

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

Projekt dyplomowy

*Projekt i implementacja aplikacji do zarządzania paragonami*

*Design and implementation of receipt management application*

Autor: *Julia Nowak*

Kierunek studiów: *Automatyka i robotyka*

Opiekun pracy: *dr inż. Marta Kraszewska*

Kraków, 2024

Spis treści

[1. Wstęp 2](#_Toc153159723)

[1.1. Wprowadzenie 2](#_Toc153159724)

[1.2. Cel i zakres pracy 3](#_Toc153159725)

[2. Istota przetwarzania i analizy obrazów w aplikacji 4](#_Toc153159726)

[2.1. Porównanie silników OCR dostępnych na rynku 5](#_Toc153159727)

[2.2. Wybrany silnik OCR - Tesseract 7](#_Toc153159728)

[2.3. Problematyka i przetwarzanie wstępne 9](#_Toc153159729)

[3. Rozwiązania dostępne na rynku 13](#_Toc153159730)

[3.1. PanParagon 14](#_Toc153159731)

[3.2. MintCloud 15](#_Toc153159732)

[3.3. Billy 16](#_Toc153159733)

[3.4. Wybór istotnych funkcjonalności 16](#_Toc153159734)

[4. Narzędzia programistyczne 17](#_Toc153159735)

[5. Implementacja 19](#_Toc153159736)

[5.1. Baza danych aplikacji 19](#_Toc153159737)

[5.2. Profile użytkowników 21](#_Toc153159738)

[5.3. Przetwarzanie obrazów i implementacja OCR 23](#_Toc153159739)

[5.4. System grup i podziału wydatków 24](#_Toc153159740)

[5.5. Kategoryzacja wydatków 28](#_Toc153159741)

[5.6. Webscraping stron popularnych sklepów 30](#_Toc153159742)

[5.7. System wiadomości 32](#_Toc153159743)

[5.8. Gwarancje, terminy zwrotów i wydatki powtarzalne 35](#_Toc153159744)

[5.9. Statystyki i wykresy 37](#_Toc153159745)

[5.10. Listy zakupowe 38](#_Toc153159746)

[5.11. Dostosowanie aplikacji do systemu Android 40](#_Toc153159747)

[6. Testowanie aplikacji 43](#_Toc153159748)

[6.1. Biblioteka danych testowych 43](#_Toc153159749)

[6.2. Miary stosowane do oceny wyników OCR 44](#_Toc153159750)

[6.3. Testowanie zaimplementowanego algorytmu OCR 46](#_Toc153159751)

[6.4. Testowanie manualne **Błąd! Nie zdefiniowano zakładki.**](#_Toc153159752)

[7. Podsumowanie 51](#_Toc153159753)

[8. Bibliografia 52](#_Toc153159754)

[Załączniki 55](#_Toc153159755)

# Wstęp

## Wprowadzenie

We współczesnym świecie coraz więcej ludzi zaczyna borykać się z problemem nadmiernego wydawania pieniędzy. Powodem takiego stanu rzeczy jest nie tylko konsumpcyjny tryb życia, powszechny wśród państw wysokorozwiniętych, ale także inflacja, która szczególnie w ostatnim czasie stała się istotnym zagrożeniem dla budżetu domowego. Rosnące ceny produktów podstawowej potrzeby oraz wzrost opłat sprawia, iż wiele osób decyduje się na zwiększenie kontroli nad wydatkami i wprowadzenie pewnych ograniczeń finansowych.

Kontrolowanie budżetu jest ściśle związane z zarządzaniem paragonami i fakturami – to one stanowią główne źródło informacji na temat dokonywanych transakcji. Bardzo często pierwszym krokiem do zmniejszenia wydatków jest ich gromadzenie i analizowanie. Ze względu na nietrwałą formę dokumentów i ich szybką degradację, często utrwalane są w formie elektronicznej. W takiej postaci ich analiza staje się prostsza, a tym samym umożliwia ich właścicielowi świadome ograniczenie ilości wydawanych pieniędzy.

Zarządzanie budżetem nie należy jednak do łatwych, gdyż wymaga istotnego nakładu pracy w celu osiągnięcia spodziewanych rezultatów. Jednym z popularnych rozwiązań jest arkusz Microsoft Excel, który umożliwia zapisywanie interesujących nas informacji. Jest on jednak narażony na błędy ludzkie, spowodowane chociażby przez nieuwagę, które mogą wpływać na wyniki i poprawność ich analizy. Dodatkowo osoby, które wcześniej nie zajmowały się analizowaniem i interpretacją takich danych mogą nie wykryć pewnych zależności, które mogłyby okazać się dla nich korzystne. Z tego względu, obecnie można zaobserwować wzmożony popyt na aplikacje, które usprawniają zarządzanie budżetem domowym i nie tylko.

Rosnące zainteresowanie tematem doprowadziło do tego, iż rozpoczęto prace nad zautomatyzowaniem sposobu zarządzania paragonami. Pierwsze wydane aplikacje zyskały dużą popularność w stosunkowo krótkim czasie, co z kolei poskutkowało wzrostem liczby rozwiązań na rynku. Aktualnie istnieje wiele aplikacji, które reprezentując alternatywne podejścia do zarządzania budżetem. Istnieją rozwiązania, które podobnie jak Microsoft Excel pozwalają na ręczne wpisywanie wszystkich informacji, ale są dostępne również takie, które starają się zautomatyzować cały proces. Duża część z nich jest jednak dostępna tylko w języku angielskim, co może być ograniczeniem dla potencjalnych użytkowników, nieznających tego języka.

Na polskim rynku można natknąć się na aplikacje z wprowadzaniem danych ręcznie, ale również takie, które zapewniają większą automatyzację i których jest stosunkowo niewiele. Te drugie są jednak znacznie bardziej popularne. Nie tylko ułatwiają zarządzanie paragonami, ale także organizują wszelkie procesy związane z ich powstawaniem, takie jak chociażby tworzenie list zakupowych, przeglądanie gazetek sklepów czy wskazywanie najbardziej korzystnych cen danych produktów. Niektóre umożliwiają także przeglądanie statystyk czy tworzenie grup w celu lepszej organizacji całego procesu. Najbardziej pożądanym elementem aplikacji do zarządzania paragonami jest jednak algorytm OCR (*ang. Optical Character Recognition*)*.*

Algorytm ten znacznie ogranicza nakład pracy użytkownika poprzez automatyczne odczytywanie najistotniejszych informacji z paragonów. Umożliwia to nie tylko wykorzystanie danych do statystyk, ale także chociażby zaklasyfikowanie transakcji do danej kategorii. OCR nie zawsze daje wyniki poprawne, ale przy rozbudowanej bazie danych, odpowiednim wstępnym i końcowym oraz umożliwieniu edycji danych użytkownikowi, znacznie przyspiesza pracę. Zmniejsza się również prawdopodobieństwo wystąpienia błędu, gdyż dane są sprawdzane pierwotnie przez program, a wtórnie – przez użytkownika.

Tak więc oczekiwania potencjalnych użytkowników aplikacji do zarządzania paragonami sięgają znacznie dalej niż początkowo mogłoby się wydawać. Oprócz funkcji podstawowego przechowywania paragonów i ręcznego wpisywania danych, użytkownicy oczekują czegoś więcej – zakupowego asystenta, jak to określił w swoim artykule [1] Michał Wachowski. W świecie, w którym czas jest najcenniejszą walutą aplikacja musi być szybka, prosta w obsłudze i w wysokim stopniu zautomatyzowana.

## Cel i zakres pracy

Celem pracy było stworzenie projektu aplikacji do zarządzania paragonami, a także jej implementacja zgodnie z przyjętymi założeniami. Projekt powstał docelowo dla urządzeń z systemem Android (w szczególności - Android 12), jednak dostęp do niej można uzyskać również poprzez przeglądarkę internetową. Do prawidłowego działania aplikacji niezbędny jest dostęp do Internetu. Przyjęto również, iż odczytywane paragony są w języku polskim, co jest istotne ze względu na różnicę w znakach alfabetu polskiego i angielskiego.

W rozdziale 2 opisane zostały wybrane zagadnienia z obszaru przetwarzania obrazów cyfrowych, które znalazły swoje zastosowanie w projekcie. Wyszczególniono również dostępne na rynku rozwiązania, pozwalające na odczyt tekstu z obrazów. Ze względu na wykorzystanie silnika Teserract OCR [2] w pracy, został on opisany najbardziej szczegółowo. W rozdziale opisano również najczęściej pojawiające się problemy przy przeprowadzaniu odczytu tekstu z obrazu oraz propozycje, które pozwalają zniwelować negatywny wpływ różnych czynników na jakość wyników.

Koncepcja realizacji powstała w oparciu o rozwiązania dostępne na rynku – w szczególności aplikację PanParagon [3]. Niektóre funkcjonalności zostały zaczerpnięte również z innych aplikacji, które szczegółowo opisano w rozdziale 3. Tam też dokonano wyboru najistotniejszych elementów oraz wskazano wady i zalety każdego z analizowanych rozwiązań.

Rozdział 4 zawiera opis wszystkich technologii wykorzystanych przy tworzeniu aplikacji z wyszczególnieniem najbardziej istotnych bibliotek. Krótko wskazano również, do czego dana technologia została użyta w projekcie.

W rozdziale 5 znalazły się informacje na temat procesu projektowania i implementacji aplikacji do zarządzania paragonami. Szczegółowo opisano tam bazę danych oraz objaśniono znaczenie utworzonych modeli, ich atrybutów oraz występujących pomiędzy nimi relacji. Aplikacja została podzielona na sekcje, które są omawiane kolejno w następujących po sobie podrozdziałach. Całość projektu rozplanowano w taki sposób, aby każda z nich realizowała co najmniej jedno z założeń głównych oraz kilka pobocznych, wynikających często z potrzeb potencjalnego użytkownika. Do najważniejszych wytycznych należały:

* wdrożenie OCR do aplikacji w celu odczytywania danych z paragonów,
* zaimplementowanie systemu przypominania, który wyświetla informacje o zbliżającym się terminie końca gwarancji lub okresu zwrotu,
* utworzenie sekcji statystyk, zawierającej podsumowanie wszystkich zebranych danych w danym okresie czasu,
* zapewnienie możliwości przeglądania wydatków z danego okresu czasu,
* utworzenie systemu grup i wiadomości,
* zapewnienie możliwości kategoryzowania paragonów i opłat,
* zaimplementowanie logiki list zakupowych oraz możliwości ich udostępniania,
* zapewnienie użytkownikowi dostępu do aktualnych ofert promocyjnych wybranych sklepów.

Oprócz tego, w rozdziale opisano również dostępne możliwości personalizacji profilu użytkownika, a także sposób wykorzystany do utworzenia aplikacji mobilnej na bazie istniejącej strony internetowej.

Rozdział 6 skupia się na testowaniu funkcjonalności opisanych we wcześniejszych rozdziałach. Przede wszystkim sprawdzono jakość algorytmu, który współpracował z silnikiem Tesseract przy odczycie danych z obrazów. Przedstawiono metryki, które pozwoliły na ocenę wyników w porównaniu do informacji wprowadzonych przez człowieka. Rozplanowano również scenariusze testowe, mające na celu wychwycenie błędów, powstałych w trakcie implementacji.

