**Приложение для расчета ипотеки любого вида в разных банках**

**Введение**

На данный момент существует много ипотечных калькуляторов, но я еще не видел такого калькулятора, который в зависимости от выбранного вида ипотеки будет рассчитывать ежемесячный платеж в разных банках и выводить наглядную таблицу со сравнением платежа за месяц в разных банках по определенного вида ипотеке. Так мне и пришла идея создания приложения, которое удовлетворяло бы эти требования.

**Описание проекта**

Мой проект – это мобильное приложение для вычисления фиксированного ежемесячного платежа по выбранному типу ипотеки и сравнения предложений разных банков по этой ипотеке. Мобильное приложение состоит из трёх экранов: Экран регистрации, экран, авторизации и экран с калькулятором.

**Постановка задачи**

Необходимо разработать приложение, в котором можно вычислить платеж по ипотеке всего за 4 шага и сравнить предложения из разных банков по выбранной ипотеке.

Перед тем, как начать разрабатывать приложение был разработан следующий план:

1. Выбор основного стека,
2. Выбор экранов (страниц) для приложения и навигации,
3. Выбор базы данных,
4. Определение нужных таблиц для базы данных,
5. Написание экрана регистрации,
6. Написание экрана авторизации,
7. Изучение формулы вычисления аннуитентного платежа (фиксированный платеж по ипотеке в месяц),
8. Написание функции вычисления ежемесячного платежа,
9. Реализация отрисовки полученных значений в виде таблицы (название банка, ставка, ежемесячный платеж).

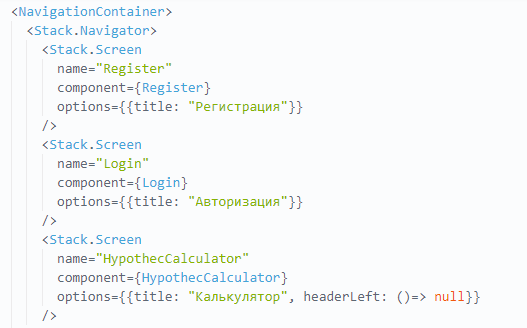
**Работа над проектом**

**Выбор стека**

Для написания такого приложения я решил использовать React native, так как у меня уже раньше был опыт написания веб-приложений на React и синтаксис очень похож, также для отладки и тестирования приложения с React native я использовал фреймворк expo. В проекте также используется JSX, JavaScript и CSS.

**Выбор экранов (страниц) для приложения и навигации**

В ходе планирования страниц приложения были выбраны следующие страницы: Регистрация, Авторизация, Калькулятор. Навигация между экранами была реализована с помощью StackNavigator:



Данный способ реализации навигации является очень удобным, ведь экраны можно раскидать по разным компонентам и из них переходить на другие экраны с помощью передающегося класса navigation.

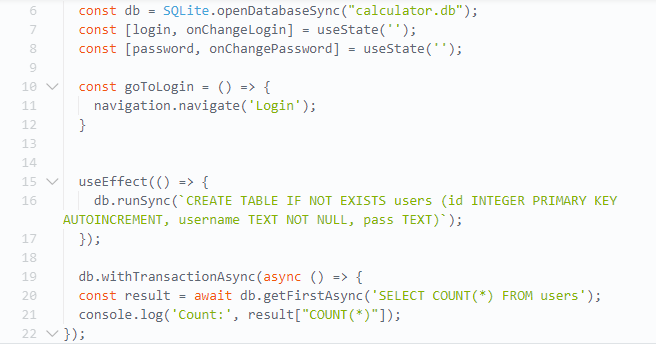
**Выбор базы данных и определение таблиц**

Так как я работаю в expo было решено использовать базу данных SQLite встроенную в expo.

Для работы приложения мне была необходима всего одна таблица users, которая нужна для хранения пользователей и авторизации.

**Написание экрана регистрации**

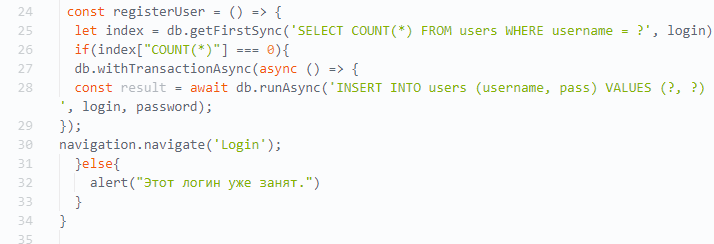
Для корректной работы регистрации была создана база данных calculator.db, таблица users и получена информация из неё, а также написана функция перехода на экран авторизации если у пользователя уже есть аккаунт.



При написании экрана были использованы основные компоненты React native: View, Text, TextInput и Button.

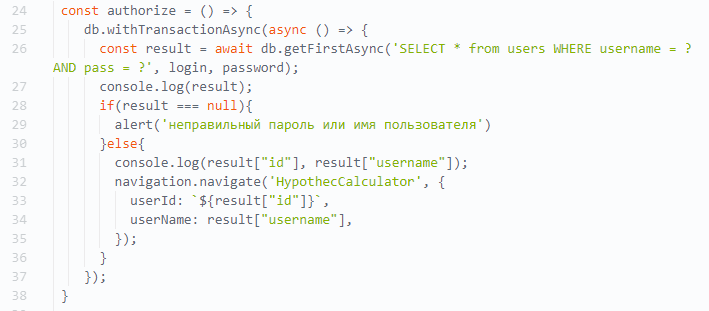


Для кнопки регистрации была написана функция, которая сначала проверяла логин пользователя на уникальность, если логин уже используется другим пользователем, то регистрация по такому же логину будет невозможна. Затем если пользователь ввел уникальный логин и ввел пароль, то он регистрировался в приложении и переходил на экран авторизации. Так же отмечу, что при обращении к базе данных были использованы плейсхолдеры, это устранит уязвимость SQL-инъекции в запросе.



