Chapter 4

Technical Implementation

Note: as the workflow progresses this Chapter is subject to change. As such, there will be a section for future extensions at the end
Overview

- Task: Create Application Interface
    - Requirements: Obtains enough user information to generate AND SAVE a person's digital ID
        - Implementation: React-Native + Expo node.js application
            - Specifications: Accessibility, Clarity, and Minimalist Design
    - Requirements: Has an interface to allow users to use NFC HCE (Host Card Emulation) Services with their ID
        - Implementation: Android
          Uses Android Studio to manage a custom APDU service …
        - Implementation: IOS (TODO)
- Task: Create a custom NFC reader
    - Requirements: (Android) Reads the APDU Protocol
        - Implementation: Uses an Arduino Uno + PN532 NFC Reader chip to read from the Arduino IDE serial monitor
    - Requirements: (IOS) Reads the ISO-type4 …
- Task: Create vendor applications to use custom data with
    - Requirements: Using a separate web application as a Vendor, NFC data should be passed to the Vendor in a way that real vendors would actually use
        - Implementation: The system reads the TNumber from a `.txt` file, validates it against corresponding JSON access data (Cafeteria, Dorms, or Library), and securely passes the verified information to the vendor application.
- Task: Create GeneratedID
    - Requirements: Use necessary fields to build a digital ID
        - Implementation: React script
            - Specification: Personal Image, Name, Date of Birth, TNumber

Creating the Application Interface

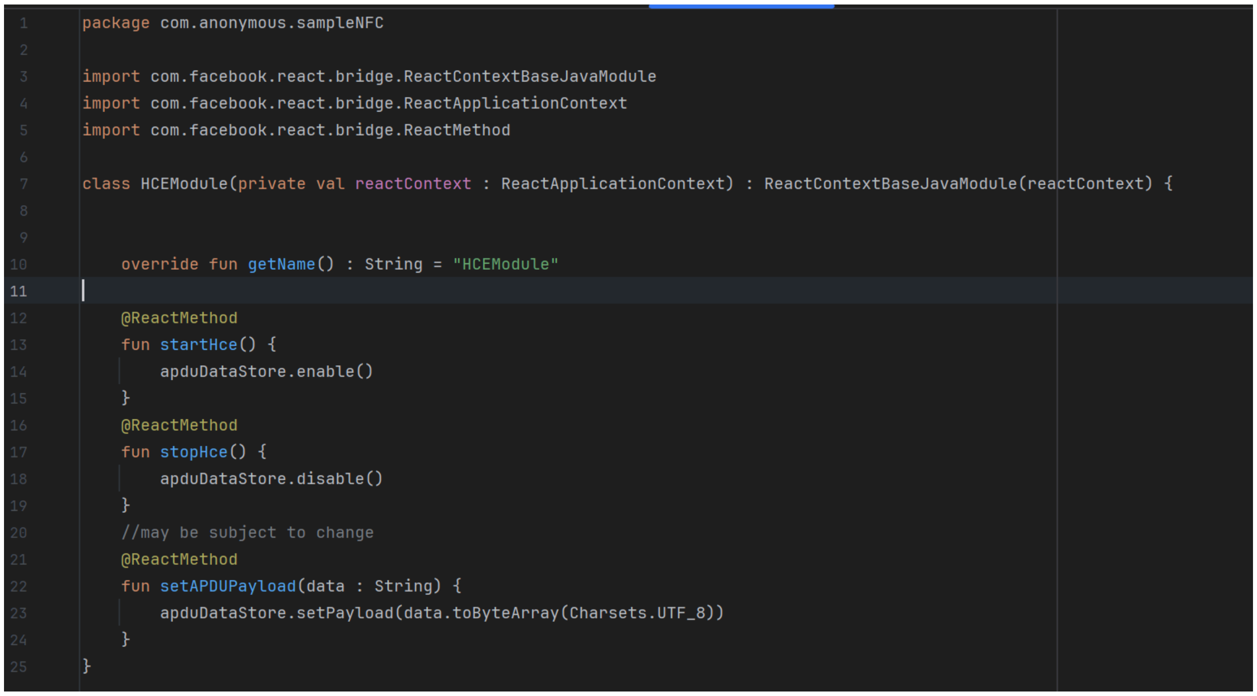
Our application, named TNPass, uses a react native environment backed by the expo runtime library. Currently, the main page uses a react bits wave background (figure1) With a simple description of the app.
Image 1 here

