Uniwersytet Wrocławski
Wydział Fizyki i Astronomii

Kamil Janus

Aplikacja mobilna do skanowania paragonów i zarządzania budżetem na urządzenia typu Android

Mobile application for scanning receipts and budget management for Android devices

Praca inżynierska wykonana pod kierunkiem
prof. dr hab. Zbigniewa Kozy

Wrocław 2024

Spis treści

1. Wstęp

1.1 Wprowadzenie

W dzisiejszym dynamicznym środowisku gospodarczym skuteczne zarządzanie finansami i kontrola wydatków stają się kluczowymi elementami sukcesu zarówno dla globalnych przedsiębiorstw, jak i dla konsumentów indywidualnych. Podczas gdy duże firmy zatrudniają w tym celu sztab księgowych i analityków, zwykły konsument nie może sobie pozwolić na taki luksus. Na szczęście dzięki rozwojowi technologii mobilnych i cyfrowych nie jest już on skazany na gromadzenie paragonów i faktur oraz skrupulatne ich analizowanie z kalkulatorem w ręku. Na rynku jest dostępnych wiele rozwiązań w formie aplikacji mobilnych, desktopowych czy chociażby serwisów internetowych mających go w tym wspomóc. Jednym z przykładów są aplikacje skanujące paragony, które korzystają z rozpoznawania tekstu w celu odczytania informacji o dokonanym zakupie. Motywacją do napisania niniejszej pracy inżynierskiej były osobiste doświadczenia autora z takimi właśnie mobilnymi aplikacjami do skanowania paragonów, z których narodziła się chęć spersonalizowania i poszerzenia ich funkcjonalności poprzez stworzenie własnej ich wersji.

W pierwszym rozdziale niniejszej pracy przybliżam historię i zagadnienia techniczne związane z optycznym rozpoznawaniem znaków oraz systemem Android, formułuję cel i założenia pracy, wymagania funkcjonalne i niefunkcjonalne tworzonej aplikacji oraz analizuję dostępne rozwiązania. Drugi rozdział stanowi opis wykorzystanych narzędzi programistycznych, technologii i bibliotek. W trzecim rozdziale omawiam implementację oraz sposób działania opracowanych przez siebie rozwiązań. W rozdziale czwartym opisuję interfejs użytkownika. Piąty rozdział poświęciłem opisowi testów aplikacji. Pracę wieńczy rozdział szósty, opisujący możliwe ścieżki rozwoju stworzonego przeze mnie rozwiązania oraz podsumowujący realizację zakładanych celów.

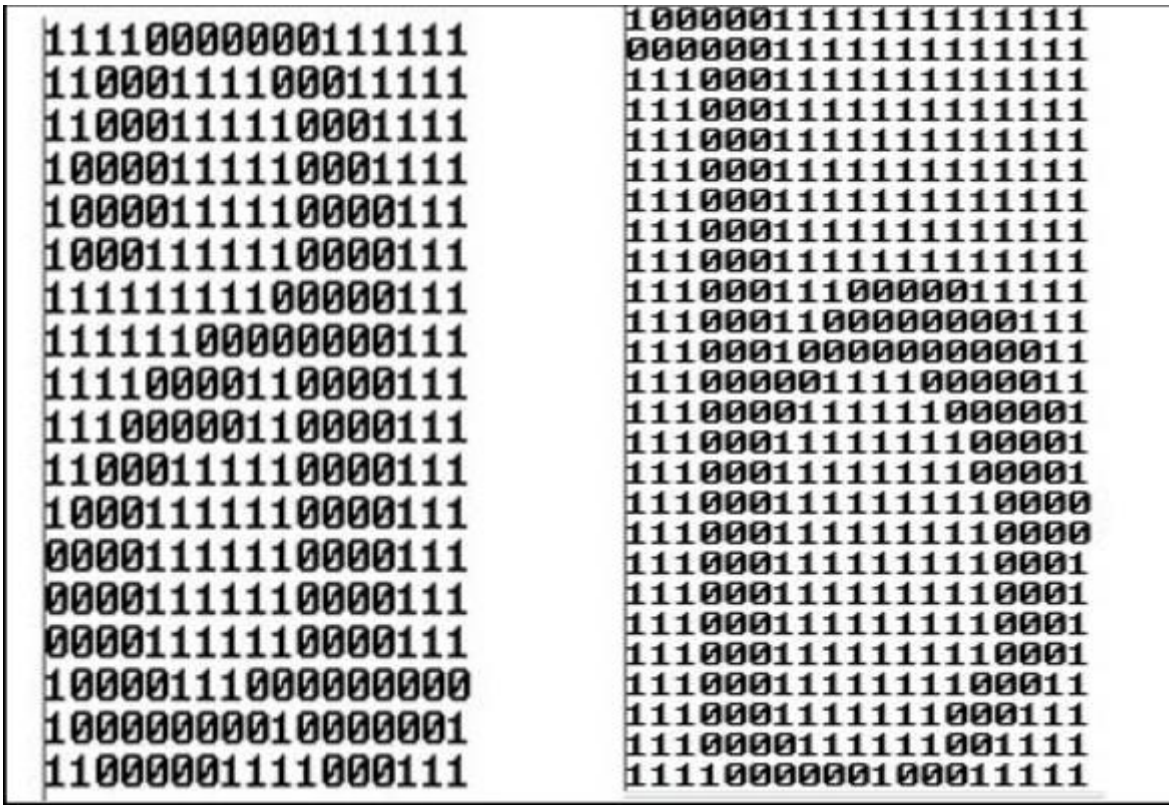
1.2. Optyczne rozpoznawanie znaków

Optical Character Recogniton (optyczne rozpoznawanie znaków), w skrócie OCR, to dziedzina technik informatycznych zajmująca się konwersją tekstu zapisanego na zdjęciach, dokumentach papierowych i innych nośnikach analogowych na tekst cyfrowy. Pierwsze próby automatycznego rozpoznawania tekstu zostały podjęte już w 1913 roku przez dr. Edmunda Fourniera d'Albe z Birmingham University, który stworzył urządzenie zwane optofonem. Miało ono formę skanera, który przesuwany nad tekstem konwertował rozpoznane litery na dźwięk. Optofon miał służyć osobą niewidomym jako narzędzie umożliwiające im „czytanie" wydrukowanego tekstu [1].

Wczesne wersje systemów OCR w formie znanej nam współcześnie sięgają lat 60. XX wieku. Wykorzystywane modele były trenowane zdjęciami pojedynczych liter i były w stanie rozpoznać tylko tekst napisany wykorzystaną podczas treningu czcionką. W latach 70. wynalazca Raymond Kurzweil skomercjalizował system „omni-font OCR", który mógł przetwarzać tekst napisany niemal dowolną czcionką. Na początku pierwszej dekady XXI wieku technologia OCR stała się dostępna online w formie serwisów internetowych [2].

Obecnie swoje usługi w zakresie OCR oferuje wiele firm takich jak między innymi Google LLC, Amazon.com Inc. czy Microsoft Corporation. Są one dostarczane najczęściej w formie Application Programming Interface (w skrócie: API, z ang. „interfejs programowania aplikacji"), umożliwiających programistom budowanie aplikacji wykorzystujących stworzone przez dostawcę usługi rozwiązania w zakresie rozpoznawania tekstu.

Pierwszym krokiem w procesie rozpoznawania tekstu jest przetworzenie obrazu. Może to obejmować normalizację, skalowanie, usuwanie szumów i inne techniki poprawy jego jakości. Kolejnym krokiem jest segmentacja, czyli identyfikacja obszarów zawierających tekst. Następnie wykorzystuje się kontrast pomiędzy znakami a tłem. Widoczny na zdjęciu tekst jest rozbijany na pojedyncze znaki, a one z kolei konwertowane są do macierzy binarnych, będących ich reprezentacją. Przykładowy wynik tej operacji przedstawiono na Rysunku 1. Uzyskane macierze są przetwarzane za pomocą algorytmów, które określają ich statystyczne podobieństwo do znaków znajdujących się w bazie danych modelu i na tej podstawie określają, jaki znak reprezentuje dana macierz. Uzyskany w ten sposób tekst może też zostać poddany dalszej analizie, uwzględniającej rozpoznanie języka, oraz analizę struktury słów i zdań w celu zagwarantowania jak najwyższej poprawności.



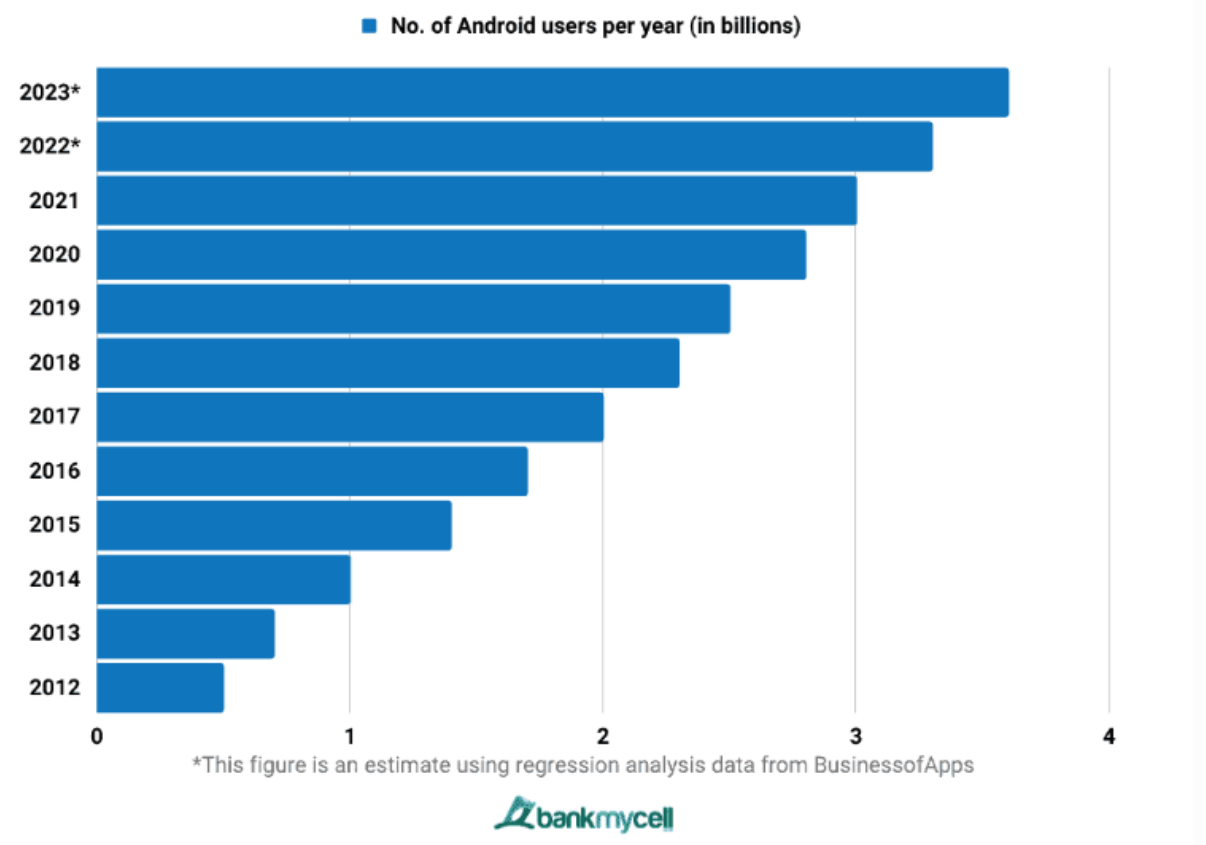
Rysunek 1. Binarna macierz wyodrębnionych znaków, gdzie 1 reprezentuje tło, a 0 reprezentuje pixel znaku [3]

1.3. System Android

Android rozpoczął swoje istnienie w 2003 roku jako system operacyjny amerykańskiej firmy Android Inc. Został stworzony z myślą o aparatach cyfrowych. Zaledwie rok później zadecydowano o zmianie jego urządzeń docelowych z aparatów na smartfony. W 2005 firma wraz z tworzonym przez nią systemem została kupiona przez amerykańskie przedsiębiorstwo Google Inc. (obecnie Google LLC), gdzie projekt był dalej rozwijany, aby w październiku 2008 roku zadebiutować na smartfonie T-Mobile G1, wyprodukowanym przez firmę HTC Corporation. W 2012 roku Android został najpopularniejszym system na urządzenia mobilne, pokonując system iOS firmy Apple Inc. [4].

