МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет
имени К.И.Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

Ораз Әкежан Ғалымжанұлы

Разработка приложений для FinTech отрасли

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту

5В070400 – Вычислительная техника и программное обеспечение

Алматы 2022

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И.Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

ДОПУЩЕН К ЗАЩИТЕ
Заведующая кафедрой ПИ
канд. физ-мат. наук, профессор
_____ А.Н. Молдагулова
«_25_» _05_____ 2022 г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту

На тему: «Разработка приложений для FinTech отрасли»

По специальности 5В070400 – Вычислительная техника и программное обеспечение

Выполнил Ораз Ә. Ғ.

Рецензент
синьор-лектор
_____ Баймиров К. М.
"_24_" _____ мая _____ 2022 г.

Научный руководитель
синьор-лектор
_____ Жумабаев А. А.
"_23_" _____ мая _____ 2022 г.

Алматы 2022

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И.Сатпаева

Институт автоматики и информационных технологий

Кафедра "Программная инженерия"

5В070400 – Вычислительная техника и программное обеспечение

УТВЕРЖДАЮ
Заведующая кафедрой ПИ
канд. физ-мат. наук, профессор
_____ А.Н. Молдагулова
«25» 05 2022 г.

ЗАДАНИЕ
на выполнение дипломного проекта

Обучающемуся: *Ораз Әкежан Ғалымжанұлы*
Тема: *Разработка приложений для FinTech отрасли*
Утверждена *приказом проректора по академической работе* № 489-п/о
от "24" 12 2021 г.
Срок сдачи законченного проекта *"17" 05 2022 г.*
Исходные данные к дипломному проекту:
А) Создание макетов страниц и фрагментов в проекте Android Studio;
Б) Создание навигации по страницам и добавление навигационной панели bottomNavigation;
В) Создание интерфейсов, presenter-ов и представлений;
Г) Подключение проекта к базе данных Firebase.
Перечень подлежащих разработке в дипломном проекте вопросов: (с точным указанием обязательных чертежей): *представлены 24 слайда презентации.*
Рекомендуемая основная литература: *из 20 наименований.*

ГРАФИК

подготовки дипломного проекта

Наименование разделов, перечень разрабатываемых вопросов	**Сроки представления научному руководителю и консультантам**	**Примечание**
1. Анализ рынка, выбор типа приложения и оптимального пути для разработки	25.03.22	Выполнено
2. Создание проекта на Android Studio, создание макетов страниц и фрагментов	30.03.22	Выполнено
3. Создание навигации по страницам и добавление bottomNavigation	1.04.22	Выполнено
4. Создание интерфейсов приложения	10.04.22	Выполнено
6. Подключение проекта к базе данных Firebase	18.04.22	Выполнено
7. Написание пояснительной записки к дипломному проекту	16.05.22	Выполнено

Подписи

консультантов и нормоконтролера на законченный дипломный проект с указанием относящихся к ним разделов проекта

Наименования разделов	**Консультанты, Ф.И.О. (уч. степень, звание)**	**Дата подписания**	**Подпись**
Нормоконтролер	Жекамбаева М.Н. Доктор Ph.D., ассоциированный профессор	23.05.22	
Программное обеспечение	Марғұлан Қ. Магистр техн.наук, лектор	23.05.22	

Научный руководитель _____ Жумабаев А. А.

Задание принял к исполнению обучающийся _____ Ораз Ә. Ғ.

Дата «13» 11 2021 г.

АННОТАЦИЯ

Тема дипломного проекта “Разработка приложений для FinTech отрасли”. Проект разрабатывался для простых пользователей. Мобильное приложение является онлайн платежной системой. В первой части дипломного проекта будет ознакомление с понятием FinTech. Будут рассмотрены важные моменты использования FinTech технологий. Будет проведен анализ ситуаций в финансовой отрасли после случая пандемии COVID-19. На основе анализа будут рассмотрены моменты, когда FinTech технологии начали развиваться. Будут распределены преимущества использования FinTech технологий.

Во второй части дипломной работы будут рассмотрены программные обеспечения, что были использованы во время разработки приложения для онлайн платежей. Будут рассмотрены функции, по которым работает интегрированная среда разработки – Android Studio. Будут расписаны преимущества языка программирования – Kotlin, в сравнение с схожим языком программирования – Java. Будут расписаны преимущества в использовании платформы Firebase, так же паттерна разработки пользовательского интерфейса – MVP.

В третьей части дипломной работы будет проведен расчет по проделанной работе. Будут подробно рассмотрены детали разработки приложения. Будут приведены примеры кодов основных функций и методов, с помощью которых осуществлялось создание проекта.

ANNOTATION

The topic of the diploma project was "Developing applications for the FinTech industry". The project was developed for ordinary users. The mobile application is an online payment system. The first part of the diploma project will introduce the concept of FinTech. It will cover important aspects of using FinTech technologies. There will be an analysis of the situations in the financial industry after the COVID-19 pandemic case. Based on the analysis, the moments when FinTech technology started to evolve will be considered. The benefits of using FinTech technologies will be distributed.

The second part of the thesis will look at the software that was used during the development of the online payment application. It will describe the functions used in the integrated development environment - Android Studio. The advantages of the Kotlin programming language will be described, in comparison with a similar programming language – Java. The advantages of using the Firebase platform will be described, as well as the user interface development pattern – MVP.

In the third part of the thesis, the calculation of the work done will be carried out. The details of the application development will be discussed in detail. Examples of the codes of the main functions and methods will be given, with which the creation of the project was carried out.

АҢДАТПА

Дипломдық жобаның тақырыбы "FinTech саласына арналған қосымшаларды әзірлеу". Жоба қарапайым пайдаланушылар үшін жасалды. Мобильді қосымша онлайн төлем жүйесі болып табылады. Дипломдық жобаның бірінші бөлімінде FinTech тұжырымдамасымен танысу болады. FinTech технологияларын қолданудың маңызды тұстары қарастырылады. Ол COVID-19 пандемиясынан кейін қаржы саласындағы жағдайларға талдау жасайды. Талдау негізінде FinTech технологиясы дами бастаған сәттер қарастырылады. Fintech технологиясын қолданудың артықшылықтары жайлы айтылады.

Дипломдық жобаның екінші бөлімінде онлайн төлем жүйесі қосымшасын әзірлеу кезінде қолданылған бағдарламалық жасақтамалар қарастырылады. Интеграцияланған әзірлеу ортасы – Android Studio жұмыс істейтін функциялар қарастырылады. Kotlin бағдарламалау тілінің артықшылықтары, ұқсас Java бағдарламалау тілімен салыстырылады. Firebase платформасын пайдаланудың артықшылықтары, сондай-ақ пайдаланушы интерфейсінің әзірлеу паттерны – MVP сипатталады.

Дипломдық жобаның үшінші бөлімінде атқарылған жұмыс бойынша есеп жүргізіледі. Қосымшаны әзірлеу туралы егжей-тегжейлі қарастырылады. Жобаны жасағандағы жүзеге асырылған негізгі функциялар мен әдістер кодтарының мысалдары келтіріледі.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сложно переоценить значение мобильных приложений в современном мире, ведь именно они создают удобные условия для обучения, работы и в повседневной жизни. Мобильные приложения не что иное, как трансформация того, как работают предприятия. Мобильные приложения в настоящее время представляют собой наиболее эффективный, прямой и настраиваемый способ предоставления информации о продукте и поощрения клиентов сохранять лояльность к определенному бренду.

Компании постоянно борются за внимание и лояльность своих клиентов. Конкуренция за то, чтобы предложить лучший продукт более широкой аудитории, не прекращается никогда. Следовательно, в такой быстро меняющейся среде победителем всегда окажется тот, кто установил наиболее эффективные и удобные каналы коммуникации со своей клиентурой.

Мобильные приложения в современном мире имеют такое же значение, как и веб-сайты десять лет назад. В настоящее время, когда мы переживаем время восстановления после пандемии COVID-19, стало понятно, насколько важными являются как мобильные приложения, так и сайты для человечества. Торговые центры, супермаркеты, офисы, школы, ВУЗы которые обычно полны людей, опустели. Только Интернет продолжал расти и развиваться. Веб-сайты продолжали свою работу как обычно и поставляли онлайн услуги людям во время карантина. Многие компании перешли на разработку или же улучшению уже имеющихся мобильных приложений для улучшения своего бренда. Это объяснило мне, почему цифровые технологии важны в эту цифровую эпоху. Именно в такое время люди со всего мира поняли, насколько важны финансовые технологии во всех отраслях жизнедеятельности.

Целью данной дипломной работы является разработка мобильного приложения для FinTech отрасли. Рассказывается о важностях и удобствах, которые принесет использование данного приложения. Подробно описаны все подробности реализаций проекта. Разработка осуществлялась с помощью методов и алгоритмов android разработки. В процессе работы использовались различные программные обеспечения, языки программирования и шаблоны проектирования. Каждая из использованных технологий имеет свои цель и задачи.

Основного круга лиц, для которого разрабатывалось приложение, нет. Следовательно, уникальных критерий для приложения нет. Приложение должен был получиться таким, чтобы каждый желающий мог заинтересоваться в его легкости и удобности использования, взамен традиционным методам платежей. Функции и задачи данного приложения схожи с теми же, что есть в “QIWI”. То есть, приложение является онлайн кошельком. Каждый желающий может зарегистрироваться и стать пользователем данного приложения. У пользователя будет свой аккаунт со сгенерированной электронной банковской картой. Тем самым, пользователю открывается возможности операций

определенной суммы денег, которая хранится в банковской карте пользователя. Доступны такие операции как платежи, переводы и так далее.

1 Исследовательский раздел

1.1 Анализ рынка

FinTech (Financial Technologies) представляют собой современные технологии, помогающие финансовым учреждениям и различным компаниям в управлении финансами бизнеса. FinTech включает в себя: приложения, программные обеспечения и бизнес-модели. Также FinTech является отраслью, в которой компании пускают в ход различные современные финансовые технологии для конкуренции с традиционными финансовыми учреждениями. В большинстве случаев это либо техностартапы, либо компании, использующие финтех-инструменты для улучшения своих услуг.

На сегодняшний день, важно знать, как использовать меняющийся ландшафт финтеха на благо вашего бизнеса. Новые растущие гиганты бизнеса ставят перед собой стратегию для лучшего будущего, и думают о том, как финтех может – и будет – влиять на вашу организацию. Разрабатываются и внедряются новые и старые технологии. А новые продукты открывают новые возможности и заменяют устаревшие решения.

Простыми словами, быстро развивающиеся рынки финтеха дают ценные уроки и понимание не только тем, кто думает о работе или создании компании в этой захватывающей и динамичной отрасли, но и тем, кто рассматривает, как внедрить технологические изменения в свой бизнес, большой или маленький.

