Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра информатики и программирования

Технология NFC

КУРСОВАЯ РАБОТА

студента 2 курса 241 группы
направления 02.03.02 – Математическое обеспечение и администрирование информационных систем (МОиАИС)
факультета компьютерных наук и информационных технологий (КНиИТ)
Беднова Андрея Викторовича

Научный руководитель
Старший преподаватель кафедры ИиП М.С. Портенко

Зав. кафедрой
к. ф.-м. н. М.В. Огнева

Саратов 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

NFC — стандарт беспроводной передачи данных. Эти три буквы являются аббревиатурой от «Near Field Communication», что на практике означает «коммуникация между двумя элементами, расположенными близко друг к другу».

Термин «NFC» в последнее время все чаще используется в смартфонах и платежах. Apple Pay и Samsung Pay работают в России уже несколько лет, а недавно была запущена платежная система Google Pay.

По данным группы «М.Видео-Эльдорадо» в 2019 г. продажи смартфонов с модулем NFC составили порядка 15 млн штук. В относительном выражении годовой рост составил более 20%, а доля таких устройств достигла половины рынка.[1]

Половина новых смартфонов включают в себя модуль NFC, но его присутствие не всем понятно.

Особенностью этой технологии является именно то, что два устройства должны находиться в непосредственной близости, на расстоянии нескольких сантиметров друг от друга, чтобы передача данных состоялась.

В этом и заключается главное преимущество NFC: этот вид связи очень безопасен, так как он работает на расстоянии нескольких сантиметров.

Скорость передачи информации примерно 424 Кбайт/с, хотя это и ниже, чем, например, в случае с Bluetooth, но её достаточно для отправки небольших объемов данных за долю секунды, например: данные для соединение с источником интернета.[2]

Основными проблемами при оплате с помощью мобильного устройства являются безопасность и однозначная идентификация платящего пользователя. Технология NFC исключает и то, и другое, что делает ее актуальной на современном рынке мобильного оборудования.

Поэтому для многих пользователей этот метод имеет свои преимущества.

Целью данной работы является изучение технологии NFC и реализация в мобильном приложении. Для достижения данной цели были поставлены следующие задачи:

- Изучить основные принципы NFC,
- Изучить протоколы передачи данных через NFC,
- Изучить детали разработки мобильных приложений,
- Определить подходящий язык программирования для реализации технологии,
- Реализовать возможности технологии на примере мобильного приложения, считывающего и сохраняющего в памяти устройства номера банковской карты.

1 Ключевые особенности технологии NFC

1.1 Технические характеристики

Данная модель передачи включает в себя устройство-инициатор, создающее электромагнитное поле, и измерительное устройство. Целевое устройство может быть либо активным (напр. другое бортовое устройство связи или платежный терминал) или пассивный (контактная карта или брелок для ключей). Поддерживаются существующие форматы радио-меток и бесконтактных карт. [3]

При работе с пассивной целью устройство-инициатор излучает сигнал непрерывно, а устройство-цель модулирует только создаваемое таким образом электромагнитное поле. В итоге, пассивное измерительное устройство может рассматриваться как приёмопередающее устройство, посылающее сигнал в ответ на принятый сигнал (транспондер). При работе с активной целью приборы меняют порядок передачи и отключают свое излучение, ожидая ответа.[2]

Аналогично работе систем с бесконтактными картами, в системах на основе технологии NFC связь устанавливается между двумя рамочными антеннами, находящимися в пределах ближнего поля друг друга. Связь осуществляется в пределах общедоступных и неавторизованных радиочастот в полосе ISM (промышленный, научный и медицинский радиочастотный спектр, промышленный, научный и медицинский радиочастотный спектр) с несущей частотой 13,56 МГц. Подавляющее большинство энергии в информационном сигнале лежит в пределах 14 кГц, но для амплитудной модуляции полная полоса пропускания может составлять 1,8 мГц.[3]

1.2 Формат передачи данных NDEF

NDEF (NFC Data Exchange Format – формат передачи данных через NFC) используется как формат обмена данными между устройствами и метками. Этот формат записывает все сообщения, используемые в NFC, причём неважно для карты это или для устройства. Каждое сообщение NDEF содержит одну или несколько записей NDEF. Каждый из них содержит уникальный тип записи, идентификатор, длину и поле для сообщаемой информации.

Существует несколько распространенных типов записей NDEF:

1. Обычные текстовые записи. В них можно отправить любую строку, они не содержат инструкций для целевого объекта, но содержат метаданные о языке и наборе символов.
2. URI. Такие записи содержат идентификатор ресурса, например, информацию об интернет-ссылках. Цель, получившая эту запись, откроет ее в приложении, которое может ее отобразить. Например, веб-браузере.
3. Умная запись. Содержит не только веб-ссылки, но и текстовое их описание, чтобы было понятно, что находится по этой ссылке. В зависимости от данных записи телефон может открыть информацию в нужном приложении, будь то SMS или e-mail, либо сменить настройки телефона (громкость звука, яркость экрана и т.д.).
4. Подпись. Она позволяет вам доказать, что информация, которая была или передается, является достоверной.

Можно использовать несколько видов записей в одном NDEF-сообщении.[4]

Представим сообщение как параграф, а записи – как предложения. Параграф – это конкретный информационный блок, содержащий одно или несколько предложений. В то время как предложение-это маленький кусочек информации, который содержит только одну мысль.

1.3 Структура NDEF

NDEF содержит информацию о байтовом представлении сообщений, которые могут содержать несколько записей. Каждая запись имеет заголовок, в котором находятся метаданные (тип, длина и т.д.), и информацию, которая будет отправлена.

Сообщения NDEF в основном короткие, каждый обмен состоит из одного сообщения, и каждая метка также содержит одно сообщение. Поскольку обмен данными NFC происходит, когда объект касается другого объекта или метки, то будет неудобно отправлять текст на всю книгу в одном сообщении, поэтому длина сообщения NDEF сравнима с длиной абзаца, но не всей книги.

NDEF-запись содержит информацию для пересылки и метаданные, как эту информацию интерпретировать. Каждая запись может быть разного типа, о чем объявляется в заголовке этой записи. Также в заголовке описывается какое место занимает запись в сообщении, после заголовка следует информация.

