Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра інформатики

ОГЛЯД СУЧАСНИХ ПІДХОДІВ ДО РОЗРОБКИ МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ БІЗНЕС ЗАСТОСУВАНЬ

Текстова частина до курсової роботи
за спеціальністю 121 «Інженерія програмного забезпечення»

Керівник курсової роботи
к.т.н.,ст.в.___Шабінська М.О.___
(прізвище та ініціали)
_____
(підпис)
“_ __ _” _____ 2020 р.
Виконав студент Папроцький І.А.
(прізвище та ініціали)
_____
(підпис)
“_ __ _” _____ 2020 р.

Київ-2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Завідувач кафедри інформатики,

канд. фіз-мат. наук, доц. – Гороховський С.С.

_____ *(підпис)*

“____” _____ 2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Папроцькому Ігорю Андрійовичу

Факультету інформатики 3 р.н. бакалаврської програми
ТЕМА: Огляд сучасних підходів до розробки мобільних застосунків для бізнес застосувань

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Огляд теоретичного матеріалу та здійснення дослідження

Висновки

Список використаної літератури та посилань

Додатки (за необхідністю)

Дата видачі “____” _____ 2019 р. Керівник _____

(підпис)

Завдання отримала _____
(підпис)

Тема: Огляд сучасних підходів до розробки мобільних застосунків для бізнес застосувань

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	25.10.2019	
2.	Пошук тематичної літератури	20.11.2019	
3.	Ознайомлення з літературою	25.12.2019	
4.	Вивчення аналогів	15.01.2020	
5.	Створення бази даних	20.01.2020	
6.	Створення директорій проекту і початкового серверу	30.01.2020	
7.	Реалізація створення користувача і видачі JSON Web токенів	10.02.2020	
8.	Створення основи додатку на React Native	30.02.2020	
9.	Створення основи додатку на Xamarin	27.02.2020	
10.	Створення основи native-додатку на Java для платформи Android	15.03.2020	
12.	Огляд та вивчення процесів створення native-додатків на Swift для платформи IOS	25.03.2020	
13.	Написання текстової частини	07.05.2020	
14.	Перегляд змісту роботи керівником	11.05.2020	
15.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	11.05.2020	
16.	Створення презентації	11.05.2020	
17.	Захист роботи	20.05.2020	

ЗМІСТ

АНОТАЦІЯ

У роботі розглянуті деякі аспекти сучасної мобільної розробки. Основна увага зосереджена на огляді різних способів написання мобільних додатків для різних платформ.

У першому розділі розглянута предметна область та поняття кросплатформеності.

У другому розділі розглянуті деякі теоретичні аспекти та відомості про архітектуру взаємодії даних у системах з мобільними застосуваннями, технології передачі даних. Детально розглянуто деякі особливості створення RESTful веб-сервісів.

У третьому розділі розглядається практична частина курсової роботи. Детально описуються реалізації серверу та трьох аналогічних мобільних додатків, написаних використовуючи різні підходи до створення мобільних застосувань. Також оглядаються можливості використаних фреймворків та деякі особливості реалізацій застосувань.

ВСТУП

Актуальність та практичне значення обраної теми

У 21 столітті значення смартфонів, планшетів та інших кишенькових гаджетів в житті людини з кожним днем зростає та набуває нових розумінь з появою нових технологій для реалізації тих, чи інших завдань.
Потреба людини у швидкому та легкому доступі до підручних засобів для отримання необхідної інформації, проведення багатьох видів вимірів та обчислень, запам'ятовування інформації та навіть просто відволікання від буденного життя за допомогою розваг легко (зі сторони користувача) реалізується наявністю смартфону, планшету, смарт-годинника або інших досягнень розробників технологій мобільної електроніки з, в більшості випадків, підключенням до мережі Інтернет.

Головними інструментами виконання навіть не завжди великих, але потрібних людині у конкретний момент часу, завдань є мобільні застосування.

Станом на 2020 рік широке використання мобільних застосунків обумовлене швидкими темпами розвитку ринку IT-технологій, існування широкого попиту на кишенькові гаджети, їх доступність та поширеність серед більшості верств населення багатьох країн світу та легкість завантаження потрібного програмного забезпечення «у кілька натиснень на екран». Щорічна поява на ринку великої кількісті нових додатків не тільки для розваг, а й таких програмних застосунків які допомагають людині вирішити буденні справи, зменшуючи кількість розумового або фізичного навантаження для досягнення мети, прозоро пояснює великий попит користування мобільними гаджетами.

Попри легку можливість скачати (або купити та скачати) власником смартфону будь-який з опублікованих в мережі застосунків, розробників мобільних застосунків завжди, окрім маркетингу, хвилює завдання зацікавити користувача та запобігти вимкненню та видаленню застосунку через кілька

перших хвилин після завантаження. Це досягається зручним та зрозумілим користувацьким інтерфейсом (UI), вірно оформленим UX (в пер. з англ. – “користувацький досвід”), ефективним використанням ресурсів пристрою, безпекою, швидкістю та безвідмовністю застосунку, правильно побудованою бізнес-логікою системи в цілому.

Для підтримання стандартів якості програмної системи в умовах комерційної розробки, запроваджених або самою фірмою-розробником, або IT-спільнотою, виконавцям планування архітектури системи та виконавцям написання коду потрібно якісно продумувати те, які саме підходи до розробки мобільних застосувань використовувати.

Плануючи проекти, більшість розробників розуміють, що від обраних методології та засобів програмування залежить кількість витраченого часу на реалізацію стійкої та ефективної системи. У сучасному світі при створенні програмного забезпечення цей лишній час може означати збільшення грошових витрат на розробку, тому у разі труднощів з реалізацією деяких частин додатку може страждати якість та швидкість виконання завдань і у інших частинах програмного застосунку, що створюється.

Технології розробки сучасних мобільних застосунків мають велику кількість аспектів, які впливають на швидкість та якість створення готових функціонуючих додатків. Аналізу цих аспектів і присвячена тема курсової роботи.

Структура роботи

Робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАННЯ КУРСОВОЇ РОБОТИ

1.1 Базовий огляд предметної області

Люди все частіше і довше користуються смартфонами кожен день. Завдяки розвитку мережних технологій та досягнень галузі електроніки, людина може мати «сервіси у кишені» майже у будь-якій точці планети. За результатами досліджень App Annie, ще у 2018 році по всьому світу пересічний користувач смартфону проводив майже 3 години на добу в сумі за роботою із мобільним пристроєм. [2]

Серед мобільних платформ, які користуються найбільшим попитом серед користувачів, можна виділити дві – Android та IOS, які розроблені компаніями-гігантами Google та Apple відповідно. Зручний для розробників з усього світу спосіб публікації та доставки своїх застосунків до кінцевого користувача забезпечений інтернет-платформами згаданих компаній – Google Play Market та App Store. При цьому, гаджети на операційній системі IOS відповідно до авторських прав, створює тільки компанія Apple. В свою чергу, операційну систему Android було розроблено як універсальну платформу на базі ядра Linux, з відкритими стандартами для мобільних пристроїв. [3]

За результатами статистики Sensor Tower, тільки за першу половину 2019 року App Store та Google Play Market в сумі отримали $39,7 млрд виручки. [1] Платформи є конкурентами, і, водночас, лідерами ринку.

Відповідно, для отримання достатньої виручки більшість виробників мобільного програмного забезпечення орієнтуються на два великі сегменти ринку, творці якого конкурують між собою.

Потрібно зазначити, що на ринку мобільних гаджетів існують також і відомі платформи Windows Phone та Windows 10 Phone, розроблені компанією

Microsoft. Але, через низьку популярність серед користувачів по всьому світу у порівнянні із зазначеними вище платформами [4], у даній роботі вона не розглядатиметься.

1.2 Типи мобільних застосувань та поняття кросплатформеності

Потрібно зазначити, що при створенні мобільного додатку має значення цільова аудиторія, на яку розрахований програмний продукт. Кожне конкретне застосування повинне виконуватись на конкретній операційній системі, тому, оскільки найпопулярніших систем є дві, компанії-виробники мобільного програмного забезпечення, повинні шукати способи підтримання свого продукту на ринку одразу для двох платформ задля уникнення втрати великої частини потенційних користувачів. При цьому незалежно від суті сервісу, який реалізовує конкретна система, постає питання вибору способу написання програмного коду. Цей вибір залежить від обраного типу мобільного застосування.

На практиці мобільні застосування можна розділити на три типи [5]:

1. Нативні (native – у дослівному пер. з англ. – “рідний, властивий”) – застосування, розроблені під одну конкретну операційну систему:

а) IOS – мова написання – Swift або Objective C;

б) Android – мова написання – Java або Kotlin.

2. Гнучкі (англ. – “responsive”) версії веб-сайтів;
3. Гібридні (англ. – “hybrid”) – застосування, зі спільним кодом для багатьох платформ.

Відповідно, нативні застосування реалізуються тільки для однієї платформи, і не можуть бути виконаними на інших платформах без зміни коду.

В свою чергу гнучкі версії веб-сайтів деколи теж відносять до повноцінних мобільних застосувань, хоча вони просто являють собою звичайні сайти, які користувач може переглянути за допомогою мобільного браузеру: такі веб-застосування реалізовані за рахунок адаптивного вебдизайну – дизайну веб-сторінок, що дозволяє оптимально розміщувати інформацію на багатьох екранах з різними параметрами висоти та широти [6].

Гібридні ж мобільні застосунки побудовані за допомогою використання мультиплатформених веб-технологій (наприклад, мов розмітки, мови стилів, та мов програмування JavaScript або TypeScript). Гібридні додатки являють собою веб-застосування, обгорнуті у нативну обгортку, яка дозволяє застосунку з одним і тим же програмним кодом бути сконфігурованим для конкретної операційної системи та запущеним на ній. [5]

Гібридні мобільні додатки, як і гнучкі адаптивні сайти дозволяють досягнути кросплатформеності у рамках написання одного коду для кількох мобільних операційних систем.

Отже, кросплатформеність – це здатність одного і того ж програмного застосунку працювати на декількох платформах або в різному програмному оточенні. Кросплатформеність досягається за рахунок реалізації можливості бути скомпільованою для різних платформ без відмінностей, або з мінімальною кількістю відмінностей початкового коду для різних платформ. [7]

В 2020 році існує досить багато технологій, які забезпечують можливість використання спільного коду для побудови мобільних застосунків на різних платформах. Деякими найпопулярнішими є фреймворки React Native, Xamarin, Flutter, та ін.

Використання таких, безумовно, зменшує об’єми написаного розробниками коду для досягнення мети, а також зменшує кількість часу, витраченого на проведення функціоналу застосунку від стадії розробки до

стадії релізу. Проте, часто в таких підходах, окрім плюсів, є й свої мінуси у порівнянні із native-розробкою. Огляд переваг та недоліків різних підходів і є основною частиною цієї курсової роботи.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Архітектурні шаблони програмного забезпечення

Безперечним є той факт, що наявність підключення до мережі Інтернет грає важливу роль у реалізації більшості задумок про створення програм.

При розробці мобільних додатків має значення де, так як будуть зберігатись дані, яким способом буде налагоджена їх комунікація між місцем збереження даних та користувачем, та як ці дані будуть відображені та оброблені.

Існує багато варіантів оформлення архітектури застосування. Напевне, найпоширенішою на даний момент є архітектура клієнт-сервер.

Звісно існують багато розробників, які намагаються «позбавитися» серверної частини, тобто будують свої застосунки навколо синхронізації даних зі сховищем в «хмарі», маючи при цьому дворівневу архітектуру. В деяких випадках, така схема виправдана. Проте, такий варіант не настільки гнучкий і масштабований в порівнянні з три- та більше- рівневою архітектурою. [17]

Класичний випадок, клієнт-серверної архітектури представляється такими складовими – клієнт, сервер та сервер сховища даних (Рис. 1):

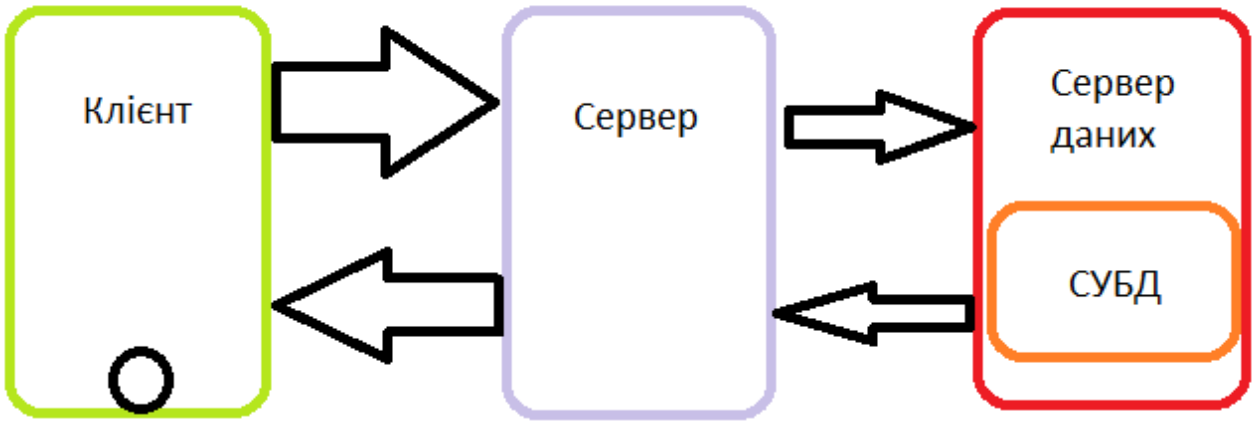

Рисунок 1. Комунікація в клієнт--серверній архітектурі

При цьому, функціональність коду по всьому застосуванню не завжди рівномірна. На кожному із рівнів код може ділитись на різні слої в залежності від потрібної функціональності. Це робиться з метою хорошого проектування програмного забезпечення, яке гарантує повторне використання коду, частково для зручності та частково для безпеки даних. [18]

В основному, для економії трафіку, часу життя батареї тощо, мобільні розробники стараються робити якнайменше обчислень на пристрої користувача.

