МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система для моніторингу стану здоров'я хворих на COVID-19»

Завідувач випускаючої кафедри	**Довбиш А.С.**
Керівник роботи	**Берест О.Б.**
Студента групи ІКм.-91	**Жигамовський Н.А.**
Нормоконтроль	**Проценко О.Б.**

СУМИ 2020

Сумський Державний Університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *Інформаційно-комунікаційні технології*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
СТУДЕНТОВІ

Жигамовському Нікіті Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна система для моніторингу стану здоров'я хворих на COVID-19*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Інформаційний огляд. 2) Вибір методу рішення. 3) Практична реалізація.*
_____
_____
_____

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 57 стор., 22 рис., 0 табл., 4 додатків, 10 джерел.

Об’єкт дослідження — процес моніторингу стану здоров’я пацієнтів.

Мета роботи — розробка інформаційної системи для моніторингу стану здоров’я хворих на COVID-19, яка повинна працювати на операційних системах iOS та Android.

Методи дослідження — теоретичні та функціональні методи моніторингу здоров’я.

Результати — розроблена інформаційна система, з можливістю авторизації та реєстрації як лікаря так і пацієнта, для збору статистичних даних для моніторингу здоров’я, на основі щоденних звітів пацієнта з опитуванням про стан здоров’я. Протестована на операційних системах: iOS, Android. Розроблена система у вигляді мобільного додатку, за допомогою технологій: React Native, Node.js, MongoDB, TypeScript.

СИСТЕМА МОНІТОРИНГУ ЗДОРОВ’Я, МОБІЛЬНИЙ ДОДАТОК, ANDROID, IOS, БАЗА ДАНИХ, MONGODB, КРОСПЛАТФОРМА, REACT, REACT NATIVE, EXPRESS.JS, NODE.JS, JAVASCRIPT, TYPESCRIPT, STYLED COMPONENTS, REACT NAVIGATION, ІНФОРМАЦІЙНА СИСТЕМА

ЗМІСТ

ВСТУП **6**

1 ІНФОРМАЦІЙНИЙ ОГЛЯД **7**

1.1. Види реалізації проектів для мобільних пристроїв 7

1.2. Типи мобільних додатків 7

1.3. Інструменти розробки мобільних додатків 12

1.4. Проблема відстеження пацієнтів хворих на COVID-19 15

2 ВИБІР МЕТОДУ РІШЕННЯ **17**

2.1. Вибір технології реалізації 17

2.2. Мова програмування JavaScript 17

2.3. Фреймворк для розробки мобільних додатків React Native 19

2.4. Платформа для розробки серверної частини NodeJS 19

2.5. Система управління базами данних MongoDB 20

2.6. Прототипи екранів інформаційної системи 22

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ **28**

ВИСНОВКИ **46**

СПИСОК ЛІТЕРАТУРИ **47**

ДОДАТОК А **48**

ДОДАТОК Б **50**

ДОДАТОК В **53**

ДОДАТОК Г **57**

ВСТУП

Інтернет-технології широко поширені в різних сферах сучасного суспільства і, насамперед, особливо в інформаційній.

Сучасні люди вже не можуть жити без використання сучасних технологій. Для багатьох людей смартфон став необхідною частиною життя, коли складно уявити один день без його використання.

Можливість доступу в Інтернет майже з будь-якого місця кардинально змінила спосіб обробки і отримання інформації. Мобільний пристрій – це один із способів спілкування, роботи та розваг. Список функцій, які підтримуються, постійно збільшується, а отже створення мобільного додатку це досить актуально для користувачів. Інколи пристрій може не підтримувати деякі функції, в таком разі ви можете завантажити необхідну програму.

Аналізуючи ситуацію як в Україні так і в усьому світі що до хвороби COVID-19, можна сказати що першочергове завдання для всіх полягає у допомозі відстеження хворих, або пацієнтів з підозрою на цю хворобу, та подальша допомога у лікуванні та одуженні. Інформаційна система повинна бути мостом між лікарями та пацієнтами.

Якщо вивчити роль та значення Інтернет-технологій для сучасного суспільства, то можна зробити висновок, що ця роль є стратегічно важливою і що найближчим часом значення цих технологій швидко зростатиме. Саме ці технології відіграють вирішальну роль у технологічному розвитку суспільства сьогодні.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Види реалізації проектів для мобільних пристроїв

Якщо оглянути основні напрямки реалізації проектів для мобільних пристроїв з технічної точки зору, то можна виділити два: адаптивний сайт і мобільний додаток.

Адаптований веб-сайт - це спеціалізований веб-сайт, який налаштований для відображення та функціонування на мобільному пристрої. Сайт може містити інтерактивні компоненти з використанням JavaScript, HTML5 та нових API браузера [9]. У цьому випадку реалізація називається веб-додатком.

Мобільний додаток - це спеціально розроблений додаток для певної мобільної платформи (iOS, Android, WP). Як правило, додаток розроблено мовою високого рівня та скомпільовано у власний код операційної системи для максимальної продуктивності.

Також існує мобільний додаток, що має компоненти браузера. В таком випадку частина додатка використовується для переходу між сторінками та інтеграції з ОС, веб-компонентами для відображення вмісту. Користувачі часто не мають змоги розрізніти цей варіант від мобільного додатку.

Один з перших аргументів тих, хто вибирає додаток – тісний контакт з операційною системою і звичний інтерфейс користувача. Додаток найбільш інтегрований з платформою дає змогу створити швидкий й гнучкий інтерфейс. Веб-сайт, який використовує мову програмування JavaScript, в свою чергу, пропонує безліч способів взаємодії.

Зараз відгук веб-сайтів менший ніж відгук нативних додатків, але зі зростянням потужності мобільних пристроїв це все може змінитись, як і браузери, які також оновлюються к кожним роком. Серед цього, різні версії

мобільних операційних систем можуть відображати свої стандарти, які потрібно підтримувати и яких потрібно дотримуватись.

З новими тенденціями у технлогіях можна побачити незрозумілість в очах звичайних користувачів. Найактивніші користувачі - це ті, кому дуже подобаються останні інновації в мобільних операційних системах. Варто звернути увагу на фінансування проектів - їх можна використовувати як союзників.

Веб-сайт, особливо інтерактивний веб-сайт, значно поступаються програмам у продуктивності. Мобільні браузери поки не можуть показати високу продуктивність. Також, веб-розробники не завжди використовують найбільш оптимізовані версії бібліотек (погана реалізація цих бібліотек не впливає на "великі" браузери).

Однак програма не може завжди добре працювати - непотрібна анімація та складні інтерфейси значно зменшують відгук. І для складної графіки та анімації для розробки або придбання окремих спеціалізованих бібліотек повинні використовуватися мови нижчого рівня.

Мобільні додатки значно випереджають сайт у цій галузі. Додаток має більше опцій для доступу до пристрою. Існує третій варіант, якщо компонент браузера реалізований у додатку. У цьому випадку ця різниця відразу зникає. Крім того, доступ до пристрою з браузера через розширений набір API постійно зростає. Інтеграція в платформу з кожним днем стає все простішою та швидшою.

Програми на багатьох платформах «пов’язані» з певною службою (AppStore, Windows Store). Якщо такого зв’язку немає, користувачі звикли знаходити програми в магазинах (Google Play). Такі послуги суттєво обмежують функції додатків (особливо у сфері платних послуг). Крім того, затвердження кожної нової версії займає багато часу. Веб-сайт доступний зараз. Просто

відкрийте браузер і введіть адресу. Нова версія веб-сайту доступна одразу на момент публікації.

Можливості надання послуг не обмежені. З одного боку, обмеження та основна публікація в магазинах, з іншого боку, магазин вже має велику кількість користувачів та систему надання послуг, які ще можуть бути платними. Для того, щоб користувачі веб-сайтів могли оплачувати свої покупки через веб-сайт на мобільному пристрої, це залишається дуже трудомістким процесом.

1.2 Типи мобільних додатків

IOS від Apple та Android від Google - це найбільші та найпопулярніші мобільні платформи у світі. Згідно зі статистичними даними, одна частина світового ринку захоплюються Apple та Android - друга (рис. 1.1).

Операційні системи, смартфони, світ (кві 2020)	%
Android	70,43
iOS	29,06
Samsung	0,16
KaiOS	0,11
Windows	0,07
Tizen	0,02
Series 40	0,02
Nokia	0,02
BlackBerry OS	0,01
Linux	0,01
Other	0,09

Рисунок 1.1 − Статистика використання мобільних платформ

Власні програми під конкретні ОС, розроблені виключно для певної платформи [1]. Ці програми розроблені мовами, сумісними з операційними системами. Наприклад, Apple віддає перевагу мовам Objective C та Swift для iOS [10], коли як Google - Java для Android. Використовуючи прийнятні мови, розробники-програмісти можуть краще використовувати нативні можливості цих платформ. Вбудований додаток, розроблений для Android, не буде працювати на iOS.

Кроссплатформенні програми можуть працювати одразу з багатьма платформами. Через частини ринку Android та iOS більшість

кроссплатформенних додатків обмежені двома ОС. Ці програми розроблені за допомогою HTML та CSS, оскільки ці частки технлогій не залежать від конкретних платформ. Існує кілька інструментів для розробки додатків на різних платформах, які допомагають розробникам створювати ці програми без особливих зусиль.

Власні програми оптимально використовують ресурси та функції платформи. Власні програми - це високоефективні додатки, які швидко реагують і з меншою ймовірністю зупиняються або закриваються з невідомих причин. Якщо розробники мають достатні знання про платформу, над якою працюють, вони можуть налаштувати власні програми, щоб виділити найкращі можливості цієї платформи.

Крос-платформні програми часто мають проблеми продуктивності. Оскільки вони є вбудованим підходом, що відповідає одній програмі, щоб ці програми діяли на певних пристроях.

Нативні програми можуть використовувати функцію пристрою, особливо з iOS, яка працює лише на власних пристроях Apple. Ще однією великою перевагою нативних програм є те, що вони надають автономні функції, що є неможливим з крос-платформними додатками.

Програми, розроблені на кросплатформі, не можуть використовувати унікальні функції своїх пристроїв, оскільки вони мають обмежений доступ до API. Оскільки вбудована розробка для різних пристроїв з різними функціями складна, розробники часто уникають деяких підстав, та звертають увагу на кросплатформу.

Розробка власної програми займає вдвічі більше часу, ніж розробка крос-програм. Вартість також вища, оскільки для створення часто потрібно більше одного додатка. Ремонт трудомісткий і дорогий, оскільки розробники виявляють

помилки та проблеми кожної програми та відповідно створюють різні оновлення..

Комбіновані програми є менш дорогими для розробки та обслуговування. Ви можете інвестувати в програму, і це все, що вам потрібно почекати. Однак іноді багато проблем та помилок затьмарюють цю перевагу [2].

Завдяки хорошій продуктивності, високій швидкості та використанню пристрою спеціальні програми забезпечують чудовий досвід. Дизайнери та розробники мають творчу свободу створювати прекрасні програми та оптимізувати програмування.

Розробники можуть також створювати програми реактивного підходу, такі функції часто обумовлені швидкістю. Розробникам та дизайнерам важко одночасно задовольнити всі вимоги UX до різних платформ. Як правило, додатки, що містять різні програми, не забезпечують бажаного користувацького досвіду.

Отже важливо вибрати платформу, яка відповідає потребам, вимогам, а також цільовій аудиторії.

1.3 Інструменти розробки мобільних додатків

Перед початком розробки було розглянуто найпопулярніші технології для розробки веб-додатків.

На основі статистичних даних спільноти StackOverflow, React та Hamarin є одними з найпопулярніших ТОП-10 фреймворків станом на 2019 рік. Згідно з опитуванням 69 000 програмістів, React використовується втричі більше, ніж Hamarin, ніж будь-який інший фреймворк у світі (28,3% проти 7,4%). Node.js та AngularJS залишаються найбільш використовуваними в цій категорії. Програми можуть вибирати з ряду варіантів. Ці статистичні дані також не є найкращою

основою для порівняння, але тенденція популярності зростає із продуктами Facebook. На рисунку 1.2 наведено статистичні дані щодо використання різних технологій.

На розвиток кросплатформи впливає необхідность мати можливість одночасно керувати додатками програм, не переписуючи весь код. Згідно зі статистичними даними, у 2019 році ринок міжплатформенного програмного забезпечення перевищить 10,5 млрд доларів.

React Native був розроблений Facebook для внутрішнього розвитку, який згодом став більш повною формою [6]. Генерація React Native дозволяє розробникам писати дані на реалізації JavaScript - TypeScript, які потім можна успішно переглянути та реалізувати у відповідних операційних системах.

