Федеральное государственное образовательное бюджетное
учреждение высшего образования
**«Финансовый университет при Правительстве
Российской Федерации»**

Департамент анализа данных и машинного обучения

Курсовая работа по дисциплине
«Технологии разработки приложений для мобильных устройств»
на тему;

«Приложение для прохождения функциональной пробы человека (проба Мартине)»

Выполнила:
студентка группы ПИ20-4
Еркебаева А. Д.

Научный руководитель:
доцент, канд. технических наук
Абашин В. Г.

Москва
2022

Оглавление

1. Введение 3
2. Постановка задачи 5
3. Описание и актуальность предметной области 7
4. Описание интерфейса приложения 9
   4.1. Главный экран 10
   4.2. Экран с информацией 11
   4.3. Экран тестирования 12
   4.4. Результирующий экран 17
5. Отправление запроса 22
6. Структура приложения 24
7. Заключение 27
8. Список использованной литературы 28
9. Приложение 29
   9.1. Приложение A. App.js 29
   9.2. Приложение B. /components/Navigator.js 29
   9.3. Приложение C. /components/Home.js 29
   9.4. Приложение D. /components/Info.js 30
   9.5. Приложение E. /components/Test.js 31
   9.6. Приложение F /components/Result.js 37

1. Введение

Поддерживать здоровый образ жизни, который включает в себя тренировки, правильное питание с контролем калорий, своевременные походы к врачу сложнее, чем кажется на первый взгляд. Именно поэтому в наше время популярны приложения, которые позволяют следить за ключевыми показателями здоровья, отслеживать прием лекарств, также такие приложения служат не просто как онлайн-календари или онлайн-дневники, они могут анализировать данные о нашем организме и предупреждать, если что-то идет не так. Например, приложения для сна могут отслеживать, когда была глубокая фаза сна или же у вас недостаток сна, из-за чего могут быть серьезные проблемы с организмом. Однако не стоит полностью доверять данным приложениям, так как организм каждого человека индивидуален. Требуется обследование специалиста для назначения и последующего лечения или профилактики.

Одними из самых популярных приложений для здоровья можно считать Apple Health для iOS и Samsung Health для Android. У них примерно одинаковый функционал, который позволяет следить за количеством шагов, пульсом, фазами сна и вести календари и дневники, а также отображать различные статистики по полученным данным.

Важнейшим условием для физических нагрузок является строгий контроль за функциональным состоянием организма. Для правильного планирования нагрузок необходимы измерения и оценки различных органов и систем, для которых и разработаны функциональные пробы. Их особенностью является использование раздражающих факторов для их прохождения, например, физические нагрузки, задержка дыхания, для измерения показателей до, во время и после проведения пробы для оценки общего состояния организма или же определенных систем.

Как работает проба Мартине, какие данные для нее нужны и какие результаты можно получить? С помощью каких методов и алгоритмов можно

отправлять данные на сервер? Именно эти вопросы и будут рассмотрены в курсовой работе.

Для демонстрации будет разработан прототип с помощью кроссплатформенного фреймворка React Native на языках программирования JavaScript для реализации прохождения функциональной пробы человека (пробы Мартине). Для отправления запроса на сервер используется интерфейс Fetch API с методом POST.

2. Постановка задачи

В соответствии с выбранной темой следует разработать приложение на React Native на платформе Expo при помощи языка программирования JavaScript с использованием Fetch API.

Со стороны клиента необходимо разработать несколько окон и логику переходов между ними с помощью навигации, а также дизайн с комфортной для глаз цветовой гаммой и удобное расположение элементов для упрощения и ускорения взаимодействия пользователя с ними.

Для отправки запроса используется интерфейс Fetch API, в который передаются: URL, HTTP метод, заголовки и тело запроса в виде строки данных, полученных от пользователя, таких как: дата рождения, пол, пульс до физических нагрузок, количество минут, через сколько пульс вернулся в норму.

Для отслеживания времени необходимо реализовать таймер, который можно запустить, приостановить и очистить, чтобы использовать заново.

Для того, чтобы пользователь понимал, что он уже заполнил, а что нет, значки при выборе пола, как и обводка полей для ввода значений должны менять цвет. Также необходимо ограничить значения, допустимые для поля для ввода, чтобы пользователь мог вводить только числа, а при выборе пола можно выбрать только один, если нажать на другой, первый автоматически меняется цвет на дефолтный.

Приложение не должно отправлять на сервер пустые данные, для этого необходимо, чтобы при нажатии кнопки завершения, проверялась заполненность всех полей, иначе выводить предупреждение.

При отправке на сервер некорректных данных, необходимо все равно переключаться на результирующий экран и выводить сообщение, что данные были ошибочными. В ином случае, если данные корректны, то сервер отправляет ответ 3 видов: плохо, средне, хорошо. В зависимости от ответа

сервера в результирующем окне выбираются соответствующие данные: изображение и надпись, разные для каждого вида.

3. Описание и актуальность предметной области

Предметной областью является функциональная проба, а конкретнее проба Мартине. Данная проба предназначена для людей с небольшим уровнем физической нагрузки, например, для начинающих спортсменов. Для изначальных данных, обследуемый должен сидеть в спокойной расслабленной позе в течение 5 минут, не разговаривая и желательно не двигаясь, после этого необходимо измерить его пульс, это и будет первым показателем – пульс в состоянии покоя. Далее необходимо выполнить 20 глубоких приседаний в спокойном темпе, вытягивая руки перед собой. Сразу после приседаний необходимо сесть обратно на стул и сразу начать отсчет до тех пор, пока пульс не будет равен пульсу в состоянии покоя. Количество минут, за которые пульс пришел в норму, будет вторым показателем. Отклонения в ту или иную сторону могут означать проблемы с сердечно-сосудистой системой.

Актуальность приложения состоит в том, что функциональные пробы являются достаточно быстрым и несложным способ проверить работоспособность организма при возмущающих факторах, оценить его общее состояние, проверить адаптацию к различным физическим нагрузкам. В настоящее время, когда по всему миру распространена коронавирусная инфекция COVID-19, стоит особо тщательно следить за состоянием здоровья и хотя бы раз в несколько дней проводить функциональные пробы.