Досить рідко можна побачити застосування, яке б не користувалось перевагами збереження даних на віддаленому репозиторії, та передачі складних для звичайного пристрою завдань більш потужним віддаленим машинам. Використання віддалених API дозволяє зберігати дані більш захищено, ніж на пристрої користувача. Також це дає можливість синхронізувати дані користувача на різних пристроях.[15] Не менш важливим є той факт, що без виконання важких обчислень, додаток на гаджеті користувача працює з набагато швидше та з меншими затратами процесорного часу та оперативної пам’яті. Зазвичай таку архітектуру називають тонким клієнтом. Тобто, більшість обчислювальних операцій покладається на сервер.

2.2 Технології обміну даними

Технології обміну даними у веб-сервісах різняться використовуваними протоколами. В нашому випадку нас цікавить саме протокол HTTP.

HTTP – (The Hypertext Transfer Protocol) – протокол передачі гіпертекстових документів прикладного рівня моделі OSI. Це загальний протокол без стану який використовується для виконання багатьох завдань шляхом використання гіпертексту з застосуванням методів запитів, кодів помилок та заголовків. [8]

В сучасному світі для розгортання веб-сервісів прийнято все більше використовувати захищений протокол HTTPS. Особливе значення це має і в мобільній розробці.

HTTPS – (The Hypertext Transfer Protocol Secure) – захищена версія HTTP. Цей протокол використовує TLS (Transport Layer Security) для кодування інформації шляхом використання сертифікату [9].

Мобільні платформи постійно підштовхують розробників використовувати HTTPS з сучасним шифруванням та довіреними підписаними сертифікатами [15]. Це дозволяє уникнути багатьох проблем із захистом мобільного застосування. Так, наприклад на платформі IOS існує програмно створена особливість платформи під назвою App Transport Security (ATS). Вона блокує всі з’єднання, які не виконують достатніх умов захисту (умова захисту з’єднань використанням TLS (Transport Layer Security) протоколом). [16]

Серед відомих технологій передачі даних які часто використовуються у мобільних застосуваннях варто відмітити протокол WebSocket та концепція RESTful API. Вони базуються на використанні HTTP(S) і часто використовуються у сучасному проектуванні додатків.

У цій курсовій роботі, оглянемо спосіб використання REST для написання сервісів для мобільних додатків, як найпопулярніший спосіб.

2.2 Опис REST

Рой Філдінг у своїй дисертації «Architectural Styles and the Design of Network-based Software Architectures» вперше описав цей термін. REST досяг успіху у широкому використанні в світі завдяки концепції використання WWW в основному для спілкування між машинами, а не тільки між людьми. Суть REST полягає у досягненні найкращого використанні протоколу HTTP.

REST – (Representational State Transfer) – це архітектурний стиль, деяка множина обмежень для побудови розподілених застосувань. [10] Важливо розуміти, що REST – не конкретний протокол, а архітектурний стиль, набір рекомендацій для реалізації гнучкого API, який не залежить від стану ресурсів, але діє на основі них.

Серед базових понять в REST – ресурси і представлення ресурсів, унікальні ідентифікатори ресурсів, стани ресурсів. [10]

Ключовою абстракцією ресурсів інформації в REST є ресурс. На цій абстракції концентрується протокол HTTP. Суть її полягає в тому, що будь яка інформація може бути названа ім’ям: документ або зображення, тимчасовий сервіс, колекція інших ресурсів, невіртуальний об’єкт (напр. персона), і т. д. Іншими словами, будь-яка концепція яка може бути ціллю гіпертекстового посилання повинна вміститись в визначення ресурсу. Ресурс – це концептуальне представлення набору сутностей, а не сутність яка відповідає представленню в будь-який конкретний момент часу. [11] Отже, сукупністю станів ресурсів можна охарактеризувати стан застосування.

Під представленням розуміється гіпертекст в форматі JSON/XML/HTML/plain text або будь-що, що дозволяє нам зрозуміти стан ресурсу або змінити його. [10]

В концепції REST також фігурує поняття уніфікованого ідентифікатору ресурсу (URI). Саме URI дозволяє уніфіковано визначити конкретний ресурс у системі.

Важливим моментом розробки з використанням концепції REST є також ретельне проектування всіх посилань на ресурси. API повинне забезпечувати повне функціонування системи незалежно від стану ресурсів [11].

Основуючись на всесвітньо відомих принципах створення архітектури веб-застосування у вигляді сервісів, у світі прийнято називати сервіси, які максимально повно виконують принципи REST – RESTful сервісами. Відповідно, API які реалізовують такі сервіси прийнято називати RESTful API.

2.3 Проблема однозначного визначення RESTful API

Досліджуючи тему REST, можна зрозуміти, що насправді велика частина людей з усього світу розуміють REST кожен по-своєму.

Наприклад, чітке строге визначення, які саме коди статусів відповідей і які саме методи запитів і тіла повідомлень використовувати в конкретній ситуації може набувати різних значень у різних предметних областях реалізації веб-застосувань. [13] Досить логічно, що за таких умов багато розробників всередині IT-компаній керуються своїми визначеннями рекомендацій щодо реалізації RESTful API.

Основні концепції, які треба враховувати для роботи з REST є наступними:

- при розробці RESTful сервісу потрібно концентрувати увагу на ресурсах та визначенню як вони повинні бути відкритими для користувача; [12]
- потрібно використовувати заголовки, вже визначені протоколом HTTP, для виконання операцій з ресурсами; [12]
- формат обміну даними не має обмежень але зазвичай використовується JSON або XML; [12]
- використання протоколу HTTP (HTTPS). REST повністю побудований на основі цього протоколу; [12]
- потрібно пам’ятати, що REST являється гнучким в плані визначення сервісу. Не існує стандарту для цього, але застосуванню потрібно розуміти формати запитів та відповідей; [12]
- потрібно старатись досягти максимально повного використання методів HTTP-запитів для конкретного визначення дії, яку потрібно використати з ресурсом (основні методи - GET, POST, PUT, DELETE, але потрібно не забувати, що існують і інші, які допоможуть конкретніше ідентифікувати дію);
- потрібно добре розуміти, в яких ситуаціях використовувати конкретні коди статусу відповіді;
- RESTful API не повинне залежати від будь-якого протоколу комунікації, проте, його успішне відображення на даний протокол має залежати від доступності метаданих, вибору методів тощо; [14]
- RESTful API не повинне містити будь-яких змін до комунікаційного протоколу; [14]
- RESTful API не повинне мати фіксованих назв ресурсів або ієрархій. Сервери повинні мати свободу контролювати свій особистий простір імен; [14]

Існує також багато інших уточнень та конкретизацій визначення REST, на які теж можна звертати увагу при побудові архітектури свого веб-застосування.

2.5 Висновки до розділу 2

REST – один з найбільш використовуваних стилів проектування API, особливо в світі мобільних застосувань. Притримування RESTful-стилю побудови архітектури веб-сервісу дозволяє запевнитись, що нові розробники, яким надано завдання підтримувати написаний раніше серверний код, будуть легко орієнтуватись в ньому, і більше того, можливо, покращувати та розширювати правильно оформлену систему для її гнучкого та правильного функціонування. [15]

Розробка мобільних додатків є дуже широкою для розгляду темою, яка має дуже багато особливостей. Ці особливості проявляються не тільки в архітектурному плані. Важливо враховувати також способи реалізації одних і тих самих завдань різними способами, що забезпечується наявністю великої кількості бібліотек, методик, технологій, протоколів та стилів проектування. До огляду різних варіантів використання цих технологій приступаємо в наступному розділі.

РОЗДІЛ 3. ОПИС ПРАКТИЧНОГО ДОСЛІДЖЕННЯ

3.1 Постановка задачі практичного дослідження

Проаналізувавши предметну область було вирішено дослідити різні типи мобільних застосувань та підходи до їх створення. Для цього вирішено створити невеликий мобільний додаток у різних варіаціях, з використанням декількох фреймворків

Отже, виділено такі задачі:

1. Використовуючи мову Java та фреймворк Spring Boot створити серверну частину для функціонування вищеописаного клієнт-серверного застосування. Для аутентифікації використати JSON Web Token.
2. Використовуючи деякі сучасні фреймворки для розробки кросплатформених застосувань для IOS та Android, а саме React Native та Xamarin, створити мобільні застосування використовуючи кожен з них.
3. Створити аналогічний додаток під ОС Android на мові Java без використання кросплатформених фреймворків. Використати Android SDK.
4. Оглянути особливості написання додатків під ОС IOS без використання кросплатформених фреймворків (тип – native) на мові Swift використовуючи джерела інформації.
5. Використовуючи кожен з методів створення мобільних додатків, описаних в попередніх пунктах постановки задачі, визначити особливості, переваги та недоліки кожного методу та порівняти зазначені методології між собою.

3.2 Аналіз технічного завдання

Для огляду деяких можливостей вищеописаних фреймворків та підходів, було вирішено створити додаток з деяким базовим функціоналом, для якого чудово підходить стиль REST. Варто зазначити, що метою курсової роботи є огляд підходів до розробки мобільних застосунків а не розробка кінцевого продукту. Тому, деякий функціонал звичних додатків упущений, адже для його реалізації існує необхідність дублювання вже розглянутих процесів розробки. Також через майже безмежний простір можливостей мобільної розробки, деякі тонкощі створення додатків на конкретній платформі можуть бути упущеними.

Отже, вирішено створити мобільний додаток, назвемо його “Tasker”, в якому користувач може аутентифікуватись, додати собі «завдання», які він хоче виконати, відображати список своїх завдань, оглядати конкретно вибране завдання, вийти з акаунту та аутентифікуватись під другим акаунтом. Завдання в даному випадку представлятимуть собою звичайний набір значень, який складається з унікального ідентифікатору, назви завдання та його опису (такі собі, замітки). Сам користувач матиме унікальний ідентифікатор, ім’я користувача та пароль.

Для реалізації поставленого завдання нам знадобиться сервер, база даних, а також кілька аналогічних між собою мобільних додатків, які реалізовують цей функціонал.

3.3 Опис реалізації серверу та використаної бази даних

Для написання так званого «бек-енду» (серверної частини) я використовував інструмент розробки – IDE IntelIJ IDEA 2020.1. Інструментом розробки серверу було обрано Java-фреймворк Spring Boot.

Spring Boot – це, Java-фреймворк з відкритим кодом, який використовується для побудови мікросервісів. [19] Він побудований на основі Spring Framework, який, в свою чергу, використовується для створення гнучких корпоративних застосувань.

Spring Boot дозволяє ефективно розробляти застосування, витрачаючи мінімальну кількість часу на конфігурацію. Також, основна концепція фреймворку – IoC (Inversion of Control) – абстрактний принцип в програмному забезпеченні, завдяки якому контроль за об’єктами або «порціями програми» передається контейнеру фреймворку. [20] Таким чином, за допомогою Dependency Injection – паттерну, який описує з’єднання об’єктів через «ін’єкції» один в одного [20], досягається концепція легкої підтримуваності коду, побудова залежностей між потрібними екземплярами класів стає прозорою та легкою.

