ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики та інформатики

«До захисту допущено»

Завідувач кафедри ПМІ

_____ Ольга ДМИТРІЄВА

(підпис) (ім'я та прізвище)

«____»_____2022 р.

Випускна кваліфікаційна робота бакалавра

на тему Розробка комплексу програм з функціями особистого кабінету начальника ділянки будівництва

зі спецчастиною Розробка модулів з застосуванням мови програмування JAVA та БД на базі MySQL

Виконала: студента 4 курсу, групи КН-18

(шифр групи)

спеціальності 122 Комп'ютерні науки

(шифр і назва спеціальності)

Коровін Д.А. _____

(прізвище та ініціали) (підпис)

Керівник _____

(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Рецензент _____ _____

(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Нормоконтроль:

_____ к.т.н., доц. Ірина НАЗАРОВА

(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

(підпис)

Луцьк – 2022 р.

ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики та інформатики
Освітній ступінь бакалавр
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПМІ
_____ /Ольга ДМИТРІЄВА/
«____»_____2022 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Коровіну Даніїлу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка комплексу програм з функціями особистого кабінету начальника ділянки будівництва

Розробка модулів з застосуванням мови програмування JAVA та СпецчастинаБД на базі MySQL

керівник роботи Валерій КОСТІН ст. викл.
(ім'я та прізвище, науковий ступінь, вчене звання)

затверджені наказом від « 06 » травня 20 22 року № 180

2. Строк подання студентом роботи 8 червня 2022 року

3. Вихідні дані до роботи результати переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) аналіз предметної області та формулювання задачі; 2) проєктування розроблюваної програми ; 3) опис процесу програмної реалізації розробки програм; 4) опис процесу роботи та результатів тестування програми

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1) загальна постановка задачі; 2) обґрунтування тематики роботи; 3) функціональне представлення системи; 4) засоби реалізації системи; 5) екранні форми розр-обки; 6) висновки

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	09.05.22-15.05.22	
2	Проектування	16.05.22-22.05.22	
3	Реалізація	23.05.22-29.05.22	
4	Контролі якості	30.05.22-03.06.22	
5	Оформлення пояснювальної записки та опис створеної системи. Проходження нормоконтролю	04.06.22-07.06.22	
6	Перевірка пояснювальної записки антиплагіатною системою. Оформлення презентації до роботи, графічного матеріалу та рецензування роботи	08.06.22-21.06.22	
7	Захист роботи	22.06.22	

Студент  **Даніїл КОРОВІН**

(підпис) (ім’я та прізвище)

Керівник роботи  Валерій КОСТІН

(ім’я та прізвище)

АНОТАЦІЯ

Даніїл КОРОВІН. Розробка комплексу програм з функціями особистого кабінету начальника ділянки/ Випускна кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 122 Комп'ютерні науки. – ДВНЗ ДонНТУ, Луцьк, 2022.

Випускну бакалаврську роботу присвячено розробці програми на мові JAVA та написання коду за допомогою фреймворку React Native, який допомогає розроблювати програми з інтерфейсом для платформ iOS та Android, використовуючи одну й теж кодову базу. Також React Native має кросплатформу, що допомогає прискорити розробку програм для різних платформ.

Метою роботи є розробка програм, які б облегшували пошук робітників та дозволяли подальшу розробку документів для підприємства.

Ідея розробки донної системи програм появилася під час тяжкого пошуку всіх робітників та їх місця роботи, для заповнення документу де описується вся робота.

Об'єктом роботи є мова JAVA та фреймворк React Native. React Native грає головну роль в цієї роботі, так як всі дії проходять через нього та його бібліотеки. Серед всіх фреймворків вибір впав на React Native тому, що він є популярним, зручним та легким фреймворком для розробки програм на мові JAVA. Він легко підключається до редактору коду Visual Studio Code, який в свою чергу є безкоштовним редактором, швидким, легким, має широкі можливості для кастомізації, та включає в себе відладчик для комфортної роботи з кодом.

Предметом роботи є програма, яка сама по собі легка в розробці та дуже проста в доопрацюванні. Має зрозумілий інтерфейс та просту маршрутизацію по ньому.

Практичне значення роботи полягає у тому щоб застосувати всі здобуті навички за період навчання.

Ключові слова: ANDROID, JAVA, EXPO, REACT NATIVE, ANDROID STUDIO, МОБІЛЬНИЙ ДОДАТОК, ЕМУЛЯТОР, БАЗА ДАНИХ, VISUAL STUDIO CODE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

дод. – додаток

ІМС – інформаційна мобільна система

МД – мобільний додаток

МЕ – мобільний емулятор

МП – мобільний пристрій

МС – мобільна система

НС – надзвичайна ситуація

п. – пункт

пр. – підрозділ

AS – Android Studio

VSC – Visual Studio Code

ПО – Програмне забезпечення

Ребілд – відновлення

ВСТУП

На сьогодні, роль мобільних пристроїв в житті людей займає одно з перших місць. У 2018 році практично у кожного першого жителя сучасного міста є особистий мобільний телефон, а доля смартфонів при цьому також займає велику частину. Це обгрунтовано тим, що мобільні пристрої, окрім виконання свого прямого обов'язку здійснювати дзвінки має масу інших функцій. Вони уміють робити фотографії, знімати відео, відтворювати медіа контент, виходити у всесвітню мережу, а також за допомогою всіляких датчиків визначати місця розташування власника, вимірювати пульс і так далі. У одному пристрої люди поєднують навігатора, фотоапарат, серфінг інтернету і соціальних мереж.

Також, мобільні пристрої все частіше використовуються на підприємствах. Це дозволяє керівникам оптимізувати бізнес-процеси компанії. Так, у фірмах таксі, смартфони вже давно замінили класичні радіо-рації. У цій роботі буде розглянутий процес оптимізації роботи будівельної компанії для бистрого обміну інформації:

Актуальність даного мобільного додатка підтверджується рішенням описаних проблем. Мобільний додаток виступатиме єдиною точкою входу і виходу інформації. Буде зручніше організовувати процес, простіше вносити коригування, в свою чергу учасникам буде легше знаходити необхідну інформацію і також відстежувати внесені зміни в захід.

1 ТЕОРЕТИКО-АНАЛІТИЧНА ЧАСТИНА

1.1 Обгрунтування постановки завдання

Мобільний додаток - це програмне забезпечення, спеціально розроблене під конкретну мобільну платформу (iOS, Android, Windows Phone

і т. Д.). Призначено для використання на смартфонах, фаблетах, планшетах, розумних годинах та інших мобільних пристроях.

Програмне забезпечення (далі «ПЗ») - це взаємодія кожної з частин системи логічного ланцюжка нулів і одиниць, що працюють за певному алгоритму обробки і роботи з інформацією, які так само можуть бути програмами.

Тобто, якщо говорити простими словами, то це програма, розроблена на якійсь мові програмування, в якій зашитий алгоритм її роботи. Мобільна платформа - це операційна система для телефонів по аналогії з комп'ютерами.

Розглянемо переваги мобільного додатку по відношенню до сайту:

1. Продуктивність - додаток досить одного разу встановити на телефон для роботи з ним. Після робота з додатком буде в два дії: відкрити телефон, відкрити додаток. Для використання сайту необхідно виконувати три дії: відкрити телефон, відкрити браузер, відкрити сайт. Додатковий фактор вибору мобільного додатку, браузери на сьогоднішній день дуже ресурсомісткі.
2. Доступність донесення інформації - за допомогою мобільного додатки досить легко розміщувати і читати інформацію. Додатково можливо реалізувати push-повідомлення, які скоротять час донесення інформації до об'єкта. Виходячи з вище написаного, вибір був зроблений в бік мобільного додатки.

1.2 Вибір засобів і методу розробки

Після прийняття остаточного рішення про початок розробки мобільного додатка з'являється питання, стосовно його архітектури, яку саме архітектуру реалізації програмного забезпечення вибрати, а також вибору мов програмування, необхідних фреймворків, програмних забезпечень і т.д.

Архітектура - це сукупність роботи компонентів в програмному забезпеченні в єдиному цілому. У розробці мобільного додатка була обрана мікросервісна архітектура.

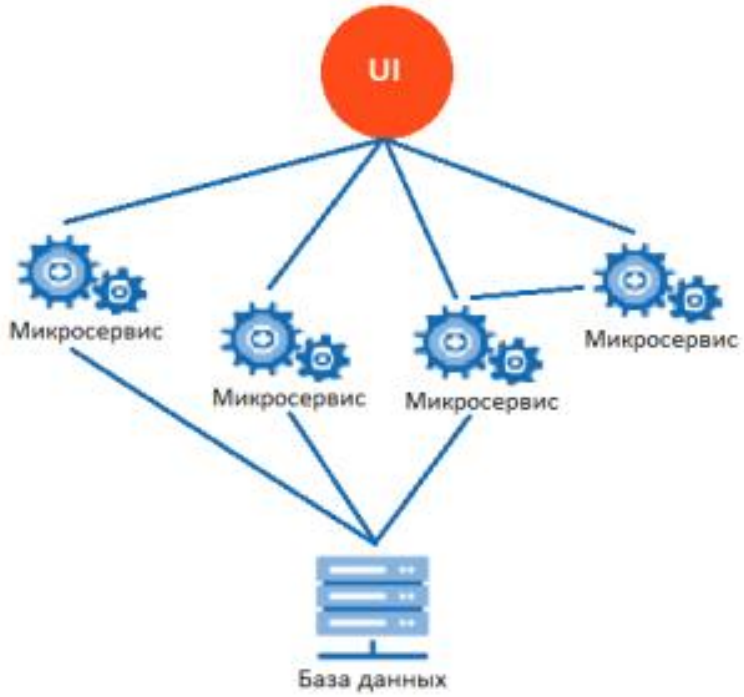


Рисунок 1.1 - Мікросервісна архітектура

Мікросервісна архітектура реалізується на основі невеликих, модульних сервісів, кожен з яких виконує свою унікальну задачу. Дані модулі взаємодіють (спілкуються) за допомогою REST API, з використанням формату JSON.

У свою чергу REST API (Representational State Transfer) - це загальні принципи взаємодії програми з сервером. JSON - це формат для зберігання і обміну інформацією, в основному використовується для передачі даних між сервером і користувачем.

Основні плюси вибору мікросервісної архітектури ПЗ:

1. Спрощує внесення доробок / виправлень, тобто легше масштабувати додаток, так як сервіси не залежні.

2. Зручний при розробці декількома розробниками, знову ж изза незалежності сервісів.
3. При необхідності можливо однією дією відключити сервіс, що дозволяє знизити навантаження на сервер.
4. Безпека. Можна сміливо гарантувати, що передаватися буде тільки та інформація, яка запитується, так як спілкування відбувається по API.

Після того, як була обрана архітектура ПЗ, вже зрозуміло, що буде наш мобільний додаток, буде ділитися на 2 частини: інтерфейсна частина (Front-end), база даних.

В першу чергу, було прийнято рішення про вибір СУБД для подальшої роботи з базою даних.

На поточний момент популярними СУБД є:

1. Oracle.
2. MySQL.
3. Microsoft SQL Server.
4. Mongo DB.

Для того щоб вибрати конкретну СУБД нам необхідні критерії, на підставі чого і буде зроблений відповідний вибір:

1. База даних повинна бути реляційної, де дані організовані в вигляді різних таблиць, а таблиця складається з стовпців (полів) і рядків (записів).
2. СУБД надається на безкоштовній основі.
3. Від компанії, які супроводжують дані СУБД, повинна бути доступна коректна і актуальна документації.

Для зручності і наочності відповідних варіантів, виходячи з вищеописаних критеріїв, представимо відомості про СУБД у вигляді таблиці.