Наглядный пример введения FinTech приложения, это Казахстанский “Kaspi.kz”, который представляет собой мобильное приложение для обычных пользователей данного банка. Вот уже с момента создания, то есть с 2017 года, выручка за данное мобильное приложение составило около 168 тысяч долларов.

Платформа привязывает банковскую карту пользователя (2,9 миллионов человек ежедневно используют приложение) и ко всем бизнес-функциям и транзакционным счетам, чтобы помочь им оплачивать многие платежи в автономном режиме на своих смартфонах. Ранее финансовые технологии были признаны только в контексте внутреннего развития финансовых учреждений. Теперь он максимально приближен к потребителям финансовых услуг, потому что является основой всех онлайн-транзакций – от денежных переводов до коммунальных платежей.

Простыми словами Fintech представляет собой электронную платежную систему, которая позволяет клиенту банка или финансового учреждения совершать финансовые или нефинансовые транзакции онлайн через Интернет. Эта услуга предоставляет онлайн-доступ практически ко всем банковским услугам, традиционно доступным через местное отделение, включая денежные переводы, депозиты и онлайн-платежи по счетам клиентам.

Пользоваться FinTech приложениями могут как физические лица с активным банковским счетом, так и любые финансовые учреждения просто зарегистрировавшись. После этого клиенту для воспользования банковскими

услугами не надо будет каждый раз ходить в банк. Данный способ предоставления банковских услуг является не только удобным, но также безопасным. Данные клиентов банка в сетевых порталах защищены паролями и уникальными идентификаторами.

1.2 Обоснование выбора разработки

Роль FinTech-а в обществе становится более важной, чем когда-либо, особенно из-за COVID-19. Будь то онлайн-платежные системы, такие как PayPal и Square, или приложения для торговли акциями, такие как Robinhood или Acorns, скорее всего, мы все так или иначе использовали финтех. Есть несколько важных преимуществ использований финтех технологий:

1) Удобство

Возможность совершать банковские операции где угодно и как угодно - одно из главных преимуществ мобильных и онлайн-банковских решений.

Наши смартфоны и компьютеры, как правило, легко доступны, что позволяет получить доступ к учетной записи 24/7, чтобы быстро выполнять любое количество банковских задач. Цифровой банкинг также предлагает более долгосрочные удобства, такие как возможность безналичного расчета.

2) Особенности

Мобильные и онлайн сервисы многих банков богаты функциями. Банки могут предлагать персонализированные финансовые консультации, инструменты для сбережений, калькуляторы крупных покупок или даже виртуальных помощников, которые могут помочь им решить, какие расходы они действительно могут себе позволить, и все это в удобном приложении.

3) Безопасность

Безопасность является приоритетом № 1 для финансовых учреждений. И это распространяется на мобильный и онлайн-банкинг.

Угрозы, конечно, существуют везде, в том числе и внутри банковского отделения. К счастью, многие банки позволяют легко принимать дополнительные меры безопасности. Например, ваш банк может разрешить вам добавить многофакторную аутентификацию в ваше мобильное приложение и онлайн-банковский счет.

Многие приложения мобильного банкинга теперь позволяют вам использовать биометрическую аутентификацию для входа в систему. Например, приложение Kaspi Bank-а предоставляет два различных варианта биометрического входа в систему – отпечаток пальца, и код доступа. Ваш банк также может автоматически проверять наличие определенных рисков. Ally Bank запрашивает дополнительную проверку, если обнаруживает вход в систему с неизвестного устройства.

4) Контроль

Контроль над своими финансами с возможностью самообслуживания является еще одним значительным преимуществом цифрового банкинга, а также доступом в режиме реального времени для управления и перемещения денег по своему усмотрению.

Это правда. В отличие от личного банковского обслуживания, мобильные банковские приложения и веб-сайты, как правило, не имеют ограничений на выполнение вами банковских задач, таких как внесение чека или перевод денег с одного счета на другой. И становится все легче ориентироваться в ежедневных транзакциях. Мир технологий предлагает возможность получать деньги и тратить их намного проще, чем в прошлые времена.

Выводы по исследовательскому разделу

В исследовательском разделе я познакомился с понятием FinTech. Были рассмотрены важные моменты использования FinTech технологий. Была проанализирована ситуация в финансовой отрасли после случая пандемии COVID-19. На основе анализа были рассмотрены моменты, когда FinTech технологии начали развиваться. Были распределены преимущества использования FinTech технологий.

2 Технологический раздел

2.1 Технологии что были использованы

2.1.1 Android Studio – IDE

Android Studio – официальная интегрированная среда разработки (IDE) под ОС Android. Она была построена на основе IntelliJ IDEA написанной на Java. Также Android Studio имеет инструменты для редактирования кода и для разработчиков.

В Android Studio используется система сборки основанная на Gradle, виртуальное устройство, шаблоны кода и макетов. В каждом из проектов есть хотя бы один или несколько модальностей с исходным кодом и файлами ресурсов. Эти методы включают в себя модули для Android приложений, библиотек и Google App Engine.

В Android Studio есть функция, которая отправляет изменения в коде и в ресурсах в запущенное приложение. Редактор кода предлагает доработку, анализ и исправление ошибок во время написания кода. После сборки проекта, можно будет сгенерировать .apk файлы, а после компиляции есть возможность опубликовать приложение в Google Play Store.

2.1.2 Kotlin

Kotlin – кроссплатформенный, объектно-ориентированный и статически типизированный язык программирования созданный для работы на JVM (Java Virtual Machine). С 2019 года является предпочитаемым Google языком программирования для разработки Android приложений. Язык ориентирован на совместимость, ясность, безопасность и поддержку инструментов.

Kotlin создали в 2010 году компания JetBrains, также ответственная за IntelliJ IDEA. Исходный код языка находится в открытом доступе с 2012 года. Kotlin используется во многих продуктах компании JetBrains не исключая IntelliJ IDEA.

2.1.3 Firebase

Успех мобильного приложения заключается в его использовании, вовлеченности и удержании, поскольку это увеличивает количество покупок в приложении и увеличивает продажи приложений. И хранение данных

мобильных приложений в облаке в наши дни имеет огромное значение.

Платформа для разработки мобильных данных от компании Google – Firebase включает в себя совокупность функции для разработки, перекомпоновки и улучшения backend разработки мобильных приложений.

Основными целями Firebase являются ускорение процесса мобильного приложения, выпуск и обеспечение надежного мониторинга работоспособности, так же, вовлечение пользователей.

При использовании данной платформы предоставляется доступ к сервисам, с помощью которых открывается возможность разрабатывать собственные продукты, что позволяет сосредоточиться непосредственно на предоставлении качественного продукта. На примере платформу Google Firebase можно сделать вывод, что использование данной платформы дает возможность хранить в себе функции базы данных, аутентификации, push-уведомлений, аналитики, хранения файлов и многие другое.

Поскольку сервисы находятся в облаке, разработчики могут поэтапно выполнять масштабирование своих продуктов, не испытывая никаких проблем. Firebase на данный момент входит в число лучших платформ для разработки приложений, которым доверяют разработчики по всему миру.

2.1.4 MVP (Model-View-Presenter)

MVP (Model-View-Presenter) – паттерн разработки пользовательского интерфейса. MVP является производным от MVC (Model-View-Controller), но при этом имеет несколько иной подход. Основное отличие – представление (ведущий) не так сильно связано моделью (модель).

- Model – используется для работы с данными. Модель может быть интерфейсом, который отвечает за доступ к API и локальной базе данных или кешу, поэтому основная роль модели заключается в управлении данными;
- View – реализует отображение данных. Он должен отвечать только за отображение объектов и не должен содержать никакой бизнес-логики. Таким образом, представление только управляет и отображает страницы;
- Presenter – известен как посредник, потому что он реализует взаимодействие между моделью и представлением. Поскольку модель и представление не взаимодействуют напрямую, презентатор используется в качестве связующего звена, он извлекает данные из модели, а затем форматирует их так, чтобы представление могло их понять.

2.2 Преимущества выбранных технологий

2.2.1 Преимущества Android Studio:

Помимо мощного редактора кода IntelliJ и инструментов разработчика, Android Studio предлагает еще больше функций, повышающих производительность при создании приложений для Android, таких как:

- Гибкая система сборки на основе Gradle;
- Быстрый и многофункциональный эмулятор;
- Единая среда, в которой вы можете разрабатывать для всех устройств Android;
- Применяйте изменения к push-коду и изменения ресурсов к вашему запущенному приложению без перезапуска приложения;
- Шаблоны кода и интеграция с GitHub помогут вам создавать общие функции приложения и импортировать примеры кода;
- Обширные инструменты и фреймворки для тестирования;
- Инструменты Lint для нахождения проблем с производительностью, несовместимостью версий и других проблем;
- Поддержка C++ и NDK;
- Встроенная поддержка облачной платформы Google, упрощающая интеграцию Google Cloud Messaging и App Engine;
- На этой странице представлено введение в основные функции Android Studio. Краткое описание последних изменений см. в примечаниях к выпуску Android Studio**.**

2.2.2 Преимущества Kotlin:

Сильные стороны Kotlin перевешивают недостатки языка того же Java. В Java существуют определенные ограничения, которые препятствуют разработке Android API. Kotlin по своей сути легкий, чистый и гораздо менее подробный, особенно с точки зрения написания обратных вызовов, классов данных и методов получения / установки. Другими словами, Kotlin специально разработан для улучшения существующих моделей Java, предлагая решения недостатков дизайна API.

Kotlin устраняет ряд недостатков Java:

1) Краткость

Многие разработчики хвалят Kotlin за лаконичность. Это качество, которым Java не славится; однако удобочитаемость всегда должна иметь приоритет над краткостью. Да, лаконичный характер Kotlin упрощает работу разработчика и снижает риск ошибок, но Kotlin не практикует краткость ради краткости. Шаблонный код является проблемой для чтения и приводит к большему количеству ошибок и пустой трате времени на их выявление

```java
public class ClearBridge {

public static double calculate (double a, String op, double b) throws Exception {
    switch (op) {
        case "add":
            return a + b;
        case "subtract":
            return a - b;
        case "multiply":
            return a * b;
        case "divide":
            return a / b;
        default:
            throw new Exception();
        }
    }

}
```

Рисунок-2.1 – Пример кода на Java

Выше (Рисунок-2.1) приведена простая функция калькулятора, написанная на Java. Для сравнения, вот такой же калькулятор в Kotlin, который показан на Рисунке-2.2:

```kotlin
fun calculate (a: Double, op: String, b: Double): Double {
    when (op) {
        "add" -> return a + b
        "subtract" -> return a - b
        "multiply" -> return a * b
        "divide" - > return a / b
        else -> throw Exception()
    }
}
```

Рисунок-2.2 – Пример кода на Kotlin

2) Совместимость

Работоспособность является основной целью Kotlin. С самого начала целью проекта было использовать существующие знания и опыт, чтобы сделать каждую библиотеку доступной для программистов Kotlin. Разработчики могут просто писать модули на Kotlin, которые безупречно работают в рамках существующего кода Java. Передавая байт-код, компилятор Kotlin позволяет двум языкам работать в унисон в одном проекте.

