БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ОМСКОЙ ОБЛАСТИ
«ОМСКИЙ АВИАЦИОННЫЙ КОЛЛЕДЖ ИМЕНИ Н.Е. ЖУКОВСКОГО»

09.02.07 ПР-312

ОТЧЕТ ПО ПРОИЗВОДСВЕННОЙ ПРАКТИКЕ

ПП.01.01 Разработка мобильных приложений

ПМ.01 Разработка модулей программного обеспечения

Практикант **Дедовец Н.С.**

Руководители практики **Домбровский Н.С.**

2025

Оглавление

Задание на разработку

Цель практики

Разработка мобильного приложения «Co.Payment».

Структура задания

1. Модуль 1: Разработка технического задания
    a. Описание функциональных требований
    b. Проектирование структуры данных
    c. Создание макета приложения
2. Модуль 2: Разработка серверной части
3. Модуль 3: Верстка мобильного приложения
4. Модуль 4: Разработка функционала мобильного приложения
5. Модуль 5: Тестирование приложения
6. Модуль 6: Документирование результатов разработки

Общие требования:

1. Использовать систему контроля версий Git, ежедневно сохранять прогресс.
2. Проект должен быть структурирован: исходные файлы в соответствующих каталогах.
3. Корректная обработка серверных ошибок и отсутствие соединения — пользователю показывать диалоговое окно с ошибкой.

Анализ индивидуального задания

Разработка мобильного приложения для управления финансами реализована в виде модульной архитектуры на языке Kotlin (Jetpack Compose).

Модуль 1. Техническое задание

Формулировка целей и задач, дизайн-макет, проектирование структуры данных в Supabase (PostgreSQL).

Модуль 2. Серверная часть

Используется Supabase для хранения данных, аутентификации и управления контентом. Реализованы механизмы восстановления пароля через OTP и политики безопасности.

Модуль 3. Верстка интерфейса

Реализованы экраны по макетам, адаптация под разные размеры экранов, интуитивная навигация между экранами, обработка ошибок.

Модуль 4. Функционал

Реализована регистрация, авторизация (email + пароль + PIN), управление транзакциями, работа с картами, профиль пользователя, история операций, поддержка двух языков.

Модуль 5. Тестирование

Тестирование навигации, форм, сценариев ошибок, работы с сервером и обработкой исключений.

Модуль 6. Документация

Создание отчёта и README, структурирование исходного кода.

Проектирование дизайна приложения

Основные принципы:

1. Минимализм и удобство — лаконичный дизайн без лишних элементов, упор на удобство навигации.
2. Адаптивность — макеты корректно отображаются на экранах разных размеров.
3. Консистентность — единый стиль цветов, шрифтов и иконок для всех экранов.
4. Доступность — соблюдение контрастности, читаемости текста и удобных размеров интерактивных элементов.

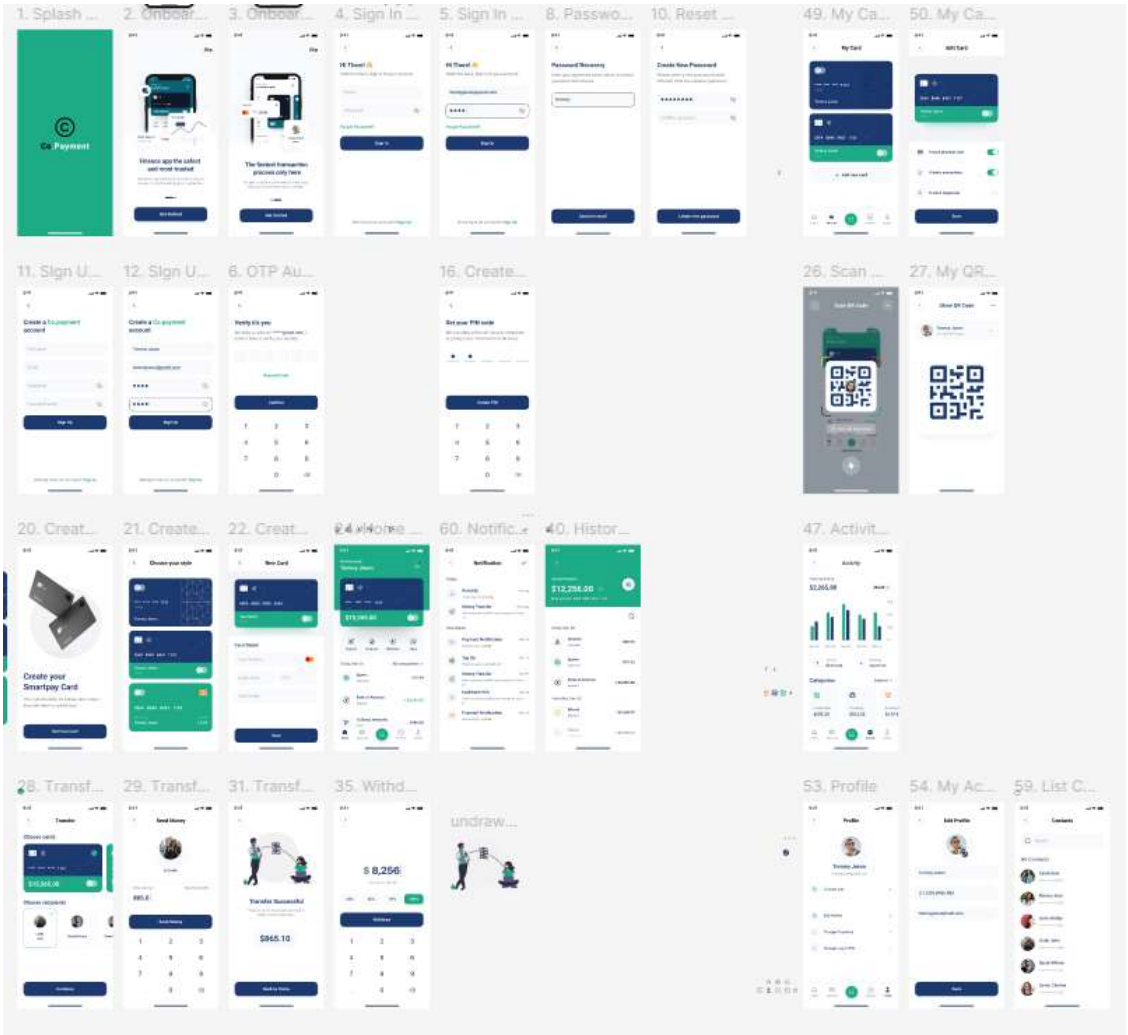

Рисунок 1 Макет Figma

Основные экраны интерфейса:

1.1. Главный экран

- Отображает приветствие пользователя и текущий баланс на выбранной карте
- Включает кнопку быстрого пополнения или перевода средств
- Содержит раздел последних транзакций с детализацией:
  - Временные метки операций
  - Названия получателей или отправителей
  - Суммы переводов с цветовой индикацией (приход/расход)
- Дизайн выполнен в минималистичном стиле с акцентом на важные финансовые показатели

1.2. Экран авторизации

- Чистый минималистичный дизайн с формой входа
- Поля для email и пароля с визуальными разделителями
- Кнопки дополнительных действий:
  - Восстановление пароля
  - Переход к регистрации
- Сохраняет последний использованный email для удобства пользователя

1.3. Процесс регистрации

- Многошаговая форма с:
  - Вводом персональных данных (имя, email, пароль, подтверждение пароля)
  - Созданием PIN-кода для быстрого входа
- Используется цифровая клавиатура для ввода PIN-кода
- Визуальное разделение на логические блоки для повышения удобства

1.4. Профиль пользователя

- Отображает:
  - Аватар и основные данные пользователя (имя, email)
  - Быстрые действия (смена пароля, смена PIN-кода)
  - Список сохранённых контактов для перевода средств
- Предусмотрены настройки аккаунта и ссылки на службу поддержки
- Четкое визуальное разделение секций для лучшей навигации

1.5. Управление картами

- Список всех карт пользователя с маскированными номерами
- Возможность добавления, редактирования и выбора активной карты
- Поддержка различных стилей оформления карт

1.6. Экран QR-кода

- Генерирует персональный QR-код с данными пользователя и карты для обмена контактами
- Возможность сканирования QR-кода для быстрого добавления новых контактов или карт
- Удобный интерфейс для обмена информацией между пользователями

1.7. Переводы средств

- Позволяет выбрать карту-отправителя и контакт-получателя из списка
- Ввод суммы перевода с мгновенным пересчётом баланса
- Подтверждение операции с отображением деталей
- Отдельный экран истории переводов с фильтрацией и поиском

Описание данных приложения

Структура данных приложения разработана для обеспечения работы основных модулей:

- Управление контактами пользователей
- Учёт и обработка финансовых транзакций, а также аналитики расходов
- Хранение информации о профилях пользователей и их банковских картах
- Система уведомлений для обратной связи и безопасности

База данных реализована на Supabase (PostgreSQL) и включает несколько взаимосвязанных таблиц (см. рис. 1), оптимизированных для быстрого доступа, масштабируемости и целостности данных.