Пространство для информации в записи NDEF ограничено $2^{32} - 1$ байтами, однако можно делать цепочки записей внутри сообщения, чтобы отправить дополнительную информацию. В теории нет ограничений на NDEF-сообщения, но на практике размер сообщения ограничивается возможностями устройств или меток, участвующих в обмене информацией.[4]

1.4 Чтение NDEF-сообщения

Когда телефон на Android считывает NFC-метку, он сначала её обрабатывает и распознает, а затем передаёт данные о ней в соответствующее приложение для последующего создания *intent*. Система распознавания определяется тремя *intent*, которые перечислены в порядке важности от самой высокой до низкой:

1. ACTION_NDEF_DISCOVERED: Этот *intent* используется для запуска *activity*, если в метке содержится NDEF-сообщение. Он имеет самый высокий приоритет, и система будет запускать его в первую очередь.
2. ACTION_TECH_DISCOVERED: Если никаких *activity* для *intent* ACTION_NDEF_DISCOVERED не зарегистрировано, то система распознавания попробует запустить приложение с этим intent. Также этот *intent* будет сразу запущен, если найденное NDEF-сообщение не подходит под MIME-тип или URI, или метка совсем не содержит сообщения.
3. ACTION_TAG_DISCOVERED: Этот *intent* будет запущен, если два предыдущих *intent* не сработали.[5]

В общем случае система распознавания работает, как представлено на рисунке 1.

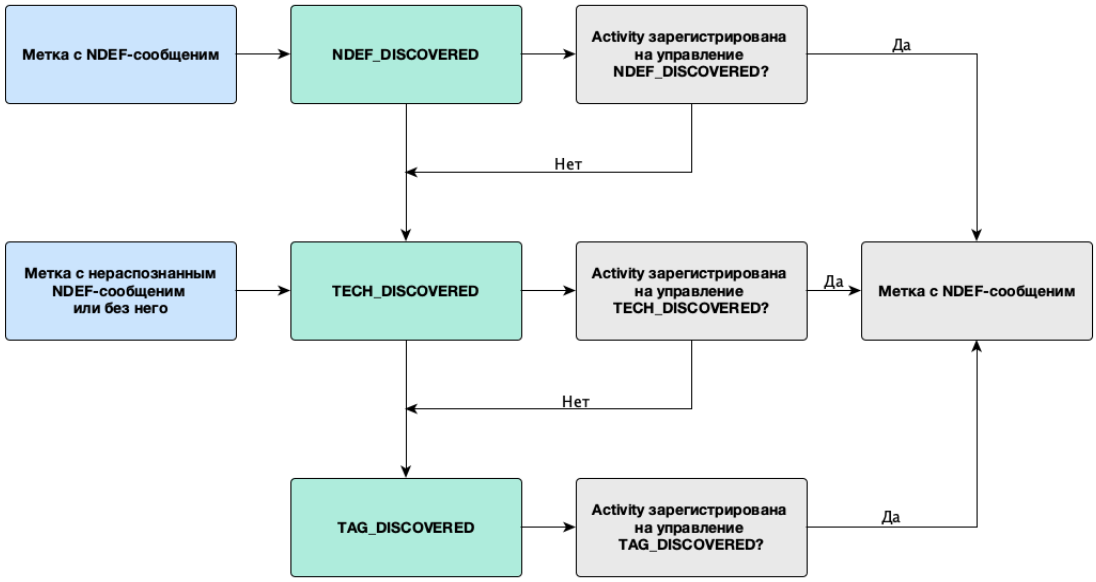

Рисунок 1 - работа системы распознавания.

2 Основы создания приложений для Android

Приложения для Android пишутся на языке программирования Java и Swift. В данной работе будет использоваться язык Java, так как это значительно более старый язык программирования, а значит будет намного больше информации и примеров, которые могут использоваться в этой работе.

Инструменты Android SDK (Software Development Kit – комплект разработки программного обеспечения) компилируют написанный код – и все требуемые файлы данных и ресурсов – в файл APK(Android Package) – программный пакет Android, который представляет собой файл архива с расширением .apk. В файле APK находится все, что требуется для работы Android-приложения, и он позволяет установить приложение на любом устройстве под управлением системы Android.[6]

2.1 Компоненты приложения

Компоненты приложения являются составляющими блоками, из которых состоит приложение для Android. Каждый компонент представляет собой отдельную точку, через которую система может войти в приложение. Не все компоненты являются точками входа для пользователя, а некоторые из них зависят друг от друга. При этом каждый компонент является самостоятельной структурной единицей и играет определенную роль – каждый из них представляет собой уникальный элемент структуры, который определяет работу приложения в целом.

Компоненты приложения можно отнести к одному из четырех типов. Компоненты каждого типа рассчитаны для определенной цели, они имеют собственный жизненный цикл, который определяет способ создания и прекращения существования компонента.

Четыре типа компонентов:

- Операции

Операция (*Activity*) представляет собой один экран с пользовательским интерфейсом. Например, операция в почтовой программе используется для отображения списка новых писем, вторая операция может быть использована для составления сообщения, а третья операция может быть использована для чтения писем. Несмотря на то, что операции совместно формируют связное взаимодействие пользователя с приложением по работе с электронной почтой, каждая из них не зависит от других операций. Любые из этих операций могут быть запущены другим приложением (если это позволяет приложение по работе с электронной почтой).

Операция относится к подклассу класса *Activity*.

- Службы

Служба (*Service*) представляет собой компонент, который работает в фоновом режиме и выполняет длительные операции, связанные с работой удаленных процессов. Служба не имеет пользовательского интерфейса. Служба может быть запущена другим компонентом, который затем будет взаимодействовать с ней, например, операцией.

Служба относится к подклассу класса *Service*.

- Поставщики контента

Поставщик контента *(Content provider)* управляет общим набором данных приложения. Данные можно хранить в файловой системе, базе данных SQLite, в Интернете или в любом другом месте постоянного хранения, к которому имеет доступ ваше приложение.

Поставщики контента также используются для чтения и записи данных, доступ к которым внешним компонентам приложение не предоставляет.

Поставщик контента относится к подклассу класса *ContentProvider*.

- Приемники широковещательных сообщений

Приемник широковещательных сообщений (*Broadcast receiver*) представляет собой компонент, который реагирует на объявления распространяемые по всей системе. Многие из этих объявлений рассылает

система – например объявление о том, что экран выключился, аккумулятор разряжен или был сделан фотоснимок.

Приемник широковещательных сообщений относится к подклассу класса *BroadcastReceiver* , а каждое такое сообщение предоставляется как объект *Intent*.

Уникальной особенностью системы Android является то, что любое приложение может запустить компонент другого приложения. [6]

2.2 Активация компонентов

Компоненты трех из четырех возможных типов – операции, службы и приемники широковещательных сообщений – активируются асинхронным сообщением, которое называется *Intent* (намерение). Объекты *Intent* связывают друг с другом отдельные компоненты во время выполнения, будь это компоненты вашего или стороннего приложения (эти объекты Intent можно представить себе в виде мессенджеров, которые посылают другим компонентам запрос на выполнение действий).