The data fields page allows the user to enter their information that is then used to generate a user's digital ID image. The NFC Start test allows the user to begin the NFC HCE process (handled based on device version and operating system)

The shared stylesheet so far looks like:
…

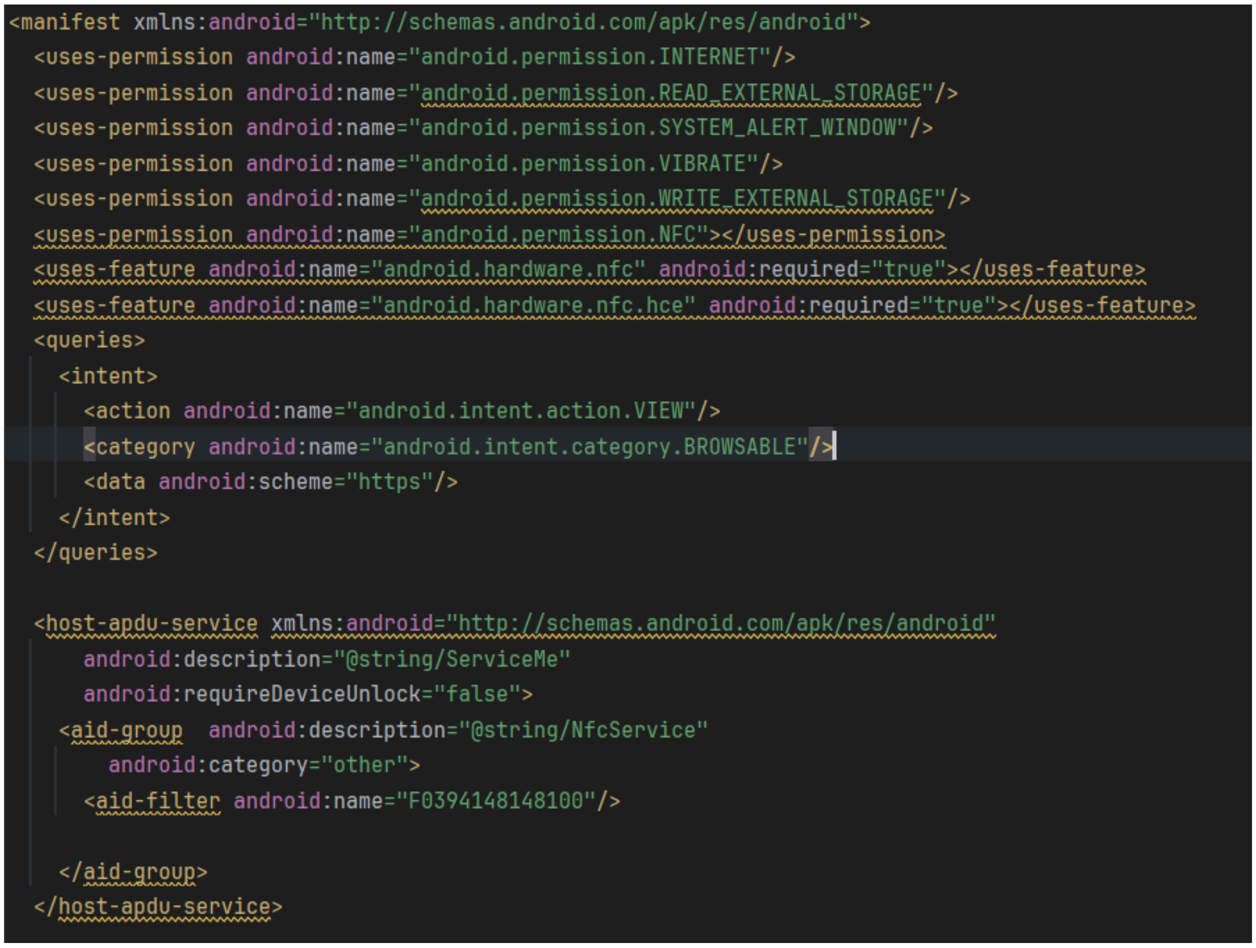
Be mindful of resource requirements when using libraries such as opengl