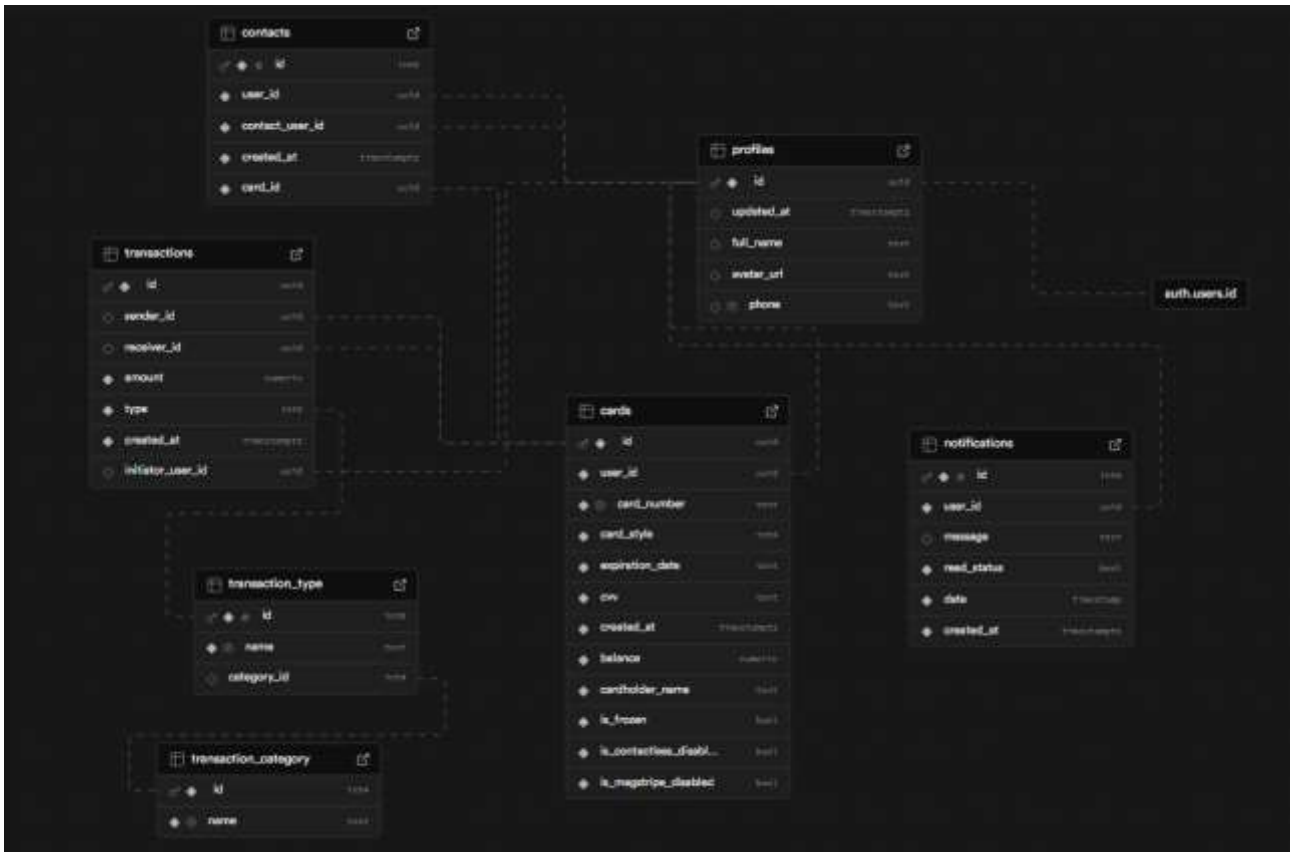

Рисунок 2 Схема БД

Основные таблицы и их назначение

Таблица	Поле	Тип	Краткое описание
contacts	id	int8	Уникальный идентификатор контакта
	user_id	uuid	ID пользователя-владельца
	contact_user_id	uuid	ID пользователя-контакта
	created_at	timestamptz	Дата и время добавления
	card_id	uuid	ID карты, связанной с контактом
transactions	id	uuid	Уникальный идентификатор транзакции
	sender_id	uuid	ID карты-отправителя
	receiver_id	uuid	ID карты-получателя
	amount	numeric	Сумма перевода
	type	int	Тип транзакции
	created_at	timestamptz	Дата и время операции
	initiator_user_id	uuid	ID пользователя-инициатора

profiles	id	uuid	Уникальный идентификатор профиля
	updated_at	timestamptz	Дата последнего обновления
	full_name	text	Полное имя пользователя
	avatar_url	text	Ссылка на аватар
	phone	text	Телефон
cards	id	uuid	Уникальный идентификатор карты
	user_id	uuid	ID владельца карты
	card_number	text	Номер карты
	card_style	int4	Стиль/тип оформления
	expiration_date	text	Срок действия
	cvv	text	CVV-код
	created_at	text	Дата создания
	balance	numeric	Баланс карты
	cardholder_name	text	Имя держателя карты
	is_frozen	bool	Признак заморозки карты
	is_contactless_disabled	bool	Отключение бесконтактной оплаты

	is_magstripe_disabled	bool	Отключение магнитной полосы
notifications	id	int8	Уникальный идентификатор уведомления
	user_id	uuid	Пользователь, для кого уведомление
	message	text	Текст уведомления
	read_status	bool	Прочитано/не прочитано
	date	timestamp	Дата события
	created_at	timestamptz	Дата создания
transaction_type	id	int8	Идентификатор типа транзакции
	name	text	Название типа
	category_id	int4	Категория транзакции
transaction_category	id	int4	Идентификатор категории транзакции
	name	text	Название категории

Связи между таблицами

1. **Контакты ↔ Профили/Карты**
    - contacts.user_id → profiles.id
    - contacts.contact_user_id → profiles.id
    - contacts.card_id → cards.id

2. **Транзакции ↔ Карты/Профили**
    - transactions.sender_id → cards.id
    - transactions.receiver_id → cards.id
    - transactions.initiator_user_id → profiles.id
    - transactions.type → transaction_type.id
3. **Карты ↔ Профили**
    - cards.user_id → profiles.id
4. **Типы и категории транзакций**
    - transaction_type.category_id → transaction_category.id
5. **Уведомления ↔ Профили**
    - notifications.user_id → profiles.id

Описание дизайна приложения

Экран загрузки

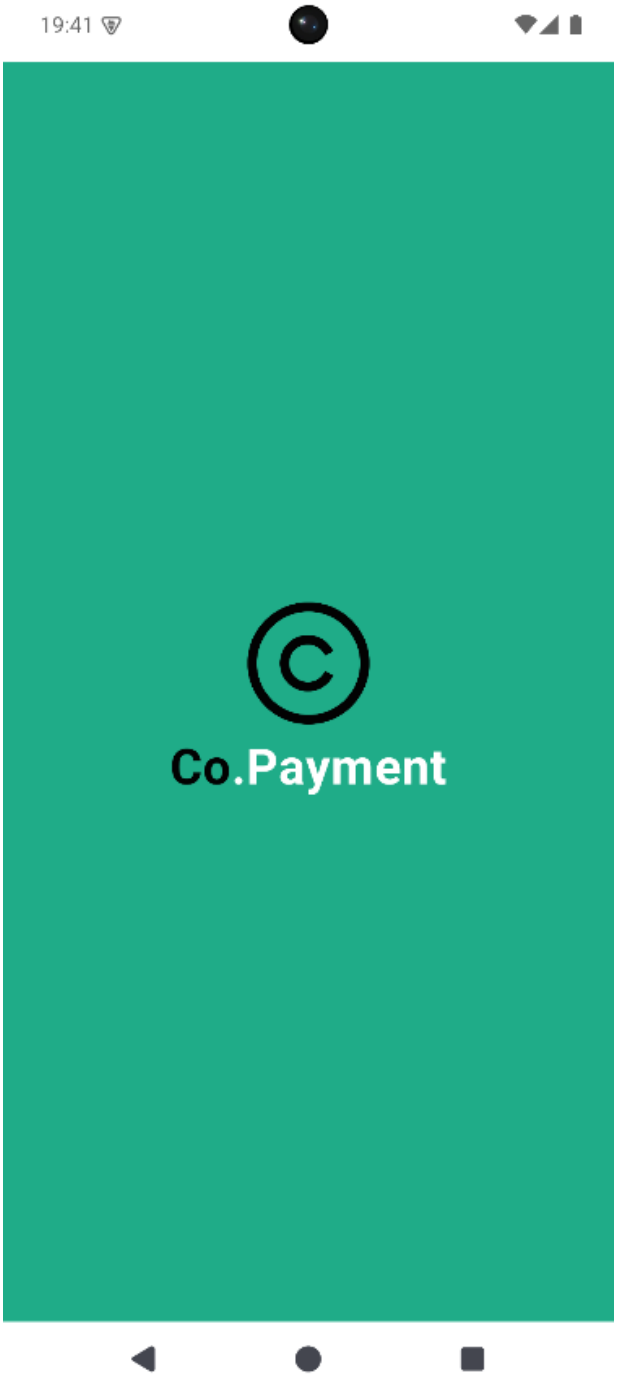

Рисунок 3 Загрузочный экран

Стартовый экран, который появляется сразу после запуска приложения. Содержит логотип Co.Payment и служит для подготовки приложения к дальнейшей работе (загрузка состояния, проверка токена или первого запуска).

Онбординг

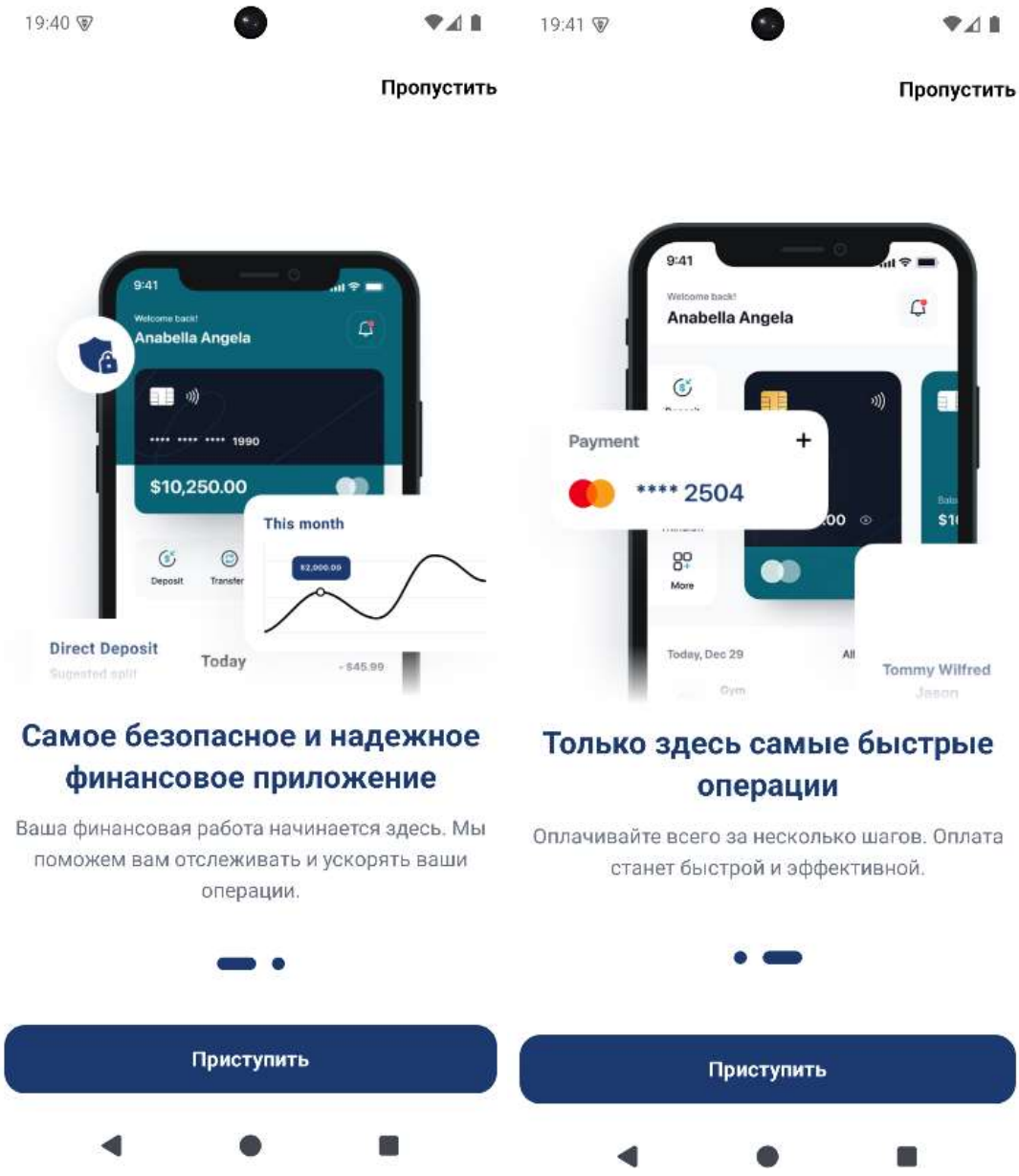

Рисунок 4 Онбординг

Два последовательных экрана с иллюстрациями и краткими тезисами о преимуществах и возможностях приложения:

- Экран 1: рассказывает о безопасности и доверии к финансовому сервису, содержит иллюстрацию, заголовок, описание и кнопку "Приступить".
- Экран 2: демонстрирует простоту и скорость транзакций, также с иллюстрацией и пояснительным текстом.