Разработать приложение необходимо с помощью Expo, платформы с открытым исходным кодом, которая облегчает работу над кроссплатформенными приложениями для iOS и Android.

Expo SDK представляет собой серию встроенных библиотек для каждой платформы iOS и Android, и он позволяет JavaScript получать доступ к системным функциям устройства, таким как камера, push-уведомления, локальное хранилище, контакты и другие аппаратные и операционные системные API.

Главными плюсами Expo являются уже готовая реализация некоторых нативных компонентов, что позволяет значительно ускорить разработку приложения, а также тестирование приложения прямо в этом окне, либо на своем смартфоне, то есть мы можем сразу проверить совместимости библиотек, расположение и работоспособность элементов.

Однако есть существенные минусы Expo, например, Expo не может работать в фоновом режиме, то есть мы не можем запустить приложение и закрыть его, чтобы оно продолжало работать и отслеживать информацию дальше. А также то, что Expo ограничен только нативными API, содержащимися в Expo SDK.

4. Описание интерфейса приложения

В проектировании интерфейса есть несколько важных пунктов, о которых не стоит забывать:

- Экран приложения не должен казаться пользователю нагруженным, если такое происходит лучше перенести что-то на другое окно;
- Цветовая гамма приложения должна быть соблюдена, а текст должен быть легкочитаем;
- Пользователь, смотря на приложение, должен интуитивно понимать, что и как работает, либо стоит оставить подсказки.

Одним из наиболее важных аспектов мобильного приложения является наличие разных экранов и возможность перемещаться между ними, причем каждый экран служит для своей цели. В React Native для этого используется библиотека React Navigation. В ней есть несколько видов навигации, однако в данной работе использован только StackNavigator, где приложение выталкивает и извлекает элементы из стека навигации, когда пользователи взаимодействуют с ним, и это приводит к тому, что пользователь видит разные экраны.

В процессе выполнения работы были использованы различные компоненты React Native, такие как: View, Text, StyleSheet, Image, Alert, ScrollView, TouchableOpacity, TextInput, Platform.

Компонент Platform применяется для проверки того, на какой платформе работает наше устройство, в основном это iOS и Android, но также бывает, например, Windows. Stylesheet для того, чтобы объединить разные стили в одной переменной, как например файл css для html, и далее не прописывать для одинаковых компонентов все параметры стиля, а просто применить один из стилей из этой переменной. View – контейнер, в котором и хранится весь вид программы. ScrollView – выполняет те же функции, что и View, только

позволяется прокручивать экран вверх/вниз, что удобно для телефонов с небольшим экраном. Text – компонент, который позволяет отображать текстовые данные на экране, которые можно стилизовать с помощью параметра style. Image используется для отображения изображений. TouchableOpacity – компонент, который является оболочкой, который реагирует на касание и затемняется, в данном проекте он работает как кнопка. TextInput – компонент для ввода текстовых данных, в данной курсовой работе использован для введения пульса и количества минут и ограничен только вводом числовых значений. Alert используется для вывода на экран уведомления.

В отдельном файле Navigator.js прописаны все окна, между которыми возможна навигация:

- Главный экран
- Экран с информацией
- Экран тестирования
- Результирующий экран

4.1. Главный экран

На главном экране отображено название теста, кнопка для перехода на окно для ознакомления с информацией, а также изображение для более приятного вида. Интерфейс представлен на рисунке 1 ниже.

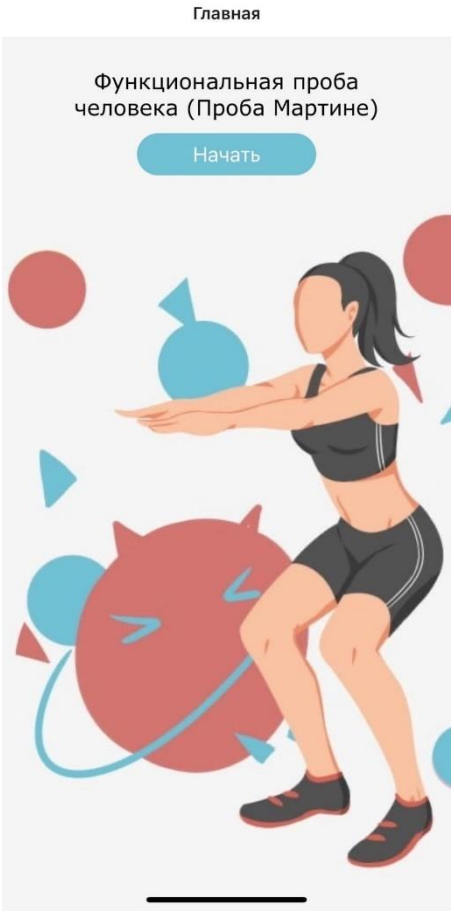

Рисунок 1 – Главный экран

Для кнопки использован TouchableOpacity вместо Button для более приятного визуального представления.

4.2. Экран с информацией

На экране представлена основная информация по тесту, также сверху находится кнопка, нажав на которую можно вернуться к предыдущему окну, чтобы перейти к самому тесту необходимо нажать кнопку «Начать». Интерфейс представлен на рисунке 2 ниже.

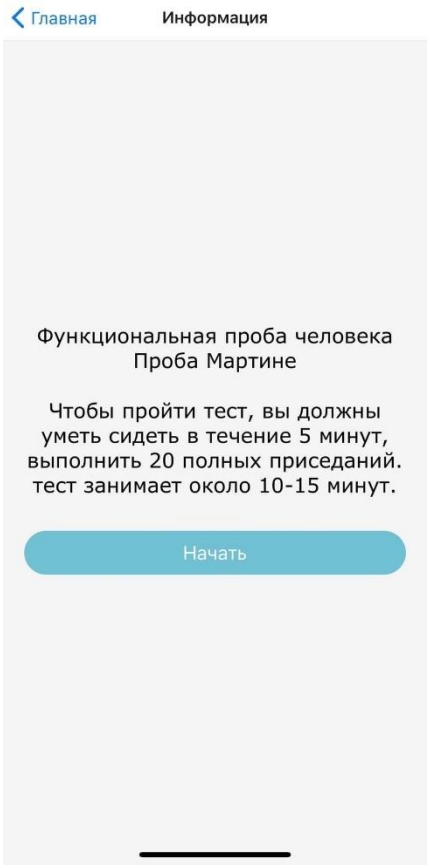

Рисунок 2 – Экран с информаицей

Здесь также использован TouchableOpacity для более приятного визуального представления.