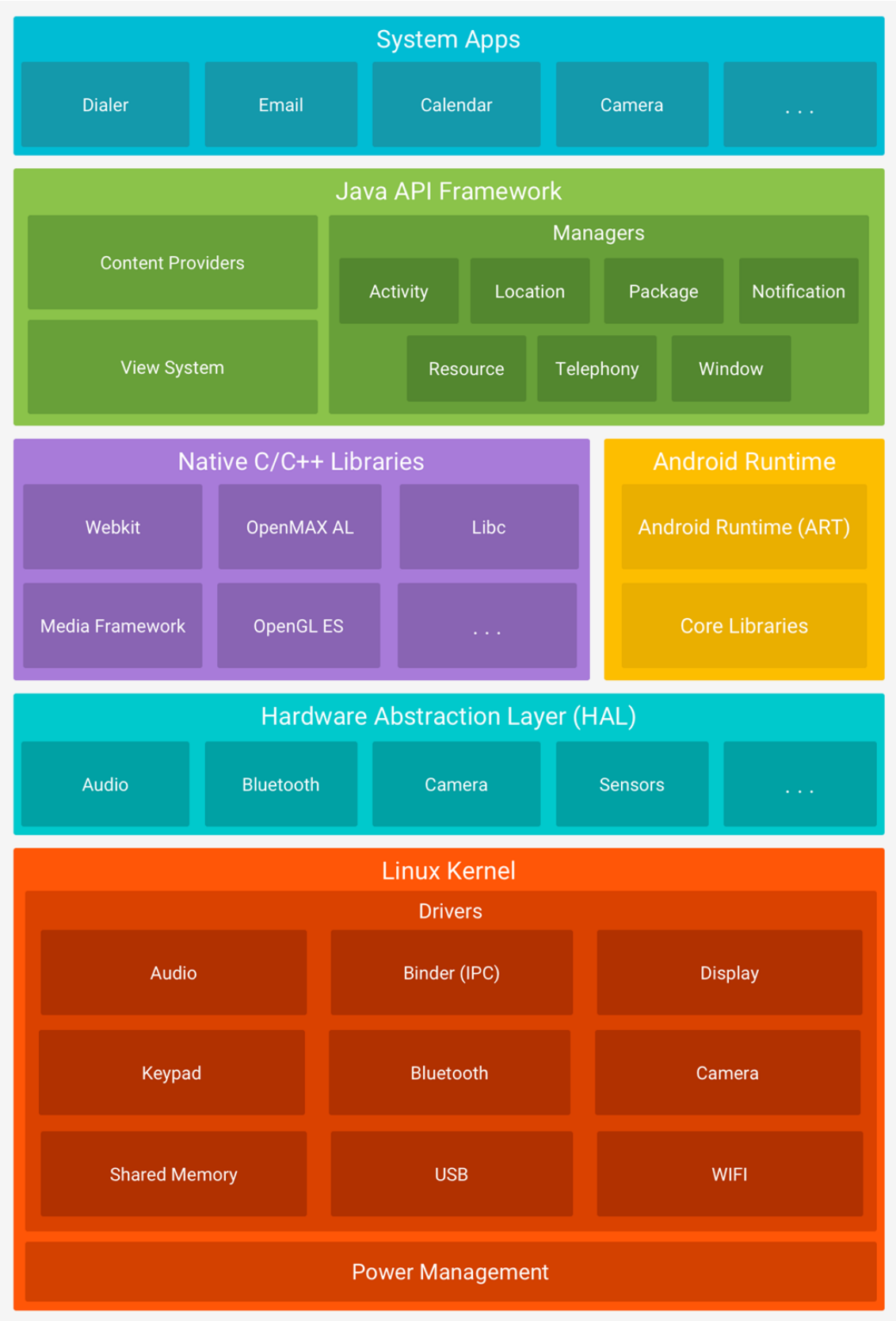
Obecnie system Android jest niekwestionowanym liderem na rynku urządzeń mobilnych, z szacowaną liczbą 3,6 miliarda użytkowników w 2023 roku [5]. Wykres jego popularności został przedstawiony na Rysunku 2. Ogromna popularność systemu sprzyja popytowi na aplikacje,

który spowodował, że w 2023 roku było ich dostępnych aż 3,718 miliona w oficjalnym sklepie platformy Google Play [6].



Rysunek 2. Liczba użytkowników urządzeń z system Android w latach 2012-2023 (w miliardach) [5]

System Android jest projektem typu open-source na licencji Apache 2.0, od czego wyjątkiem jest jednak jądro Linuxa, które jest rozpowszechniane na licencji GNU GPLv2 [7]. Fakt, że system jest zbudowany na bazie jądra Linuxa, czyni go podatnym na modyfikacje i gwarantuje jego kompatybilność z szeroką gamą urządzeń takich jak smartfony, tablety, notebooki czy komputery PC. Architekturę systemu przedstawiono na Rysunku 3. Jądro Linuxa odpowiada za zarządzanie wielowątkowością, pamięcią, zasilaniem i zasobami sprzętowymi. Natywne biblioteki napisane w językach C i C++ służą między innymi renderowaniu grafiki, zarządzaniu bazami danych aplikacji czy obsłudze przeglądarki internetowej. HAL, czyli warstwa abstrakcji sprzętowej, pozwala na komunikację z komponentami sprzętowymi takimi jak moduł Bluetooth, kamera, głośniki i różne sensory wbudowane w urządzenie. Środowisko uruchomieniowe Android Runtime odpowiada za zarządzanie maszynami wirtualnymi i kompilację kodu bajtowego do wykonywalnego kodu maszynowego zgodnego z architekturą procesora docelowego. Java API Framework zawiera klasy wykorzystywane przy tworzeniu aplikacji i interfejsu użytkownika oraz odpowiada za komunikację z natywnymi bibliotekami systemowymi. System Apps, czyli aplikacje systemowe, to zestaw wbudowanych aplikacji takich jak kalendarz, poczta email, wiadomości SMS, kontakty i wiele innych, umożliwiających korzystanie z podstawowych funkcjonalności urządzenia [8].



Rysunek 3. Architektura systemu Android [8]

1.4. Analiza dostępnych rozwiązań

PanParagon to aplikacja dedykowana na polski rynek. Podczas skanowania paragonu aplikacja ta wyszukuje datę, nazwę sklepu i sumę należności, po czym pozwala użytkownikowi na dodanie go do bazy paragonów z przypisaną pojedynczą kategorią, bez możliwości rozłożenia na poszczególne produkty. Oprócz tego posiada funkcjonalności takie jak tworzenie list zakupów, dodawanie kart lojalnościowych, przeglądanie dostępnych kuponów i gazetek z aktualnymi ofertami różnych sklepów, porównywanie wydatków z innymi użytkownikami, obsługa e-paragonów, automatyczne przetwarzanie zdjęć paragonów znajdujących się w pamięci urządzenia, śledzenie terminów upływu gwarancji i zwrotu. Na rysunkach 4, 5 i 6 przedstawiono zrzuty ekranu wykonane podczas korzystania z aplikacji PanParagon.

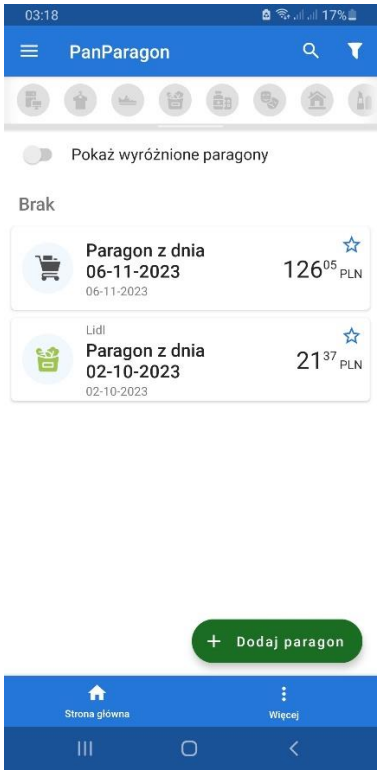
Wśród zalet opisanej wyżej aplikacji warto wymienić:

- stabilne działanie,
- intuicyjny i przejrzysty interfejs użytkownika,
- dużo praktycznych funkcjonalności.

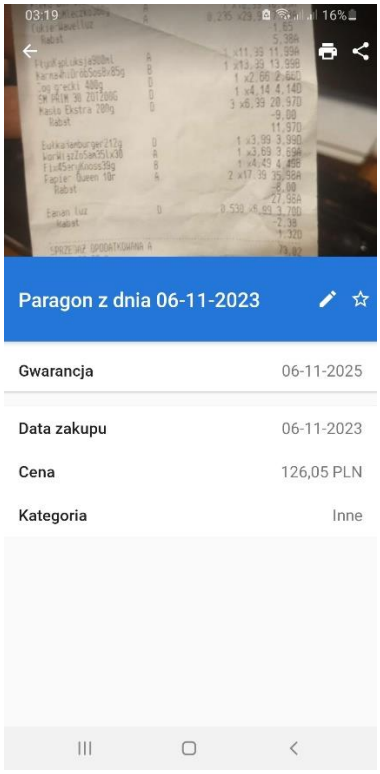
Do głównych jej wad należą:

- traktowanie paragonu jako całości, bez możliwości wyodrębnienia poszczególnych produktów,
- brak możliwości tworzenia własnych kategorii zakupów.

Rysunek 4. Widok główny aplikacji PanParagon



Rysunek 5. Widok listy paragonów w aplikacji PanParagon



Rysunek 6. Widok pojedynczego paragonu w aplikacji PanParagon

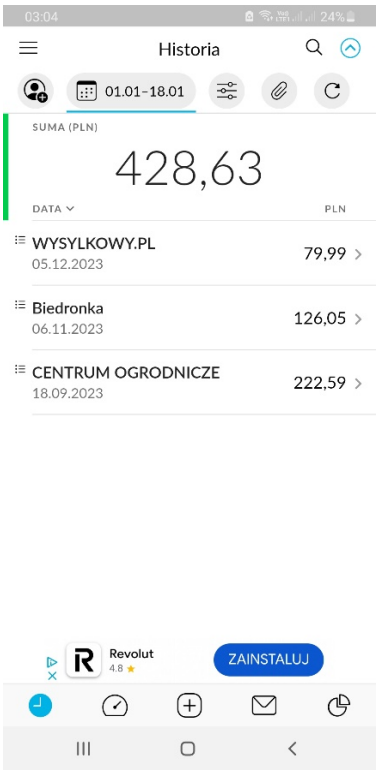
Kolejnym przykładem aplikacji do skanowania paragonów jest Billy, który tak samo jak omawiany wcześniej PanParagon przy skanowaniu paragonu wyszukuje datę, nazwę sklepu oraz sumę należności, z tą różnicą, że posiada opcję wyodrębnienia poszczególnych produktów. Niestety, wyodrębnianie produktów nie działa zbyt dobrze, gdyż jest mało kompatybilne z polskimi paragonami, ponieważ bardzo często skanowanie kończy komunikat przedstawiony na Rysunku 9, oznaczający, że użytkownik będzie miał jedynie opcję dodania paragonu z pojedynczym produktem o nazwie „Wszystkie zakupy" i cenie równej sumie należności widniejącej na paragonie. Aplikacja posiada też opcję planowania budżetu, dzięki której można wprowadzić miesięczne wpływy, zobowiązania i planowane oszczędności w celu późniejszego zestawienia ich z dodanymi w trakcie miesiąca wydatkami. Na rysunkach 4, 5 i 6 przedstawiono zrzuty ekranu wykonane podczas korzystania z aplikacji Billy.

Zaletami aplikacji Billy są:

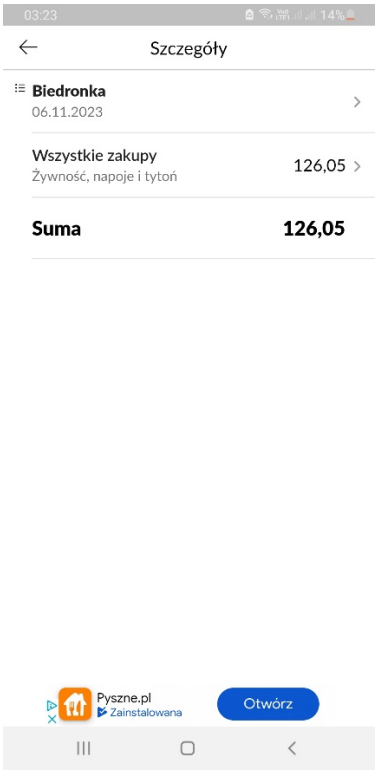
- intuicyjny i przejrzysty interfejs użytkownika,
- opcja planowania miesięcznego budżetu.

Z kolei jej wady to:

- częste problemy z wyodrębnieniem produktów znajdujących się na zeskanowanym paragonie,
- brak możliwości tworzenia własnych kategorii zakupów.