Для реалізації аутентифікації та (як приклад, умовної авторизації) – було використано Spring Security – фреймворк, який став, де-факто, стандартом захисту Spring-застосувань. [21]

В свою чергу базою даних обрано PostgreSQL – одну з найбільш популярних об’єктно-реляційних систем керування базами даних, яка розвивається як «opensource», тобто її розробляє та вдосконалює спільнота. [22]

Отже, розглянемо схему бази даних. Оскільки для функціоналу наших додатків в даному випадку не потрібно зберігати багато лишньої інформації, маємо наступну ER-діаграму:

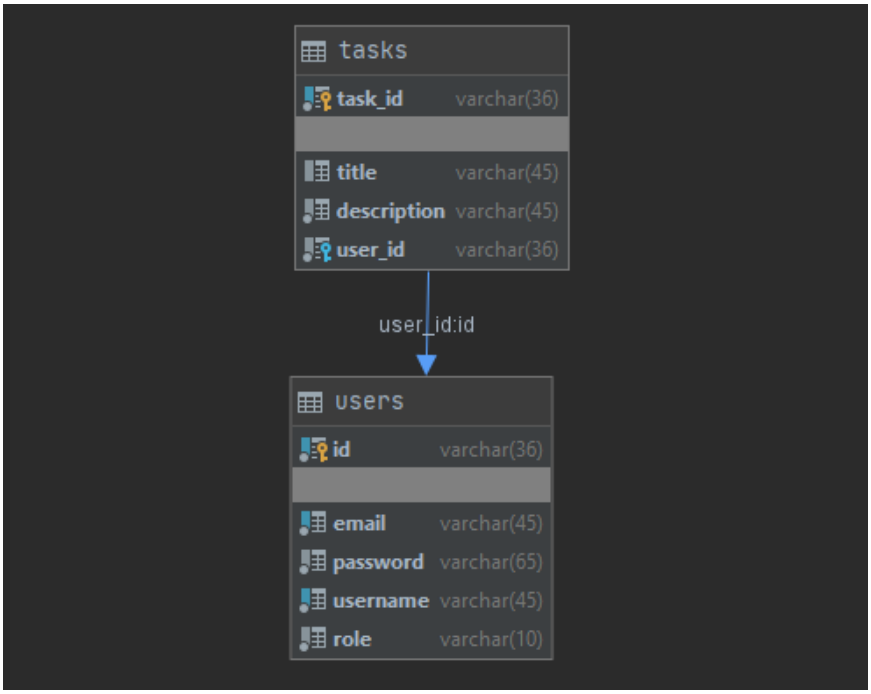

Рисунок 2 ER-модель бази даних

Для показового випадку, зберігатимемо в таблиці “Users” окрім унікального ідентифікатору (id), електронної пошти (email), паролю (password), логіну (або імені користувача – username), ще й role. Насправді, в даному випадку роль буде тільки одна – «ROLE_USER», проте зазвичай їх може бути більше. Оскільки часто в мобільних застосуваннях потрібна авторизація користувачів системи – ми реалізуємо її і тут, тільки на більш простому рівні. В таблиці “Tasks”, в свою чергу, матимемо унікальний ідентифікатор (task_id), назву «завдання» (title), опис «завдання» (description), і Foreign Key – user_id (який зв’язуватиме таблиці між собою зв’язком «один до багатьох»).

Перейдемо до огляду серверу. Архітектуру всередині розроблюваного в цій роботі серверу можна умовно розділити на такі складові частини:

- «контролери» – класи, які відповідатимуть за обробку запитів користувача;
- сервіси – класи бізнес-логіки і їх інтерфейси;
- класи та інтерфейси доступу до даних – класи, що відповідають за звернення за даними до СУБД;

- моделі - класи представлення сутностей;
- класи роботи із Spring Security – фільтр, провайдер JWT-токенів, клас конфігурації, клас-обробник помилок при авторизації, всі допоміжні класи для них;
- допоміжні класи-утиліти – класи помилок, RowMapper-и, клас конфігурації роботи з jdbc (програмний інтерфейс для підключення до бази даних), класи валідації запитів користувача тощо.

Оскільки їх досить багато, розглянемо деякі з них.

Наступний клас представляє собою «контролер» - клас, позначений анотацією @RestController. (Рис. 3)

```
@RestController
@CrossOrigin
@RequestMapping(Constants.TASKS_URLS)
public class TasksController {

    TaskService taskService;
    IAuthenticationFacade authenticationFacade;

    @Autowired
    public TasksController(TaskService taskService, IAuthenticationFacade authenticationFacade) {
        this.taskService = taskService;
        this.authenticationFacade = authenticationFacade;
    }

    @PostMapping
    public ResponseEntity<?> addTaskToUserProfile(@RequestBody TaskToCreate taskToCreate) {
        TaskValidator.validate(taskToCreate);

        taskService.addTask(authenticationFacade.getUserId(), taskToCreate);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @GetMapping
    public ResponseEntity<List<Task>> getTasksOfUser() {
        List<Task> res = taskService.getTasksByUserId(authenticationFacade.getUserId());
        return new ResponseEntity<>(res, HttpStatus.OK);
    }
}
```

Рисунок 3. Лістинг коду: клас-обробник запитів

Таких класів в нашому сервері буде два: перший відповідає за запити, url яких починаються з “…/api/tasks” (Рис. 3). Відповідно, в ньому є два методи, кожен з яких викликатиметься при різних методах HTTP-запиту – POST та GET

на рисунку відповідно. Другий, схожий на попередній, але відповідатиме за аутентифікацію користувача і url якого починається з “…/api/auth”. У останньому буде два методи, один з яких відповідатиме за приставку до url “/login” (відповідно, оброблюватиме запити для аутентифікації), а другий – “/signup” (оброблюватиме запити для реєстрації користувача в системі). Варто зазначити, що останній зі згаданих методів використовуватимемо лише для додавання користувачів в систему вручну (в даному випадку, за допомогою ПЗ Postman, яке дозволяє програмно відсилати будь-які запити) і для спрощення не будемо реалізовувати реєстрацію користувача через мобільні додатки, які будуть створені, оскільки така функція з точки зору реалізації є майже аналогічною до реалізації аутентифікації користувача, тому вважатимемо її не обов’язковою.

Розглянемо також наступний фрагмент коду (Рис. 4):

```
@Repository
public class UserDaoImpl implements UserDao {
    private final JdbcTemplate jdbcTemplate;
    private final String SELECT_QUERY = "SELECT id, email, password, username, role " +
            "FROM users ";
    private final String INSERT_QUERY_SINGLE = "INSERT INTO users (id, email, password, username, role) " +
            "VALUES (?, ?, ?, ?, ?);";
    @Autowired
    public UserDaoImpl(JdbcTemplate jdbcTemplate) { this.jdbcTemplate = jdbcTemplate; }

    @Override
    public User getByUsername(String username) {
        try{
            return jdbcTemplate.queryForObject( sql: SELECT_QUERY +
                                                "WHERE username = ? ;",
                    new Object[]{username}, new UserRowMapper());
        } catch (EmptyResultDataAccessException exception){
            throw new EmptyResultException("User Not found with such username: " + username);
        } catch (Exception e){
            throw new DbException(e.getMessage());
        }
    }
```

Рисунок 4. Лістинг коду - частина класу доступу до даних

На Рисунку 4 зображений лістинг частини класу доступу до даних, на якому зображений один з методів цього класу призначений для отримання користувача за іменем. Такий метод потрібен для того, щоб вірно робити

спробу аутентифікувати користувача. Аналогічні методи, окрім цього класу, знаходяться і в схожому класі для доступу до даних – TaskDaoImpl.

Хотілось би звернути увагу також на один з класів конфігурації Spring Security (Рис. 5):

SecurityConfiguration.java

```
42  @  public void configure(WebSecurity webSecurity) {
43         webSecurity
44                 .ignoring()
45                 .antMatchers(
46                         Constants.AUTH_URLS
47                 );
48     }
49
50     @Override
51  @  protected void configure(HttpSecurity http) throws Exception {
52         http
53                 .httpBasic().disable() HttpSecurity
54                 .csrf().disable()
55                 .exceptionHandling().authenticationEntryPoint(unauthorizedHandler) ExceptionHandlingConfig
56                 .and() HttpSecurity
57                 .sessionManagement() SessionManagementConfigurer<HttpSecurity>
58                 .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
59                 .and() HttpSecurity
60                 .authorizeRequests().requestMatchers(CorsUtils::isPreFlightRequest).permitAll() Expressi
61                 .anyRequest().authenticated()
62                 .and() HttpSecurity
63                 .anonymous() AnonymousConfigurer<HttpSecurity>
64                 .and() HttpSecurity
65                 .cors() CorsConfigurer<HttpSecurity>
66                 .and() HttpSecurity
67                 .apply(new JwtConfigurer(jwtTokenProvider));
68     }
```

Рисунок 5. Лістинг коду - частина класу конфігурації Spring Security

В даних методах відбувається виклик та передача в проміжні методи потрібних значень для конфігурації захисту доступу до API. Перший метод визначає потрібні для ігнорування захисту шляхи до нашого API. Під значенням змінної класу-утиліти Constants.AUTH_URLS криється значення “/api/auth/**”. Передавання цього значення в метод antMatchers визначить, що для всіх шляхів до ресурсів у нашому API, які починаються з цього значення захист буде ігноруватись, тому доступ до таких ресурсів буде дозволеним навіть неаутентифікованим та неавторизованим користувачам. У другому ж

методі ми визначаємо потрібні нам настройки захисту нашого сервісу, такі, як передача обробників неавторизованих запитів, визначення політики захисту без сесій, настройки CORS (Cross-origin resource sharing – інструмент захисту від запитів з конкретних доменів), та ін.

Схема роботи з сервером наступна (Рис. 6):

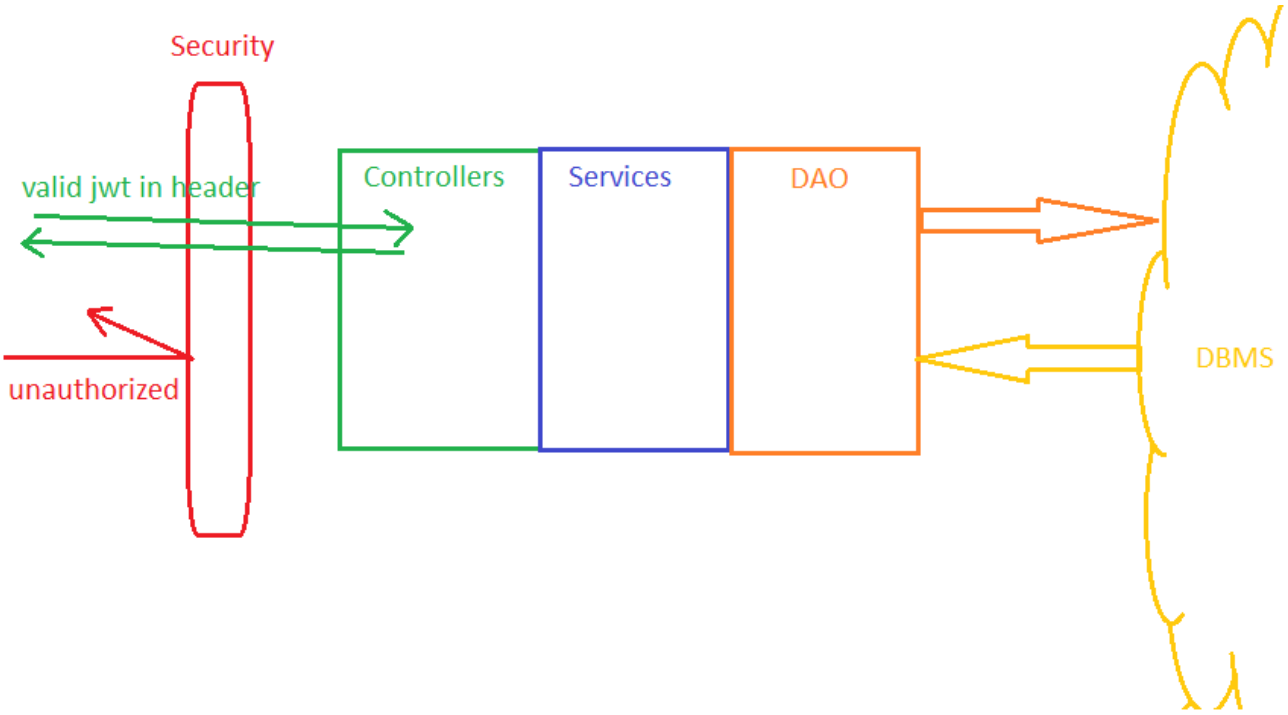

Рисунок 6. Схема взаємодії із сервером

На Рисунку 6 «DBMS» означає Система керування базою даних у перекладі з англійської. В свою чергу, DAO – Data Access Object – об’єкт доступу до даних (відповідає за надсилання визначених в його методах запитів до бази даних).