4.3. Экран тестирования

На данном окне расположен сам тест, расписанный по пунктам. Сверху также расположена кнопка, при нажатии на которую можно вернуться к предыдущему окну. Для удобства и совместимости с разными экрана добавлен ScrollView. При возвращении на предыдущее окно уже введенные данные будут потеряны и их придется вводить заново. При нажатии на кнопку Завершить будет отправлен HTTP запрос, как это происходит подробно описано в главе «Отправление запроса». Интерфейс представлен на рисунке 3 ниже.

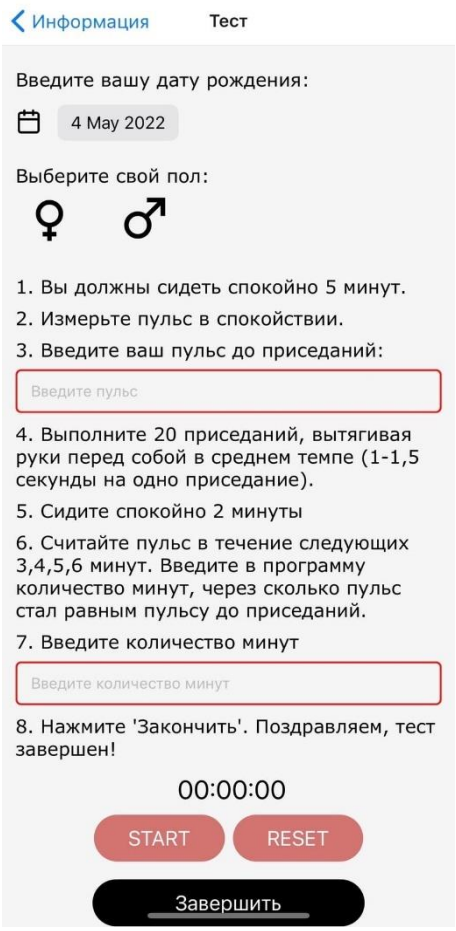

Рисунок 3 – Экран тестирования

С самого начала теста спрашивается дата рождения, чтобы далее подсчитать возраст. Для выбора даты установлена библиотека React Native DateTimePicker, который позволяет использовать компонент DateTimePicker. Изначально в нем стоит сегодняшняя дата, но открыв мы можем ее легко поменять на любую кроме будущей. При выборе сегодняшняя дата изменяется на указанную. Для компонента стоит ограничение на максимальную дату сегодняшним днем из-за этого мы не можем выбрать, например, завтра, что показано на рисунке 4 ниже.

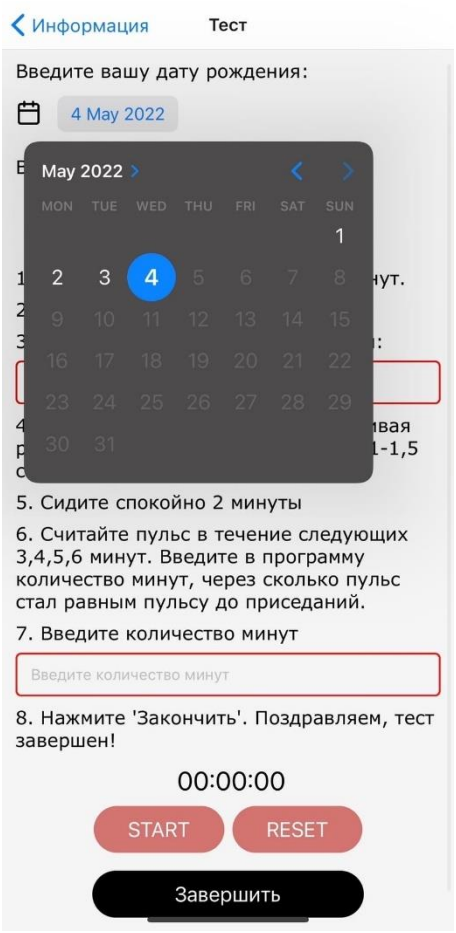

Рисунок 4 – Выбор даты

Рядом с компонентом для выбора даты также находится иконка календаря, которая добавлена с помощью библиотеки React Native Vector Icons.

Далее следует выбрать пол, для этого нужно нажать на один из значков, которые являются IconButton и добавлены с помощью библиотеки React Native Paper. Изначально оба значка неактивны и имеют черный цвет, но если нажать, то значок для девушки меняет цвет на розовый, а для мужчины на голубой. Для них стоит ограничение, что нельзя, чтобы были активны сразу обе кнопки, при нажатии на другую, предыдущая становится неактивной. Данный элемент представлен на рисунках 5 и 6 ниже.

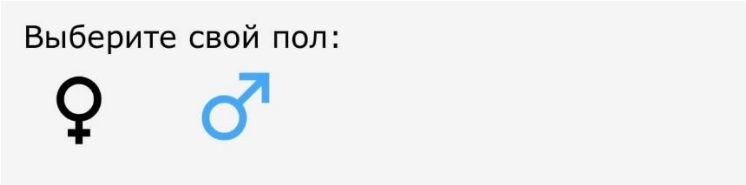

Рисунок 5 – Выбран мужской пол

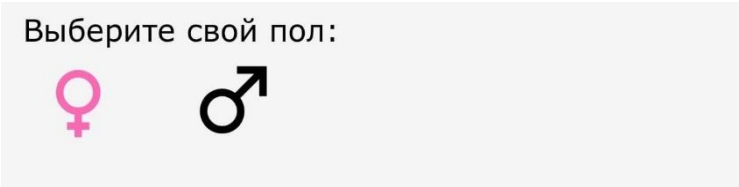

Рисунок 6 – Выбран женский пол

Выполнить пункт 1 может помочь таймер, находящийся снизу страницы, изображенный на рисунке 7 ниже.