Таблиця 1.1 - Порівняння СУБД

Наименование СУБД	1 критерий	2 критерий	3 критерий
Oracle	+	-	+
MySQL	+	+	+
Microsoft SQL Server	+	-	+
Mongo DB	-	+	+

Виходячи з наведеної таблиці, робимо вибір на користь MySQL, так як ця СУБД відповідає всім заданим критеріям. MySQL - це система управління базами даних, поширювана як вільне програмне забезпечення. Додатково можна вказати, що дана СУБД показала надійність в роботі, можливість забезпечення безпеки даних, а також надає зручний інтерфейс для користувача.

Після вибору СУБД наступним кроком є реалізація інтерфейсу програмного забезпечення, а саме необхідно визначиться з вибором мови програмування або Фреймворка і середовища програмування інтерфейсу. Для того, щоб не писати інтерфейс двічі під різні платформи мобільних пристроїв, будемо розглядати виключно кроссплатформенние інструменти програмування.

Кросплатформенность - це властивість сторінки, що дозволяє працювати відразу на декількох видах операційних систем або платформ.

Популярні інструменти:

1. Apache Cordova- мобільне середовище розробки додатків, має друга народна назва PhoneGap. Дозволяє програмістам створювати додатки для мобільних пристроїв за допомогою CSS3, HTML5 і JavaScript. Це забезпечується за рахунок перетворення з CSS, HTML і JavaScript в код, який будь-яка платформа сприймає як елемент web. Випущений в

2012 році. Основним мінусом є ресурсомісткість, внаслідок чого додаток подтормажівает на слабких мобільних пристроях.

2. Flutter-середовище розробки з відкритим вихідним кодом для розробки мобільних додатків випущений від компанії Google. Данне середовище було випущено відносно недавно в 2018, раніше мала назву Sky, і спеціалізована тільки під Android. На жаль, багато мінуси, проблеми були успадковані від попередніх версій. На офіційному сайті Google, для даного Фреймворка вказано три основних плюса:
    a) Швидка розробка (присуще тільки для досвідченого розробника);
    b) Виразний і гнучкий інтерфейс (Фреймворк сам вміє налаштовувати шрифти, розташування і фізику скроллов і т.д.);
    c) Продуктивність (власний графічний движок).
3. React Native - Фреймворк для розробки мобільних додатків, реалізований на основі React (це інструмент для створення користувальницьких веб-с.торінок). Випущений був в 2015 році. React Native популярний так як компактний і відрізняється високою продуктивністю. Для написання коду використовується JavaScript. Особливу популярність додало використання великими компаніями, такими як: Facebook, Instagram, Skype, Tesla і т.д., щоб підтверджує його перспективу на майбутні роки.

Аналізуючи вищенаписане, робимо висновок, що Flutter не підходить, так як ще знаходиться на стадії розробки і вдосконалення. також можна виключити Apache Cordova в зв'язку з низькою продуктивністю, відповідно вибір падає на React Native. На додаток до React Native можна сказати, що у нього відсутні звична html розмітка, присутні свої, вбудовані компоненти. Також присутня всім звична формальна мова опису зовнішнього вигляду компонентів (CSS). В свою чергу CSS дасть нам можливість налаштовувати компоненти повністю на нашу бажанням.

На наступному етапі слід встановити вибір середовища розробки. На даний момент існує більше 50 програмних систем для написання коду, причому частина з них є спеціалізованими, тобто призначені для певної мови, а частина є універсальними. З огляду на це розглянемо найбільш підходящі для цього – React Native та Java.

Популярні редактори для React-Native:

1. Nuclide - спеціалізоване середовище розробки з відкритим кодом, представлений від компанії Facebook. Безкоштовний. Досить простий і зручний інтерфейс, представляє першокласну середу розробки для проекту React Native, має вбудовану підтримку даного Фреймворка. Більш не супроводжується, знаходиться у фінальній версії. Недоліком є відсутність повноцінної підтримка ОС Windows.
2. Sublime Text - пропріетарний текстовий редактор. Документально є спеціалізований для мови програмування Python, але по фактом підтримує програмування на будь-якому. дозволяє програмувати безкоштовно, проте постійно просити придбати ліцензію. Більш не супроводжується.
3. Visual Studio Code - універсальний редактор вихідного коду, представлений від компанії Microsoft. На поточний день супроводжується. В редакторі присутній функціонал по установці компонентів, який дозволяють розробляти на будь-якій мові програмуванні. Безкоштовний. Легкий і дуже приємний, а також інтуїтивний інтерфейс.

1.3Життєвий цикл програмного забезпечення

Розробка будь-якого програмного забезпечення обов'язково проходить п'ять стадій формування готового продукту : аналіз вимог, проектування, програмування, тестування і експлуатація [3]. Сукупність цих етапів називають послідовністю фаз життєвого циклу програмного

забезпечення(ПО). Під життєвим циклом ПЗ мають на увазі ряд подій, що відбуваються з системою в процесі еѐ створення і подальшого використання. Так само життєвий цикл характеризується часом від початкового моменту ухвалення рішення про необхідність створення ПЗ, до моменту його введення в експлуатацію [3].

Кожен етап життєвого циклу визначається однозначними завданнями, методами їх рішення, початковими даними, отриманими на попередньому етапі, і результатами. Результатами аналізу, зокрема, являються функціональні і інформаційні моделі, і діаграми, що відповідають їм. Життєвий цикл ПЗ носить ітераційний характер: результати чергового етапу часто викликають зміни в проектних рішеннях, вироблених на більше ранніх етапах [13].

Структуру, що визначає послідовність виконання і взаємозв'язку процесів, дій і завдань упродовж життєвого циклу називають моделлю життєвого циклу ПЗ. Модель залежить від масштабу, складності проекту і специфіки умов, в яких система створюється і функціонує. Будь-яка технологія базується на певних уявленнях про життєвий цикл, вибудовує свої методи і інструменти навколо фаз і етапів життєвого циклу, саме тому необхідно спиратися на різні моделі при розробці ПЗ.

Залежно від потреб проекту вибирається відповідний підхід до циклу розробки. В процесі створення програмного забезпечення використовується п'ять основних видів життєвих циклів :

Каскадна, V- образна, інкрементна, ітераційна і спіральна моделі. Каскадна модель. Типовий цикл розробки програмного забезпечення називається каскадним, у тому числі "водоспад"(waterfall), і характеризується тим, що етапи строго послідовні і перехід між ними неповоротний.

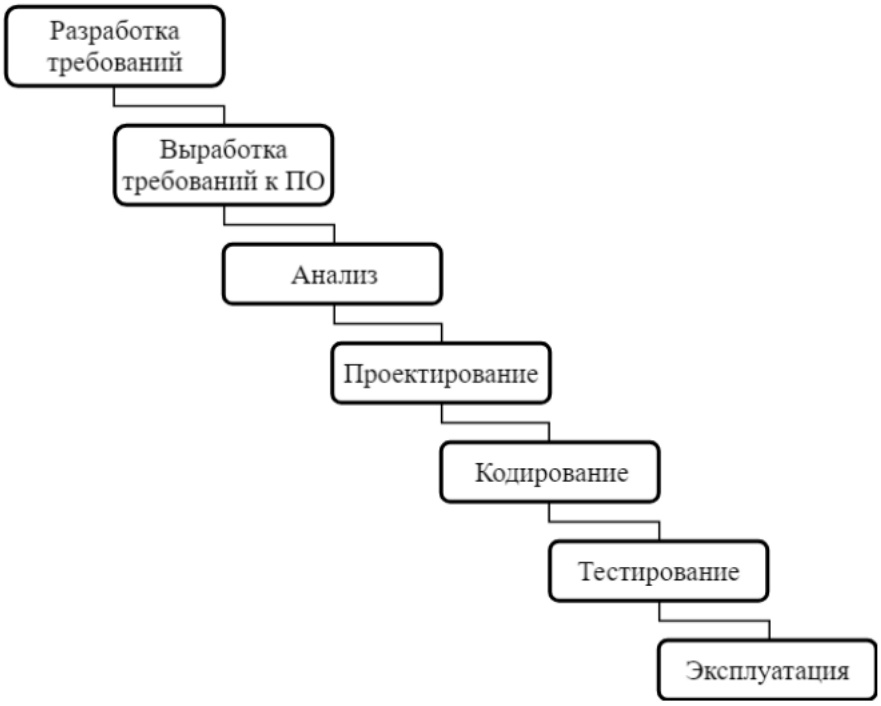


Рисунок 1.2 Каскадна модель

Ця модель рідко застосовується на практиці, оскільки вимоги замовника можуть уточнюватися, змінюватися і доповнюватися, таким чином, проект або система не задовольнятиме потрібним вимогам. Каскадна модель з проміжним контролем. Враховуючи такий серйозний недолік каскадної моделі, в неѐ були додані зворотні зв'язки з кожним етапом життєвого циклу (рис. 1.3), що у свою чергу в десятки разів збільшило витрати на розробку, але і дозволило підвищити надійність системи в цілому. Допрацьована модель дістала назву 10 каскадна модель з проміжним контролем або вир. Найбільш прийнятна така модель в невеликих проектах, де вимоги заздалегідь відомі, зрозумілі і чітко зафіксовані.

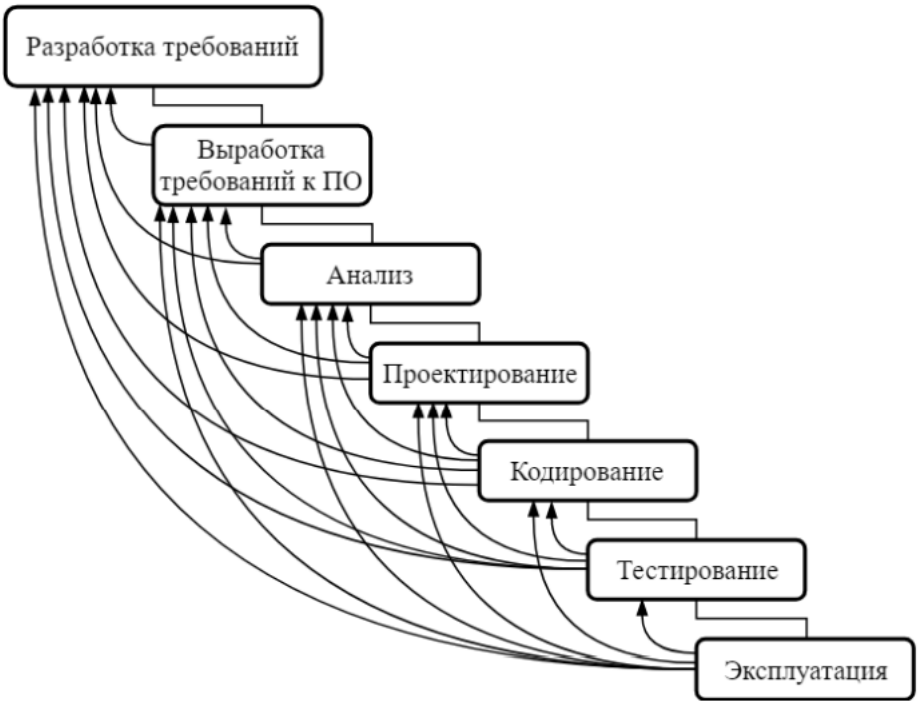


Рисунок 1.3 Каскадна модель з проміжним контролем

V- образна модель. Успадкувала структуру "крок за кроком" від каскадної моделі. V- образна модель застосована до систем, яким особливо важливе безперебійне функціонування. Наприклад, застосовні програми в клініках для спостереження за пацієнтами, інтегроване ПЗ для механізмів управління аварійними подушками безпеки в транспортних засобах і так далі [6].

Особливістю моделі можна вважати те, що вона спрямована на ретельну перевірку і тестування продукту, що знаходиться вже на первинних стадіях проектування. Стадія тестування проводиться одночасно з відповідною стадією розробки, наприклад, під час кодування пишуться модульні тести. Така модель використовується в малих і середніх проектах, де вимоги чітко визначені і фіксовані.