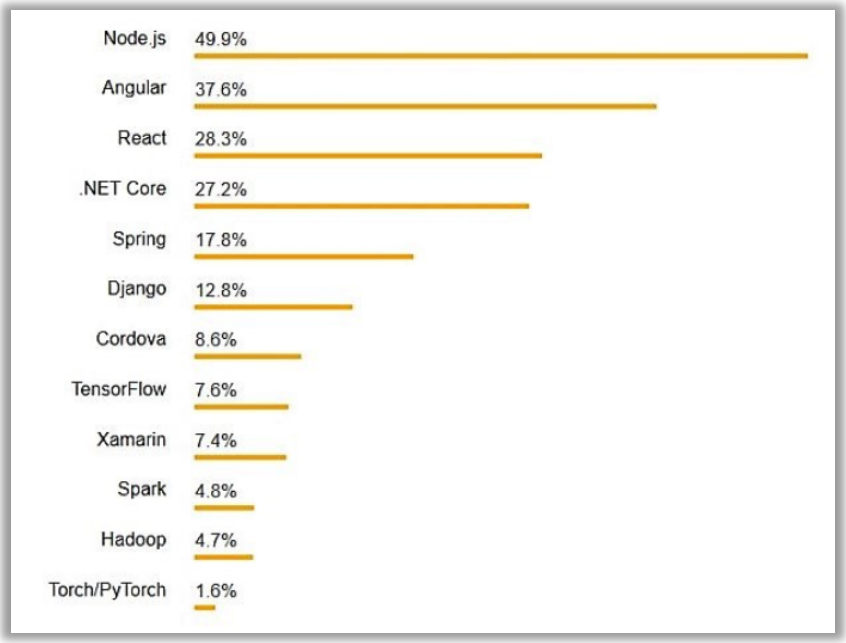

Рисунок 1.2 − Статистика використання фреймворків та бібліотек

З 2017 Google представив свою власну кросплатформенну реалізацію для розробників.

Google пропонує три основні функції, що виділяють його серед іншого крос-платформного вмісту: швидкий розвиток, гнучкий користувальницький інтерфейс та ефективна продуктивність [7]. Ці бренди стали головною опорою продажів Flutter'a.

Переваги Flutter:

– Hot Reload − ви можете швидко «зберегти» проект і миттєво протестувати його, створити взаємодію, додати нові функції та виправити дефекти в процесі розробки;

– поставляється з інструментами, побудованими на матеріальному дизайні (апаратні компоненти, використані приклади), шаблонами, API анімації, "рідною" ротацією;

– пропонує сучасну модель розробки та широкий асортимент двовимірних моделей. Він також постачається з потужним та гнучким API для 2D, анімації, ефектів тощо;

– дозволяє розробникам повторно використовувати наявний код Java, Swift і Purpose C та отримати доступ до своєї роботи та SDK для iOS та Android;

– різноманітні інструменти та бібліотеки, які розробники можуть використовувати для побудови, вдосконалення та розгортання своїх програм;

– вбудована схема вводить створення нових реалізацій шляхом підключення додаткових пучків.

– Недоліки Flutter'a:

– складна мова програмування;

– немає JSX;

– складності при стилізації;

– не дає інструментів для збереження стану додатків, однак це може бути виправлено шляхом серіалізації поточного стану.

1.4 Проблема відстеження пацієнтів хворих на COVID-19

Додатки для відстеження контактів з хворими з'явилися ще задовго до пандемії коронавируса і були досить ефективними. Саме тому після різкого збільшення кількості інфікованих COVID-19 приватні організації та уряди багатьох країн почали роботу над розробкою власних програм, які повинні допомогти у відстеженні хворих і скорочення кількості жертв[3].

Нещодавно уряд Великобританії оголосив, що припиняє розробку власного додатка для відстеження можливих контактів з інфікованими коронавірусів, в той же день уряд Канади заявило, що починає розробляти власний додаток. Одночасно з цим, за рішенням норвезького державного органу, який займається захистом даних, орган охорони здоров'я Норвегії мав видалити всі зібрані дані і припинити їх подальше використання.

Така неоднозначна ситуація в різних країнах демонструє відсутність єдиного і швидкого вирішення, яке дозволяло б використовувати сучасні технології для зниження рівня інфікування COVID-19 і допомагало б світовим медичним установам відстежувати розповсюдження хвороби.

Існує два основні методи, які використовуються для отримання даних про контакти громадян. Перший - за допомогою глобальної системи позиціонування (GPS). У цьому методі для визначення розташування громадян використовується супутникова радіонавігації.

Для другого й найбільш безпечного методу використовується Bluetooth і визначається близькість інших користувачів на основі потужності сигналу - таким чином вимірюється відстань між громадянами, а не відстежується їх фактичне місцезнаходження. Частина рішень використовують комбінацію Bluetooth і GPS, а в деяких навіть використовується мережеве відстеження місцезнаходження, однак останній метод має проблеми з порушенням

конфіденційності місцезнаходження, і використовується тільки декількома розробниками.

Якщо ж інфікованого користувача можна ідентифікувати, а всі дані зберігаються і обробляються централізовано, виникає серйозна проблема з порушенням конфіденційності.

Однак і така система має свої переваги: частина даних, які обробляються централізовано, може бути використана для інформування вчених про переміщення населення, а також для швидкої ідентифікації гарячих точок, куди необхідно терміново виділити медичні ресурси. Якщо, наприклад, при установці додатка користувачам необхідно ввести поштовий індекс, за цими даними вчені можуть мати можливість передбачити поширення захворювання. Це навряд чи дозволить точно ідентифікувати користувача, оскільки однаковий поштовий індекс використовують сотні або тисячі людей, але значно звужує коло підозрюваних осіб.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Вибір технології реалізації

Було обрано певний стек технологій для реалізації інформаційної системи з огляду на власне дослідження, яке було описано у попередніх розділах.

Перший стек узагальнює всі переваги та недоліки різних технологій і вибирається для першої частини: JavaScript, TypeScript, React Native. При порівнянні технологій бекенду було вирішено зосередити увагу на NodeJS, MongoDB, Express.js.

Кожна реалізація є однією з найпопулярніших та найоптимізованих технологій сьогодні і відповідає певним стандартам, що дозволяють іншим програмам підтримувати майбутні інформаційні системи.

2.2 Мова програмування JavaScript

JavaScript - це мова програмування, що є прототипна-орієнтованою. Він відображає мову ECMAScript, чиїм прототипом спочатку і був.

Основною характеристикою цієї мови є те, що на неї вплинуло інші мови програмування (Python, Java, Pascal), щоб забезпечити максимальну зручність тим JavaScript розробникам, яким не мають відповідної підготовки та глибоких знань.

Переваги JavaScript.

– Жоден сучасний браузер не обходиться без підтримки JavaScript.

– З використанням написаних на JavaScript плагінів і скриптів впорається навіть не фахівець.

– Корисні функціональні настройки.

– Постійно удосконалюється мова - зараз розробляється бета-варіація проекту, JavaScript2.

– Взаємодія з додатком може здійснюється навіть через текстові редактори - Microsoft Office і Open Office.

– Перспектива використання мови в процесі навчання програмуванню і інформатиці.

Переваги React JS:

– Virtual DOM може підвищити продуктивність високонавантажених додатків, що може знизити ймовірність виникнення можливих незручностей і покращує користувальницький досвід;

– Використання ізоморфного підходу допомагає виробляти рендеринг сторінок швидше, тим самим дозволяючи користувачам відчувати себе більш комфортно під час роботи з вашим додатком. Пошукові системи індексують такі сторінки краще. Оскільки один і той же код може бути використаний як в клієнтської, так і в серверній частині програми, немає необхідності в дублюванні одного і того ж функціоналу. В результаті час розробки і витрати знижуються;

– Завдяки перевикористанню коду стало набагато простіше створювати мобільні додатки. Код, який був написаний під час створення сайту, може бути знову використаний для створення мобільного застосування. Якщо ви плануєте використовувати не тільки сайт, але і мобільний додаток, немає необхідності наймати дві великі команди розробників[5].

JavaScript вийшов з браузера і потрапив на смартфони. На ньому дійсно можна писати і створювати повноцінні мобільні додатки. і якщо вони показані звичайному користувачеві, навряд чи він помітить будь-яку різницю з «нативними» рішеннями.

Саме тому було вирішено використовувати саме цю мову програмування для створення інформаційної системи. Її переваги дуже істотні, а недоліків мало.

2.3 Фреймворк для розробки мобільних додатків React Native

React Native - є інтегрованою структурою, яка дозволяє створювати програми, використовуючи лише JavaScript. Однак, на відміну від інших вбудованих мобільних технологій, ви не створюєте мобільний веб-додаток (веб-додаток записується у власний контейнер). Основу вашого коду JavaScript складає мобільний додаток, який нічим не відрізняється від додатка iOS, побудованого за допомогою Objective-C, або додатка Android, який використовує Java. Це означає, що React Native має переваги як для мобільних, так і для інтегрованих додатків. У той же час переваги React React повністю розкриті, а мобільний додаток реалізований.

React Native має світле майбутнє, і всім потрібно працювати в цьому напрямку. Це дивовижна можливість побудувати міжплатформові системи, яка займе менше часу та отримати доступ до всієї інтеграції. Підтримка та спільнота дуже сильні та дають можливість розвиватися у цьому напрямку.

Отже, якщо ви знаєте лише JavaScript (HTML, CSS) та його бібліотеку React and React Native, ви можете сміливо продовжувати розробку сайту та мобільної програми.

2.4 Платформа для розробки серверної частини NodeJS

Node.js – це середовище виконання JavaScript з відкритим вихідним кодом і крос-платформенним середовищем виконання JavaScript. Це популярний інструмент для будь-якого проекту.

Більшість веб-розробників реалізують Node.js завдяки своїм потужним функціям. Деякі з особливостей Node.js:

- швидке виконання коду;
- високомасштабність;

- неблокуючі API;
- відсутність буферизації.

Програма Node.js запускається в одному процесі без створення нового потоку для кожного запиту. Node.js надає набір асинхронних примітивів вводу/виводу в стандартній бібліотеці, що запобігає блокуванню коду JavaScript.

Характеристики Node.js:

- Node.js – це в основному сервер, здатний виконувати JavaScript;
- Node.js – відкрита і крос-платформна система для створення веб-додатків;
- забезпечує асинхронні та керовані подіями API;
- будь-який програміст, знайомий з JavaScript, може швидко вивчатися Node..

Є багато компаній, які використовують Node.js, такі як eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, Wikipins, Yahoo !, IBM, Groupon, LinkedIn, Netflix та багато інших. Це вагомий показник, саме тому було обрано саме NodeJS.

2.5 Система управління базами даних MongoDB

MongoDB – це документна орієнтована система управління базами даних з відкритим вихідним кодом, яка не вимагає опису схеми таблиць[3]. Вважається класичним прикладом відповідних NoSQL-систем, використовує документи, тип яких подібен до формату JSON і схему бази даних. Написана на мові C++. Застосовується в веб-розробці, наприклад, в рамках JavaScript-орієнтованих стекік MEAN, MERN, MEVN.

Система спокійно працює з набором даних, тобто репліки можуть містити від двох копій даних на різних точках. Умовно екземпляр набору може завжди виступати в ролі допоміжної або головної репліки. Всі CRUD операції можна

здійснити як з основною так і з допоміжною реплікою. Допоміжні репліки підтримують в синхронізованому стані копії даних.[4] У разі, коли основна репліка дає збій, набір реплік проводить вибір, яка з реплік повинна стати основною. Другорядні репліки можуть додатково бути джерелом для операцій читання.

MongoDB підходить для наступних застосувань:

- зберігання та реєстрація подій;
- системи управління документами і контентом;
- електронна комерція;
- гри;
- дані моніторингу, датчиків;
- мобільні додатки;
- сховище операційних даних веб-сторінок (наприклад, зберігання коментарів, оцінок, профілів користувачів, сеанси користувачів).

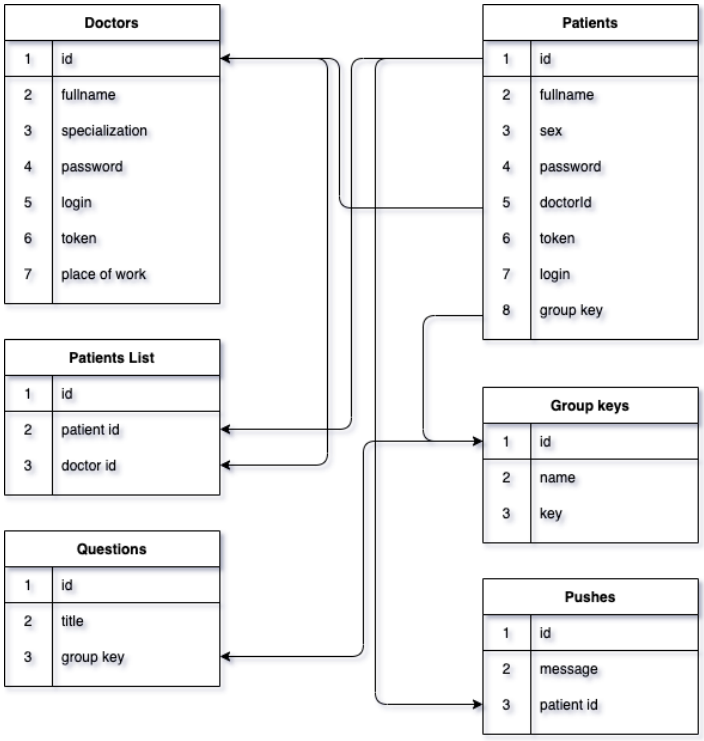

Рисунок 2.1 – Прототип бази даних інформаційної системи

2.6 Прототипи екранів інформаційної системи

Для створення прототипів було застосовано інструмент для UI/UX дизайну під назвою Figma. Вона дозволяє повністю з нуля створити будь який прототип чи повноцінний дизайн, логотип чи проекцію у повному обсязі.