Экран входа

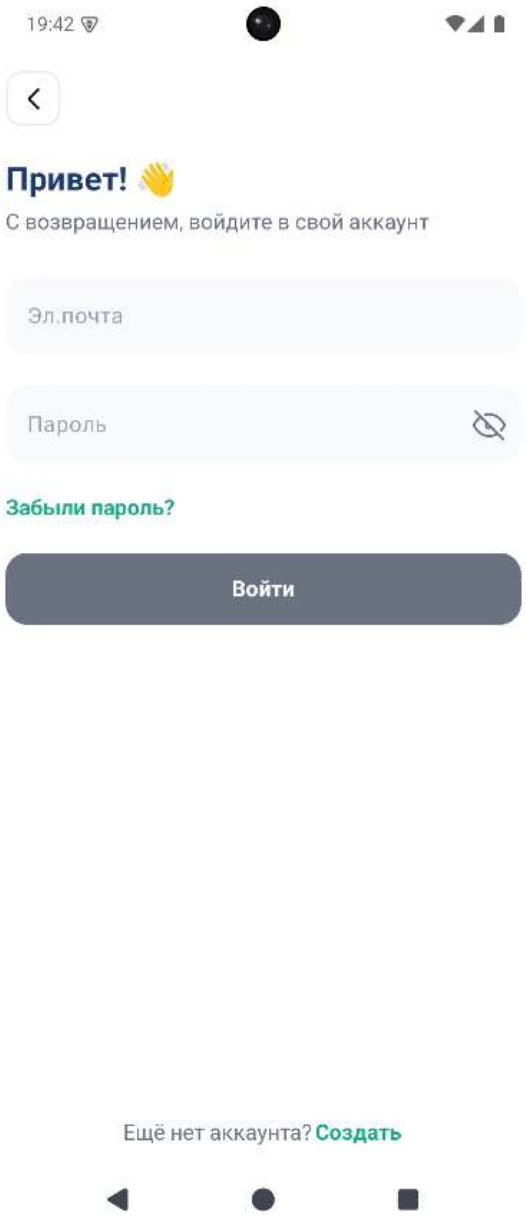

Рисунок 5 Вход

Экран входа в приложение содержит:

- Поля для ввода email и пароля.
- Кнопку войти для авторизации.
- Ссылку "Забыли пароль?" для перехода к восстановлению пароля.
- Ссылку для перехода на экран регистрации ("Зарегистрироваться").

Восстановление пароля

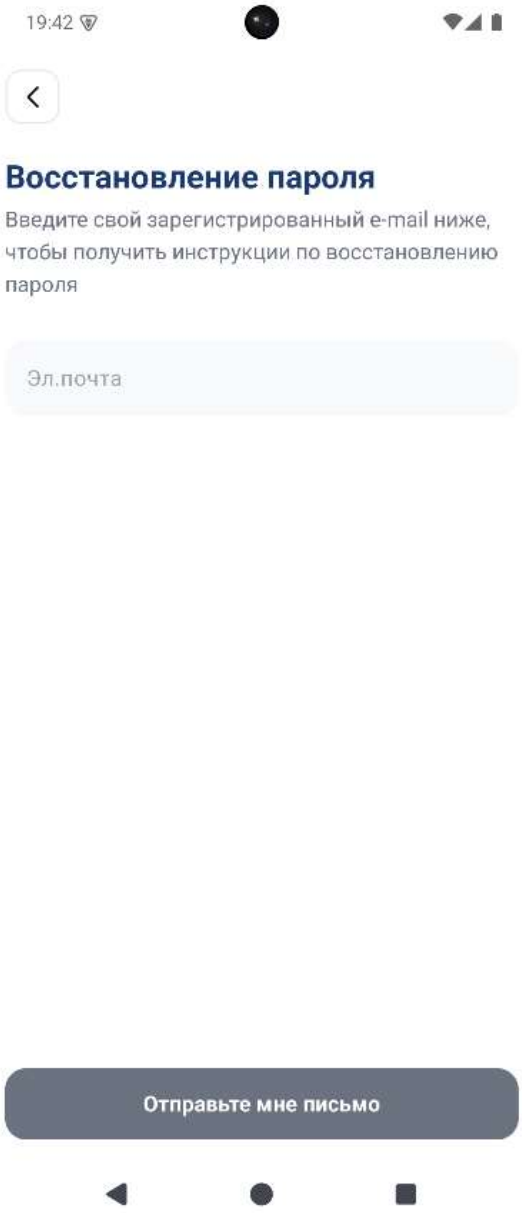

Рисунок 6 Восстановление

Экран для восстановления доступа в случае утери пароля:

- Поле для ввода зарегистрированного email.
- Кнопка "Отправить мне письмо" для отправки инструкции по восстановлению.
- После отправки пользователь получает письмо с кодом или ссылкой для сброса пароля.

Сброс пароля

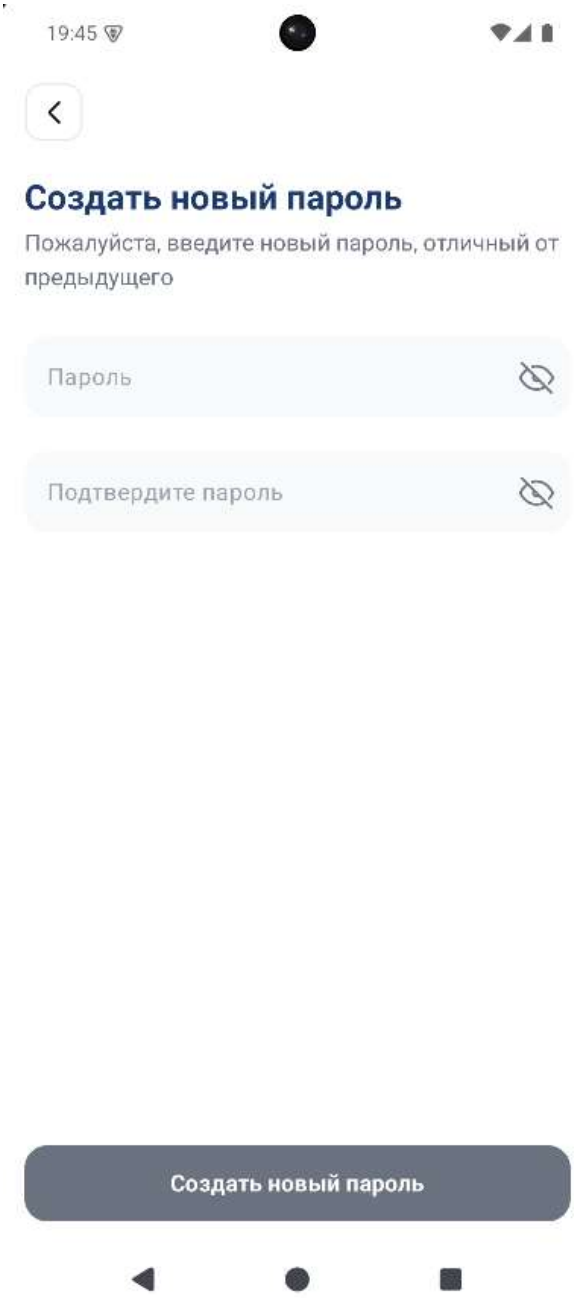

Рисунок 7 Создание нового пароля

Экран для создания нового пароля:

- Два поля: новый пароль и его подтверждение.
- Кнопка "Создать новый пароль" для завершения процесса сброса.
- После успешного восстановления пользователь может войти с новым паролем.

Регистрация

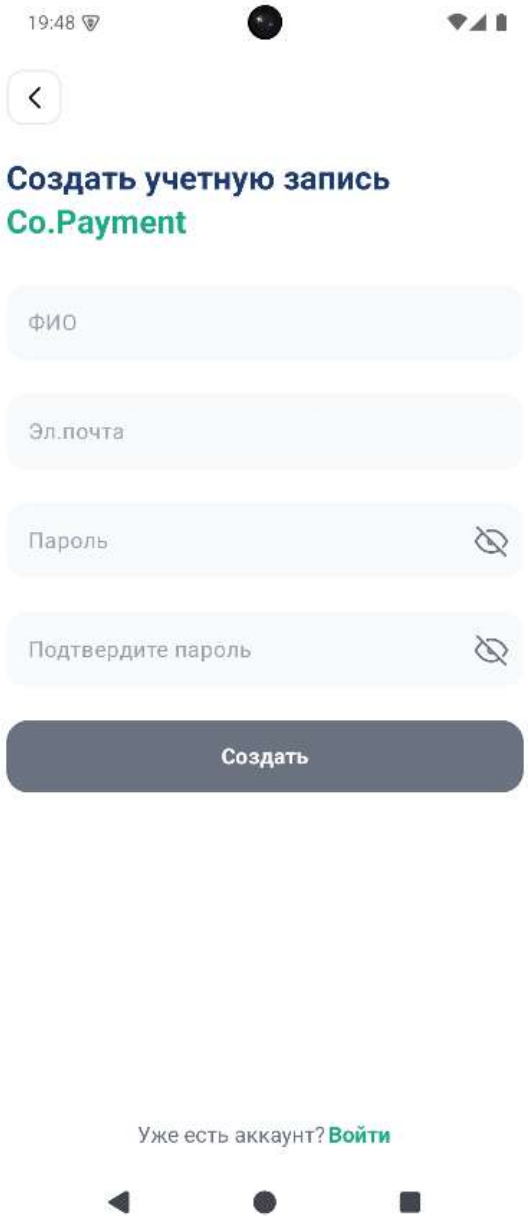

Рисунок 8 Регистрация

Экран регистрации нового пользователя:

- Поля для ввода полного имени, email, пароля и его подтверждения.
- Кнопка "Создать" для создания аккаунта.
- Ссылка для перехода к экрану входа ("Войти"), если аккаунт уже создан.
- Валидация всех полей, информирование пользователя об ошибках ввода.