Результатом реалізації серверу став такий реалізований процес взаємодії із ним:

- кожен запит на ресурс, який не ігнорується Spring Security повинен містити JSON Web Token (JSON-об’єкт, який складається із заголовку, корисного навантаження та підпису, вважається одним із безпечних способів комунікації між двома учасниками [23]). У разі, успішної валідації токену та

відповідності конкретному користувачу, наявному в базі даних, запит проходить на потрібний метод контролеру. Якщо валідація завершилась невдало – запит відхиляється назад із HTTP-кодом відповіді 401 - Unauthorized;

- кожен запит на ресурс, який ігнорується Spring Security, вільно досягає контролеру. При цьому, доступ до ресурсу із URI “api/auth/login” шляхом передачі в тілі запиту імені користувача та паролю, при наявності в базі даних буде повертати в тілі відповіді новосформований токен на основі даних про унікальний ідентифікатор користувача, його імені, пошти та ролі в системі (ці дані будуть міститись в корисному навантаженні токену). Цей токен зберігатиметься мобільним додатком користувача у захищеному локальному сховищі мобільного пристрою, і його наявність буде ідентифікувати успішну аутентифікацію та авторизацію (в даному випадку, умовну, бо роль одна) користувача в системі.

Під час проектування серверу було вирішено розгорнути сервіс у «хмарі», використовуючи Heroku – cloud-платформу яка дозволяє розміщувати свої застосування на віддаленому сервері. Таким чином всі наступні розроблені мобільні додатки зможуть з легкістю в будь-який момент звертатись до серверу, до того ж, використовуючи протокол HTTPS.

Процес розгортання серверу виявився нескладним через взаємодію Heroku з Github – веб-сервісом зберігання коду та використання системи контролю версій Git, куди й завантажуватимемо наші проекти під час розробки.

3.4 Опис реалізації додатку, побудованого на React Native

React Native – фреймворк з відкритим кодом для створення мобільних додатків, який дозволяє створювати кросплатформенні мобільні застосування для IOS та Android на основі React. [24] В свою чергу, React – JavaScript-бібліотека для створення користувацьких інтерфейсів. [25]

React Native створений компанією Facebook. На ньому написані багато відомих мобільних додатків. Наприклад – Facebook, Instagram, Pinterest, Tesla, Uber Eats, Skype, Discord, та багато інших. [24]

Рендерити – (з англ. render - візуалізувати) означає промальовувати програмно на користувацькому інтерфейсі.

Використовуючи React Native, застосунки пишуться на мові JavaScript використовуючи примітиви React. React Native використовує декларативну UI парадигму React. [24] Це означає що створення інтерактивних інтерфейсів відбувається за рахунок декларативного опису елементів. При цьому React ефективно оновлює і рендерить лише ті компоненти, які треба оновити при зміні якихось даних.[25]

При цьому фреймворк надає основний набір елементів, таких як View, Text, Image, що рендеряться в прямо в нативні UI-блоки платформи. Деякі з них та їх відповідності нативним елементам представлені на Рисунку 7:

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrollling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	A generic scrolling container that can contain multiple components and views
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Allows the user to enter text

Рисунок 7. Відповідності Core-елементів React Native нативним[24]

Такі елементи називаться Core Components, і насправді їх існує набагато більше, ніж представлено на цьому рисунку (Рис. 7).

Важливо відмітити, що для легшої розробки з React Native існує надзвичайно потужна за своїми можливостями та функціоналом платформа Expo. Expo – це набір програмного забезпечення та бібліотек, який допомагає при розробці додатків на React Native.

Серед потрібних нам:

- мобільний додаток Expo, який можна скачати через Google Play Market або App Store;
- Expo Cli – оточення, яке завантажується за допомогою пакетного менеджера npm (також потрібна наявність NodeJs – платформи для виконання мережевих застосунків, на JavaScript [27]).
- Expo Snack – веб-застосунок, який дозволяє не тільки тестувати деякий код прямо в браузері та відображати веб-версію створюваного додатку, а і через наданий сервіс користуватись емуляторами Android та IOS прямо у вікні браузеру. Він стає в нагоді тоді, коли у розробника немає можливості протестувати застосунок на фізичному пристрої якоїсь однієї з платформ, як і у моєму випадку (IOS).

Отже, при завантаженні всього необхідного вищеописаного програмного забезпечення, розробник отримує можливість створювати та управляти конфігурацією React Native проекту за допомогою командного рядка, керуючись командами Expo Cli. Також відкривається можливість за допомогою однієї з варіацій команди “expo start” у командному рядку, запускати локальний сервер. Цей сервер за допомогою мобільного додатку Expo та технології тунелювання, дозволяє в живому часі передавати дані про збудовані застосунки на фізичний мобільний пристрій в одній мережі Wi-Fi з ним. Розробнику необхідно просто відсканувати QR-код (за допомогою звичайної камери на IOS,

або додатку Expo на Android), який з’являється у вікні командного рядка (або у вікні браузеру, яке відкривається при запуску серверу, де наявна вся інформація про поточний стан застосунку, всі підключені пристрої тощо).

Важливим моментом є те, що підтримується hot reload (з англ. – швидке, «гаряче» перезавантаження), тобто при оновленні коду розробником, зміни швидко, в режимі реального часу, відображаються прямо на фізичному пристрої.

Отже, розглянемо виконаний проект. Написання проекту дуже схоже до написання звичайного веб-застосування з використанням бібліотеки React. Концепція базується на використанні власних класів-компонентів (або ж, новий синтаксис – функціональних компонентів), котрі можуть бути сконфігурованими та заповнені різними даними перед- та під час промальовування на користувацькому інтерфейсі. Прив’язка до даних відбувається з допомогою використання «станів» даних компоненту.

Відслідкувати зміни станів дозволяють «хуки», які являються відносно новим синтаксисом (поточна версія React під час написання – 16.13.1). Хуки введені в React 16.8. Вони дозволяють використовувати стан та інші можливості React без написання класів. [25]

Розглянемо наступний фрагмент коду (Рис 8):

```
export default function Home({navigation, route}){

    const {signOut} = React.useContext(AuthContext);

    const [toLoad, setToLoad] = useState(true);
    const [tasks, setTasks] = useState([]);

    useEffect(() => {
        async function getData(){
            let userToken;
                try {
                    userToken = await AsyncStorage.getItem('userToken');
                } catch (e) {
                    console.log(`Failed to restore token: ${e}`)
                }
            fetch("https://taskerappbc.herokuapp.com/api/tasks",
            {
                method: 'get',
                headers: new Headers({
                    'Authorization': 'Bearer ' + userToken
                })
            })
            .then((res) => res.json())
            .then(res => setTasks(res))
            .catch((error) => {
                console.log(error);
                setTasks([]);
            });
        }
        getData();
    },[toLoad]);
```

Рисунок 8. Лістинг коду - частина функції компоненти

На даному фрагменті ми визначаємо частину функції під назвою Home, яка відповідає за рендеринг компоненти головної сторінки. Хук useEffect дозволяє виконати функцію всередині (яка, в свою чергу надсилає запит на віддалений сервер для отримання даних про список завдань даного користувача), при зміні значення стану записаного в змінній toLoad. Даний хук автоматично виконується при ініціалізації компоненти, а також при кожній наступній зміні toLoad, яку ми використовуємо як маркер для отримання даних від серверу в потрібний нам момент.

На рисунку 8 також можна замітити використання AsyncStorage – постійного локального сховища даних. Його основна функція – зберігати дані локально на пристрої, які не знищуються при закритті застосунку. Саме сюди вирішено класти JWT токен при аутентифікації користувача.

Також на попередньому фрагменті ми бачимо змінну signOut, яку ми отримуємо з контексту застосунку. Вона являє собою функцію яка змінить стан маркеру стану в іншому модулі, та дозволить нам видалити токен користувача з

локального сховища та здійснити навігацію на сторінку введення логіну та паролю.

Хотілось би звернути увагу і на наступний лістинг коду (Рис. 9):

screens > JS home.js

```
return (
    <ImageBackground source ={require('../assets/background.jpg')} style={globalStyles.background}>
        <View style = {globalStyles.container}>
        <MaterialIcons style={styles.modalOpen} name='close' size={20} onPress={() => logOut()}></MaterialIcons>
        <MaterialIcons style={styles.modalOpen} name='add' size={22} onPress={() => setToggledModal(true)}></MaterialIcons>
            <Modal animationType='slide' visible={toggledModal}>
            <TouchableWithoutFeedback onPress={Keyboard.dismiss}>
                <View style={styles.modal}>
                    <MaterialIcons name='close' style={styles.modalHide} size={22} onPress={() => setToggledModal(false)}>
                    </MaterialIcons>
                    <AddForm addTask = {addTask}></AddForm>
                </View>
            </TouchableWithoutFeedback>
            </Modal>
            <FlatList
                data={ tasks }
                keyExtractor = { (item) => item.taskId}
                renderItem = {({ item }) => (
                    <TouchableOpacity onPress={() => navigation.push('TaskDetails',
                    {taskId: item.taskId,
                    title: item.title,
                    description: item.description})}>
                        <Card>
                            <Text style={globalStyles.text}>
                                {item.title}
                            </Text>
                        </Card>
                    </TouchableOpacity>
                )}
            />
        </View>
    </ImageBackground>
);
```

Рисунок 9. Лістинг коду - функція промальовування компоненти

На зображеному фрагменті лістингу коду ми можемо бачити фунцію, яка декларує вигляд та розмітку компоненти з попереднього рисунку. Перед вами синтаксис JSX – декларативний синтаксис, який дозволяє визначати шаблон, відповідно до якого будуть промальовані дані. Він включає в себе інші складові компоненти, в які ми передаємо необхідні їм параметри – «пропси».

Важливим елементом на Рисунку 9 є використання компоненти FlatList. Така компонента зручно для розробника містить в собі переданий в неї список однакових елементів, які виводяться з необхідними розробнику параметру на екран.

Для огляду визначення стилів компоненти розглянемо наступний фрагмент коду (Рис. 10):

```
const styles = StyleSheet.create({
    modal: {
        fontSize: 30,
        flex: 1
    },
    modalOpen:{
        borderColor: '#8420CE',
        borderWidth: 1,
        marginBottom: 10,
        padding: 10,
        borderRadius: 10,
        alignSelf: 'center'
    },
    modalHide: {
        borderColor: '#8420CE',
        borderWidth: 1,
        padding: 10,
        borderRadius: 10,
        alignSelf: 'center',
        marginTop: 20,
        marginBottom: 0
    }
});
```

Рисунок 10. Лістинг коду - визначення стилів у React

Даний фрагмент показує спосіб декларування стилів складових елементів, які ми використовуємо у розмітці JSX. Таку змінну стилів можна визначати як локально для кожної компоненти, так і глобально по застосунку, експортуючи змінну в інші модулі.

Для руху між екранами використовується бібліотека “react-navigation”. Вона підтримує нативні жести та анімації, необхідні для красивішого відображення на екрані. [28]

У React Native існує можливість класичного способу навігації – стеками. Цей механізм є базовим у мобільній розробці. При необхідності перейти на іншу сторінку, необхідний елемент поміщається у стек. Таким чином при поверненні відбувається діставання верхнього елементу зі стеку. Так зберігається історія переходів користувача між вікнами. При необхідності можна використовувати декілька зв’язаних між собою стеків, або інші комбінації навігаційних елементів.

Окрім стеків, існують і інші типи навігації, такі як вкладки, модальні вікна тощо.