Рисунок 1.4 V-образна модель

Ітеративна і інкрементна моделі. У інкрементній моделі(рис. 1.5) повні вимоги до системи діляться на різні складки. Термінологія часто використовується для опису поетапного складання ПЗ. Мають місце декілька циклів розробки, і разом вони складають життєвий цикл "мульти-водопад".

Цикл розділений на дрібніші легко створювані модулі. Кожен модуль проходить через фази визначення вимог, проектування, кодування, впровадження і тестування. Процедура розробки по інкрементній моделі припускає випуск на першому великому етапі продукту у базовій функціональності, а потім вже послідовне додавання нових функцій, так званих "інкрементів". Процес триває до тих пір, поки не буде створена повна система. Інкрементні моделі використовуються там, де окремі запити на зміну ясні, можуть бути легко формалізовані і реалізовані.

Ітераційна модель життєвого циклу не вимагає для початку повної специфікації вимог. Замість цього, створення розпочинається з реалізації частини функціонала, базою, що стає, для визначення подальших вимог. Цей процес повторюється. Версія може бути неідеальна, 12 головне, щоб вона

працювала. Розуміючи кінцеву мету, прагнуть до неї так, щоб кожен крок був результативний, а кожна версія - працездатна.

Рисунок 1.5 Інкрементна модель

Прикладом ітераційної розробки може служити розпізнавання голосу. Перші дослідження і підготовка наукового апарату почалися давно, на початку - в думках, потім - на папері. З кожною новою ітерацією якість розпізнавання покращувалася. Проте, ідеальне розпізнавання ще не досягнуте, отже, завдання ще не вирішене повністю [13].

Спіральна модель схожа на інкрементну, але з акцентом на аналіз ризиків. Вона добре працює при рішенні критично важливих бізнес-завдань, коли невдача несумісна з діяльністю компанії, в умовах випуску нових продуктових лінійок, при необхідності наукових досліджень і практичної апробації.

Спіральна модель припускає чотири етапи для кожного витка: планування, аналіз ризиків, конструювання, оцінка результату і при задовільній якості перехід до нового витка. Ця модель не підходить для малих проектів, вона резонна для складних і дорогих, наприклад, таких, як розробка системи документообігу для банку, коли кожен наступний крок вимагає більшого аналізу для оцінки наслідків, ніж програмування. На практиці моделі ЖЦ рідко використовуються в чистому вигляді. У рамках

однієї або декількох моделей створюються різні методології, які прискорюють процес розробки і впровадження проекту.

1.4 React Native

Перш ніж починати розробку програм на мові React Native, спочатку треба розібратися що він з себе представляє, та який спектор він охоплює в самих розробках ПО.

React - це JS-бібліотека для створення користувальницьких інтерфейсів, зазвичай для веб-додатків. Вона розроблена в Facebook і поширюється під ліцензією open source з 2013 року. React широко поширена, і на відміну від більших MVC-фреймворків вирішує відносно вузьке завдання: рендеринг інтерфейсу.

Популярність React має ряд обставин. Вона компактна і має високу продуктивність, особливо при роботі з швидкозмінними даними. Завдяки компонентної структурі, React заохочує до написання модульного, перевиконаємого коду.

React Native - це та ж React, але для мобільних платформ. У неї є ряд відмінностей: замість тега div використовується компонент View, а замість тега img - Image. Процес розробки залишився тим же. Вам може стати в нагоді знання Objective-C або Java. Крім того, в мобільній розробці є свої підступ (протестував я на різних пристроях? Чи достатньо великі об'єкти, щоб на них комфортно натискати?). Проте, якщо ви працювали з React, то React Native здасться вам практично такий же, настільки ж комфортною.

React Native - він дуже нативний. Перше що хочеться про нього сказати так це те, що він дуже легкий в освоюванні та в написанню коду. Інші рішення JavaScript-для-мобільних-платформ мають складну архітектуру та тяжкі в освоєнні. Для React Native створена також веб-сторінка, в якій описані всі модулі, бібліотеки які можливо підключити, опис роботи них в екранній формі та можливістю там же змінювати все та експериментувати.

У React компонент описує власне відображення, а потім бібліотека обробляє для вас рендеринг. Ці дві функції розділені ясним рівнем абстракції. Якщо потрібно відмалювати компоненти для інтернету, то React використовує стандартні HTML-теги. Завдяки тому ж рівню абстракції - «мосту» - для рендеринга в iOS і Android React Native викликає відповідні API. У iOS компоненти вимальовуються в справжні UI-види, а в Android - в нативні види (8).

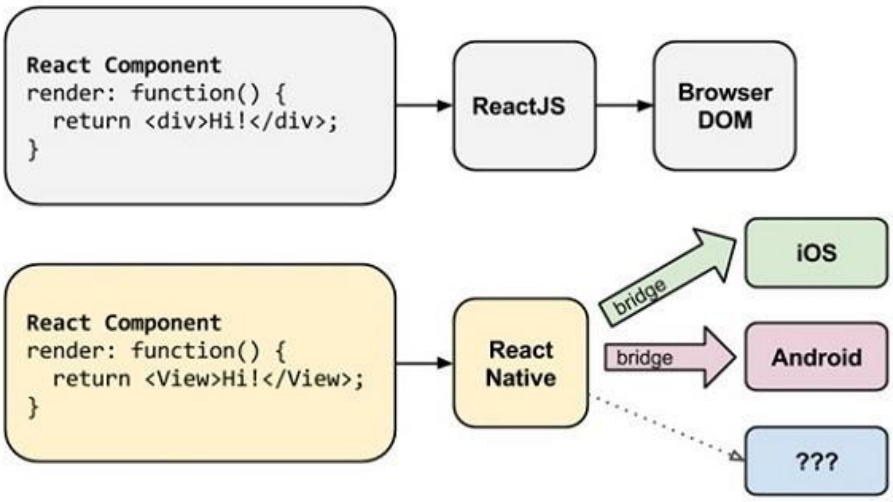


Рисунок 1.6 - Відносини між React та React Native

На перший погляд написаний код жахливо виглядає, дуже схожий на стандартний JavaScript, CSS і HTML. Замість компілювання в нативний код, React Native бере додаток і запускає його за допомогою JS-двигуна хост-платформи, без блокування основного UI-потоку. В цьому є перевага нативної продуктивності, анімації та поведінки без необхідності писати на Objective-C або Java. Інші методи розробки кросплатформенних додатків, на зразок Cordova або Titanium, не мають такого рівня нативної продуктивності або відображення.

Переваги для розробника. У порівнянні зі стандартною розробкою під iOS і Android, React Native має набагато більше переваг. Оскільки ваше додаток здебільшого складається з JavaScript, ви можете користуватися численними достоїнствами веб-розробки. Наприклад, щоб побачити внесені в

код зміни, можна миттєво «оновити» додаток замість тривалого очікування завершення традиційного ребілда.

Переваги роботи з JavaScript, також доступні і в розробці мобільних застосувань, використовуючи React Native:

- JavaScript дуже популярна мова: це єдина мова для розробки клієнтських додатків і у нього величезне співтовариство розробників. Тому існує безліч бібліотек і готових модулів, написаних іншими розробниками, які знаходяться у відкритому доступі і які можна використати у своєму проекті;
- JavaScript має низький поріг входження: цю мову можна вивчити в короткі терміни і відразу ж почати писати код для вирішення реальних завдань;
- Як правило, при роботі з цією мовою не використовуються масивні фреймворки з великою кодовою базою, тому можна плавно підвищувати складність проектів, що реалізовуються, шляхом вивчення окремих бібліотек при їх інтеграції;
- Для створення мобільних застосувань розробникові не потрібні будуть знання і досвід в їх розробці. Досить мати досвід в розробці клієнтський додатків з JavaScript. Відповідно, набагато легше буде знайти людей для розробки/підтримки проекту;
- Одна кодова база для різних типів проектів. Можна використати одні і ті ж функції в проектах серверних, клієнтський і мобільних застосувань. Якщо при роботі над клієнтським застосуванням, добре себе зарекомендувала одна з підключених бібліотек, то її можна буде використати і в розробці мобільного застосування.

Крім того, React Native надає «розумну» систему повідомлень про помилки і стандартні інструменти налагодження JavaScript, що сильно полегшує процес мобільного розробки.

1.5 Expo

Це фреймворк для швидко розвиваються нативних React додатків. Це як Laravel або Symphony для PHP-розробників, або Ruby on Rails для Ruby розробників. Expo надає шар поверх команд React Native API, щоб ними було зручніше користуватися і управляти. Він також надає інструменти, які спрощують початкове створення і тестування React Native додатків. І нарешті він надає компоненти призначеного для користувача інтерфейсу і сервіси, які зазвичай доступні тільки при установці сторонніх нативних компонентів React. Всі вони доступні через Expo SDK.

Обмеження Expo:

1. Expo додатки не підтримують фонове виконання коду. Це означає, що ви не можете, наприклад, запустити код, який прослуховує зміни місця розташування, коли додаток закрито.
2. Expo додатки обмежені нативними API, які підтримує Expo SDK. Це означає, що якщо ваш додаток має вельми специфічні завдання, як наприклад обмін даними з Bluetooth пристроєм, єдиний варіант для реалізації такої функціональності - це використання простого React Native, або написання власного коду за допомогою бібліотеки ExpoKit.
3. Expo прив'язує вас до використання його інструментів. Це означає, що ви не можете просто встановити і використовувати більшу частину відмінних коштів, доступних для розробки React Native, таких як інструменти командного рядка, допоміжні бібліотеки і фреймворки призначеного для користувача інтерфейсу. Але хороша новина в тому, що Expo SDK сумісний з простими додатками React Native, тому ви не будете мати ніяких проблем, якщо витягнете додаток з Expo.
4. Автономні виконувані файли додатків Expo можуть бути побудовані тільки при наявності підключення до мережі. Expo надає інструменти командного рядка під назвою Exp. Це дозволяє

розробникам запускати процес складання проекту на серверах Expo. Після завершення збірки буде доступна URL-посилання для скачування файлу .apk або .ipa

1.6 Висновки за розділом

У даному розділі були представлені основні положення, та опис язиків та фреймворків які використовуються далі в роботі.

2 РОЗРОБКА БАЗИ ДАНИХ

1.1 Кваліфікації баз даних

Потоки інформації, які циркулюють в світі і оточують нас, колосальні. Протягом часу вони мають тенденцію до збільшення, отже, в будь-якій організації виникає питання про ефективне управлінні інформацією та даними, які забезпечили б найбільш 18 ефективну роботу в даній сфері. Найбільш популярними є комп'ютеризовані способи обробки інформації, тобто бази даних, які дозволяють ефективно зберігати, структурувати і систематизувати великий обсяг інформації і даних. У сучасному світі без використання баз даних неможливо уявити роботу більшості фінансових, промислових, торгових та інших організацій, так як, якщо не буде баз даних, то вони просто захлинуться в інформаційному потоці, що негативно позначиться на їх діяльності та конкурентоспроможності.

Різноманітність характеристик і видів баз даних породжує різноманіття їх класифікації. Класифікація баз даних може бути проведена за різними ознаками, що відносяться до різних компонентів і сторонам їх функціонування, серед яких можна виділити наступні: характер інформації, що зберігається, спосіб зберігання даних, структура організації даних, спосіб доступу до даних, сфера застосування та інші.