Rysunek 7. Widok historii zakupów w aplikacji Billy



Rysunek 8. Widok pojedynczego paragonu w aplikacji Billy

Rysunek 9. Komunikat po skanowaniu paragonu w przypadku nieudanego wyodrębnienia produktów w Billy

1.5. Cel i założenia pracy

Celem pracy jest opracowanie i implementacja aplikacji mobilnej na urządzenia z systemem Android w wersji co najmniej 8.0 (Oreo). Aplikacja ma umożliwiać zeskanowanie paragonu i dodanie poszczególnych produktów do bazy danych, z możliwością przypisania każdemu z nich wcześniej utworzonej przez użytkownika kategorii. Ponadto ma pozwalać na przeglądanie dokonanych zakupów i związanych z nimi wydatków w łatwy i przejrzysty sposób.

Praca obejmuje stworzenie aplikacji z wykorzystaniem:

- środowiska programistycznego Android Studio,
- języka Java w wersji 1.8,
- systemu zarządzania relacyjną bazą danych SQLite,
- interfejsu programowania aplikacji Machine Learning Kit Text Recognition v2 firmy Google, służącego do rozpoznawania tekstu ze zdjęć,
- biblioteki open-source, służącej do skanowania i przycinania skanów dokumentów.

1.6. Wymagania funkcjonalne i niefunkcjonalne aplikacji

Wymagania funkcjonalne:

- Możliwość skanowania paragonu za pomocą kamery urządzenia
- Możliwość skanowania paragonu ze zdjęcia znajdującego się w pamięci urządzenia
- Możliwość ograniczenia obszaru skanowania poprzez przycięcie zdjęcia
- Możliwość dodawania do bazy danych rozpoznanych produktów z ich ceną, nazwą, kategorią i datą
- Możliwość modyfikacji cen, nazw i kategorii produktów przed dodaniem ich do bazy danych oraz po dodaniu
- Możliwość tworzenia kategorii zakupów
- Możliwość usuwania utworzonych kategorii zakupów i modyfikacji ich nazw oraz odzwierciedlanie tych operacji na produktach dodanych z modyfikowaną kategorią
- Przeglądanie listy dodanych produktów z możliwością sortowania ze względu na cenę, nazwę i datę dodania
- Możliwość sprawdzenia kwoty wydanej na produkty z danej kategorii lub/i określonej nazwie w określonym przez użytkownika przedziale czasowym

Wymagania niefunkcjonalne:

- Działanie na systemach Android w wersji 8.0 lub wyższej

2. Wykorzystane narzędzia programistyczne

2.1. Środowisko programistyczne

Android Studio to oficjalne środowisko programistyczne dedykowane platformie Android, oparte na programie IntelliJ IDEA, czyli rozwijanym przez firmę JetBrains środowisku programistycznym do tworzenia oprogramowania w językach Java i Kotlin. Projekt środowiska został ogłoszony przez firmę Google w maju 2013 roku, a jego pierwsza stabilna wersja została wydana w grudniu 2014. Z końcem 2015 roku Google ogłosiło zakończenie wsparcia dla Eclipse Android Development Tools, poprzedniego oficjalnego środowiska programistycznego platformy Android, i zastąpienie go przez Android Studio [9].

Android Studio jest kompletnym narzędziem służącym do tworzenia aplikacji na wszystkie urządzenia z systemem Android. Wspiera języki Java, Kotlin, JavaScript i C++. Umożliwia budowanie aplikacji poprzez system automatyzacji kompilacji Gradle, wspiera proces pisania kodu oraz debugowanie. Zawiera również rozbudowany emulator, umożliwiający emulowanie urządzeń takich jak smartfony, tablety, smartwatche, pozwalając na wybór konkretnych modeli dostępnych na rynku lub stworzenie urządzenia o określonych przez użytkownika parametrach i testowanie działania rozwijanej aplikacji.

Aplikacja opisana w pracy była tworzona z wykorzystaniem Android Studio w wersji Giraffe z lipca 2023 roku.

2.2. Język Java

Java to obiektowy język programowania stworzony przez Jamesa Goslinga. Jego pierwsza publiczna implementacja zadebiutowała w 1996 roku. Została wydana jako Java 1.0 przez firmę Sun Microsystems, której pracownikiem był wspomniany twórca. W listopadzie 2006 roku ówczesna implementacja została udostępniona do użytku publicznego na licencji otwartej GNU General Public License [10].

Dzięki wykorzystaniu maszyny wirtualnej *Java virtual machine* (JVM), będącej częścią środowiska uruchomieniowego, Java jest językiem niezależnym od architektury i systemu operacyjnego [11], co czyni z niej idealne narzędzie do tworzenia aplikacji na system Android.

Aplikacja opisana w pracy została stworzona z wykorzystaniem języka Java w wersji 1.8.

2.3. SQLite

SQLite to system zarządzania relacyjną bazą danych napisany w języku C. Jest projektem typu *open source* stworzonym w 2000 roku przez Dwayna Richarda Hippa [12].

Android posiada własną, oficjalną bibliotekę SQLiteDatabase przeznaczoną do implementacji bazy danych, pozwalającą tworzyć, usuwać oraz zarządzać bazą przy pomocy komend języka SQL. Jej dokumentacja jest dostępna pod adresem [13]. Dla deweloperów dostępna jest również klasa pomocnicza SQLiteOpenHelper zawierająca zestaw podklas pozwalających uprościć kod wykorzystany przy implementacji bazy danych i uczynić go bardziej czytelnym. Jej dokumentacja jest dostępna pod adresem [14].

2.4. Google ML Kit

ML Kit to zestaw narzędzi opracowany przez firmę Google, zawierający gotowe rozwiązania wykorzystujące uczenie maszynowe, stworzone dla deweloperów aplikacji mobilnych przeznaczonych na systemy Android i iOS. Zawiera 13 interfejsów API przeznaczonych do różnych zadań takich jak między innymi rozpoznawanie tekstu, wykrywanie twarzy, skanowanie kodów kreskowych, dodawanie etykiet do obrazów czy identyfikacja języka i

tłumaczenie na inny język [15]. Każdy interfejs można importować do rozwijanej aplikacji niezależnie od innych, biorąc pod uwagę jakie funkcjonalności są potrzebne.

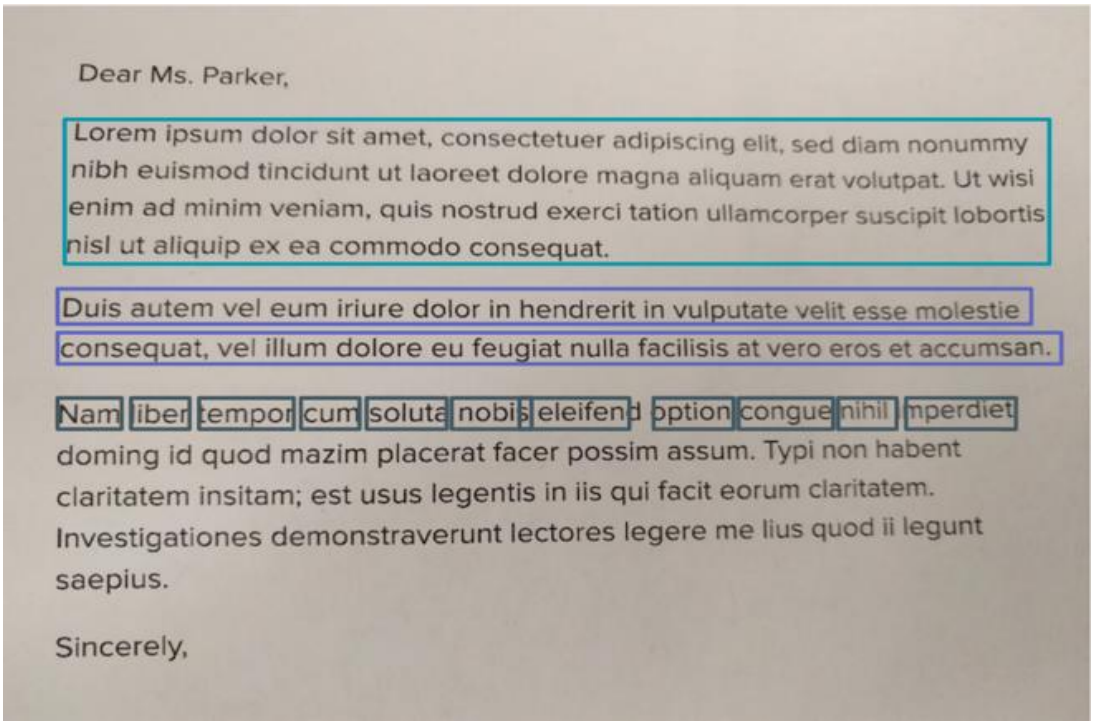
W aplikacji został wykorzystany interfejs ML Kit Text Recognition v2, służący do rozpoznawania tekstu. Na dzień 19 stycznia 2024 roku obsługuje on 37 języków, w tym język polski. W fazie rozwoju jest 16 innych języków [16]. Na oficjalnej stronie projektu ML Kit, dostępnej pod adresem [17], znajduje się opis jego funkcjonalności i sposobu działania. Fragment tego opisu został przytoczony poniżej.

„Moduł rozpoznawania tekstu dzieli tekst na bloki, linie, elementy i symbole. Ogólnie rzecz biorąc:

- Blok to sąsiadujący się z nimi zestaw wierszy tekstu, np. akapit lub kolumna,
- Linia to sąsiedni zestaw słów na tej samej osi,
- Element to ciągły zestaw znaków alfanumerycznych („słowo") na tej samej osi w większości języków łacińskich lub słowo w innych
- Symbol to pojedynczy znak alfanumeryczny na tej samej osi w większości języków łacińskich lub znak w innych językach.

(…) W przypadku wszystkich wykrytych bloków, linii, elementów i symboli interfejs API zwraca ramki ograniczające, punkty narożne, informacje o rotacji, wskaźnik ufności, rozpoznawany język i rozpoznany tekst" [17].

Wizualizacja ramek ograniczających bloki, linie i elementy została zilustrowana na Rysunku 10. Wyniki rozpoznawania tekstu są zwracane w formie tablicy bloków tekstu, gdzie każdy blok zawiera tablicę linijek, a każda linijka zawiera tablicę elementów.



Rysunek 10. Wizualizacja ramek ograniczających bloki (turkusowy), linijki (niebieski), elementy (ciemnoniebieski) [17]

2.5. Biblioteka Android Document Scanner

Android Document Scanner to biblioteka *open source* stworzona w języku Kotlin przez Davida Marcusa. Jest ona udostępniona na platformie GitHub w repozytorium autora, dostępnym pod adresem [18]. Bazuje na zapoczątkowanej przez firmę Intel bibliotece OpenCV [19], która jest dostępna na licencji *open source* (Apache License 2.0).

Android Document Scanner umożliwia automatyczne wykrywanie narożników dokumentu na zdjęciu, jeżeli wystarczająco kontrastuje on z tłem, czego przykład pokazano na Rysunku 11. W przypadku niedokładnego lub błędnego wykrycia użytkownik ma możliwość ich ręcznej korekcji. Po wybraniu i zatwierdzeniu narożników dokument zostaje przycięty do prostokątnego kształtu oraz skorygowana zostaje jego perspektywa. Wynik tej operacji ilustruje Rysunek 12.

Rysunek 11. Automatycznie wykryte narożniki paragonu ze zdjęcia

Rysunek 12. Podgląd przycięcia paragonu ze zdjęcia

2.6. Wyrażenia regularne

Wyrażenie regularne jest wzorcem opisującym łańcuch symboli [20], W informatyce wykorzystuje się je do wyszukiwania łańcuchów znaków w tekście pasujących do zadeklarowanego wzorca.

W języku Java wyrażenia regularne zostały zaimplementowane w postaci klas o nazwach „Pattern” i „Matcher”. Klasa Pattern, której dokumentacja jest dostępna pod adresem [21], pozwala na kompilację wzorca wyrażenia regularnego, którego późniejsze wyszukanie w przetwarzanym tekście jest możliwe z wykorzystaniem klasy Matcher (dokumentacja dostępna pod adresem [22]).