Объект *Intent* создается с помощью класса *Intent*, который описывает запрос на активацию либо конкретного компонента, либо компонента конкретного типа – соответственно, намерение *Intent* может быть явным или неявным.

Для операций и служб объект *Intent* определяет действие, которое требуется выполнить (например, просмотреть (*view*) или отправить (*send*) что-то), а также может указывать *URI* (Uniform Resource Identifier – унифицированный идентификатор ресурса) данных, с которыми это действие нужно выполнить (помимо прочих сведений, которые нужно знать запускаемому компоненту).

Компоненты четвертого типа – поставщики контента – сообщениями *Intent* не активируются. Они активируются по запросу от *ContentResolver*. Процедура определения контента (*content resolver*) обрабатывает все прямые транзакции с поставщиком контента, с тем чтобы этого не пришлось делать

компоненту, который выполняет транзакции с поставщиком. Вместо этого он вызывает методы для объекта *ContentResolver*. Это формирует слой, абстрагирующий (в целях безопасности) поставщика контента от компонента, запрашивающего информацию.[6]

2.3 Файл манифеста

Для запуска компонента приложения системе Android необходимо знать, что компонент существует. Для этого она читает файл *AndroidManifest.xml* приложения (файл манифеста). В этом файле, который должен находиться в корневой папке приложения, должны быть объявлены все компоненты приложения.

Помимо объявления компонентов приложения, манифест служит и для других целей, среди которых:

- указание всех полномочий пользователя, которые требуются приложению;
- объявление минимального уровня *API*(Application Programming Interface – Интерфейс Программирования Приложений), требуемого приложению, с учетом того, какие API-интерфейсы оно использует;
- объявление аппаратных и программных функций, которые нужны приложению или используются им, например камеры, службы Bluetooth или сенсорного экрана;
- указание библиотек API, с которыми необходимо связать приложение (отличные от API-интерфейсов платформы Android), например библиотеки *Google Maps* ;
- и многое другое.[6]

2.4 Библиотеки для работы с NFC

Библиотека *android.nfc* предоставляет доступ к функциям Near Field Communication (NFC), позволяя приложениям читать сообщения NDEF в тегах NFC. «Тег» фактически может быть другим устройством, которое определяется тег. Нам понадобится класс *NfcAdapter*. Он представляет адаптер NFC устройства, который будет необходим для выполнения операций NFC. Экземпляр можно получить с помощью *getDefaultAdapter()* или *getDefaultAdapter(android.content.Context)*.[7]

Библиотека *creditCardNfcReader* позволяет считывать номер пластиковой банковской карты и срок ее действия, а также определять тип карты (если карта поддерживает бесконтактную оплату).

Библиотека предоставляет класс *CardNfcAsyncTask*, который и будет считывать информацию с карты, а также интерфейс CardNfcAsyncTask.CardNfcInterface, который должна реализовывать Activity, работающая с NFC. Интерфейс содержит следующие методы:

- startNfcReadCard() – вызывается перед началом чтения информации с карты.
- *startNfcReadCard()* – вызывается, когда карта готова для чтения, именно в этом методе мы можем получить номер, срок действия и тип карты. Для этого нужно использовать методы экземпляра класса *CardNfcAsyncTask*: *getCardNumber(), getCardExpireDate()* и *getCardType()* соответственно.
- *doNotMoveCardSoFast()* – вызывается, когда NFC адаптер не успел считать данные с карты за время “касания”, можно использовать, чтобы попросить пользователя не двигать карту во время считывания.
- *unknownEmvCard()* – вызывается, когда найденная карта имеет неизвестный тип, и прочитать ее не удалось.
- *cardWithLockedNfc()* – вызывается, когда найденная карта имеет заблокированный NFC тег.
- *finishNfcReadCard()* – вызывается, когда чтение карты завершено.[8]

3 Реализация мобильного приложения для Android

Функциональные возможности приложения:

1. Чтение информации с пластиковой карты (при помощи NFC), и ее последующее отображение.
2. Сохранение информации о нескольких картах во внутренней памяти устройства.
3. Копирование номера карты в буфер обмена устройства.
4. Редактирование списка карт (удаление ненужных).

В ходе данной работы было разработано Android-приложение.

В приложение имеется один экран, один Activity-класс, в котором происходит вся основная работа приложения и один Java-класс для работы с базой данных.

Разработка была поделена на несколько частей:

- Настройка AndroidManifest.xml;
- Создание класса для работы с базой данных;
- Организация считывания;
- Добавление информации о карте в базу данных;
- Вывод информации на экран;
- Добавление функций копирования и удаления.

3.1 AndroidManifest

Для того, чтобы получить доступ к аппаратной части NFC и корректно считывать информацию, нужно указать в manifest-файле, что нашему приложение потребуется доступ к NFC.

```
<uses-permission android:name="android.permission.NFC" />
```

В этом же файле объявлен Intent-filter для Activity, чтобы она вызывалась, даже когда устройство находит карту, а приложение не запущено. [6]

```
<intent-filter>
  <action android:name="android.nfc.action.TECH_DISCOVERED" />
  <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Полный код AndroidManifest.xml см. приложение А.

3.2 База данных

Хранение данных в приложении осуществляется при помощи SQLite. Для управления БД(базой данных) был создан класс *DPHelper.java*, являющийся наследником для *SQLiteOpenHelper*. Этот класс предоставит нам методы для создания или обновления БД в случаях ее отсутствия или устаревания.

onCreate – метод, который будет вызван, если БД, к которой мы хотим подключиться – не существует, в нем описано создание базы данных.[9]

```
@Override
public void onCreate(SQLiteDatabase db) {
  // создаем таблицу с полями
  db.execSQL("create table mytable ("
        + "id integer primary key autoincrement,"
        + "card text,"
        + "name text,"
        + "date text" + ");");
}
```

Полный текст *DPHelper.java* приведен в приложении Б

3.3 MainActivity.java

MainActivity – главный класс приложения. В нем будет происходит обработка данных, полученных при считывании, заполнение и редактирование списка карт.

Нам потребуется вложенный класс Card для удобного сохранения и транспортировки данных карты внутри программы, и здесь же будет поле для списка карт.