Android:
When the NFC start button is pressed, the following HCEModule(Figure 2) will allow the APDU service to begin listening for SELECT_AID.
Figure 2

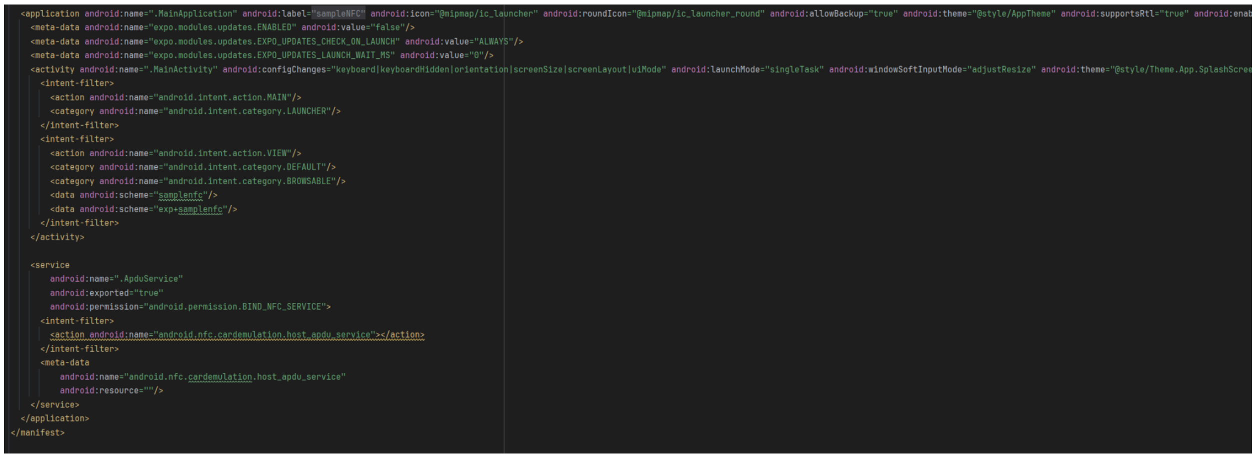


```kotlin
package com.anonymous.sampleNFC

import com.facebook.react.bridge.ReactContextBaseJavaModule
import com.facebook.react.bridge.ReactApplicationContext
import com.facebook.react.bridge.ReactMethod

class HCEModule(private val reactContext : ReactApplicationContext) : ReactContextBaseJavaModule(reactContext) {


    override fun getName() : String = "HCEModule"

    @ReactMethod
    fun startHce() {
        apduDataStore.enable()
    }
    @ReactMethod
    fun stopHce() {
        apduDataStore.disable()
    }
    //may be subject to change
    @ReactMethod
    fun setAPDUPayload(data : String) {
        apduDataStore.setPayload(data.toByteArray(Charsets.UTF_8))
    }
}
```

SELECT_AID is a APDU formatted byte array that uses the AID (application identifier) defined in the AndroidManifest.xml
Figure3



```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
  <uses-permission android:name="android.permission.VIBRATE"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.NFC"></uses-permission>
  <uses-feature android:name="android.hardware.nfc" android:required="true"></uses-feature>
  <uses-feature android:name="android.hardware.nfc.hce" android:required="true"></uses-feature>
  <queries>
    <intent>
      <action android:name="android.intent.action.VIEW"/>
      <category android:name="android.intent.category.BROWSABLE"/>
      <data android:scheme="https"/>
    </intent>
  </queries>

  <host-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
      android:description="@string/ServiceMe"
      android:requireDeviceUnlock="false">
    <aid-group  android:description="@string/NfcService"
        android:category="other">
      <aid-filter android:name="F0394148148100"/>

    </aid-group>
  </host-apdu-service>
```