Подтверждение через код

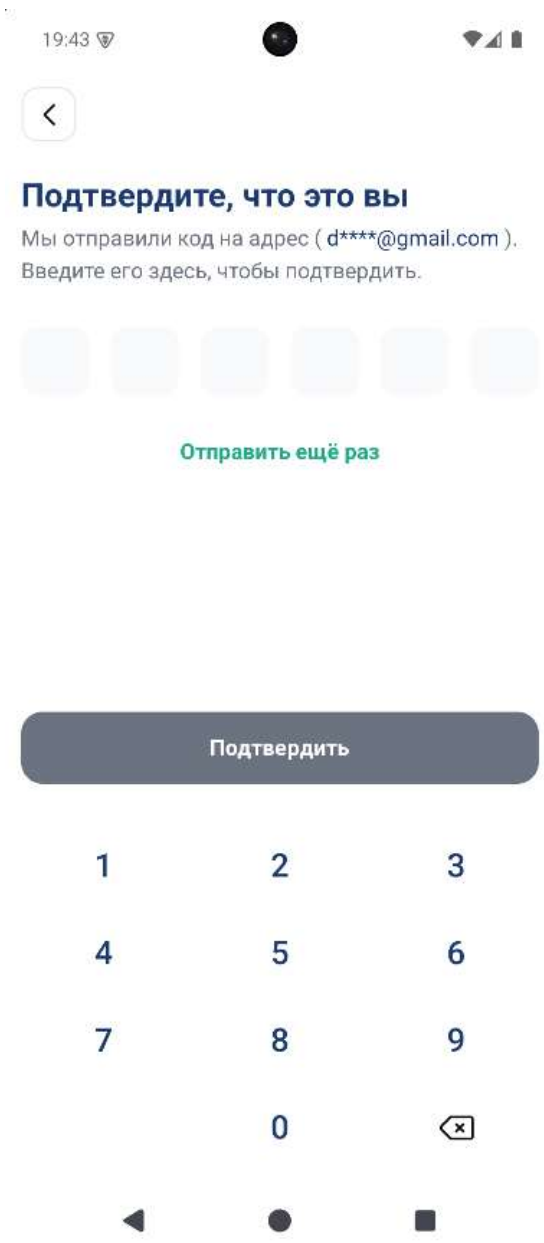

Рисунок 9 Подтверждение кода

Экран подтверждения действий через одноразовый код (OTP):

- Отображает email, на который отправлен код.
- Поле для ввода кода из email.
- Кнопка "Подтвердить" для подтверждения.
- Возможность повторной отправки кода (Отправить еще раз).

Создание PIN-кода

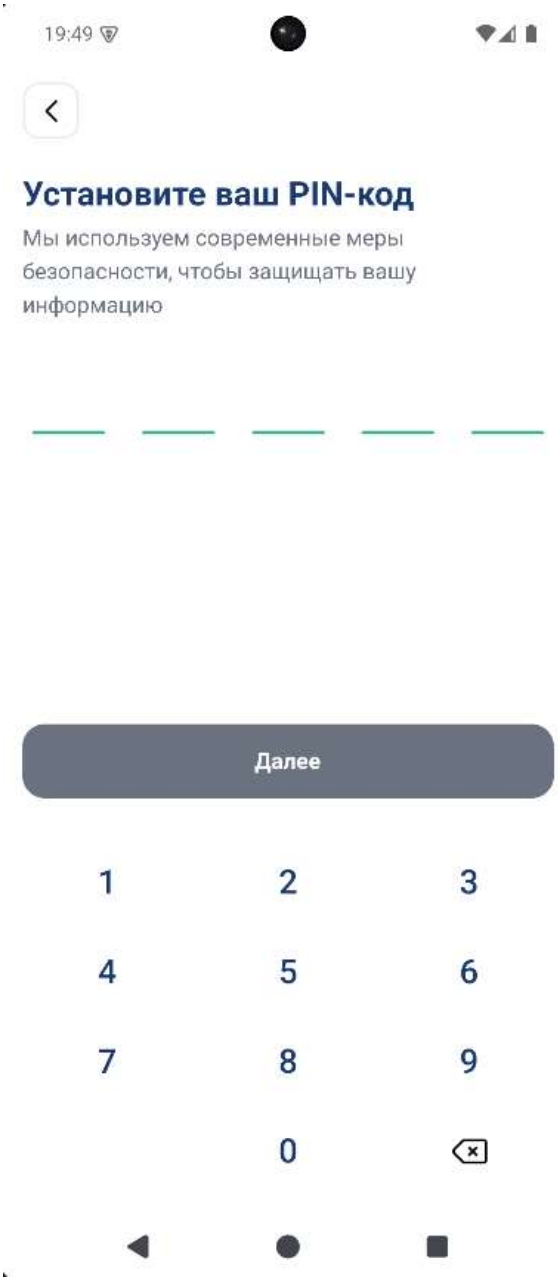

Рисунок 10 Создание ПИН

Экран установки персонального PIN-кода для быстрого входа:

- Краткое описание и инструкция по безопасности.
- Форма для ввода PIN-кода.
- Кнопка "Далее" для сохранения кода.
- Используется цифровая клавиатура для ввода.

Выбор стиля карты

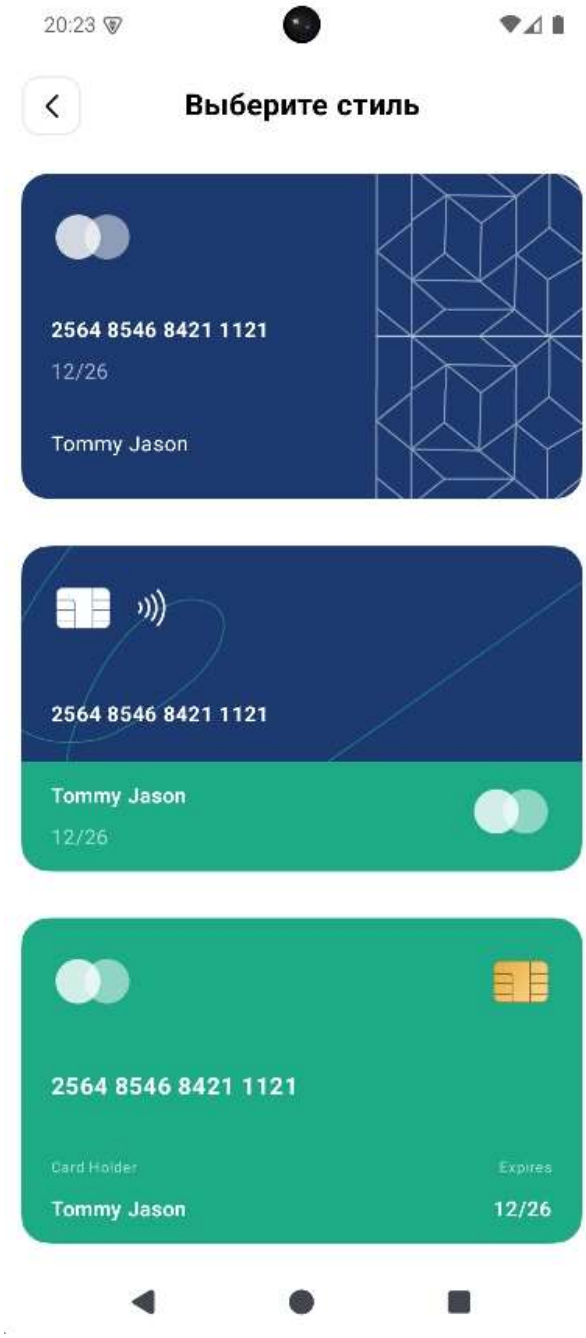

Рисунок 11 Выбор стиля карты

Экран выбора внешнего оформления карты.

- Пользователь видит несколько примеров дизайна (цвета, узоры, оформление).
- Можно выбрать любой понравившийся стиль перед переходом к вводу данных карты.

Создать карту

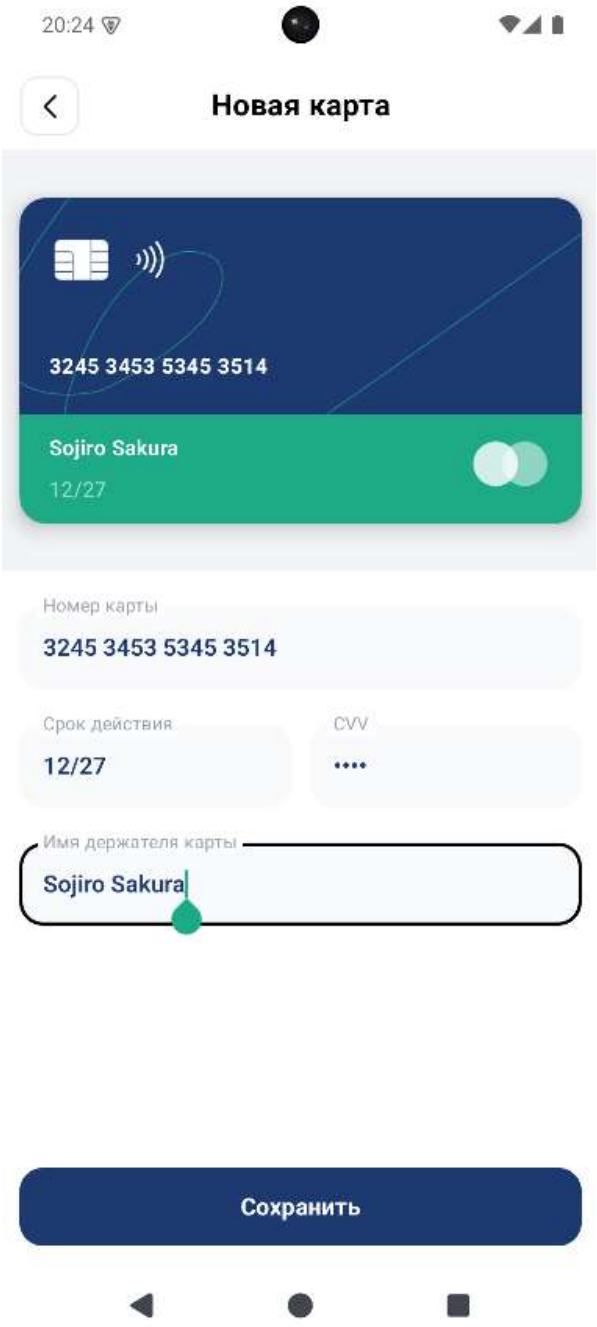

Рисунок 12 Создание карты

Экран ввода данных новой карты.

- Вводится номер карты, срок действия, CVV-код, имя держателя.
- Форма интуитивная, после заполнения пользователь нажимает «Save» для выпуска карты.

Дом

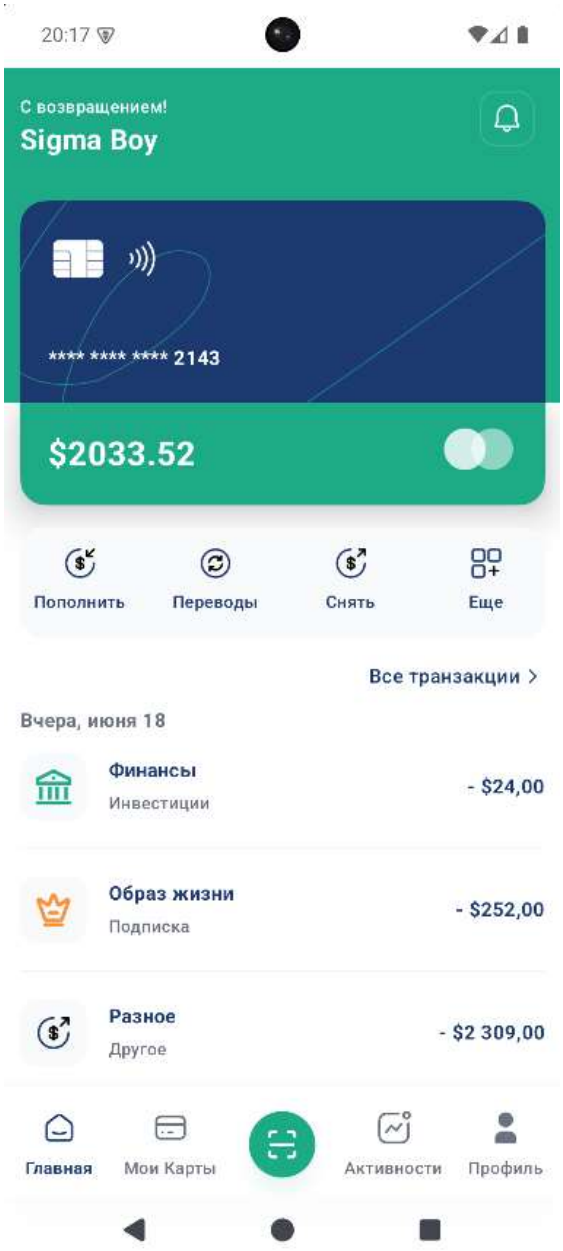

Рисунок 13 Главный экран

Главный экран приложения с обзором счета и быстрыми действиями.