За характером інформації, що зберігається виділяються фактографічні, документальні та лексикографічні бази даних. Фактографічні бази даних - це

бази даних, які містять короткі констатують відомості про описуваному об'єкті реальної предметної області, представлені в строго певній формі. Отже, одиницею зберігання в фактографічні базах даних є факт, тобто певний елемент змістовної інформації. документальні бази даних - бази даних, які об'єднують документи, згруповані за різними ознаками і властивостями. Документальні бази даних можуть містити значну кількість інформації різного типу: текстову і графічну. Сучасні інформаційні технології стирають кордон між фактографічними і документальними базами даних, так як існують засоби, які дозволяють легко підключати будь-який документ до фактографічної бази даних. Лексикографічні бази даних – це 19 бази даних, що представляють собою різні машинозчитувані словникові масиви даних, об'єктом опису в якому є лексична одиниця. До лексикографічним баз даних можна віднести тезауруси, рубрикатори, термінологічні словники і класифікатори.

За способом зберігання даних бази даних поділяються на централізовані і розподільні. Централізовані бази даних - це бази даних, що розробляються і функціонують на принципах централізації (20). Такі бази даних знаходяться на одній електронній обчислювальній машині, у вигляді одного інформаційного масиву. В такому випадку говорять про централізований або монопольному володінні даними. Централізована база даних доступна тільки одному користувачеві, так як виключається одночасна робота декількох користувачів. Управління базою даних, тобто її коригування та інші процедури, підтримують її цілісність і безпеку, здійснюється централізовано в одному місці одним користувачем. основний недолік централізованих баз даних полягає в необхідності передачі великого потоку даних, а також низький ступінь надійності і продуктивності. Перевага централізованих база даних - мінімальні витрати часу на коригування даних, що зберігаються в базі.

Для зниження актуальності перерахованих недоліків створюються розподілені бази даних, тобто бази даних частини яких розташовуються в різних частинах мережі. Розподілені бази даних є сукупністю баз даних, які фізично розподілені по взаємопов'язаним ресурсів локальної мережі і доступні для спільного застосування і використання в різних місцях різними користувачами. Головний критерій розподілу даних в мережі полягає в тому, що дані повинні розташовуватися там, де існує найбільша частота звернень користувачів до них. Розподілена база даних роз'єднана тільки фізично, а не логічно, тобто вся база даних потенційно доступна з будь-якого автоматизованого робочого місця користувача.

За характером організації даних бази даних можуть бути розділені на неструктуровані, частково структуровані, структуровані. Даний класифікаційний ознака відноситься до інформації, яка представлена в символьному вигляді і міститься в базі даних. До неструктурованим баз даними можна віднести бази даних, які організовані у вигляді семантичних мереж. частково структуровані можна вважати бази даних, представлені у вигляді звичайного тексту або гіпертекстові системи. Структуровані бази даних вимагають завчасного проектування, писання і розробки структури бази даних, тільки після даних процедур в бази даних даного типу можуть бути занесені дані, необхідні для роботи з базою даних. Структуровані бази даних, в свою чергу, по типу використовуваної моделі даних можуть ділиться на: ієрархічні, реляційні, мережеві, змішані і мультимедійні.

Ієрархічні бази даних - це бази даних, які графічно можуть бути представлені як дерево, що складається з об'єктів різних рівнів (21). На верхньому рівні знаходиться тільки один об'єкт, на другому рівні - об'єкти другого рівні і так далі. Між об'єктами, які розташовані на різних рівнях, існують горизонтальні зв'язки, кожен об'єкт може включати декілька об'єктів, які розташовані нижче рівня цього об'єкту. Такі об'єкти знаходяться відносно предка(об'єкт, що знаходиться ближче до кореня) до нащадка(об'єкт, що

знаходиться нижче рівнем), при цьому можлива ситуація, коли об'єкт-предок не має нащадків або може мати декілька, тоді як у об'єкту нащадка обов'язково має бути тільки один предок. Ієрархічна база даних має кореневу теку, що поступово розгалужується донизу. Звертаючи увагу на те, що подібна структура бази даних аналогічна файловій системі, такі бази даних ефективно використовуються для виконання різних операцій на даними, що зберігаються в пам'яті комп'ютера. Ієрархічна модель ідеально застосовується для структурованої і впорядкованої інформації. Основним недоліком використання ієрархічних баз даних є їх громіздкість і складність проектування логічних зв'язків між об'єктами.

Мережевий підхід до організації даних є розширенням ієрархічного підходу, оскільки в ієрархічних структурах запис-нащадок повинна мати в точності одного предка, а в мережевій структурі даних нащадок може мати будь-яке число предків, а між об'єктами нижнього рівня можуть існувати горизонтальні зв'язки. Отже, мережева база даних - це база даних, що утворюється узагальненням ієрархічної бази даних за рахунок дозволу мати об'єктам більше за одного предка, тобто кожен елемент вищестоящого рівня може бути пов'язаний одночасно з будь-якими елементами наступного рівня. На зв'язку між об'єктами в мережевих базах даних не накладаються ніякі обмеження. Прикладом мережевої бази даних може служити всесвітня павутина глобальної комп'ютерної мережі інтернет, гіперпосилання зв'язує між собою величезна кількість документів в єдину розподілену мережеву базу даних доступ до якої можливий з будь-якого робочого місця користувача, що має доступ в глобальну павутину.

Наступному видом структурованих баз даних являється реляційні бази даних. Модель реляційних баз даних була розроблена Едгаром Франком Коддом і обнародувана ним в 1970 году (22). Реляційна модель є логічно структурованою таблицею з полями, які описують дані і їх стосунки між собою, а також операції, зроблені над ними, а головне - правила, що

гарантують їх цілісність. Модель називається реляційною, тому що в основі її лежать стосунки між даними.

Реляційні бази даних - це бази даних, які є сукупністю взаємозв'язаних таблиць, кожна з яких містить інформацію про об'єкти певного типа27. Таблиці складаються з рядків, які називаються записами, і стовпців, які називаються полями, або атрибутами. У рядках таблиці міститися дані про один об'єкт, а в стовпцях таблиці містяться різні характеристики цих об'єктів. Записи, тобто рядки таблиці, мають однакову структуру. Вони складаються з полів, які зберігають атрибути об'єкту. Кожне поле, тобто стовпець таблиці, описує тільки єдину характеристику об'єкту і має строго певний тип даних. Усі записи мають одні і ті ж поля, тільки в них відображаються різні інформаційні властивості об'єкту.

Таблиці в реляційних базах даних мають ряд властивостей : в таблиці не може бути двох однакових рядків, в таблиці може не бути жодного рядка, але обов'язково має бути хоч би один стовпець, стовпці таблиці не залежать один від одного, усі значення в одному стовпці мають один тип даних, будь-яка таблиця повинна мати первинний ключ. Первинний ключ - це поле або комбінацію полів таблиці, таблиці, що ідентифікують кожен рядок, єдиним чином. Ключ може складатися з одного поля або декількох полів таблиці, тоді він називається складеним. Первинний ключ має бути унікальним і однозначно визначати запис в таблиці. Значення первинного ключа дозволяє реалізовувати процес пошуку і впорядковування інформації у базі даних. Таблиці в реляційній базі даних повинні відповідати вимогам нормалізації стосунків, тобто відповідати апарату обмежень на формування таблиць, який дозволяє виключити дублювання і суперечність даних, що зберігаються у базі даних.

Таблиці в реляційних базах даних можуть бути пов'язані один з одним, а це означає, що дані можуть витягатися з декількох таблиць одночасно. Таблиці зв'язуються між собою для того, щоб зменшити об'єм бази даних і

виключити дублювання інформації. Зв'язок між таблицями забезпечується наявністю в них однакових стовпців. Оскільки реляційні бази даних найбільш популярні, то у них є певні достоїнства. До основних достоїнств реляційних баз даних можна віднести наступні: модель даних, що зберігаються, представляє інформацію в найбільш простій і зрозумілій для користувача формі, в основі бази даних лежить добре розвинений математичний апарат, який дозволяє доступно описати основні операції, вироблювані над даними, при маніпулюванні і доступі до даних використовуються мови не процедурного типу, маніпулювання даними на рівні вихідної інформації і можливість динамічної зміни даних.

Усупереч перерахованим достоїнствам, у сучасному світі, при розширенні меж моделювання інформаційних систем, були виявлені і істотні обмеження при використанні реляційних баз даних як основного сховища інформації. До основних недоліків використання реляційних баз даних можна віднести: трудомісткість їх проектування і розробки, повільний доступ до даних, проблематичність моделювання і реалізації складних зв'язків між даними, результатом запиту до бази даних є інформація, що зберігається в самій базі даних, при цьому часто вимагається, щоб в результаті запиту був отриманий логічний висновок на основі даних, що зберігалися.

Розвиток інформаційних технологій в області проектування і побудови баз даних призводить до потреби зберігання і обробки нових Розвиток інформаційних технологій в області проектування і побудови баз даних призводить до потреби зберігання і обробки нових видів представлення інформації у базах даних, мультимедіа, що відносяться до класу, які характеризуються високим рівнем інформационно структурної складності. Мультимедійні бази даних - це бази даних, які містять мультимедійну інформацію (25). До основних особливостей мультимедійних баз даних, що відрізняють їх від інших баз даних, можна віднести їх здатність зберігати і

обробляти не лише числа, символи і масиви інформації, але і такі дані, як документи, зображення, відео і звукозаписи, а також цифрові карти.

По сфері застосування розрізняють універсальні і проблемноорієнтированих баз даних. Універсальні бази даних - це бази даних, які призначені для вирішення універсальних завдань користувача (23). До основних завдань, що вирішуються за допомогою універсальних баз даних, можна віднести забезпечення зберігання у базі даних необхідної інформації і можливості отримання даних по різних запитах, скорочення надмірності і дублювання даних, що зберігаються у базі даних, забезпечення цілісності самої бази даних. Проблемноорієнтировані бази даних - це бази даних, які містять тематично пов'язані документи і дані, призначені для вирішення прикладних завдань певного виду конкретної предметною області (24). При проектування проблемно-орієнтованих баз даних необхідно орієнтуватися на потреби користувача, який працюватиме з цією базою даних.

Розробка бази даних буде тривати на легкій платформі ніж MySQL, на модулі JSON Server, це Node Module, який використовується для створення швидкого веб сервісу меш ніж за одну хвилину.

JSON добре рішення для онлайн профілів, імітує повноцінну базу даних на базі JSON-файлів. До неї можна отримати доступ простим запитом до БД через fetch. JSON можна використовувати в програмах JavaScript без необхідності синтаксичного аналізу або серіалізації.

Це текстовий спосіб представлення об'єктних літералів JavaScript, масивів і скалярних даних. JSON ідеально підходить для зберігання тимчасових даних. Наприклад, тимчасові дані можуть бути даними, створеними користувачем, такими як відправлена форма на веб-сайті. JSON також можна використовувати в якості формату даних для будь-якої мови програмування, щоб забезпечити високий рівень взаємодії.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```


Рисунок 2.1 - Приклад JSON

3 РОЗРОБКА ІНТЕРФЕЙСУ

На даному етапі приступаємо до розробки інтерфейсної частини мобільного додатка.

Першим кроком, необхідно встановити програмні забезпечення, то встановимо Visual Studio Code. Дистрибутив програмного забезпечення Visual Studio Code, можна взяти з офіційного сайту https://code.visualstudio.com. (26)

Після установки Visual Studio Code, приступаємо до створення проекту. Для цього необхідно встановити набір інструментів Expo, виконавши через командний рядок наступне «npm install --global expo-cli».

Далі, продовжуючи роботу в командному рядку, виконаємо команду по створення порожнього проекту «expo init dp».