3. Implementacja

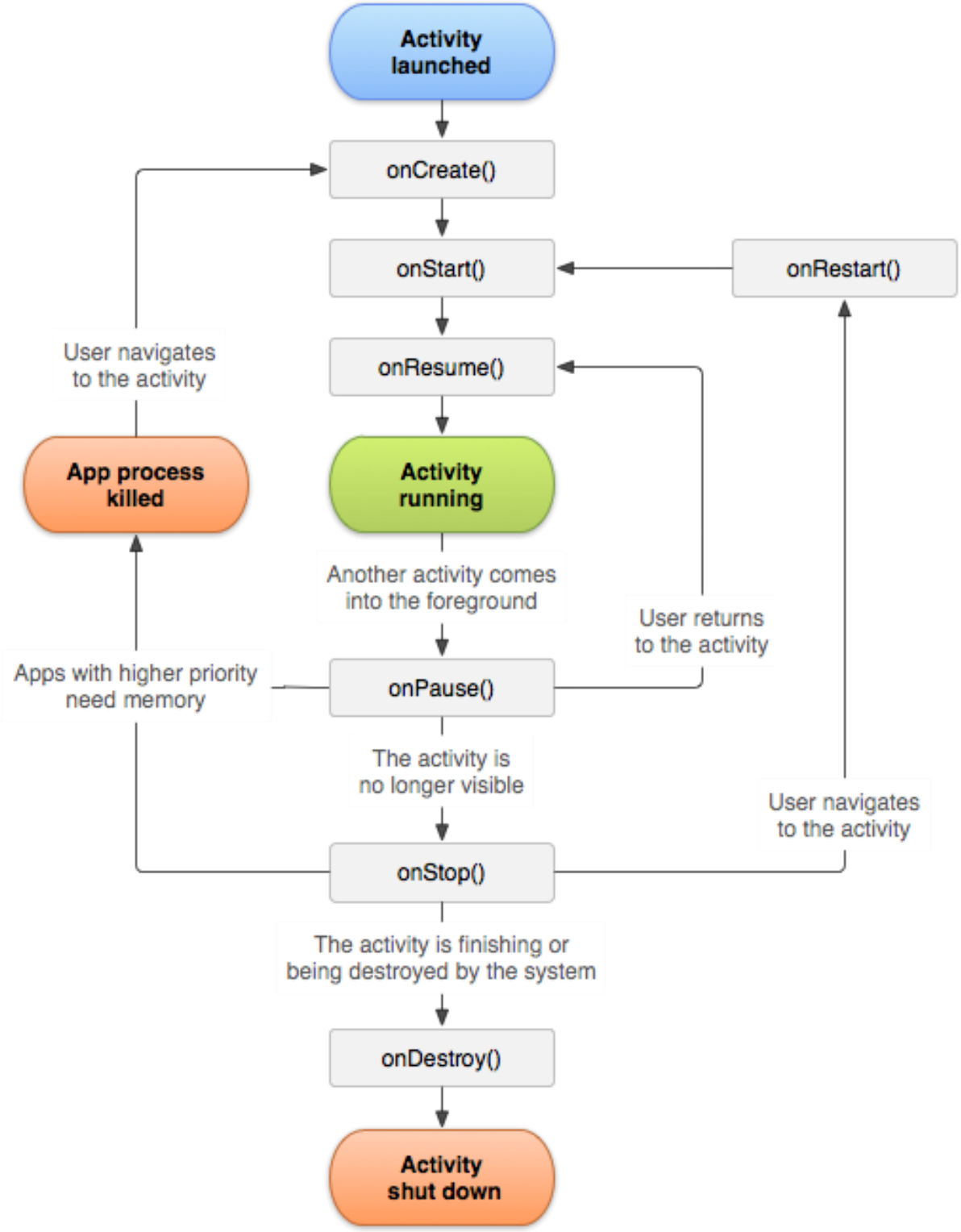
3.1. Aktywności

Aktywność w kontekście programowania na systemy Android to jeden z kluczowych komponentów wykorzystywanych przy tworzeniu aplikacji. Aktywność reprezentuje jedno z okien interfejsu użytkownika. Jej definicja składa się z pliku XML zawierającego układ poszczególnych komponentów na ekranie urządzenia (takich jak przyciski, napisy, suwaki, widoki listy itp.) oraz pliku napisanego w języku Java (lub Kotlin), zawierającego kod odpowiedzialny za działanie wspomnianych komponentów i innych funkcji aplikacji.

Do kierowania procesem przełączania się pomiędzy aktywnościami służy klasa Activity [23]. Zawiera ona zestaw metod służących do zarządzania cyklem życia aktywności. W sporządzonej na podstawie dokumentacji klasy Tabeli 1 przedstawiono krótki opis funkcji każdej z metod oraz kolejność ich wykonywania. Cykl życia aktywności w systemie Android przedstawia Rysunek 13.

Tabela 1. Opis metod klasy Activity

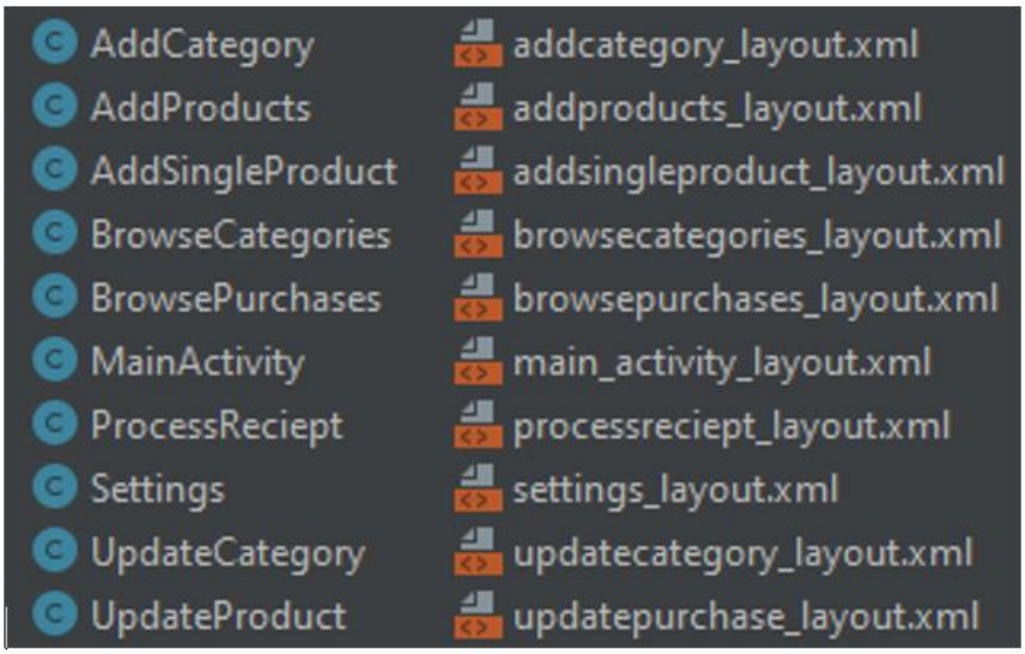
Metoda	Opis	Kolejna wykonywana metoda
onCreate()	Jest wykonywana w momencie utworzenia aktywności. Odpowiada za utworzenie widoku lub połączenie z zawierającym go plikiem XML. Podczas jej wykonywania przypisuje się funkcje do elementów widoku (np. przycisków) i wczytuje potrzebne dane. Pozwala również na wczytanie poprzedniego stanu aktywności.	onStart()
onRestart()	Jest wykonywana przy ponownym rozpoczęciu aktywności w przypadku jej wcześniejszego zatrzymania.	onStart()
onStart()	Jest wykonywana w momencie, w którym aktywność staje się widoczna dla użytkownika.	onResume() lub onStop()
onResume()	Jej zadaniem jest przygotowanie aktywności do interakcji z użytkownikiem tzn. przyjmowania informacji takich jak gesty na ekranie itp.	onPause()
onPause()	Jest wykonywana w przypadku, kiedy nie ma potrzeby na interakcje użytkownika z aktywnością, ale ma ona pozostać dla niego widoczna.	onResume() lub onStop()
onStop()	Jest wykonywana w momencie, kiedy aktywność przestaje być widoczna dla użytkownika z powodu rozpoczęcia innej aktywności, przywołania istniejącej w tle lub zniszczenia obecnej.	onRestart() lub onDestroy()
onDestroy()	Metoda kończąca cykl życia aktywności. Jest wykonywana w momencie zniszczenia obecnej instancji aktywności i zwolnienia pamięci.	



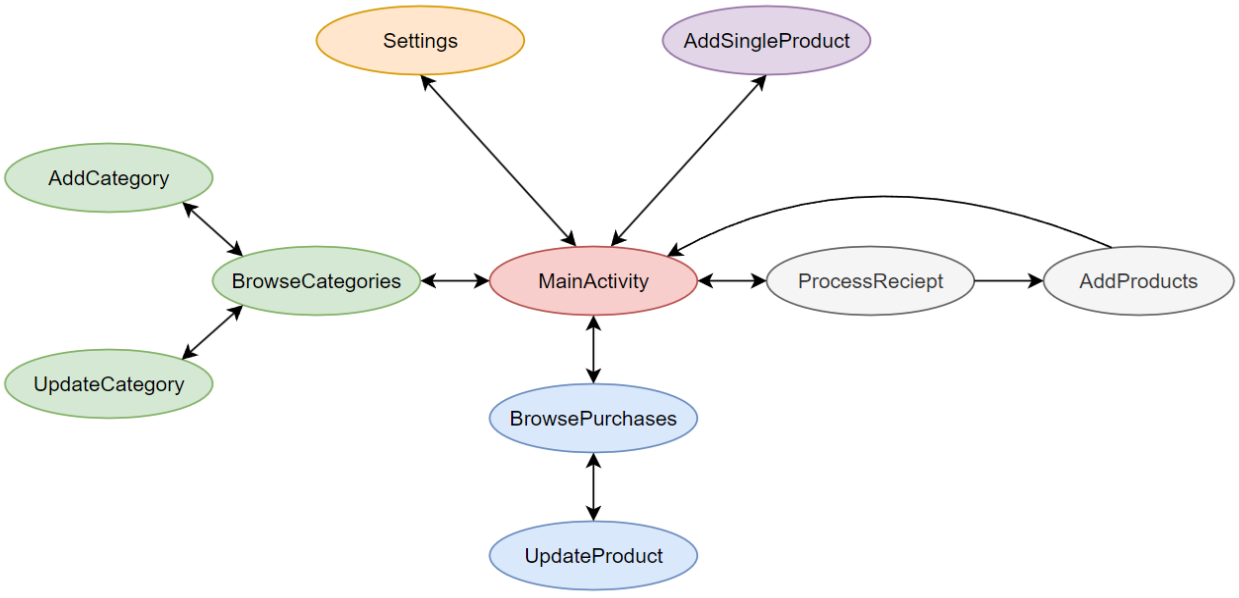
Rysunek 13. Cykl życia aktywności w systemie Android [23]

Po uruchomieniu opisanej tu aplikacji zostaje rozpoczęta aktywność „MainActivity", zawierająca przyciski uruchamiające aktywności odpowiadające za poszczególne widoki aplikacji, które z kolei mogą uruchamiać inne aktywności odpowiadające za realizację poszczególnych funkcjonalności. Na Rysunku 14 znajduje się lista zaimplementowanych

aktywności wraz z odpowiadającymi im plikami XML. Rysunek 15 obrazuje schemat nawigacji pomiędzy utworzonymi aktywnościami, przy czym zwroty strzałek symbolizują możliwe przejścia między nimi.



Rysunek 14. Aktywności aplikacji (lewa kolumna) i odpowiadające im pliki XML (prawa kolumna)



Rysunek 15. Schemat nawigacji pomiędzy aktywnościami

3.2. Baza danych

Baza danych SQLite jest tworzona przy pierwszym uruchomieniu aplikacji lub po wyczyszczeniu jej danych z pamięci urządzenia. Wywoływana jest wtedy przedstawiona na Rysunku 16. metoda „onCreate” klasy SQLiteOpenHelper, która tworzy puste tabele zakupów i kategorii oraz tabele przechowującą ustawienia aplikacji, do której zostaje dodane ustawienie dotyczące przetwarzania rabatów z domyślną wartością 0, oznaczającą „nie przetwarzaj”. Skutkuje to utworzeniem bazy danych o strukturze widocznej na Rysunku 17.