- Отображается имя пользователя, активная карта и текущий баланс.
- Есть иконки для пополнения, перевода, снятия, просмотра истории.
- Ниже — список последних операций с датой, суммой и кратким описанием.

История транзакций

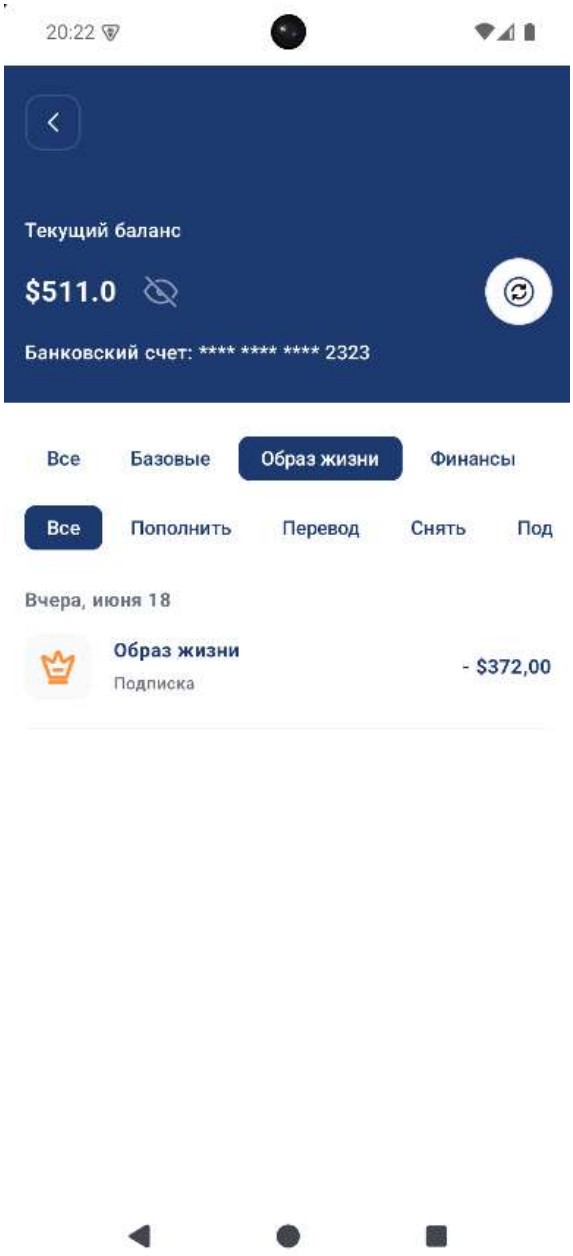

Рисунок 14 История транзакций

Экран истории транзакций по выбранной карте.

- Большой блок текущего баланса, а также список недавних операций.
- Каждая транзакция содержит дату, сумму, категорию и получателя/отправителя.
- История разделена по дням и типам операций для удобного поиска.

Выбор карты и получателя

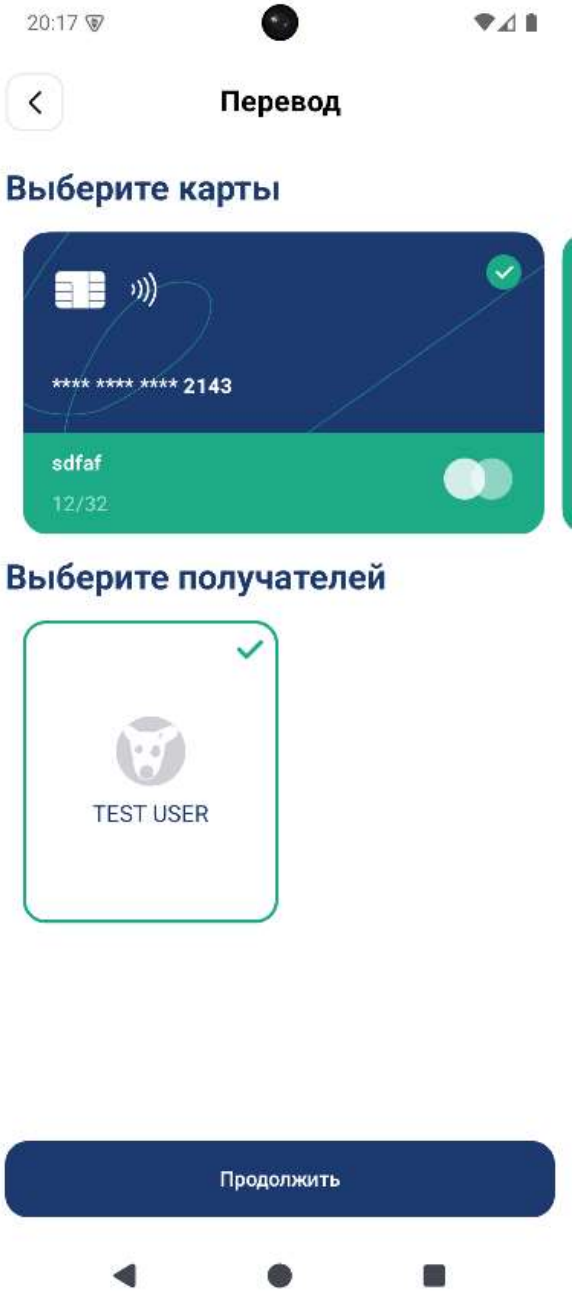

Рисунок 15 Выбор получателя и карты

Экран для начала перевода средств.

- Пользователь выбирает карту, с которой будет производиться списание.
- Выбирает получателя из списка контактов.
- После выбора нажимает «Продолжить» для перехода к вводу суммы.

Ввод суммы

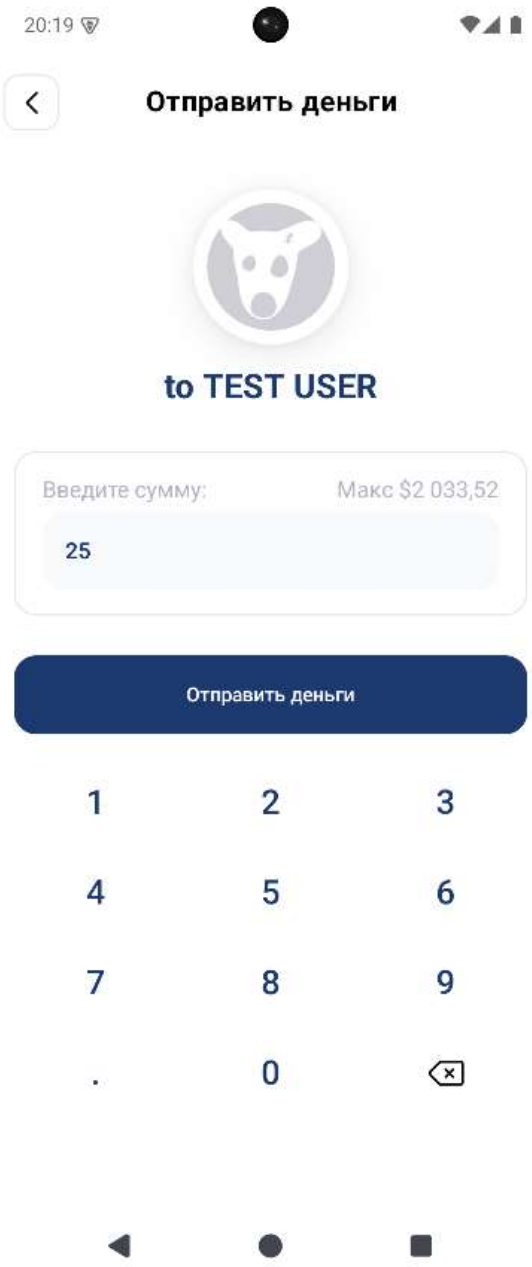

Рисунок 16 Ввод суммы перевода

Экран отправки денег.

- Отображается аватар получателя и выбранная карта.
- Пользователь вводит сумму перевода, видит максимальный доступный баланс.
- Кнопка "Отправить деньги" отправляет перевод.

Подтверждение перевода

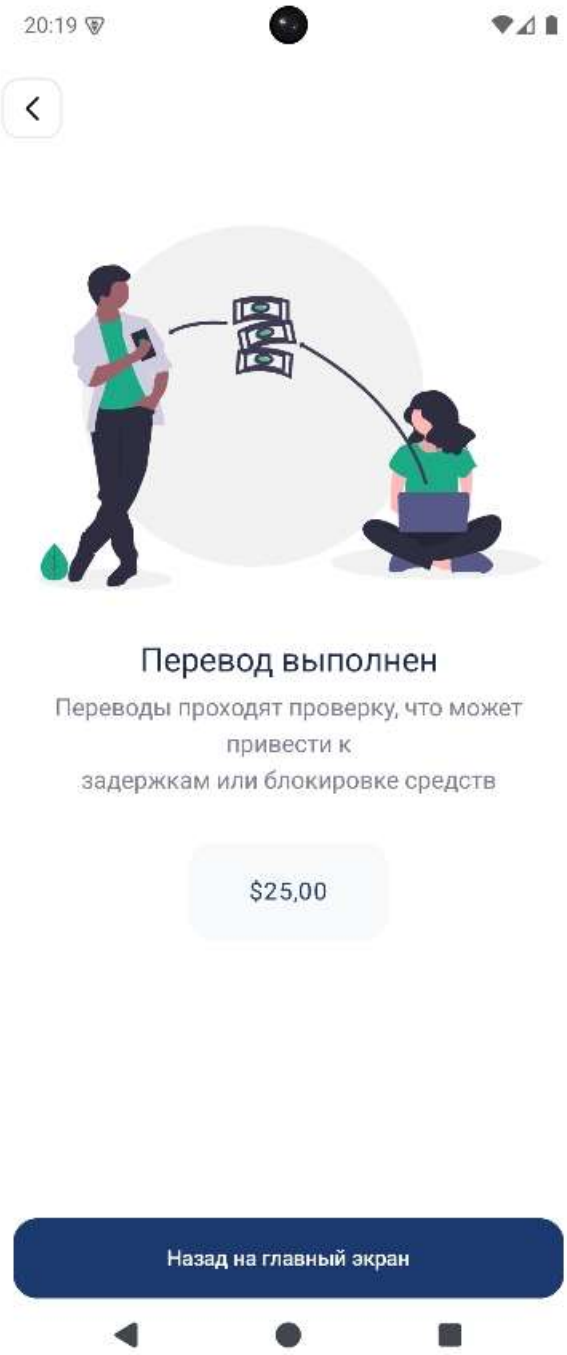

Рисунок 17 Подтверждение перевода

Экран успешного завершения перевода.