# Istota przetwarzania i analizy obrazów w aplikacji

Przetwarzanie obrazów cyfrowych jest dziedziną niezwykle rozległą. Ciągły rozwój zawdzięcza potrzebie zobrazowania maszynie tego, co jest bez większych trudności dostrzegalne dla ludzkiego oka. Człowiek widząc dany obraz jest w stanie niemal natychmiastowo stwierdzić, co on przedstawia, wykorzystując do tego jedynie zmysł wzroku. Komputer natomiast musi opierać się na dostarczonych mu danych wejściowych. W swojej książce Tadeusiewicz [4] już na początku odnosi się do tego zjawiska, pisząc, iż przetwarzanie obrazów cyfrowych dla komputera jest czymś o wiele cięższym i bardziej złożonym niż dla człowieka. Warto jednak zauważyć, iż niesie to ze sobą korzyści współmierne do wzrostu wymaganego nakładu pracy.

Aplikacja tworzona w ramach pracy jest silnie osadzona w tematyce związanej z przetwarzaniem obrazów cyfrowych. Zgodnie z założeniami, użytkownicy mają możliwość dodawania zdjęć paragonów, na podstawie których, odczytywane są najistotniejsze informacje. W celu realizacji tej funkcjonalności, zdecydowano się na wykorzystanie ogólnodostępnego silnika OCR, rozwijanego przez firmę Google – Tesseracta [2]. Warto jednak zwrócić uwagę, iż sam silnik nie zawsze daje poprawne wyniki, a w kontekście założeń samej aplikacji, bez dodatkowych funkcji nie daje zadawalających rezultatów. Z tego też względu, podjęte działania skupiały się w głównej mierze na utworzeniu funkcji odpowiedzialnych za przetwarzanie wstępne (*ang. preprocessing*) oraz przetwarzanie końcowe (*ang. postprocessing*) obrazu wejściowego. Głównym celem było przeprowadzenie takich operacji, aby zwiększyć jakość odczytanego tekstu oraz otrzymać jak najwięcej pożądanych danych z paragonu.

## Porównanie silników OCR dostępnych na rynku

Na rynku aktualnie dostępne są różne aplikacje, które wykorzystują OCR do odczytywania tekstu z obrazów. W przeważającej większości nie służą one jednak do analizy odczytanych danych, tak jak chociażby aplikacje wymienione w rozdziale 2, a dla użytkownika istotne jest otrzymanie jedynie samego tekstu. Tekst ten może być odczytywany przykładowo tylko ze skanów i dokumentów PDF o określonej i dobrze rozpoznawalnej czcionce. Może być także tworzony na podstawie obrazów otoczenia, takich jak chociażby zdjęcie tablicy rejestracyjnej samochodu. Określenie przeznaczenia aplikacji jest pierwszym krokiem do wyboru odpowiedniego silnika OCR.

Na rynku dostępnych jest wiele różnych rozwiązań, które często porównywane są ze sobą w różnych artykułach. Zdecydowałam się na wybór jednego z nich [5], w którym zobrazowano zachowanie każdego z trzech silników w formie przejrzystych statystyk. Autor tekstu wybrał najczęściej wykorzystywane silniki OCR i przeprowadził eksperyment na 3 tys. obrazów. W zestawieniu znalazły się:

* PyTessseract (Tesseract)
* EasyOCR
* Amazon Textract

Analizując przedstawione w artykule wykresy można dostrzec, iż EasyOCR wypadł najgorzej w porównaniu do dwóch pozostałych silników. Dla każdego z przeprowadzonych testów uzyskał on największą wartość miary odmienności napisów, zwanej odległością Levenshteina. Miara ta została wykorzystana przez autora i polega na określeniu ilości operacji niezbędnych do przekształcenia danego słowa w taki sposób, aby pasowało do słowa wzorcowego. Im większe wartości osiąga, tym więcej operacji musi zostać wykonanych, co świadczy o gorszej jakości rozpoznania tekstu.

Przyglądając się wynikom pozostałych dwóch silników OCR na Rys. 1 można zauważyć, że Tesseract w przeważającej liczbie przypadków osiągnął niższe wartości średniej odległości Levenshteina. Amazon Textract, o czym wspomina sam autor, zyskuje przewagę nad Tesseractem, jeżeli zamiast średnich wartości zostaną wzięte pod uwagę wartości median. Można więc wysnuć wniosek, iż Tesseract to silnik OCR, który wykazuje mniejsze wartości odchylenia standardowego, a jego wyniki utrzymują się na stabilnym poziomie. Przy Amazon Textract natomiast, wyniki przyjmują najczęściej wartości skrajnie pozytywne lub negatywne.

Oprócz pomiarów dokładności, autor artykułu dodatkowo zdecydował się na dokonanie pomiaru szybkości silnika EasyOCR oraz PyTesseract. Wyniki tego badania przedstawiono w Tab.1. Amazon Textract nie został uwzględniony, gdyż musiałby zostać uruchomiony w chmurze, a nie jak w przypadku pozostałych dwóch lokalnie. Mogłoby to zaburzyć odczyty i dać niemiarodajne wyniki. Dlatego też, porównane zostały tylko pierwsze dwa silniki OCR. Rezultaty pomiaru jednoznacznie określiły znaczącą przewagę EasyOCR nad Tesseractem.

Podsumowując powyższe wnioskowanie można stwierdzić, iż:

* Amazon Textract najlepiej sprawdzi się przy zadaniach, w których akceptowalny jest duży rozrzut rozwiązań przy zwiększonej szansie na otrzymanie lepszego jakościowo rozwiązania oraz przy obrazach o rozdzielczości 32 pikseli.
* Tesseract najlepiej sprawdzi się przy zadaniach, gdzie najistotniejsza jest powtarzalność rozwiązań dobrych jakościowo, a nie dążenie do osiągnięcia pojedynczych, ale lepszych rozwiązań.
* EasyOCR najlepiej sprawdzi się przy zadaniach, gdzie przede wszystkim istotna jest szybkość wykonania, kosztem nieco gorszych wyników.

Z punktu widzenia aplikacji, najlepszym wyborem okazuje się więc silnik OCR Tesseract, gdyż głównym celem jest zapewnienie bardziej uniwersalnej detekcji, która daje stosunkowo dobre wyniki dla różnorodnych paragonów.

Obraz zawierający tekst, diagram, zrzut ekranu, Prostokąt

Opis wygenerowany automatycznie

Rys. 1 Rezultaty testów przeprowadzonych na wybranych silnikach OCR (źródło: [7])

Tab. 1 Porównanie szybkości badanych silników OCR (opracowano na podstawie [7])

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Liczba obrazów | Czas (s) | FPS | ms/obraz | Urządzenie |
| EasyOCR | 500 | 17,37 | 28,79 | 34,74 | GPU (T4) |
| EasyOCR | 500 | 71,49 | 6,99 | 142,98 | CPU |
| PyTesseract | 500 | 131,65 | 3,8 | 263,31 | CPU |

## Wybrany silnik OCR - Tesseract

Tesseract [2] jest silnikiem OCR, który swe początki datuje na rok 1984. Początkowo miał on formę projektu pracy doktorskiej Raya Smitha, sponsorowanego przez firmę Hewlett-Packard, znaną powszechnie jako HP. W 2005 roku został udostępniony publicznie w formie otwartego źródła. W latach 2006 - 2018 był rozwijany przez firmę Google. Tesseract został napisany przy pomocy języków C i C++. Z czasem powstały różne *wrappery,* które umożliwiły wykorzystanie silnika w innych językach programowania. Przykładem może być PyTesseract [6], który był wykorzystany w kodzie źródłowym niniejszej pracy.

Tesseract swą dużą popularność zawdzięcza nie tylko stosunkowo wysokiej jakości odczytu, ale także temu, iż obsługuje ponad 100 różnych języków oraz ponad 35 różnych rodzajów pisma. Dodatkowo, silnik zapewnia możliwość nauczenia modelu na własnym zestawie danych, co umożliwia zwiększenie rozpoznawalności danej czcionki lub dotąd nierozpoznawalnego rodzaju pisma.

Architektura silnika OCR Tesseract została opisana przez jego twórcę, Raya Smitha, w swoich publikacjach [7][8]. Opisywane przez nią etapy są wykonywane na zasadzie szeregowego przepływu (*ang. pipeline*) z modyfikacją w postaci rozgałęzienia z wykorzystaniem LSTM, które zostało wprowadzono dopiero w wersji Tesseract 4*.* Zostało to uwidocznione w sposób schematyczny na Rys. 2.

Obraz zawierający tekst, zrzut ekranu, Czcionka, Prostokąt

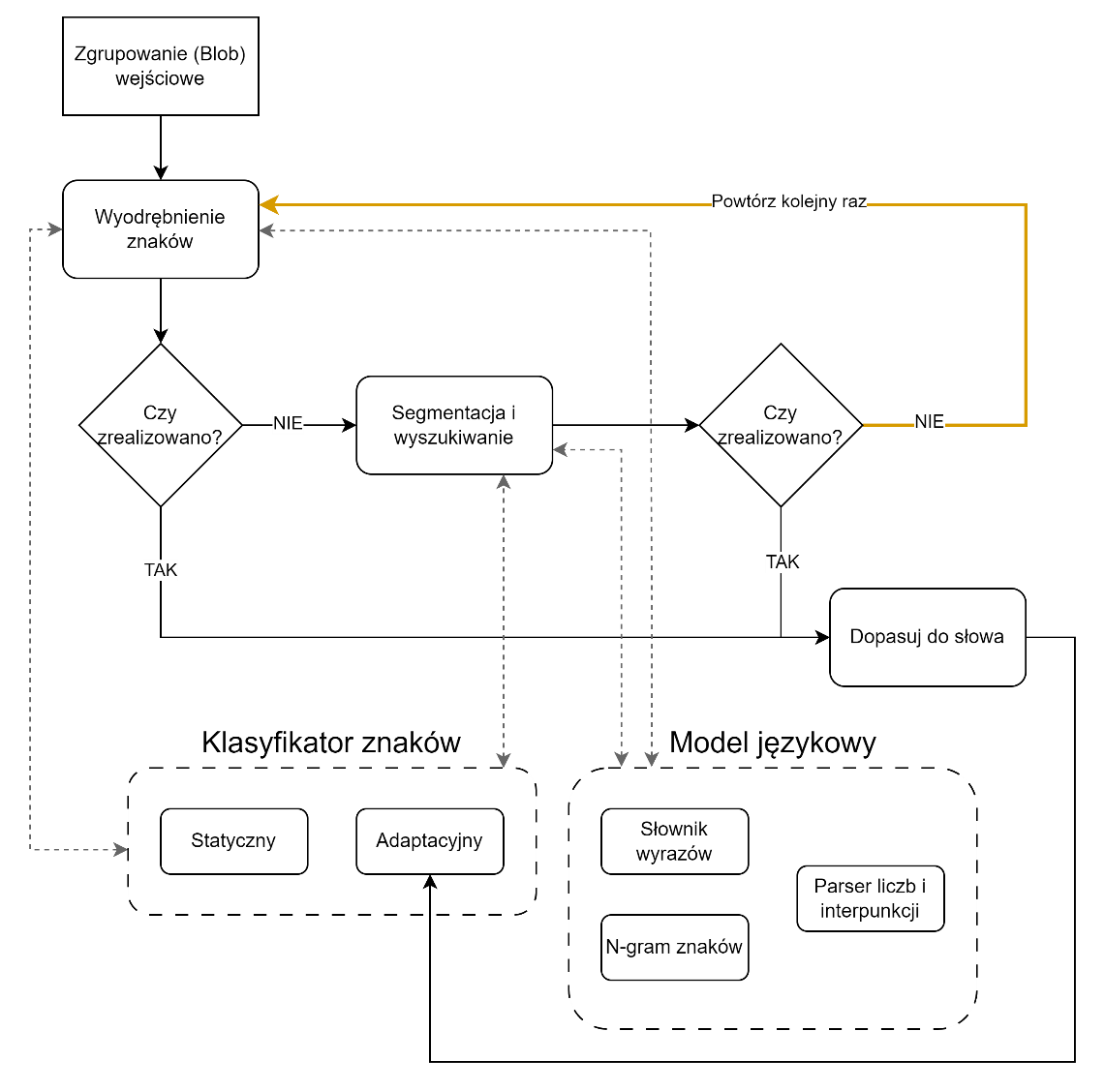
Opis wygenerowany automatycznie

Rys. 2 Schemat architektury systemu Tesseract (opracowanie własne na podstawie [7] i [8])

W pierwszym etapie wykonywana jest binaryzacja obrazu wejściowego, który zgodnie z dokumentacją, może być w formacie PNG, JPEG lub TIFF. Następnie, przetworzony obraz zostaje poddany analizie składowych łączonych (*ang. connected-component analysis*), w trakcie której wyodrębnione zostają obszary tekstu, które w późniejszej fazie są grupowane w formacje zwane blobami (*ang.blobs*).