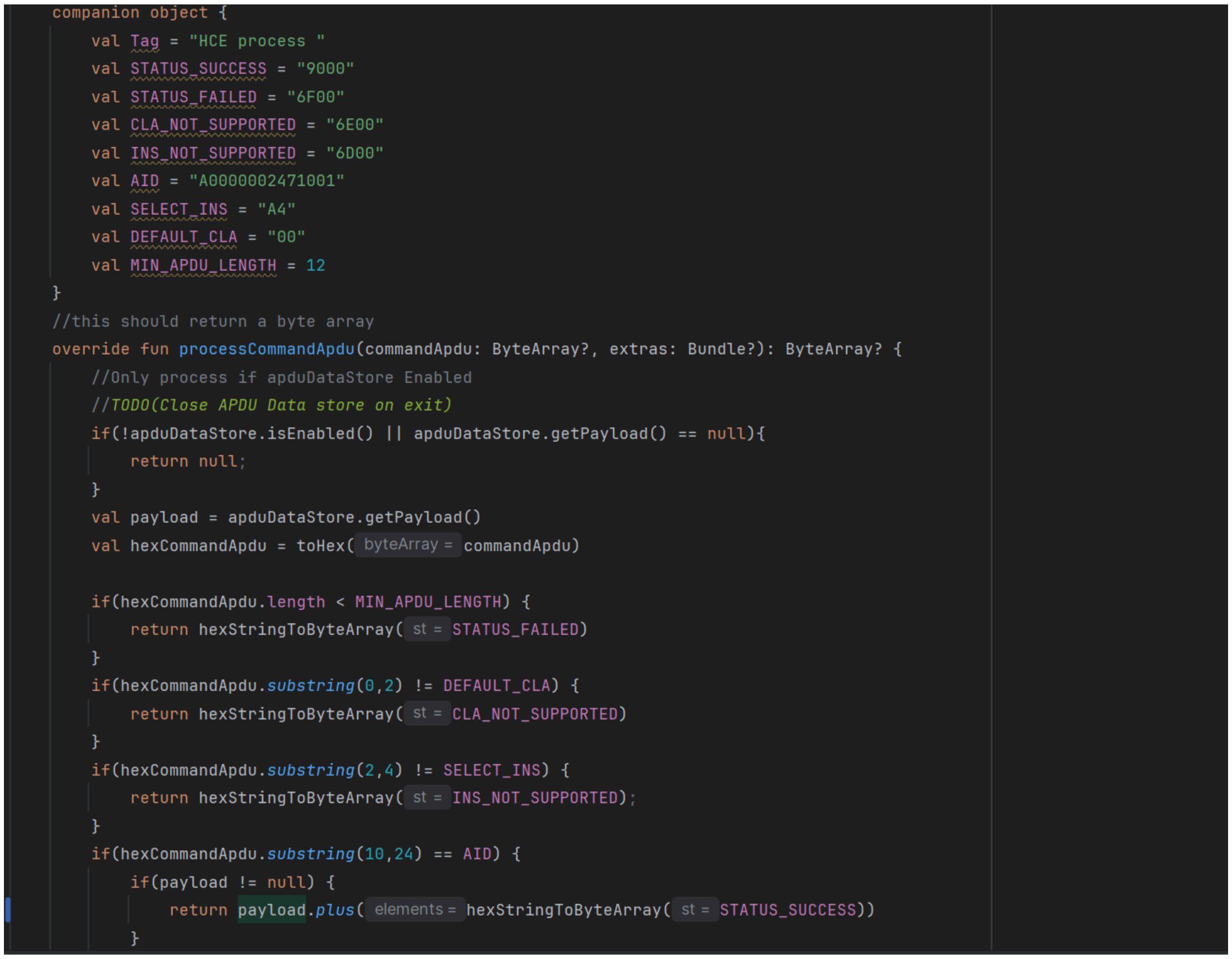


```xml
  <application android:name=".MainApplication" android:label="sampleNFC" android:icon="@mipmap/ic_launcher" android:roundIcon="@mipmap/ic_launcher_round" android:allowBackup="true" android:theme="@style/AppTheme" android:supportsRtl="true" android:enab
    <meta-data android:name="expo.modules.updates.ENABLED" android:value="false"/>
    <meta-data android:name="expo.modules.updates.EXPO_UPDATES_CHECK_ON_LAUNCH" android:value="ALWAYS"/>
    <meta-data android:name="expo.modules.updates.EXPO_UPDATES_LAUNCH_WAIT_MS" android:value="0"/>
    <activity android:name=".MainActivity" android:configChanges="keyboard|keyboardHidden|orientation|screenSize|screenLayout|uiMode" android:launchMode="singleTask" android:windowSoftInputMode="adjustResize" android:theme="@style/Theme.App.SplashScree
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
      <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="samplenfc"/>
        <data android:scheme="exp+samplenfc"/>
      </intent-filter>
    </activity>

    <service
        android:name=".ApduService"
        android:exported="true"
        android:permission="android.permission.BIND_NFC_SERVICE">
      <intent-filter>
        <action android:name="android.nfc.cardemulation.host_apdu_service"></action>
      </intent-filter>
      <meta-data
          android:name="android.nfc.cardemulation.host_apdu_service"
          android:resource=""/>
    </service>
  </application>
</manifest>
```