Запустимо наше мобільне застосування (Рис. 11):

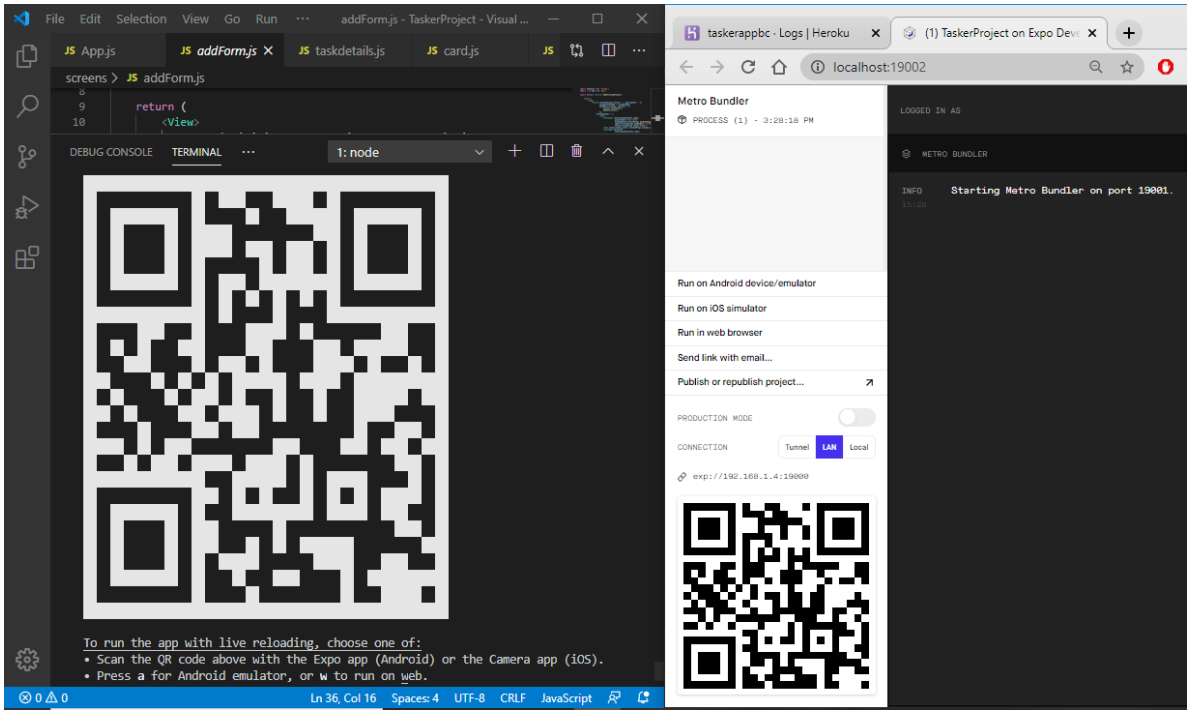

Рисунок 11. Тестовий запуск застосування - зразок QR-коду на момент запуску

Відсканувавши QR-код (Рис. 11), на телефоні після завантаження збірки, відкриється створене застосування. Оскільки користувач не авторизований на даний момент, відповідно, перед нами вікно логіну (Рис. 12.А). При спробі ввести логін або пароль користувачів, яких нема в базі даних, буде виведене повідомлення про те, що користувача не знайдено (Рис. 12.Б). Відповідно, у консолі на сайті Heroku ми бачимо відповідний запис, підтверджуючий про відсутність такого користувача (Рис. 13):

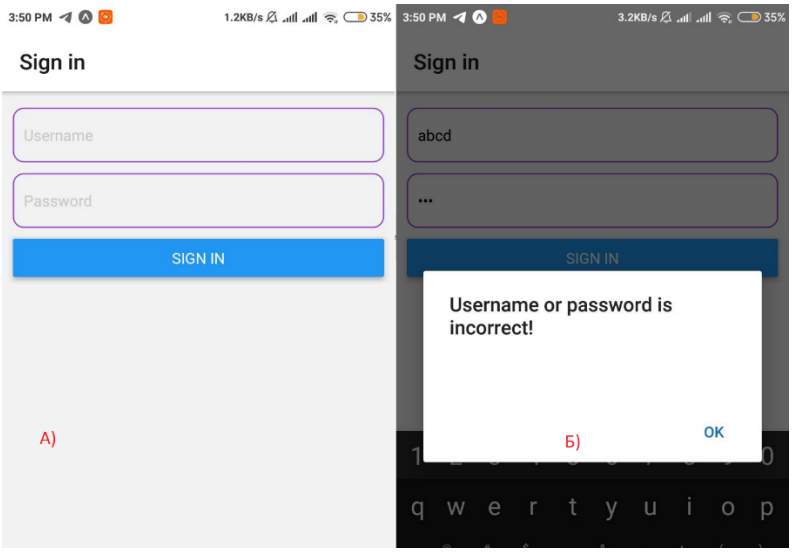

Рисунок 12. (А, Б) - Вікно логіну та невірно введений логін або пароль

```
2020-05-08T12:50:56.065539+00:00 heroku[router]: at=info method=POST path="/api/auth/login"
host=taskerappbc.herokuapp.com request_id=343ffbeb-9619-49e1-aaa1-ff99e1774912 fwd="188.163.8.152"
dyno=web.1 connect=1ms service=236ms status=404 bytes=427 protocol=https
```

Рисунок 13. Консоль серверу у хмарі

Отже, інтерфейс та функціонал застосування є наступними (Рис 14, 15):

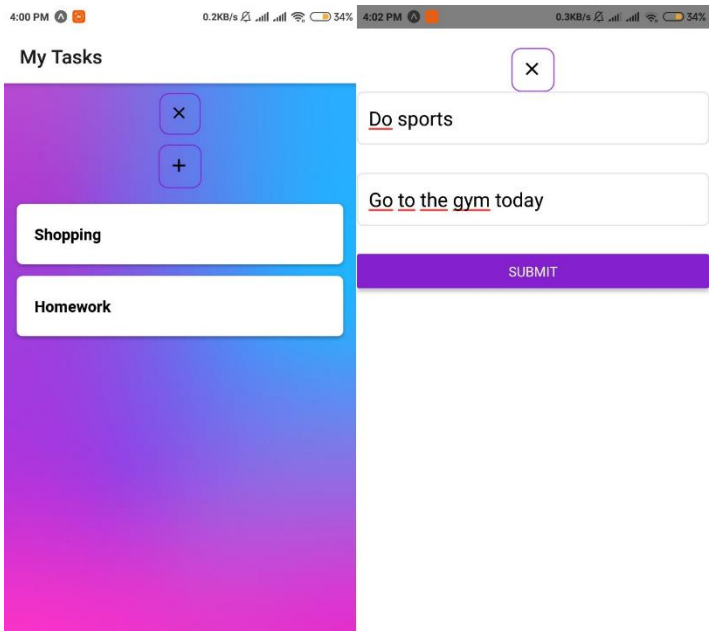

Рисунок 14. Скріншоти екрану пристрою: головна сторінка та модальне вікно додавання "завдання"

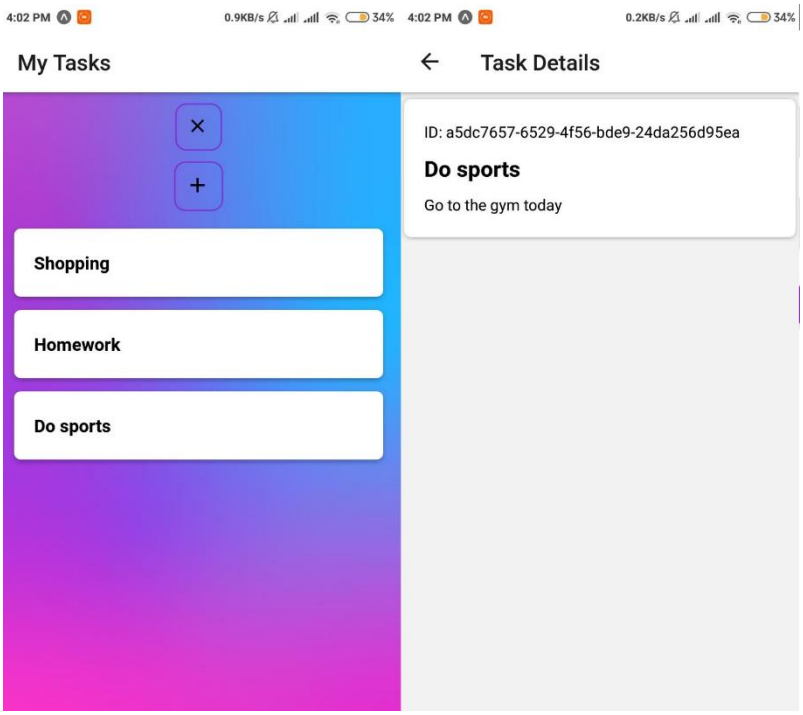

Рисунок 15. Скріншоти екрану пристрою: головне вікно після додавання завдання, перегляд новоствореного завдання

На цих рисунках бачимо процес перегляду списку завдань на головному екрані та додавання нового завдання використовуючи модальне вікно (Рис. 14), а також стан списку завдань користувача після додавання завдання і перегляд конкретної замітки з завданням.

Після виходу з застосування не натискаючи кнопку виходу з акаунту “X” – при новому відкритті застосунку, ми опинимось на головній сторінці, де знову запитом із серверу будуть взяті свіжі дані про наші завдання. У разі натискання на вищезгадану кнопку, користувач виходить із акаунту, його токен видаляється із локального сховища, і йому відкривається вікно логіну аналогічне вікну на рисунку 12.А. Якщо при цьому не вводити логін та пароль і вийти з застосунку, то після перезапуску перед нами знову буде вікно логіну, адже даних про токен в сховищі не буде. Після проведених операцій бачимо такі дані в базі даних, що підтверджує успішну роботу застосунку:

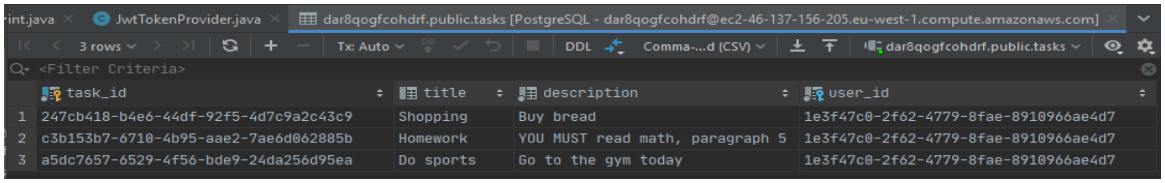

Рисунок 16. Стан бази даних після проведених дій

Потрібно також відмітити, що мною була зроблена спроба тестувати застосунок на платформі IOS у онлайн-середовищі Snack від Expo, але через відсутність деяких npm-модулів, наявних в цьому середовищі, це виявилось поки неможливим.

3.5 Опис реалізації додатку, побудованого на Xamarin Forms

Xamarin – фреймворк для кросплатформеної розробки мобільних застосувань (IOS, Android, Windows Phone), з використанням мови програмування C#. [29]

На Xamarin побудовані такі додатки, як Storyo, Just Giving, The World Bank, Olo, Insightly, та ін. [33]

Фреймворк складається з кількох частин:

- Xamarin.IOS – бібліотека класів C#, яка представляє розробнику доступ до IOS SDK;
- Xamarin.Android – бібліотека класів, яка дає доступ до Android SDK;
- Компілятори Android та IOS.

Також існують плагіни для IDE XCode таVisual Studio, а також IDE Xamarin Studio. У своїй розробці я використовував Visual Studio 2019 із необхідними аддонами для розробки на Xamarin Forms.

Xamarin Forms – платформа користувацького інтерфейсу з відкритим кодом, яка дала можливість покращити розробку на Xamarin шляхом

збільшення кількості спільного для платформ коду. За допомогою Xamarin.Forms, розробники можуть створювати застосування для IOS, Android, та Windows на основі спільної бази коду. [30]

При цьому, за словами створювачів Xamarin Forms, можна досягти використання більш ніж 75% спільного для платформ коду. [30]

На жаль, виявилось, що для запуску та навіть тестування додатків на платформі IOS, необхідний комп'ютер з ОС MacOS. Через фізичну відсутність в наявності такого пристрою у мене, можливості тестування додатку відкрились лише для платформ Android та Windows.

Писати додатки на Xamarin Forms прийнято використовуючи відомий в осередку розробки на C# патерн проєктування MVVM. [30]

Model, View, ViewModel (MVVM) – шаблон, який складається з трьох компонентів – моделі (Model), моделі представлення (ViewModel) та представлення (View). Модель відповідає за опис даних у застосуванні, представлення визначає візуальний інтерфейс, до якого прив'язана модель представлення. Модель представлення реалізовує логіку необхідну для роботи з даними та оновлення даних. [31]

Для представлення використовується відома мова розмітки XAML (eXtensible Application Markup Language).

Перейдемо до огляду проєкту. При створенні застосунку на Xamarin Forms перед нами відкривається Solution (з англ. – рішення) – набір проектів. Структура є наступною (Рис. 17):

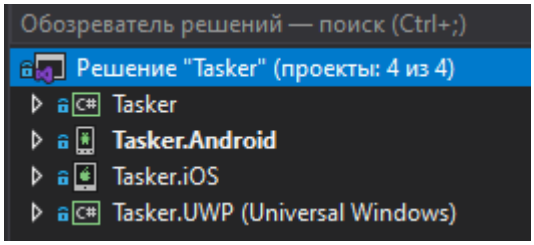

Рисунок 17. Структура рішення в Xamarin Forms

Як можна побачити з фрагменту екрану (Рис. 17), у «рішенні» міститься 4 проекти. Перший (верхній) – бібліотека коду, спільна для наступних проектів.

Наступні три – проекти коду для кожної з платформ (Android, IOS, Uniwersal Windows), назвемо їх залежними проектами.

Спільний для всіх платформ код пишеться у першій бібліотеці, специфічний код для кожної з платформ пишеться у кожному з відповідних залежних проектів. Відповідно, в автоматично згенерованих проектах уже є потрібний код для початкового запуску застосування, у залежних проектах є посилання на спільну бібліотеку.