```
private class Card {
  public int id;
  public String name;
  public String number;
  public String date;
  public int delete_id;
  public int copy_id;
  public Card(int id, String name, String number, String date)
  {
     this.id = id;
```

```
        this.name = name;
        this.number = number.substring(0, 4) + " "
                + number.substring(4, 8) + " "
                + number.substring(8, 12) + " "
                + number.substring(12, 16);
        this.date = date;
    }
}
private List<Card> cards;
```

В методе *onCreate()* происходит инициализация списка

cards = new ArrayList<Card>();

3.3.1 Организация считывания

Для считывания информации с карты при помощи NFC был добавлен следующий код.

Метод *onCreate(Bundle savedInstanceState)*:

```
mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
if (mNfcAdapter == null) { }
else {
  mCardNfcUtils = new CardNfcUtils(this);
  mIntentFromCreate = true;
  onNewIntent(getIntent());
}
```

В поле *NfcAdapter mNfcAdapter* помещаются данные о NFC адаптере устройства, далее, если адаптер был найден, создается экземпляр класса *CardNfcUtils,* который нужен для работы библиотеки *creditCardNfcReader*. Также ставится флаг *mIntentFromCreate* (будет объяснено позже) и вызывается метод *onNewIntent(getIntent())* метод *getIntent()* возвращает Intent-объект вызвавший нашу Activity.

Метод *onNewIntent(getIntent(Intent intent)):*

```
if (mNfcAdapter != null && mNfcAdapter.isEnabled()) {
  mCardNfcAsyncTask = new CardNfcAsyncTask.Builder(this, intent, mIntentFromCreate)
        .build();
}
```

Этот метод вызывается, когда работающее приложение получило новый *Intent.* Также в этом приложении он вызывается из *onCreate()*. В этом методе проверяется есть ли связь с NFC адаптером устройства и, если все хорошо, вызывается вызывается метод *build()* вложенного класса *Builder* класса *CardNfcAsyncTask,* конструктор класса *Builder* имеет три аргумента: Context(передается текущая Activity), Intent(передается intent) и флаг *IntentFromCreate,* который будет иметь значение *true* если текущий метод был вызван из метода *onCreate. [8]*

Класс MainActivity реализует интерфейс CardNfcAsyncTask.CardNfcInterface, о его методах говорилось в разделе 2.4, рассмотрим реализацию некоторых из них:

Метод *cardIsReadyToRead()*

```
String card = mCardNfcAsyncTask.getCardNumber();
String expiredDate = mCardNfcAsyncTask.getCardExpireDate();
String cardType = mCardNfcAsyncTask.getCardType();

if (cardType.equals(CardNfcAsyncTask.CARD_VISA))
{
  cardType = "VISA";
}
else if(cardType.equals(CardNfcAsyncTask.CARD_MASTER_CARD))
{
  cardType = "MASTER CARD";
}
else
{
  cardType = "UNKNOWN CARD";
}
```

В переменные типа String помещаются значения считанные с карты и определяется тип карты.[8]

Метод *doNotMoveCardSoFast():*

```
Toast.makeText(this,"Держите карту неподвижно",
                    Toast.LENGTH_SHORT).show();
```

Выводится *Toast*-сообщение пользователю, чтобы он не передвигал карту.

3.3.2 Добавление информации о карте в базу данных

В методе *onCreate()* происходит инициализация экземпляра класса *DBHelper*(подробнее об этом классе см. раздел 3.2)

dbHelper = new DBHelper(this);

В качестве аргумента конструктору передается контекст (текущая *Activity*).

Далее в метод *cardIsReadyToRead()* помещается следующий код:

```
ContentValues cv = new ContentValues();
SQLiteDatabase db = dbHelper.getWritableDatabase();
cv.put("card", card);
cv.put("date", expiredDate);
cv.put("name", cardType);
long rowID = db.insert("mytable", null, cv);
dbHelper.close();
```

Создается экземпляр класса *ContentValues,* который нужен для хранения набора значений. Далее подключаемся к базе данных при помощи метода *getWritableDatabase()* класса *DBHelper,* который создает соединение с базой данных и возвращает объект *SQLiteDatabase*, использующийся для работы с информацией в БД. Номер карты, срок ее действия и тип помещаются в экземпляр класса *ContentValues*, который затем передается в метод *insert()* класса *SQLiteDatabase,* данный класс принимает три аргумента: название таблицы, в которую производится вставка, второй аргумент отвечает за возможность вставки пустых столбцов, в данной работе это не требуется, поэтому отправляется *null,* третий аргумент – объект типа *ContentValues,* описанный выше, в котором должны содержаться вставляемые данные, а ключ совпадать с названием столбца. В качестве результата метод возвращает ID (Identifier – идентификатор) вставленной строки. По завершении работы с БД требуется закрыть соединение при помощи метода *close().[9]*

3.3.3 Вывод информации на экран

Вначале требуется описать главный экран приложения. Среда разработки Android Studio автоматически генерирует файл *activity_main.xml*, который и будет главным экраном приложения, после генерации в нем содержится следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:background="#E8EAF6"
  tools:context=".MainActivity">
  ...
</androidx.constraintlayout.widget.ConstraintLayout>
```

На данном этапе экран содержит только пустую *ConstraintLayout. Layout* – это такой тип *View* который может содержать внутри себя другие View-элементы. Внутрь этого layout помещен *LinearLayout*, *View* внутри этого элемента будут располагаться друг за другом по вертикали. [10]

```
<LinearLayout
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical">
...
</LinearLayout>
```

Далее внутрь *LinearLayout* помещен ImageView(это View для отображения изображений).

```
 <ImageView
  android:layout_width="match_parent"
  android:layout_height="218dp"
  android:layout_marginTop="30dp"
  android:layout_weight="0"
  android:background="#00000000"
  app:srcCompat="@drawable/pngwing_com" />
```

После *ImageView* помещен ScrollView, этот *View* позволяет “прокручивать” помещенные в него *View*-элементы.

```
<ScrollView
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:layout_marginTop="10dp"
  android:layout_weight="1">
  ...
</ScrollView>
```

Далее, внутрь *ScrollView* помещен еще один *LinearLayout,* в котором будут отображаться *View*-элементы с информацией о карте.

```
<LinearLayout
  android:id="@+id/list_cards"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
</LinearLayout>
```

Стоит отметить, что в данном *Layout* было определено поле *id (list_cards),* так как этот элемент нужен в Java части приложение, и именно по этому *id* программа сможет найти данный *View.[10]*

Графическое отображение *activity_main.xml* см. рисунок 2

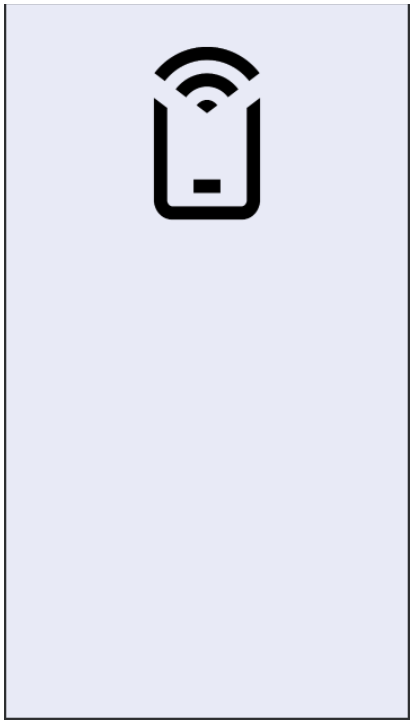

Рисунок 2. Главный экран приложения

Полный код activity_main.xml приведен в Приложении В

Далее требуется описать макет карты, чтобы наполнить его информацией о считанной карте и выводить на экран. Для этого был создан отдельный *xml*-файл *list_item.xml.*