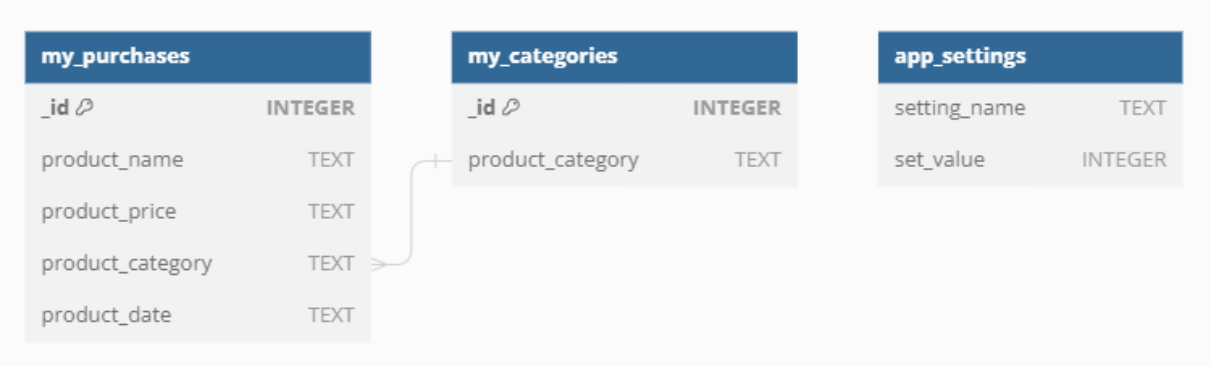
```
@Override
public void onCreate(SQLiteDatabase db) {
    String query = "CREATE TABLE " + TABLE_NAME_PURCHASES +
            " (" + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            COLUMN_PRODUCT + " TEXT, " +
            COLUMN_PRICE + " TEXT, " +
            COLUMN_CATEGORY + " TEXT, " +
            COLUMN_DATE + " TEXT);";
    db.execSQL(query);

    query = "CREATE TABLE " + TABLE_NAME_CATEGORIES +
            " (" + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            COLUMN_CATEGORY + " TEXT UNIQUE);";
    db.execSQL(query);

    query = "CREATE TABLE app_settings (setting_name TEXT UNIQUE, set_value INTEGER);";
    db.execSQL(query);

    query = "INSERT INTO app_settings (setting_name,set_value) VALUES('include_discounts', 0);";
    db.execSQL(query);
}
```

Rysunek 16. Implementacja metody „onCreate”



Rysunek 17. Schemat przedstawiający strukturę utworzonej bazy danych

Dodawanie, usuwanie i aktualizowanie wartości tabel z zachowaniem ich relacji odbywa się za pomocą zaimplementowanych do każdej z tych operacji metod. Na Rysunku 18 jako przykład pokazano metodę „deleteOneRowPurchase”, odpowiadającą za usuwanie pojedynczego zakupu z tabeli „my_purchases”.

```
void deleteOneRowPurchases(String row_id){
    SQLiteDatabase db = this.getWritableDatabase();
    long result = db.delete(TABLE_NAME_PURCHASES,  whereClause: "_id=?", new String[]{row_id});
    if(result == -1){
        Toast.makeText(context,  text: "Coś poszło nie tak", Toast.LENGTH_SHORT).show();
    }else{
        Toast.makeText(context,  text: "Usunięto", Toast.LENGTH_SHORT).show();
    }
}
```

Rysunek 18. Implementacja metody „deleteOneRowPurchases”

3.3. Ekstrakcja nazw i cen produktów

Przepisy regulujące jakie informacje powinny znajdować się na paragonie fiskalnym oraz ich kolejność znajdują się w rozporządzeniu Ministra Finansów z dnia 14 marca 2013 r.[1] Fragment paragrafu § 8.1 tego rozporządzenia, istotny w kontekście implementacji, brzmi następująco:

> § 8. 1. Paragon fiskalny zawiera co najmniej:
> (…)
> 5) oznaczenie „PARAGON FISKALNY”;
> 6) nazwę towaru lub usługi pozwalającą na jednoznaczną ich identyfikację;
> 7) cenę jednostkową towaru lub usługi;
> 8) ilość i wartość sumaryczną sprzedaży danego towaru lub usługi z oznaczeniem literowym przypisanej stawki podatku;
> (…)
> 13) łączną kwotę sprzedaży brutto;

Metoda ekstrakcji nazw i cen produktów korzysta z opisanego powyższym cytatem rozmieszczenia informacji na paragonie. Za jej realizację odpowiadają trzy funkcje z aktywności „ProcessReciept”. Zadaniem pierwszej z nich, nazwanej „prepareTextRecognitionResult”, jest:

1. Stworzenie tablicy linijek z tablicy bloków tekstu, będącej wynikiem skanowania z wykorzystaniem modułu rozpoznawania tekstu opisanego w podrozdziale 2.4.
2. Znalezienie linijki zawierającej oznaczenie „PARAGON FISKALNY” i linijki zawierającej oznaczenie sumy należności na paragonie. Znalezienie oznaczenia „PARAGON FISKALNY” jest konieczne do przetworzenia paragonu, w przeciwieństwie do sumy należności, której zdjęcie nie musi obejmować.
3. Wykorzystanie punktów narożnych ramek ograniczających linijki znalezione w poprzednim kroku do wyznaczenia obszaru zawierającego nazwy i ceny produktów.
4. Usunięcie linijek, których ramki ograniczające nie znajdują się w wyznaczonym obszarze.
5. Przekazanie zmodyfikowanej w poprzednim kroku tablicy linijek do dalszego przetworzenia w funkcji „process_result” lub „process_result_including_discounts”, w zależności od wybranej w ustawieniach aplikacji opcji.

[1] (Dz.U. z 2013 roku, poz. 363)

Wyznaczony na przykładowym paragonie obszar zawierający nazwy i ceny produktów ilustruje Rysunek 19.

Rysunek 19. Wyznaczony obszar zawierający nazwy i ceny produktów (zaznaczony kolorem zielonym)

Funkcje „process_result” i „process_result_including_discounts” wykorzystują wyrażenia regularne do wyszukania konkretnych informacji w otrzymanej tablicy zawierającej linijki teksu. Ich zadania to:

1. Usunięcie oznaczeń podatkowych nieznajdujących się przy cenach produktu, cen jednostkowych, ilości sztuk, wagi.
2. Znalezienie ostatecznych cen produktów, korzystając z faktu, że są liczbami z dwoma miejscami dziesiętnymi, są zawsze na końcu linijki i znajduje się przy nich oznaczenie literowe przypisanej stawki podatku (litera A,B,C,D,E,F lub G). Dodanie znalezionych cen do tablicy cen i wycięcie ich z przetwarzanej linijki, która następnie trafia do tablicy produktów.
3. Obliczenie rabatów w przypadku funkcji „process_result_including_discounts”.

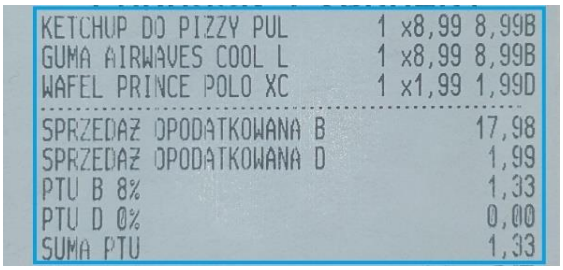
W tabeli 2. pokazano wykorzystane wzorce wyrażeń regularnych i ich zastosowanie na przykładowych napisach. Człon „\d+[,.]\d{2}” reprezentuje liczbę z dwoma miejscami dziesiętnymi po przecinku lub kropce, natomiast człon „[ABCcDEFG]” reprezentuje istniejące oznaczenia literowe stawki podatkowej (ze względu na częste błędne rozpoznania oznaczenia „C” jako „c”, mała litera została również uwzględniona). Znak „$” oznacza umiejscowienie na końcu wiersza tekstu (linijki).

Tabela 2. Zastosowane główne wzorce rozpoznawania cen

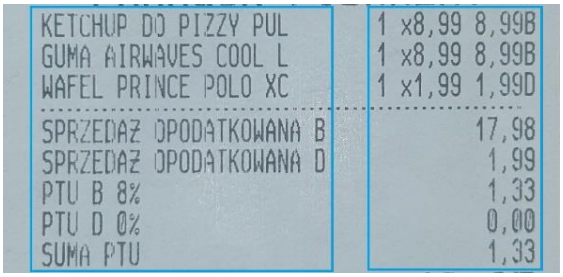
Wzorzec wyrażenia regularnego	Przykład przetwarzanego napisu	Znalezione dopasowanie
\d+[,.]\d{2}[ABCcDEFG]$	Nap. Fanta 0,5L 3,99A	3,99A
\d+[,.]\d{2}[ ][ABCcDEFG]$	Majonez 500ml 8,79 B	8,79 B
[-]+\d+[,.]\d{2}[ABCcDEFG]$	Rabat -5,80D	-5,80D
[-]+\d+[,.]\d{2}[ ][ABCcDEFG]$	Rabat -9,99 E	-9,99 E

Wyniki działania funkcji „proces_result” na tablicy wierszy zwróconej przez funkcję „prepareTextRecognitionResult” po skanowaniu paragonu z Rysunku 20 pokazano w Tabeli 2. i 3. Podział na przypadki występuje ze względu na różne możliwości rozpoznania bloków tekstu. Przypadek, w którym cały brany pod uwagę obszar znajduje się w jednym bloku tekstu (Rysunek 21), jest przedstawiony w Tabeli 2. Może jednak wystąpić sytuacja, zilustrowana w Tabeli 3, w której ze względu na odległość pomiędzy kolumnami produktów i cen, znajdują się one w oddzielnych blokach tekstu (Rysunek 22).

Rysunek 20. Skanowany w przykładzie paragon



Rysunek 21. Rozpoznanie jako pojedynczy blok tekstu, zaznaczony niebieskimi ramkami



Rysunek 22. Rozpoznanie jako dwa oddzielne bloki tekstu, zaznaczone niebieskimi ramkami

Tabela 2. Przedstawienie działania funkcji „proces_result" w przypadku, kiedy kolumny produktów i cen są w jednym bloku tekstu.

Pozycja w tablicy	Przetwarzana tablica linijek	Tablica produktów	Tablica cen
0	KETCHUP DO PIZZY PUL 1 x8,99 8,99B	KETCHUP DO PIZZY PUL	8,99B
1	GUMA AIRWAVES COOL L 1 x8,99 8,99B	GUMA AIRWAVES COOL L	8,99B
2	WAFEL PRINCE POLO XC 1 x1,99 1,99D	WAFEL PRINCE POLO XC	1,99D
3	SPRZEDAŻ OPODATKOWANA B 17,98	SPRZEDAŻ OPODATKOWANA B	
4	SPRZEDAŻ OPODATKOWANA D 1,99	SPRZEDAŻ OPODATKOWANA D	
5	PTU B 8% 1,33	PTU B 8%	
6	PTU D 8% 0,00	PTU D 8%	
7	SUMA PTU 1,33	SUMA PTU	

Tabela 3. Przedstawienie działania funkcji „proces_result” w przypadku, kiedy kolumny produktów i cen są w dwóch różnych blokach tekstu.

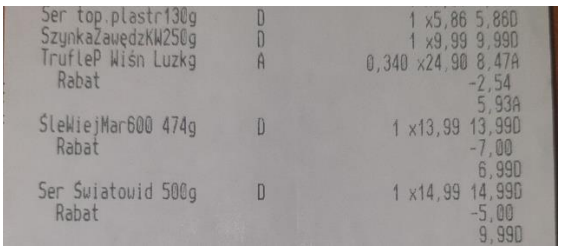
Pozycja w tablicy	Przetwarzana tablica linijek	Tablica produktów	Tablica cen
0	KETCHUP DO PIZZY PUL	KETCHUP DO PIZZY PUL	8,99B
1	GUMA AIRWAVES COOL L	GUMA AIRWAVES COOL L	8,99B
2	WAFEL PRINCE POLO XC	WAFEL PRINCE POLO XC	1,99D
3	SPRZEDAŻ OPODATKOWANA B	SPRZEDAŻ OPODATKOWANA B	
4	SPRZEDAŻ OPODATKOWANA D	SPRZEDAŻ OPODATKOWANA D	
5	PTU B 8%	PTU B 8%	
6	PTU D 8%	PTU D 8%	
7	SUMA PTU	SUMA PTU	
8	1 x8,99 8,99B		
9	1 x8,99 8,99B		
10	1 x1,99 1,99D		
11	17,98		
12	1,99		
13	1,33		
14	0,00		
15	1,33		

Uzyskane tablice są przekazywane do aktywności „AddProducts”, gdzie zostają ze sobą zestawione względem rozmiaru mniejszej, co skutkuje utworzeniem pokazanego na Rysunku 23 widoku dodawania produktów, zawierającego rozpoznane produkty oraz ich ceny.

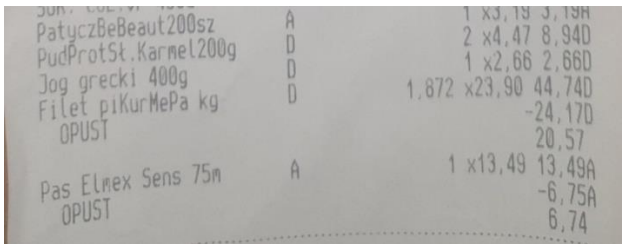
Rysunek 23. Utworzony widok dodawania produktów

3.4. Obliczanie rabatów

Sposób umieszczania rabatów na paragonach fiskalnych nie jest w żaden sposób uregulowany i każdy sklep może stosować własny wzór. Czyni to implementację funkcji automatycznego obliczania wszystkich możliwych wzorów trudnym zadaniem. W opracowanej aplikacji ta funkcjonalność jest dostępna po wybraniu odpowiedniej opcji w ustawieniach i obsługuje dwa wzory umieszczania rabatów na paragonach, widoczne na Rysunkach 24 i 25.