3) Безопасность

Система типов Kotlin имеет встроенную нулевую защиту. Печально известное исключение NullPointerException в значительной степени ответственно за ошибки разработки Android. Android полагается на null для представления отсутствия значения, но null может легко уничтожить приложение. Kotlin решает эту проблему, включив неотъемлемую нулевую безопасность. Это дополнение избавляет разработчиков от необходимости писать дополнительный код для решения проблемы.

4) Отсутствие необработанных типов

До того, как в игру вступили дженерики, необработанные типы использовались довольно часто. Необработанные типы допускают обратную совместимость, но rawtypes могут вызвать исключение CastClassException, и ошибка возникнет во время выполнения, а не на этапе компиляции. Kotlin не допускает необработанных типов и, как следствие, создает более безопасный код

2.2.3 Преимущества Firebase

Firebase представляет собой полный пакет продуктов, который позволяет создавать веб- и мобильные приложения, улучшать качество приложений и помогать вашим клиентам развивать свой бизнес.

Firebase может использоваться для следующих целей:

– Управление всеми данными в базе данных в режиме реального времени. Таким образом, обмен данными из базы данных в базу данных происходит легко и быстро. Следовательно, если вы хотите

разрабатывать мобильные приложения, такие как прямая трансляция, обмен сообщениями в чате и т.д., Вы можете использовать Firebase;

- Синхронизация данных в режиме реального времени на всех устройствах - Android, iOS и в Интернете без обновления экрана;
- Интеграция с Google Ads, AdMob, DoubleClick, Play Store, Data Studio, BigQuery и Slack, чтобы упростить разработку ваших приложений с эффективным и точным управлением и обслуживанием;
- Все, начиная от баз данных, аналитики и заканчивая отчетами о сбоях, включено в Firebase. Таким образом, команды разработчиков приложений могут оставаться сосредоточенными на улучшении пользовательского опыта.

Firebase предлагает базу данных Firebase в реальном времени, которая размещается в облаке и представляет собой базу данных NoSQL. Следовательно, он позволяет вам хранить и синхронизировать данные JSON в режиме реального времени.

Выгоды использования Firebase для базы данных:

- Работает не только онлайн, но даже когда пользователи находятся в автономном режиме, чтобы сохранять изменения и синхронизировать их всякий раз, когда пользователи выходят в Интернет
- База данных Firebase в реальном времени хорошо интегрируется с аутентификацией Firebase, что помогает создать простую и интуитивно понятную модель безопасности для таких разработчиков, как вы
- Вам не нужно использовать серверы для создания приложений, потому что база данных Firebase в режиме реального времени позволяет это с помощью мобильных и веб-SDK
- Синхронизация данных в режиме реального времени возможна на любом устройстве с помощью базы данных Firebase в режиме реального времени

2.2.4 Преимущества MVP

MVP – это производное от MVC. Он очень гибкий и позволяет добавлять новые функции и идеи при гораздо меньших затратах, что, в свою очередь, обеспечивает высокое качество продукции. Многие среды поддерживают шаблон, подобный MVP .NET, Java, Kotlin, PHP.

Уровень представления и бизнес-логика разделены, что гарантирует простоту тестирования и понимания кода. Следует отметить, что не рекомендуется использовать его в небольших приложениях.

Плюсы использования MVP:

1. Повторное использование кода – Возможность повторного использования кода будет обеспечиваться принципом разделения интересов. Дизайн будет иметь надлежащую модель предметной области и бизнес-логику в своей логической единице.

2. Подход, основанный на тестировании – Применение изоляции позволяет тестировать компоненты отдельно. Это разделение логической части представления, которая является презентером, с визуальным представлением, которое является фактическим представлением, облегчает модульное тестирование.

3. Адаптируемый дизайн – Изменения и дополнения могут быть намного легче применены, если дизайн хорош. Изолированный код обеспечивает свободу выбора нескольких представлений и источников данных.

4. Многоуровневость – MVP отделяет логику представления от бизнес-логики.

Выводы по технологическому разделу

В технологическом разделе были рассмотрены программные обеспечения, что были использованы во время разработки приложения для онлайн платежной системы. Были рассмотрены функции, по которым работает интегрированная среда разработки – Android Studio. Были расписаны преимущества языка программирования – Kotlin, в сравнение с схожим языком программирования – Java. Были расписаны преимущества в использования платформы Firebase, так же паттерна разработки пользовательского интерфейса – MVP.

3 Проектный раздел

Функционал приложения онлайн платежной системы будет состоять из регистрации, входа / выхода из аккаунта, платежей, переводов и истории платежей.

При открытии приложения пользователь увидит страницу входа. Если пользователь уже зарегистрирован, то он может войти в аккаунт, используя свой логин и пароль. В противном случае, пользователю необходимо зарегистрироваться, введя свой логин, пароль, имя и фамилию. При входе в аккаунт пользователь попадет на страницу с платежами, снизу будет навигационная панель с помощью которого можно переходить на другие страницы. Среди других страниц можно перечислить:

- Страницу с банковской картой, где будет написан ее номер, дата истечения срока и производитель, также там можно заблокировать карту;
- Страницу с историей, где будут указаны суммы, даты и вид платежей;
- Страницу переводов, где можно отправить деньги другому зарегистрировавшемуся пользователю по его номеру карты;
- Страницу с сообщениями;
- Страницу с профилем пользователя с именем и фамилией и кнопкой выхода из аккаунта в правом верхнем углу.

При переводе или платеже проверяется счет пользователя, если сумма перевода или платежа превышает сумма на счете, то появиться всплывающее окно с сообщением, что не хватает денег на счете. При успешном переводе или платеже появиться страница с сообщением о деталях операции. Если пользователь решит заблокировать карту, то он не сможет совершать никакие операции с этого аккаунта.

Выводы по проектному разделу

В проектном разделе были рассмотрены и описаны функционал, дизайн приложения. Также была описана навигация по страницам приложения.

4 Экспериментальный раздел

4.1 Создание макетов страниц приложения

Для начала, в проекте нужно создать макеты страниц приложений. Начал делать макеты страниц после входа в аккаунт.

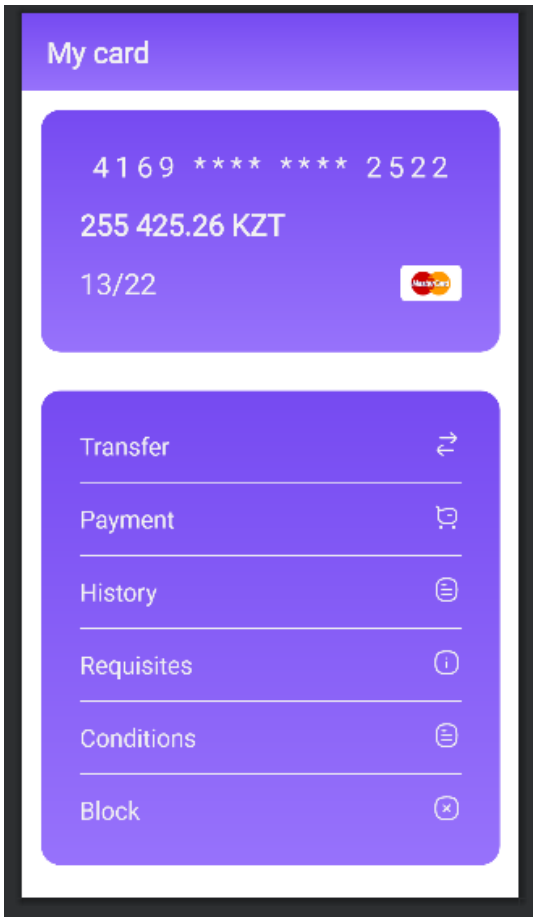


Рисунок-4.1 – Макет страницы с картой и списком операций

На Рисунке-4.1 показаны списки операций, доступные пользователю с помощью электронной банковской карты.

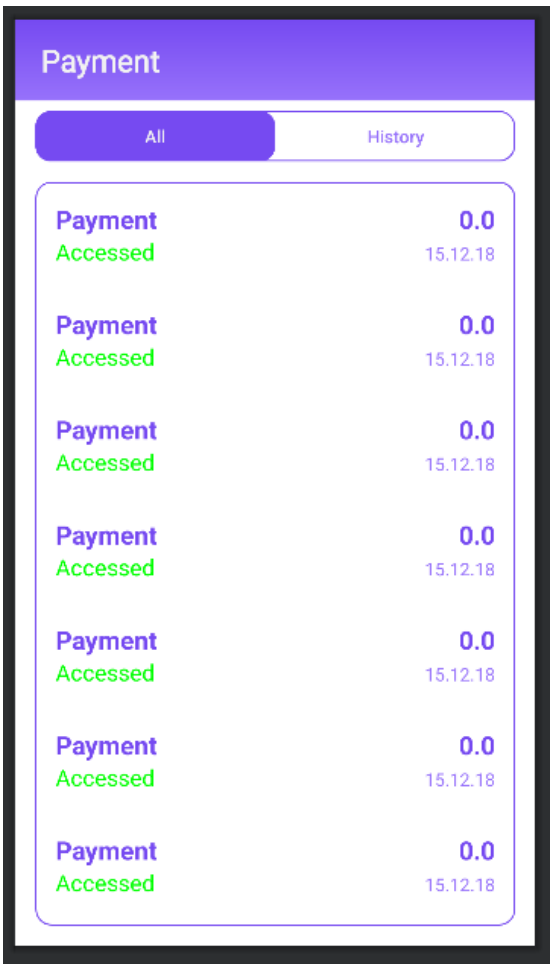


Рисунок-4.2 – Макет страницы историй платежей

На Рисунке-4.2 показаны все операции по платежам, которые были сделаны за последнее время.

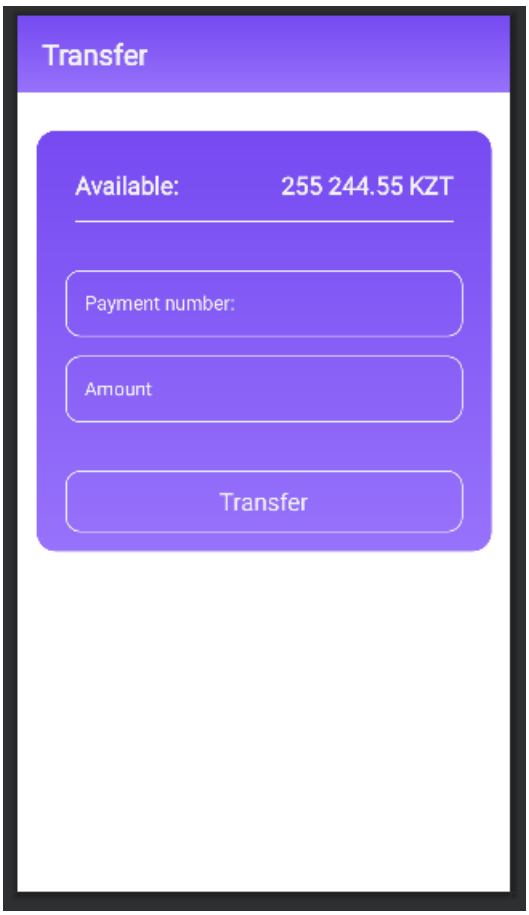


Рисунок-4.3 – Макет страницы совершения платежа

На Рисунке-4.3 показаны данные для совершения платежа, номер платежа (мобильный телефон, коммунальные услуги, т.д.) и сумма. Нажав на кнопку Transfer можно совершить платеж.