- Иллюстрация и сообщение «Перевод выполнен».
- Показывается сумма отправленных средств.
- Кнопка возврата на главный экран («Назад на главный экран»).

Снятие средств

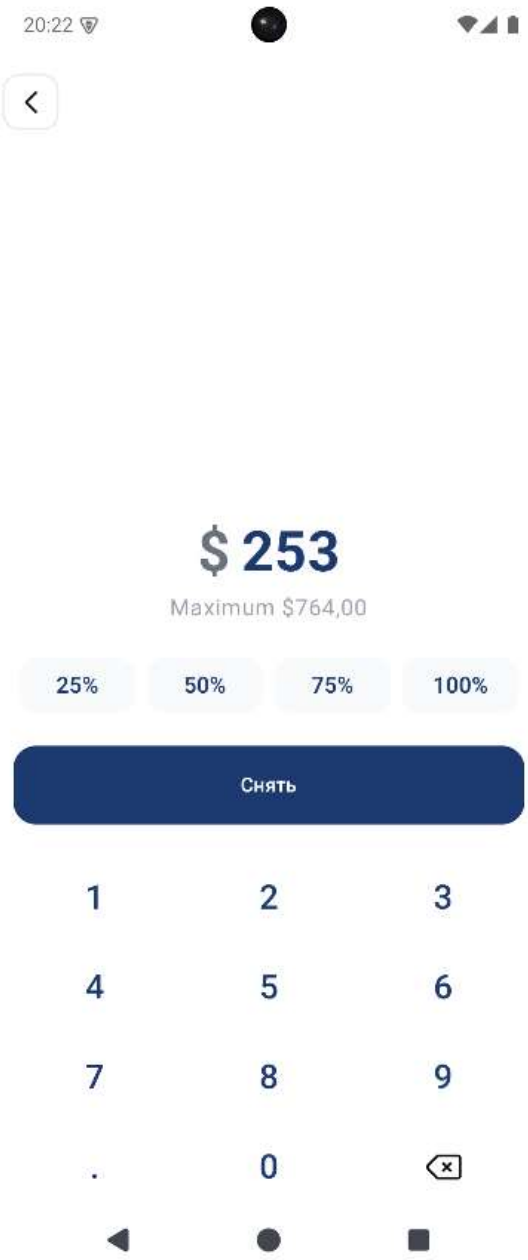

Рисунок 18 Величина снятия

Экран для снятия денег с карты.

- Крупно отображается доступная к снятию сумма.
- Можно быстро выбрать фиксированный процент или вручную ввести сумму через цифровую клавиатуру.
- Кнопка "Снять" подтверждает операцию.

Мои карты

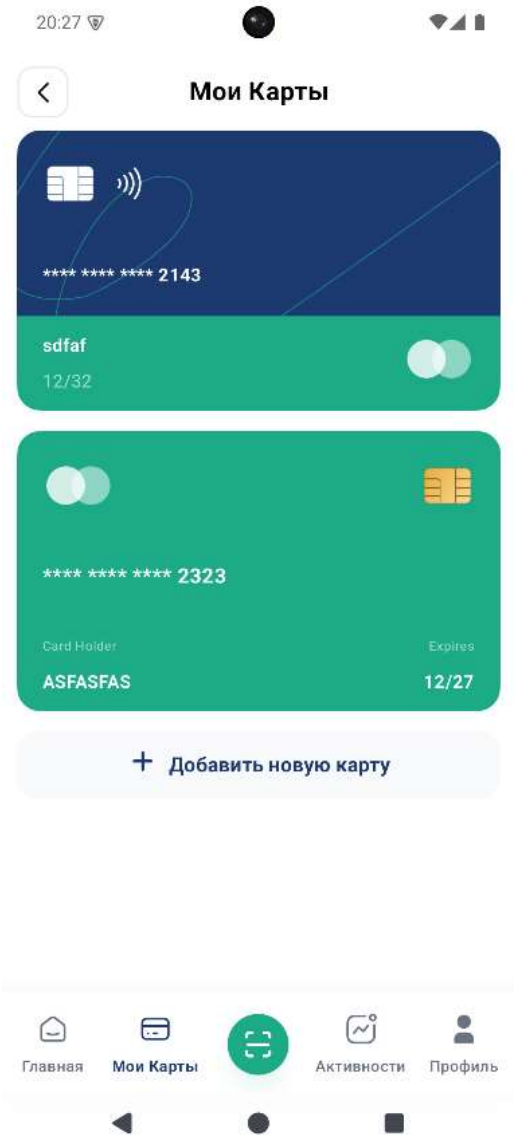

Рисунок 19 Экран карт

Экран управления банковскими картами пользователя.

- В верхней части отображается список всех добавленных карт с маскированными номерами.
- Для каждой карты указывается имя владельца и визуальный стиль карты.
- Ниже — кнопка "Добавить новую карту" для добавления новой карты.
- Экран выполнен в минималистичном стиле, поддерживает быстрый выбор карты для операций.

Редактирование карты

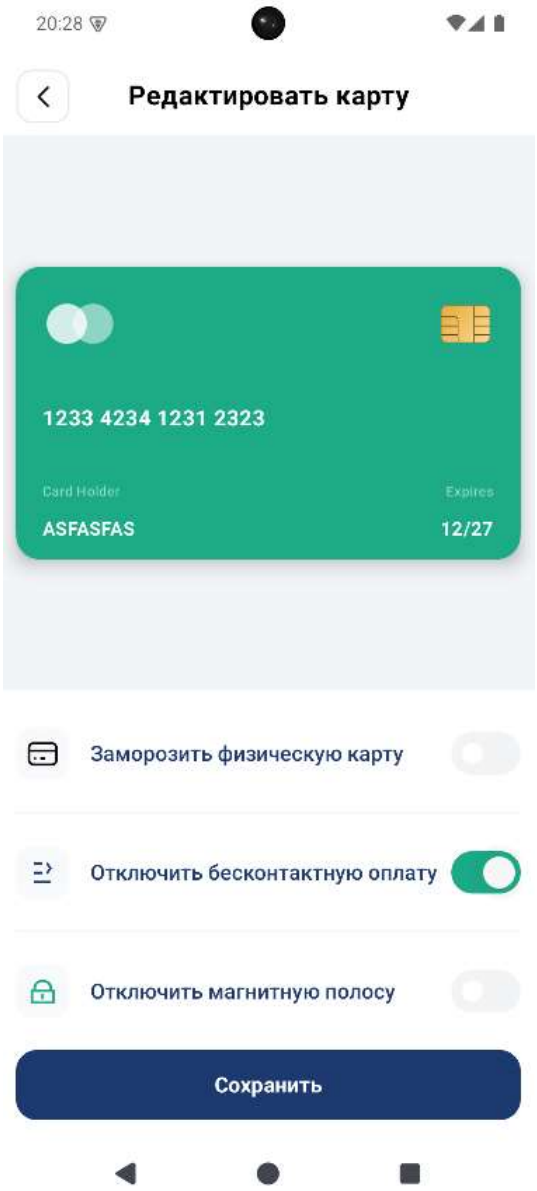

Рисунок 20 Редактирование карты

Экран редактирования выбранной карты.

- Показывает основные параметры карты (дизайн, имя владельца).
- Доступны переключатели для управления функциями карты: заморозка, включение/отключение бесконтактной оплаты, отключение магнитной полосы.
- Для применения изменений используется кнопка "Сохранить".

Сканирование QR-кода

Рисунок 21 Сканирование QR

Экран для сканирования QR-кодов.

- Центральное место занимает видоискатель для наведения камеры на QR-код.
- Поддерживает мгновенное распознавание кода для обмена контактами или получения данных карты.

Мой QR-код

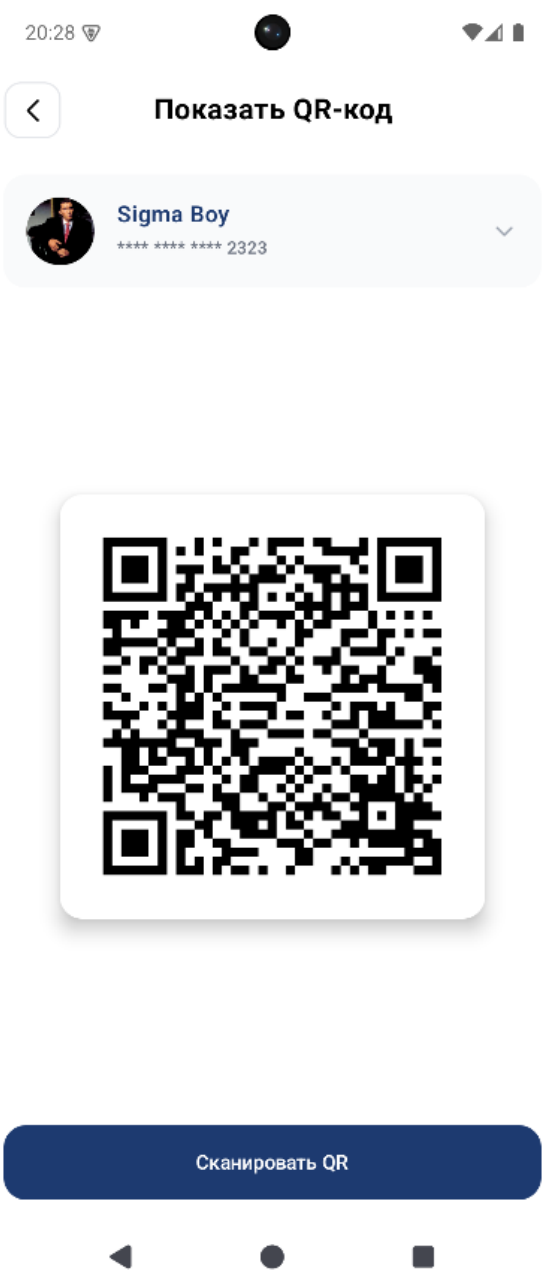

Рисунок 22 Экран QR

Экран отображения собственного QR-кода пользователя.

- В верхней части показаны имя и карта пользователя.
- Основная часть — крупный QR-код, который можно показать другому пользователю для быстрого обмена контактами или реквизитами карты.
- Дизайн минималистичный, акцент на удобстве обмена.