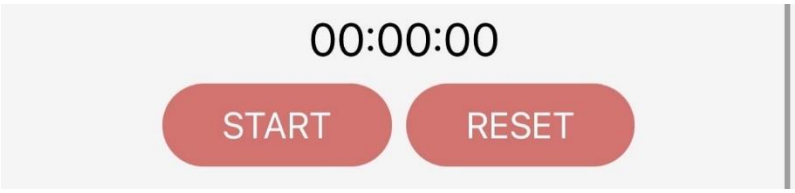

Рисунок 7 – Таймер

Таймер реализован с помощью библиотеки React Native Stopwatch Timer. Сверху время, которое изменяется с помощью хуков, а снизу расположены 2 кнопки, которые оформлены с помощью TouchableOpacity. Кнопка «Reset» обнуляет таймер, а кнопка «Start» его запускает. При нажатии на кнопку “Start”, она меняет свое название на «Stop», что показано на рисунке 8 ниже.

Рисунок 8 – Таймер запущен

В пунктах 3 и 7 нам нужно ввести числовые значения пульса и количества минут, для чего мы используем TextInput с параметрами returnKeyType=«done» для того, чтобы можно было свернуть клавиатуру

нажав либо “Done”, либо нажав куда-угодно, кроме клавиатуры, и keyboardType=«number-pad», что позволяет ограничить ввод пользователя только числами. При вводе каких-либо данных обводка меняется на зеленую, что означает что данные поле не пустое. Интерфейс компонента представлен на рисунке 9 ниже.

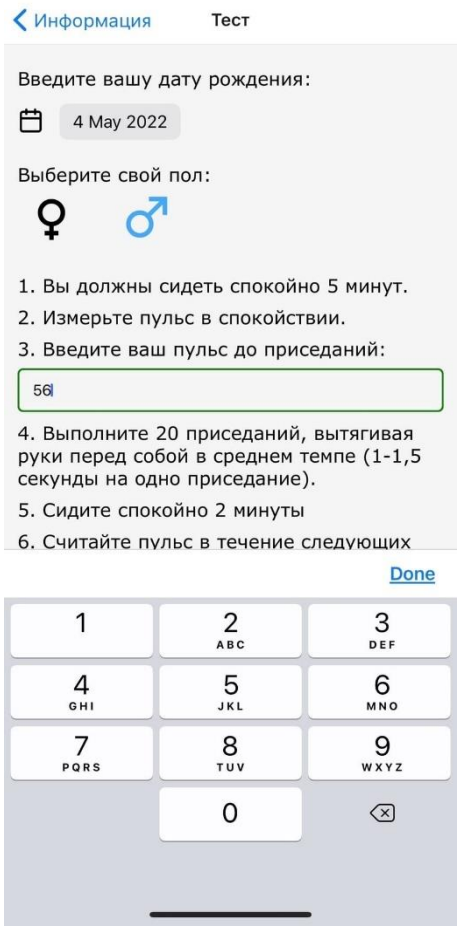

Рисунок 9 – Ввод данных

В конце находится кнопка «Завершить», нажав на которую данные, полученные от пользователя, отправляются на сервер, а ответ, пришедший с сервера, посылается в результирующее окно для вывода итоговых данных, но если введены не все данные, то пользователь получит предупреждение, изображенное на рисунке 10 ниже.

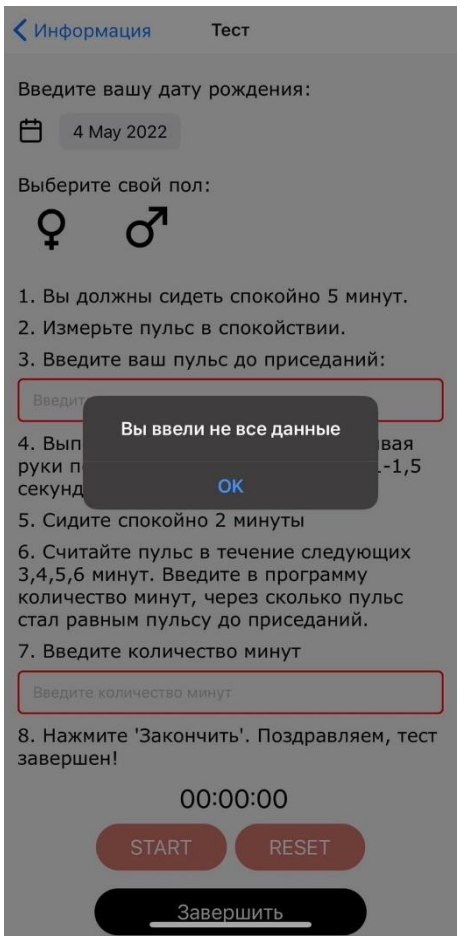

Рисунок 10 – Предупреждение

4.4. Результирующий экран

На данном экране данные компонентов меняются в зависимости от результатов, а точнее меняется надпись и изображение, заголовок и важная информация о посещении врача остается неизменной. Какой будет результат зависит от сервера, на который мы посылали на предыдущем экране данные и откуда получили ответ для результата, который мы в последствии парсили. Всего сервер может вернуть 4 варианта. Вариант для хорошего результата представлен на рисунке 11 ниже.

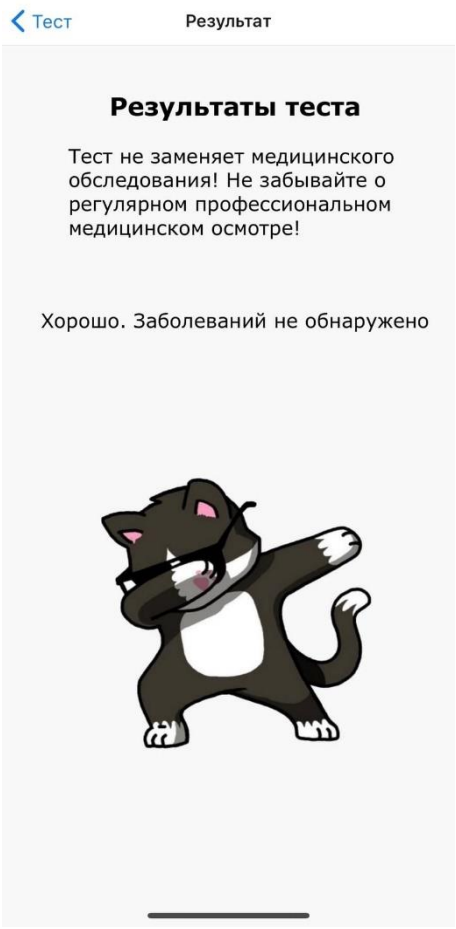

Рисунок 11 – Хороший результат

Ответ сервера с экрана тестирования, как и получается, так и отправляется с помощью библиотеки React Navigation. Получаем данные на экране результата с помощью route.