За допомогою готових компонентів можна створити екрани мобільного додатку (рис. 2.2) й розмістити на них такі елементи: buttons, checkboxes, radiobuttons, textfields, images, blocks.

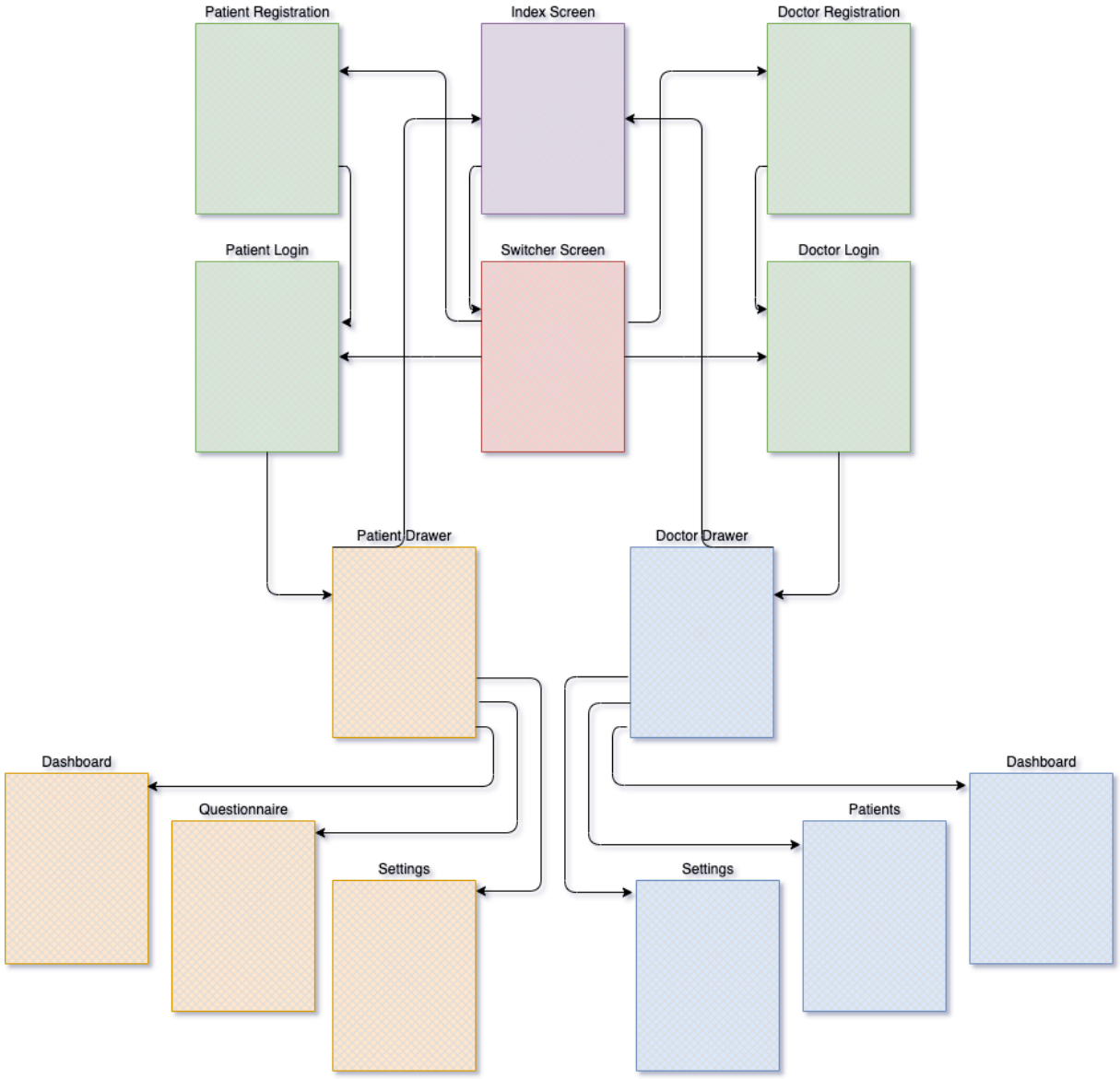

Рисунок 2.2 – Взаємозв’язок екранів інформаційної системи

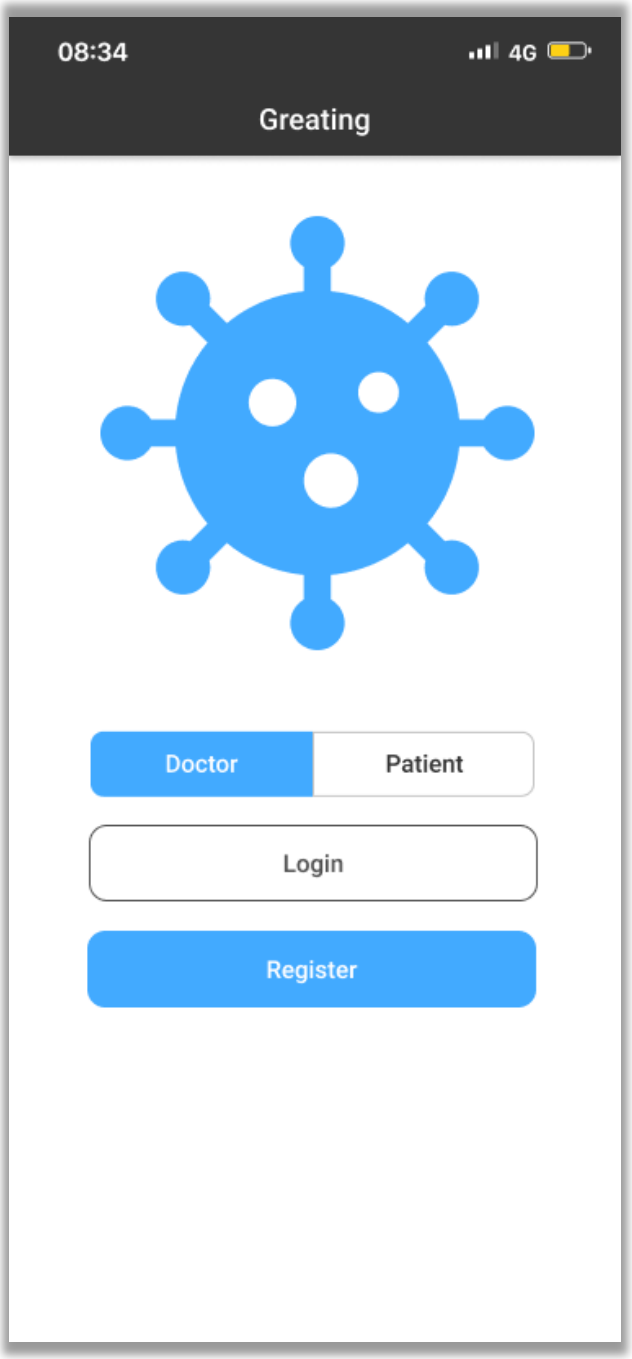

Рисунок 2.3 – Прототип екрану входу

На прототипі (рис. 2.3) відображається логотип мобільного додатку. Також є segments з можливістю у будь який час перемикати під яким користувачем потрібно зареєструватися або увійти (доктор, пацієнт).

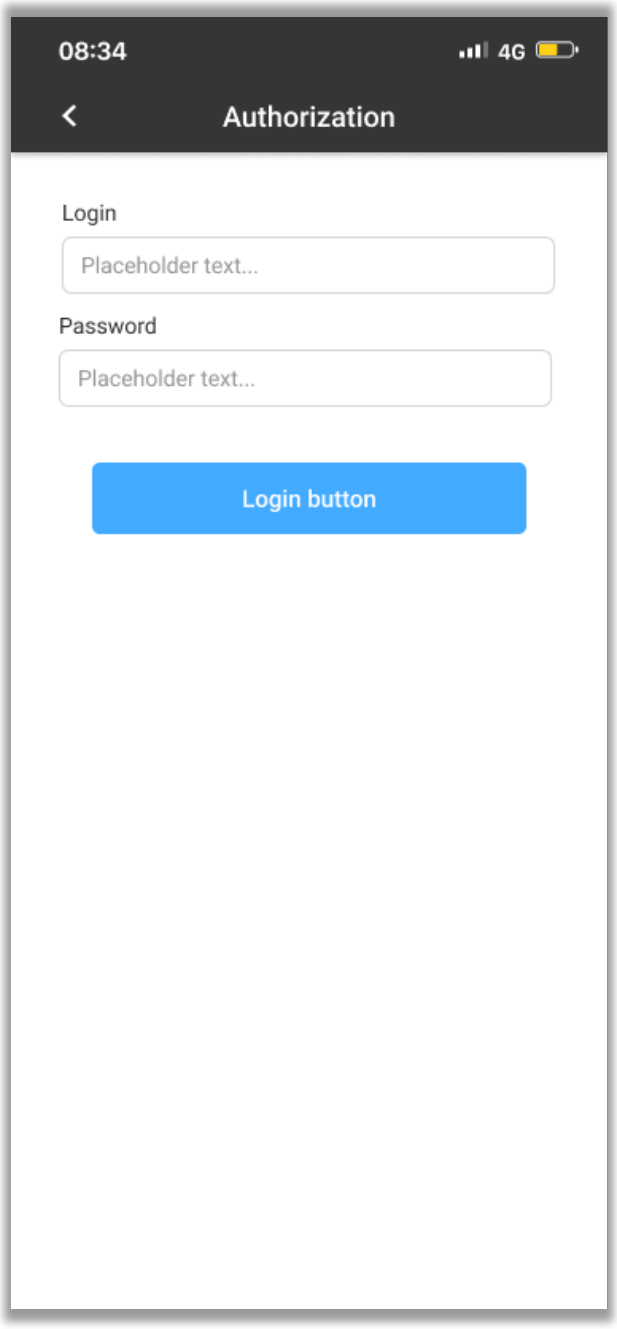

Рисунок 2.4 – Прототип екрану авторизації

На прототипі (рис. 2.4) зображена форма з двума текстовими полями, для вводу логіну та паролю. Також зображена кнопка, яка буде запускати функціонал авторизації. Для доступу до попереднього екрану зображена кнопка «Назад»

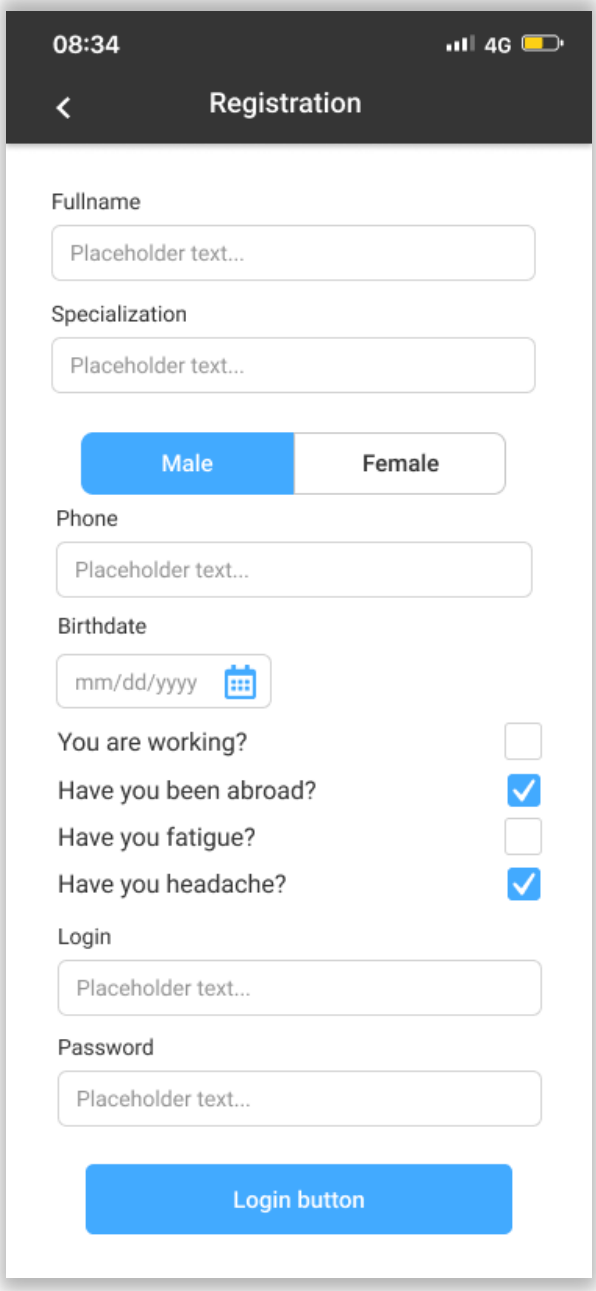

Рисунок 2.5 – Прототип екрану реєстрації пацієнта

На прототипі (рис. 2.5) зображена форма реєстрації пацієнта в інформаційній системі. На ній зображені основні поля для вводу, такі як ім’я, телефон, логін та пароль. Також зображений вибір дати народження. За допомогою прапорців можна відповісти на декілька уточнюючих питань а кнопка служить для запуску функціоналу реєстрації.