Профиль

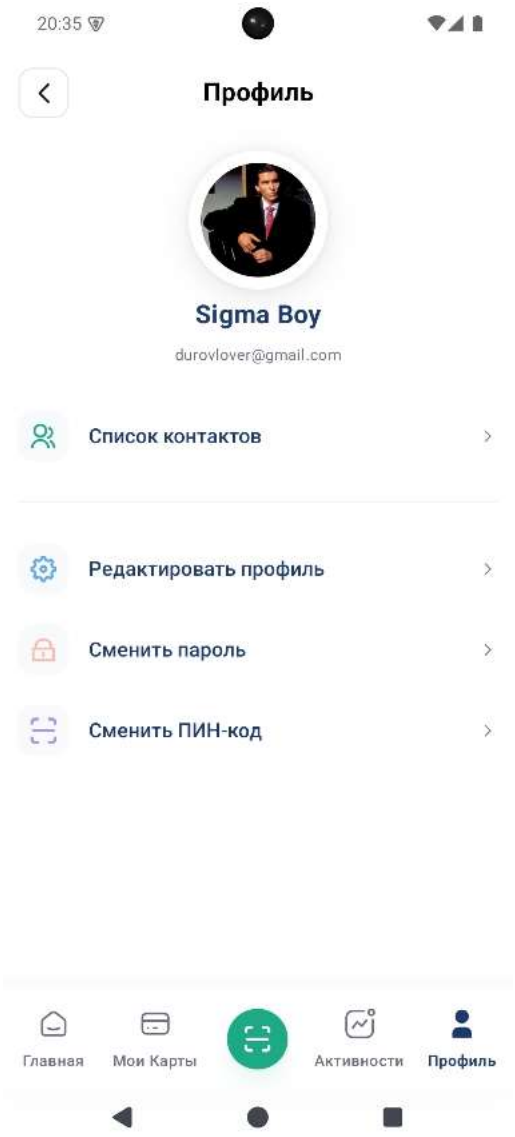

Рисунок 23 Профиль

Экран профиля пользователя отображает основную информацию о пользователе:

- В верхней части — аватар, имя и адрес электронной почты пользователя.
- Доступны быстрые переходы к списку контактов, редактированию профиля, смене пароля и изменению PIN-кода.
- Интерфейс лаконичен, с акцентом на ключевые функции управления аккаунтом.

Редактирование профиля

Рисунок 24 Редактирование профиля

Экран редактирования профиля предоставляет возможность изменить личные данные пользователя:

- Можно отредактировать имя, номер телефона и email.
- Над формой расположен аватар с возможностью смены фотографии.
- После внесения изменений пользователь подтверждает их кнопкой «Save».
- Форма организована удобно для быстрого обновления информации.

Список контактов

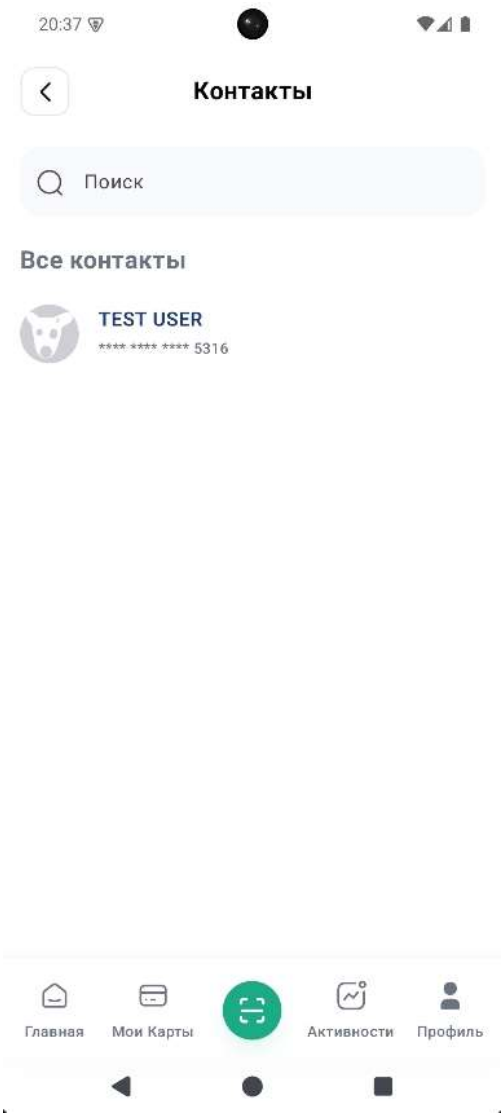

Рисунок 25 Список контактов

Экран контактов отображает всех сохранённых пользователей для быстрых операций и связи:

- В верхней части — строка поиска для фильтрации контактов по имени или номеру карты.
- Список контактов содержит имя, мини-аватар и маскированный номер карты для каждого пользователя.
- Внешний вид списка выполнен в минималистичном стиле для легкости навигации и поиска.

Реализация функций приложения

Входа пользователя (signIn)

Назначение:

Осуществляет аутентификацию пользователя по email и паролю, обеспечивает переход к дальнейшим экранам авторизации либо главному экрану приложения.

Логика работы:

1. Сохраняет email и пароль во внутренних настройках для возможного повторного использования.
2. Вызывает use-case авторизации с введёнными email и паролем.
3. При успешном входе проверяет наличие сохранённого PIN-кода:
    - Если PIN не задан — переходит к созданию PIN.
    - Если PIN есть — переходит к экрану его ввода.
4. Обработка ошибок и возврат сообщений пользователю при неудаче.

Особенности реализации:

- Вся логика реализована асинхронно с использованием viewModelScope.
- Используется централизованный обработчик ошибок.
- Переходы между экранами осуществляются через NavigationEventBus.

Листинг:

```
fun signIn(email:String, password:String) {

    //Saving in preferences because there is no endpoint for resending
the email confirmation

    userPreferences.saveUserEmail(email)

    userPreferences.saveUserPassword(password)

    viewModelScope.launch {
```

```
        val result = signInUseCase(email, password)
        result.fold(
            onSuccess = {
                val storedPin = userPreferences.getUserPin()
                if (storedPin == null) {
                    navigateToCreatePinInternal()
                } else {
                    navigateToPinInternal()
                }
            },
            onFailure = ::handleError
        )
    }
}
```

Регистрации нового пользователя (signUp)

Назначение:

Создаёт новый учётный аккаунт пользователя с последующим подтверждением email и переходом к созданию PIN-кода.

Логика работы:

1. Сохраняет email и пароль во внутренних настройках.
2. Формирует объект с персональными данными (например, полное имя).
3. Вызывает use-case регистрации.
4. При успехе инициирует переход на экран подтверждения OTP-кода, затем — к созданию PIN.
5. В случае ошибки отображает соответствующее сообщение.

Особенности реализации:

- Аутентификационные данные кэшируются для повторной отправки OTP.
- Реализована поддержка повторной отправки кода с таймером.
- Все операции проходят в корутинах для поддержки асинхронности.

Листинг:

```
fun signUp(fullName: String, email: String, password: String) {
    cachedEmail = email
    cachedPassword = password
    cachedSignUpData = SignUpData(full_name = fullName)

    //Saving in preferences because there is no endpoint for resending
the email confirmation
    userPreferences.saveUserEmail(email)
    userPreferences.saveUserPassword(password)
    viewModelScope.launch {
        val data = SignUpData(full_name = fullName)
        val result = signUpUseCase(email = email, password = password,
data = data)
        result.fold(
            onSuccess = {
                val nextRoute = NavigationRoute.CreatePinCodeScreen(
                    nextDestination = NavigationRoute.PinCodeScreen.route
                ).route
                val otpRoute = NavigationRoute.VerifyOTPScreen(
                    type = OTPType.EMAIL,
                    email = email,
                    nextDestination = nextRoute
                ).route
```

```
                navigateTo(otpRoute)
            },
            onFailure = ::handleError
        )
    }
}
```

Восстановление пароля (recover)

Назначение:

Позволяет пользователю сбросить забытый пароль через email с подтверждением по коду (OTP).

Логика работы:

1. Пользователь вводит email для восстановления.
2. Вызывается use-case отправки письма для восстановления.
3. После успешной отправки переходит на экран ввода OTP.
4. После ввода кода — открывается экран смены пароля.
5. После смены пароля — переход к экрану входа.

Особенности реализации:

- Предусмотрена возможность повторной отправки кода с учётом задержки (cooldown).
- Вся обработка ошибок централизована.
- Навигация между этапами восстановления реализована через NavigationEventBus.

Листинг:

```
fun recover(email: String) {
    viewModelScope.launch {
        val result = recoverUseCase(email)
        result.fold(
            onSuccess = {
                val nextRoute = NavigationRoute.PasswordChangeScreen(
                    nextDestination = NavigationRoute.SignInScreen.route
                ).route
                val otpRoute = NavigationRoute.VerifyOTPScreen(
                    type = OTPType.RECOVERY,
                    email = email,
                    nextDestination = nextRoute
                ).route
                navigateTo(otpRoute)
            },
            onFailure = ::handleError
        )
    }
}
```

Подтверждения OTP-кода (verifyOTP)

Назначение:

Подтверждает действия пользователя (регистрация, смена email, восстановление пароля) с помощью одноразового кода, отправленного на почту.

Логика работы:

1. Пользователь получает код OTP на email.

2. После ввода кода и нажатия "Подтвердить" вызывается соответствующий use-case.
3. При успешной верификации выполняется переход на следующий экран (например, создание PIN или смена пароля).
4. В случае ошибки отображается уведомление.

Особенности реализации:

- Реализована поддержка разных типов OTP (регистрация, восстановление, смена email).
- Поддержка повторной отправки кода с ограничением по времени.

Листинг:

```
fun verifyOTP(type: OTPType, email: String, token: String, next: String)
{
    viewModelScope.launch {
        val result = verifyOTPUseCase(type, email, token)
        result.fold(
            onSuccess = { navigateTo(next) },
            onFailure = ::handleError
        )
    }
}
```

Создания и проверки PIN-кода (createPin / loginWithPin)

Назначение:

Создаёт и проверяет персональный PIN-код для быстрого доступа к приложению.

Логика работы:

- **Создание PIN:**
    i. Пользователь вводит PIN дважды для подтверждения.
    ii. PIN сохраняется во внутреннем хранилище.
    iii. После успешного ввода переход на следующий этап (главный экран).
- **Вход по PIN:**
    i. Пользователь вводит сохранённый PIN.
    ii. Если совпадает — доступ к основному функционалу.
    iii. В случае ошибки — отображается сообщение.

Особенности реализации:

- Валидация формата PIN (5 цифр).
- Поддержка смены PIN из профиля.

Листинг:

```
fun createPin(pin: String, next: String) {
    viewModelScope.launch {
        try {
            if (pin.length == 5 && pin.all { it.isDigit() }) {
                userPreferences.saveUserPin(pin)
                navigateTo(next)
            } else {
                throw
AppException.UiResError(R.string.error_invalid_pin_format)
            }
        } catch (e: Exception) {
            handleError(e)
        }
```

```
    }
}


fun loginWithPin(pin: String) {
    viewModelScope.launch {
        try {
            val storedPin = userPreferences.getUserPin()
            if (storedPin == null) {
                navigateToCreatePinInternal()
            } else if (pin == storedPin) {
                navigateToHomeInternal()
            } else {
                throw AppException.UiResError(R.string.error_invalid_pin)
            }
        } catch (e: Exception) {
            handleError(e)
        }
    }
}
```

Управления картами (loadCards, insertCard, updateCard)

Назначение:

Позволяет добавлять, просматривать и изменять параметры банковских карт пользователя.

Логика работы:

1. Загрузка списка карт пользователя через use-case.
2. Добавление новой карты с привязкой к владельцу.

3. Изменение параметров карты (заморозка, отключение функций).
4. После успешных операций — обновление состояния и переход к нужным экранам.

Особенности реализации:

- Используется внутреннее хранилище для идентификатора пользователя.
- Все операции асинхронны, с обработкой ошибок.
- Навигация реализована через NavigationEventBus.

