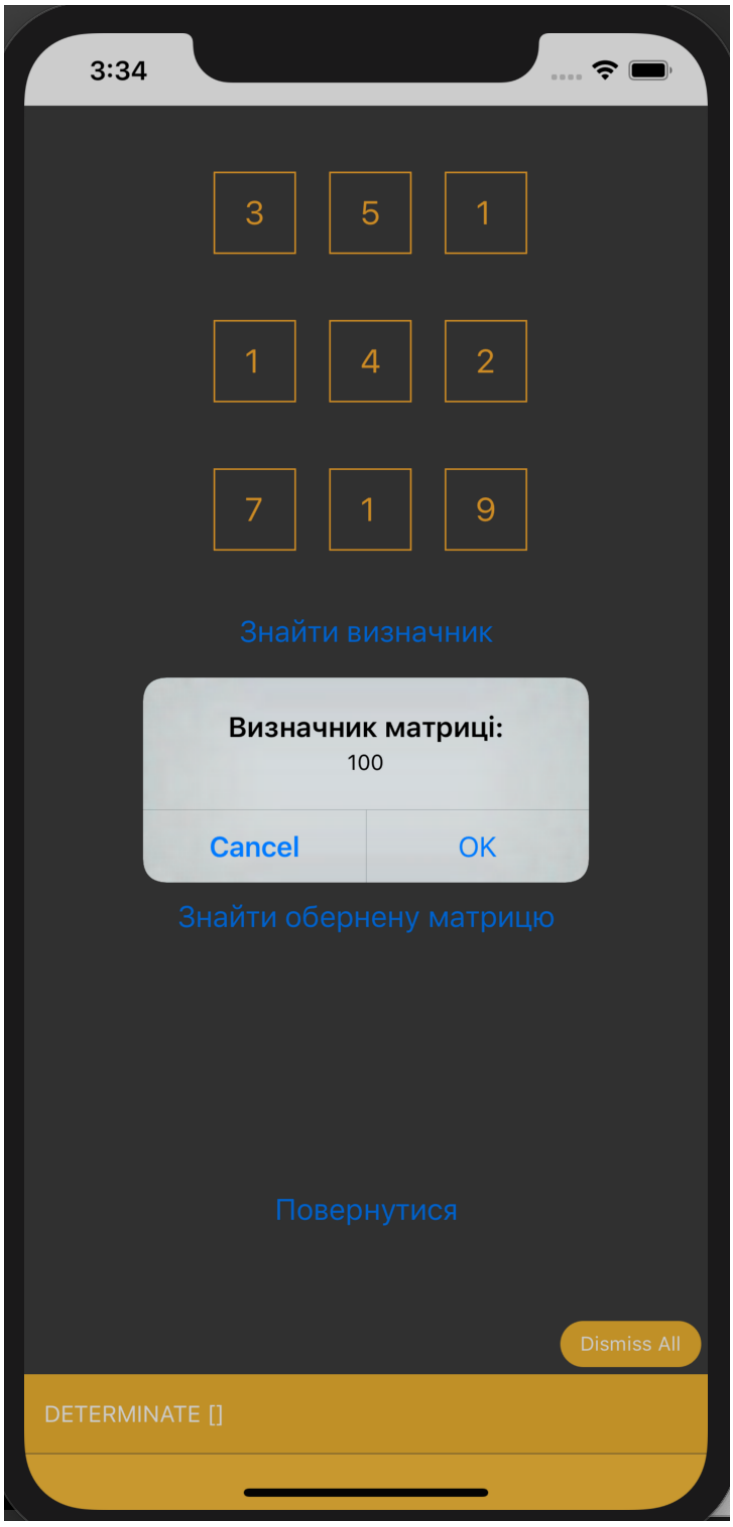
```
findScalar = function (arr, val) {
  for (var i = 0; i < arr.length; i++) {
    for (var j = 0; j < arr[i].length; j++) {
      arr[i][j] = val * arr[i][j];
    }
  }

  return arr;
};
```