AndroidManifest.xml
TODO(highlight APDUService aspect)

Using an Arduino Uno attached to the PN532 NFC Scanner (canvas1). The service will use the SELECT_AID command it receives to initiate and exchange with the scanner(figure4)

Figure4



```kotlin
    companion object {
        val Tag = "HCE process "
        val STATUS_SUCCESS = "9000"
        val STATUS_FAILED = "6F00"
        val CLA_NOT_SUPPORTED = "6E00"
        val INS_NOT_SUPPORTED = "6D00"
        val AID = "A0000002471001"
        val SELECT_INS = "A4"
        val DEFAULT_CLA = "00"
        val MIN_APDU_LENGTH = 12
    }
    //this should return a byte array
    override fun processCommandApdu(commandApdu: ByteArray?, extras: Bundle?): ByteArray? {
        //Only process if apduDataStore Enabled
        //TODO(Close APDU Data store on exit)
        if(!apduDataStore.isEnabled() || apduDataStore.getPayload() == null){
            return null;
        }
        val payload = apduDataStore.getPayload()
        val hexCommandApdu = toHex( byteArray = commandApdu)

        if(hexCommandApdu.length < MIN_APDU_LENGTH) {
            return hexStringToByteArray( st = STATUS_FAILED)
        }
        if(hexCommandApdu.substring(0,2) != DEFAULT_CLA) {
            return hexStringToByteArray( st = CLA_NOT_SUPPORTED)
        }
        if(hexCommandApdu.substring(2,4) != SELECT_INS) {
            return hexStringToByteArray( st = INS_NOT_SUPPORTED);
        }
        if(hexCommandApdu.substring(10,24) == AID) {
            if(payload != null) {
                return payload.plus( elements = hexStringToByteArray( st = STATUS_SUCCESS))
            }
```

APDUService Process

Creating A Custom NFC Reader

That is to say, the exchange begins with the reader. When the reader discovers a valid NFC communication device it will send a byte array with the following format(figure5)
Figure5

```
+-----+-----+-----+-----+-----+-------------------------+-----+
| CLA | INS | P1  | P2  | Lc  | DATA                    | Le  |
+-----+-----+-----+-----+-----+-------------------------+-----+
| 00  | A4  | 04  | 00  | XX  | AID                     | 00  |
+-----+-----+-----+-----+-----+-------------------------+-----+
```

Sources=android - How to send a command APDU to a HCE device? - Stack Overflow and Smart card application protocol data unit - Wikipedia