Отже, розглянемо деякі наступні елементи написаного коду. Оглянемо наступний фрагмент коду (Рис. 18):

View.xaml.cs | TasksView.xaml | LoginView.xaml | TaskDetailsView.xaml | TaskDetailsView.xaml.cs

ntentPage | ContentPage

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3                xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4                xmlns:d="http://xamarin.com/schemas/2014/forms/design"
5                xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6                xmlns:viewmodels="clr-namespace:Tasker.ViewModels"
7                mc:Ignorable="d"
8                x:Class="Tasker.Views.LoginView"
9                Title="Sign in">
10      <ContentPage.BindingContext>
11          <viewmodels:LoginViewModel></viewmodels:LoginViewModel>
12      </ContentPage.BindingContext>
13
14      <StackLayout Orientation="Vertical" VerticalOptions="Center" Spacing="20" Padding="20">
15          <Entry Text="{Binding Username}" Placeholder="Usename"></Entry>
16          <Entry Text="{Binding Password}" Placeholder="Password" IsPassword="True"></Entry>
17          <Button Command="{Binding LoginCommand}" Text="Sign In" FontSize="30"></Button>
18      </StackLayout>
19
20
21  </ContentPage>
```

Рисунок 18. Лістинг коду - XAML-розмітка представлення вікна логіну

Даний файл містить в собі XAML-розмітку для вікна, користувач може авторизуватись. В ньому є елемент оформлення розміщення у вигляді стеку (StackLayout), та елементи керування які доступні користувачу – два поля введення тексту для імені користувача та паролю, а також кнопка. В рядку під номером 11 можна замітити оголошення прив’язки до моделі представлення.

Відповідно, поля Username, Password та LoginCommand будуть доступні із цього класу. В даному випадку присутня двостороння прив’язка даних, тобто введені користувачем в полях дані будуть автоматично в режимі реального часу змінювати дані у полях класу моделі представлення. Це досягається завдяки реалізації інтерфейсу INotifyPropertyChanged у класах моделей представлення.

Оглянемо тепер клас моделі представлення логіну користувача (Рис. 19):

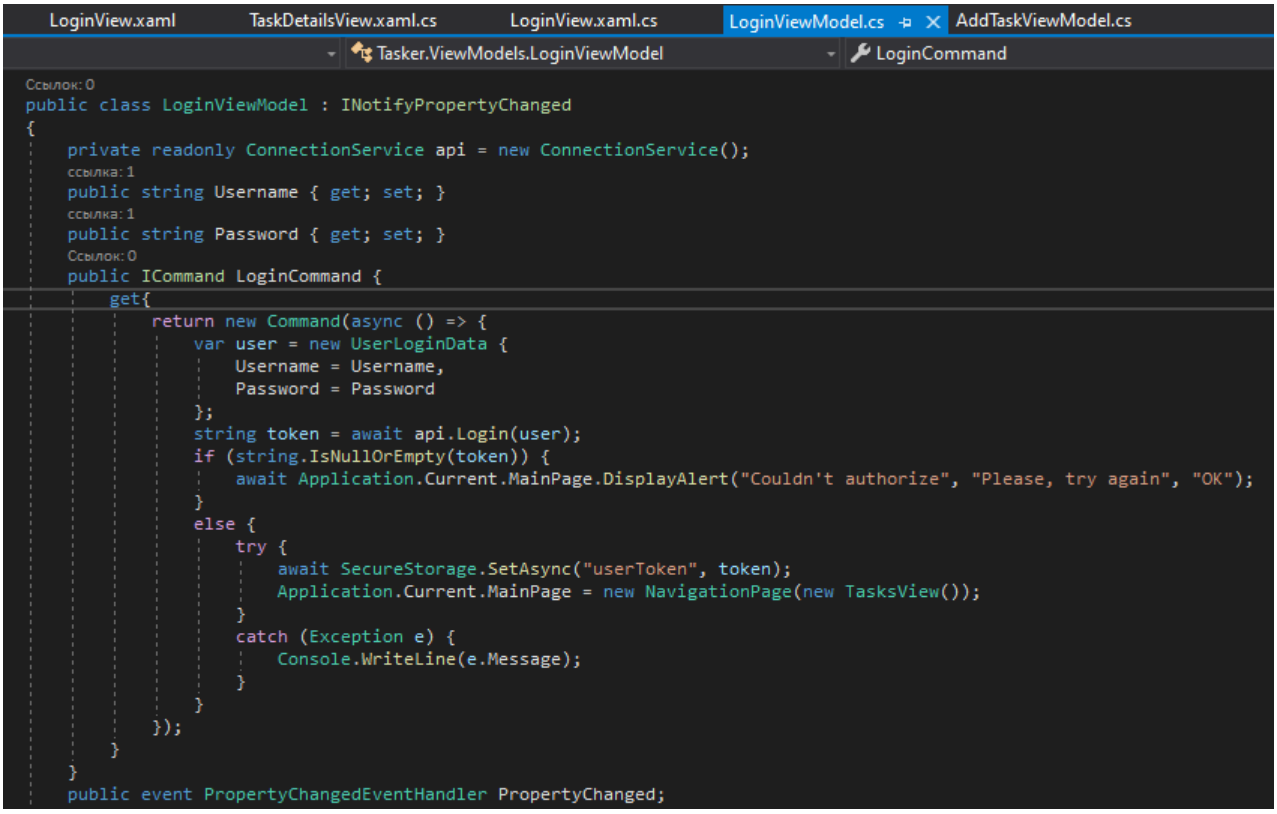

Рисунок 19. Лістинг коду: фрагмент класу моделі представлення логіну

Даний фрагмент коду з класу LoginViewModel містить три властивості (синтаксис C#, поля класу із визначеними функціями селекторів та модифікаторів) – Username, Password, та LoginCommand. Остання властивість являється командою, яка виконується під час натискання на кнопку “Sign In” на екрані авторизації.

Окрім згаданих вище, в проекті містяться класи моделей Завдання, Запиту для логіну та Відповіді на запит логіну, допоміжні класи до

представлень, класи моделей представлень, та інші допоміжні класи. Варто згадати також клас сервісу для з'єднання з сервером. Розглянемо приклад одного з методів (Рис. 20):

```
public const string BASE_ADDRESS = "https://taskerappbc.herokuapp.com/api/";
public const string TASKS_ADDRESS = "tasks";
ссылка: 1
public async Task<List<MyTask>> GetTasks(string accessToken)
{
    var client = new HttpClient();
    client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(
        "Bearer", accessToken);

    var json = await client.GetStringAsync(BASE_ADDRESS + TASKS_ADDRESS);
    var tasks = JsonConvert.DeserializeObject<List<MyTask>>(json);

    return tasks;
}
```

Рисунок 20. Лістинг коду - метод для отримання списку завдань поточного користувача

В такому методі надсилаємо запит на сервер із переданим раніше токеном користувача в заголовку запиту. Обробка пустого негативної відповіді відбувається в класі моделі представлення, що його викликає.

Для збереження токену використовуємо бібліотеку Xamarin.Essentials, яку можна вільно скачати через менеджер пакетів NuGet. Нас цікавить клас SecureStorage. При компіляції в Андроїд він використовуватиме захищене за допомогою MD5 шифрування локальне сховище спільних настройок. [30] До них програмно може доступитись в пристрої лише застосунок, який його створив.

Перейдемо до тестування застосунку. Для цього можна використовувати USB-з'єднання пристрою з ПК. IDE Visual Studio 2019 легко розпізнає пристрій (на якому в настройках увімкнено режим розробника то доступу через USB), та дає можливість запустити застосунок на ньому.

Для навігації в застосунку, знову ж таки обраний один із можливих класичних способу навігації – стек навігації. Варто відмітити, що всі стандартні інструменти для навігації вже присутні і нічого додатково завантажувати не треба. Для цього у будь який момент можна доступитись до оточення всіх

сторінок, до поточної сторінки та об’єкту Navigation який містить кожен із таких екранів. Це досить зручно і не вимагає додаткових зусиль при оформленні навігації в застосуванні.

Маємо наступний інтерфейс та функціонал (Рис. 21, 22, 23):

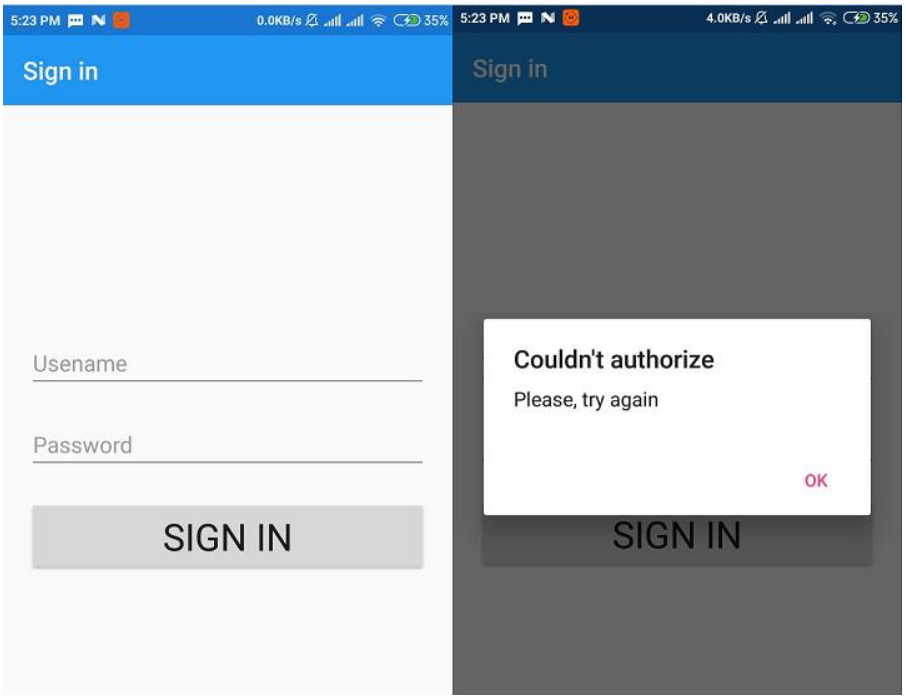

Рисунок 21. Скріншоти інтерфейсу застосунку: вікно логіну та неправильно введені дані

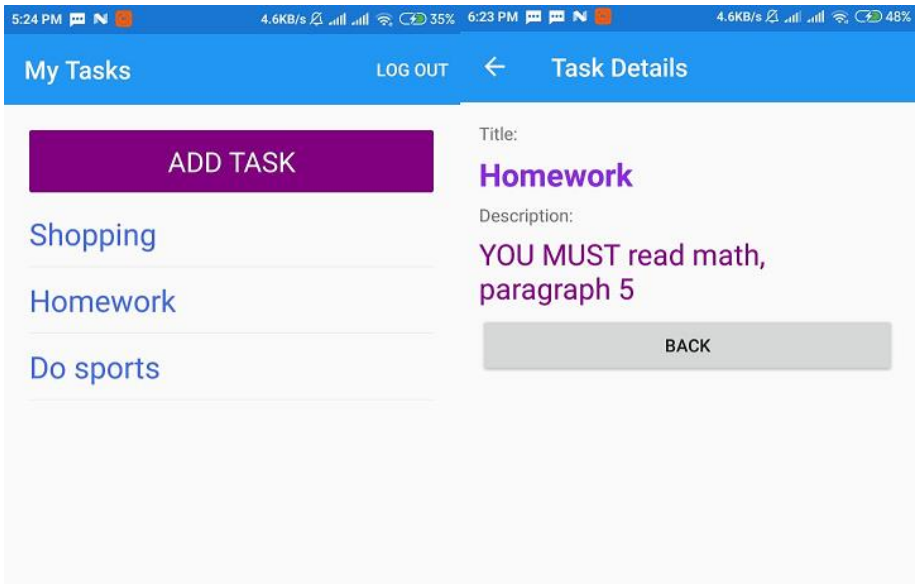

Рисунок 22. Скріншоти інтерфейсу програми: головний екран та екран перегляду завдання

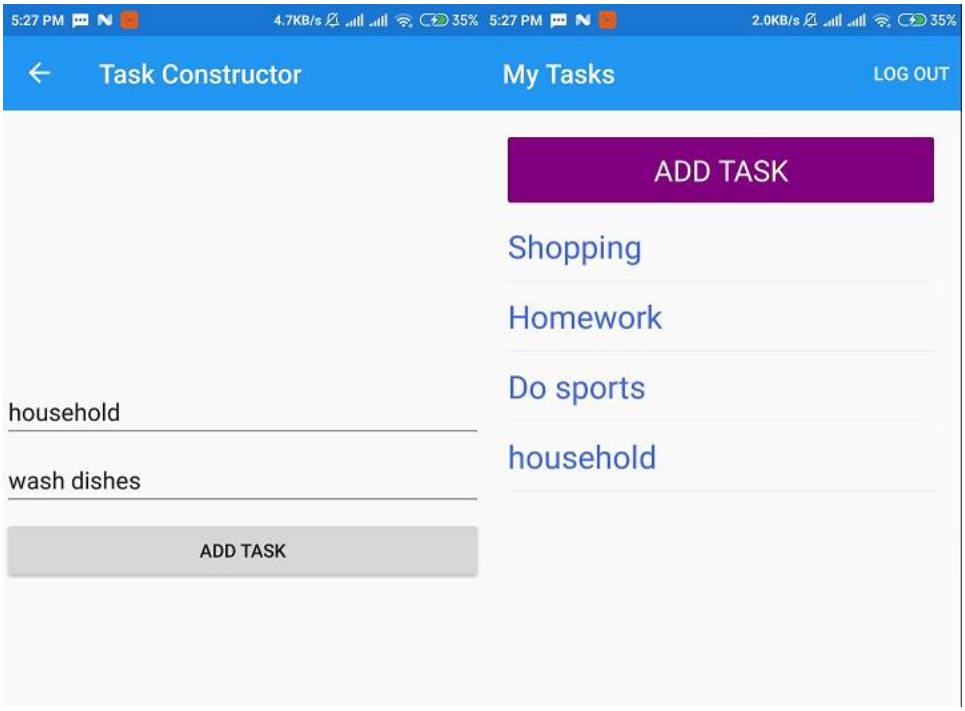

Рисунок 23. Скріншоти інтерфейсу програми: правильне додавання завдання та результат додавання

Аналогічно попередньому застосунку, маємо набір зі схожих вікон та з такими ж діями. Також при виході з застосування без виходу з акаунту, при повторному запуску за0441тосування відкриє одразу головний екран.

Переглянемо базу даних, щоб засвідчитись, що нове завдання додано (Рис. 24):

4 rows | Tx: Auto | DDL | Comma-...d (CSV) | dar8qogfcohdrf.public.tasks

<Filter Criteria>

	task_id	title	description	user_id
1	247cb418-b4e6-44df-92f5-4d7c9a2c43c9	Shopping	Buy bread	1e3f47c0-2f62-4779-8fae-8910966ae4d7
2	c3b153b7-6710-4b95-aae2-7ae6d062885b	Homework	YOU MUST read math, paragraph 5	1e3f47c0-2f62-4779-8fae-8910966ae4d7
3	a5dc7657-6529-4f56-bde9-24da256d95ea	Do sports	Go to the gym today	1e3f47c0-2f62-4779-8fae-8910966ae4d7
4	6777a6ab-4eb1-4e64-81e3-aa96557add4f	household	wash dishes	1e3f47c0-2f62-4779-8fae-8910966ae4d7

Рисунок 24. Скріншот даних у базі даних

3.6 Опис реалізації нативного додатку Android

Для написання додатку для платформи Android використовуватимемо Android SDK та мову Java. Для розробки також можна використовувати мову Kotlin. Це набір інструментів для розробки, який використовується для написання додатків на цій платформі. Він включає в себе потрібні бібліотеки, дебагер, емулятор, документацію для Android API, необхідні інструменти з заготовками коду, та ін. [32]

Для створення використовуватимемо IDE Android Studio. Це зручний інструмент з усіма необхідними інструментами.

Отже, в Android SDK, основним рушієм створення екранів є Activities – (з англ. досл. - активності). Кожна окрема активність являє собою представлення екрану з деякою розміткою у форматі xml, Java-класом, та зв’язаними з ним ресурсами.

Android-проект конфігурується файлом маніфесту у форматі xml (в ньому описані включені в проект активності, їх деякі параметри, а також дозволи програми робити певні дії в системі пристрою), а також використовується у даному випадку менеджер пакетів Gradle. Для останнього існують декілька файлів, важливими з них для нас є два файли з назвою та розширенням «build.gradle» - в них описані потрібні настройки застосунку, версія SDK та залежності, які потрібні для роботи програми. Також в створений проект вже включені необхідні заготовки для Unit-тестування.

Отже, маємо 5 активностей: початкова (буде визначати, куди переходити при запуску застосунку), сторінка авторизації, головна сторінка, сторінка додавання нового завдання та сторінка перегляду завдання. Також в проекті

присутні додаткові класи: клас для надсилання запитів на сервер, модель Завдання, клас помилок та адаптер для списку, який зв’язуватиме дані з пердставленням списку на екрані користувача. У кожної активності є свій життєвий цикл, який супроводжується викликами різних методів, таких, як onCreate, onPause та інші.

Оскільки активності досить громіздкі, розглянемо найменшу з них – Initial Activity. Ця активність викликається при запуску застосунку і вона є невидимою, адже в її методі onCreate ми одразу ж вирішуємо на основі наявності токену у локальному сховищі – куди направити користувача. Перевірка токену відбувається переглядом SharedPreferences – локального сховища даних у вигляді «ключ-значення», яке є приватним для кожного застосунку в системі пристрою. Отже, код активності (Рис. 25):