W kolejnym etapie, te zgrupowania są sprawdzane pod kątem odstępów pomiędzy znakami i proporcjach tekstu. Na ich podstawie, z blobów ekstraktowane są słowa, które następnie przechodzą rozpoznanie z wykorzystaniem LSTM (*ang.* *long short-term memory*). Jest to rekurencyjna sieć neuronowa (RNN), która zachowuje informacje o istotnych zależnościach przez długi okres czasu.

Jeśli dane słowo jest poprawne, wówczas od razu staje się częścią tekstu wyjściowego, a jeśli nie – wówczas przechodzi rozpoznanie 2-etapowe. Pierwszy etap rozpoczyna się od odpowiedniego rozdzielenia znaków i przeprowadzenia segmentacji z wykorzystaniem modelu językowego. W takim modelu zawarte są informacje na temat słów, n-gramów i separatorów obecnych w danym języku. Na ich podstawie możliwe jest wyodrębnienie i rozpoznanie słowa, które następnie jest wykorzystywane do uczenia klasyfikatora adaptacyjnego. Druga część rozpoznania przebiega w identyczny sposób, z tym, że klasyfikator został wytrenowany i ma szansę na osiągnięcie lepszych wyników.



Rys. 3 Schematyczne przedstawienie 2-etapowego rozpoznawania słów (opracowano na podstawie [7] i [8])

W ostatnim etapie następuje ponowne sprawdzenie wysokości tekstu (*ang. x-height*) oraz obszarów oznaczonych jako niejasne (*ang. fuzzy spaces*) w celu odnalezienia ewentualnie pominiętego, mniejszego tekstu. Słowa po 2-etapowym rozpoznaniu są zawsze poddawane operacji wyszukiwania słów tworzących pary (*ang. word bigram*) w celu polepszenia predykcji.

## Problematyka i przetwarzanie wstępne

Podejmując się tworzenia aplikacji do zarządzania paragonami, należy być świadomych potencjalnych problemów, które mogą się wiązać z odczytem danych z obrazów. Już w rozdziale 1 wspomniano, iż dla maszyny jest to niezwykle złożone zadanie w porównaniu do tego, z jaką łatwością wykonuje je człowiek. Z pozoru niepozorne czynniki, takie jak chociażby zagięcie paragonu, niejednolity kolor tła czy też światło padające pod nieodpowiednim kątem, mogą całkowicie uniemożliwić odczyt danych.

Tematyka ta została poruszona w dokumentacji silnika OCR Tesseract [2]. Wskazano tam różne możliwości poprawienia jakości tekstu wyjściowego, poprzez stosowanie metod z zakresu przetwarzania obrazów, segmentacji strony czy też tworzenia bazy poszukiwanych wzorów czy słów. Jednocześnie, autorzy zaznaczyli, iż trenowanie silnika poskutkuje poprawą rezultatów przeważnie tylko wtedy, gdy odczytywana czcionka ma nietypowy wygląd lub tekst jest językiem nieobsługiwanym. W kontekście aplikacji do zarządzania paragonami można więc założyć, że trenowanie nie przyniesie znaczącej poprawy odczytu informacji z paragonu. Przyczyną takiego stanu rzeczy jest fakt, iż:

* paragony nie mają określonej formy – zarówno czcionka, jak i ułożenie informacji na zdjęciu może się znacząco różnić, w zależności od sklepu, w którym dokonano zakupu,
* czcionka na paragonie zazwyczaj w przeważającej większości nie ma nietypowego wyglądu ani nie przypomina pisma ręcznego,
* język polski jest obsługiwany przez silnik OCR Tesseract, co wyklucza potrzebę uczenia go polskich znaków.

Dlatego też, wykorzystywany silnik OCR nie został wytrenowany pod kątem lepszego rozpoznawania paragonów z bazy. Do poprawy jakości odczytu zostały jednak wykorzystane wszystkie metody, wskazane przez autorów dokumentacji.

Przetwarzanie obrazów wejściowych, które są przekazywane do analizy silnikowi OCR jest opcjonalne, gdyż Tesseract sam ma możliwość odpowiedniego dostosowania obrazu do swoich wymagań, z wykorzystaniem biblioteki Leptonica. W dokumentacji zaznaczono jednak, że nie zawsze przetwarzanie to jest skuteczne. Stąd też wynika potrzeba stosowania przetwarzania wstępnego.

W przypadku odczytu danych z paragonów, możliwe jest pominięcie inwersji obrazu, wskazanej przez autorów tekstu. Znacząca większość z nich posiada bowiem formę czarnych liter na białym tle. Coraz częściej spotykane stają się również ekologiczne paragony szare, które mimo iż nie spełniają tego założenia, przy odpowiednim oświetleniu i przeprowadzeniu binaryzacji, również stają się możliwe do odczytania.

Zgodnie z informacjami zawartymi w dokumentacji, Tesseract daje najlepsze rezultaty dla obrazów wejściowych o rozdzielczości co najmniej 300 dpi (*ang. dots per inch*), a więc jest przystosowany przede wszystkim do dokumentów skanowanych. Zdjęcia zrobione telefonem przyjmują zazwyczaj wartość 72dpi. Jeśli zostaną przesłane na przykład poprzez komunikator internetowy, a następnie pobrane, wartość ta zwiększa się do 96 dpi. Te wartości znacznie się różnią, co może prowadzić do problemów z poprawnym rozpoznaniem tekstu. Efekt ten można zniwelować poprzez skalowanie obrazu.

Skalowanie [9] polega na zmianie wymiarów obrazu wejściowego w taki sposób, aby zachować stosunek wysokości do szerokości. Operacja ta jest przeprowadzana w celu powiększenia lub pomniejszenia obrazu do dokładniejszej analizy tekstu, który znajduje się na obrazie. Jeśli przyjmiemy współczynnik skali jako , współrzędne piksela jako , współrzędne piksela początkowego jako oraz nowe współrzędne piksela po skalowaniu jako , wówczas zależność będzie określona jako:

|  |  |
| --- | --- |
|  | (2.3.1) |

Jeśli współczynnik skali *s* > 1, wówczas obraz zostaje powiększony, co jednocześnie wpływa na spadek wartości dpi. Jest to skutek tego, iż po powiększeniu obszaru obrazu rastrowego, liczba pikseli nie zwiększa się w stosunku do obrazu oryginalnego – zostają one jednak rozszerzone, czego efektem jest pogorszenie jakości. W odwrotnej sytuacji, gdy współczynnik skali *s* < 1, obraz zostaje zmniejszony, a wartość dpi wzrasta. Istniejące piksele ulegają kompresji, gdyż dysponują mniejszą przestrzenią obrazu.

Innym elementem, który znacząco wpływa na odczyt tekstu, jest jego poprawna orientacja. Tesseract OCR nie da zadawalających rezultatów dla obrazów, które są w niewłaściwej pozycji. Dlatego też ważne jest, aby określić kąt pod którym znajduje się tekst na obrazie, a następnie dokonać odpowiedniej rotacji.

Użytkownicy aplikacji w większości przypadków są w stanie prawidłowo określić orientację paragonu. Można więc przyjąć, iż obraz wejściowy przyjmuje pozycję prawidłową i czytelną dla człowieka. Przy robieniu zdjęcia często zdarza się jednak, że zostanie ono przekrzywione o pewien kąt, co negatywnie wpływa na wyniki detekcji. Do wykrycia tego kąta najczęściej wykorzystywana jest transformacja Hougha.

Transformacja Hougha [10] to metoda wykorzystywana do wyszukiwania na obrazach linii prostych lub innych kształtów regularnych. Istota funkcjonowania opiera się o fakt, iż linia na obrazie może zostać opisana jako w postaci funkcji liniowej w układzie kartezjańskim

|  |  |  |
| --- | --- | --- |
|  |  | (2.3.2) |

Forma tego równania może zostać wykorzystana w biegunowym układzie współrzędnych, poprzez wykorzystanie zależności trygonometrycznych. Dla oznaczeń przyjętych na Rys. 4, można zapisać, że

|  |  |  |
| --- | --- | --- |
|  |  | (2.3.3) |

Obraz zawierający linia, Wykres, stok

Opis wygenerowany automatycznie

Rys. Obraz linii w układzie współrzędnych z przyjętymi oznaczeniami (źródło [10]))

Rotacja [9] polega natomiast na ustawieniu obrazu wejściowego w określonej pozycji w taki sposób, żeby litery na obrazie były jak najlepiej rozpoznawane przez algorytm. Jeśli przyjmiemy, że docelowy kąt obrotu to , wówczas nowe koordynaty każdego z pikseli obrazu, oprócz piksela będącego początkowym punktem obrotu , będą opisane równaniami

|  |  |  |
| --- | --- | --- |
|  |  | (2.3.1) |

Nowy obraz, dla punktu początkowego układu współrzędnych (0, 0), może być zatem opisany poprzez odpowiedni wektor oraz macierz rotacji

|  |  |  |
| --- | --- | --- |
|  |  | (2.3.2) |

Kolejnym krokiem ku otrzymaniu lepszych rezultatów jest binaryzacja, która została opisana w książce Tadeusiewicza [4]. Binaryzacja to proces konwersji obrazu wejściowego, najczęściej w odcieniach szarości po obróbce wstępnej, do obrazu binarnego o kolorystyce czarno-białej. Odbywa się na zasadzie tzw. progowania. Jeśli więc przyjęta zostanie wartość progu , a jasność punktu w obrazie źródłowym zostanie opisana jako , gdzie to współrzędne jednego piksela obrazu, a , wówczas binaryzacja z dolnym progiem może zostać opisana jako

|  |  |  |
| --- | --- | --- |
|  |  | (2.3.2) |

Nie jest to jedyna

+ OTSU

In global thresholding, we used an arbitrary chosen value as a threshold. In contrast, Otsu's method avoids having to choose a value and determines it automatically.

Consider an image with only two distinct image values (bimodal image), where the histogram would only consist of two peaks. A good threshold would be in the middle of those two values. Similarly, Otsu's method determines an optimal global threshold value from the image histogram.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

## + adaptacja [opencv] Adaptive Thresholding

In the previous section, we used one global value as a threshold. But this might not be good in all cases, e.g. if an image has different lighting conditions in different areas. In that case, adaptive thresholding can help. Here, the algorithm determines the threshold for a pixel based on a small region around it. So we get different thresholds for different regions of the same image which gives better results for images with varying illumination.