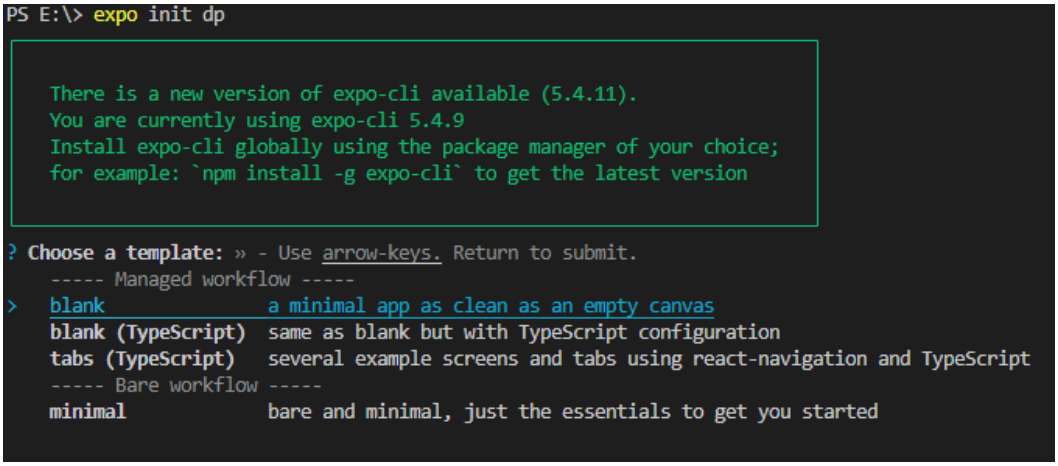


Рисунок 3.1 - Створення порожнього шаблону проекту

Відповідно після виконання даної команди, у нас створиться проект мобільного додатка, з порожнім шаблоном файлів, який вже Зараз можна запустити і протестувати на телефоні.

Наступним кроком, необхідно створений проект відкрити за допомогою Visual Studio Code, натискання поєднання клавіш «Ctrrl + O», після вибравши папку, раніше створеного проекту. Далі створимо папку «screens» в нашому проекті, яка буде зберігати в собі всі екрани програми. У ній створимо файл «Home.jsx» призначений, для відображення інформації профілю користувача.

```
import React from "react";
import { View, Button, Text, StyleSheet } from "react-native";

const Home = ({ navigation }) => {
  return (
    <View style={styles.main}>
      <Text style={styles.text}>This is the home screen</Text>
      <Button
        title="Go to Notes Screen"
        onPress={() => navigation.navigate("Notes")}
      />
    </View>
  );
};

const styles = StyleSheet.create({
  main: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    textAlign: "center",
    backgroundColor: "#16191b",
  },
  text:{
    color:'#fff',
  }
});

export default Home;

```

Рисунок 3.2 - Головний екран, який зустрічає при вході

На наступному етапі потрібно в головному файлі створити весь маршрут нашої програми, так як всі деталі маршруту будуть проходи через нього та інші маршрутні файли. Підключаємо до нашої програми всі необхідні модулі, так як без них вона не буде працювати. Для мого варіанту розробки треба підключити модулі навігації, використовувати будемо офіційну бібліотеку React Navigation (документація - https://reactnavigation.org/docs). (*npm install –save react-navigation react-navigation-stack react-native-reanimated react-native-gesture-handler react-native-screens react-native-vector-icons).*(27)

Для установки нижньої навігаційної панелі необхідно виконати всередині нашого проекту команду

npm install @ reactnavigation / bottom-tabs

в командному рядку (командний рядок викликається в Visual Studio Code поєднанням клавіш «Ctrl + `»).Цих модулів досить для подальшої праці над програмою.

```jsx
import React from "react";
import { NavigationContainer } from "@react-navigation/native";
import DrawerNavigator from "./navigation/DrawerNavigator";

const App = () => {
  return (
    <NavigationContainer>
      <DrawerNavigator />
    </NavigationContainer>
  );
};

export default App;
```

Рисунок 3.3 - Головна програма

Так як у нашій програми буде не одне вікно а декілька, для більш зручної праці над самою програмою винесемо в окрему папку проекту та назвемо її «navigation».

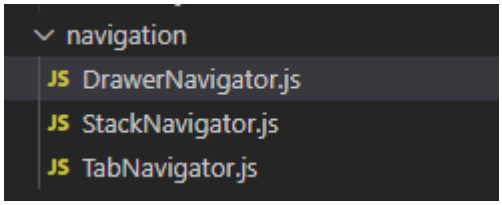


Рисунок 3.4 - Папка «navigation»

В самій папці у нас не один файл а три, зроблено це для послідовного навігаційного переходу між нашими екранами. Починається весь маршрут у файлі «DrawerNawigation.js». В цьому файлі не виконується чогось неймовірного, він будує маршрут з самого головного файлу, до всіх інших двох файлів.

```
import React from "react";
import { createDrawerNavigator } from "@react-navigation/drawer";
import TabNavigator from "./TabNavigator";
import LoginScreen from "../screens/LoginScreen";

const Drawer = createDrawerNavigator();

const DrawerNavigator = () => {
  return (
    <Drawer.Navigator>
      <Drawer.Screen name="Home" component={TabNavigator} />
      <Drawer.Screen name="AllWorkers" component={TabNavigator} /
      <Drawer.Screen name="LoginScreen" component={LoginScreen} /
    </Drawer.Navigator>
  );
};

export default DrawerNavigator;
```

Рисунок 3.5 - Файл «DrawerNawigation.js»

Далі подорож триває через файл «TabNavigation.js» в цьому файлі створюється нижня панель навігацій.

```
1  import React from 'react';
2  import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
3  import { MainStackNavigator, ProfileStackNavigator, AllWorkersnavigator} from './StackNavigator'
4
5
6  const Tab = createBottomTabNavigator();
7
8  const BottomTabNavigator = () => {
9    return (
10
11     <Tab.Navigator>
12       <Tab.Screen name="Home" component={MainStackNavigator} />
13       <Tab.Screen name="AllWorkers" component={AllWorkersnavigator} />
14       <Tab.Screen name="Profile" component={ProfileStackNavigator} />
15     </Tab.Navigator>
16   );
17 }
18
19 export default BottomTabNavigator
```

Рисунок 3.6 - Файл «TabNavigation.js»

На наступному кроці нас зустрічає вже важливіший за ці файли файл, в даному файлі, буде лежати настройка по роботі і стилістиці нижньої навігаційної панелі програми.

Згідно з малюнком 3.7, в основній частині додатка у нас буде три екрани, для яких ми раніше створили порожні шаблони, тому в навігаційній панелі у нас також буде три кнопки.

```
import React from "react";
import { createStackNavigator } from "@react-navigation/stack";
import Home from "../screens/Home";
import Notes from "../screens/Notes";
import AllWorkers from "../screens/AllWorkers"
import Profile from "../screens/Profile";

const Stack = createStackNavigator();

const screenOptionStyle = {
  headerStyle: {
    backgroundColor: "#9AC4F8",
  },
  headerTintColor: "white",
  headerBackTitle: "Back",
};

const MainStackNavigator = () => {
  return (
    <Stack.Navigator screenOptions={screenOptionStyle}>
      <Stack.Screen name="Home" component={Home} />
      <Stack.Screen name="Notes" component={Notes} />
    </Stack.Navigator>
  );
};

const ProfileStackNavigator = () => {
  return (
    <Stack.Navigator screenOptions={screenOptionStyle}>
      <Stack.Screen name="Profile" component={Profile} />
    </Stack.Navigator>
  );
};

const AllWorkersnavigator = () => {
  return (
    <Stack.Navigator screenOptions={screenOptionStyle}>
      <Stack.Screen name="AllWorkers" component={AllWorkers} />
    </Stack.Navigator>
  );
};

export { MainStackNavigator, ProfileStackNavigator, AllWorkersnavigator };
```

Рисунок 3.7 - Файл «StackNavigator.js»

Також була створена ще одна кнопка «Notes», вона відповідає за створення заміток. За цю кнопку відповідає 2 файли, які в свою чергу виконують послідовну роботу для виконання роботи.

```
import React from "react";
import { View, StyleSheet, Text } from "react-native";
import { useState } from "react";
import { FlatList } from "react-native-web";
import ListItem from "../components/ListItem";
import Form from "../components/Form";


export default function Notes(){
const [listOfItems, setListOfItems] = useState([
  {text: 'Приехать на работу', key: '1'},
  {text: 'Сесть за стул', key: '2'},
  {text: 'Выпить чашку кофе', key: '3'},
  {text: 'Открыть ворлд', key: '4'},
  {text: 'Отправить на переработку', key: '5'},
  {text: 'Выпить чашку кофе', key: '6'},
  {text: 'Уехать с работы', key: '7'},
])

const addHandler = (text) =>{
  setListOfItems((list) => {
    return[
      { text: text, key: Math.random().toString(36).substring(7) },
      ...list
    ]
  });
}

const deleteHandler = (key) =>{
  setListOfItems((list) => {
    return list.filter(listOfItems => listOfItems.key != key)
  });
}

  return (
    <View style={styles.main}>
      <Text style={styles.text}>Notes List</Text>
      <Form addHandler={addHandler}/>
      <FlatList data={listOfItems} renderItem={({item}) =>(
        <ListItem el={item} deleteHandler={deleteHandler}/>

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

const styles = StyleSheet.create({
  main: {
    flex: 1,
    alignItems: "center",
  },
  text:{
    fontSize:30,
    paddingTop: 10,
  },
});
```

Рисунок 3.8 - Файл «Notes»

В самому файлу створюється список задач які потрібно виконати, на початку роботи програми, всі замітки остаються, але їх можна видалити як в самий програмі, або вже через код якщо початкові замітки не потрібні. Також в самому файлі створюється 2 функції:

1. Перша це створює випадкове число, та присвоює його ключу самої задачі. Зроблено це було для того щоб у своєї задачі був свій ключ, та щоб випадково не видалити все або декілька задач.
2. Друга функція видаляє та добавляє самі задачі, з'єднана ця функція з іншими файлами які відповідають за файл «Notes».

Наступний файл «Form.js», він відповідає за саму кнопку добавлення заміток, та за поле вводу самого тексту замітки.

```
import React, { useState } from "react";
import { TextInput, Text, StyleSheet, Button } from "react-native";
import { View } from "react-native-web";

export default function Form ({ addHandler }){
  const [text, setValue] = useState('');

    const onChange = (text) =>{
      setValue(text);
    };
  return (
    <View>
        <TextInput style={styles.tinput} onChangeText={onChange} placeholder='Ввести заметку' />
        <Button color='green' title="Добавить заметку" onPress={() => addHandler(text) } />
    </View>
  );
};

const styles = StyleSheet.create({
 tinput:{
  alignItems: 'center',
  borderBottomWidth: 1,
  padding: 12,
  marginVertical: 30,
  width: '100%',
  color: '#16191b'
 },
});
```

Рисунок 3.9 - Файл «Form.js»

Далі йде файл «Listitem.js», він в свою чергу відповідає за видалення замітки за ключем, який був створений раніше в файлу «Notes».

```
import React from "react";
import { TouchableOpacity, Text, StyleSheet } from "react-native";

export default function ListItem ({el, deleteHandler }){
  return (
    <TouchableOpacity onPress={() => deleteHandler(el.key) } >
        <Text style={styles.text}>{el.text}</Text>
    </TouchableOpacity>
  );
}

const styles = StyleSheet.create({
  text: {
    padding: 12,
    textAlign: "center",
    borderRadius: 6,
    borderColor: '#red',
    borderWidth: 1,
    marginTop: 15,
    width: '100%',



  }
});
```