Также можно заметить, что в зависимости от результата меняется итоговое изображение, для этого изначально мы их импортируем в Result.js, а затем проверяя, что содержится в ответе, который мы до этого парсили, меняем изображение на то, которое нам нужно. Изменение изображения мы можем увидеть на рисунке 12 ниже.

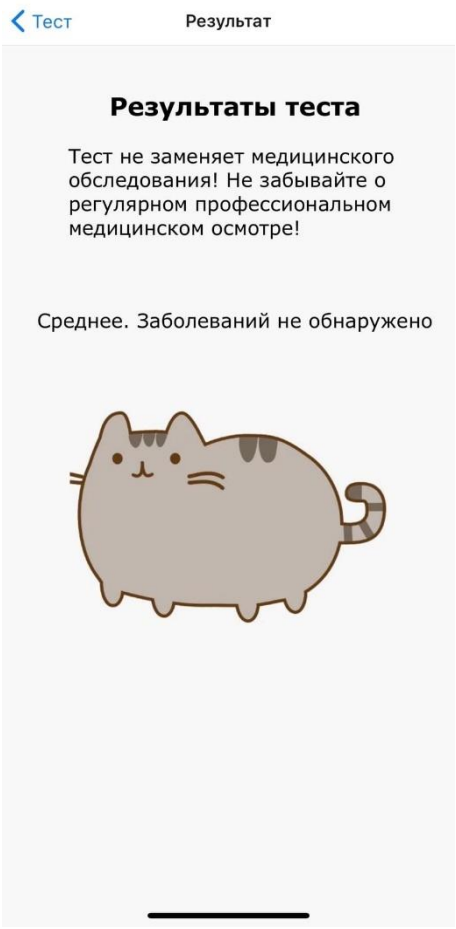

Рисунок 12 – Средний результат

Для того, чтобы парсить данные, мы изначально разделяем на несколько компонентов массив и проверяем чтобы в определенном элементе массива содержалась буква «Е», которая будет означать, что это ошибка и дальше проверять нет смысла. Далее мы берем уже другой элемент массива и проверяем, если там есть «П», то это плохой результат, если «Х» - хороший, а если «С» - средний. На рисунке 13 ниже можно увидеть, что в отличие от предыдущего рисунка 12 надпись в результате другая.

Рисунок 13 – Плохой результат

Сервер может вернуть в ответе ошибку, если пользователь ввел некорректные данные, например слишком маленькое или наоборот слишком большое значение для пульса или количества минут, либо забыл поменять изначальную дату на свою дату рождения. В этом случае сервер не может ничего посчитать и отправляет обратно ошибку, что можно увидеть на рисунке 14 ниже.

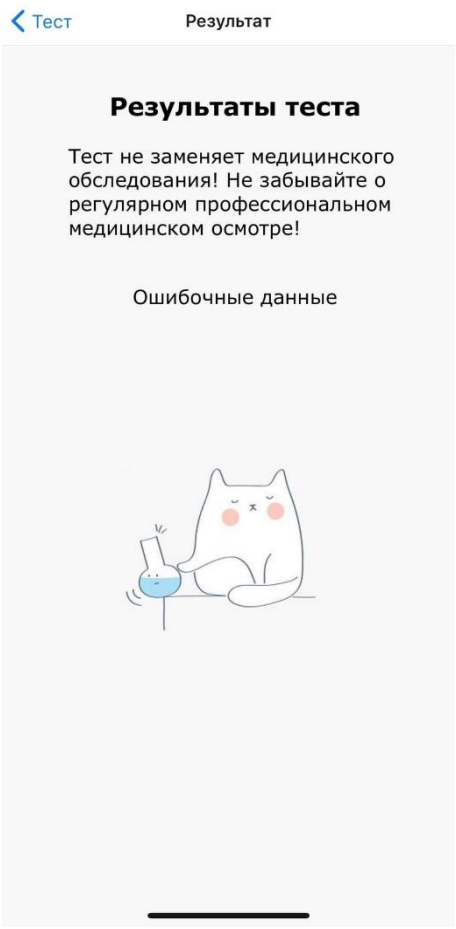

Рисунок 14 – Результат с ошибочными данными

5. Отправление запроса

В какой-то момент любое мобильное приложение может потребовать загрузить ресурсы с удаленного URL-адреса, самым распространенным случаем являются звонки по сети.

В данной курсовой работе для отправки и получения данных с сервера используется Fetch API с методом POST. Fetch API предоставляет интерфейс JavaScript, который может асинхронно извлекать ресурсы по сети. Он очень похож на XMLHttpRequest, который используется для той же цели. Но fetch API является мощным и гибким.

Для того, чтобы запросить данные нужно использовать метод fetch(), который первым обязательным аргументом принимает путь, с которого и требуется получить данные, а вторым могут быть дополнительные опции, такие как: заголовки, тело запроса и метод.

Телом запроса являют данные о дне рождении пользователя, его поле, показатели пульса и количества минут.

Заголовки для запроса, используемые в данной курсовой работе:

- Host;
- Connection, сведения о соединении;
- Content-Type, способ представления сущности;
- Accept, список допустимых форматов;
- Accept-Language, список поддерживаемых языков;
- Cache-Control, для управления кэшированием.

В приложении для более эффективной его работы Fetch API реализован со специальным синтаксисом async/await, который позволяется не получать ответ, пока он полностью не пришел с сервера. А затем используется .then, чтобы записать ответ и отправить на следующий экран.

Главным плюсом Fetch API является он возвращает Promise, при котором запрос все равно будет обрабатываться в обычном режиме и не будет отклонен из-за ошибки, потому что всегда устанавливается статус 200.

6. Структура приложения

Для более удобной работы с состояниями и методами жизненного цикла в приложении используются хуки. Весь функционал реализован с помощью функций, без классов, так как классы чаще всего в больших проектах становятся нечитаемыми, ибо хранят в одном методе разную логику. В одном месте может быть одновременно и задание стиля, и смена надписи, и подключение к серверу.