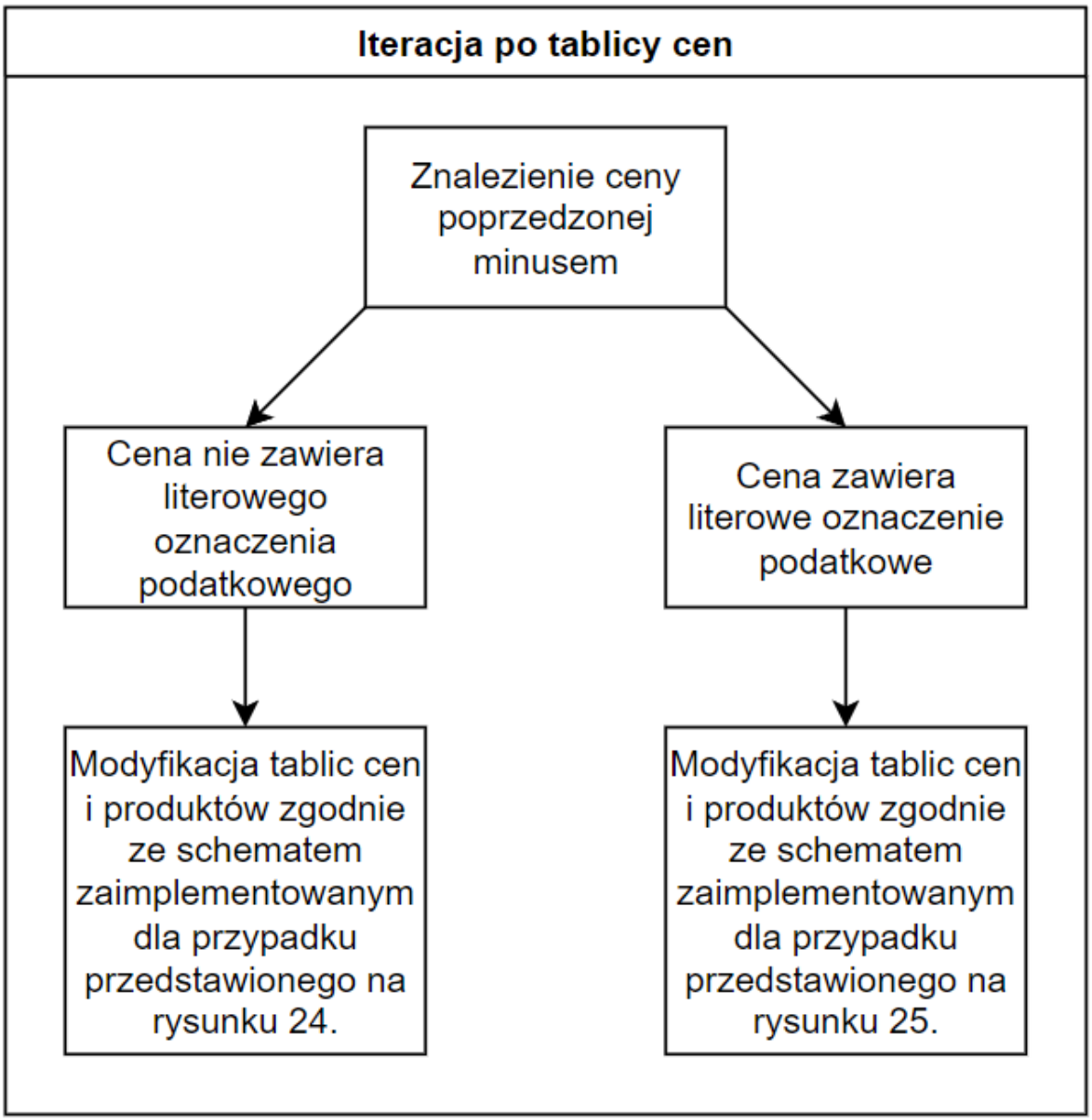


Rysunek 24. Pierwszy obsługiwany wzór rabatu



Rysunek 25. Drugi obsługiwany wzór rabatu

Za obliczenie rabatów odpowiada funkcja „process_result_including_discounts”. Różni się od funkcji „proces_result” tym, że do tablicy cen trafiają dodatkowo liczby ujemne, niezawierające literowego oznaczenia podatkowego. Następnie podczas iteracji po tej tablicy, zilustrowanej schematem na Rysunku 26, wyszukiwane są ceny poprzedzone minusem, a w przypadku ich znalezienia tablice cen i produktów są odpowiednio modyfikowane.

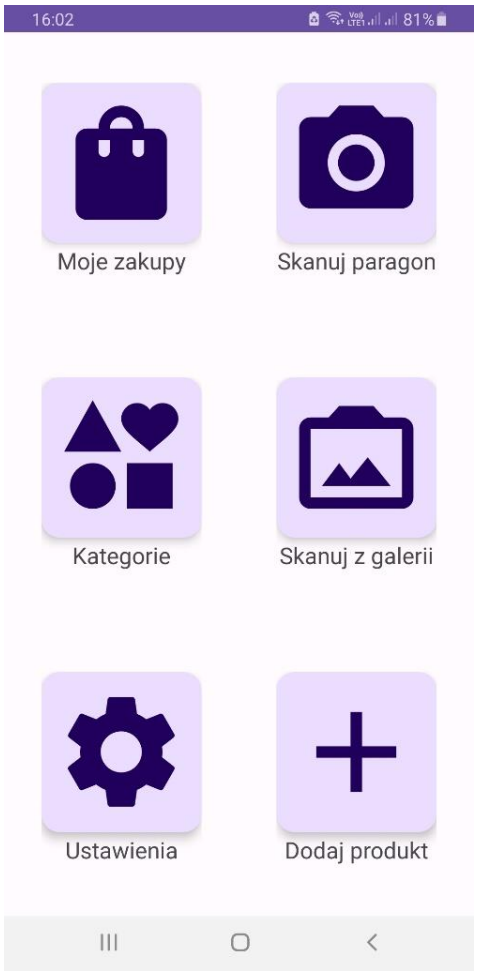


Rysunek 26. Schemat iteracji po tablicy cen

4. Interfejs użytkownika

4.1. Widok startowy

Widok startowy (Rysunek 27) pełni funkcję menu aplikacji. Zawiera 6 przycisków przeznaczonych do nawigacji. Funkcje poszczególnych przycisków zostały opisane w tabeli 4.



Rysunek 27. Widok startowy aplikacji

Tabela 4. Opis funkcji przycisków menu aplikacji

Przycisk	***Funkcja***
Moje zakupy	Przejście do widoku dodanych produktów.
Kategorie	Przejście do widoku kategorii.
Ustawienia	Przejście do widoku ustawień aplikacji.
Skanuj paragon	Przejście do skanowania paragonu z wykorzystaniem kamery urządzenia.
Skanuj z galerii	Przejście do galerii w celu wyboru zdjęcia paragonu do skanowania.
Dodaj produkt	Przejście do widoku dodawania pojedynczego produktu.

4.2. Widok ustawień aplikacji

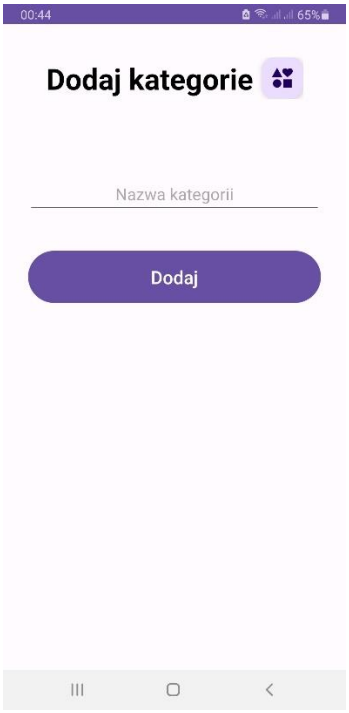
W widoku ustawień aplikacji (Rysunek 28) użytkownik ma dostęp do dwóch przycisków odpowiadających za usunięcie wszystkich dokonanych produktów i wszystkich utworzonych kategorii. Zawiera również suwak pozwalający na włączenie lub wyłączenie opcji przetwarzania znajdujących się na skanowanym paragonie rabatów.

Rysunek 28. Widok ustawień aplikacji

4.3. Widok kategorii

W widoku kategorii (Rysunek 29) użytkownik ma możliwość dodawania własnych kategorii zakupów (Rysunek 30) po kliknięciu przycisku z symbolem „+” lub aktualizacje utworzonych po kliknięciu w ich nazwy (Rysunek 31). Operacje usunięcia kategorii lub zmiany jej nazwy zostają odzwierciedlone w bazie danych na produktach dodanych z aktualnie modyfikowaną kategorią.

Rysunek 29. Widok kategorii z trzema przykładowymi kategoriami

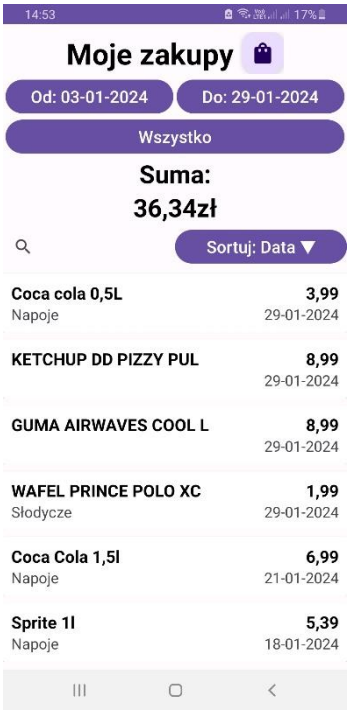


Rysunek 30. Widok dodawania kategorii

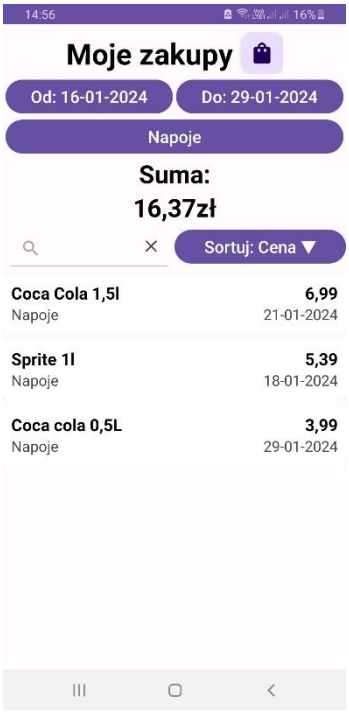
Rysunek 31. Widok aktualizacji kategorii „Napoje”

4.4. Widok dodanych produktów

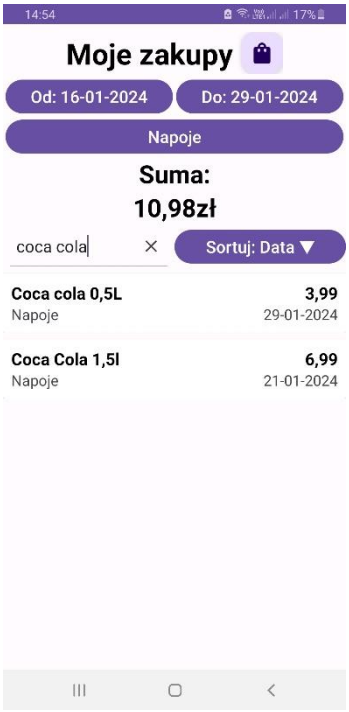
Po przejściu do widoku dodanych produktów wyświetlana jest lista wszystkich dokonanych zakupów (Rysunek 32). Korzystając z odpowiednich przycisków można wybrać kategorię i zakres dat w celu wyodrębnienia konkretnych informacji (Rysunek 33). Możliwe jest również sortowanie ze względu na datę, cenę i nazwę zakupu. Opcja wyszukiwania po nazwie pozwala na znalezienie produktów, których ceny zawierają ciąg znaków wpisany w wyszukiwarkę (Rysunek 34). Użytkownik widzi sumę pieniędzy wydanych na aktualnie wyświetlane, w zależności od wybranych filtrów, produkty.