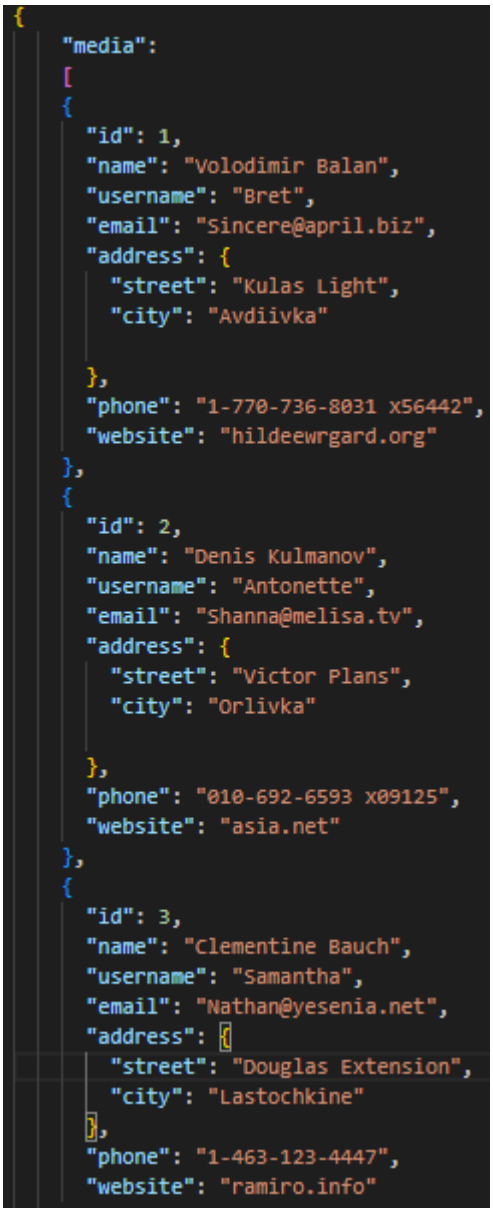
Рисунок 3.10 - Файл «Listitem.js»

Notes це окремий файл, який не залежить від якихось складних файлів або операцій. Далі розглянемо файл «AllWorkers.jsx», даний файл відповідає за вивід бази даних всіх працівників, всі працівники виводяться списком, який можна задати програмно, тобто якщо ви хочете щоб виводилось тільки місце проживання працівника та номер телефону, то це можливо легко задати в самій програмі.

```
import React, {Component} from 'react';
import { StyleSheet, Text, View,FlatList} from 'react-native';
export default class App extends Component {

  state ={
    data:[]
  }

  fetchData= async()=>{
    const response = await fetch('http://localhost:3001/media');
    const users = await response.json();
    this.setState({data: users});

  }
componentDidMount(){
  this.fetchData();
}
  render() {
    return (
      <View style={styles.container}>
       <FlatList
       data={this.state.data}
       keyExtractor={(item,index) => index.toString()}
       renderItem={({item}) =>

       <View style={styles.cub}>
          <Text style={styles.text}>{item.name}</Text>
          <Text style={styles.text}>{item.email}</Text>
          <Text style={styles.text}>City: {item.address.city}</Text>
         </View>

       }

       />
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#16191b',
  },
  cub: {
    backgroundColor:'#303234',
    padding:10,
    margin:10,
  },
  text: {
    color:'#fff',
    fontWeight:'bold'
    },
});
```

Рисунок 3.11 - Файл «AllWorkers.jsx»

Цей файл прив’язаний до локального хосту, та вибірково виводить буд то ім’я робітника, емайл, або адресу, що воно може виводить залежить від самої бази даних, чим більше потрібних даних в базі даних, тим більше вона може їх виводити. Сама база даних розташовується на локальному або віртуальному сервері, в моєму випадку створено файл, який містить в собі JSON модель бузи даних, та знаходиться в окремій папці сервер.



```
{
  "media":
  [
  {
    "id": 1,
    "name": "Volodimir Balan",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "city": "Avdiivka"

    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildeewrgard.org"
  },
  {
    "id": 2,
    "name": "Denis Kulmanov",
    "username": "Antonette",
    "email": "Shanna@melisa.tv",
    "address": {
      "street": "Victor Plans",
      "city": "Orlivka"

    },
    "phone": "010-692-6593 x09125",
    "website": "asia.net"
  },
  {
    "id": 3,
    "name": "Clementine Bauch",
    "username": "Samantha",
    "email": "Nathan@yesenia.net",
    "address": {
      "street": "Douglas Extension",
      "city": "Lastochkine"
    },
    "phone": "1-463-123-4447",
    "website": "ramiro.info"
```


Рисунок 3.12 - Файл за базою даних

Також є головна сторін від якої йде перехід в «Notes», в ній є лише інформація що це головна сторінка, та кнопка переходу до «Notes».

```jsx
import React from "react";
import { View, Button, Text, StyleSheet } from "react-native";

const Home = ({ navigation }) => {
  return (
    <View style={styles.main}>
      <Text>This is the home screen</Text>
      <Button
        title="Go to Workers Screen"
        onPress={() => navigation.navigate("Notes")}
      />
    </View>
  );
};

const styles = StyleSheet.create({
  main: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    textAlign: "center",
  },
});

export default Home;
```

Рисунок 3.13 - Сторінка «Home»

4 ТЕСТУВАННЯ

Тестування запиту до бази даних на локальному серверу. Спочатку запускається в консолі сам хостинг, далі йде авто налаштування. При встановлені модулю, він запускається на трьохсотому порту. Для зручності змінено його в файлу package.json на 3001, щоб подальше використання не складало труднощів.

```json
{
  "main": "node_modules/expo/AppEntry.js",
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web",
    "eject": "expo eject",
    "server": "json-server -w server/db.json -p 3001"
  },
  "dependencies": {
    "@react-native-community/masked-view": "0.1.6",
    "@react-navigation/bottom-tabs": "^5.5.1",
    "@react-navigation/drawer": "^5.8.1",
    "@react-navigation/native": "^5.5.0",
    "@react-navigation/stack": "^5.4.1",
    "@types/json-server": "^0.14.4",
    "expo": "~37.0.3",
    "firebase": "^9.8.3",
    "react": "~16.9.0",
    "react-dom": "~16.9.0",
    "react-native": "https://github.com/expo/react-native/archive/sdk-37.0.1.tar.gz",
    "react-native-gesture-handler": "~1.6.0",
    "react-native-reanimated": "~1.7.0",
    "react-native-safe-area-context": "0.7.3",
    "react-native-screens": "~2.2.0",
    "react-native-web": "~0.11.7"
  },
  "devDependencies": {
    "@babel/core": "^7.8.6",
    "babel-preset-expo": "~8.1.0"
  },
  "private": true
}
```

Рисунок 4.1 - Зміна порту

Далі для запуску локальної бази даних, создамо нову консоль в visual studio(якщо в вас запущена ваша програма), та введемо в поле даний запит npm run server, який в свою чергу був прописаний в файлу package.json.

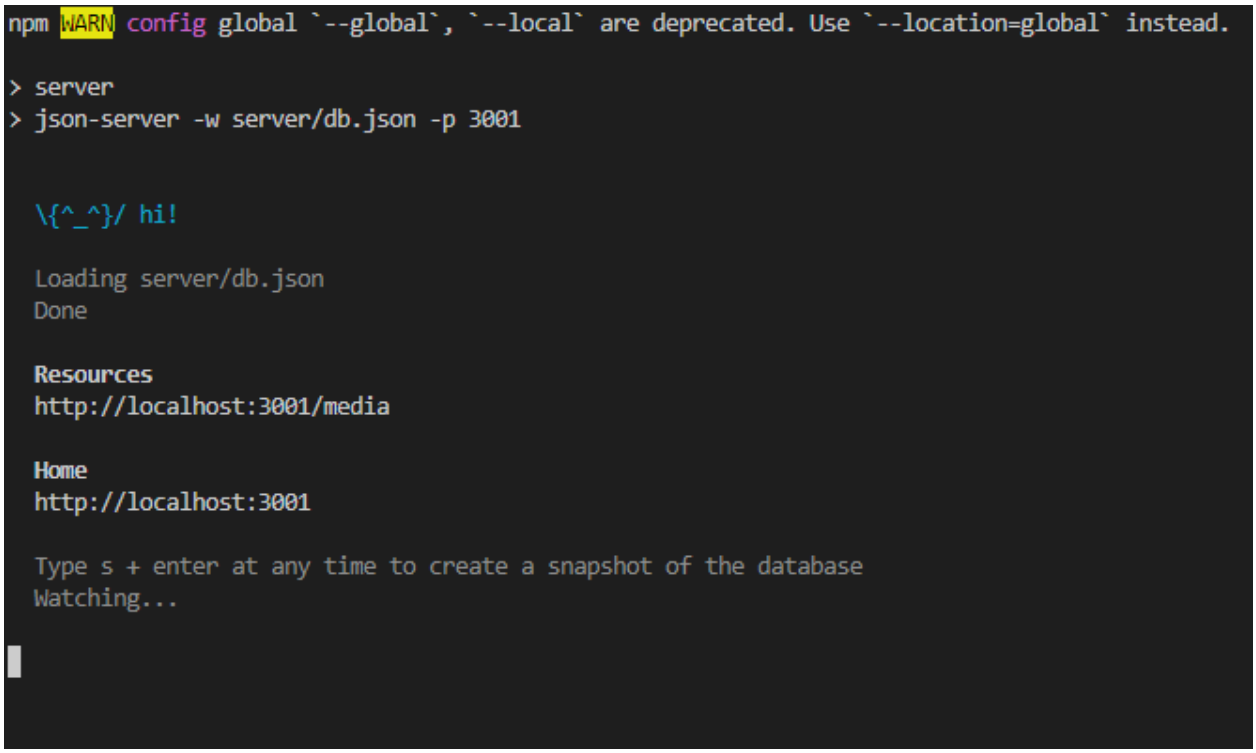


Рисунок 4.2 - Консоль запуску

Рисунок 4.3 - Локальний хост

Висновки за розділом: Зроблено аналіз і обгрунтований вибір технологій реалізації та програмних платформ. Нами вибрано: Фреймворк React Native, JSON, openServer, Ide Visual Studio Code, операційна система Windows 10.

5 ТЕХНІЧНА ДОКУМЕНТАЦІЯ

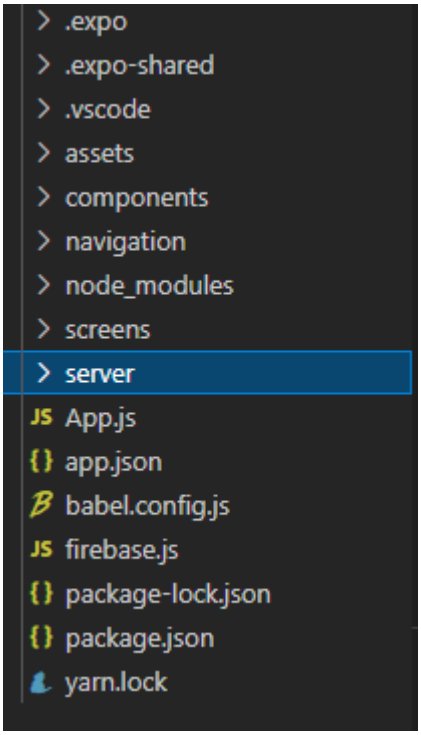


Рисунок 5.1 - Структура інтерфейсу мобільного приладу

Опис призначення основних папок проекту:

- Assets - зберігання стилів компонентів.
- Components - опис компонентів, що використовуються на екранах.
- Navigators - зберігання налаштувань навігації екранів програми.
- Screens - екрани програми.
- Server - база даних.

Опис екранів мобільного приладу:

- App.js – Головний файл.
- Form.js – Файл додавання заміток.
- Listitem.js – Файл видалення заміток.
- DarwerNavigator.js – Файл навігації.
- StackNavigator.js - Файл підключення сцен.