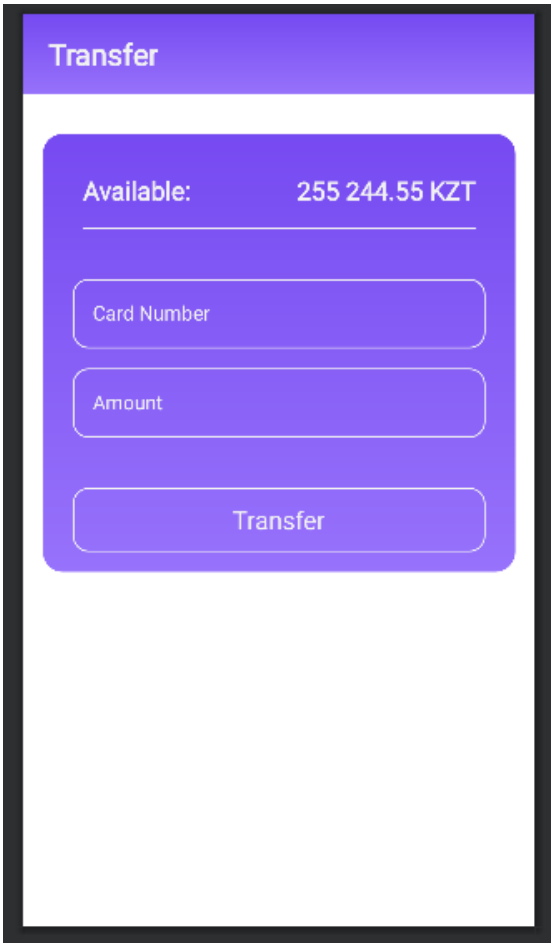


Рисунок-4.4 – Макет страницы совершения перевода

На Рисунке-4.4 показаны данные для совершения перевода, номер карты пользователя и сумма. Нажав на кнопку Transfer можно перевести деньги пользователю.

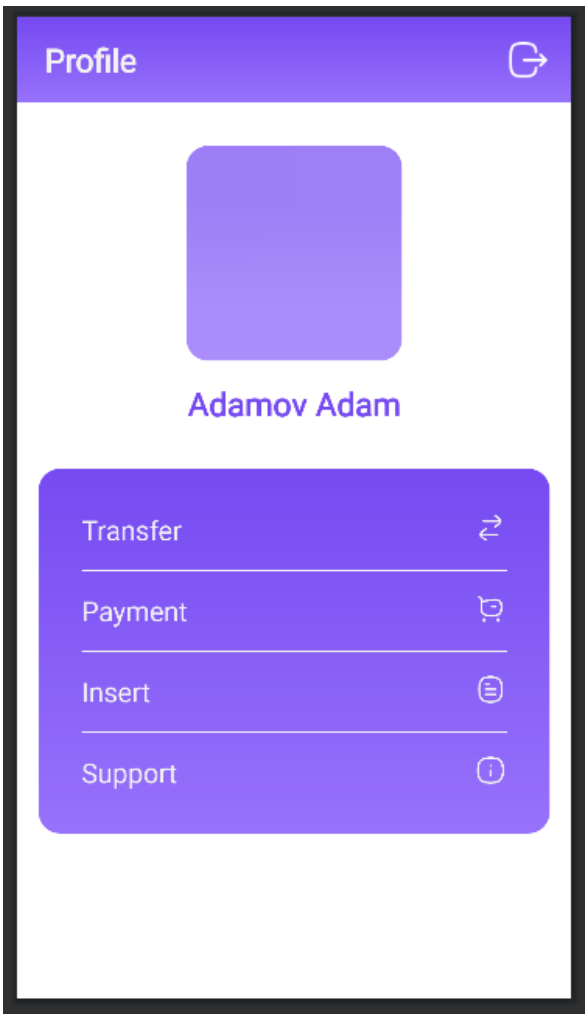


Рисунок-4.5 – Макет страницы профиля

На Рисунке-4.5 можно рассмотреть страницу профиля пользователя. Там есть фамилия и имя пользователя, его аватар и списки доступных операций, такие как переводы, платежи и выход из аккаунта на правом верхнем углу.

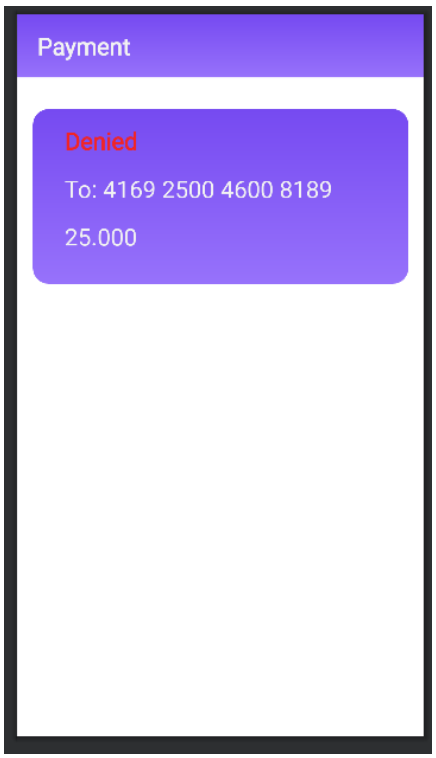


Рисунок-4.6 – Макет страницы результата платежа

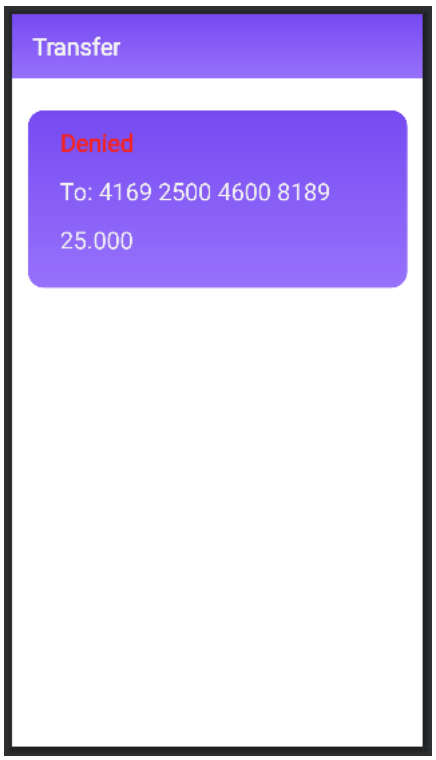


Рисунок-4.7 – Макет страницы результата перевода

После страниц аккаунта я создал макеты страниц для результатов платежа (Рисунок-4.6) и перевода (Рисунок-4.7). Указаны сам результат (перевод / платеж совершен или же отказано), данные получателя и сумма.

Далее я создал макеты для регистрации и входа в аккаунт.

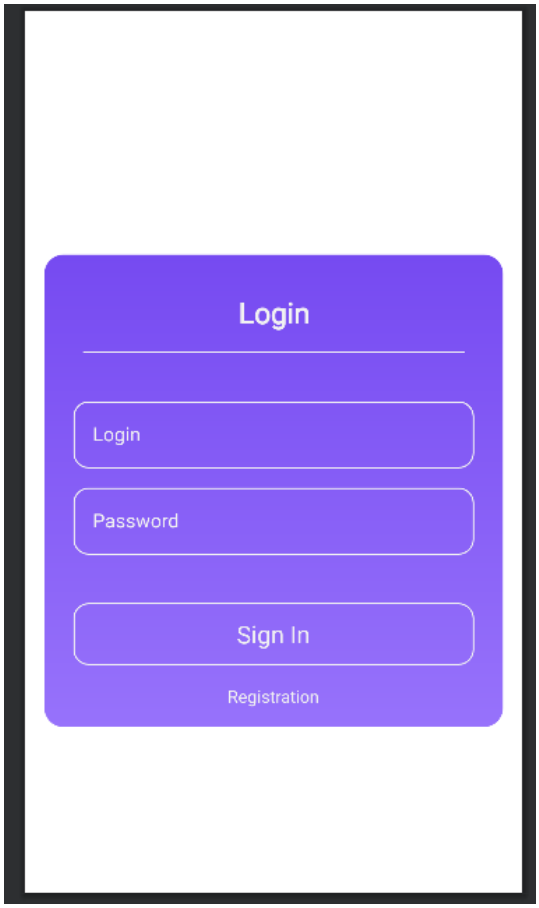


Рисунок-4.8 – Макет страницы входа в аккаунт

На Рисунке-4.8 указаны данные для входа в аккаунт, логин и пароль. Внизу кнопки входа и регистрации.

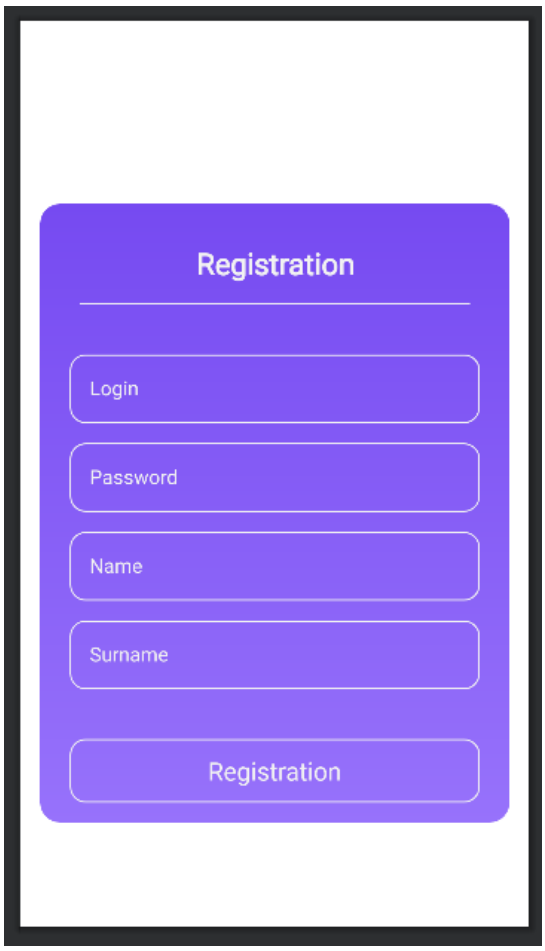


Рисунок-4.9 – Макет страницы с регистрацией

На Рисунке-4.9 указаны данные для регистрации нового пользователя, логин, пароль, имя и фамилия. Внизу находится кнопка регистрации.