```
public class InitialActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_initial);

        SharedPreferences pref = getApplicationContext().getSharedPreferences( name: "Tasker",  mode: 0)
        if(pref.getString( key: "token", defValue: null) != null){
            Intent intent = new Intent ( packageContext: InitialActivity.this, MainActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
            startActivity(intent);
        }
        else{
            Intent intent = new Intent ( packageContext: InitialActivity.this, LoginActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
            startActivity(intent);
        }
    }
}
```

Рисунок 25. Лістинг коду: початкова активність

Для навігації між активностями використовуються Intents – об’єкти які дозволяють викликати інші активності та передавати та отримувати дані з них. На Рисунку 25, можемо побачити приклад створення Intent-у, який супроводжується маркерами для не збереження історії переходу, тобто щоб користувач не міг натиснути клавішу «Назад» та парейти на цю активність.

Для відображення списку завдань використовується клас RecyclerView, який оптимізовано показує дані у вигляді списку. Для списку в потрібний момент підвантажуються з колекції (яка отримана від серверу) тільки ті дані, які треба показати на екрані. Для такого класу використовується адаптер для зв’язування списку із даними.

Для відправки даних на сервер використовуються AsyncTask – клас, який в задньому фоні застосування виконує деякі дії і повертає результат, викликаючи потрібний нам метод.

Отже, перед вами інтерфейс користувача та функціонал програми (Рис. 26, 27):

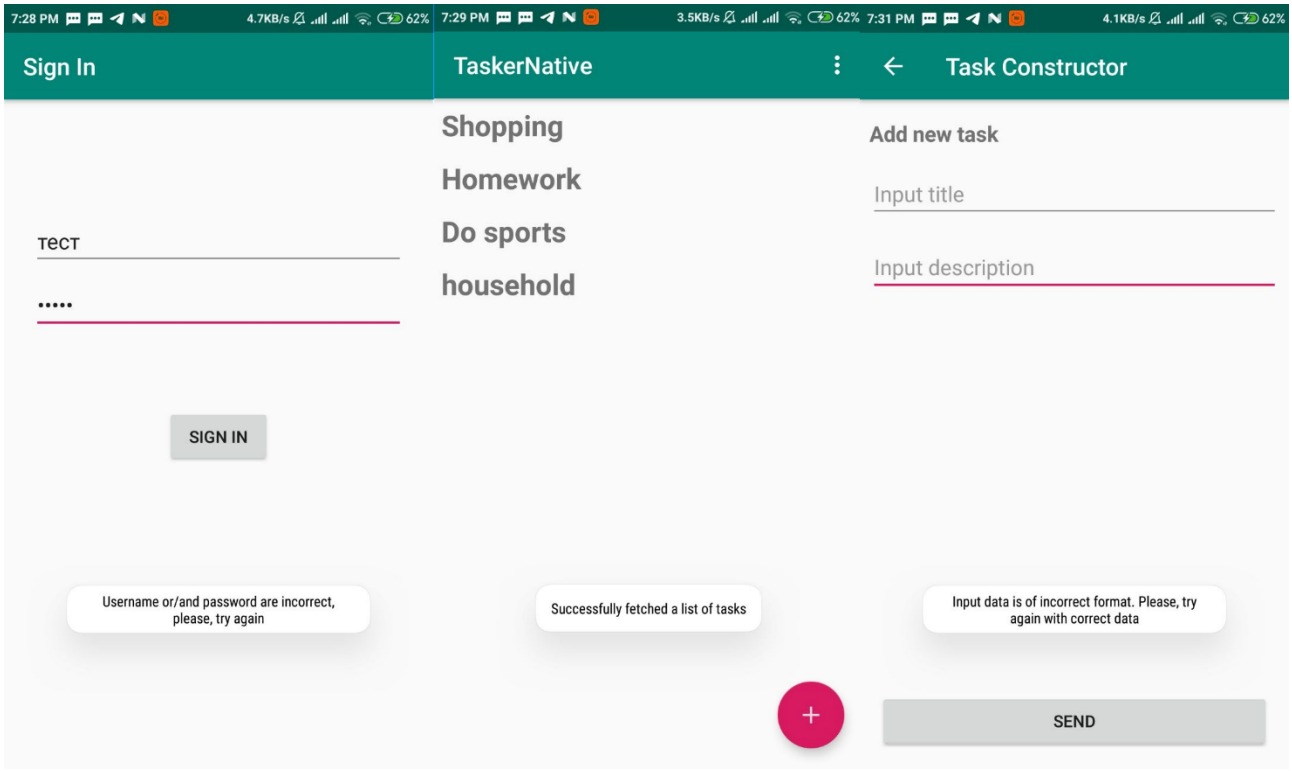

Рисунок 26. Скріншоти програми: екран логіну(з неправильно введеними даними), головний екран зі списком завдань, конструктор завдань з неправильно введеними даними

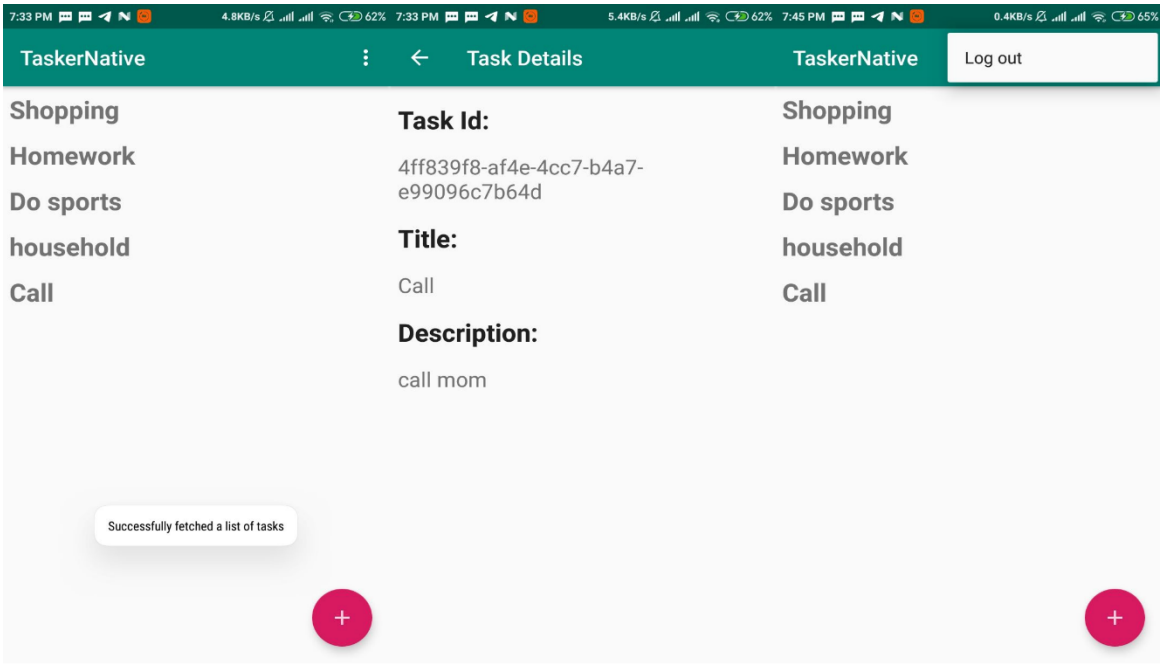

Рисунок 27. Скріншоти програми: головна сторінка після правильно доданого завдання, опис конкретного завдання, натиснутий інструмент на навігаційній стрічці

Як і в попередніх застосунках, матимемо вірно визначене вікно після запуску застосунку на основі присутності чи відсутності токену серед локальних даних застосунку.

3.7 Огляд написання нативних додатків для платформи IOS

Оскільки для нативної розробки для платформи IOS, потрібен комп’ютер з MacOS, що ускладнює ситуацію, в даному випадку вирішено оглянути способи нативної розробки під IOS за допомогою джерел інформації.

Отже серед мов програмування тут існує два варіанти. Мова Swift відносно нова та активно витісняє мову програмування Objective-C, яку багато людей вважає застарілою, але яка використовувалась для написання застосувань вже біля 30 років. [34]

По словам експертів, мова Swift – дуже проста та зрозуміла, дружелюбна до новачків. Вона є строго типізованою. Проте, оскільки за всі роки існування Objective-C назбиралась величезна кількість коду, написаному на останній, навіть зараз існує великий попит для обох вищезгаданих мов. [34]

Для створення застосувань потрібна IDE XCode. Вона безкоштовна, і потребує тільки Apple ID для скачування. [35]

Як і в Android SDK, в IOS SDK існують свої основні набори елементів контролю користувацького інтерфейсу. Тут, аналогічно класу View, існує батьківський для всіх елементів управління клас UIView. Для розробки додаткової поведінки програми, використовується клас UIControl для різних кнопок, перемикачів, і інших views, створених для взаємодії з користувачем. [36]

Так само, існують аналогічним чином влаштовані класи для контейнерів, навігації, візуальних ефектів та ін. [36]

3.8 Порівняння написаних та оглянутих підходів

Оглянувши кожен з вищеописаних методів, можна зробити деякі висновки та перерахувати особливості кожного з них.

Розпочнемо з фреймворку React Native.

Важливо зазначити, що є два підходи до розробки React Native застосунки – bare та managed workflow. Bare workflow означає, що застосунок має повний доступ до всіх нативних функцій SDK платформ, проте він є набагато важчим у написанні та опануванні. Managed workflow використовує Expo SDK та інші інструменти Expo для легшої розробки застосунків, проте він має деякі мінуси.

Також, потрібно згадати, що через фізичну відсутність IOS-пристрою, може стати неможливим тестування додатку для платформи IOS, якщо у проекті наявні деякі сторонні npm-модулі.

При використанні React Native у розробці можна виділити такі переваги та недоліки:

Серед переваг:

1. Кросплатформеність;
2. використання мови програмування JavaScript (за бажання TypeScript) а також бібліотеки React, що дає можливість легкого входження для розробників, знайомих з принципами React;
3. наявність інструментів для полегшеної розробки – Expo. Це має на увазі під собою використання managed workflow;
4. наявність широкої аудиторії розробників-користувачів фреймворком, хорошої та зрозумілої офіційної документації;
5. легкість встановлення засобів для розробки;
6. можливість повного доступу до нативних елементів SDK при використанні bare workflow.

Серед недоліків:

1. Існування наступних обмежень при використанні managed workflow (тобто, при використанні Expo SDK) [36] :
    - При тестуванні роботи додатку у онлайн-середовищі Snack Expo, виникають проблеми відсутності багатьох потрібних для роботи модулів, що ускладнює тестування таким способом;
    - Поки недоступні такі особливості IOS та Android API як Bluetooth, WebRTC, покупки всередині додатків, Apple та Google Pay інтеграція – вони ще в розробці;

- Не всі дії в на задньому фоні застосувань вже доступні (обробка push-нотифікацій на задньому фоні програми);
- Розмір готового кінцевого застосування на IOS та Android займає більше 15 Мб пам’яті мінімум;
- Немає підтримки версій Android менших за 5 версію та IOS менших за 10 версію;
- Поки недоступне будування застосунку створеного у managed workflow та публікація на Google Play Market. Для цього потрібно спочатку перейти у bare workflow деякими командами Expo Cli, і перенастроїти застосунок на роботу без Expo.

2. складність розробки при bare workflow;
3. при використанні нового синтаксису «хуків» - складна для швидкого опанування тема, особливо для реалізації навігації.

Перейдемо до Xamarin. Варто згадати про розширення Xamarin.Forms, яке дає можливість писати застосунки з мінімальною кількістю залежного від платформи коду.

Можна виділити такі переваги та недоліки Xamarin.Forms:

Серед переваг:

1. кросплатформеність;
2. активна підтримка творцями та широка аудиторія розробників. За традицією, документація від Microsoft дуже читабельна та зрозуміла, а на форумах часто можна легко знайти відповіді на питання;
3. використання мови C#, що дає можливість легкого входу в технологію знавцям мови або знавцям інших C-подібних мов;
4. легка реалізація навігації у застосунку;
5. опис UI в одному місці – спільному для платформ коді;
6. використання MVVM – зручного патерну проектування.

Серед недоліків:

1. Для компіляції та запуску застосування для IOS потрібно мати комп’ютер з MacOS;
2. повна кросплатформеність є «умовною»: вона досягається тільки за умови мінімального використання специфічного для платформи коду. Розробнику в будь-якому разі потрібно переглянути код для конкретної платформи, і, деколи, навіть правити його та додавати необхідну реалізацію. Так, наприклад за відсутності потрібних для якоїсь задачі бібліотек, зроблених спільнотою, потрібно самому реалізовувати різний для платформ код;
3. додатковий рівень абстракції PCL (переносна бібліотека класів) зменшує продуктивність застосування, особливо коли потрібно перемальовувати все дерево UI. [37]