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/list_item"
  android:layout_width="match_parent"
  android:layout_height="200dp"
  android:layout_marginLeft="20dp"
  android:layout_marginTop="10dp"
  android:layout_marginRight="20dp"
  android:layout_marginBottom="10dp"
  app:cardBackgroundColor="#7986CB"
  app:cardCornerRadius="20dp"
  app:cardElevation="5dp"
  app:cardMaxElevation="10dp">
  ...
</androidx.cardview.widget.CardView>
```

В качестве основы был выбран *CardView*, этот *View* выглядит как карточка с закругленными краями и тенью, а также допускает добавление внутрь себя других *View*-элементов. В *CardView* помещен *RelativeLayout:*

```
<RelativeLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:padding="16dp">
  ...
</RelativeLayout>
```

Данный *Layout* отображает элементы внутри себе в относительных позициях, то есть относительно себя или других элементов. Далее требуется разместить *View*-элементы для отображения информации о карте.

T*extView* для отображения информации о типе карты:

```
<TextView
  android:id="@+id/card_type"
  android:layout_width="wrap_content"
```

```
  android:layout_height="wrap_content"
  android:layout_alignTop="@+id/btn_delete"
  android:layout_alignParentStart="true"
  android:layout_marginTop="6dp"
  android:text="TextView"
  android:textColor="#FFFFFF" />
```

TextView для отображения информации о номере карты:

```
<TextView
  android:id="@+id/card_number"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerVertical="true"
  android:fontFamily="@font/pragati_narrow"
  android:text="XXXX XXXX XXXX XXXX"
  android:textColor="#FFFFFF"
  android:textSize="32sp" />
```

TextView для отображения срока действия карты:

```
<TextView
  android:id="@+id/card_date"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@+id/card_number"
  android:layout_centerHorizontal="true"
  android:fontFamily="@font/pragati_narrow"
  android:text="XX/XX"
  android:textColor="#FFFFFF" />
```

TextView для отображения надписи ExpDate(сокр. от expire date – срок действия):

```
<TextView
  android:id="@+id/textView"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@+id/card_number"
  android:layout_toStartOf="@id/card_date"
  android:fontFamily="@font/pragati_narrow"
  android:text="ExpDate:"
  android:textColor="#FFFFFF" />
```

Далее описаны две кнопки для копирования и удаления информации о карте, для этих кнопок использован элемент *ImageButton*, он позволяет изображает рисунок, которую можно использовать как кнопку.

```
<ImageButton
  android:id="@+id/btn_copy"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerVertical="true"
  android:layout_marginLeft="5dp"
  android:layout_toRightOf="@id/card_number"
  android:background="#00E3CECE"
  android:src="?attr/actionModeCopyDrawable" />
<ImageButton
  android:id="@+id/btn_delete"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentRight="true"
  android:background="#00E3CECE"
  android:src="@android:drawable/ic_delete" />
```

На этом описание макета карты окончено, графическое представление *list_item.xml* представлено на рисунке 3, полный код данного компонента представлен в Приложении Г.

Рисунок 3. Макет считанной карты

Теперь требуется описать заполнение макета информацией и его добавление в *list_cards*. Для этого в классе MainActivity было добавлено поле *LinearLayout listCards;*

А в методе onCreate() произведена его инициализация.

listCards = findViewById(R.id.list_cards);

Метод *findViewById()* возвращает View-элемент по его *id*. В этот Layout будут добавляться экземпляры карт. Далее был описан метод *private void AddCardView(Card card),* он получает экземпляр класса *Card* и выводит его в *listCards.* Для этого нужно найти id макета карты

int cardView = R.layout.list_item;

Затем, при помощи статического метода *from()* класса *LayoutInflater*, получаем экземпляр этого класса. Он нам нужен для того, чтобы создать Java-оболочку xml-файла. Далее для этого вызываем метод экземпляра класса *LayoutInflater – inflate*

View view = inflater.inflate(cardView, listCards, false);

Первым аргументом передается *id* макета, вторым – *View*-элемент, в который макет будет вставляться и третьим – *bool* значение, указывающее, требуется ли сразу вставить этот макет, в данном случае указано *false*, так как макет будет вставлен позже. Затем находятся *View*-элементы макета[11]

```
TextView cardNumberView = view.findViewById(R.id.card_number);
TextView cardDateView = view.findViewById(R.id.card_date);
TextView card_type = view.findViewById(R.id.card_type);
```

Затем в эти элементы вставляются данные карты и макет добавляется в *listCards.*

```
cardNumberView.setText(card.number);
cardDateView.setText(card.date);
card_type.setText(card.name);
listCards.addView(view);
```

Далее был описан метод *Refresh()*, необходимый для обновления списка *listCards*, из которого и вызывается метод *AddCardView()* описанный выше. Первым делом очищаются список карт в программе и список карт на экране

```
listCards.removeAllViews();
cards.clear();
```

Затем идет подключение к БД

SQLiteDatabase db = dbHelper.getWritableDatabase();

После этого при помощи метода *query()* выполняется запрос всех карт из базы, данный метод возвращает объект типа *Cursor*, данный объект представляет собой набор строк в табличном виде. Затем позиция курсора ставится на первую строку выборки и заодно проверяется есть ли там вообще элементы.[9]

```
if (c.moveToFirst()) {
...
}
```

Затем, если элементы есть, нужно определить номера столбцов по имени в выборке.