Rysunek 32. Widok wszystkich dodanych produktów



Rysunek 33. Widok produktów po wybraniu kategorii i zakresu dat



Rysunek 34. Widok produktów wybraniu kategorii i zakresu dat oraz wyszukaniu nazwy

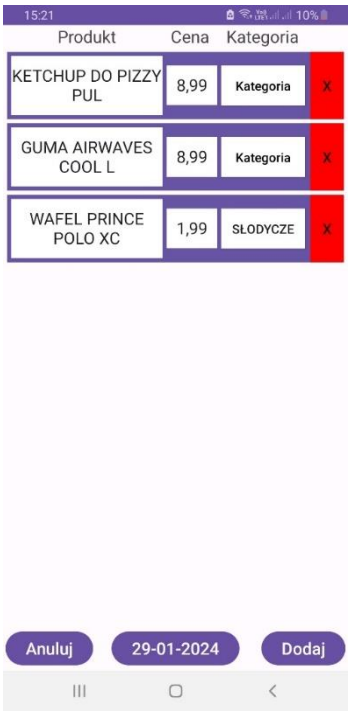
4.5. Widok potwierdzenia zdjęcia paragonu

Widok potwierdzenia zdjęcia paragonu (Rysunek 35) jest wyświetlany po przycięciu zdjęcia paragonu z kamery lub pamięci urządzenia. Służy jako podgląd, umożliwiający sprawdzenie, czy zdjęcie zostało przycięte poprawnie. Użytkownik może anulować operację przyciskiem „Anuluj” lub korzystając z paska nawigacyjnego urządzenia. Po kliknięciu przycisku „Przetwarzaj” następuje przejście do widoku dodawania produktów z paragonu.

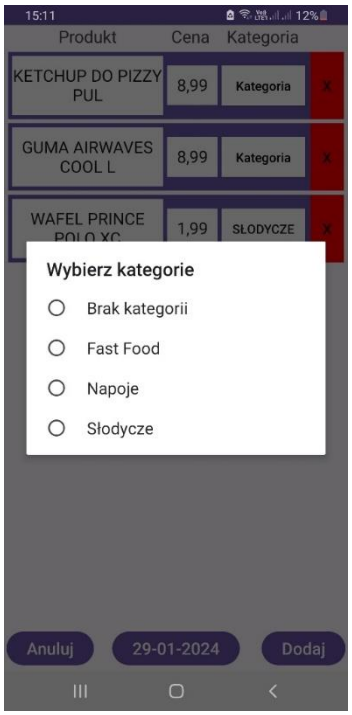
Rysunek 34. Widok potwierdzenia zdjęcia paragonu

4.6. Widok dodawania produktów

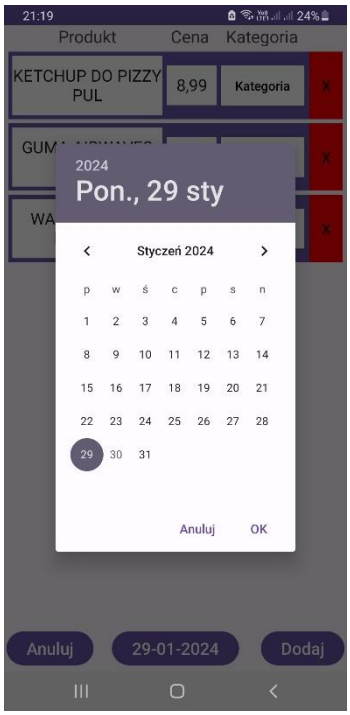
Widok dodawania produktów (Rysunek 36) zawiera listę rozpoznanych w procesie skanowania produktów. Ich nazwy oraz ceny są umieszczone w edytowalnych polach tekstowych, umożliwiających ich modyfikację. Dla każdego produktu dostępny jest przycisk wyboru kategorii, po którego kliknięciu pokazuje się okno dialogowe z listą utworzonych kategorii (Rysunek 37), umożliwiające wybór jednej z nich. Użytkownik ma również możliwość usunięcia wybranego produktu po kliknięciu czerwonego przycisku z symbolem „X”. Przycisk zawierający datę pozwala na wybranie daty zakupu (Rysunek 38). Po kliknięciu przycisku „Dodaj” produkty zostają dodane do bazy danych. Proces dodawania można anulować przyciskiem „Anuluj” lub korzystając z paska nawigacyjnego urządzenia.



Rysunek 36. Widok dodawania produktów



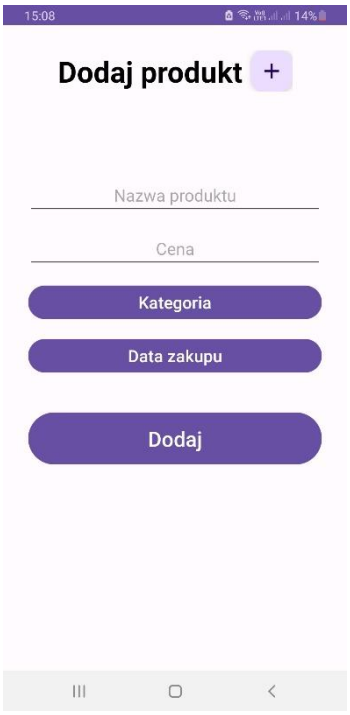
Rysunek 37. Okno dialogowe pozwalające na wybór kategorii podczas dodawania produktów



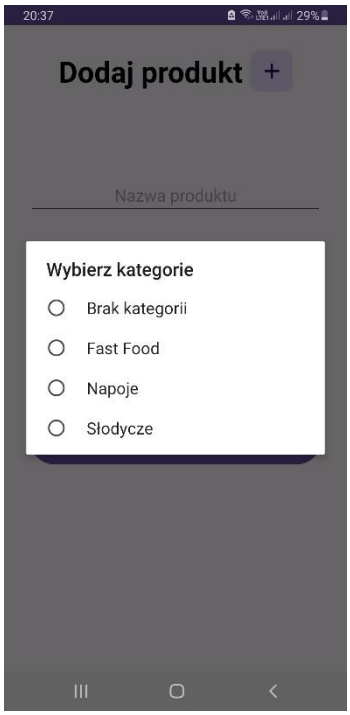
Rysunek 38. Okno dialogowe pozwalające na wybór daty podczas dodawania produktów

4.7. Widok dodawania pojedynczego produktu

Widok dodawania pojedynczego produktu (Rysunek 39) pozwala użytkownikowi na dodanie do bazy danych produktu bez konieczności skanowania paragonu. Zawiera dwa pola tekstowe pozwalające na wprowadzenie nazwy i ceny produktu oraz przyciski służące do wyboru kategorii (Rysunek 40) i daty (Rysunek 41). Produkt zostaje dodany do bazy danych po kliknięciu przycisku „Dodaj". Operację można anulować, korzystając z paska nawigacyjnego urządzenia.



Rysunek 39. Widok dodawania pojedynczego produktu



Rysunek 40. Okno dialogowe pozwalające na wybór kategorii podczas dodawania pojedynczego produktu

Rysunek 41. Okno dialogowe pozwalające na wybór daty podczas dodawania pojedynczego produktu

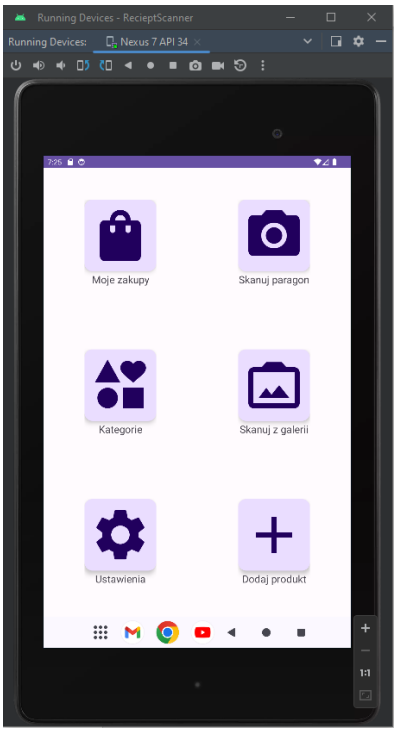
4.8. Zasoby wykorzystane przy tworzeniu interfejsu użytkownika

Podczas tworzenia interfejsu użytkownika zostały wykorzystane cliparty. Clipart to plik graficzny w formie mapy bitowej lub grafiki wektorowej. Środowisko programistyczne Android Studio posiada bazę takich plików, przeznaczonych do użytku w aplikacjach i dostępnych na otwartej licencji Apache License Version 2.0. Możliwe jest bezpośrednie zaimportowanie ich do rozwijanej aplikacji. Wszystkie wykorzystane cliparty pochodzą z tego źródła, a ich zestawienie przedstawiono na Rysunku 42.

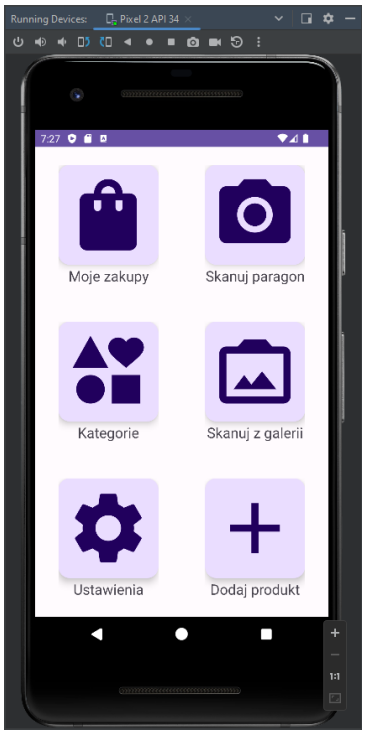
Rysunek 42. Zestawienie wykorzystanych w interfejsie użytkownika clipartów

5. Testy

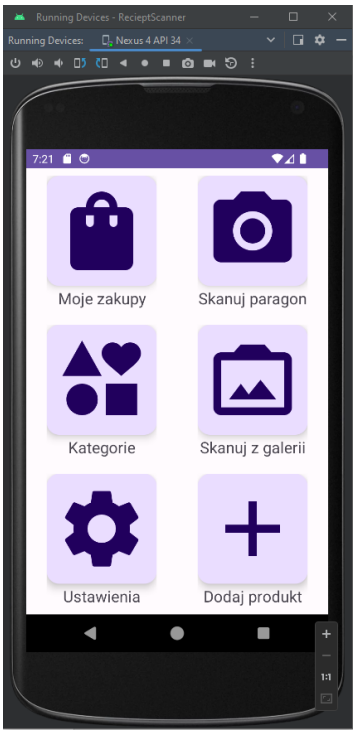
Aplikacja była testowana na emulatorach urządzeń z różnymi rozmiarami i rozdzielczościami ekranów, korzystając z opisanego w podrozdziale 2.1. środowiska programistycznego Android Studio, w celu sprawdzenia skalowalności interfejsu. Wyniki były zadowalające, interfejs pozostawał czytelny i funkcjonalny na wyświetlaczach o przekątnej większej, bądź równej 4 cale. Na rysunkach 43, 44 i 45 został przedstawiony widok startowy aplikacji na emulowanym tablecie Nexus 7 (7,02 cala, 1200x1920), smartfonie Pixel 2 (5 cali, 1080x1920) i smartfonie Nexus 4 (4,7 cala, 768x1280).