4.2 Процесс разработки приложения

```
class MainActivity : AppCompatActivity() {

    lateinit var binding: ActivityMainBinding
    lateinit var navController:NavController
    lateinit var transfer:Transfer
    lateinit var payment:String
    lateinit var paymentObject:Payment
    var isAccess = false
    lateinit var token:String

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        init()
    }


    private fun init() {
        Constants.mainActivity = this
        token = Constants.token
        val navHostFragment =
            supportFragmentManager.findFragmentById(R.id.fragmentContainerView) as NavHostFragment
        navController = navHostFragment.navController
        binding.bottomNavigation.setupWithNavController(navController)
    }


}
```

Рисунок-4.10 – Создание и инициализация глобальных переменных

В самом начале разработки приложения в MainActivity создаем и инициализируем все глобальные переменные, которые можно увидеть на Рисунке-4.10.

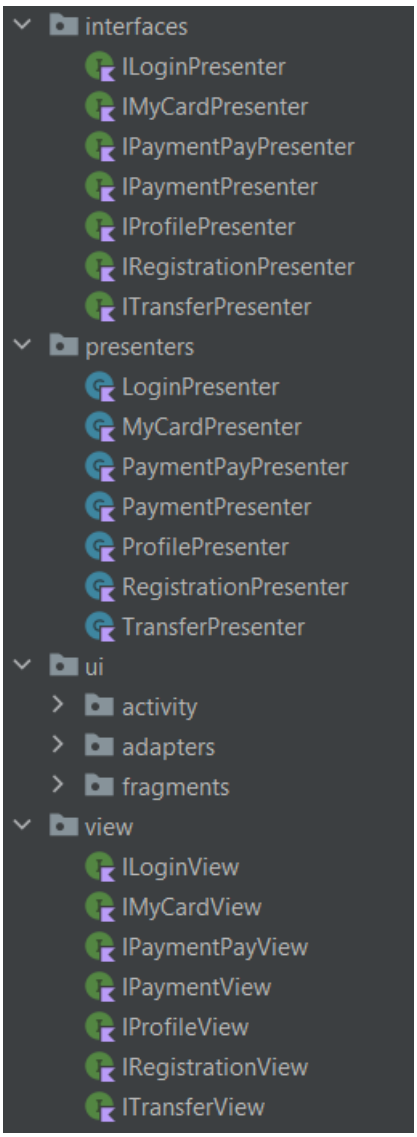


Рисунок-4.11 – Создание классов и интерфейсов

В следствии того, что при создании данного проекта была задействована архитектура MVP, мы создаем классы и интерфейсы презентеров (Рисунок-4.11), где есть запросы.

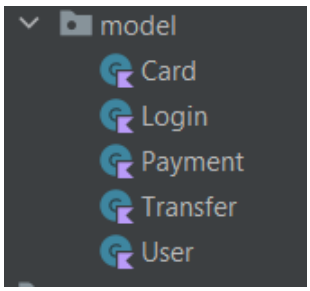


Рисунок-4.12 – Создание моделей для объектов

Так же были созданы модели для созданных объектов (Рисунок-4.12).

```
package com.example.fintech.data.firebase

import android.util.Log
import com.example.fintech.data.model.Card
import com.example.fintech.data.model.Login
import com.example.fintech.data.model.User
import com.example.fintech.utils.Constants
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.SetOptions
import java.util.*
import kotlin.collections.HashMap

class FirebaseDao (
    private var db:FirebaseFirestore,
    private var token:String
        ) {
```

Рисунок-4.13 – Подключение моделей в файле FirebaseDao

```
fun addLogin(login: Login) {
    val loginHash = hashMapOf(
        "login" to login.login,
        "password" to login.password,
        "token" to login.token
    )

    db.collection( collectionPath: "logins")
        .document(login.token)
        .set(loginHash)
}

fun addUser(user: User) {
    val userHash = hashMapOf(
        "name" to user.name,
        "last_name" to user.last_name,
        "card" to user.card,
        "bonuses" to user.bonuses,
        "userImagePath" to user.userImagePath,
        "uuid" to user.uuid
    )

    db.collection( collectionPath: "users")
        .document(user.uuid)
        .set(userHash)
}
```

Рисунок-4.14 – Добавление пользователя в базу данных в FirebaseDao

```
fun setMoney (card:Card, cardHashMap: HashMap<String, Card>) {
    Constants.db.collection( collectionPath: "users")
        .document(card.uuid)
        .set(cardHashMap, SetOptions.merge())
        .addOnSuccessListener { documentReference ->
            Log.d( tag: "TAG",  msg: "DocumentSnapshot added with ID: ${documentReference}")
        }
        .addOnFailureListener { e ->
            Log.w( tag: "TAG",  msg: "Error adding document", e)
        }
}
```

Рисунок-4.15 – Отображение суммы из карты пользователя в базе данных

В классе FirebaseDao данные для базы данных берутся из моделей (Рисунок-4.13). Далее стоят функции добавления пользователей (Рисунок-4.14) и функция отображения суммы из карты пользователя (Рисунок-4.15).

```
class MyCardPresenter (
    private val view:IMyCardView
): IMyCardPresenter {
    lateinit var card:Card
    private val firebaseDao = FirebaseDao(Constants.db, Constants.token)

    override fun getCard(token: String) {
        Constants.db.collection( collectionPath: "users")
            .whereEqualTo( field: "uuid", token)
            .get()
            .addOnSuccessListener { result ->
                for (document in result) {
                    card = document.toObject(User::class.java).card
                    Log.d( tag: "TAG+token", token)
                }
                view.setData(card)
            }
            .addOnFailureListener { exception ->
                Log.w( tag: "TAG",  msg: "Error getting documents.", exception)
            }
    }

    override fun blockCard() {
        card.isBlock = true
        val cardFromHash = hashMapOf(
            "card" to card
        )
        firebaseDao.setMoney(card, cardFromHash)
    }

    override fun unBlockCard() {
        card.isBlock = false
        val cardFromHash = hashMapOf(
            "card" to card
        )
        firebaseDao.setMoney(card, cardFromHash)
    }
}
```

Рисунок-4.16 – Запросы для получения данных карт

В классе MyCardPresenter создаются запросы для получения данных карты пользователя, блокировки и разблокировки карты (Рисунок-4.16).

```
override fun doPayment (amount:Double, paymentNumber: String) {
    if (myCard.amount > amount && !myCard.isBlock) {
        myCard.amount -= amount
        val cardFromHash = hashMapOf(
            "card" to myCard
        )
        val payment = Payment(UUID.randomUUID().toString(), Date(), isAccess: true, Constants.mainActivity.payment, description: "", amount)
        myCard.historyPayment.add(payment)
        firebaseDao.setMoney(myCard, cardFromHash)

        view.doPayment(payment)
    }
}
```