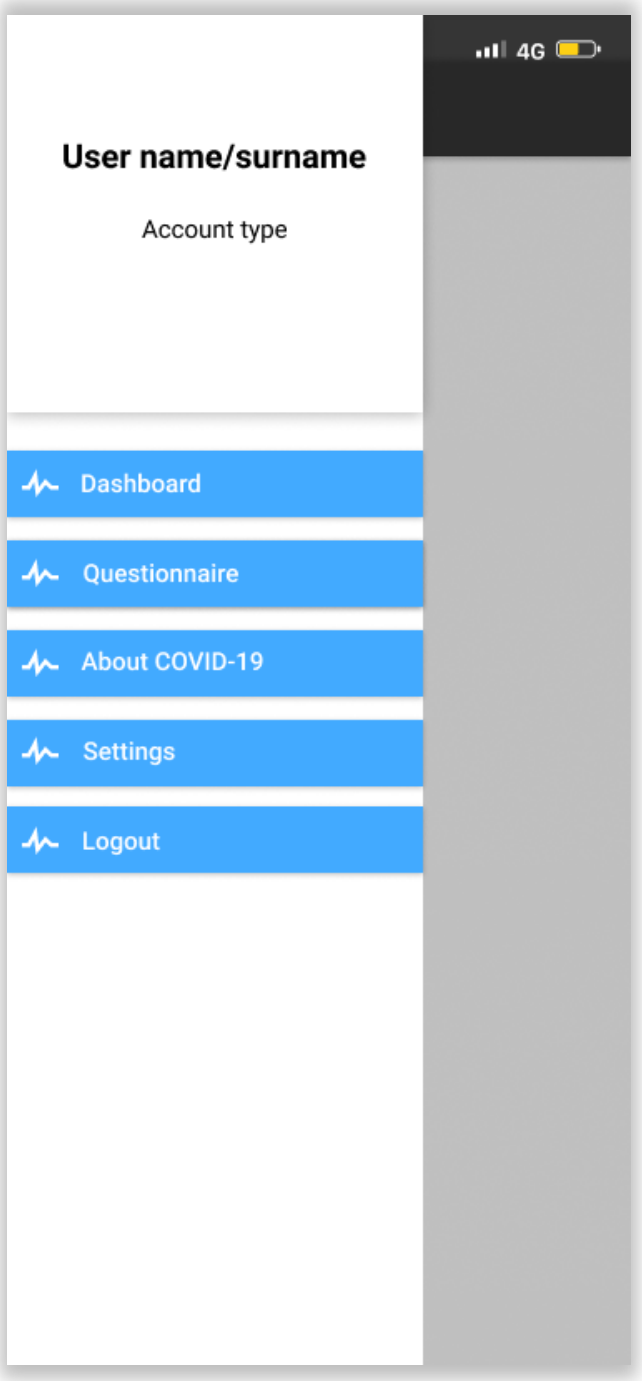

Рисунок 2.6 – Прототип відкритого меню для пацієнта

На прототипі (рис. 2.6) зображено бокове відкрите меню для типу профіля – пацієнт. На ньому зображено основна інформація профіля і можливість перейти на всі доступні сторінки інформаційної системи для пацієнта.

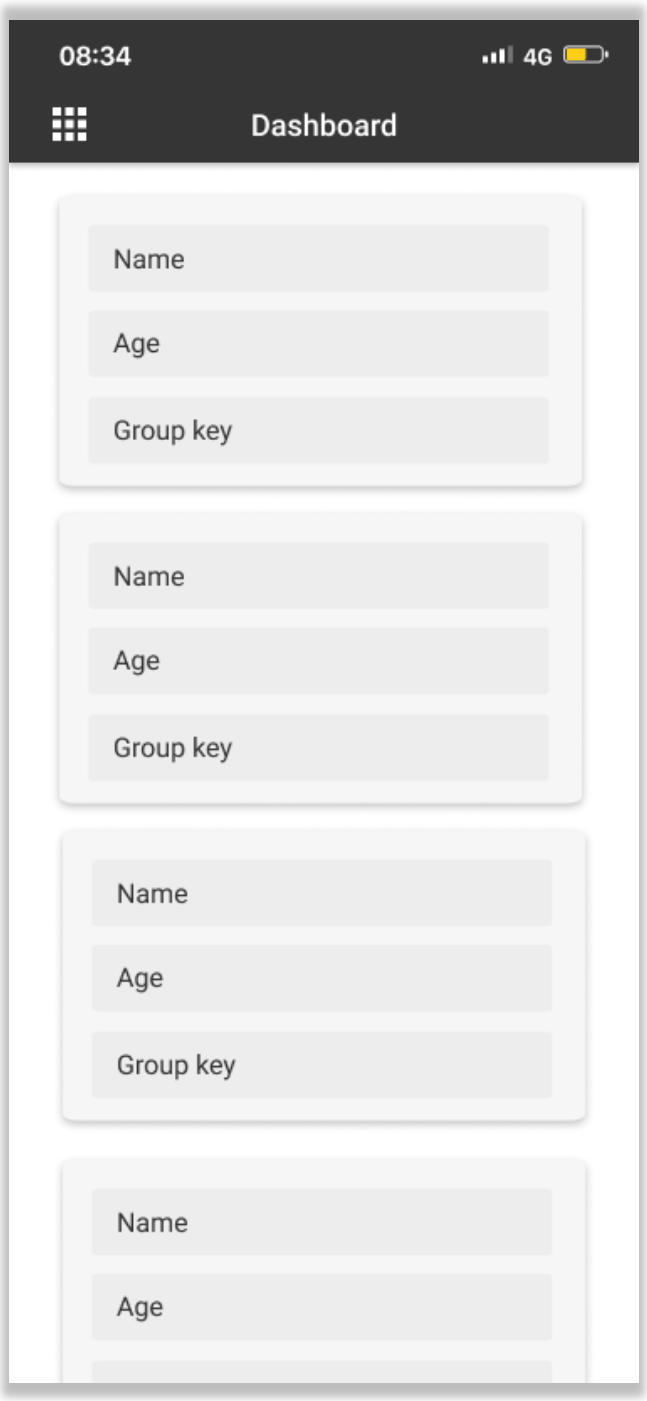

Рисунок 2.7 – Прототип головної сторінки лікаря

Прототип (рис. 2.7) відображає список пацієнтів, які закріплені за певним лікарем. У кожному елементі списку відображається основна інформація про пацієнтів, їх стан здоров’я. Також зверху є можливість відкрити бокове меню лікаря для доступу до інших сторінок.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

Спочатку був розроблений повноцінний дизайн інформаційної системи, який базується на прототипі наведеному у розділі 2. Було застосовано інструмент для UI/UX дизайну під назвою Figma, який дозволяє легко перейти від прототипу до повноцінної реалізації дизайну (рис. 3.1).

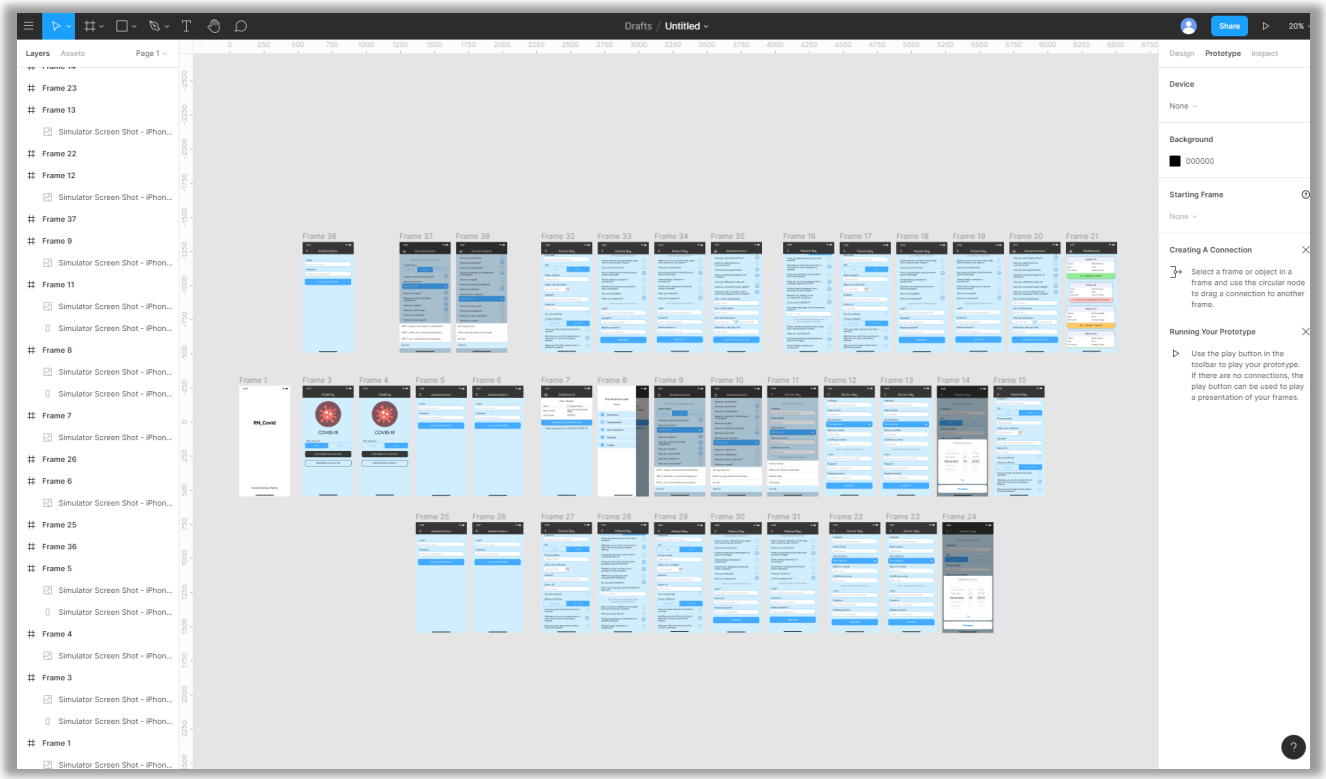

Рисунок 3.1 – Структура дизайну головних сторінок

Для налаштування початкового проекту на React Native було застосовано команду: `react-native init CovidApplication --template react-native-template-typescript`

Ця команда автоматично додає, та встановлює наступний список залежностей, який потрібен для реалізації проекту:

- @react-native-community/masked-view
- react
- react-native

- react-native-loader
- react-native-exit-app
- react-native-fast-image
- react-native-gesture-handler
- react-native-indicators
- react-native-masked-text
- react-native-reanimated
- react-native-safe-area-context
- react-native-swiper

Також для підтримки і реалізації TypeScript у проекті було встановлено наступні компоненти:

- typescript
- @types/jest
- @types/react-native
- @types/react-test-renderer
- @typescript-eslint/eslint-plugin
- @typescript-eslint/parser
- @types/react-native-vector-icons
- @types/react-native-snap-carousel

Весь файл залежностей та пакетів package.json наведений у Додаток А.

Для реалізації декількох екранів та повноцінної навігації між сторінками (рис. 3.2) було застосовано пакет react-navigation. React Navigation дозволяє розробити такі види реалізації навігації: Drawer Navigator, Screens Navigator, Stack Navigator, Bottom Tab Navigator.

В нашому випадку потрібно встановити Drawer, Screens, Stack командою: npm install @react-navigation/native react-native-reanimated react-native-gesture-

handler react-native-screens react-native-safe-area-context @react-native-community/masked-view. Код частини навігації наведений у Додаток Б.

Рисунок 3.2 – Приклад навігації react-navigation між сторінками

Для стилізації екранів інформаційної системи було використано технлогію CSS. CSS займається зовнішнім виглядом додатку.

За допомогою CSS можна керувати кольором тексту, розміром шрифту (рис. 3.3), шириною абзацу, розміром і розміщенням стовпців, оригінальними зображеннями або кольорами, що використовуються, макетом, параметрами відображення для різних пристроїв та розміром екрана тощо. Часто CSS додається до мов розмітки HTML або XHTML.