Для того, чтобы было удобнее и понятнее работать с приложением в дальнейшем, у каждого экрана есть свой файл, в котором прописан его функционал, также есть отдельный файл для навигации и основной файл App.js, который запускает всю программу.

Файл App.js

- App – определяющая функция, является главной в проекте, в ней содержится Navigator Container для реализации навигации между окнами.

Файл Navigator.js

- Navigate – функция, в которой прописана навигация для приложения.

Файл Home.js

- Home – функция для отображения главного экрана приложения.

Файл Info.js

- Info – функция для отображения информации для ознакомления перед прохождением теста.

Файл Test.js

- Test – основная функция экрана с тестированием, которая и отображает весь тест;

– onChange – функция для DateTimePicker, которая меняет отображаемую дату на выбранную пользователем;
– onClickBtnF – меняет цвет значка для женского пола, если значок мужского активен, делает его неактивным;
– onClickBtnM – меняет цвет значка для мужского пола, если значок женского активен, делает его неактивным;
– onChangePulse – меняет цвет обводки на поле ввода пульса, если поле не пустое;
– onChanegTime – меняет цвет обводки на поле ввода количества минут, если поле не пустое;
– setTimer – срабатывает при нажатии кнопки Start, запускает таймер, меняет надпись на кнопке со «Start» на «Stop»;
– getSex – функция, для получения значения 1, соответствующего мужскому полу, и значения 2, соответствующего женскому, для формирования запроса на сервер;
– toResult – функция, которая сначала формирует в виде строки тело запроса, а затем отправляет его на сервер, после чего получает от него ответ, который отправляет на результирующий экран, чтобы вывести итог;
– check – функция срабатывает после нажатия на кнопку завершить в конце экрана, она проверяет все ли поля заполнены, если нет, то вызывает предупреждение, если все правильно, то обращается к функции toResult.

Файл Result.js

– Result – основная функция результирующего экрана, в которой изначально получаются данные, отправленные с экрана тестирования, затем они же парсятся и проверяются на вхождение определенных букв, чтобы подобрать текст и изображение, которые соответствуют полученному результату.

Все изображения, используемые в данной курсовой работе, хранятся в папке images, а все экраны хранятся в папке components для структурированности

7. Заключение

В ходе данной работы было разработано приложение на фреймворке React Native для прохождения функциональной пробы человека, а конкретнее пробы Мартине, для измерения состояния сердечно-сосудистой системы.

Особенностью данной работы является среда разработки, выбранная для этого приложения, Expo для более удобной реализации на фреймворке React Native. Для более приятного визуального представления вместе с каждый из результатов выводятся изображения, различные для каждого вида ответа сервера.

Созданное приложение соответствует и удовлетворяет все поставленные требования и задачи: реализует обработку полученный данных на сервере с помощью Fetch API, отображение нескольких вариантов результирующего экрана в зависимости от ответа сервера, навигация между несколькими экранами приложения.

Решение может модернизироваться и обновляться. Например, можно добавить возможно ведения истории проведения тестов по дням для ведения онлайн-календаря или же просто в виде списка, чтобы пользователь мог в любой момент посмотреть и сравнить полученные результаты. Также можно добавить в приложение возможность выбора других функциональных проб для прохождения.

8. Список использованной литературы

1. Гусев Александр Владимирович, Ившин А. А., Владзимирский А. В. РОССИЙСКИЕ МОБИЛЬНЫЕ ПРИЛОЖЕНИЯ ДЛЯ ЗДОРОВЬЯ: СИСТЕМАТИЧЕСКИЙ ПОИСК В МАГАЗИНАХ ПРИЛОЖЕНИЙ // Журнал телемедицины и электронного здравоохранения. 2021. №3.
2. Функциональные пробы в лечебной и массовой физической культуре: учебное пособие / О. М. Буйкова, Г. И. Булнаева; ФГБОУ ВО ИГМУ Минздрава России, Курс лечебной физкультуры и спортивной медицины, Кафедра физического воспитания – Иркутск: ИГМУ, 2017. – 24 с.
3. Zammetti F. Practical React Native: Build Two Full Projects and One Full Game Using React Native. - 1st ed. изд. - Pottstown, PA: Sprimger, 2018. - 334 с.
4. Байдыбеков А. А., Гильванов Р. Г., Молодкин И. А. СОВРЕМЕННЫЕ ФРЕЙМВОРКИ ДЛЯ РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ // Интеллектуальные технологии на транспорте. 2020. №4 (24).

9. Приложение

9.1. Приложение A. App.js

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import Navigate from './components/Navigator'

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

export default App;
```

9.2. Приложение B. /components/Navigator.js

```
import React from 'react';
import { Text, View } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import Home from './Home'
import Info from './Info'
import Test from './Test'
import Result from './Result'

const AppStack = createStackNavigator();

const Navigate = (props) => {

  return (
    <AppStack.Navigator>
      <AppStack.Screen name="Главная" component={Home}/>
      <AppStack.Screen name="Информация" component={Info}/>
      <AppStack.Screen name="Тест" component={Test}/>
      <AppStack.Screen name="Результат" component={Result}/>
    </AppStack.Navigator>
  );
};

export default Navigate;
```

9.3. Приложение C. /components/Home.js

```
import React, {useState} from 'react';
import { Text, View, StyleSheet, Image, ScrollView, TouchableOpacity } from 'reac
t-native';
import { useNavigation } from '@react-navigation/native';
```