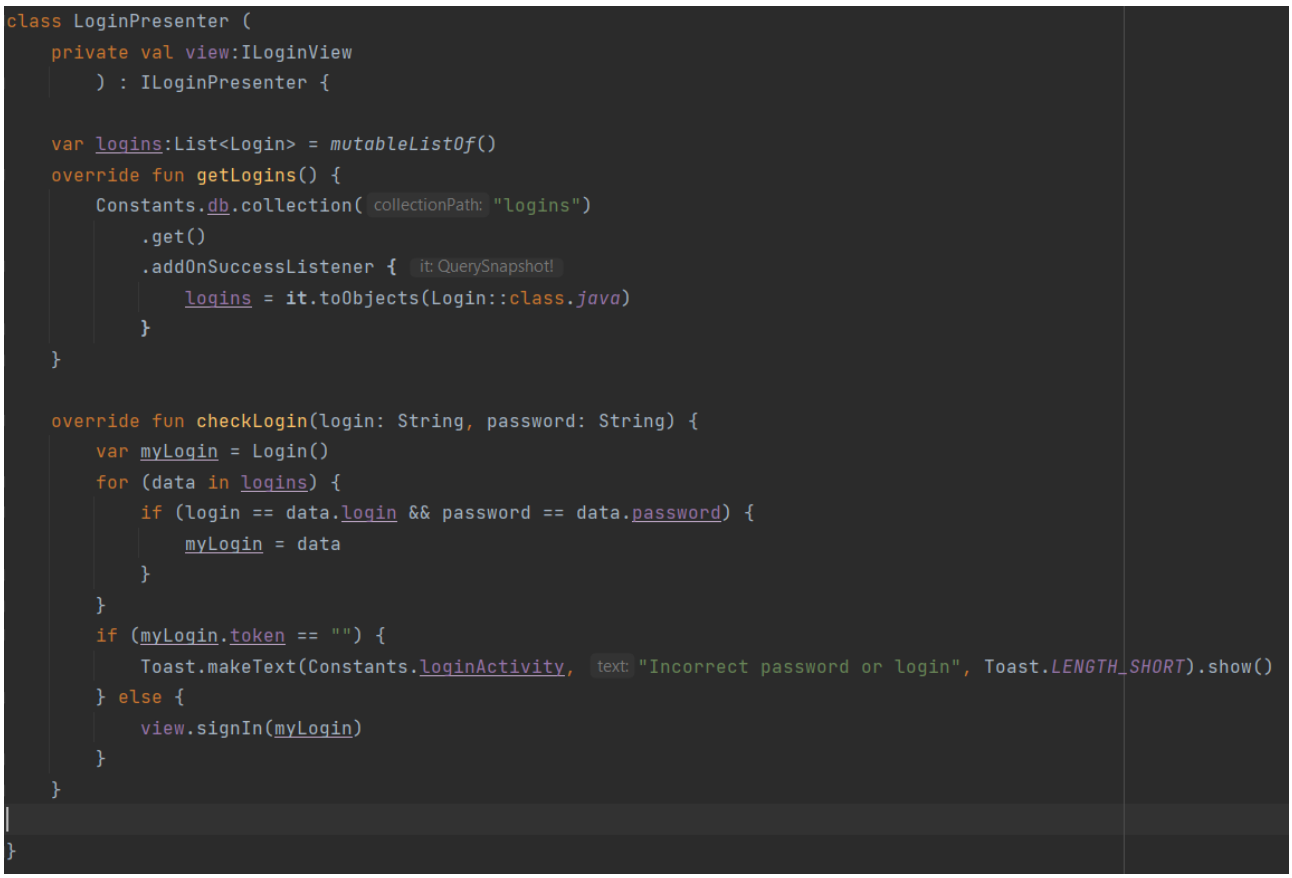
Рисунок-4.17 – Проверка счета

Платеж проходит проверку состояния счета. То есть, уточняются вопросы в нехватки или хватки определенной суммы на счету (Рисунок-4.17). Так же счет проверяется на доступ, то есть, заблокирован или нет. Все исходящие запросы находятся внутри класса FirebaseDao.

```
override fun doTransfer(cardNumber: String, amount: Double): Boolean {
    if (checkTransfer(cardNumber,amount)) {
        changeAmount(amount)
        Toast.makeText(Constants.mainActivity,  text: "Transfer successful", Toast.LENGTH_SHORT).show()
        return true
    }
    Toast.makeText(Constants.mainActivity,  text: "Transfer failed", Toast.LENGTH_SHORT).show()
    return false
}

private fun checkTransfer(cardNumber:String, amount: Double):Boolean {
    var isCardTrue = false
    for (card in cards) {
        if (card.cardNumber.filterNot { it.isWhitespace() } == cardNumber) {
            isCardTrue = true
            secondCard = card
        }
    }

    return !myCard.isBlock || isCardTrue || myCard.amount > amount
}

private fun changeAmount(amount: Double) {
    myCard.amount -= amount
    secondCard.amount += amount
    val cardFromHash = hashMapOf(
        "card" to myCard
    )
    val cardToHash = hashMapOf(
        "card" to secondCard
    )

    val transfer = Transfer(myCard.cardNumber, secondCard.cardNumber, amount)

    myCard.historyTransfer.add(transfer)
    firebaseDao.setMoney(myCard, cardFromHash)
    firebaseDao.setMoney(secondCard, cardToHash)

    view.doTransfer( isAccess: true, transfer)
}
```

Рисунок-4.18 – Функции перевода, проверки и обновления данных при переводе

Перед переводом денег, проверяется присутствие других карт, доступность суммы на счету, доступ к операциям. После удостоверения в правильности после данных проверок, осуществляется переводы на другие счета (Рисунок-4.18).



```
class LoginPresenter (
    private val view:ILoginView
        ) : ILoginPresenter {

    var logins:List<Login> = mutableListOf()
    override fun getLogins() {
        Constants.db.collection( collectionPath: "logins")
            .get()
            .addOnSuccessListener {  it: QuerySnapshot!
                logins = it.toObjects(Login::class.java)
            }
    }

    override fun checkLogin(login: String, password: String) {
        var myLogin = Login()
        for (data in logins) {
            if (login == data.login && password == data.password) {
                myLogin = data
            }
        }
        if (myLogin.token == "") {
            Toast.makeText(Constants.loginActivity,  text: "Incorrect password or login", Toast.LENGTH_SHORT).show()
        } else {
            view.signIn(myLogin)
        }
    }

}
```

Рисунок-4.19 – Функции для проверки данных при входе

В классе LoginPresenter написаны проверки при входе в аккаунт. Функция getLogins берет данные для входа из базы данных. checkLogin проверяет правильность написания данных при входе и выводит соответствующее сообщение при неправильном его вводе (Рисунок-4.19).

```
class RegistrationPresenter (
    private val view: IRegistrationView
        ) : IRegistrationPresenter {
    var logins = mutableListOf<Login>()
    lateinit var firebaseDao:FirebaseDao
    override fun getLogins() {
        Constants.db
            .collection( collectionPath: "logins")
            .get()
            .addOnSuccessListener {  it: QuerySnapshot!
                logins = it.toObjects(Login::class.java)
            }
    }

    override fun registration(login: String, password: String, name: String, surname: String) {
        var isAccess = true
        for (data in logins) {
            if (data.login == login) {
                isAccess = false
            }
        }

        if (isAccess) {
            val token = UUID.randomUUID().toString()
            val loginObject = Login(login, password, token)
            val cardNumber = "4169 2500 4600 ${(1000..9999).random()}"
            val card = Card(cardNumber,  date: "11/24",  code: 882,  amount: 10000.0, uuid=token)
            val userObject = User(name, surname, card,  bonuses: 0.0,  userImagePath: "", token)
            firebaseDao = FirebaseDao(Constants.db, token)
            firebaseDao.addUser(userObject)
            firebaseDao.addLogin(loginObject)
            view.registration(token)
        }
    }
}
```

Рисунок-4.20 – Функции для регистрации и проверки данных

В классе RegistrationPresenter находятся функции для регистрации и проверки данных. Функция getLogins берет данные пользователей из базы данных. Функция registration проверяет, существует ли вводимый во время регистрации логин в базе данных, и если он уже есть, то надо выбрать другой логин. При правильном заполнении всех форм, данные нового пользователя записываются в базу данных и ему присваиваются номер карты и токен (Рисунок-4.20).

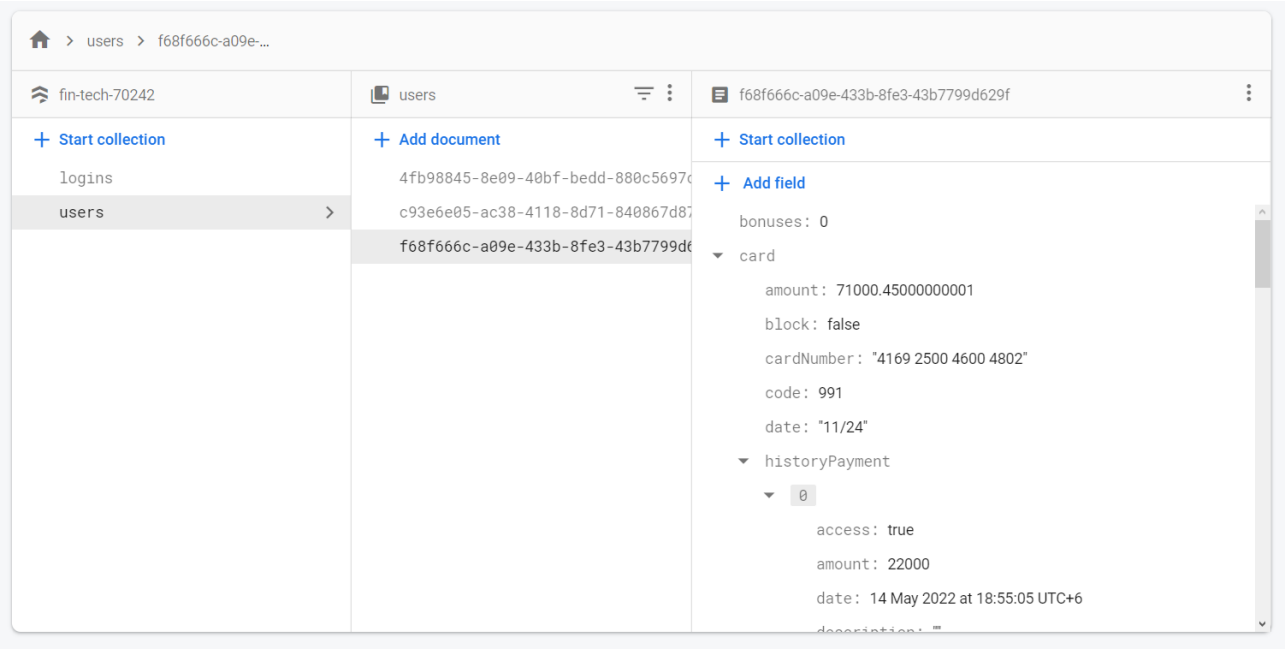


Рисунок-4.21 – Firestore Database

После авторизации в Firebase через Google аккаунт, можно перейти в консоль и оттуда перейти в Firestore Database для просмотра базы данных. В базе данных хранятся данные о пользователях, их данные карты, суммы на счету, истории переводов и платежей (Рисунок-4.21). При переводе или платежах данные о счете пользователя будут обновляться в реальном времени.

Вывод по экспериментальному разделу

В данной части дипломной работы был проведен отчет по проделанной работе. Подробно рассмотрены детали разработки приложения. Были рассмотрены макеты страниц приложения и приведены примеры кодов основных функций и методов, с помощью которых осуществлялась создание проекта.

ЗАКЛЮЧЕНИЕ

Результатом дипломного проекта является автоматизированное мобильное приложение для FinTech отрасли. Продукт является общедоступным приложением для пользователей разных возрастных категорий. Приложение используется для ведения онлайн кошелька и проведения операций с имеющимся счетом онлайн. Основным требованием к приложению была простота в использовании. По итогу работы можно сказать, что приложение является более чем удобным в использовании и приятным для визуального впечатления.

В процессе исследования современного ведения финансовых дел после пандемии COVID-19. На основе исследования сделан следующий вывод: современные крупные компании используют меняющийся ландшафт финтеха на благо своего бизнеса. Новые растущие гиганты бизнеса ставят перед собой стратегию для лучшего будущего, и думают о том, как финтех может – и будет – влиять на вашу организацию. Разрабатываются и внедряются новые и старые технологии. А новые продукты открывают новые возможности и заменяют устаревшие решения.

В процессе проектирования уточнялись требования в создании мобильного приложения в среде Android Studio. Были задействованы различные функции и методы свойственные разработке мобильного приложения в среде Android Studio. Был так же использован объектно-ориентированный язык программирования Kotlin. Был проведен анализ между тем же Kotlin, со схожим языком прогроммирования в ООП отрасли Java. По результатам анализа были сделаны выводы, которые могут обоснованно подтвердить выбор языка программирования Kotlin, как одного из основных инструментов в процессе разработки мобильного приложения. Были задействованы различные методы в разработке мобильного приложения. Была проделана работа с использованием базы данных.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 Советы по созданию мобильных приложений // https://www.appypie.com/business-app-maker
2 8 ошибок при созданий мобильных приложений // Электронная версия на сайте https://habr.com/ru/post/282031/
3 Опыт измерения рынка разработки мобильных приложений в России. Интервью с Анатолием Лариным // Электронная версия на сайте https://habr.com/ru/company/webprofessionals/blog/156305/
4 Обзор современных средств для разработки мобильных приложений // Электронная версия на сайте https://cyberleninka.ru/article/n/obzor-sovremennyh-sredstv-dlya-razrabotki-mobilnyh-prilozheniy
5 Роль разработки мобильного приложений в предприятии // Электронная версия на сайте https://cyberleninka.ru/article/n/rol-razrabotki-mobilnyh-prilozheniy-v-predpriyatii
6 Инструкция по запуску Android Studio // Электронная версия на сайте https://code.tutsplus.com/tutorials/how-to-use-android-studio--cms-37378
7 Android studio // Электронная версия на сайте https://www.techtarget.com/searchmobilecomputing/definition/Android-Studio
8 Разработка информационной системы изучения языков в среде Android // Электронная версия на сайте https://cyberleninka.ru/article/n/razrabotka-informatsionnoy-sistemy-izucheniya-yazykov-v-srede-android
9 Kotlin // Электронная версия на сайте https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html
10 Java или Kotlin // Электронная версия на сайте https://clearbridgemobile.com/java-vs-kotlin-which-is-the-better-option-for-android-app-development/
11 Важность финтеха // Электронная версия на сайте https://professional.dce.harvard.edu/blog/five-important-trends-in-fintech-and-what-they-mean-for-you/#:~:text=Fintech%20is%20the%20application%20of,money%20matters%20easier%20and%20faster.
12 История появления финтеха // Электронная версия на сайте https://finacademy.net/materials/article/fintech
13 Статья про Kaspi.kz // Электронная версия на сайте https://forbes.kz/ranking/object/935#:~:text=%D0%90%D0%9E%20%C2%ABKaspi.kz%C2%BB&text=%D0%92%202017%20%D0%B3%D0%BE%D0%B4%D1%83%20%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BB%D1%81%D1%8F%20%D0%B7%D0%B0%D0%BF%D1%83%D1%81%D0%BA,%D0%B2%D1%81%D0%B5%D1%85%20%D0%B1%D0%B0%D0%BD%D0%BA%D0%BE%D0%B2%20%C2%ABKaspi%20Maps%C2%BB.