Wzory z materiałów [9] i [4]

Erozja to pewien rodzaj filtru minimalnego. Jeśli przyjmiemy, że jasność piksela o współrzędnych jest określona jako , a element centralny z punktem centralnym w miejscu w miejscu piksela jest oznaczony jako , wówczas erozja może być opisana jako

|  |  |  |
| --- | --- | --- |
|  |  | (2.3.5) |

Dylatacja to przekształcenie, które może być traktowane jako filtr maksymalny. Jest ona więc poniekąd przeciwieństwem erozji. Jeśli przyjmiemy, że jasność piksela o współrzędnych jest określona jako , a element centralny z punktem centralnym w miejscu w miejscu piksela jest oznaczony jako , wówczas dylatacja może być opisana jako:

|  |  |  |
| --- | --- | --- |
|  |  | (2.3.6) |

Otwarcie to przekształcenie, które powstaje poprzez kolejne wykorzystanie erozji oraz dylatacji. Zamknięcie natomiast, to przekształcenie, które powstaje poprzez kolejne wykorzystanie dylatacji oraz erozji.

Oprócz operacji przetwarzania obrazu, do polepszenia jakości rezultatów działania OCR, została wykorzystana odpowiednia konfiguracja.

Po otrzymaniu wyników w formie tekstu, było możliwe przeprowadzenie przetwarzania końcowego, w celu wykrycia interesujących dla użytkownika informacji.

Canny, rozmycie gaussowskie, usunięcie szumu, operacje morfologiczne + gradient

# Rozwiązania dostępne na rynku

Jak już zostało wspomniane w rozdziale 1, na rynku dostępnych jest wiele aplikacji mobilnych, które pomagają nie tylko w zarządzaniu wydatkami, ale także w znaczny sposób automatyzują proces dokonywania zakupów i dzielenia wydatków. Utrwalają również paragony i faktury w wersji elektronicznej, co zapobiega utracie danych w wyniku ich degradacji. Aplikacje te, mimo wielu cech wspólnych, często reprezentując różne podejścia do rozwiązywania danych problemów. Z punktu widzenia użytkownika, taka różnorodność jest bardzo korzystna, gdyż umożliwia dobranie najbardziej odpowiedniej aplikacji pod potrzeby danej osoby. Celem każdego twórcy jest natomiast dotarcie do jak największej grupy odbiorców, co wiąże się z tym, iż wprowadzane funkcjonalności prawdopodobnie nie zadowolą wszystkich użytkowników, a będą nakierowane jedynie na usatysfakcjonowanie najbardziej znaczącej grupy. Naturalne wydaje się więc, iż odczucia na temat zalet i wad poszczególnych aplikacje mogą znacznie się od siebie różnić. Stając przed koniecznością wyboru, zdecydowałam się zapoznać z artykułem Michała Wachowskiego [1], w którym zawarto przegląd najpopularniejszych rozwiązań według liczby pobrań w sklepie Google Play. Na jego podstawie oraz po przeglądzie stron przedstawionych aplikacji, postaram się w sposób obiektywny ocenić ich pozytywne i negatywne aspekty.

## PanParagon

Pan Paragon [3] to aplikacja, która na rynku istnieje już od ponad 5 lat i została pobrana ponad 700 tys. razy, jak podano na stronie wydawcy. Na stronie zostało określone, iż za rozwój aplikacji odpowiada firma Primesoft Polska, która może poszczycić się uzyskaniem licznych certyfikatów bezpieczeństwa. Na tej podstawie zasadne wydaje się stwierdzenie, iż ta aplikacja przyciągnie w szczególności użytkowników ceniących sobie w głównej mierze bezpieczeństwo swoich danych. Czynnikiem zachęcającym może być w tym przypadku fakt, iż do korzystania z aplikacji PanParagon nie ma potrzeby tworzenia konta. Zapewniona jest więc pełna prywatność w zamian, za ograniczenie takich funkcjonalności, jak chociażby tworzenie wspólnych grup z innymi.

Wachowski w swoim artykule wspomina fakt, iż aplikacja ma możliwość informowania o produktach oznaczonych przez Główny Inspektorat Sanitarny jako niebezpieczne. Powiadomienie przychodzi wówczas, gdy takie produkty znajdują się na paragonie wprowadzonym przez użytkownika. Autor zwraca uwagę, iż wiele osób nie sprawdza regularnie takich ostrzeżeń, co może stanowić potencjalne zagrożenie. Ponownie, osoby ceniące sobie bezpieczeństwo, z pewnością zwrócą na to uwagę.

Oprócz możliwości przechowywania paragonów, Pan Paragon wykorzystuje algorytmy umożliwiające odczytanie danych o kwocie, sklepie i dacie zakupu wprost ze zdjęcia lub pliku pdf. Jakość odczytu można określić jako zadawalającą, jednak zdarza się, iż aplikacja nie poradzi sobie z odczytem. Wówczas, użytkownik ma możliwość edycji odczytanych danych.

PanParagon, jak wspomina Wachowski, umożliwia również przechowywanie kart lojalnościowych oraz przeglądanie gazetek promocyjnych i kuponów promocyjnych wybranych sklepów. Informuje także o tym, że upływa okres gwarancji lub termin zwrotu danych produktów. We wspomnianym wcześniej artykule, wyszczególniona została także możliwość wykupienia ubezpieczenia na sprzęt elektroniczny wprost przez aplikację. Jest to z pewnością ciekawe rozwiązanie, do którego autor jednak podchodzi z pewną dozą sceptyzmu. Zwraca on jednak uwagę, iż niektórzy mogą docenić szybkość i prostotę dopełnienia formalności.

Do paragonów zostają przypisane kategoria, które mogą zostać wybrane spośród przygotowanych propozycji lub dodane przez użytkownika. Mają one na celu usprawnić tworzenie statystyk i pomóc w rozpoznaniu paragonu. Aplikacja umożliwia także oznaczanie niektórych paragonów jako wyróżnione. Zabieg ten wykorzystywany jest następnie przy filtrowaniu, co znacznie przyspiesza wyszukiwanie konkretnego paragonu. Przewidziana została również funkcjonalność obsługi e-paragonów, które jednak jak dotąd nie są jeszcze bardzo powszechne.

Statystyki i zestawienia wydatków są generowane na podstawie odczytu cen z wcześniej dodanych paragonów. Wartości podzielone są według kategorii oraz miesięcy, co umożliwia użytkownikowi przeprowadzenie analizy i podjęcie działań w celu ewentualnego ograniczenia ponoszonych kosztów.

Pan Paragon wspiera także tworzenie list zakupowych, które mogą, lecz nie muszą, zostać automatycznie uzupełnione produktami na podstawie przepisu z wybranych stron internetowych. Listy takie mogą zostać udostępnione innym użytkownikom z tym, iż aby wykonać tą operację, należy już utworzyć konto. Oprócz tego, listy oferują wszelkie podstawowe funkcjonalności, takie jak chociażby sortowanie, usuwanie i dodawanie produktów, a także możliwość ich odznaczania po dokonaniu zakupu.

## MintCloud

MintCloud [10] jest aplikacją, która swoją premierę miała w 2014 roku, a jej wydawcą jest firma Profit Network LLC. Wachowski w swoim artykule określił ją jako rozwiązanie o większych możliwościach w porównaniu do swojego poprzednika. MintCloud skłania się w kierunku klasycznego podejścia opartego o tworzenie konta, aby zapewnić swoim użytkownikom chociażby możliwość kooperacji przy nadzorowaniu budżetu domowego.

Zaimplementowane algorytmy umożliwiają skanowanie paragonu, jednak w przeciwieństwie do aplikacji PanParagon, z paragonu odczytywana jest jedynie kwota łączna. Skutkuje to zmniejszeniem stopnia automatyzacji zadań, a większość istotnych informacji, takich jak sklep zakupu, data czy też kategoria muszą być wprowadzone ręcznie. Statystki tworzone są główni na podstawie wprowadzonej sumy. W razie zaistnienia potrzeby, do wyszukania konkretnego paragonu może przysłużyć się zawarty w aplikacji filtr.

Dodatkowo należy zaznaczyć, iż MintCloud nie oferuje gotowych kategorii, a użytkownicy muszą utworzyć je sami. Obie wymienione kwestie mogą być powodem niezadowolenia użytkowników, co prawdopodobnie przekłada się na mniejszą popularność aplikacji w stosunku do PanParagon.

Wachowski wspomina także o ciekawym rozwiązaniu, jakim jest możliwość tworzenia wspólnego budżetu obejmującego przychody oraz wydatki. Istnieje również możliwość ustalenia celi oszczędnościowych, co z pewnością działa motywująco na uczestników takiego budżetu. Takie podejście zapewnia o wiele lepszą kontrolę wydatków, niż w przypadku, gdy każdy użytkownik zbiera dane osobno.

MintCloud oferuje również możliwość przechowywania kart lojalnościowych oraz przeglądania gazetek zakupowych i promocji wybranych sklepów. Podobnie jak PanParagon, zapewnia funkcjonalność tworzenia list zakupowych, udostępniania ich oraz odznaczania kupionych produktów. Występuje również opcja oznaczania danego paragonu jako gwarancyjny.

Oprócz tego, Wachowski wspomina, iż aplikacja wprowadziła możliwość skorzystania z programów lojalnościowych swoich partnerów, które pozwalają na zdobywanie punktów za zeskanowanie niektórych paragonów. Zdobyte punkty następnie można wymienić na nagrody, co z pewnością jest ciekawą inicjatywą podjętą przez twórców aplikacji.

## Billy

Billy [11], w porównaniu do wcześniej opisywanych rozwiązań dostępnych na rynku, to aplikacja, która odnotowała jak dotąd jedynie 10 tysięcy pobrań. Oprogramowanie pojawiło się na rynku w 2020 roku. Podobnie jak wcześniej wspomniane rozwiązania, przechowuje paragony, a także odczytuje z nich informacje na temat łącznej sumy za zakupy, ilości zakupionych produktów, a także ich ceny i naliczone rabaty.

To, co wyróżnia aplikację Billy od innych to fakt, iż oprócz odczytania samej kwoty z paragonu, odczytuje także produkty, które się na nim znajdują. Według autora, ta funkcjonalność może przyczynić się do powstania bardziej szczegółowych statystyk.

Zaimplementowano również automatyczną kategoryzację wydatków – niestety, nie ma opcji dodania kategorii własnych. Billy nie obsługuje paragonów elektronicznych – zapewnia jednak możliwość wprowadzenia informacji ręcznie. Dodatkowo, istnieje możliwość wprowadzenia opłat powtarzalnych okresowo.

Podobnie jak MintCloud, aplikacja umożliwia stworzenie wspólnego budżetu z innymi użytkownikami, w celu usprawnienia prowadzenia wspólnej bazy wydatków gospodarstwa domowego. Statystyki są łatwo dostępne i zobrazowane w postaci wykresów na podstawie wartości odczytanych z wprowadzonych paragonów.

Billy umożliwia także archiwizację paragonów, które nie są już dla nas istotne. W tym przypadku nie zapewniono jednak możliwości przeglądania gazetek, wprowadzania kart lojalnościowych czy też prowadzenia własnych list zakupowych. Ciekawą funkcjonalnością, wprowadzoną przez aplikację, jest też możliwość odgórnego ustalenia limitu wydatków. Oprócz tego Billy został wyposażony w umiejętność tworzenia prognoz oszczędności, które mają wesprzeć potencjalnego użytkownika w oszczędzaniu.