```
const Home = () => {
  const navigation = useNavigation();
  return (
    <ScrollView style={styles.container}>
      <Text style={styles.text}>Функциональная проба человека (Проба Мартине)</Te
xt>
      <TouchableOpacity onPress={() => {navigation.navigate("Информация")}}>
        <View style={styles.button}>
          <Text style={{ color: 'white', padding:8, fontSize: 20 }}>Начать</Text>
        </View>
      </TouchableOpacity>
      <Image source={require('../images/martine.jpg')} style={{width: '140%'}}/>
    </ScrollView>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    backgroundColor: '#f5f5f5'
  },
  button: {
    margin: '3%',
    marginHorizontal: '30%',
    backgroundColor: '#70c1d4',
    alignItems: 'center',
    justifyContent: 'center',
    borderRadius: 30
  },
  text: {
    marginHorizontal: '15%',
    fontFamily: 'Verdana',
    justifyContent: 'center',
    textAlign: 'center',
    fontSize: 21,
    fontWeight: '450',
  },
});

export default Home;
```

9.4. Приложение D. /components/Info.js

```
import React, {useState} from 'react';
import { Text, View, StyleSheet, Image, Button, Alert, ScrollView, SafeAreaView,
TouchableOpacity } from 'react-native';
import { useNavigation } from '@react-navigation/native';

const Info = () => {
  const navigation = useNavigation();
```

```
  return (
    <ScrollView style={styles.container}>
      <Text style={styles.text}>Функциональная проба человека{"\n"}Проба Мартине<
/Text>
      <Text style={styles.text}>Чтобы пройти тест, вы должны уметь сидеть в течен
ие 5 минут, выполнить 20 полных приседаний. тест занимает около 10-
15 минут.</Text>
      <TouchableOpacity onPress={() => {navigation.navigate("Тест")}}>
        <View style={styles.button}>
          <Text style={{ color: 'white', padding:10, fontSize: 20 }}>Начать</Text
>
        </View>
      </TouchableOpacity>
    </ScrollView>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    backgroundColor: '#f5f5f5'
  },
  button: {
    margin: '5%',
    backgroundColor: '#70c1d4',
    alignItems: 'center',
    justifyContent: 'center',
    borderRadius: 30
  },
  text: {
    margin: '3%',
    marginHorizontal: '5%',
    fontFamily: 'Verdana',
    justifyContent: 'center',
    textAlign: 'center',
    fontSize: 21,
    fontWeight: '450',
  },
});

export default Info;
```

9.5. Приложение Е. /components/Test.js

```
import React, {useState} from 'react';
import { Text, View, StyleSheet, Image, Alert, ScrollView, TouchableOpacity, Plat
form, TextInput} from 'react-native';
import { IconButton, Colors } from 'react-native-paper';
import DateTimePicker from '@react-native-community/datetimepicker';
import { useNavigation } from '@react-navigation/native';
```

```
import { Feather } from '@expo/vector-icons';
import { Stopwatch, Timer } from 'react-native-stopwatch-timer';

const Test = () => {
  const [date, setDate] = useState(new Date());
  const [mode, setMode] = useState('date');
  const [colorM, setColorM] = useState(false);
  const [colorF, setColorF] = useState(false);
  const [show, setShow] = useState(false);
  const [pulse, onChangepulse] = useState(null);
  const [min, onChangetime] = useState(null);
  const [border1, setBorder1] = useState(false);
  const [border2, setBorder2] = useState(false);
  const [play, setPlay] = useState(false);
  const [num, setNum] = useState(0)
  const [isStopwatchStart, setIsStopwatchStart] = useState(false);
  const [resetStopwatch, setResetStopwatch] = useState(false);
  const [time, setTime] = useState(0)
  const [sex, setSex] = useState(0)
  const navigation = useNavigation();

  const onChange = (event, selectedDate) => {
    const currentDate = selectedDate || date;
    setShow(Platform.OS === 'ios');
    setDate(currentDate);
  }

  const onClickBtnF = color => {
    setColorF(colorF => !colorF)
    setColorM(false)
  }

  const onClickBtnM = color => {
    setColorM(colorM => !colorM)
    setColorF(false)
  }

  const onChangePulse = (pulse) => {
    onChangepulse(pulse)
    if (pulse==null) {
      setBorder1(false);
    } else if (pulse.length == 0) {
      setBorder1(false);
    } else {
      setBorder1(true);
    }
  }

  const onChangeTime = (min) => {
    onChangetime(min)
```

```
    if (min==null) {
      setBorder2(false);
    } else if (min.length == 0) {
      setBorder2(false);
    } else {
      setBorder2(true);
    }
  }

  const setTimer = (time) => {
    setNum(time)
    setPlay(true)
  }

  const getSex = () => {
    if (colorF==true) { return 2; } else { return 1; }
  }

  const toResult = async () => {
        try {
        const full = 'day='+date.getDate()+'&month='+(date.getMonth()+1)+'&year='
+date.getFullYear()+'&sex='+getSex()+'&m1='+pulse+'&m2='+min
        console.log(full)
        await fetch(
          "http://abashin.ru/cgi-bin/ru/tests/heart", {
            method: "POST",
            headers: {
               "Host": "abashin.ru",
               "Connection": "close",
               "Content-Type": "application/x-www-form-urlencoded",
               "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,
   image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
   exchange;v=b3;q=0.9",
               "Accept-Language": "ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7",
               "Cache-Control": "max-age=0",
            },
            body: full
          })
              .then(response => {
                response.text()
              .then(res => {
                navigation.navigate('Результат', {params: res})
            });
          })
        }
        catch (error) {
        console.error(error);
        }
      }
```

```
  const check = () => {
    if ( (colorF==false && colorM==false) || (pulse==0) || (min==0) ) {
      Alert.alert('Вы ввели не все данные')
    } else {
      toResult()
    }
  }

  return (
    <ScrollView style={styles.container}>
        <Text style={styles.text}>Введите вашу дату рождения:</Text>
        <Feather name="calendar" size={24} color="black" style={{marginTop:'4%',
marginLeft: '3%'}}/>
        <View style={{marginTop: '-7%', marginRight: '61%'}}>
        <DateTimePicker
          value={date}
          mode={mode}
          maximumDate={new Date()}
          display="default"
          onChange={onChange}
        />
        </View>
        <Text style={styles.text}>Выберите свой пол:</Text>
        <View style={{flexDirection: 'row'}}>
        <IconButton
          style={{marginTop: '-2%'}}
          icon="gender-female"
          color={colorF ?  'hotpink' : 'black'}
          size={50}
          onPress={() => onClickBtnF()}
        />
        <IconButton
          style={{marginTop: '-2%'}}
          icon="gender-male"
          color={colorM ? '#42aaff' : 'black'}
          size={50}
          onPress={() => onClickBtnM()}
        />
        </View>
      <Text style={styles.points}>1. Вы должны сидеть спокойно 5 минут.</Text>
      <Text style={styles.points}>2. Измерьте пульс в спокойствии.</Text>
      <Text style={styles.points}>3. Введите ваш пульс до приседаний:</Text>
      <TextInput
        style={styles.input}
        borderColor={border1 ? 'green' : 'red'}
        onChangeText={onChangePulse}
        value={pulse}
        placeholder="Введите пульс"
        keyboardType="number-pad"
        returnKeyType="done"
```