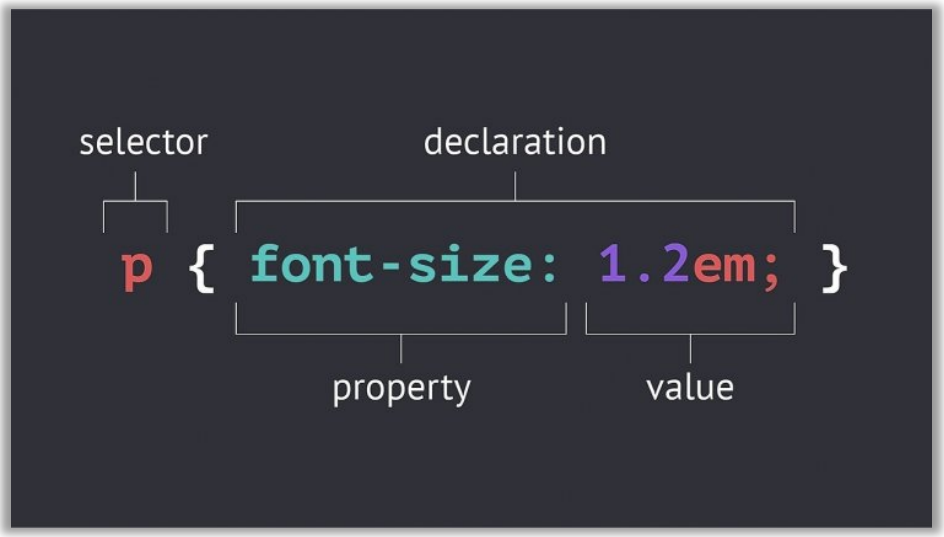

Рисунок 3.3 – Структура типового CSS-селектора

Для комфортної розробки та спрощення перевикористання CSS-компонентів було використано технологію Styled Components. Стилізовані компоненти - це інструмент CSS-in-JS, який запобігає розриву між компонентами та стилем, пропонуючи численні функції, які дозволять функціонально та багаторазово використовувати стилістичні компоненти.

Основою CSS є можливість націлювання на будь-який елемент HTML - глобально - незалежно від його позиції в дереві DOM. Це може бути перешкодою при використанні з компонентами, оскільки компоненти вимагають до розумної міри колорації (тобто збереження таких активів, як стан та стиль) ближче до місця їх використання (відоме як локалізація).

За словами власних слів React, стилізовані компоненти є «візуальними примітивами для компонентів», і їх метою є надання нам гнучкого способу стилізації компонентів. Результатом є щільне зчеплення між компонентами та їх стилями.

Приклад стилізації компонента BottomContainer:

```
const BottomContainer = styled.View<TemplateProps>`
```

```
width: 100%;
${FLEX('row', 'flex-start', 'flex-start')}
position: relative;
${({template=e.default})=>template.slug === 'new'? `
  padding: 5px 10px 0;
` : `
  padding: 0 10px 0;
`}
min-height: 35px;
background: white;
border-bottom-left-radius: ${ORDER_CONFIG.borderRadius}px;
border-bottom-right-radius: ${ORDER_CONFIG.borderRadius}px;
margin-top: 10px;
${({ displayType, template }) => displayType === 'expanded' ? `
  box-shadow: 0px 0px 5px #0000000D;
  elevation: 2;
  border-top-color: ${template.borderColor};
  border-top-width: 1px;
` : ``}`
```

Для використання потрібних комплексних векторних іконок, потрібно налаштувати пакет react-native-vector-icons. React Native Vector Icons - дуже популярні іконки в React Native. Векторні іконки ідеально підходять для кнопок, логотипів та панелей навігації / вкладок. Векторні іконки легко розширити, оформити та інтегрувати у проект (рис. 3.4).

Список категорій іконок, доступних у React Native Vector Icons:

- AntDesign
- Entypo
- EvilIcons
- Feather
- FontAwesome
- Fontisto
- Foundation
- Ionicons
- MaterialIcons
- MaterialCommunityIcons

- Ocicons
- Zocial
- SimpleLineIcons

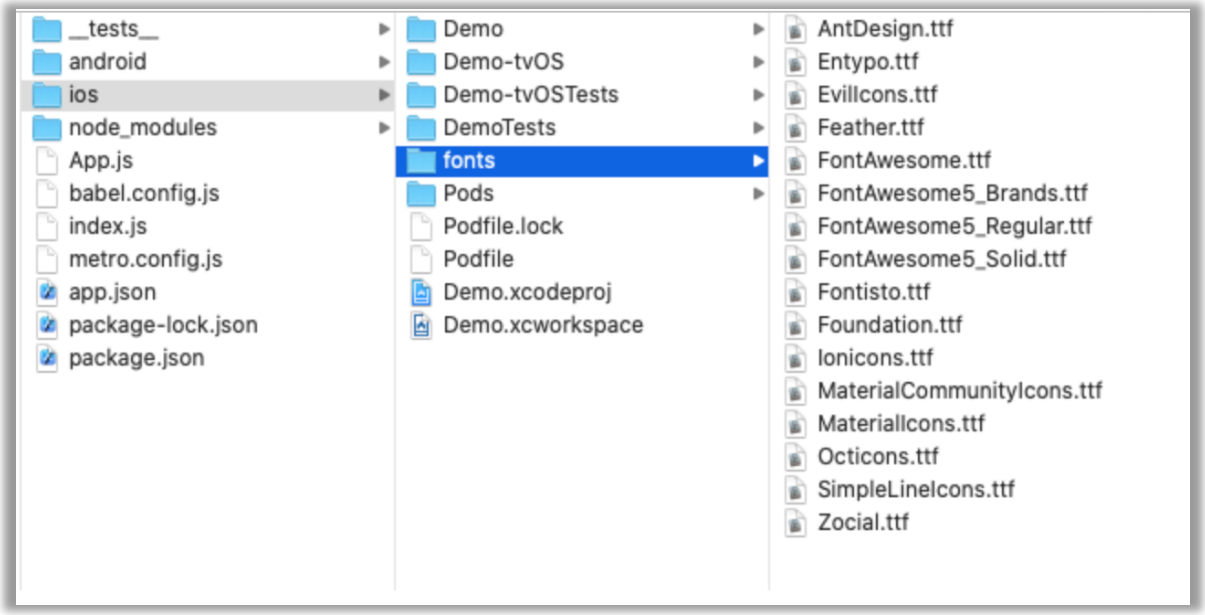

Рисунок 3.4 – Налаштовані векторні іконки у XCode для iOS

Список доступних шрифтових іконок, використаних в інформаційній системі в Info.plist:

```
<key>UIFonts</key>
<array>
  ...
   <string>FontAwesome.ttf</string>
   <string>FontAwesome5_Brands.ttf</string>
   <string>FontAwesome5_Regular.ttf</string>
   <string>FontAwesome5_Solid.ttf</string>
   <string>Foundation.ttf</string>
   <string>Ionicons.ttf</string>
   <string>MaterialIcons.ttf</string>
   <string>MaterialCommunityIcons.ttf</string>
   <string>SimpleLineIcons.ttf</string>
   <string>Octicons.ttf</string>
   <string>Zocial.ttf</string>
</array>
```

Для реалізації http запитів для отримання інформації с серверу використано бібліотеку Axios. Axios - це бібліотека Javascript, яка використовується для надсилання HTTP-запитів з node.js або XMLHttpRequests від браузера, який також підтримує ES6 Promise API. Axios використовує двоетапний процес при роботі з даними JSON; після первинного запиту вам потрібно буде викликати метод .json () у відповіді, щоб отримати фактичний об’єкт даних.

Axios просто відразу повертає об'єкт даних, який очікується. Крім того, будь-яка помилка із запитом HTTP успішно виконає блок .catch () прямо з коробки.

Переваги Axios над іншими реалізаціями:

- Легко зробити XMLHttpRequests з браузеру або додатку
- Здійснює http-запити у node.js
- Підтримує API Promise
- Запит та відповідь на перехоплення
- Трансформація данних запитів та відповідей
- Скасування запитів
- Автоматичне перетворення даних JSON
- Підтримка клієнта для захисту від XSRF

Приклад універсальної функції MAKE_FETCH, яка була реалізована у проекті:

```
const MAKE_FETCH: TAxiosMakeFetchMethod = (
    axiosConfig,
    config,
    nativeConfig = {
      identifier: 'UNDEFINED'
    },
    callback = () => {}
  ) => {
```

```
    let enabledLoader: boolean = false;
    let enabledErrorHandler: boolean = false;
    enabledLoader && loaderActions.SHOW();
    return new Promise(async (resolve, reject) => {
      try {
        let response = await Axios.request(axiosConfig);
        let message = response.status + axiosConfig.method + '
Axios ' + nativeConfig.identifier + ' URL:' + axiosConfig.url;
        Log.zelda(message, response);
        resolve(response);
        enabledLoader && await loaderActions.HIDE();
      }
      catch(e) {
        let error: TErrorConstructor = await (ERROR_TRANSLATER(e)
as Promise<TErrorConstructor>);
        let message = error.code + ' ' + axiosConfig.method + '
Axios ' + nativeConfig.identifier + ' URL:' + axiosConfig.url;
        Log.ruddy(message, error);
        reject(error);
        enabledLoader && await loaderActions.HIDE();
        enabledErrorHandler && alertActions.SHOW({
          title:
Translater.axios__error_default_title[language.slug],
          buttons: ['OK'],
          template: 'ERROR_TEMPLATE',
          description: error?.message || e
        });
      }})}
```

Повна реалізація контексту для запитів наведена у Додаток В.

У проекті Axios було використано для авторизації, реєстрації лікаря та пацієнта, для збереження данних на сервері, для отримання даних поточного користувача, який увійшов у систему, отримання списку лікарів, списку пацієнтів, пагінації, для опитування пацієнтів, отримання інформаційних сторінок, відправки сповіщень й подібних функцій.

Так як React Native це кросплатформа, то після закінчення розробки було автоматично сгенеровано дві платформи з відповідними даними для запуску на різних операційних системах – Android та iOS. Відображення структури наведено на рис. 3.5-3.6.

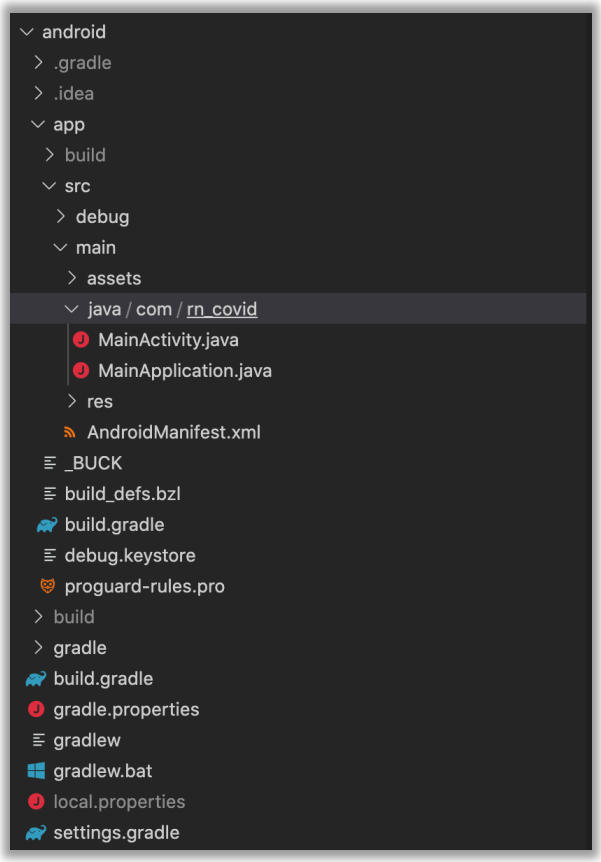

Рисунок 3.5 – Структура сгенерованої платформи Android

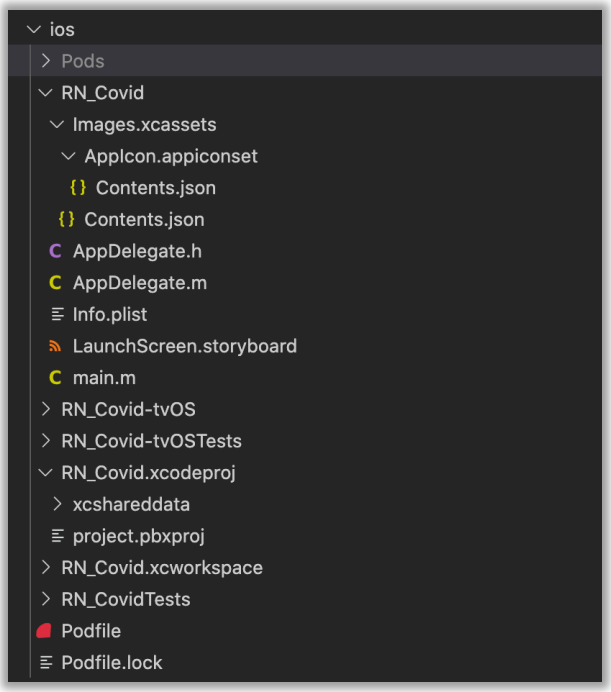

Рисунок 3.6 – Структура сгенерованої платформи iOS

Для перевикористання коду і компонентів була проведена декомпоновка та структуризація всіх файлів у відповідні директорії та файли (рис 3.7).