More specifically:

```
byte AID[] = {0xF0, 0x39, 0x41, 0x48, 0x14, 0x81, 0x00}; //AID/Data
unsigned int aidLen = sizeof(AID);
byte* SELECT_AID = new byte[6 + aidLen];
```

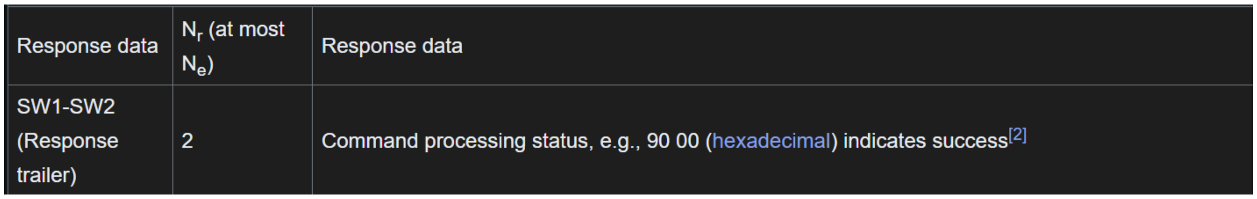
The AID is preset for the reader and the SELECT_AID is distinctly 6 bytes greater in size than the AID.

```
  SELECT_AID[0] = (byte)0x00; //CLA
  SELECT_AID[1] = (byte)0xA4; //INS
  SELECT_AID[2] = (byte)0x04; //P1
  SELECT_AID[3] = (byte)0x00; //P2
  SELECT_AID[4] = (byte)(aidLen); //Lc
  SELECT_AID[aidLen+5] = (byte)0x00; //Le special max length case
```

Mirroring the APDU formatting

```
Serial.println("Board Found, starting APDU exchange");
//Build SELECT AID
memcpy(SELECT_AID + 5, AID, aidLen);
//Read
//TODO()CyberSecurity Requirement, generate a private key for authentication

 uint8_t response[255];
 uint8_t responseLength = sizeof(response); //can do this because uint8_t is the size of one byte

 if(nfc.inDataExchange(SELECT_AID, sizeof(SELECT_AID), response, &responseLength)) {
  Serial.print("Reponse: ");
  //response data should be user data
  nfc.PrintHex(response, responseLength);
  Serial.println();
  //check the last four bytes
  unsigned int tst;
  getTrailingBytes(4, tst, response, responseLength);
  if(tst == STATUS_SUCCESS){
  //call function to send data to vendor
  Vendor(response, responseLength, suffix);
  }
 }
 else {
  Serial.println("APDU exchange failed");
 }

 delay(1000);
```

The AID itself is copied into the data portion of the select AID APDU. nfc.inDataExchange() allows for an immediate data exchange between the reader and the mobile device. However, the response format is not the same as the APDU command format.