Rysunek 43. Widok startowy aplikacji tablecie Nexus 7



Rysunek 44. Widok startowy aplikacji na smarfonie Pixel 2

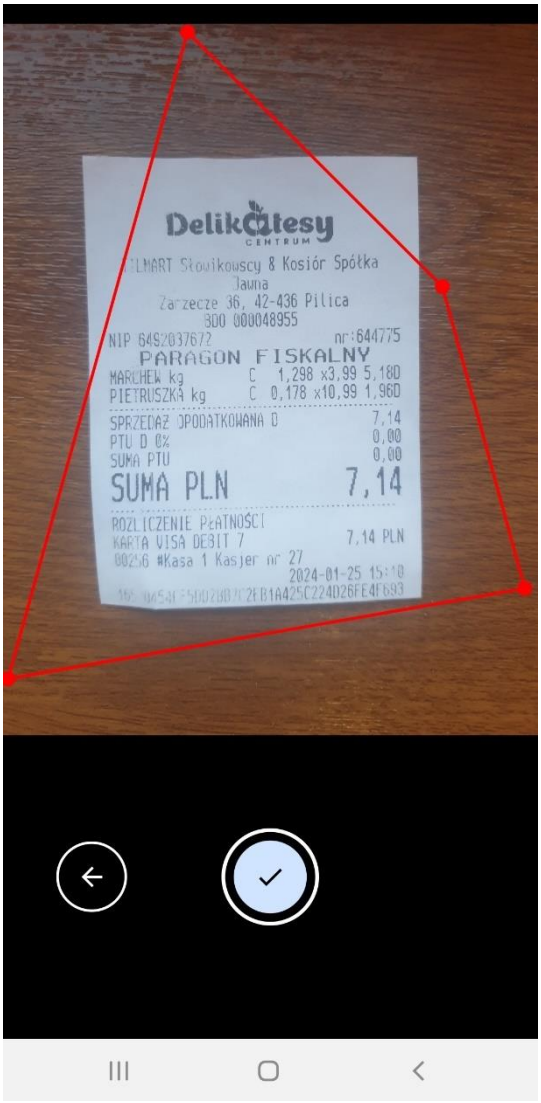


Rysunek 45. Widok startowy aplikacji na smarfonie Nexus 4

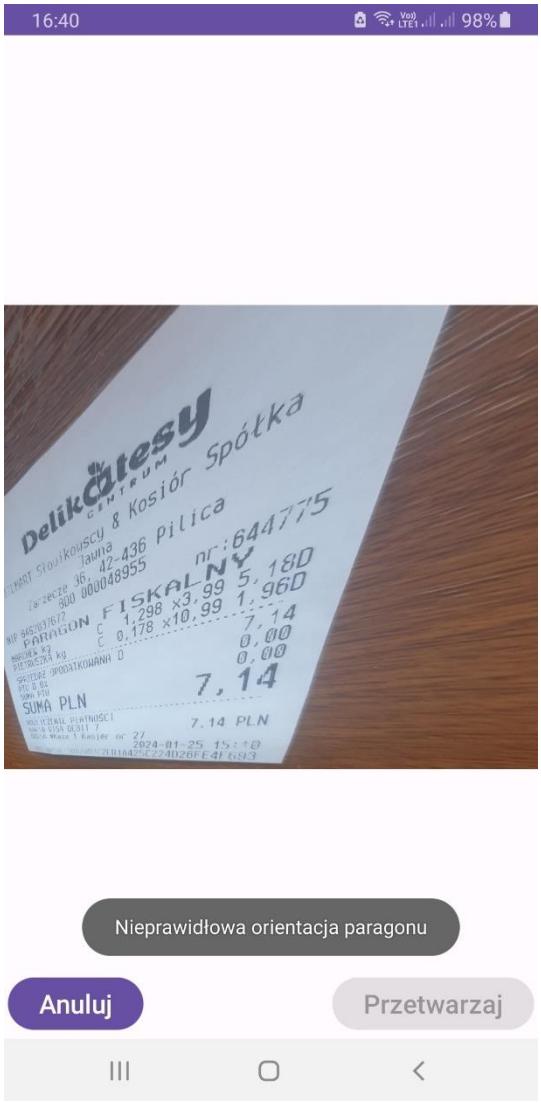
Urządzeniem fizycznym wybranym do testowania aplikacji był Samsung Galaxy S9+. Zrzuty ekranu przedstawione na rysunkach w rozdziale 4. pochodzą z tego urządzenia. Aplikacja była testowana przez autora pracy przez okres dwóch tygodni. Jej działanie jest stabilne, a interfejs responsywny. Do testowania funkcji skanowania paragonów zostało wykorzystane około trzydzieści fizycznych paragonów fiskalnych oraz liczne zdjęcia paragonów znalezione w internecie. Aplikacja dobrze radzi sobie z paragonami fiskalnymi spełniającymi standardy określone we wspominanym w podrozdziale 3.3. rozporządzeniu Ministra Finansów. Do poprawnego rozpoznania konieczne jest również, aby nazwy pojedynczych produktów zawierały się w nie więcej niż jednej linijce, a tekst na paragonie po przycięciu zdjęcia nie był zbytnio pochylony czy zniekształcony. Wykorzystana biblioteka Android Studio Scanner pozwala na dowolny wybór narożników dokumentu i zawsze przycina go do prostokątnego kształtu, co może powodować zniekształcenie obrazu po niepoprawnym wyborze. Przykład nieprawidłowo wybranych narożników przedstawia Rysunek 46, a dokonane dla tego przypadku przycięcie jest pokazane na Rysunku 47, na którym widać również komunikat o nieprawidłowej orientacji paragonu oznaczający, że wykryty tekst jest obrócony o ponad 10° względem osi poziomej. Pojawienie się tego komunikatu oznacza, że

zdjęcia nie będzie można przetworzyć, ponieważ zaimplementowana metoda ekstrakcji nazw i cen wymaga, aby paragon nie był obrócony, gdyż wpływa to na poprawne wyznaczenie obszaru zawierającego nazwy i ceny produktów, jak i kolejność, w jakiej zwracane są rozpoznane wersy tekstu.

Ze względu na małą czcionkę często zdarzają się literówki w rozpoznanym tekście, które użytkownik może skorygować w procesie dodawania produktów. Wyniki skanowania są zależne od jakości zdjęcia, odległości, z jakiej zostało wykonane i warunków oświetleniowych (użycie lampy błyskowej zwykle zwiększa poprawność rozpoznania). Oczywiście skanowany paragon musi być w relatywnie dobrym stanie. Podczas skanowania wyblakłych, zużytych paragonów istnieje duża szansa na niepowodzenia skanowania lub nieprawny jego wynik.



Rysunek 46. Nieprawidłowo wybrane narożniki paragonu



Rysunek 47. Zniekształcenie zdjęcia spowodowane nieprawidłowym wyborem narożników paragonu

6. Podsumowanie

6.1. Efekt końcowy pracy

Celem niniejszej pracy inżynierskiej było opracowanie oraz implementacja aplikacji do skanowania paragonów i kontrolowania budżetu dla systemu Android wersji 8.0 oraz wyższych. Stworzona aplikacja spełnia wszystkie wymagania funkcjonalne i niefunkcjonalne sprecyzowane w podrozdziale 1.5. Dzięki opracowanej metodzie ekstrakcji nazw i cen produktów moja aplikacja pozwala na skanowanie większości paragonów wystawianych przez polskie sklepy, a jej dalszy rozwój może praktycznie wyeliminować obecne ograniczenia w tym zakresie. Możliwość dodawania kupionych produktów do bazy danych oraz przeglądania dokonanych zakupów i związanych z nimi wydatków czyni z aplikacji przydatne narzędzie dla osób chcących kontrolować swój budżet. Możliwość tworzenia własnych kategorii zakupów i rozkładanie paragonu na pojedyncze produkty umożliwia prowadzenie dokładnej ewidencji wydatków. Projekt jest podatny na rozwój i rozbudowę, co czyni z niego dobrą bazę do stworzenia bardziej zaawansowanego rozwiązania, które mogłoby zostać wypuszczone na rynek aplikacji mobilnych.

Kod źródłowy projektu oraz nagranie działania stworzonej aplikacji znajdują się w repozytorium dostępnym pod adresem https://github.com/kamil3214/Praca-Inzynierska.

6.2. Napotkane problemy

Najbardziej istotnym problemem w kwestii implementacji aplikacji, której opracowanie było celem niniejszej pracy, jest fakt, że paragony fiskalne w Polsce nie są ustandaryzowane w wystarczającym stopniu. Przepisy regulujące zakres wymaganych na paragonie informacji i ich kolejność dają sklepom dużą swobodę w kwestii tego, jak finalnie wyglądać będzie wystawiony dokument sprzedaży. Ze względu na występowanie dużych rozbieżności pomiędzy paragonami opracowana metoda ekstrakcji nazw i cen produktów wymaga dużego nakładu pracy i ciągłego rozwoju w celu zagwarantowania jej poprawnego działania dla wszystkich możliwych przypadków. Opracowane rozwiązanie sprawdziłoby się idealnie w sytuacji, w której wszystkie paragony podlegałyby pod jeden, ściśle określony wzór, co ułatwiłoby implementację i pozwoliło na dostosowanie metody do tego konkretnego wzoru, przez co wzrosłaby poprawność i niezawodność rozpoznania.

Kolejnym napotkanym problemem okazała się słaba dostępność i jakość darmowych bibliotek do skanowania dokumentów przeznaczonych na system Android. Wykorzystana biblioteka Android Document Scanner okazała się jedynym wyborem *open source* posiadającym potrzebne dla aplikacji funkcjonalności. Ma ona jednak wiele mankamentów, takich jak wymóg dużego kontrastu pomiędzy skanowanym dokumentem i tłem, zniekształcanie dokumentu w przypadku wyboru niewłaściwych narożników oraz nierozpoznawanie narożników graniczących z krawędzią zdjęcia. Znacznie lepszym wyborem byłoby zastosowanie narzędzi takich jak CamScanner SDK [24], Genius Scan SDK [25] czy Scanbot SDK [26]. Niestety, są to rozwiązania płatne, wymagające zakupu licencji lub subskrypcji.

6.3. Możliwości rozwoju aplikacji

Aplikacja opisana w pracy stanowi bazę projektu, który można rozwijać na wiele sposobów. Dobrym posunięciem mogłoby okazać się udostępnienie jej do testów większemu gronu użytkowników i zapewnienie im możliwości zgłaszania błędów oraz załączania paragonów, których nie jest w stanie poprawnie przetworzyć. Podczas rozwoju kolejnych wersji można

zaimplementować metody radzenia sobie z paragonami, które zawierają nazwy produktów w więcej niż jednej linijce oraz zapewnić obsługiwanie większej ilości wzorów rabatów. W celu zwiększenia atrakcyjności aplikacji dla grupy jej docelowych odbiorców, osób zainteresowanych kontrolą swoich wydatków, można zaimplementować przydatne w tym aspekcie funkcjonalności, takie jak na przykład tworzenie list zakupów czy śledzenie aktualnych promocji sieci sklepów i marketów.

6. Bibliografia

[1] Historia optofonu, https://en.wikipedia.org/wiki/Optophone; dostęp 15.01.2024
[2] Historia optycznego rozpoznawania znaków, https://www.pitneybowes.com/content/dam/pitneybowes/uk/en/shipping-and-mailing/e-invoicing/Blog_E-invoicing-The_Brief_History_of_OCR.pdf?itid=lk_inline_enhanced-template; dostęp 15.01.2024
[3] Binarna macierz wyodrębnionych znaków, https://www.researchgate.net/figure/The-binary-matrix-for-the-extracted-character-where-1-represents-background-and-0_fig9_279806456; dostęp 15.01.2024
[4] Historia systemu Android, https://www.britannica.com/technology/Android-operating-system; dostęp 17.01.2024
[5] Liczba użytkowników systemu Android, https://www.bankmycell.com/blog/how-many-android-users-are-there; dostęp 17.01.2024
[6] Liczba aplikacji dostępnych w Sklepie Play, https://www.bankmycell.com/blog/number-of-google-play-store-apps/; dostęp 17.01.2024
[7] Licencjonowanie systemu Android, https://source.android.com/docs/setup/about/licenses; dostęp 17.01.2024
[8] Architektura systemu Android, https://developer.android.com/guide/platform; dostęp 17.01.2024
[9] Historia środowiska programistycznego Android Studio, https://en.wikipedia.org/wiki/Android_Studio; dostęp 17.01.2024
[10] Historia języka programowania Java, https://en.wikipedia.org/wiki/Java_(software_platform); dostęp 18.01.2024
[11] Wirtualna maszyna Javy, https://en.wikipedia.org/wiki/Java_virtual_machine; dostęp 18.01.2024
[12] SQLite, https://en.wikipedia.org/wiki/SQLite; dostęp 18.01.2024
[13] Dokumentacja klasy SQLiteDatabase, https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase; dostęp 18.01.2024
[14] Dokumentacja klasy SQLiteOpenHelper, https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper; dostęp 18.01.2024
[15] Google ML Kit, https://developers.google.com/ml-kit/guides?hl=pl; dostęp 19.01.2024
[16] Lista języków obsługiwanych przez ML Kit Text Recognition v2, https://developers.google.com/ml-kit/vision/text-recognition/v2/languages?hl=pl; dostęp 19.01.2024
[17] Opis funkcjonalności ML Kit Text Recognition v2, https://developers.google.com/ml-kit/vision/text-recognition/v2?hl=pl; dostęp 23.01.2024
[18] Biblioteka Android Document Scanner, https://github.com/WebsiteBeaver/android-document-scanner; dostęp 25.01.2024
[19] Biblioteka OpenCV, https://opencv.org/; dostęp 25.01.2024
[20] Definicja wyrażenia regularnego, https://pl.wikipedia.org/wiki/Wyra%C5%BCenie_regularne; dostęp 27.01.2024
[21] Dokumentacja klasy Pattern, https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html; dostęp 27.01.2024
[22] Dokumentacja klasy Matcher, https://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html; dostęp 27.01.2024

[23] Dokumentacja klasy Activity, https://developer.android.com/reference/android/app/Activity; dostęp 27.01.2024
[24] CamScanner SDK, https://dev.camscanner.com/; dostęp 10.02.2024
[25] Genius Scan SDK, https://geniusscansdk.com/; dostęp 10.02.2024
[26] Scanbot SDK, https://scanbot.io/; dostęp 10.02.2024