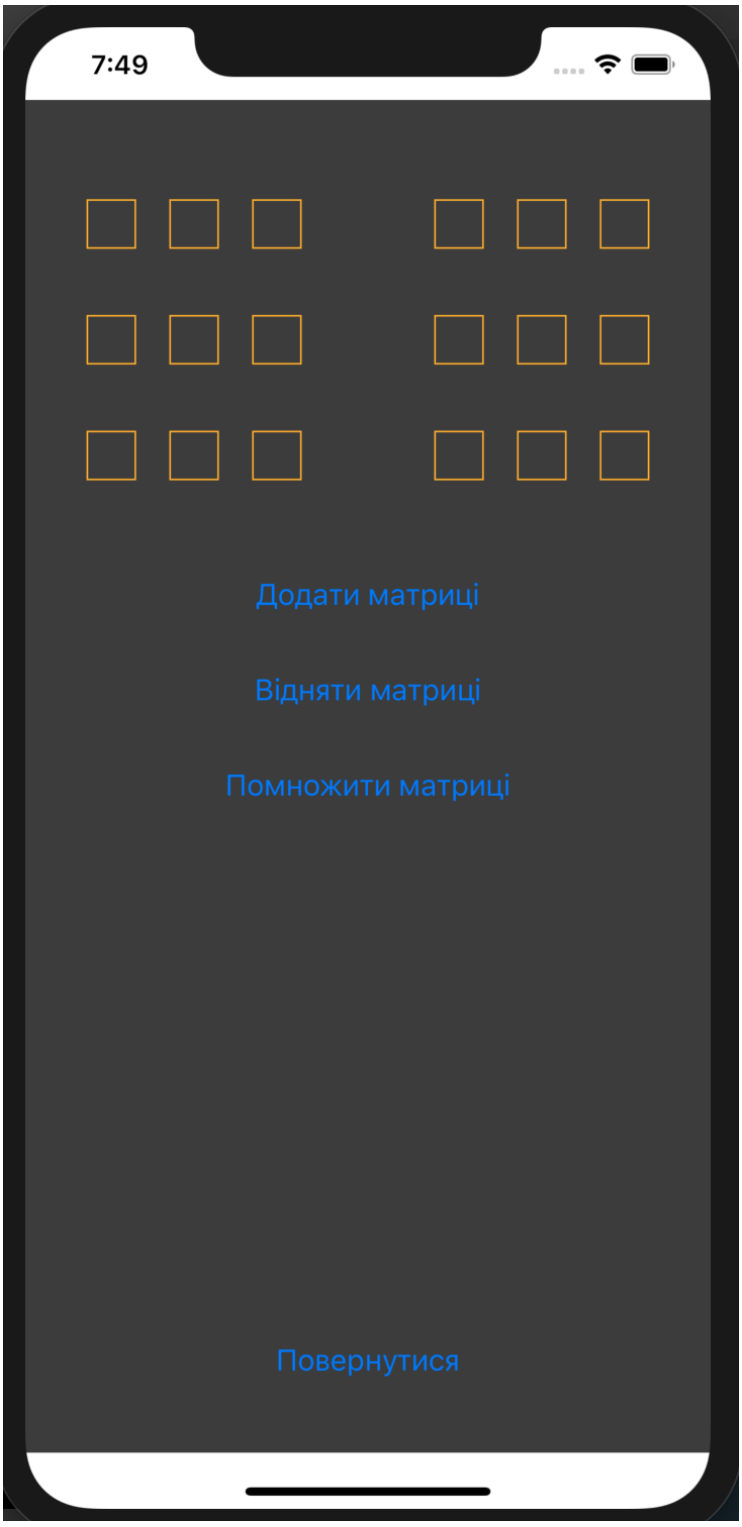
Додаток G

Інтерфейс розробленого мобільного ПЗ




3:31
3
5
1
1
4
2
7
1
9
Знайти визначник
Помножити на число
Транспонувати
Знайти обернену матрицю
Повернутися


3:34
3
5
1
1
4
2
7
1
9
Знайти визначник
Визначник матриці:
100
Cancel
OK
Знайти обернену матрицю
Повернутися
Dismiss All
DETERMINATE []


7:49
Додати матриці
Відняти матриці
Помножити матриці
Повернутися