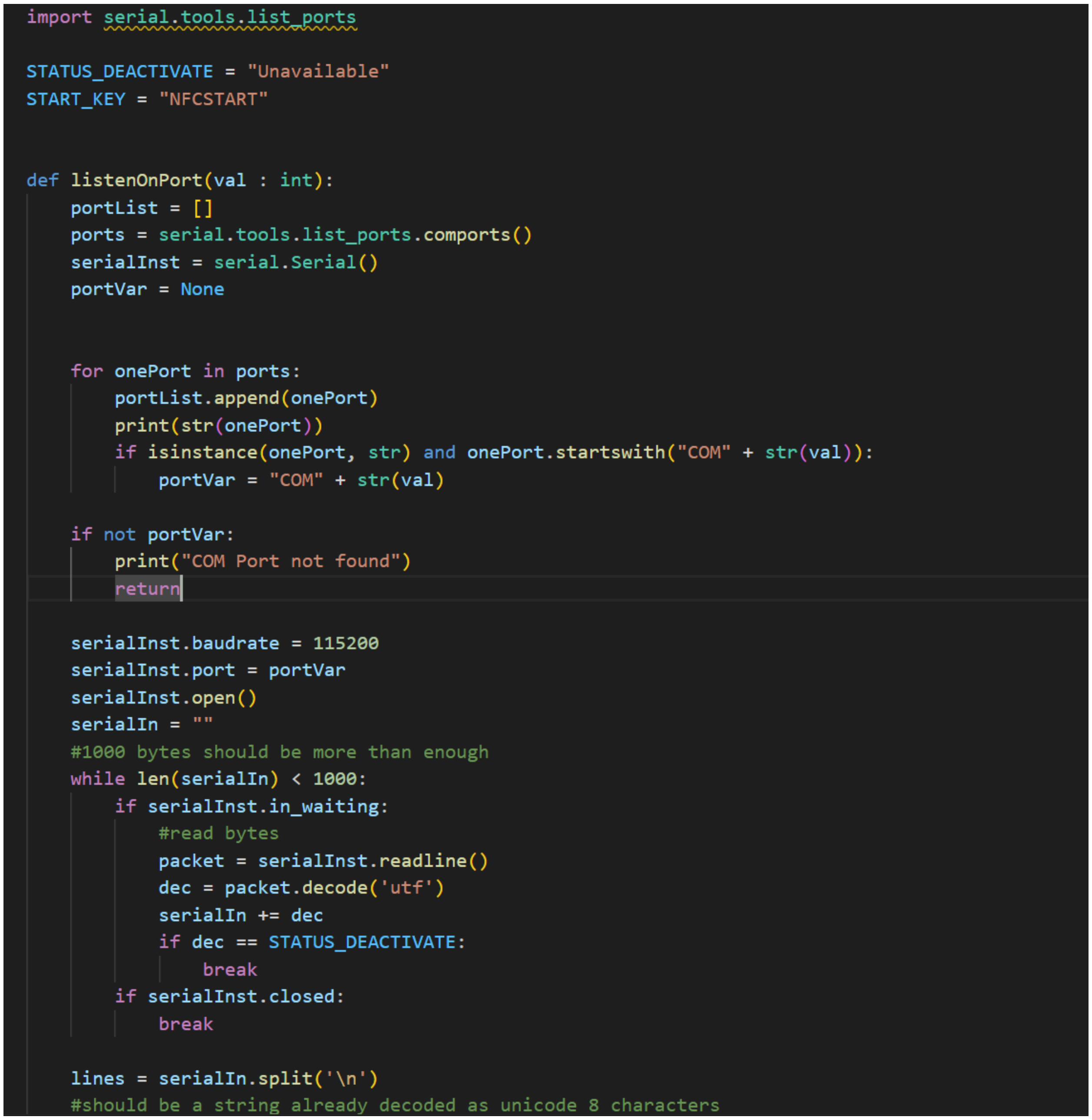


Response data	N_r (at most N_e)	Response data
SW1-SW2 (Response trailer)	2	Command processing status, e.g., 90 00 (hexadecimal) indicates success[2]