Перейдемо до нативної розробки для платформ IOS та Android. Тут важливо замітити, один з недоліків для однієї платформи – IOS. Нативна розробка для цієї платформи можлива тільки за наявності комп’ютера з MacOS. При цьому, за наявності у розробника такого пристрою, цей мінус одразу зникає.

Можна виділити такі плюси та мінуси:

Серед переваг:

1. повний контроль над аспектами роботи пристрою;
2. повна свобода дій для реалізації задумок на конкретній платформі;

Серед недоліків:

1. необхідність реалізовувати додаток одразу для двох платформ через наявність розділення ринку мобільних застосувань на дві аудиторії;
2. складні для опанування (у порівнянні з попередніми методами) тонкощі SDK;
3. важкість підтримки та оновлення коду одразу на двох платформах.

Варто згадати також і про адаптивні сайти як спосіб побудови мобільних застосувань. Звісно, це не повноцінні застосунки, які майже завжди, не мають доступу до ресурсів пристрою, проте, такий варіант використовується як стандарт при розробці веб-сайтів, адже в 2020 році рідко можна зустріти сайт без адаптивного дизайну.

ВИСНОВКИ

Мобільна розробка повинна бути одночасно націленою на широку аудиторію, швидке та ефективне написання коду, легкість підтримування коду, хорошу документацію та широку спільноту розробників. Безперечно, важливим моментом є підтримка фреймворків та бібліотек своїми творцями. Оточення для розробки повинне давати зручні можливості розробникам створювати функціонал, оновлювати його, тестувати, реалізовувати нові ідеї та покращувати старі, тобто давати можливість без лишніх складнощів доводити новий функціонал від стадії початку розробки до стадії релізу.

Під час написання курсової роботи, для реалізації мобільних застосунків на різних фреймворках, мною було розроблено повністю функціонуючий сервер (як REST-сервіс) на базі Spring Boot та з використанням СУБД PostgreSQL. В ньому була також реалізована аутентифікація та авторизація (умовна, бо з однією роллю) користувачів. Важливо підмітити, що на момент написання цієї роботи, сервер був постійно запущений в хмарі за допомогою сервісу Heroku.

Для огляду різних підходів до розробки мобільних застосувань для платформ IOS та Android, мною було створено три аналогічні застосування – на базі фреймворку React Native, на базі фреймворку Xamarin.Forms, та нативний варіант Android-додатку.

Як результат, було досліджено деякі особливості цих методів, розглянуті їх переваги та недоліки. Як переважаючий в зручності тестування під час розробки хочеться виділити фреймворк React Native і зручне програмне забезпечення Expo, яке спрощувало роботу при розробці додатку. Як переважаючий по простоті реалізації, на мою думку, потрібно виділити фреймворк Xamarin.Forms. Обидва підходи чудово виділяються, хоч і зі своїми особливостями, як ті, які призначені для кросплатформеної розробки мобільних

додатків. Проте, у нативної розробки під платформи IOS та Android теж є свої переваги в плані продуктивності та контролю багатьох процесів на пристрої, хоч і зі складнішим способом опанування технологій, ніж у попередніх методів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ain.ua – статистика з Sensor Tower – 1 half of 2019 [Електронний ресурс] https://ain.ua/2019/07/04/skolko-zarabotali-mobilnye-prilozheniya-2019/
2. App Annie – The state of mobile – 2019 [Електронний ресурс] https://www.appannie.com/en/insights/market-data/the-state-of-mobile-2019/
3. Android – Офіційний сайт [Електронний ресурс] https://www.android.com/
4. GlobalStats - Mobile Operating System Market Share Worldwide - April 2020 [Електронний ресурс] https://gs.statcounter.com/os-market-share/mobile/worldwide
5. ThinkMobiles - What are the popular types and categories of apps – Daniella Rollus, 2019 [Електронний ресурс] https://thinkmobiles.com/blog/popular-types-of-apps/
6. A List Apart. – Responsive Web design - Marcotte, Ethan (2010) [Електронний ресурс] https://alistapart.com/article/responsive-web-design/
7. Techopedia – Cross-Platform meaning [Електронний ресурс] https://www.techopedia.com/definition/17056/cross-platform
8. Hypertext Transfer Protocol – HTTP/1.1 - R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee - June 1999 [Електронний ресурс] https://www.w3.org/Protocols/rfc2616/rfc2616.txt
9. Websecurity DigiSert – What is SSL, TLS and HTTPS [Електронний ресурс] https://www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https
10. Dou.ua – “Синхронізуємо розуміння Rest” - Dmytro Krasun [Електронний ресурс]

https://dou.ua/lenta/articles/rest-conception/

11. Architectural Styles and the Design of Network-based Software Architectures - Chapter 5, Representational State Transfer (REST) – Roy Fielding, dissertation [Електронний ресурс]
https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#sec_5_2_1_1

12. Habr – «Введення в RESTful API – RESTful веб-сервіси» [Електронний ресурс]
https://habr.com/ru/post/483202/

13. Habr – “RESTful API – велика брехня” - переклад [Електронний ресурс]
https://habr.com/en/post/265845/

14. Блог Роя Філдінга – REST APIs must be hypertext-driven – 2008 [Електронний ресурс]
https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

15. SavvyApps – How to build resful mobile app [Електронний ресурс]
https://savvyapps.com/blog/how-to-build-restful-api-mobile-app

16. Developer Apple – Documentation – Preventing insecure network connections [Електронний ресурс]
https://developer.apple.com/documentation/security/preventing_insecure_network_connections

17. Habr – «Архітектура мобільного клієнт-серверного застосування» (пер.) [Електронний ресурс]
https://habr.com/ru/post/246877/

18. Studref – Аріхтектура мобільних застосувань [Електронний ресурс]
https://studref.com/463512/informatika/arhitektura_mobilnyh_prilozheniy

19. Tutorialspoint – Spring Boot [Електронний ресурс]
https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm

20. Baeldung – Inversion of Control and Dependency Injection

[Електронний ресурс]
https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring
21. Spring – Spring Security [Електронний ресурс]
https://spring.io/projects/spring-security
22. Metanit – PosgreSQL [Електронний ресурс]
https://metanit.com/sql/postgresql/1.1.php
23. Habr – JWT [Електронний ресурс]
https://habr.com/ru/post/340146/
24. React Native – Documentation [Електронний ресурс]
https://reactnative.dev/
25. React – Documentation [Електронний ресурс]
https://uk.reactjs.org/
26. Expo – Documentation [Електронний ресурс]
https://expo.io/
27. Tutorialspoint – NodeJs [Електронний ресурс]
https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm
28. Reactnavigation [Електронний ресурс]
https://reactnavigation.org/docs/hello-react-navigation/
29. Habr – “Детально про Xamarin” – пер. [Електронний ресурс]
https://habr.com/ru/post/188130/
30. Docs Microsoft [Електронний ресурс]
https://docs.microsoft.com/en-us/xamarin/
31. Metanit – MVVM [Електронний ресурс]
https://metanit.com/sharp/wpf/22.1.php
32. Techopedia – Android SDK
[Електронний ресурс]
https://www.techopedia.com/definition/4220/android-sdk
33. Insights – 10 popular mobile apps that are built using Xamarin

https://insights.daffodilsw.com/blog/10-popular-mobile-apps-that-are-built-using-xamarin

34. Habr – «Чого очікувати, коли хочеш стати мобільним розробником» - пер. [Електронний ресурс]

https://habr.com/ru/company/habr_career/blog/461709/

35. Developer Apple – Documentation [Електронний ресурс]

https://developer.apple.com/

36. Expo – Limitations [Електронний ресурс]

https://docs.expo.io/introduction/why-not-expo/?redirected

37. Habr – «Особливості розробки під Xamarin.Forms» - пер. [Електронний ресурс]

https://habr.com/ru/company/devexpress/blog/263645/