Листинг:

```
fun loadCards() {
    viewModelScope.launch {
        _uiState.value = CardUiState.Loading
        try {
            val userId = userPreferences.getUserId()
            if (userId != null) {
                val result = getCardsUseCase.invoke(userId)
                result.fold(
                    onSuccess = {
                        _uiState.value = CardUiState.Success(it)
                    },
                    onFailure = {
                        _uiState.value = CardUiState.Error
                        handleError(it)
                    }
                )
            } else {
                _uiState.value = CardUiState.Error
```

```
                handleError(AppException.UiResError(resId =
R.string.unknown_error))
            }
        } catch (e: AppException) {
            _uiState.value = CardUiState.Error
            handleError(e)
        }
    }
}
```

Функция перевода средств контактам

Назначение:

Осуществляет перевод денег между картами пользователей приложения.

Логика работы:

1. Пользователь выбирает карту-отправителя и получателя из списка контактов.
2. Вводит сумму перевода.
3. Формируется объект транзакции с указанием отправителя, получателя, суммы, даты и типа.
4. Вызывается use-case для записи транзакции.
5. При успешной операции отображается экран подтверждения.

Особенности реализации:

- Используется кэширование выбранной карты и контакта.
- Поддержка просмотра истории переводов и детальной информации о транзакции.

Листинг:

```
fun onAmountEntered(amount: Double) {
    val card = selectedCard.value
    val contact = selectedContact.value
    if (card == null || contact == null) {
        _uiState.value = TransferUiState.Error
        return
    }
    _uiState.value = TransferUiState.Loading
    viewModelScope.launch {
        runCatching {
            val profile = getCachedProfileUseCase.invoke()
                ?: throw AppException.UiResError(R.string.unknown_error)
            val transaction = Transaction(
                id = "",
                senderId = card.id,
                receiverId = contact.cardId,
                amount = amount,
                initiatorProfile = profile,
                createdAt = OffsetDateTime.now(),
                transactionType = TransactionType.TRANSFER
            )
           insertTransactionUseCase.invoke(transaction).getOrThrow()
        }.onSuccess {
            _uiState.value = TransferUiState.Success(amount)
            navigateTo(NavigationRoute.TransferProofScreen.route)
        }.onFailure { error ->
            _uiState.value = TransferUiState.Error
            handleError(error)
```

```
        }
    }
}
```

Снятие и депозит средств (onAmountEntered)

Назначение:

Позволяет пользователю пополнить или снять деньги с выбранной карты.

Логика работы:

1. Пользователь выбирает карту и вводит сумму.
2. Формируется объект транзакции с типом (пополнение или снятие).
3. Операция проводится через use-case, после чего обновляется баланс карты.
4. В случае успеха — переход на главный экран.

Особенности реализации:

- Валидация суммы и типа операции.
- Разделение логики снятия/пополнения по типу транзакции.

Листинг:

```
fun onAmountEntered(card: Card, amount: Double, transactionType:
TransactionType) {
    _uiState.value = WithdrawUiState.Loading
    viewModelScope.launch {


        runCatching {
            val isWithdrawLike =
TransactionType.isWithdrawLike(transactionType)
            val profile = getCachedProfileUseCase.invoke()
                ?: throw AppException.UiResError(R.string.unknown_error)
```

```
            val transaction = Transaction(
                id = "",
                senderId = if (isWithdrawLike) card.id else null,
                receiverId = if (isWithdrawLike) null else card.id,
                amount = amount,
                initiatorProfile = profile,
                createdAt = OffsetDateTime.now(),
                transactionType = transactionType
            )
            insertTransactionUseCase.invoke(transaction).getOrThrow()
        }.onSuccess {
            _uiState.value = WithdrawUiState.Success
            navigateTo(NavigationRoute.HomeScreen.route)
        }.onFailure { error ->
            _uiState.value = WithdrawUiState.Error
            handleError(error)
        }
    }
}
```

Управление профилем пользователя (loadUser, updateProfile)

Назначение:

Позволяет просматривать и изменять личные данные пользователя, в том числе аватар и email.

Логика работы:

1. Загрузка профиля (кэшированного и актуального).
2. Редактирование имени, телефона, email, аватара.

3. Если меняется email — инициируется подтверждение через OTP.
4. После успешного обновления — возврат на экран профиля.

Особенности реализации:

- Поддержка загрузки аватара с генерацией уникального имени файла.
- Обработка ошибок и навигация реализованы централизованно.

Листинг:

```
fun updateProfile(profile: Profile, newAvatarFile: File?) {
    viewModelScope.launch {
        _uiState.value = ProfileUiState.Loading

        var updatedProfile = profile

        if (newAvatarFile != null) {
            val newAvatarFileName =
FileUtils.generateTimeStampFileName(".png")
            val avatarResult = uploadAvatarUseCase.invoke(
                userId = profile.id,
                fileName = newAvatarFileName,
                file = newAvatarFile
            )

            if (avatarResult.isFailure) {

handleError(AppException.UiResError(R.string.unknown_error))
                _uiState.value = ProfileUiState.Error
                return@launch
            }
```

```
            updatedProfile = profile.copy(
                avatarUrl = "${profile.id}/${newAvatarFileName}"
            )
        }

        val profileResult = updateProfileUseCase.invoke(updatedProfile)

        if (profileResult.isFailure) {
            handleError(AppException.UiResError(R.string.unknown_error))
            _uiState.value = ProfileUiState.Error
            return@launch
        }

        val cached = getCachedProfileUseCase.invoke()
        var userResult: Result<Unit> = Result.success(Unit)

        if (cached != null && profile.email != cached.email) {
            userResult = updateUserUseCase.invoke(email = profile.email,
password = null)
            if (userResult.isFailure) {

handleError(AppException.UiResError(R.string.unknown_error))
                _uiState.value = ProfileUiState.Error
                return@launch
            }
            // Saving in preferences because there is no endpoint for
resending the email confirmation
            userPreferences.saveUserNewEmail(email = profile.email)

            navigateToVerifyOTP(
                OTPType.EMAIL_CHANGE,
```

```
                cached.email,
                NavigationRoute.VerifyOTPScreen(
                    type = OTPType.EMAIL_CHANGE,
                    email = profile.email,
                    nextDestination = NavigationRoute.ProfileScreen.route
                ).route
            )
        } else {
            navigateToProfileInternal()
        }

        loadUser()
    }
}
```

Работа с контактами (getContacts, insertContact)

Назначение:

Реализует добавление новых контактов (например, по QR-коду), просмотр и фильтрацию списка контактов.

Логика работы:

1. Загрузка списка контактов пользователя с маскировкой номеров карт.
2. Добавление нового контакта через сканирование QR-кода или вручную.
3. Сопоставление контакта с картой.
4. После успешного добавления — обновление списка или переход на главный экран.

Особенности реализации:

- Используется отдельный use-case для маскировки номеров карт.
- Поддержка быстрого добавления контакта через QR.

Листинг:

```
fun insertContact(contact: Contact) {
    viewModelScope.launch {
        runCatching {
            val contact = Contact(
                id = "",
                userId = contact.userId,
                profile = contact.profile,
                cardId = contact.cardId
            )

            if (contact.userId == contact.profile.id)
                throw AppException.UiResError(resId =
R.string.unknown_error)

            insertContactUseCase.invoke(contact)
        }.onSuccess {
            navigateTo(NavigationRoute.HomeScreen.route)
        }.onFailure {
            handleError(it)
        }
    }
}
```

Работа с QR-кодами (generateQrCodeInfo, decodeQrContent)

Назначение:

Позволяет генерировать и сканировать QR-коды для обмена контактами и реквизитами карт.

Логика работы:

1. Генерация QR-кода с закодированными идентификаторами профиля и карты.
2. Отображение собственного QR-кода для передачи другому пользователю.
3. Сканирование QR-кода и автоматическое добавление контакта на основе закодированных данных.

Особенности реализации:

- Используется сериализация/десериализация объектов через Gson.
- Все действия сопровождаются обработкой ошибок и переходами между экранами.

Листинг:

```
fun generateQrCodeInfo(profile: Profile, card: Card): String {
    return Gson().toJson(QRCodeInfo(profile.id, card.id))
}


fun decodeQrContent(qrContent: String) {
    runCatching {
        Gson().fromJson(qrContent, QRCodeInfo::class.java)
    }.onSuccess {
        val userId = userPreferences.getUserId()
        if (userId == null) {
            _uiState.value = QRUiState.Error
```

```
            handleError(AppException.UiResError(resId =
R.string.unknown_error))
            return
        }

        val contact = Contact(
            id = "",
            userId = userId,
            profile = Profile(
                id = it.id,
                fullName = "",
                email = "",
                avatarUrl = "",
                phone = ""
            ),
            cardId = it.card_id
        )

        insertContact(contact)
    }.onFailure {
        handleError(it)
    }
}
```

Заключение

В ходе практики было разработано мобильное приложение «CoPayment» с модульной архитектурой, реализующее:

- Полный цикл аутентификации: регистрация, вход, восстановление через email и PIN-код.
- Управление банковскими картами: добавление, редактирование, выбор.
- Финансовые операции: переводы между картами, история транзакций.
- Персонализация профиля: смена аватара, пароля, PIN-кода.
- Современный минималистичный дизайн и удобная навигация.
- Реализация серверной части на Supabase (PostgreSQL).

Работа позволила закрепить теоретические знания и получить практические навыки в мобильной разработке на Kotlin (Jetpack Compose), проектировании UI/UX, работе с сервером и базами данных.

Список литературы

1. Getting Started with Jetpack Compose // Official Android Developers Documentation
   URL: https://developer.android.com/jetpack/compose/getting-started (дата обращения: 04.05.2025).

2. Introduction to Supabase // Supabase Docs
   URL: https://supabase.com/docs/guides/with-your-framework/android (дата обращения: 06.05.2025).

3. Material Design Guidelines // Material.io
   URL: https://m3.material.io/ (дата обращения: 10.05.2025).

4. Android Architecture Components Overview // Official Android Developers
   URL: https://developer.android.com/topic/libraries/architecture (дата обращения: 07.05.2025).

5. Kotlin Coroutines Guide // Kotlinlang.org
   URL: https://kotlinlang.org/docs/coroutines-guide.html (дата обращения: 18.05.2025).

6. Разработка мобильных приложений на Android: руководство для начинающих // Habr
   URL: https://habr.com/ru/articles/502282/ (дата обращения: 03.05.2025).

7. Аутентификация пользователей в мобильных приложениях // Хабр
   URL: https://habr.com/ru/companies/ruvds/articles/456780/ (дата обращения: 12.05.2025).

8. Современные подходы к архитектуре Android-приложений // Хабр
   URL: https://habr.com/ru/companies/otus/articles/523412/ (дата обращения: 15.05.2025).

9. Jetpack Compose: новое поколение UI для Android // Android Broadcast URL: https://androidbroadcast.ru/materials/jetpack-compose-novoe-pokolenie-ui-dlya-android/ (дата обращения: 18.05.2025).

10. Supabase: знакомство и первые шаги // Хабр URL: https://habr.com/ru/companies/ruvds/articles/636099/ (дата обращения: 11.05.2025).