## Wybór istotnych funkcjonalności

Aplikacje wymienione w poprzednich podrozdziałach mają w założeniu identyczny cel, jakim jest wsparcie użytkownika w zarządzaniu budżetem domowym. Każda z nich oferuje własne i unikalne podejście do realizacji założonych funkcjonalności. Jednocześnie, aby wyróżnić się na tle innych, aplikacje starają się dodać dodatkowe opcje, aby zainteresować potencjalnego użytkownika.

Przygotowując koncepcję własnej aplikacji, starałam się wybrać te funkcjonalności, które z mojej perspektywy są najbardziej istotne. Jednocześnie, kierując się ideą przyciągnięcia nowych użytkowników poprzez wzbudzenie ich zainteresowania, zdecydowałam się również na dodanie kilku unikalnych opcji, które dotąd nie zostały jeszcze wykorzystane w podobnych aplikacjach. Ostatecznie, do realizacji zostały przyjęte następujące funkcjonalności:

* Odczytywanie informacji na temat sumy łącznej, sklepu, daty oraz produktów znajdujących się na paragonie
* Odczytywanie informacji z dokumentów typu .pdf
* Tworzenie wykresów i statystyk na podstawie informacji odczytanych z paragonów
* Możliwość zarządzania czasem gwarancji i zwrotu produktów oraz wysyłanie przypomnień, gdy zbliża się ku końcowi
* Możliwość wprowadzania wydatków powtarzalnych okresowo
* Tworzenie wspólnego budżetu i zarządzanie nim
* Tworzenie kont użytkowników
* Tworzenie list zakupowych ręcznie lub na podstawie przepisu i zarządzanie nimi
* Udostępnianie list zakupowych i możliwość odznaczania kupionych produktów
* Przeglądanie aktualnych promocji w wybranych sklepach
* Możliwość tworzenia własnych kategorii i przyporządkowywania produktów do danej kategorii
* Filtrowanie wyników i możliwość oznaczania najważniejszych paragonów

Wskazane funkcjonalności są w mojej opinii kluczowe do zrealizowania aplikacji, która ma wspierać zarządzanie paragonami. Modyfikacją, która została wprowadzona, jest zamiana budżetu domowego na budżet grupy, co umożliwi kontrolę wydatków również podczas wyjazdów z grupą innych użytkowników.

# Narzędzia programistyczne wykorzystywane w aplikacji

**Python** [12] to interpretowany, wysokopoziomowy język programowania, który został stworzony w roku 1991 przez Guido van Rossum. Do utworzenia aplikacji wykorzystano wersję Python 3.11, a także biblioteki:

* Django [13] – jest to framework webowy języka Python, służący do tworzenia backendu stron internetowych oraz aplikacji mobilnych w rozwiązaniach hybrydowych. W głównej mierze umożliwił tworzenie modeli bazy danych oraz relacji między nimi, a także całej struktury strony internetowej.
* Pandas [14] – biblioteka służąca do analizy danych, która została zastosowana do utworzenia ramek danych. Ramki te następnie wykorzystano do kalkulacji, a następnie na ich podstawie utworzono wykresy w plotly.
* BeautifulSoup [15] – to biblioteka, która została wykorzystana do webscrapingu, czyli odczytu danych z kodu źródłowego stron internetowych. Została użyta w sekcji list do tworzenia listy zakupowej na bazie przepisu internetowego oraz przy odczytywaniu aktualnych promocji danego sklepu w sekcji promocji i obniżek.
* OpenCV [16] – biblioteka zawierające funkcje służące do przetwarzania i analizy obrazów cyfrowych. Została wykorzystana w przetwarzaniu wstępnym do przeprowadzenia modyfikacji na obrazie wejściowym w celu poprawienia wyników OCR.
* Plotly [17] – jest biblioteką umożliwiającą tworzenie wykresów na podstawie dostarczonych danych. Została wykorzystana przy tworzeniu interaktywnych wykresów znajdujących się w sekcji statystyk, sekcji grup oraz w głównym panelu.
* PyTesseract [6] – jest to tzw. *wrapper* silnika OCR Tesseract, który umożliwia wykorzystanie go w programie języka Python.
* pdf2img [18] – biblioteka zajmująca się konwersją pliku PDF do obrazu w formacie JPG. Wykorzystano ją w przetwarzaniu wstępnym, aby zamienić plik wejściowy z formatu PDF na format obrazu JPG.
* numpy [19] – jest to biblioteka służąca do obliczeń numerycznych, przeprowadzanych w głównej mierze na macierzach.
* imutils [20] – jest biblioteką służącą do wykonywania różnych transformacji na obrazach. Została wykorzystana do skalowania obrazu.
* deskew [21] – biblioteka służąca do przeprowadzania rotacji i detekcji kąta obrotu. Została wykorzystana do wykrycia kąta przekrzywienia obrazu po wstępnym przeprowadzeniu szacowania orientacji. Działa w oparciu o detekcję Canny i transformatę Hougha.

**Javascript** to skryptowy [21] język programowania, który powstał w 1995 roku. Za jego twórcę uznawany jest Brendan Eich. Język ten znajduje zastosowanie między innymi przy tworzeniu interaktywnych stron internetowych. Zapewnia on szereg różnych funkcji, które mają na celu przeprowadzić pewne działanie w odpowiedzi na dane zdarzenie.

W mojej aplikacji, Javascript odpowiada w głównej mierze za efekty wizualne, które mają pokazać lub ukryć dane pola w odpowiedzi na zdarzenie wywołane przez użytkownika. Wspomaga tworzenie logiki występującej na stronie, która ze względu na strukturę Django, nie mogła zostać zaimplementowana w języku Python.

**HTML** (*ang.* HyperText Markup Language) to język znaczników [22], który jest stosowany w głównej mierze do tworzenia głównej struktury strony internetowej. Został stworzony w 1993 roku przez Tima Berners-Lee.

Do stworzenia aplikacji została wykorzystana najnowsza wersja języka – HTML5. Został on wykorzystany do stworzenia szablonu (*ang. template*)każdej strony, który następnie był wykorzystywany w strukturze funkcji w języku Python.

**CSS** (*ang.* Cascading Style Sheets) to język [23], który służy do określenia stylu strony internetowej. Ściśle współpracuje z językiem znaczników HTML i umożliwia dostosowanie elementów do potrzeb programisty. Obejmuje to między innymi zmianę koloru czcionki i tła, wielkości liter czy rozmiaru elementu. CSS powstał w 1996 roku i został przygotowany przez organizację W3C.

**Bootstrap** [24] natomiast, to jedna z bibliotek arkusza stylów CSS, która ułatwia tworzenie stylu stron internetowych. Pierwsza wersja biblioteki została wydana 2011 roku przez programistów Twittera i jest rozwijana do dzisiaj.

W aplikacji, CSS został wykorzystany do nadania stylu poszczególnym elementom strony, a także do ich odpowiedniego umiejscowienia. Bootstrap natomiast dostarczył niezbędne ikony, wprowadził styl dla pasków ładowania, przycisków i formularzy.

**Kotlin** [25][26] jest międzyplatformowym, wysokopoziomowym językiem programowania, który powstał w 2011 roku. Za jego stworzenie odpowiedzialne jest przedsiębiorstwo JetBrains. Jego głównym przeznaczeniem jest rozwój aplikacji natywnych dla urządzeń z systemem Android.

W moim przypadku, Kotlin jest wykorzystywany do stworzenia aplikacji mobilnej, która może być uruchomiona na urządzeniu z systemem Android, które posiada dostęp do Internetu. Jest to realizowane poprzez element języka Kotlin – WebView - który umożliwia wyświetlenie strony internetowej wewnątrz aplikacji.

**PostgreSQL** to system zarządzania relacyjnymi bazami danych, który powstał w 1995 roku [27]. Umożliwia tworzenie baz danych oraz relacji, które występują pomiędzy tabelami. W aplikacji, postgreSQL wykorzystywany jest do tworzenia bazy danych obiektów oraz relacji między nimi. W tym celu, silnie współpracuje on z Django i kodem napisanym w języku Python. Oprócz tego, umożliwia szybką edycję rekordów z bazy danych, przy pomocy zapytań języka SQL, a także zapewnia możliwość zarządzania i nadzorowania zawartości bazy.

# Implementacja

## Baza danych aplikacji

Baza danych początkowo utworzona była z wykorzystaniem SQLite – podstawowej bazy danych, która jest domyślną bazą danych, obsługiwaną i zapewnianą przez Django. W późniejszym etapie, ta została przekształcona na bazę drugorzędną, służącą do generacji wykresów ERD na podstawie pliku w formacie .sqlite3. Aktualna baza danych została stworzona z wykorzystaniem PostgreSQL, co znacznie usprawniło proces zarządzania rekordami w tabelach.

Na Rys.4 przedstawione zostały relacje pomiędzy poszczególnymi modelami. Można zaobserwować, iż najwięcej powiązań posiada tabela odpowiedzialna za przechowywanie użytkowników. Obrazuje to, jak silny wpływ ma użytkownik na kształtowanie pozostałych tabel. Modele posiadające tylko strzałki wchodzące do bloku służą tylko do pobierania z nich danych przez inne modele. Wychodzące świadczą natomiast o pobieraniu informacji z innej tabeli.

Obraz zawierający tekst, diagram, Równolegle, zrzut ekranu

Opis wygenerowany automatycznie

Rys. 5 Struktura bazy danych aplikacji oraz relacje występujące pomiędzy poszczególnymi tabelami

Niektóre tabele bazy danych zostały już na początku wygenerowane zgodnie z normami obowiązującymi w Django [X]. Odpowiadają one za organizację podstawowej struktury i zapewnienie podstawowych funkcjonalności każdej aplikacji webowej. Należą do nich:

* *Auth\_group –* zawiera dane o grupach użytkowników, jednak innych od tych, które służą do wspólnego zarządzania wydatkami (za to odpowiedzialna jest tabela *Commongroups*). Te grupy służą do celów administracyjnych, przykładowo chociażby do nadania przez administratora pewnej grupie użytkowników pewnej rangi lub uprawnień.
* *Auth\_ permission –* posiadadane na temat istniejących w projekcie uprawnień, które mogą być nadawane użytkownikom i grupom
* *Auth\_user –* przechowuje ogólne informacje oużytkownikach, takie jak ich hasło, nazwa oraz status.
* *Auth\_group\_permissions* – zawiera dane na temat powiązań pomiędzy grupami a uprawnieniami.
* *Auth\_user\_groups –* posiada dane dotyczące powiązania użytkowników z grupami.
* *Auth\_user\_user\_permissions –* przechowuje dane o uprawnieniach konkretnych użytkowników.
* *Django\_content\_type* – zawiera dane o każdym uwzględnianym w projekcie i zainstalowanym modelu.
* *Django\_admin\_log –* odpowiada za rejestrowanie logów zmian wprowadzonych w panelu administracyjnym oraz ich wyświetlanie. Dostępne są informacje przykładowo o usunięciu, edycji lub dodaniu rekordu do jednej z tabel.
* *Django\_migrations –* przechowuje dane na temat migracji przeprowadzonych w projekcie.
* *Django\_session –* posiada informacje na temat obecnej sesji użytkowników.

Nie wszystkie możliwości wymienionych tabel zostały w pełni wykorzystane w aplikacji, jednak zostały pozostawione z myślą o dalszym rozwoju i usprawnianiu istniejących funkcjonalności w przyszłości. Najczęściej wykorzystywana była tabela „*Auth\_user*”, natomiast „Auth\_permission” oraz „Auth\_group” i ich powiązania nie były używane.

Ze względu na dużą złożoność bazy danych, na Rys.4 przedstawiono jedynie nazwy poszczególnych tabel oraz relacje między nimi. W celu łatwiejszej analizy, aplikacja została podzielona na mniejsze sekcje, które zostaną omówione bardziej szczegółowo w kolejnych podrozdziałach.