У директорії src/components було виділено компоненти форми, кнопок, екранні компоненті, індікатори завантаження, компоненти анімації та компоненти навигації. У src/components/form-components реалізовані такі компоненти: Accordion, Button, Checkbox, DateTimePicker, Divider, Input, ListPicker, Segments.

Директорія config містить конфігураційні файли інформаційної системи, такі як конфігурація асинхронного сховища, навігаційного дерева, закликів, enums та різних конфігураційних датасетів.

В /navigation знаходяться всі контейнери навігації для її подальшого використання на сторінкаї, які знаходяться у /screens.

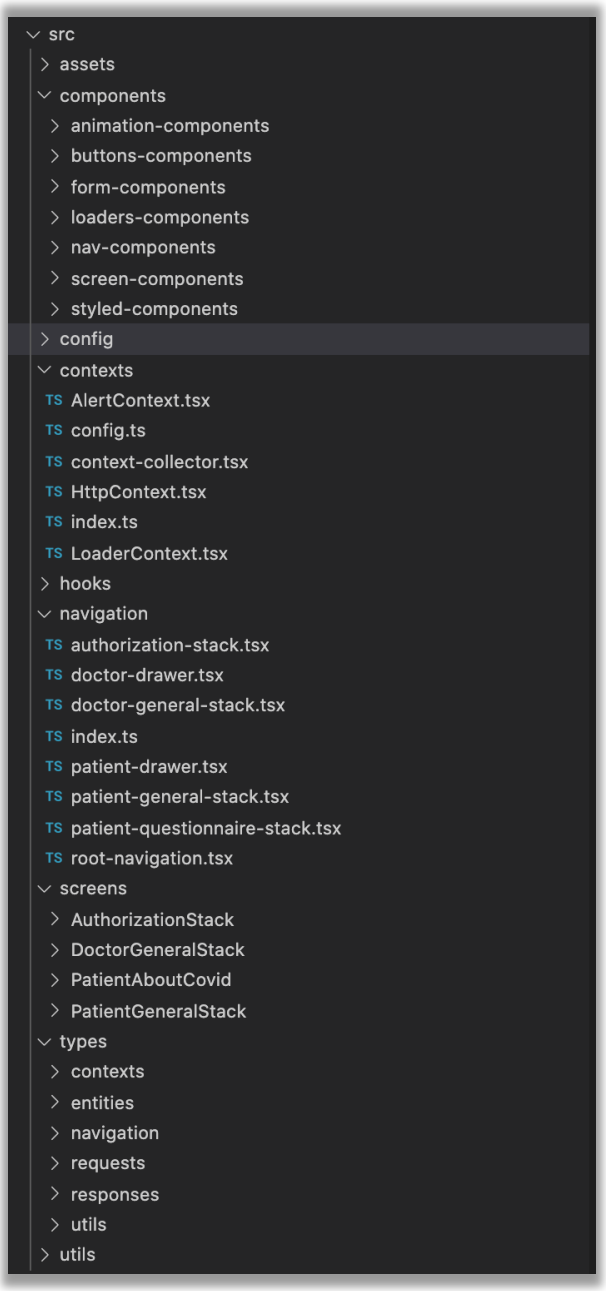

Рисунок 3.7 – Структура файлів проекту.

Зі сторони серверу було створено базу даних MongoDB, та моделі структур даних таблиць до неї. У якості фреймворку для бекенду було взято express.js.

Express - це легкий фреймворк веб-додатків, який допоможе впорядкувати веб-додаток у архітектуру MVC на стороні сервера. Можна використовувати різні варіанти для мови шаблонів (наприклад, EJS, Jade та Dust.js).

Потім використовуємо базу даних, як MongoDB з Mongoose (для моделювання), щоб забезпечити серверну програму для додатку Node.js. Express.js в основному допомагає керувати усім, починаючи від маршрутів, закінчуючи обробкою запитів та переглядів.

Redis - це сховище ключів / значень, яке зазвичай використовується для сеансів та кешування в програмах Node.js. З ним можна зробити набагато більше, але для цього його і використовують. MongoDB використовують для більш складних взаємозв'язків, таких як позиція, порядок, взаємозв'язок з користувачем (рис. 3.8). Є модулі (зокрема, connect-redis), які працюватимуть із Express.js.

Приклад створення моделі (таблиці) користувача:

```
import { model, Schema, Document } from 'mongoose'
import bcrypt, { compareSync } from 'bcrypt'

const UserModel = new Schema({
  fullname: {
    type: String,
    required: true
  },
  login: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true,
    set: (password: string) => {
      return bcrypt.hashSync(password, 10)
    }
  }
})

interface UserEntity extends Document {
  fullname: string,
  login: string,
  password: string
}
```

```
export default model<UserEntity>('users', UserModel)
```

Приклад коду прийняття запитів для реєстраці та авторизації користувача наведений у Додаток Г.

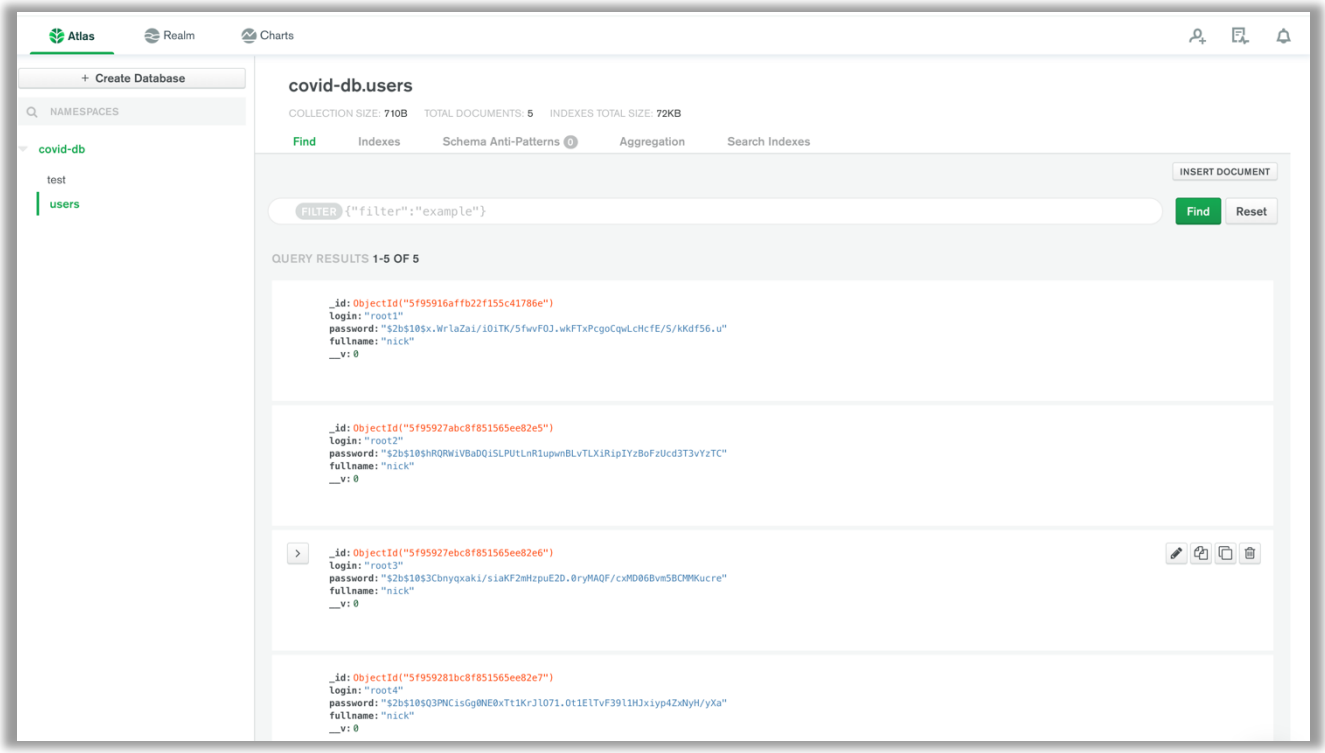

Рисунок 3.8 – Відображення таблиці Users у Compass MongoDB.

Для завершення роботи над інформаційною системою для моніторингу стану здоров’я хворих на COVID-19 були реалізовані всі необхідні сторінки. Нижче наведені зображення функціоналу інформаційної системи (рис. 3.9 – 3.13).

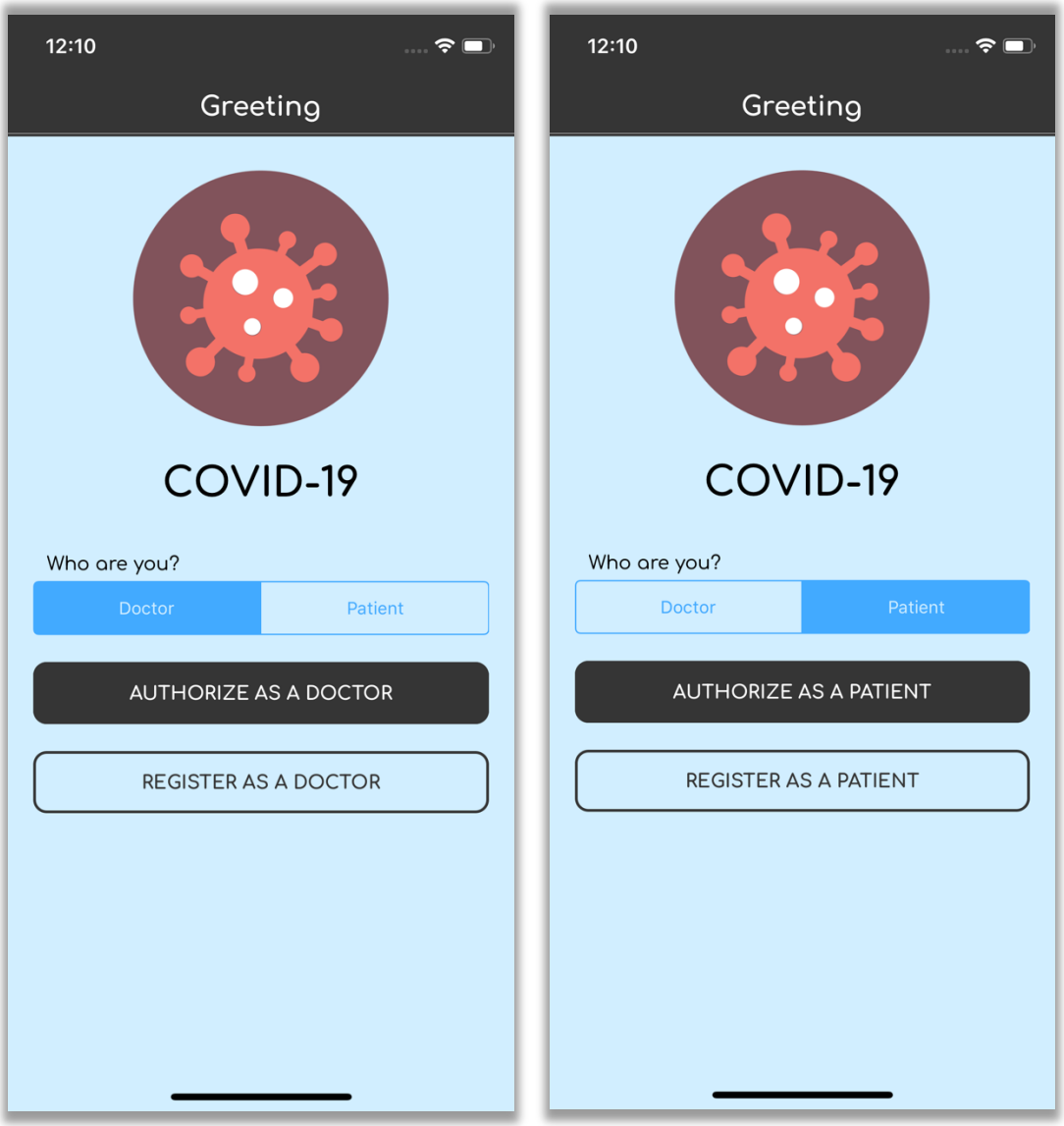

Рисунок 3.9 – Головні сторінки для пацієнта та доктора.