14 Внутри Секретного Банка, Стоящего За Бумом Финтеха // Электронная версия на сайте https://forbes.kz/process/probing/the_forbes_investigation_inside_the_secret_bank_behind_the_fintech_boom/
15 5 причин, почему Финтех важен // Электронная версия на сайте https://globalfintechnews.com/5-reasons-why-fintech-is-important/
16 Удобства в использований мобильных приложений Финтех // Электронная версия на сайте https://www.planetcompliance.com/the-importance-and-benefits-of-fintech-apps/
17 Почему FireBase лучший бэкенд сервис в разработке мобильных приложений // Электронная версия на сайте https://www.tristatetechnology.com/blog/firebase-backend-mobile-app/#:~:text=Firebase%20allow%20syncing%20the%20real,and%20accurate%20management%20and%20maintenance
18 Использование FireBase Analytics // Электронная версия на сайте https://code.tutsplus.com/ru/tutorials/get-started-with-firebase-for-android--cms-27248
19 Model View Presenter // Электронная версия на сайте https://medium.datadriveninvestor.com/model-view-presenter-mvp-5c3439227f83
20 Model View Presenter – компромисс и универсальный рецепт // Электронная версия на сайте https://habr.com/ru/post/343438/

ПРИЛОЖЕНИЕ А

Техническое задание

А.1.5 Техническое задание на разработку мобильного приложения для онлайн платежной системы

Настоящее техническое задание распространяется на разработку мобильного приложения для онлайн платежной системы, предназначенной для переводов и платежей средств. Предполагается, что использовать данное приложение будут при онлайн покупках, платеже услуг или переводе средств.

А.1.5.1 Основание для разработки

Система разрабатывается на основании приказа проректора университета № 489-П/Ө от 24.12.2021 г.

А.1.5.2 Назначение

Приложение предназначено для использования при онлайн покупках, платеже услуг или переводе средств на другой счет.

А.1.5.3 Требования к функциональным характеристикам

Приложение должно иметь следующий функционал:
- Регистрация / вход;
- Переводы;
- Платежи;
- История платежей.

А.1.5.4 Требования к надежности

Предусмотреть контроль вводимой информации при регистрации, входе и

переводах.

А.1.5.5 Требования к информационной и программной совместимости

Приложение должно работать под управлением операционных систем Android, реализующих Android API 32.

ПРИЛОЖЕНИЕ Б

Текст программы

MainActivity.kt

```
package com.example.fintech.presentation.ui.activity

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import androidx.navigation.NavController
import androidx.navigation.fragment.NavHostFragment
import androidx.navigation.ui.setupWithNavController
import com.example.fintech.R
import com.example.fintech.data.model.Card
import com.example.fintech.data.model.Payment
import com.example.fintech.data.model.Transfer
import com.example.fintech.data.model.User
import com.example.fintech.databinding.ActivityMainBinding
import com.example.fintech.utils.Constants
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase
import java.util.*

class MainActivity : AppCompatActivity() {

    lateinit var binding: ActivityMainBinding
    lateinit var navController:NavController
    lateinit var transfer:Transfer
    lateinit var payment:String
    lateinit var paymentObject:Payment
    var isAccess = false
    lateinit var token:String

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        init()
    }
```

```
    private fun init() {
        Constants.mainActivity = this
        token = Constants.token
        val navHostFragment =
            supportFragmentManager.findFragmentById(R.id.fragmentContainerView)
as NavHostFragment
        navController = navHostFragment.navController
        binding.bottomNavigation.setupWithNavController(navController)
    }


}
```

LoginPresenter.kt

```
package com.example.fintech.presentation.presenters

import android.widget.Toast
import com.example.fintech.data.model.Login
import com.example.fintech.presentation.interfaces.ILoginPresenter
import com.example.fintech.presentation.view.ILoginView
import com.example.fintech.utils.Constants
import com.google.firebase.firestore.ktx.toObjects
import kotlin.math.log

class LoginPresenter (
    private val view:ILoginView
        ) : ILoginPresenter {

    var logins:List<Login> = mutableListOf()
    override fun getLogins() {
        Constants.db.collection("logins")
            .get()
            .addOnSuccessListener {
                logins = it.toObjects(Login::class.java)
            }
    }

    override fun checkLogin(login: String, password: String) {
        var myLogin = Login()
```

```
        for (data in logins) {
            if (login == data.login && password == data.password) {
                myLogin = data
            }
        }
        if (myLogin.token == "") {
            Toast.makeText(Constants.loginActivity, "Incorrect password or login",
Toast.LENGTH_SHORT).show()
        } else {
            view.signIn(myLogin)
        }
    }

}
```

RegistrationPresenter.kt

```
package com.example.fintech.presentation.presenters

import com.example.fintech.data.firebase.FirebaseDao
import com.example.fintech.data.model.Card
import com.example.fintech.data.model.Login
import com.example.fintech.data.model.User
import com.example.fintech.presentation.interfaces.IRegistrationPresenter
import com.example.fintech.presentation.view.IRegistrationView
import com.example.fintech.utils.Constants
import java.util.*

class RegistrationPresenter (
    private val view: IRegistrationView
        ) : IRegistrationPresenter {
    var logins = mutableListOf<Login>()
    lateinit var firebaseDao:FirebaseDao
    override fun getLogins() {
        Constants.db
            .collection("logins")
            .get()
            .addOnSuccessListener {
                logins = it.toObjects(Login::class.java)
            }
```

```
    }

    override fun registration(login: String, password: String, name: String, surname:
String) {
        var isAccess = true
        for (data in logins) {
            if (data.login == login) {
                isAccess = false
            }
        }

        if (isAccess) {
            val token = UUID.randomUUID().toString()
            val loginObject = Login(login, password, token)
            val cardNumber = "4169 2500 4600 ${(1000..9999).random()}"
            val card = Card(cardNumber, "11/24", 882, 10000.0, uuid=token)
            val userObject = User(name, surname, card, 0.0, "", token)
            firebaseDao = FirebaseDao(Constants.db, token)
            firebaseDao.addUser(userObject)
            firebaseDao.addLogin(loginObject)
            view.registration(token)
        }
    }
}
```

MyCardPresenter.kt

```
package com.example.fintech.presentation.presenters

import android.util.Log
import com.example.fintech.data.firebase.FirebaseDao
import com.example.fintech.data.model.Card
import com.example.fintech.data.model.User
import com.example.fintech.presentation.interfaces.IMyCardPresenter
import com.example.fintech.presentation.view.IMyCardView
import com.example.fintech.utils.Constants

class MyCardPresenter (
    private val view:IMyCardView
): IMyCardPresenter {
```

```
    lateinit var card:Card
    private val firebaseDao = FirebaseDao(Constants.db, Constants.token)

    override fun getCard(token: String) {
        Constants.db.collection("users")
            .whereEqualTo("uuid", token)
            .get()
            .addOnSuccessListener { result ->
                for (document in result) {
                    card = document.toObject(User::class.java).card
                    Log.d("TAG+token", token)
                }
                view.setData(card)
            }
            .addOnFailureListener { exception ->
                Log.w("TAG", "Error getting documents.", exception)
            }
    }

    override fun blockCard() {
        card.isBlock = true
        val cardFromHash = hashMapOf(
            "card" to card
        )
        firebaseDao.setMoney(card, cardFromHash)
    }

    override fun unBlockCard() {
        card.isBlock = false
        val cardFromHash = hashMapOf(
            "card" to card
        )
        firebaseDao.setMoney(card, cardFromHash)
    }
}
```

PaymentPresenter.kt

```
package com.example.fintech.presentation.presenters
```

```
import android.util.Log
import com.example.fintech.data.firebase.FirebaseDao
import com.example.fintech.data.model.Card
import com.example.fintech.data.model.Payment
import com.example.fintech.data.model.User
import com.example.fintech.presentation.interfaces.IPaymentPresenter
import com.example.fintech.presentation.view.IPaymentView
import com.example.fintech.utils.Constants

class PaymentPresenter (
    private val view: IPaymentView
        ) : IPaymentPresenter {
    var paymentAllList = Constants.paymentList
    var paymentHistory = mutableListOf<Payment>()
    lateinit var myCard:Card
    var firebaseDao = FirebaseDao(Constants.db, Constants.token)

    override fun loadPaymentsHistory() {
        view.setPaymentHistory(paymentHistory)
    }

    override fun loadPaymentsAll() {
        view.setPaymentAll(paymentAllList)
    }

    override fun getCard(token: String) {
        Constants.db.collection("users")
            .whereEqualTo("uuid", token)
            .get()
            .addOnSuccessListener { result ->
                for (document in result) {
                    myCard = document.toObject(User::class.java).card
                    paymentHistory = myCard.historyPayment
                    Log.d("TAG+token", token)
                }
            }
            .addOnFailureListener { exception ->
                Log.w("TAG", "Error getting documents.", exception)
            }

    }
}
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б

PaymentPayPresenter.kt

```
package com.example.fintech.presentation.presenters

import android.util.Log
import com.example.fintech.data.firebase.FirebaseDao
import com.example.fintech.data.model.Card
import com.example.fintech.data.model.Payment
import com.example.fintech.data.model.User
import com.example.fintech.presentation.interfaces.IPaymentPayPresenter
import com.example.fintech.presentation.view.IPaymentPayView
import com.example.fintech.utils.Constants
import java.util.*

class PaymentPayPresenter (
    private val view: IPaymentPayView
        ): IPaymentPayPresenter {
    lateinit var myCard: Card
    lateinit var firebaseDao: FirebaseDao

    override fun getCard(token: String) {
        firebaseDao = FirebaseDao(Constants.db, token)
        Constants.db.collection("users")
            .whereEqualTo("uuid", token)
            .get()
            .addOnSuccessListener {
                for (document in it) {
                    myCard = document.toObject(User::class.java).card
                }
                view.setAmount(myCard)
            }
            .addOnFailureListener { exception ->
                Log.w("TAG", "Error getting documents.", exception)
            }
    }

    override fun doPayment (amount:Double, paymentNumber: String) {
        if (myCard.amount > amount && !myCard.isBlock) {
            myCard.amount -= amount
            val cardFromHash = hashMapOf(
                "card" to myCard
```