## Profile użytkowników

Aby swobodnie korzystać z aplikacji, użytkownik musi się na początku zarejestrować na odpowiedniej stronie. Wykorzystanie specjalnych dekoratorów zapewnionych przez bibliotekę Django, ograniczono możliwość dostępu do poszczególnych stron użytkownikom niezalogowanym. Operacja ta ma na celu zapewnienie bezpieczeństwa danych użytkownika, przy jednoczesnym zapewnieniu dostępu do treści tworzonych przez innych użytkowników.

Po poprawnym wypełnieniu formularza rejestracji, nowy użytkownik zostaje od razu zalogowany i przekierowany na stronę panelu głównego. Automatycznie tworzona jest również instancja modelu profilu, która zostaje przypisana do nowego użytkownika. Tym sposobem, każda osoba w bazie danych może w pewnym stopniu spersonalizować swoje konto. Do profilu może zostać dodany krótki opis, a ikona go reprezentująca może zostać zmieniona na inną, dostępną w bazie. Użytkownik może zmienić kolor tła ikony, jak i jej samej, co umożliwia utworzenie ogromnej ilości wariantów kolorystycznych.

W profilu zawarta jest także informacja o dacie urodzenia danego użytkownika. Informacja ta jest przede wszystkim wykorzystywana do sprawdzenia wieku osoby, która rejestruje się do aplikacji. Założono, iż osoby poniżej 13 roku życia nie będą miały możliwości dołączenia. Dodatkowo, data urodzenia służy do wyświetlenia wieku na stronie użytkownika i ewentualnej informacji, że użytkownik obchodzi urodziny danego dnia.

Jak można zauważyć na Rys.5, model profilu zawiera także trzy flagi typu *Bool*. Dwie z nich – „*messages\_by\_users*” oraz „*messages\_by\_groups*” wykorzystywane są w sekcji wiadomości. Informują o tym, czy funkcja blokowania wiadomości od grup oraz wybranych użytkowników jest włączona, czy też nie. Flaga „is public” określa natomiast, czy użytkownik chce, aby jego profil był widoczny dla innych, czy też nie.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

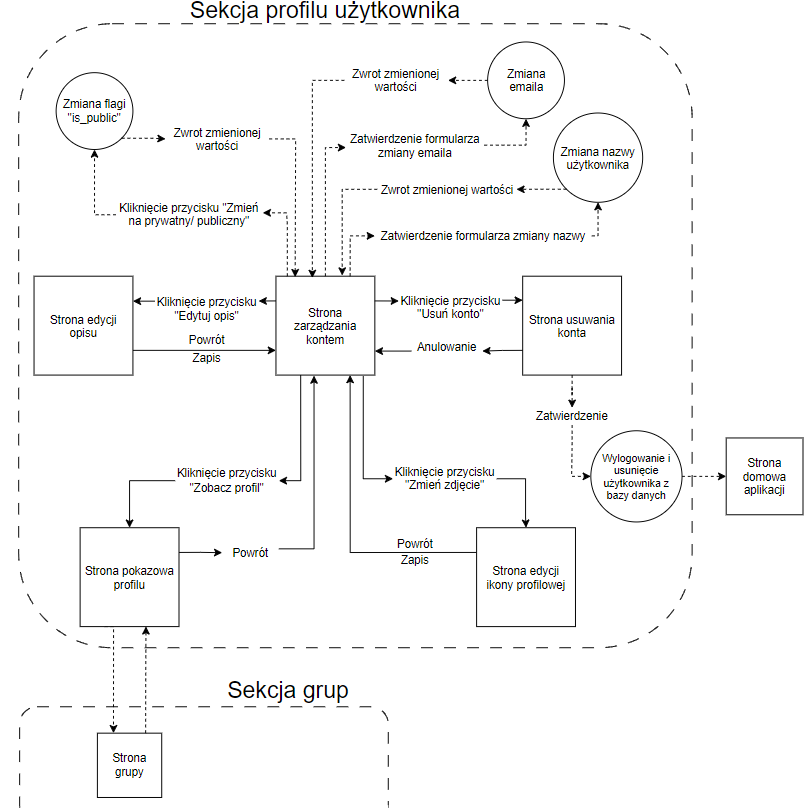
Opis wygenerowany automatycznie

Rys. 6 Fragment bazy danych odpowiedzialny za sekcję profili użytkowników.

Na stronie zarządzania kontem istnieje możliwość edycji opisu, zdjęcia, nazwy użytkownika czy też emaila wprowadzonego przy rejestracji. W tym miejscu zmienić można również wartość flagi „*is\_public*”. Jej aktualny status jest zawsze wyświetlany na stronie zarządzania kontem. Po każdej zmianie, w górnej części ekranu wyświetli się powiadomienie o tym, czy żądanie udało się zrealizować.

Na stronie pokazowej znajdują się informacje o wieku użytkownika, ostatniej dacie logowania oraz dacie dołączenia do aplikacji. Oprócz tego wyświetlana jest ikona profilu, opis oraz ewentualnie oznaczenie, czy dana osoba jest członkiem administracji. Dostęp do tej strony jest możliwy jedynie poprzez stronę zarządzania kontem (tylko dla właściciela profilu) oraz przez stronę grupy (dla jej użytkowników), gdy flaga „*is\_public*” jest ustawiona na wartość *True*.

Na samym dole strony znajduje się przycisk, umożliwiający usunięcie konta użytkownika. Po jego naciśnięciu, następuje przekierowanie na stronę z ostrzeżeniem o utracie danych. Tam użytkownik może zrezygnować lub kontynuować. Druga opcja poskutkuje wylogowaniem, a następnie usunięciem użytkownika oraz wszystkich powiązanych z nim obiektów z bazy danych. Na Rys. 6 przedstawiono schematycznie, jak wyglądają procesy mające miejsce w sekcji profilu użytkownika.



Rys. 7 Rysunek poglądowy stron i przejść pomiędzy nimi dla sekcji profilu użytkownika

## Przetwarzanie obrazów i implementacja OCR

Pods

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

Rys. 8 Schemat ideowy działania algorytmu odczytu danych z paragonów (opracowano na podstawie [28])

Szczegółowy opis zaimplementowanego OCR, wszelki jego usprawnienia a także kod.

- schemat działania

- Przyjęte podejście

- Zastosowanie pytesseract

Obraz zawierający tekst, Czcionka, zrzut ekranu, dokument

Opis wygenerowany automatycznie

Rys. 9 Fragment bazy danych odpowiedzialny za sekcję paragonów, wydatków, gwarancji i terminów zwrotu

## System grup i podziału wydatków

W sekcji grup użytkownicy mogą tworzyć swoje własne grupy, które mają za zadanie wspierać. Wydawanie podczas wyjazdów grupowych oraz w gospodarstwach domowych. Ma to na celu zwiększanie możliwości komunikacji pomiędzy członkami grupy, a także lepszego podziału wydatków pomiędzy nich. Grupy zapewniają możliwość dodawania wspólnych list zakupowych, paragonów oraz opłat. Osoba będąca właścicielem grupy ma możliwość zarządzania wszystkim. W prosty i przejrzysty sposób. Jednocześnie system grup zapewnia możliwość łatwego podziału wydatków pomiędzy członków grupy. A także szybkiego rozliczenia się pomiędzy. Członkami grupy a ich właściciele.

Właściciel może decydować o tym, które wydatki zostaną przyjęte do grupy może także zapraszać członków oraz zarządzać propozycjami dołączenia do grupy. Jeżeli wykryje on jakąś niewłaściwość, może on w szybki sposób odczepić dany paragon lub opłatę od grupy. Oprócz tego może on wprowadzić limit wydatków danej grupy. A także wstrzymać możliwość dodawania opłat i wydatków.

Właściciel może odejść z grupy, a wówczas grupa zostanie przekazana użytkownikowi, który znajduje się w niej najdłużej. Lub temu, który został wyznaczony poprzez poprzedniego właściciela.

Na stronie grupy dla każdego użytkownika dostępne są krótkie statystyki, które pokazują Wkład poszczególnych członków grupy, z czego 3 pierwszych zostaje wyszczególnionych na wykresie, a pozostali są określani jako inni użytkownicy. Na wykresie kołowym możemy dodatkowo zobaczyć. Wkład tych 3. Użytkowników. Oraz jaki procent oraz ile złoty zostało zrealizowane z ustalonego wcześniej limitu. Nie został określony wówczas Zostają pokazane tylko wydatki wniesione poprzez użytkowników grupy.

Na stronie głównej grupy mamy również możliwość wyszukania grupy poprzez wprowadzenie kilku liter, które są zawarte w jej nazwie. Im większa liczba liter zgadzające się z nazwą grupy, tym większa szansa na to, że zostanie ona wykryta jako pierwsza.

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Rys. 10 Fragment bazy danych odpowiedzialny za sekcję grup.

Obraz zawierający tekst, diagram, szkic, rysowanie

Opis wygenerowany automatycznie

Rys. 11 Rysunek poglądowy stron i przejść pomiędzy nimi dla sekcji grup

Opis systemu do sprawnego podziału wydatków, zarządzania grupą, wyszukiwania i tworzenia własnych grup.

- dodawanie i usuwanie użytkowników

- tryb spłaty

- możliwość dodawania paragonów

- zarządzanie grupą

- odejście właściciela

- limity

-wyszukiwarka grup

- Statystyki wydatków

- listy zakupowe grup

## Kategorie

Kategorie zostały stworzone w celu usprawnienia procesu tworzenia statystyk na podstawie wprowadzonych przez użytkownika paragonów i opłat. Oprócz tego, znacznie przyspieszają wyszukiwanie istotnych dla użytkownika informacji poprzez filtrację. Ikony i kolory nadane poszczególnym kategoriom umożliwiają proste rozpoznanie opłat i paragonów, które pod nie podlegają.

Użytkownikom została udostępniona możliwość korzystania z kategorii przygotowanych wcześniej, a także takich, które zostały stworzone przez nich. Pierwsze z nich zostały nazwane kategoriami głównymi. Charakteryzują się one tym, że nie posiadają właściciela, a wartość flagi *public* został u nich ustawiona na *True.* Są one więc publiczne i dostępne dla każdego użytkownika. Nie udostępniono ich jednak do edycji – posiadają z góry określone podkategorie oraz produkty, które do nich należą. Kategorie własne są natomiast widoczne jedynie dla ich właściciela, co zostało zagwarantowane przez ustawienie flagi *public* na *False*. Użytkownik, do którego należą, może je dowolnie edytować, dodawać własne podkategorie oraz produkty.

Niezależnie od rodzaju, kategorie działają na identycznej zasadzie – przy tworzeniu nowej opłaty lub nowego paragonu mogą zostać do nich przypisane. Opłaty mogą mieć przypisaną tylko jedną kategorię, natomiast paragony – więcej niż jedną. Odbywa się to poprzez ręczne wskazanie lub, w przypadku wykorzystania odczytu z OCR, poprzez automatyczną klasyfikację na podstawie wykrytego sklepu lub produktów. Schemat fragmentu bazy danych, odpowiedzialnej za sekcję kategorii, przedstawiono na Rys.11.

Obraz zawierający tekst, zrzut ekranu, Czcionka, Równolegle

Opis wygenerowany automatycznie

Rys. 12 Fragment bazy danych odpowiedzialny za sekcję kategorii.

Na Rys. 12 przedstawiono w sposób poglądowy, jak prezentują się przejścia pomiędzy poszczególnymi stronami sekcji.