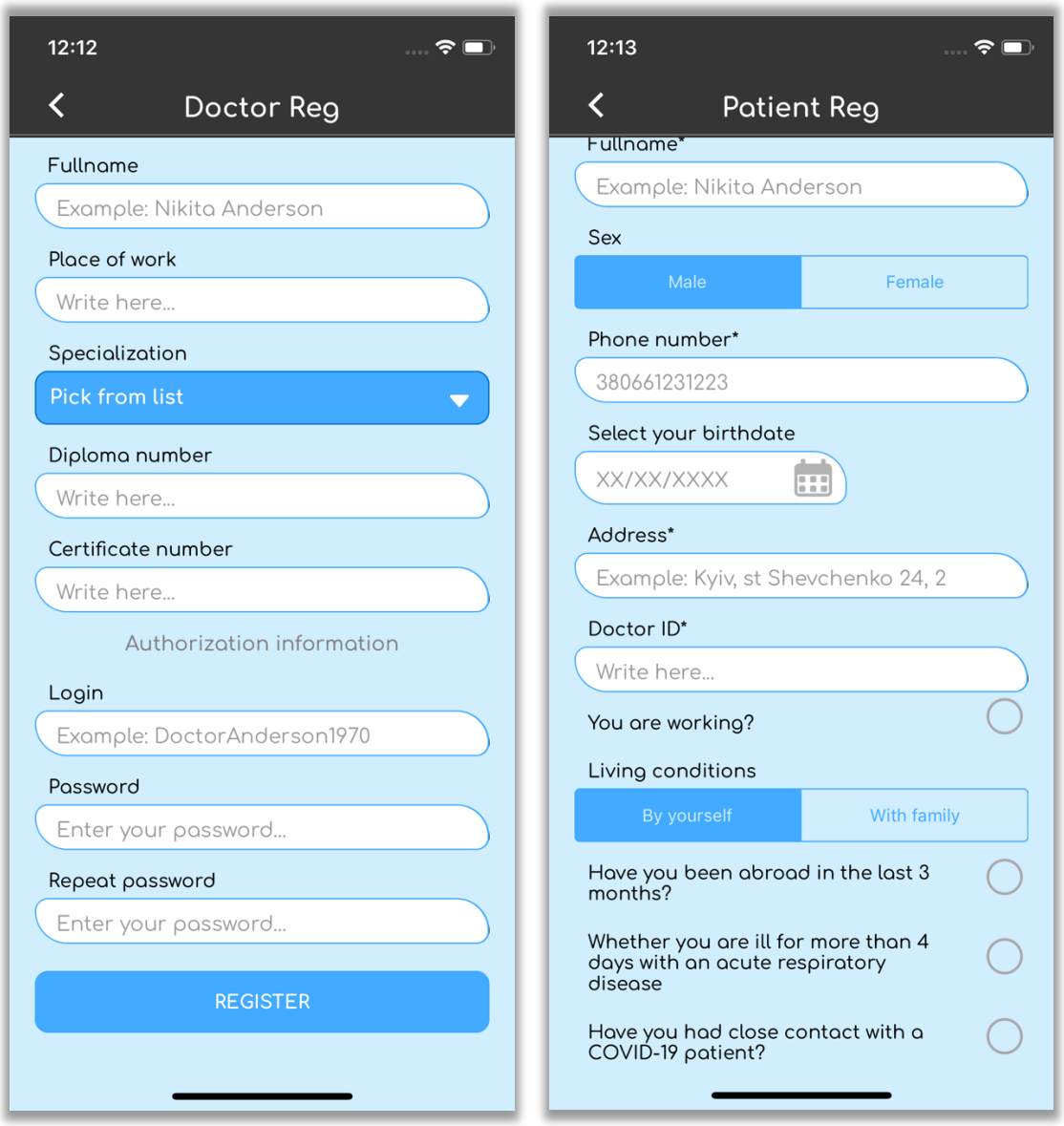

Рисунок 3.10 – Сторінки реєстрації користувача для різних ролей.

Рисунок 3.11 – Головна сторінка для авторизованих доктора і пацієнта

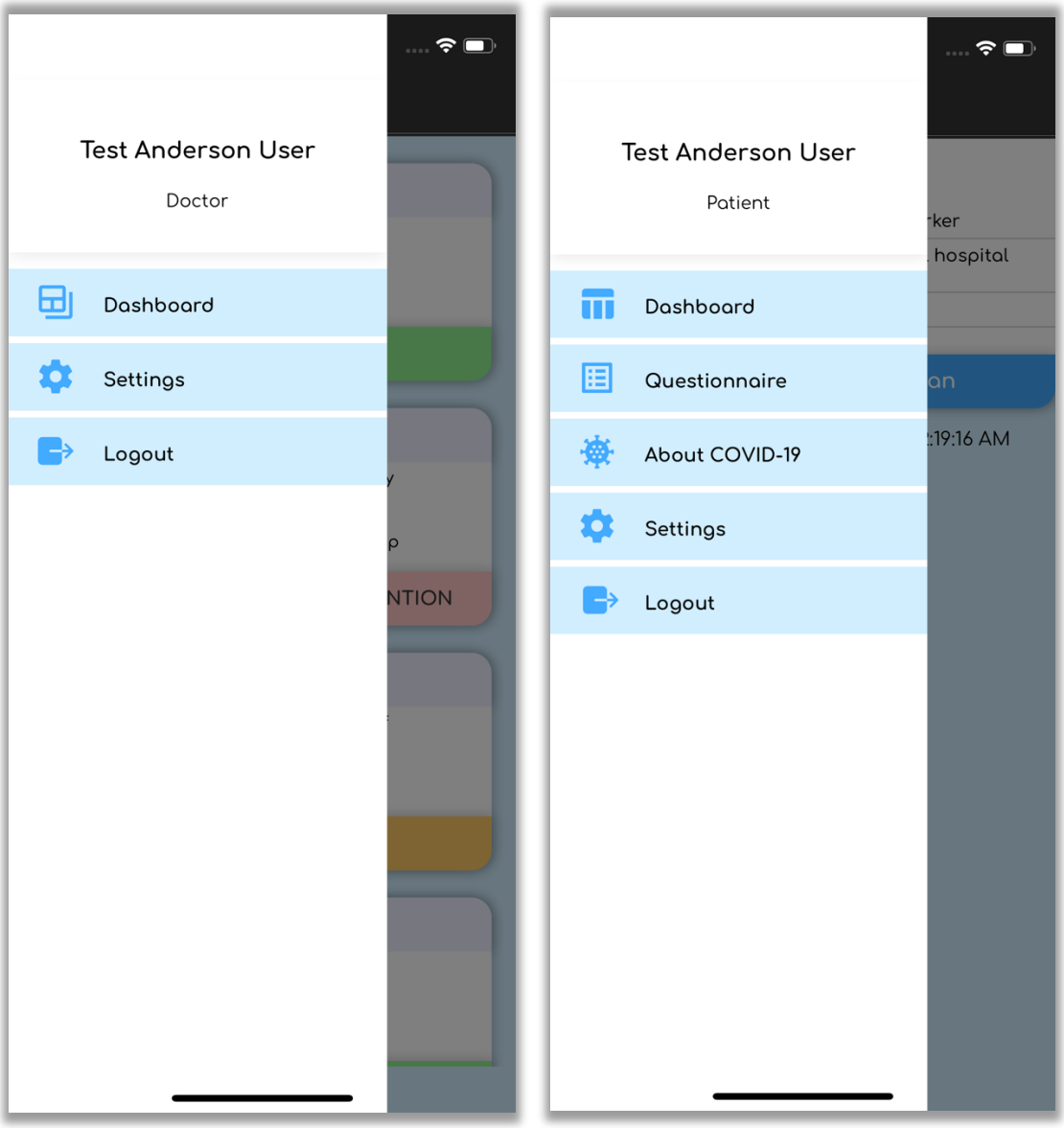

Рисунок 3.12 – Бокове меню для двох ролей користувача

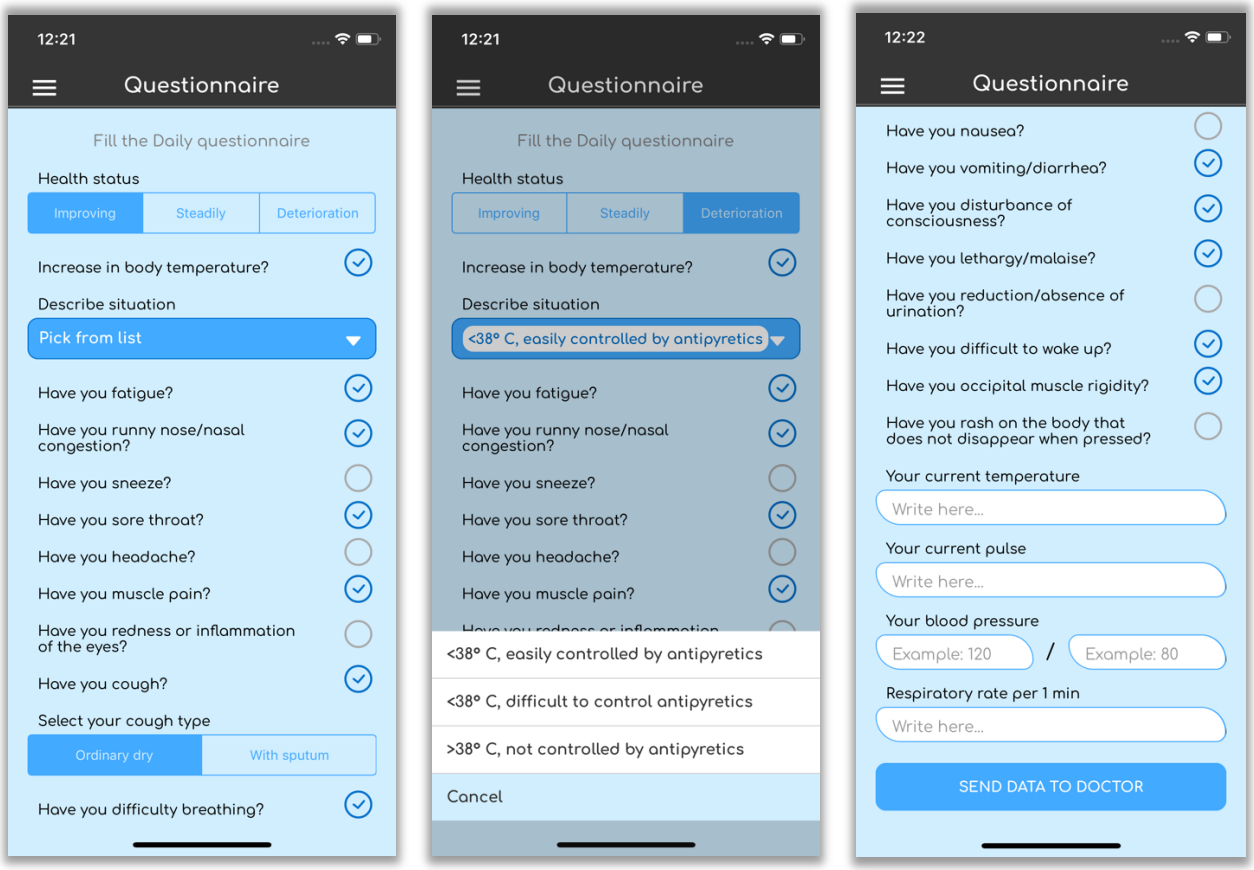

Рисунок 3.13 – Екран щоденного опитування пацієнта

ВИСНОВКИ

Теперішня ситуація як у світі так і в Україні диктує свої правила щодо знаходження в публічних місцях. Все через хворобу COVID-19, яка до сих пір поширюється у всьому світі. Все більше людей страждає від неї, тому потрібно приймати певні запобіжні заходи для відстеження хворих і допомагати їм.

Саме цю проблему вирішує інформаційна система для відстеження хворих на COVID-19. За допомогою неї можна відстежувати хворих, слідкувати за їх станом здоров’я, так приймати щоденні анкети від них з оновленими даними щодо їх самопочуття.

Пацієнт в свою чергу заповнює щоденну анкету, слідкує за своїм здоров’ям, у будь-який час може викликати лікаря, або почитати інформацію про хворобу та заходи запобігання неї. Мобільний додаток на популярні мобільні операційні системи повинен вирішити більшість питань у комунікації між хворими та лікарями, та сприяти зменшенню кількості захворювань, швидкому одужанню, та , у екстрених випадках, швидкій транспортації тяжко-хворих до лікарень.

Через те, що смартфони є майже у кожного, навіть у бабусь та дідусів, які відносяться до груп підвищеного ризику, то інформаційна система у вигляді мобільного додатку є дуже актуальною, а простий дизайн, інтерфейс користувача, та UX повинні сприйматися користувачами будь якого віку дуже просто та легко.

СПИСОК ЛІТЕРАТУРИ

1 Smile-Ukraine [Електроний ресурс] – 2018. – Режим доступу: smile-ukraine.com/ua/mobile-apps/mobile-apps-types – Назва з екрану.

2 Ресурс для IT-спеціалістів [Електроний ресурс] – 2018. – Режим доступу: www.habr.com – Назва з екрану.