```
            )
            val payment = Payment(UUID.randomUUID().toString(), Date(), true,
Constants.mainActivity.payment, "", amount)
            myCard.historyPayment.add(payment)
            firebaseDao.setMoney(myCard, cardFromHash)

            view.doPayment(payment)
        }
    }

}
```

TransferPresenter.kt

```
package com.example.fintech.presentation.presenters

import android.util.Log
import android.widget.Toast
import com.example.fintech.data.firebase.FirebaseDao
import com.example.fintech.data.model.Card
import com.example.fintech.data.model.Transfer
import com.example.fintech.data.model.User
import com.example.fintech.presentation.interfaces.ITransferPresenter
import com.example.fintech.presentation.view.ITransferView
import com.example.fintech.utils.Constants
import com.example.fintech.utils.Constants.db

class TransferPresenter (
    private val view:ITransferView
): ITransferPresenter {

    lateinit var myCard:Card
    lateinit var secondCard:Card
    private val cards = mutableListOf<Card>()
    lateinit var firebaseDao: FirebaseDao

    override fun getCards(token: String) {
        firebaseDao = FirebaseDao(db, token)
        db.collection("users")
            .get()
```

```
            .addOnSuccessListener { result ->
                for (document in result) {
                    cards.add(document.toObject(User::class.java).card)
                    if (document.toObject(User::class.java).uuid == token) {
                        myCard = document.toObject(User::class.java).card
                        view.setAmount(myCard)
                    }
                }
                view.setAmount(myCard)
            }
            .addOnFailureListener { exception ->
                Log.w("TAG", "Error getting documents.", exception)
            }
    }

    override fun doTransfer(cardNumber: String, amount: Double): Boolean {
        if (checkTransfer(cardNumber,amount)) {
            changeAmount(amount)
            Toast.makeText(Constants.mainActivity, "Transfer successful",
Toast.LENGTH_SHORT).show()
            return true
        }
        Toast.makeText(Constants.mainActivity, "Transfer failed",
Toast.LENGTH_SHORT).show()
        return false
    }

    private fun checkTransfer(cardNumber:String, amount: Double):Boolean {
        var isCardTrue = false
        for (card in cards) {
            if (card.cardNumber.filterNot { it.isWhitespace() } == cardNumber) {
                isCardTrue = true
                secondCard = card
            }
        }

        return !myCard.isBlock || isCardTrue || myCard.amount > amount
    }

    private fun changeAmount(amount: Double) {
        myCard.amount -= amount
        secondCard.amount += amount
```

```
        val cardFromHash = hashMapOf(
            "card" to myCard
        )
        val cardToHash = hashMapOf(
            "card" to secondCard
        )

        val transfer = Transfer(myCard.cardNumber, secondCard.cardNumber, amount)

        myCard.historyTransfer.add(transfer)
        firebaseDao.setMoney(myCard, cardFromHash)
        firebaseDao.setMoney(secondCard, cardToHash)

        view.doTransfer(true, transfer)
    }
}
```

ProfilePresenter.kt

```
package com.example.fintech.presentation.presenters

import android.util.Log
import com.example.fintech.data.model.User
import com.example.fintech.presentation.interfaces.IProfilePresenter
import com.example.fintech.presentation.view.IProfileView
import com.example.fintech.utils.Constants

class ProfilePresenter(
    private val view:IProfileView
):IProfilePresenter {

    lateinit var user: User


    override fun getUser(token:String) {
        Constants.db.collection("users")
            .whereEqualTo("uuid", token)
            .get()
            .addOnSuccessListener { result ->
                for (document in result) {
```

```
                user = document.toObject(User::class.java)
                Log.d("TAG+token", token)
            }
            view.setData(user)
        }
        .addOnFailureListener { exception ->
            Log.w("TAG", "Error getting documents.", exception)
        }
    }

    override fun getImage(path: String) {

    }

}
```

FirebaseDao.kt

```
package com.example.fintech.data.firebase

import android.util.Log
import com.example.fintech.data.model.Card
import com.example.fintech.data.model.Login
import com.example.fintech.data.model.User
import com.example.fintech.utils.Constants
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.SetOptions
import java.util.*
import kotlin.collections.HashMap

class FirebaseDao (
    private var db:FirebaseFirestore,
    private var token:String
        ) {

    fun getUser(): User {
        var user:User = User()
        db.collection("users")
            .get()
            .addOnSuccessListener { result ->
```

```
            for (document in result) {
                user = document.toObject(User::class.java)
                Log.d("TAG+token", token)
            }
        }
        .addOnFailureListener { exception ->
            Log.w("TAG", "Error getting documents.", exception)
        }

    return user
}

fun addLogin(login: Login) {
    val loginHash = hashMapOf(
        "login" to login.login,
        "password" to login.password,
        "token" to login.token
    )

    db.collection("logins")
        .document(login.token)
        .set(loginHash)
}

fun addUser(user: User) {
    val userHash = hashMapOf(
        "name" to user.name,
        "last_name" to user.last_name,
        "card" to user.card,
        "bonuses" to user.bonuses,
        "userImagePath" to user.userImagePath,
        "uuid" to user.uuid
    )

    db.collection("users")
        .document(user.uuid)
        .set(userHash)
}

fun setMoney (card:Card, cardHashMap: HashMap<String, Card>) {
    Constants.db.collection("users")
        .document(card.uuid)
```

```
            .set(cardHashMap, SetOptions.merge())
            .addOnSuccessListener { documentReference ->
                Log.d("TAG", "DocumentSnapshot added with ID:
${documentReference}")
            }
            .addOnFailureListener { e ->
                Log.w("TAG", "Error adding document", e)
            }
    }

    fun getCard (token: String) {

    }


}
```

РЕЦЕНЗИЯ

на дипломный проект студента 4 курса Казахского национального исследовательского технического университета им. К. И. Сатпаева, кафедры Программной инженерии, специальности 5В070400 – Вычислительная техника и программное обеспечение, Ораза Әкежана на тему: «Разработка приложений для FinTech отрасли»

Структура дипломной работы включает в себя: введение, основные разделы, заключение, список использованной литературы и приложений, а также 23 рисунка.

Во введении рассказывается об актуальности темы, целях и задачах дипломного проекта.

В первом разделе определяются основные термины и понятия, в анализе рынка приводятся схожие проекты. Также обосновывается выбор разработки.

Во втором разделе приводятся технологии и архитектуры, использованные при разработке проекта и их преимущества.

В третьем разделе описываются функционал и пользовательский интерфейс приложения.

В последнем разделе расписывается ход разработки приложения с соответствующими рисунками и их разъяснениями.

В заключении приводятся выводы к проделанной работе.

Заключение по проекту:

В целом, представленная работа считается законченной и может быть оценена на «отлично», при успешной защите Ораз Ә. Ғ. достоин присвоения академической степени бакалавра технических наук.

Рецензент:

Ph.D., синьор-лектор кафедры картографии и геоинформатики

Казахский национальный университет им. аль-Фараби

Баймиров К. М.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К. И. Сатпаева

Институт автоматики и информационных технологий

Ораз Әкежан Ғалымжанұлы
5В070400 – Вычислительная техника и программное обеспечение

ОТЗЫВ НАУЧНОГО РУКОВОДИТЕЛЯ

к дипломному проекту

Тема: «Разработка приложений для FinTech отрасли».

Представленная дипломная работа вполне соответствует заданию и содержит все необходимые блоки. Ораз Әкежан был вовлечен в работу с самого начала и компетентен в разработке мобильных приложений. Назначенные задания и условия были успешно выполнены.

Дипломная работа состоит из введения, четырех разделов, заключения, списка использованной литературы и двух обязательных приложений, а также 23 рисунка.

Введение раскрывает актуальность темы и объясняет цели и задачи дипломного проекта.

Первый раздел объясняет основные понятия и термины, приводятся примеры схожих проектов, и обосновывается выбор разработки.

Второй раздел описывает использованные при разработке технологии и архитектуры, и их преимущества.

Третий раздел рассказывает о функционале и описывает пользовательский интерфейс.

Четвертый, последний раздел расписывает ход разработки с разъяснениями разных частей проекта.

Заключение приводит выводы ко всей проделанной работе.

Пояснительная записка соответствует всем требованиям данных типов работ. Работа представлена в конечной форме и подлежит оценке на «отлично», и при успешной защите Ораз Әкежан заслуживает присвоения академической степени бакалавра технических наук.

Научный руководитель:

Доктор PhD
Ассоциированный профессор _____ **Жумабаев А. А.**

«23» мая **2022 г.**

Университеттің жүйе администраторы мен Академиялық мәселелер департаменті директорының ұқсастық есебіне талдау хаттамасы

Жүйе администраторы мен Академиялық мәселелер департаментінің директоры көрсетілген еңбекке қатысты дайындалған Плагиаттың алдын алу және анықтау жүйесінің толық ұқсастық есебімен танысқанын мәлімдейді:

Автор: Ораз Әкежан

Тақырыбы: Разработка приложений для FinTech отрасли (2)

Жетекшісі: Жибек Алибиева

1-ұқсастық коэффициенті (30): 4.2

2-ұқсастық коэффициенті (5): 1.8

Дәйексөз (35): 0.4

Әріптерді ауыстыру: 1

Аралықтар: 0

Шағын кеңістіктер: 2

Ақ белгілер: 0

Ұқсастық есебін талдай отырып, Жүйе администраторы мен Академиялық мәселелер департаментінің директоры келесі шешімдерді мәлімдейді :

☐ Ғылыми еңбекте табылған ұқсастықтар плагиат болып есептелмейді. Осыған байланысты жұмыс өз бетінше жазылған болып санала отырып, қорғауға жіберіледі.

☐ Осы жұмыстағы ұқсастықтар плагиат болып есептелмейді, бірақ олардың шамадан тыс көптігі еңбектің құндылығына және автордың ғылыми жұмысты өзі жазғанына қатысты күмән тудырады. Осыған байланысты ұқсастықтарды шектеу мақсатында жұмыс қайта өңдеуге жіберілсін.

☐ Еңбекте анықталған ұқсастықтар жосықсыз және плагиаттың белгілері болып саналады немесе мәтіндері қасақана бұрмаланып плагиат белгілері жасырылған. Осыған байланысты жұмыс қорғауға жіберілмейді.

Негіздеме:

Күні *Кафедра меңгерушісі*

24.05.2022