Obraz zawierający tekst, diagram, Plan, wykres

Opis wygenerowany automatycznie

Rys. 13 Rysunek poglądowy stron i przejść pomiędzy nimi dla sekcji kategorii

## Webscraping stron popularnych sklepów

W trakcie planowania przyszłych zakupów, użytkownicy często zwracają swoją uwagę na promocje aktualnie dostępne w interesujących ich sklepach. Produkty kupione zawczasu na przecenie mogą bowiem odciążyć budżet w przyszłości. Z punktu widzenia klienta istotne jest więc, aby być na bieżący z przecenami. Jako, iż jest to czynnik mający wpływ na proces tworzenia paragonów, aplikacja zapewnia użytkownikowi dostęp do informacji na temat promocji wybranych sklepów dostępnych w bazie danych. Na chwilę obecną jedynie przeceny w sklepie Rossmann są odczytywane i wyświetlane. Fragment bazy danych odpowiedzialny za przechowywanie informacji na temat promocji został uwidoczniony na Rys. 13.

Obraz zawierający tekst, zrzut ekranu, Czcionka, Równolegle

Opis wygenerowany automatycznie

Rys. 14 Fragment bazy danych odpowiedzialny za sekcję promocji

Dane odczytywane są ze strony internetowej danego sklepu przy pomocy funkcji biblioteki Beautiful Soup, służących do odczytywania określonych informacji z kodu źródłowego strony. Odbywa się to na zasadzie wyszukania odpowiednich znaczników HTML ze wskazanym id lub nazwą klasy dla podanego url. Elementy w kodzie źródłowym stron internetowych są różne, a więc dla każdego sklepu istnieje potrzeba ręcznego zlokalizowania istotnych informacji.

W wyniku przeprowadzenia webscrapingu otrzymywany jest ciąg znaków, który oprócz poszukiwanej zawartości posiada także zbędne dla użytkownika informacje, takie jak atrybuty opisujące odczytywany znacznik. Do wyodrębnienia istotnych danych wykorzystana została biblioteka re, służąca do dopasowywania pewnego wzorca do podanego ciągu znaków. Tym sposobem, ze strony internetowej sklepu zostały odczytane informacje odnośnie nazwy danego produktu oraz jego poprzedniej i aktualnej ceny. Na ich podstawie wyliczona została procentowa obniżka, zgodnie ze wzorem (5.6.1).

|  |  |  |
| --- | --- | --- |
|  |  | (5.6.1) |

gdzie

p – procentowa obniżka,

m – stara cena,

n – nowa cena.

Każdemu wynikowi przyporządkowano odpowiedni kolor, mający na celu dodatkowe wskazanie użytkownikowi, które promocje są najbardziej opłacalne.

Do wyświetlenia wyników zostało wykorzystane stronicowanie, które powoduje, iż na jednej stronie wyświetlają się maksymalnie 24 wyniki. Jednorazowy odczyt wszystkich informacji ze wszystkich stron z promocjami danego sklepu okazał się być bardzo czasochłonny. Rozwiązanie to mogłoby posortować wszystkie oferty od najbardziej do najmniej korzystnej, jednak długi czas oczekiwania mógłby zniechęcić użytkownika. Dlatego też, dane są odczytywane i sortowane na każdej stronie dla niewielkiego zbioru ofert. Znacznie zmniejsza to czas oczekiwania na załadowanie strony, a jednocześnie wskazuje lokalnie najbardziej opłacalne oferty. Dodatkowo, użytkownik może zawęzić zakres poszukiwań poprzez zastosowanie filtru cenowego, co może przyspieszyć wyszukanie najbardziej istotnych dla użytkownika ofert.

Na Rys.14 przedstawiono wszystkie strony oraz przejścia między nimi dla sekcji promocji.

Obraz zawierający tekst, diagram, zrzut ekranu, Plan

Opis wygenerowany automatycznie

Rys. 15 Rysunek poglądowy stron i przejść pomiędzy nimi dla sekcji promocji

## System wiadomości

Koncepcja systemu wiadomości, znajdującego się wewnątrz aplikacji, powstała w oparciu o rozwiązania stosowane w niektórych aplikacjach społecznościowych czy grach przeglądarkowych. Zaletą takiego podejścia do problemu jest przede wszystkim możliwość utrzymania wszystkich wiadomości i powiadomień w jednym miejscu. Użytkownik ma dostęp zarówno do wiadomości odebranych, jak i wysłanych. W zależności od rodzaju wiadomości, logika ich wyświetlania jest różna.

Na stronie wiadomości odebranych wyświetlają się wszystkie wiadomości, które spełniają warunek bycia widocznym (*is\_seen\_by\_receiver*), a odbiorcą jest dany użytkownik. Warunki widoczności, widoczne w tabeli na Rys.15, pełnią istotną rolę przy usuwaniu wiadomości, która odbywa się tylko wtedy, gdy obie strony, zarówno nadawca jak i odbiorca, postanowią usunąć wiadomość. Usunięcie jest więc równoznaczne ze zmianą wartości Bool u danej osoby, a nie z fizycznym usunięciem.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rys. 16 Fragment bazy danych odpowiedzialny za sekcję wiadomości.

Flaga *new* przy wiadomości oznacza, czy dana wiadomość została odczytana czy też nie. Występuje tylko przy wiadomościach odebranych i po zajrzeniu do wnętrza wiadomości, zostaje ona odznaczona. Liczba wiadomości nieodczytanych jest zawsze pokazana na pasku bocznym, przy odpowiedniej ikonie koperty. Można ją także zobaczyć w głównym panelu użytkownika.

Flaga *answered* natomiast odpowiada za to, czy na daną wiadomość została już wysłana odpowiedź. Jeśli tak, niektóre funkcjonalności mogą zostać ograniczone dla poszczególnych typów wiadomości (*message\_type*). Przekładowo, jeśli użytkownik otrzymał zaproszenie do grupy, wyświetlone zostają przyciski akceptacji i odmowy, które po naciśnięciu wykonują określone funkcje. Przy naciśnięciu takiego przycisku, flaga *answered* zmienia swą wartość na *True* i tym samym ukrywa widoczność przycisków. Ma to na celu zapobiec kilkukrotnemu wykonywaniu pewnych czynności. Flaga może się również zmienić, gdy użytkownik prześle odpowiedź na wiadomość standardową.

Aktualnie w aplikacji dostępnych jest 7 typów wiadomości:

* *invitation* – typ wiadomości będącej zaproszeniem danego użytkownika do grupy przez jej członka lub właściciela. Wraz z otwarciem wiadomości, tworzone są 2 przyciski do akceptacji bądź odrzucenia propozycji.
* *invitation\_accepted* – typ wiadomości, który zostaje wykorzystany przy przesłaniu informacji zwrotnej do użytkownika lub właściciela grupy o akceptacji zaproszenia.
* *invitation\_rejected* – typ wiadomości, który zostaje wykorzystany przy przesłaniu informacji zwrotnej do użytkownika lub właściciela grupy o odrzuceniu zaproszenia.
* *request* – typ wiadomości będący prośbą danego użytkownika o możliwość dołączenia do danej grupy. Wiadomość wysyłana jest wprost do właściciela grupy i wraz z jej otwarciem, tworzone są 2 przyciski do akceptacji bądź odrzucenia prośby.
* *request\_accepted* – typ wiadomości, który zostaje wykorzystany przy przesłaniu informacji zwrotnej do użytkownika ubiegającego się o przyjęcie do grupy o akceptacji prośby.
* *request\_rejected* – typ wiadomości, który zostaje wykorzystany przy przesłaniu informacji zwrotnej do użytkownika ubiegającego się o przyjęcie do grupy o odrzuceniu prośby.
* *normal* – typ wiadomości podstawowej.

D

Obraz zawierający tekst, diagram, Plan, rysowanie

Opis wygenerowany automatycznie

Rys. 17 Rysunek poglądowy stron i przejść pomiędzy nimi dla sekcji wiadomości

## Gwarancje, terminy zwrotów i opłaty

W sekcji tej znajdują się informacje na temat tych informacji, które nie są zawarte na paragonach, a biorą udział w tworzeniu budżetu domowego. Wyróżniono dwie główne części – opłaty i wydatki oraz terminy gwarancji i zwrotów, które obok paragonów są głównymi elementami aplikacji, wykonywanej w ramach pracy.

Pierwsza z nich odnosi się do wszystkich wydatków, do których z pewnych względów nie można przypisać zdjęcia bądź też pliku PDF. Mogą zostać wprowadzone tylko w sposób manualny przez użytkownika. Podobnie, jak przy paragonach, opłata może zostać przypisana do grupy i podpięta pod określoną kategorię. Istnieje również funkcja umożliwiająca oznaczenie opłaty gwiazdką, w celu szybszego wyszukiwania poprzez filtrowanie. To, co wyróżnia opłaty, to przede wszystkim możliwość dodania powtarzalności co ustalony okres czasu. Użytkownik może wpisać okres powtarzalności co kilka dni, miesięcy lub lat, a w statystykach opłata ta będzie już naliczana automatycznie do czasu usunięcia danej opłaty.

Część odpowiedzialna za terminy gwarancji i zwrotów

Zarówno opłaty, jak i gwarancje i terminy zwrotu, nie są usuwane w sposób fizyczny – gdy użytkownik naciśnie przycisk usuwania, wówczas zmieniana jest jedynie flaga *is\_deleted*, która informuje o widoczności danych dla użytkownika. W bazie obiekt nadal figuruje i jest wykorzystywany do tworzenia statystyk.

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Rys. 18 Fragment bazy danych odpowiedzialny za sekcję opłat, gwarancji i terminy zwrotu

## Statystyki i wykresy

W sekcji statystyk użytkownik ma możliwość zobaczenia wykresów i podsumowań, generowanych na podstawie wcześniej wprowadzonych danych. Informacje wprowadzane są poprzez formularze w sekcji paragonów, opłat, gwarancji i terminów, a następnie są zapisywane w odpowiedniej tabeli bazy danych aplikacji. Dane są przypisane do konkretnego użytkownika, który je wprowadził – umożliwia to przefiltrowanie tabeli w poszukiwaniu obiektów podlegających pod dane konto. W procesie tworzenia wykresów biorą udział modele, których tabele zostały już wcześniej przedstawione na Rys.X oraz Rys.Y.

Stwierdzono, iż przy analizie pod kątem ograniczenia wydatków najbardziej istotne dla użytkownika są:

* Koszty poniesione w ramach opłat powtarzalnych, opłat jednorazowych oraz paragonów w danym okresie czasu,
* Koszty poniesione w ramach danej kategorii w danym okresie czasu,
* Koszty poniesione w danym sklepie w danym okresie czasu,
* Koszty poniesione w ramach wydatków dla danej grupy w danym okresie czasu.
* Koszty poniesione w ramach danego okresu czasu

Utworzone wykresy są interaktywne i umożliwiają użytkownikowi filtrację interesujących go danych. Oprócz wizualizacji, w sekcji znalazło się również podsumowanie dotyczące liczby wszystkich dodanych przez użytkownika obiektów oraz łączną kwotę ze wszystkich opłat i paragonów.

Na Rys.17 przedstawiono, jak wyglądają poszczególne przejścia pomiędzy stronami. Kolejne strony zostały wprowadzone w celu ograniczenia złożoności obliczeniowej przy dużych bazach danych.