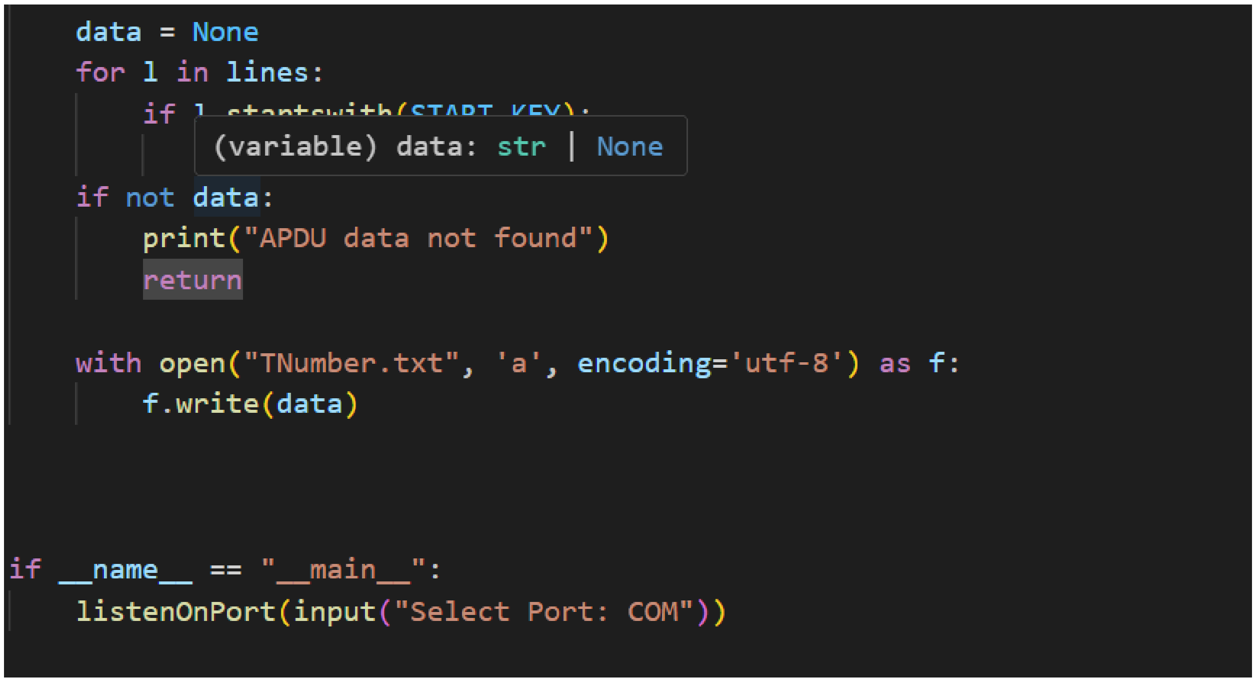
Using an Le value of 0, the Ne data response size is unbounded but should be no greater than 255 bytes. The last 2 bytes represent the command processing status (STATUS_SUCCESS = 0X9000, STATUS_FAILED = 0x6F00), so these can be checked for success.

Creating Vendor Applications

The main issue surrounding a vendor application is hosting a separate application instance that receives data from the scanner. The following script allows this communication via .txt File



```python
import serial.tools.list_ports

STATUS_DEACTIVATE = "Unavailable"
START_KEY = "NFCSTART"


def listenOnPort(val : int):
    portList = []
    ports = serial.tools.list_ports.comports()
    serialInst = serial.Serial()
    portVar = None


    for onePort in ports:
        portList.append(onePort)
        print(str(onePort))
        if isinstance(onePort, str) and onePort.startswith("COM" + str(val)):
            portVar = "COM" + str(val)

    if not portVar:
        print("COM Port not found")
        return

    serialInst.baudrate = 115200
    serialInst.port = portVar
    serialInst.open()
    serialIn = ""
    #1000 bytes should be more than enough
    while len(serialIn) < 1000:
        if serialInst.in_waiting:
            #read bytes
            packet = serialInst.readline()
            dec = packet.decode('utf')
            serialIn += dec
            if dec == STATUS_DEACTIVATE:
                break
        if serialInst.closed:
            break

    lines = serialIn.split('\n')
    #should be a string already decoded as unicode 8 characters
```



```
    data = None
    for l in lines:
        if l.startswith(START_KEY):
            (variable) data: str | None
    if not data:
        print("APDU data not found")
        return

    with open("TNumber.txt", 'a', encoding='utf-8') as f:
        f.write(data)



if __name__ == "__main__":
    listenOnPort(input("Select Port: COM"))
```

The actual vendor application will be using TNumber.txt to update parameters of access, permission, and accessibility. The vendor is implemented using raw html, css, and javascript.

Creating the GeneratedID

//I don’t really have any info on this

Glueing it all together

Future Plans

In the future, we could use a database system like supabase to allow users to log in and link their data to their accounts. This way, a user can have a saved payload ready to go whenever they launch the application. Additionally, TNumber.txt can be converted to JSON format and only send a primary key. This would allow us to limit the amount of information available to each vendor to only the amount necessary. The Vendor could also be modernized to use React as well.