**Написание экрана авторизации**

Разметка экрана авторизации похожа на экран регистрации так что я не буду заострять внимание на рендер экрана, а перейду сразу к функции авторизации. Её суть заключается в запросе который ищет зарегистрированного пользователя в базе данных по введенным пользователем данным, если логин и пароль совпадают хоть с одним рядом в базе данных, то пользователь успешно авторизуется в приложении, если логин или пароль введены неверно, то запрос вернет пустой объект, это значит что пользователь не будет авторизован так как ошибся во вводе логина или пароля. Если же пользователь ввел все верно, то запрос вернет объект с информацией о нем, передаст эту информацию в экран с калькулятором и перенесет пользователя на калькулятор.

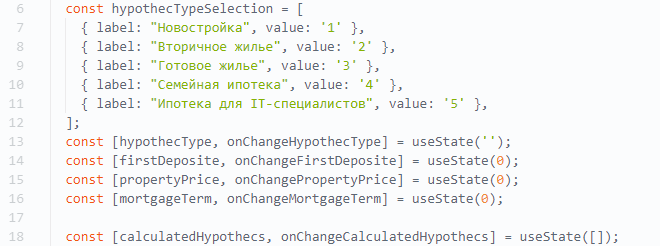


**Написание экрана калькулятора ежемесячного платежа по ипотеке**

Сначала необходимо было сделать так, чтобы после авторизации экран калькулятора приветствовал пользователя его именем. Это было реализовано через ранее переданные при навигации параметры (данные).



К обычным компонентам React был добавлен выпадающий список Dropdown, который позволяет выбрать элемент из списка:

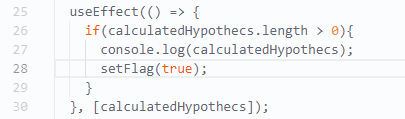


Были добавлены состояния для отслеживания и использования ввода пользователя для расчета ипотеки. Также была написана функция вычисления ипотеки, покажу работу функции на одном из видов ипотеки.



Для расчета ипотеки на новостройку сначала обнуляется состояние “Вычисленных ипотек” – эта переменная отвечает за отрисовку предложений разных банков, так как при каждом вычислении мы должны перерисовывать таблицу, я обнуляю состояние каждый раз как нажиматся кнопка расчета ипотеки. Затем я подготавливаю данные для расчета ипотеки и для каждого банка рассчитываю ипотеку с определенными в объекте процентами. Массив bankList отвечает за конечное состояние объекта для отрисовки таблицы, при каждом вычислении в цикле, все ряды таблицы будут записываться в этот массив и в конце будут записаны как новое состояние массива объектов calculatedHypothecs. По такому же принципу работают и другие кейсы, но с другими банками и процентами.

После вычисления ипотеки и получения всей нужной информации сразу же отрисовывается таблица так как изменяется состояние объекта отрисовки:



Был использован условный рендеринг, то есть если массив с объектами для отрисовки пустой, то отрисовываться ничего не будет, но если он не пустой, то отрисуется список предложений банков.



**Нереализованные идеи**

Была попытка сделать парсер, для того чтобы получать Название банка, Вид ипотеки и ставку ипотеки в процентах. После кучи потраченного времени мне так и не удалось создать парсер.

Мой парсер в теории должен был работать так:

С помощью запроса асинхронного запроса fetch я обращался к странице где лежала интересующая меня информация, а именно таблица с названием банка и его процентной ставки. Затем я парсил полученный ответ - HTML(в строковом формате) и получал реальный HTML, где уже мог по селекторам находить нужную информацию, получать все значения и записывать их в отдельный массив, так как размер всех трех массивов был бы одинаковым, я мог бы отрисисовать таблицу со всей информацией с помощью одного цикла for либо использовать массив объектов и пройти по нему с помощью метода map.

Данный парсер не удалось реализовать так как я получал неполный HTML-код, его неполнота заключалась в том, что большинство классов в div не было прописано в ответе на запрос, соответственно я не смог по классам найти нужную информацию на странице, так как самих классов не было.

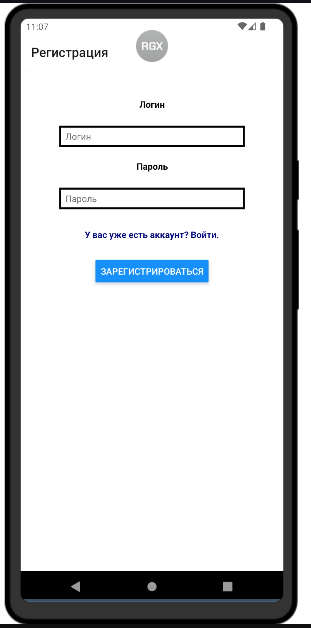
Если бы мне удалось реализовать мой парсер, то я бы смог динамически получать проценты по ипотеки со всех банков и также динамически рассчитывать ипотеку для пользователя. В текущей версии проекта я решил отрисовывать информацию о названии банка и текущих процентах по ипотеке на данный момент вручную. Данный код не входит в итоговый проект.

Код парсера прикрепляю ниже:

**Подведение итогов**

В ходе работы были разработаны экраны регистрации авторизации и калькулятора, выглядят они следующим образом.

Экран регистрации



Экран авторизации

A screen shot of a phone

Description automatically generated

Экран калькулятора