```
int idColIndex = c.getColumnIndex("id");
int nameColIndex = c.getColumnIndex("name");
int cardColIndex = c.getColumnIndex("card");
int dateColIndex = c.getColumnIndex("date");
```

Затем нужно описать цикл *do-while* для того, чтобы пройти по всем строкам таблицы

```
do {
...
} while (c.moveToNext());
```

Внутри этого цикла считывается информации из таблицы и сразу создается объект типа *Card*, он же добавляется в программный список карт и передается в метод *AddCardView()*

```
Card card = new Card(c.getInt(idColIndex), c.getString(nameColIndex),
      c.getString(cardColIndex), c.getString(dateColIndex));
cards.add(card);
AddCardView(card);
```

После завершения цикла *do-while* и условного оператора *if* необходимо закрыть соединение с БД

```
c.close();
```

На этом описание метода *Refresh()* завершено. Вызовы этого метода были добавлены в *onCreate()*, чтобы при запуске приложения отображать уже существующие в памяти карты и в метод *finishNfcReadCard()*, чтобы список карт на экране обновлялся при прочтении новой.

3.3.4 Добавление функций копирования и удаления

Для реализации функции копировании в классе *MainActivity* был описан метод *private void CopyNumber(String number).* Его код:

```
ClipboardManager clipboardManager = (ClipboardManager)
getSystemService(CLIPBOARD_SERVICE);
ClipData clipData = ClipData.newPlainText("", number);
clipboardManager.setPrimaryClip(clipData);
Toast.makeText(this,"Номер карты скопирован",
Toast.LENGTH_SHORT).show();
```

Метод *getSystemService()* с параметром *CLIPBOARD_SERVICE* возвращает объект типа *ClipboardManager*, который необходим для доступа и изменения содержимого глобального буфера обмена. Далее для того чтобы добавить что-то в буфер обмена нужно создать объект типа *ClipData* и при помощи метода *newPlainText* добавить туда данные, параметром данного метода является метка(можно передать пустую строку), а вторым текст который нужно скопировать. Затем при помощи метода *setPrimaryClip()* экземпляра класса *ClipboardManager* помещаем данные в буфер обмена и выводим Toast-сообщение, уведомляя пользователя об этом. [12]

Кнопка для копирования в макете карты уже есть, нужно добавить в метод *AddCardView()* следующий код:

```
ImageButton btn_copy = view.findViewById(R.id.btn_copy);
card.copy_id = View.generateViewId();
btn_copy.setId(card.copy_id);
btn_copy.setOnClickListener(clickCopy);
```

Сначала находится кнопка, затем ей присваивается новый уникальный *id* и этот же *id* сохраняется в программном экземпляре карты, после этого кнопке присваивается обработчик нажатия, он описывается как поле класса MainActivity, его код представлен ниже

```
View.OnClickListener clickCopy = new View.OnClickListener() {
  @Override
  public void onClick(View v) {
     for (Card item : cards) {
        if (item.copy_id == v.getId())
        {
           CopyNumber(item.number);
```

```
            break;
        }
    }
  }
};
```

Обработчик находит в программном списке карт кнопка какой именно карты была нажата и передает ее номер методу *CopyNumber()*. Также данный метод был добавлен в конец *cardIsReadyToRead()*

```
CopyNumber(card);
```

Чтобы, после прочтения при помощи NFC, номер карты сразу добавлялся в буфер обмена.

Для реализации функции удаления в метод *AddCardView()* был добавлен код, аналогичный коду для кнопки копирования

```
ImageButton btn_delete = view.findViewById(R.id.btn_delete);
card.delete_id = View.generateViewId();
btn_delete.setId(card.delete_id);
btn_delete.setOnClickListener(clickDelete);
```

Обработчик клика для кнопки удаления:

```
View.OnClickListener clickDelete = new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    for (Card item : cards) {
      if (item.delete_id == v.getId())
      {
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        int delCount = db.delete("mytable", "id = " + item.id, null);
       db.close();
        break;
      }
    }
    Refresh();
  }
};
```

Сначала так же как и для функции удаления определяется кнопка удаления для какой именно карты была нажата. Далее открывается соединение с базой данных, затем вызывается метод *delete*, которому

передается *id* удаляемой карты, после этого соединение закрывается и вызывается метод *Refresh()* для обновления списка карт.[9]

На этом разработку приложения можно считать завершенной, пример его работы можно увидеть на рисунке 4. Полный код MainActivity.java см. в Приложении Д

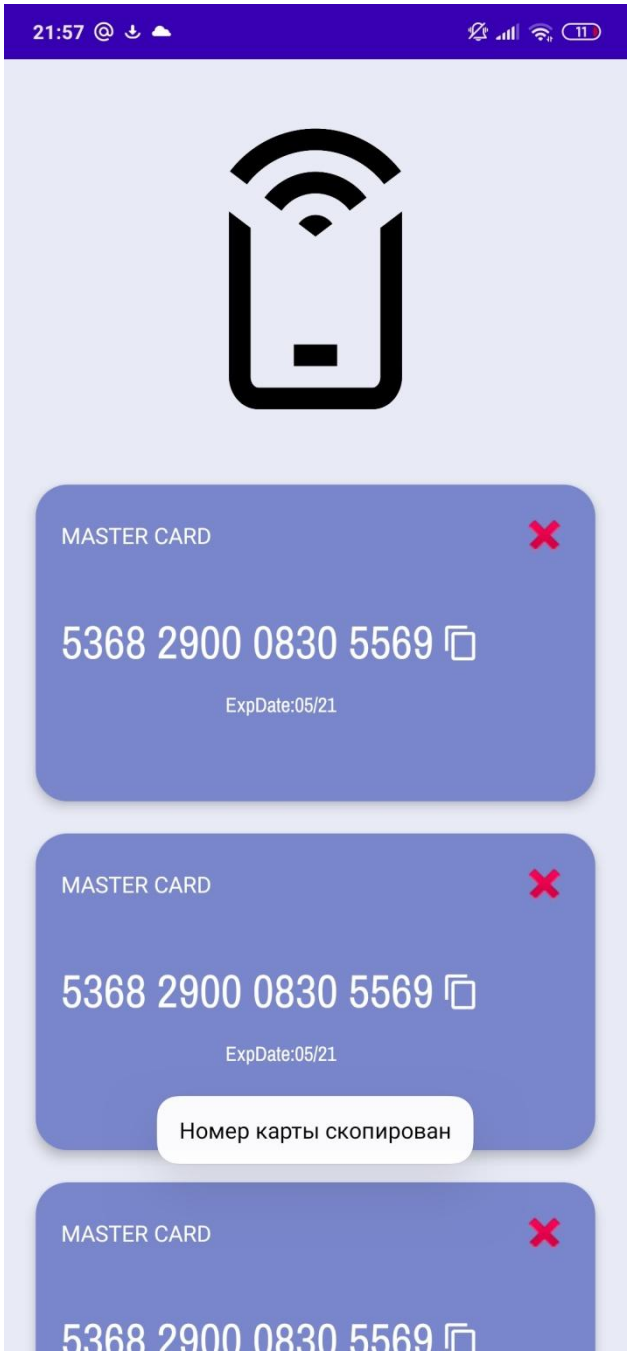

Рисунок 4. Снимок экрана работающего приложения

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были изучены основные принципы технологии NFC, формат передачи данных NDEF, детали разработки мобильных приложений для Android при помощи среды разработки Android Studio на языке Java.

Также результатом данной курсовой работы стало мобильное приложение для Android, которое может считывать информацию с пластиковой банковской карты (номер, срок действия, тип), поддерживающей технологию NFC, копировать ее и сохранять в своей памяти. Сценарии использования данного приложения разнообразны, например, пользователю для оплаты покупки в интернете не нужно будет переписывать номер карты вручную, он может просто приложить ее к задней панели телефона и номер сам скопируется в буфер обмена устройства.

Существует масса других сфер применения технологии, например, NFC-метки, запрограммированные на определенные действия, такие как подключение к Wi-Fi или установка будильника.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Каждый второй смартфон поддерживает NFC [Электронный ресурс]. – URL: https://www.itbestsellers.ru/statistics/detail.php?ID=31492 (дата обращения 24.06.2020)
2. ABOUT THE TECHNOLOGY [Электронный ресурс]. –URL: https://nfc-forum.org/what-is-nfc/about-the-technology/ (дата обращения 24.06.2020)
3. Tom Igoe, Don Coleman, and Brian Jepson. Beginning NFC.O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2014. - 12 с.
4. Используем NFC для автоматизации [Электронный ресурс]. –URL: https://nfcexpert.ru/nf-ograniheniya (дата обращения 24.06.2020)
5. Liang Zhang (Intel) [Электронный ресурс] : - URL:https://software.intel.com/en-us/android/articles/nfc-application-development-on-android-with-case-studies . (дата обращения: 26.11.2019)
6. Основы создания приложений [Электронный ресурс]. – URL:https://developer.android.com/guide/components/fundamentals?hl=ru (дата обращения 24.06.2020)
7. android.nfc [Электронный ресурс]. –URL: https://developer.android.com/reference/android/nfc/package-summary (дата обращения 24.06.2020)
8. GitHub – Credit-Card-NFC-Reader [Электронный ресурс]. –URL: https://github.com/pro100svitlo/Credit-Card-NFC-Reader (дата обращения 24.06.2020)
9. SQLiteOpenHelper [Электронный ресурс]. –URL: https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper (дата обращения 24.06.2020)
10. Макеты [Электронный ресурс]. –URL: https://developer.android.com/guide/topics/ui/declaring-layout?hl=ru (дата обращения 24.06.2020)

11. Layoutinflater [Электронный ресурс]. –URL: https://developer.android.com/reference/android/view/LayoutInflater (дата обращения 24.06.2020)
12. ClipboardManager [Электронный ресурс]. –URL: https://developer.android.com/reference/android/content/ClipboardManager (дата обращения 24.06.2020)

ПРИЛОЖЕНИЕ А Файл манифеста

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.concept">

    <uses-permission android:name="android.permission.NFC" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic launcher round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

            <intent-filter>
                <action android:name="android.nfc.action.TECH_DISCOVERED" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>

            <meta-data
                android:name="android.nfc.action.TECH_DISCOVERED"
                android:resource="@xml/nfc_tech_filter" />

        </activity>
        <meta-data
            android:name="preloaded_fonts"
            android:resource="@array/preloaded_fonts" />
    </application>

</manifest>
```