```
      />
      <Text style={styles.points}>4. Выполните 20 приседаний, вытягивая руки пере
д собой в среднем темпе (1-1,5 секунды на одно приседание).</Text>
      <Text style={styles.points}>5. Сидите спокойно 2 минуты</Text>
      <Text style={styles.points}>6. Считайте пульс в течение следующих 3,4,5,6 м
инут. Введите в программу количество минут, через сколько пульс стал равным пульс
у до приседаний.</Text>
      <Text style={styles.points}>7. Введите количество минут</Text>
      <TextInput
        style={styles.input}
        borderColor={border2 ? 'green' : 'red'}
        onChangeText={onChangeTime}
        value={min}
        placeholder="Введите количество минут"
        keyboardType="number-pad"
        returnKeyType="done"
      />
      <Text style={styles.points}>8. Нажмите 'Закончить'. Поздравляем, тест завер
шен!</Text>
      <View style={styles.sectionStyle}>
          <Stopwatch
            style={{borderRadius: '30px'}}
            laps
            start={isStopwatchStart}
            reset={resetStopwatch}
            options={options}
            getTime={(time) => {
              setTime(time);
            }}
          />
          <View style={{flexDirection: 'row'}}>
          <TouchableOpacity onPress={() => {
              setIsStopwatchStart(!isStopwatchStart);
              setResetStopwatch(false);
            }}>
            <View style={styles.button}>
              <Text style={{ color: 'white', padding:10, fontSize: 20 }}>{!isStop
watchStart ? 'START' : 'STOP'}</Text>
            </View>
          </TouchableOpacity>
          <TouchableOpacity onPress={() => {
              setIsStopwatchStart(false);
              setResetStopwatch(true);
            }}>
            <View style={styles.button}>
                 <Text style={{ color: 'white', padding:10, fontSize: 20 }}>RESE
   T</Text>
            </View>
          </TouchableOpacity>
          </View>
```

```
        </View>
        <TouchableOpacity onPress={() => {check()}}>
            <View style={styles.submit}>
              <Text style={{ color: 'white', padding:10, fontSize: 20 }}>Завершит
ь</Text>
            </View>
        </TouchableOpacity>
      </ScrollView>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#f5f5f5'
  },
  text: {
    marginTop: '5%',
    marginStart: '3%',
    fontFamily: 'Verdana',
    fontSize: 18,
    fontWeight: '450',
  },
  points: {
    marginRight: '2%',
    marginTop: '2%',
    marginStart: '3%',
    fontFamily: 'Verdana',
    fontSize: 18,
    fontWeight: '450',
  },
  input: {
    fontSize: 15,
    borderRadius: '5px',
    height: 40,
    marginTop: '2%',
    marginStart: '3%',
    marginEnd: '3%',
    borderWidth: 1.5,
    padding: '3%',
  },
  sectionStyle: {
    flex: 1,
    marginTop: '2%',
    alignItems: 'center',
    justifyContent: 'center',
  },
  button: {
    paddingHorizontal: '3%',
    margin: '3%',
```

```
    backgroundColor: '#d37470',
    alignItems: 'center',
    justifyContent: 'center',
    borderRadius: 30
  },
  submit: {
    marginHorizontal: '20%',
    paddingHorizontal: '3%',
    margin: '3%',
    backgroundColor: '#000',
    alignItems: 'center',
    justifyContent: 'center',
    borderRadius: 30
  }
});

const options = {
  container: {
    borderRadius: 30,
    backgroundColor: '#f5f5f5',
    padding: 5,
    width: 200,
    alignItems: 'center',
  },
  text: {
    fontSize: 25,
    color: '#000',
    marginLeft: 7,
  },
};

export default Test;
```

9.6. Приложение F /components/Result.js

```
import React, {useState} from 'react';
import { Text, View, StyleSheet, Image, ScrollView, Number } from 'react-native';
import { useNavigation } from '@react-navigation/native';
import '../images/error.jpg'
import '../images/catGood.jpg'
import '../images/krutoy.jpg'
import '../images/meow.jpg'

const Result = ({ route, navigation }) => {
  const { params } = route.params;
  var img = 'error'
  var text = null;
  var back = '#f8f8f8'

  if (params.split('>')[2].includes('E')==true) {
    text = 'Ошибочные данные';
```

```
  } else {
    text = params.split('>')[4].split('<')[0].split('. ');
    text = text[0] + '. ' + text[1]
  }

  if (text.includes('Пло')) {
    img = 'meow';
  } else if (text.includes('Сре')) {
    img = 'catGood';
  } else if (text.includes('Хор')) {
    img = 'krutoy';
  }
  const image = require('../images/'+img+'.jpg')
  return (
    <ScrollView style={styles.container, {backgroundColor: back}}>
    <View>
    <Text style={styles.title}>Результаты теста</Text>
    <Text style={styles.text}>Тест не заменяет медицинского обследования! Не забы
вайте о регулярном профессиональном медицинском осмотре!</Text>
    <Text style={styles.info}>{text}</Text>
    <Image source={image} style={{width: '70%', resizeMode:'contain', alignSelf:
'center'}}/>
    </View>
    </ScrollView>
  );
};


const styles = StyleSheet.create({
  container: {
    flex: '100%'
  },
  text: {
    alignSelf: 'center',
    marginTop: '5%',
    marginStart: '3%',
    fontFamily: 'Verdana',
    fontSize: 19,
    fontWeight: '450',
  },
  title: {
    alignSelf: 'center',
    marginTop: '10%',
    marginStart: '3%',
    fontFamily: 'Verdana',
    fontSize: 24,
    fontWeight: 'bold',
  },
  info: {
    alignSelf: 'center',
```

```
    marginTop: '10%',
    marginStart: '3%',
    fontFamily: 'Verdana',
    fontSize: 19,
    fontWeight: '450',
  },
});

export default Result;
```