- TabNavigator.js – Файл нижньої панелі.
- AllWorkers.jsx – Файл з виводом даних.
- Home.jsx – Файл з головною сторінкою.
- Notes.jsx – Файл з замітками.
- db.json – Файл з базою даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Семенчук В. Мобільний додаток як інструмент бізнесу - «Альпіна Діджитал », 2016. - 270 с.
2. Молчанов А.Ю. Системне програмне забезпечення: Підручник для вузів 2006. - 396 с.
3. Анохін Антон Борисович Android для телефонів і планшетів: відсутню керівництво для всіх! 2012. - 224 с.
4. Мартін Роберт Чистий архітектура. Мистецтво розробки програмного забезпечення. 2018. - 352 с.
5. Машнін Т.С. Web-сервіси Java. 2012. - 560 с.
6. Дашнер С. Вивчаємо Java EE. Сучасне програмування для великих Підприємств. 2018. - 384 с.
7. Вонг Уоллес Основи програмування для «чайників», 3 -е видання. Пер. з англ. - М .: Видавничий дім «Вільямс», 2004. - 384 с.
8. Машнін Т.С. Розробка Android-додатків в деталях. 2020. - 665 с.
9. Томас Марк Тіленс React в дії. 2019. - 368 с.
10. Рейтц К, Шлюссер Т. Автостопом по Python. 2017. - 336 с.
11. Фрімен А. Angular для професіоналів. 2018. - 800 с.
12. Хорсдал К. Мікросервіси на платформі .NET. 2018. - 352 с.
13. Ченців В.В. Язак програмування Java і середовище NetBeans. - 3 -е изд., перераб. і доп. 2011. - 704 с.

14. Mukund Chaudhary, Ankur Kumar PhpStorm Cookbook. Packt Publishing, 2014. 254 с.
15. Прохореня Н.А. HTML, JavaScript, PHP і MySQL. Джентельменскій набір Web - майстри. - 3 -е изд., Перераб. і доп. 2010. - 912 с.
16. Колісніченко Д.Н. PHP і MySQL. Розробка веб-додатків. - 6 -е изд., перераб. і доп. 2017. - 640 с.
17. Дронов В.А. Laravel. Швидка розробка сучасних динамічних Webсайтов на PHP, MySQL, HTML і CSS. 2017. - 768 с.
18. Фіртман М. jQuery Mobile: розробка додатків для смартфонів і планшетів: Пер. з англ. 2013. - 256 с.
19. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Паттерні об'ектноорієнтірованного проектування. 2020. - 448 с.
20. Кузнєцов С.Д. Основи сучасних баз даних. 2011. С.139.
21. Карпова И.П. Бази даних. Навчальний посібник. - С. 31.
22. Ребека М., Райордан Р. Основи реляційних баз даних. 2012. С. 84.
23. Рад Б.Я., Цехановский В.В., Диявольською В.Д. Бази даних. Теорія і практика. 2014. С. 215.
24. Шаймарданов Р.Б. Моделювання і автоматизація проектування структур баз даних. 2008. С. 169.
25. Петров Г.А., Тихов С.В., Яковлєв В.П. Бази даних : навчальний посібник. 2015. С. 24.
26. Офіційний сайт Visual Studio Code - https://code.visualstudio.com/
27. https://reactnavigation.org/docs/5.x/getting-started/