ПРИЛОЖЕНИЕ Б Класс для работы с базой данных

DPHelper.java

```
package com.example.concept;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context) {
        // конструктор суперкласса
        super(context, "myDB", null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // создаем таблицу с полями
        db.execSQL("create table mytable ("
                + "id integer primary key autoincrement,"
                + "card text,"
                + "name text,"
                + "date text" + ");");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{

    }
}
```

ПРИЛОЖЕНИЕ В Главный экран

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#E8EAF6"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/imageView"
            android:layout_width="match_parent"
            android:layout_height="218dp"
            android:layout_marginTop="30dp"
            android:layout_weight="0"
            android:background="#00000000"
            app:srcCompat="@drawable/pngwing_com" />

        <ScrollView
            android:id="@+id/scrollView2"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginTop="10dp"
            android:layout_weight="1">

            <LinearLayout
                android:id="@+id/list_cards"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="vertical">

            </LinearLayout>
        </ScrollView>

    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

ПРИЛОЖЕНИЕ Г Макет карты

list_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/list_item"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp"
    android:layout marginRight="20dp"
    android:layout_marginBottom="10dp"
    app:cardBackgroundColor="#7986CB"
    app:cardCornerRadius="20dp"
    app:cardElevation="5dp"
    app:cardMaxElevation="10dp">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="16dp">


        <TextView
            android:id="@+id/card_number"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerVertical="true"
            android:fontFamily="@font/pragati_narrow"
            android:text="XXXX XXXX XXXX XXXX"
            android:textColor="#FFFFFF"
            android:textSize="32sp" />

        <TextView
            android:id="@+id/card_date"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@+id/card_number"
            android:layout_centerHorizontal="true"
            android:fontFamily="@font/pragati_narrow"
            android:text="XX/XX"
            android:textColor="#FFFFFF" />

        <ImageButton
            android:id="@+id/btn_delete"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:background="#00E3CECE"
            android:src="@android:drawable/ic_delete" />

        <ImageButton
            android:id="@+id/btn_copy"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerVertical="true"
            android:layout_marginLeft="5dp"
            android:layout_toRightOf="@id/card_number"
```

```
            android:background="#00E3CECE"
            android:src="?attr/actionModeCopyDrawable" />

        <TextView
            android:id="@+id/textView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@+id/card_number"
            android:layout_toStartOf="@id/card_date"
            android:fontFamily="@font/pragati_narrow"
            android:text="ExpDate:"
            android:textColor="#FFFFFF" />



        <TextView
            android:id="@+id/card_type"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignTop="@+id/btn_delete"
            android:layout_alignParentStart="true"
            android:layout_marginTop="6dp"
            android:text="TextView"
            android:textColor="#FFFFFF" />

    </RelativeLayout>
</androidx.cardview.widget.CardView>
```

ПРИЛОЖЕНИЕ Д Основное Activity

MainActivity.java

```
package com.example.concept;

import android.content.ClipData;
import android.content.ClipboardManager;
import android.content.ContentValues;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.nfc.NfcAdapter;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.ImageButton;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import com.pro100svitlo.creditCardNfcReader.CardNfcAsyncTask;
import com.pro100svitlo.creditCardNfcReader.utils.CardNfcUtils;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity implements
CardNfcAsyncTask.CardNfcInterface{

    private NfcAdapter mNfcAdapter;
    private CardNfcUtils mCardNfcUtils;
    private boolean mIntentFromCreate;
    private CardNfcAsyncTask mCardNfcAsyncTask;
    private DBHelper dbHelper;
    private LinearLayout listCards;