3 ESET [Електроний ресурс] – 2020. – Режим доступу: eset.ua/ru/blog/view/80– Назва з екрану.

4 Wikipedia [Електроний ресурс] – 2020. – Режим доступу: ru.wikipedia.org/wiki/MongoDB – Назва з екрану.

5 Documentation site of ReactJS [Electronic resourse] – 2018. – Mode of access: la.kpi.ua/jspui/bitstream/123456789/26972/11/Amirov_magistr.docx – Screen title.

6 React Native Docs [Електроний ресурс] – 2020. – Режим доступу: reactnative.dev/ – Назва з екрану.

7 Flutter Docs [Електроний ресурс] – 2020. – Режим доступу: flutter.dev/– Назва з екрану.

8 Cordova [Електроний ресурс] – 2020. – Режим доступу: cordova.apache.org/– Назва з екрану.

9 Webtec [Електроний ресурс] – 2020. – Режим доступу: webtec.com.ua/uk/articles/index/view/2011-05-05/web-site – Назва з екрану.

10 Apple iOS [Electronic resourse] – 2018. – .apple.com/ru/ios/ios-14/ – Screen title.

ДОДАТОК А

Лістинг файлу залежностей package.json:

```
{
  "name": "RN_Covid",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "android": "react-native run-android",
    "ios": "react-native run-ios",
    "start": "react-native start",
    "test": "jest",
    "lint": "eslint . --ext .js,.jsx,.ts,.tsx",
    "createDebugAPK": "cd android && ./gradlew assembleRelease &&
cd .. && open ./android/app/build/outputs/apk/release",
    "cleanAndroidFolder": "cd android && ./gradlew clean && cd ..",
    "cleanAndroidBuild": "cd android && ./gradlew cleanBuildCache
&& cd ..",
    "createCleanDebugAPK": "cd android && ./gradlew clean &&
./gradlew assembleRelease && cd .. && open
./android/app/build/outputs/apk/release"
  },
  "dependencies": {
    "@babel/plugin-proposal-class-properties": "^7.10.4",
    "@babel/plugin-proposal-decorators": "^7.10.5",
    "@babel/plugin-transform-flow-strip-types": "^7.10.4",
    "@react-native-community/checkbox": "^0.5.5",
    "@react-native-community/datetimepicker": "^3.0.3",
    "@react-native-community/masked-view": "^0.1.10",
    "@react-native-community/segmented-control": "^2.1.2",
    "@react-navigation/drawer": "^5.9.2",
    "@react-navigation/native": "^5.7.5",
    "@react-navigation/stack": "^5.9.2",
    "@zhigamovsky/styled-console-log": "^1.0.0",
    "axios": "^0.21.0",
    "mobx": "^6.0.1",
    "mobx-react": "^7.0.0",
    "react": "16.13.1",
    "react-native": "0.63.2",
    "react-native-collapsible": "^1.5.3",
    "react-native-elements": "^2.3.2",
    "react-native-gesture-handler": "^1.8.0",
```

```
    "react-native-keyboard-aware-scroll-view": "^0.9.3",
    "react-native-modal": "^11.5.6",
    "react-native-modal-datetime-picker": "^9.0.0",
    "react-native-reanimated": "^1.13.1",
    "react-native-safe-area-context": "^3.1.8",
    "react-native-screens": "^2.11.0",
    "react-native-vector-icons": "^7.1.0",
    "styled-components": "^5.2.0"
  },
  "devDependencies": {
    "@babel/core": "^7.8.4",
    "@babel/runtime": "^7.8.4",
    "@react-native-community/eslint-config": "^1.1.0",
    "@types/jest": "^25.2.3",
    "@types/react-native": "^0.63.2",
    "@types/react-test-renderer": "^16.9.2",
    "@types/styled-components": "^5.1.3",
    "@typescript-eslint/eslint-plugin": "^2.27.0",
    "@typescript-eslint/parser": "^2.27.0",
    "babel-jest": "^25.1.0",
    "babel-plugin-transform-decorators-legacy": "^1.3.5",
    "eslint": "^6.5.1",
    "jest": "^25.1.0",
    "metro-react-native-babel-preset": "^0.59.0",
    "react-test-renderer": "16.13.1",
    "typescript": "^3.8.3"
  },
  "jest": {
    "preset": "react-native",
    "moduleFileExtensions": [
      "ts",
      "tsx",
      "js",
      "jsx",
      "json",
      "node"
    ]
  }
}
```

ДОДАТОК Б

Лістинг файлу навігації navigation.patient-drawer.tsx:

```
import React from 'react'
import { createDrawerNavigator } from "@react-navigation/drawer"

import { NavElement } from "../types/navigation"
import { NavTree } from "../config/navigation-tree"
import { PatientGeneralStackNavigator } from './patient-general-
stack'
import { PatientQuestionnaireStackNavigator } from './patient-
questionnaire-stack'
import AboutCovidScreen from
'../screens/PatientAboutCovid/AboutCovid'
import { DrawerContent } from '../components/nav-
components/DrawerContent'
import { EAccountTypes } from '../config/enums/enum.account.types'

// import { DrawerContent } from '../components/nav-components'

const DrawerNavigator = createDrawerNavigator();

export const DrawerStackNavigator: React.FC<NavElement> = ({
  navigation
}) => {
  return (
    <DrawerNavigator.Navigator
      drawerContent={props => (
        <DrawerContent
          {...props}
          accountType={EAccountTypes.Patient}
        />
      )}
      hideStatusBar={false}

initialRouteName={NavTree.P__DrawerBranch.GeneralStackRudiment.path
}
    >
      <DrawerNavigator.Screen
        name={NavTree.P__DrawerBranch.GeneralStackRudiment.path}
        component={PatientGeneralStackNavigator}
      />
```

```
      <DrawerNavigator.Screen

name={NavTree.P__DrawerBranch.QuestionnaireStackRudiment.path}
        component={PatientQuestionnaireStackNavigator}
      />
      <DrawerNavigator.Screen
        name={NavTree.P__DrawerBranch.AboutCOVIDFetus.path}
        component={AboutCovidScreen}
      />
    </DrawerNavigator.Navigator>
  )
}
export type NavigationVariants =
  'stack' |
  'tabs' |
  'drawer' |
  'root'

export interface NavigationFetus {
  title: string,
  path: string
}

export interface NavigationRudiment extends NavigationFetus {
  type?: NavigationVariants,
  child?: NavigationFetus['path']
}

export interface NavigationTree {
  Root: {
    AuthorizatinRudiment: NavigationRudiment
  },
  AuthorizationBranch: {
    LoaderFetus: NavigationFetus,
    SwitcherFetus: NavigationFetus,
    AuthorizationFetus: NavigationFetus,
    D__RegistrationFetus: NavigationFetus,
    P__RegistrationFetus: NavigationFetus,
    D__Drawer: NavigationRudiment,
    P__Drawer: NavigationRudiment
  },
  D__DrawerBranch: {
    GeneralStackRudiment: NavigationRudiment,
```

```
  },
  P__DrawerBranch: {
    GeneralStackRudiment: NavigationRudiment,
    QuestionnaireStackRudiment: NavigationRudiment,
    AboutCOVIDFetus: NavigationFetus
  },
  D__GeneralBranch: {
    GeneralFetus: NavigationFetus
  },
  P__GeneralBranch: {
    GeneralFetus: NavigationFetus
  },
  P__QuestionnaireBranch: {
    QuestionnaireFetus: NavigationFetus
  }
}
```

ДОДАТОК В

Лістинг файлу httpContext.tsx:

```
import React, { createContext, useContext } from 'react';
import Axios, { AxiosError } from 'axios'

import { Log } from '../utils/log';
import * as Context from './'
import {
  TAxiosContext,
  TAxiosContextActions,
  TElementaryContextProvider,
  TAxiosMakeFetchMethod,
  TAxiosDefaultRequestOptions,
  TErrorConstructor,
  TErrorTranslater
} from '../types/contexts/axios';
import { QUERIES_URL } from '../config/rest-config';
import { ERROR_TRANSLATER } from './context-utils'
import { IUser } from '../types/contexts/user';
import { TCartCredentials } from '../types/contexts/cart';
import LanguageContext from './LanguageContext';
import { Translater } from '../config/Translater';

// @ts-ignore
const AxiosContext: TAxiosContext = createContext({ actions: {} })
export default AxiosContext;

export const AxiosContextProvider: TElementaryContextProvider =
({children}) => {
  const {language} = useContext(LanguageContext)
  const { actions: loaderActions } =
useContext(Context.LoaderContext)
  const { actions: alertActions } =
useContext(Context.AlertContext)
  const { actions: configActions } =
useContext(Context.ConfigContext)
  const { user, actions: userActions } =
useContext(Context.UserContext)
  const { deviceID } = useContext(Context.PushContext)
  const { cart, cartCredentials, actions: cartActions} =
useContext(Context.CartContext)
```

```
const actions = {
  SEND_CODE: async request => {
      return new Promise(async (resolve, reject) => {
        let { phone } = request.data;
        try {
          let response = await MAKE_FETCH({
            url: QUERIES_URL.login,
            method: 'POST',
            data: {
              phone
            }
          }, request.config, {
            identifier: 'SEND_CODE'
          });
          resolve(response)
        }
        catch(e) {
          reject(e);
        }
      })
    },
    VERIFY_CODE: async request => {
      return new Promise(async (resolve, reject) => {
        let { phone, code } = request.data;
        try {
          let response = await MAKE_FETCH({
            url: QUERIES_URL.verify_code,
            method: 'POST',
            data: { phone, code, player_id: deviceID },
          }, request.config, {
            identifier: 'VERIFY_CODE'
          });
          await userActions.UPDATE_USER(response.data);
          if(cart?.items.length && cartCredentials){
            await actions.ATTACH_UNREGISTERED_CART_TO_USER({
              data: {
                user: {
                  token: (response.data as IUser).token
                },
                cart: (cartCredentials as TCartCredentials)
              }
            })
```

```
          }
          await cartActions.CLEAR_CART_CREDENTIALS();
          await actions.GET_CART({
            data: { currentUser: response.data }
          })
          resolve(response);
        }
        catch(e) { reject(e) }
      })
    },
    GET_USER: async request => {
      return new Promise(async (resolve, reject) => {
        try {
          let response = await MAKE_FETCH({
            url: QUERIES_URL.profile,
            method: 'GET',
            headers: {
              'Authorization': request.option?.token || user?.token
|| 'without-token'
            }
          }, request.config, { identifier: 'GET_USER' });
          await userActions.MERGE_USER(response.data)
          resolve(response);
        }
        catch(e) { reject(e) }
      })
    },
    UPDATE_USER: async request => {
      return new Promise(async (resolve, reject) => {
        let { name } = request.data;
        try {
          let response = await MAKE_FETCH({
            url: QUERIES_URL.profile,
            method: 'POST',
            data: {
              name
            },
            headers: {
              'Authorization': user?.token || 'query-without-token'
            }
          }, request.config, {
            identifier: 'UPDATE_USER'
          });
```

```
          await userActions.MERGE_USER(response.data)
          resolve(response);
        }
        catch(e) {
          reject(e);
        }
      })
    },
    LOGOUT_USER: async request => {
      return new Promise(async(resolve, reject) => {
        try {
          /* Send query to logout User */
          let response = await MAKE_FETCH({
            url: QUERIES_URL.logout,
            method: 'POST',
            headers: { 'Authorization': user?.token || 'query-
without-token'  },
            data: {
              player_id: deviceID
            }
          }, request.config, { identifier: 'LOGOUT' })
          await userActions.CLEAR_USER();
          await actions.GET_CART({});
          resolve(response);
        }
        catch(e) {
          reject(e);
        }
      });
    },
}

  return (
    <AxiosContext.Provider value={{actions}}>
      { children }
    </AxiosContext.Provider>
  )
}
```

ДОДАТОК Г

Лістинг файлу авторизації auth.ts на стороні back-end:

```
import React, { createContext, useContext } from 'react';
import Axios, { AxiosError } from 'axios'
import { Router, json } from 'express'
import bcrypt from 'bcrypt'
import user from '../models/user'
import UserModel from '../models/user'
const isValid = require('isvalid')
const app = Router()
app.post('/registration', json(), isValid.validate.body({
  login: {
    type: String,
    required: true,
    len: '2-'
  },
  password: {
    type: String,
    required: true,
    len: '2-'
  },
  fullname: {
    type: String,
    required: true,
    len: '2-'
  },
}), async (req, res, next) => {


  const User = new UserModel(req.body)

  try {
    await User.save()
    res.send(User)
  }
  catch(e) {
    next(e)
  }
})

app.post('/login', json(), isValid.validate.body({
```

```
  login: {
    type: String,
    required: true,
    len: '2-'
  },
  password: {
    type: String,
    required: true,
    len: '2-'
  },
}), async (req, res, next) => {

  try {
    const user = await UserModel.findOne({
      login: req.body.login
    })

    if(user && await bcrypt.compare(req.body.password,
user.password)) {
      res.send(user)
    }
    else {
      res.status(400).send('Fuck you')
    }
  }
  catch(e) {
    next(e)
  }
})


export default app
```