Додаток А Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження

Додаток Б Екранні форми

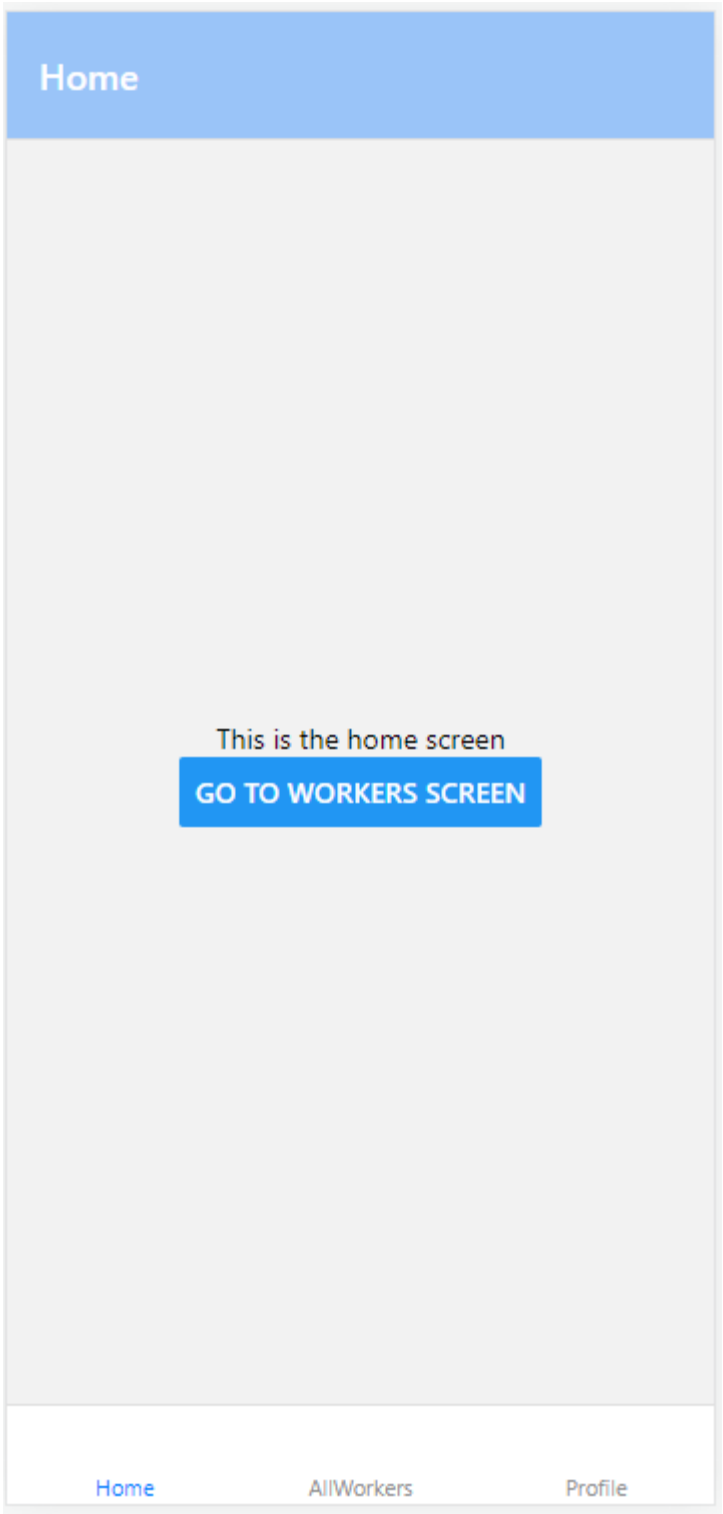


Рисунок. 22. Головна домашня сторінка

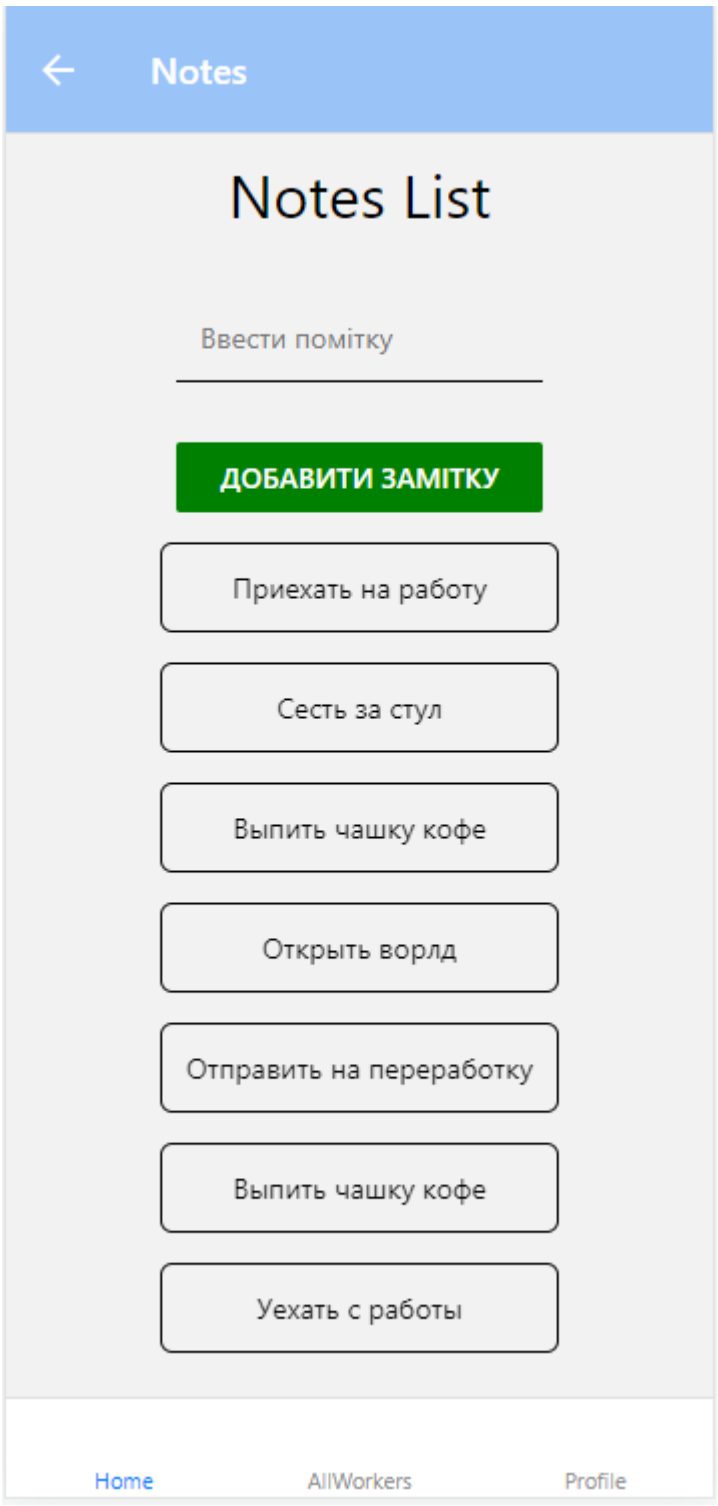


Рисунок .23. Сторінка з замітками

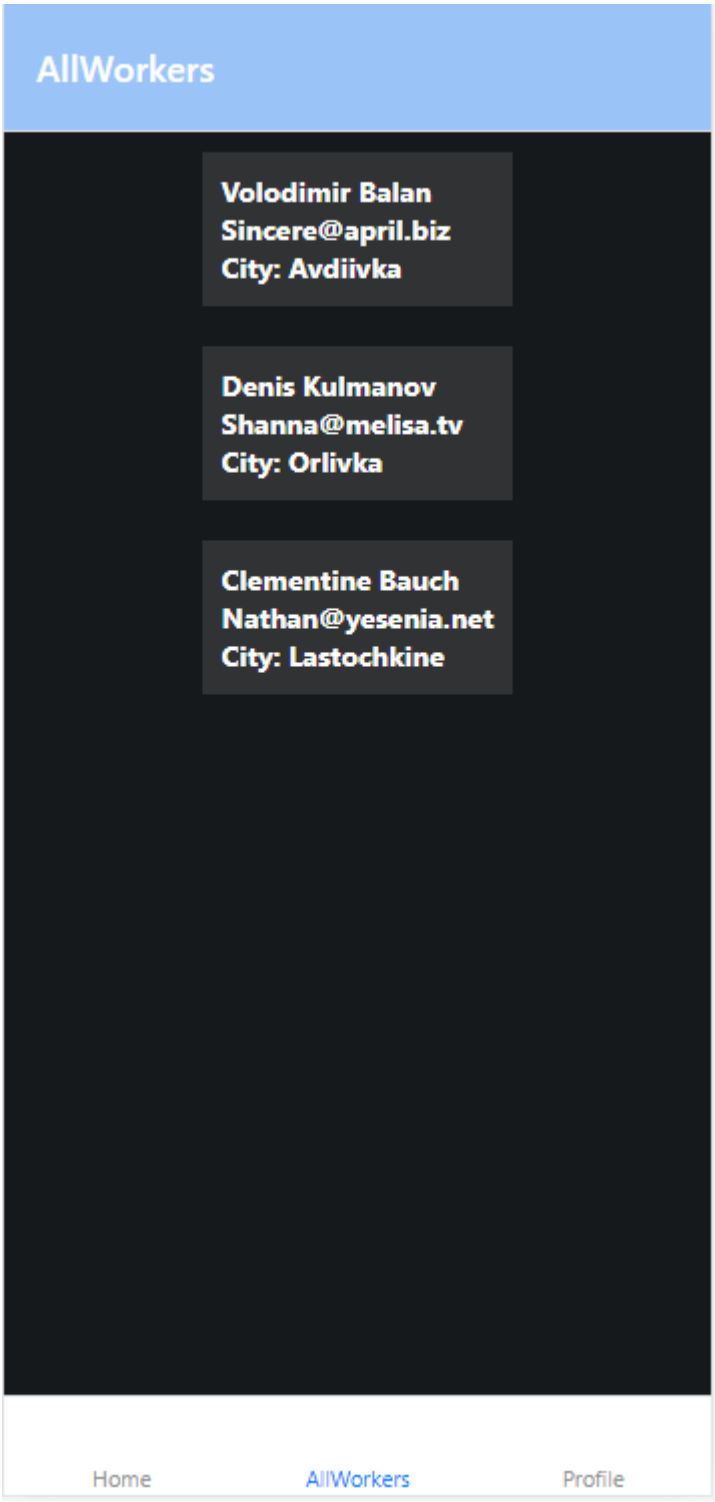


Рисунок. 25. Сторінка з даними всіх робочих

Додаток В Програмний код

Form.js

Даний файл відповідає за працездатність кнопки вводу тексту нашої нової замітки, та создає поле для вводу тексту. Підключається цей файл до Notes. Входом в цей є тільки новий текст який користувач вводить. До нього підключені стандартні бібліотеки, добавлені стилі які роблять строку вводу статичною для всіх розрішень та є функція яка відповідає за текст.

```
import React, { useState } from "react";
import { TextInput, Text, StyleSheet, Button } from "react-native";
import { View } from "react-native-web";


export default function Form ({ addHandler }){
  const [text, setValue] = useState('');


    const onChange = (text) =>{
      setValue(text);
    };
  return (
    <View>
        <TextInput style={styles.tinput} onChangeText={onChange} placeholder='Ввести помітку' />
        <Button color='green' title="Добавити замітку" onPress={() => addHandler(text) } />
    </View>
  );
};


const styles = StyleSheet.create({
 tinput:{
  alignItems: 'center',
  borderBottomWidth: 1,
  padding: 12,
  marginVertical: 30,
  width: '100%',
  color: '#16191b'
 },
});
```

Listitem.js

Цей файл є також прохідним і вважається як своєрідний модуль для файлу Notes, як і Form.js, підключаються стандартні бібліотеки. Цей файл відповідає за видалення непотрібної замітки за головного файлу. Має також стандартну стилістику так як і для всіх файлів.

```
import React from "react";
import { TouchableOpacity, Text, StyleSheet } from "react-native";

export default function ListItem ({el, deleteHandler }){
  return (
    <TouchableOpacity onPress={() => deleteHandler(el.key) } >
        <Text style={styles.text}>{el.text}</Text>
    </TouchableOpacity>
  );
}

const styles = StyleSheet.create({
  text: {
    padding: 12,
    textAlign: "center",
    borderRadius: 6,
    borderColor: '#red',
    borderWidth: 1,
    marginTop: 15,
    width: '100%',



  }
});
```

DrawerNavigator.js

Даний файл є головним в маршрутизації, так як він відповідає за вивід зображення на головний екран, та за перехід між всіма сторінкам. Підключені вже бібліотеки які відповідають за створення самого маршруту між всіма вікнами та підключений файл TabNavigator.js, який в свою чергу відповідає за створення самої нижньої панелі. Також цей файл підключений до головного файлу App.js, який в свою чергу вже виводить все зображення на екран.

```
import React from "react";
import { createDrawerNavigator } from "@react-navigation/drawer";
import TabNavigator from "./TabNavigator";
import Profile from "../screens/Profile";

const Drawer = createDrawerNavigator();

const DrawerNavigator = () => {
```

```
  return (
    <Drawer.Navigator>
      <Drawer.Screen name="Home" component={TabNavigator} />
      <Drawer.Screen name="AllWorkers" component={TabNavigator} />
      <Drawer.Screen name="Profile" component={Profile} />
    </Drawer.Navigator>
  );
};

export default DrawerNavigator;
```

StackNavigator.js

Даний файл відповідає за створення маршрутизації між кнопками нижньої панелі. Підключені всі файли до нього, та бібліотека StackNavigator. Створює особливі теги які в подальшому можливо підключати до окремих файлів.

```
import React from "react";
import { createStackNavigator } from "@react-navigation/stack";
import Home from "../screens/Home";
import Notes from "../screens/Notes";
import AllWorkers from "../screens/AllWorkers"
import Profile from "../screens/Profile";

const Stack = createStackNavigator();

const screenOptionStyle = {
  headerStyle: {
    backgroundColor: "#9AC4F8",
  },
  headerTintColor: "white",
  headerBackTitle: "Back",
};

const MainStackNavigator = () => {
  return (
    <Stack.Navigator screenOptions={screenOptionStyle}>
      <Stack.Screen name="Home" component={Home} />
      <Stack.Screen name="Notes" component={Notes} />
    </Stack.Navigator>
  );
};

const ProfileStackNavigator = () => {
  return (
    <Stack.Navigator screenOptions={screenOptionStyle}>
      <Stack.Screen name="Profile" component={Profile} />
    </Stack.Navigator>
  );
};

const AllWorkersnavigator = () => {
  return (
    <Stack.Navigator screenOptions={screenOptionStyle}>
      <Stack.Screen name="AllWorkers" component={AllWorkers} />
    </Stack.Navigator>
```

```
  );
};

export { MainStackNavigator, ProfileStackNavigator, AllWorkersnavigator };
```

TabNavigator.js

До цього файлу підключені вже не самі файли, а тільки бібліотека яка відповідає за створення нижньої панелі, та всі теги які задавалися в StackNavigator.js. Замість прописування імпорту попередніх файлів які відповідають за маршрут, прописані лише теги для зручності.

```
import React from 'react';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { MainStackNavigator, ProfileStackNavigator, AllWorkersnavigator} from './StackNavigator'


const Tab = createBottomTabNavigator();

const BottomTabNavigator = () => {
  return (

    <Tab.Navigator>
      <Tab.Screen name="Home" component={MainStackNavigator} />
      <Tab.Screen name="AllWorkers" component={AllWorkersnavigator} />
      <Tab.Screen name="Profile" component={ProfileStackNavigator} />
    </Tab.Navigator>
  );
}

export default BottomTabNavigator
```

AllWorkers.jsx

На цій сторінці відбувається вивід з локального хосту нашої бази даних, та вивід їх на наш екран, має стандартну бібліотеку, та більше нікуди не підключається.

```
import React, {Component} from 'react';
import { StyleSheet, Text, View,FlatList} from 'react-native';
export default class App extends Component {

  state ={
    data:[]
  }

  fetchData= async()=>{
    const response = await fetch('http://localhost:3001/media');
    const users = await response.json();
    this.setState({data: users});
```

```
  }
componentDidMount(){
  this.fetchData();
}
  render() {
    return (
      <View style={styles.container}>
       <FlatList
       data={this.state.data}
       keyExtractor={(item,index) => index.toString()}
       renderItem={({item}) =>

       <View style={styles.cub}>
          <Text style={styles.text}>{item.name}</Text>
          <Text style={styles.text}>{item.email}</Text>
          <Text style={styles.text}>City: {item.address.city}</Text>
         </View>

       }

       />
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#16191b',
  },
  cub: {
    backgroundColor:'#303234',
    padding:10,
    margin:10,
  },
  text: {
    color:'#fff',
    fontWeight:'bold'
    },
});
```

Home.jsx

Ця сторінка нас вітає при заході в мобільний додаток, підключається до головного маршрутного файлу. Виводить на сторінку текст з тим що це домашня сторінка, та має кнопку переходу на сторінку за замітками.

```
import React from "react";
import { View, Button, Text, StyleSheet } from "react-native";

const Home = ({ navigation }) => {
  return (
    <View style={styles.main}>
      <Text>This is the home screen</Text>
      <Button
        title="Go to Workers Screen"
        onPress={() => navigation.navigate("Notes")}
```

```
      />
    </View>
  );
};

const styles = StyleSheet.create({
  main: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    textAlign: "center",
  },
});

export default Home;
```

Notes.jsx

Сторінка замітки включає в себе файли ListItem та Form, які в свою чергу передають дані які оброблювались в цих файлах на дану сторінку. Ця сторінка включає в себе список який вже заданий програмно для наглядності. Має генерацію ключів, які присвоюються кожній замітці, щоб подалі не було помилок з видаленням. Також має підключення функцій з інших файлів.

```
import React from "react";
import { View, StyleSheet, Text } from "react-native";
import { useState } from "react";
import { FlatList } from "react-native-web";
import ListItem from "../components/ListItem";
import Form from "../components/Form";


export default function Notes(){
const [listOfItems, setListOfItems] = useState([
  {text: 'Приехать на работу', key: '1'},
  {text: 'Сесть за стул', key: '2'},
  {text: 'Выпить чашку кофе', key: '3'},
  {text: 'Открыть ворлд', key: '4'},
  {text: 'Отправить на переработку', key: '5'},
  {text: 'Выпить чашку кофе', key: '6'},
  {text: 'Уехать с работы', key: '7'},
])

const addHandler = (text) =>{
  setListOfItems((list) => {
    return[
      { text: text, key: Math.random().toString(36).substring(7) },
      ...list
    ]
  });
}

const deleteHandler = (key) =>{
  setListOfItems((list) => {
    return list.filter(listOfItems => listOfItems.key != key)
  });
}
```

```
  return (
    <View style={styles.main}>
      <Text style={styles.text}>Notes List</Text>
      <Form addHandler={addHandler}/>
      <FlatList data={listOfItems} renderItem={({item}) =>(
        <ListItem el={item} deleteHandler={deleteHandler}/>

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

const styles = StyleSheet.create({
  main: {
    flex: 1,
    alignItems: "center",
  },
  text:{
    fontSize:30,
    paddingTop: 10,
  },
});
```

db.json

Це файл база даних, він запускається за допомогою консолі. До нього нічого не підключено, і сам цей файл можливо замінити на будь який, в якому є ім’я, імейл та город, так як програмно ми задали що виводитися буду лише ці строки.

```
{
    "media":
    [
    {
      "id": 1,
      "name": "Volodimir Balan",
      "username": "Bret",
      "email": "Sincere@april.biz",
      "address": {
        "street": "Kulas Light",
        "city": "Avdiivka"

      },
      "phone": "1-770-736-8031 x56442",
      "website": "hildeewrgard.org"
    },
    {
      "id": 2,
      "name": "Denis Kulmanov",
      "username": "Antonette",
      "email": "Shanna@melisa.tv",
      "address": {
        "street": "Victor Plans",
        "city": "Orlivka"

      },
      "phone": "010-692-6593 x09125",
      "website": "asia.net"
```

```
    },
    {
      "id": 3,
      "name": "Clementine Bauch",
      "username": "Samantha",
      "email": "Nathan@yesenia.net",
      "address": {
        "street": "Douglas Extension",
        "city": "Lastochkine"
      },
      "phone": "1-463-123-4447",
      "website": "ramiro.info"
    }

  ]}
```

App.js

```
import React from "react";
import { NavigationContainer } from "@react-navigation/native";

import DrawerNavigator from "./navigation/DrawerNavigator";

const App = () => {
  return (
    <NavigationContainer>
      <DrawerNavigator />
    </NavigationContainer>
  );
};

export default App;
```