    // обработчик клика для кнопки удаления
    View.OnClickListener clickDelete = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            SQLiteDatabase db = dbHelper.getWritableDatabase();

            for (Card item : cards) {
                if (item.delete_id == v.getId())
                {
                    int delCount = db.delete("mytable", "id = " + item.id,
null);
                    break;
                }
            }

            db.close();
            Refresh();

        }
    };
```

```
    // обработчик клика для кнопки копирования
    View.OnClickListener clickCopy = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            for (Card item : cards) {
                if (item.copy_id == v.getId())
                {
                    CopyNumber(item.number);
                    break;
                }
            }
            Refresh();

        }
    };

    private class Card {
        public int id;
        public String name;
        public String number;
        public String date;
        public int delete_id;
        public int copy_id;
        public Card(int id, String name, String number, String date)
        {
            this.id = id;
            this.name = name;
            this.number = number.substring(0, 4) + " "
                    + number.substring(4, 8) + " "
                    + number.substring(8, 12) + " "
                    + number.substring(12, 16);
            this.date = date;
        }
    }

    private List<Card> cards;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        if (mNfcAdapter == null) { }
        else {
            mCardNfcUtils = new CardNfcUtils(this);
            //next few lines here needed in case you will scan credit card
when app is closed
            mIntentFromCreate = true;
            onNewIntent(getIntent());
        }

        // view в которую мы будем вставлять карты
        listCards = findViewById(R.id.list_cards);

        // коллекция для карт
        cards = new ArrayList<Card>();

        // создаем объект для создания и управления версиями БД
        dbHelper = new DBHelper(this);

        // вызываем метод отрисовки существующих в памяти карт
```

```
        Refresh();
    }

    @Override
    protected void onResume() {
        super.onResume();
        mIntentFromCreate = false;
        if (mNfcAdapter != null && !mNfcAdapter.isEnabled()) {
            //show some turn on nfc dialog here. take a look in the samle ;-)
        } else if (mNfcAdapter != null) {
            mCardNfcUtils.enableDispatch();
        }
    }

    @Override
    public void onPause() {
        super.onPause();
        if (mNfcAdapter != null) {
            mCardNfcUtils.disableDispatch();
        }
    }

    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
        if (mNfcAdapter != null && mNfcAdapter.isEnabled()) {
            //this - interface for callbacks
            //intent = intent :)
            //mIntentFromCreate - boolean flag, for understanding if
onNewIntent() was called from onCreate or not
            mCardNfcAsyncTask = new CardNfcAsyncTask.Builder(this, intent,
mIntentFromCreate)
                    .build();
        }


    }

    @Override
    public void startNfcReadCard() {

    }

    @Override
    public void cardIsReadyToRead() {
        String card = mCardNfcAsyncTask.getCardNumber();
        String expiredDate = mCardNfcAsyncTask.getCardExpireDate();
        String cardType = mCardNfcAsyncTask.getCardType();

        if (cardType.equals(CardNfcAsyncTask.CARD_VISA))
        {
            cardType = "VISA";
        }
        else if(cardType.equals(CardNfcAsyncTask.CARD_MASTER_CARD))
        {
            cardType = "MASTER CARD";
        }
        else
        {
            cardType = "UNKNOWN CARD";
        }
```

```
        // создаем объект для данных
        ContentValues cv = new ContentValues();

        // подключаемся к БД
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        cv.put("card", card);
        cv.put("date", expiredDate);
        cv.put("name", cardType);
        // вставляем запись и получаем ее ID
        long rowID = db.insert("mytable", null, cv);

        dbHelper.close();

        CopyNumber(card);
    }

    @Override
    public void doNotMoveCardSoFast() {
        Toast.makeText(this,"Не перемещайте карту",
Toast.LENGTH_SHORT).show();
    }

    @Override
    public void unknownEmvCard() {

    }

    @Override
    public void cardWithLockedNfc() {

    }

    @Override
    public void finishNfcReadCard() {
        Refresh();
    }

    // обновляет экран
    private void Refresh() {
        // отчищаем список карт
        listCards.removeAllViews();
        cards.clear();

        // подключаемся к БД
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        // делаем запрос всех данных из таблицы mytable, получаем Cursor
        Cursor c = db.query("mytable", null, null, null, null, null, null);

        // ставим позицию курсора на первую строку выборки
        // если в выборке нет строк, вернется false
        if (c.moveToFirst()) {

            // определяем номера столбцов по имени в выборке
            int idColIndex = c.getColumnIndex("id");
            int nameColIndex = c.getColumnIndex("name");
            int cardColIndex = c.getColumnIndex("card");
            int dateColIndex = c.getColumnIndex("date");

            do {
                Card card = new Card(c.getInt(idColIndex),
c.getString(nameColIndex),
```

```
                        c.getString(cardColIndex),
c.getString(dateColIndex));

                cards.add(card);

                AddCardView(card);

                // переход на следующую строку
            } while (c.moveToNext());
        }
        c.close();
    }

    // добавление карты на экран
    private void AddCardView(Card card) {
        // Находим ID макета карты
        int cardView = R.layout.list_item;

        // создаем Inflater, для создания Java оболочки макета
        LayoutInflater inflater = LayoutInflater.from(this);

        // создаем программную оболочку макета карты
        View view = inflater.inflate(cardView, listCards, false);

        // находим  View компоненты макета
        TextView cardNumberView = view.findViewById(R.id.card_number);
        TextView cardDateView = view.findViewById(R.id.card_date);
        ImageButton btn_delete = view.findViewById(R.id.btn_delete);
        ImageButton btn_copy = view.findViewById(R.id.btn_copy);
        TextView card_type = view.findViewById(R.id.card_type);


        // генерируем и устанавливаем ID кнопки удаления для конкретной карты
        card.delete_id = View.generateViewId();
        btn_delete.setId(card.delete_id);


        // генерируем и устанавливаем ID кнопки копирования для конкретной
карты
        card.copy_id = View.generateViewId();
        btn_copy.setId(card.copy_id);
        // устанавливаем для кнопок обработчик нажатия
        btn_delete.setOnClickListener(clickDelete);
        btn_copy.setOnClickListener(clickCopy);

        // вставляем данные в макет карты
        cardNumberView.setText(card.number);
        cardDateView.setText(card.date);
        card_type.setText(card.name);

        // добавляем карту на экран
        listCards.addView(view);
    }
    // копирует аргумент в буфер обмена
    private void CopyNumber(String number) {
        ClipboardManager clipboardManager = (ClipboardManager)
getSystemService(CLIPBOARD_SERVICE);
        ClipData clipData = ClipData.newPlainText("", number);
        clipboardManager.setPrimaryClip(clipData);
        Toast.makeText(this,"Номер карты скопирован",
Toast.LENGTH_SHORT).show();
    }
}
```