Obraz zawierający tekst, linia, diagram, Czcionka

Opis wygenerowany automatycznie

Rys. 19 Rysunek poglądowy stron i przejść pomiędzy nimi dla sekcji profilu użytkownika

## Listy zakupowe

Tworzenie list zakupowych może się wydawać nieoczywistym elementem aplikacji do zarządzania paragonami. Wiele dostępnych na rynku rozwiązań oferuje jednak tą funkcjonalność. Może to być uwarunkowane tym, że jest to jeden z pierwszych etapów całego procesu „tworzenia się” paragonu. Zazwyczaj bowiem dokonanie zakupu danych produktów jest planowane zawczasu.

Aplikacja umożliwia użytkownikom tworzenie takich list na dwa sposoby. Pierwszym z nich jest klasyczne utworzenie pustej listy. Drugi natomiast oferuje możliwość automatycznego wygenerowania takiej listy na podstawie przepisu dostępnego w Internecie. Na ten moment, aplikacja obsługuje przepisy ze strony www.kwestiasmaku.pl oraz www.aniagotuje.pl. Algorytm opiera się o bibliotekę Beautiful Soup oraz webscraping, czyli odczytywanie danych z danej strony internetowej. Informacje na temat poszczególnych produktów i ich ilości są odczytywane, a następnie dodawane do bazy danych i przypisywane do nowoutworzonej listy.

Aplikacja umożliwia także tworzenie własnych, pustych list zakupowych. Niezależnie od typu utworzonej listy, użytkownik może dodawać składniki, edytować nazwy istniejących produktów, a także samej listy, czy też usunąć produkt lub listę. Wyjątek stanowią listy, które zostały udostępnione przez innych użytkowników, a możliwość edycji została zablokowana przez ich właścicieli.

Na Rys.18 przedstawiono fragment bazy danych, który odpowiada za logikę sekcji list zakupowych i ukazuje relacje pomiędzy poszczególnymi tabelami.

Obraz zawierający tekst, zrzut ekranu, Czcionka, diagram

Opis wygenerowany automatycznie

Rys. 20 Fragment bazy danych odpowiedzialny za sekcję list zakupowych

Na głównej stronie sekcji list, użytkownik ma dostęp do wszystkich list zakupowych, których jest realizatorem. Można nim zostać na dwa sposoby – poprzez stworzenie własnej listy lub gdy zostanie ona przekazana. Przy każdej liście wyświetla się ilość produktów zakupionych ze wszystkich produktów znajdujących się na danej liście. Jeśli wszystkie produkty zostaną zakupione, wówczas lista przyjmuje zostaje oznaczona jako ukończona (*is\_completed*), co zostaje odpowiednio wyświetlone na stronie. Wykorzystano tu również stronicowanie, aby wyświetlić z góry ustaloną liczbę list na ekranie i usprawnić ich przeglądanie.

Przechodząc na stronę konkretnej listy, można zobaczyć wszystkie produkty, które zostały do niej przypisane. Po kliknięciu na produkt, zostanie on oznaczony jako kupiony (*is\_bought*) przez użytkownika, który ją oznaczył. Kolejne kliknięcie odznaczy produkt i ponownie zmieni wartość flagi. Listy udostępnione, niezależnie od flagi możliwości edycji, mają możliwość zaznaczania kupionych produktów przez różnych użytkowników. Informacje na temat tego, czy dana lista jest udostępniona, przekazana czy własna, znajduje się na jej stronie. W przypadku tych, które zostały przekazane, właściciel może zobaczyć, kto ma aktualnie dostęp do listy. Przejścia pomiędzy poszczególnymi stronami zostały przedstawione na Rys.19.

Obraz zawierający tekst, diagram, Plan, wykres

Opis wygenerowany automatycznie

Rys. 21 Rysunek poglądowy stron i przejść pomiędzy nimi dla sekcji profilu użytkownika

## Dostosowanie aplikacji do systemu Android

Aplikacja webowa, napisana w Django, dostępna jest dla każdego urządzenia mającego dostęp do Internetu. Stronę można wyszukać poprzez przeglądarkę internetową po wpisaniu odpowiedniego adresu url. Hosting realizowany jest poprzez platformę ngrok [37].

Współcześnie istniejące technologie umożliwiają bardzo zróżnicowane podejście do tworzenia aplikacji mobilnych, jak zostało to chociażby opisane w artykule A. Skalskiej [29]. Mogą być one tworzone bezpośrednio dla konkretnego systemu operacyjnego, przykładowo w języku Kotlin/Java dla systemu Android lub w języku Swift/Objective C dla systemu iOS. Podejście takie nazywane jest tworzeniem aplikacji natywnych, które są dostosowane pod konkretne urządzenia. Jak określiła autorka tekstu, charakteryzuje je wysoka wydajność, szybkość oraz przyjazny użytkownikowi i dobrze dostosowany interfejs użytkownika.

Na rynku są dostępne urządzenia od różnych firm, które oferują własne rozwiązania i często także – różne systemy operacyjne. Pisząc kod aplikacji mobilnej można natknąć się więc na problem dotyczący wyboru odpowiedniego języka programowania. Głównym celem aplikacji jest bowiem trafienie do jak najszerszego grona odbiorców. W przypadku rozwiązania w postaci aplikacji natywnych, pojawia się wówczas potrzeba napisania odrębnych aplikacji, nakierowanych na konkretne systemy operacyjne. Takie podejście nie tylko znacząco wydłużyła czas wdrożenia, ale także zwiększa koszty produkcji. Z tego powodu, coraz więcej aplikacji tworzonych jest w formie hybrydowej. Jest to rozwiązanie, które umożliwia połączenie elementów aplikacji natywnej z elementami aplikacji webowej, jak zostało to wspomniane w artykule. Do głównych zalet takiego podejścia należą przede wszystkim krótki czas wdrożenia na różnych platformach oraz niższe koszty niż w przypadku aplikacji natywnej. Wadą może być jednak wydajność oraz mniejsza elastyczność implementacji.

Aplikacja tworzona w ramach pracy wykorzystuje drugie z wymienionych rozwiązań. Backend aplikacji jest napisany przy pomocy frameworka Pythona Django, natomiast za frontend odpowiedzialne są języki webowe oraz język programowania Kotlin. Na Rys. 20 przedstawiono schemat ideowy części aplikacji, która pełni funkcję opakowania (*ang. wrapper*) dla aplikacji webowej i nadaje jej formę aplikacji mobilnej.

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

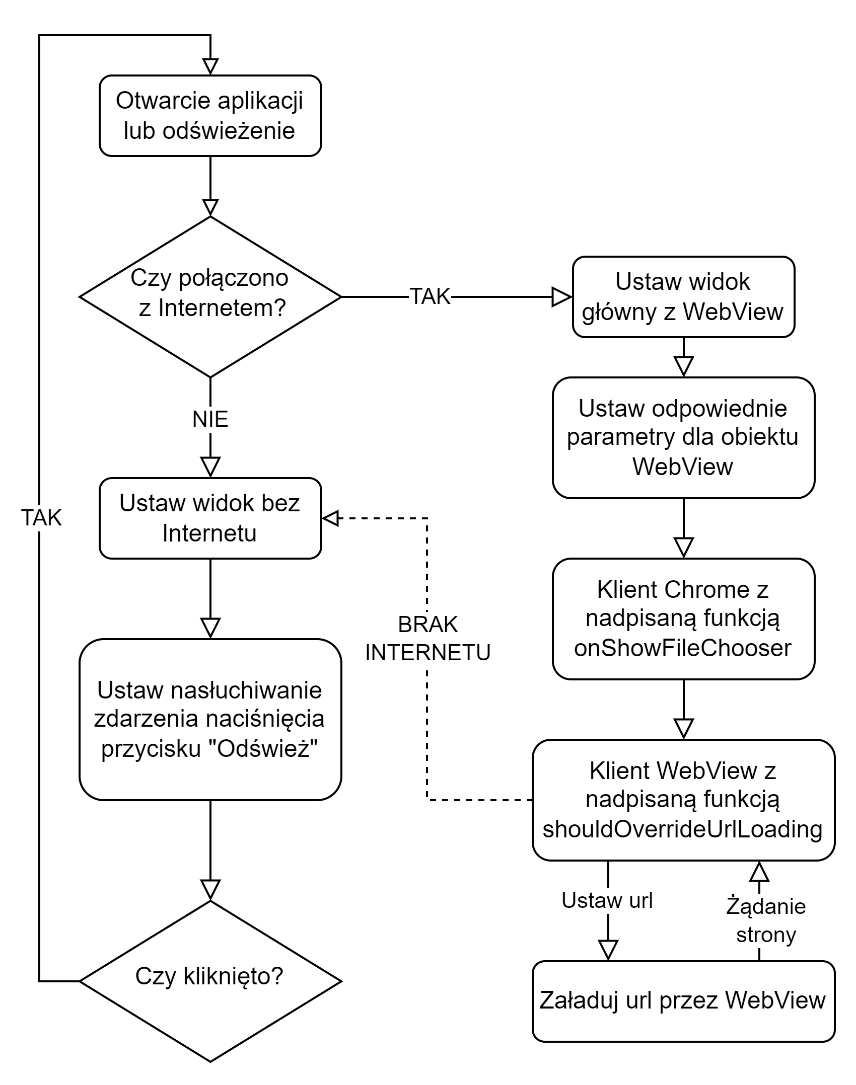
Rys. 22 Schemat ideowy aplikacji mobilnej z wykorzystaniem WebView

Kod źródłowy powstał w oparciu o dokumentację środowiska Android Studio [X] i fragmentów kodu, które zostały tam przedstawione. Aplikacja nie jest złożona, gdyż większość funkcjonalności została zaimplementowana z wykorzystaniem Django. W Kotlinie utworzone zostały dwa widoki w obrębie jednej aktywności.

Pierwszy z nich jest widokiem domyślnym, który zostaje ustawiony w momencie uruchomienia aplikacji. Zawiera on jedynie obiekt WebView, służący do otwarcia danej strony wewnątrz aplikacji mobilnej, bez konieczności otwierania przeglądarki. Strona jest ustawiana automatycznie zgodnie z url, który został zawarty w kodzie. Jeśli podczas korzystania z aplikacji połączenie z Internetem zostanie przerwane, użytkownik zostanie przekierowany na widok drugi. Jest to zachowanie istotne ze względu na to, iż aplikacja nie może działać poprawnie bez dostępu do Internetu. Widok ten zawiera przycisk odświeżania, który po kliknięciu wywołuje odpowiednią funkcję i sprawdza, czy połączenie zostało przywrócone. Jeśli tak, wówczas użytkownik jest przekierowywany do widoku pierwszego, a jeśli nie – pozostaje na widoku aktualnym.

Wykorzystywane w aplikacji funkcje to:

* **onCreate** – funkcja niezbędna przy implementacji, będąca wywoływana tylko raz, gdy system po raz pierwszy tworzy aktywność. Stanowi jeden z elementów tzw. cyklu życia aktywności (*ang. activity lifecycle*), występującego w Android Studio. W aplikacji będącej tematem pracy, zawiera tylko funkcję **refresh**.
* **isConnectedToTheInternet** – funkcja przyjmująca jako argument aktywność i zwracająca wartość typu Bool. Wewnątrz niej znajduje się kod, który tworzy obiekt menadżera połączeń (*ang. ConnectivityManager*). Menadżer ten umożliwia odczytanie informacji o tym, czy w danym obiekcie sieci (*ang. NetworkCapabilities*) wykorzystywany jest transport Wifi albo transport danych komórkowych.
* **refresh** – odpowiada za ustawienie odpowiednich widoków przy uprzednim wykryciu, czy połączenie z Internetem zostało wykryte. Pseudokod funkcji został pokazany na Rys.21.



Rys. 23 Pseudokod funkcji refresh

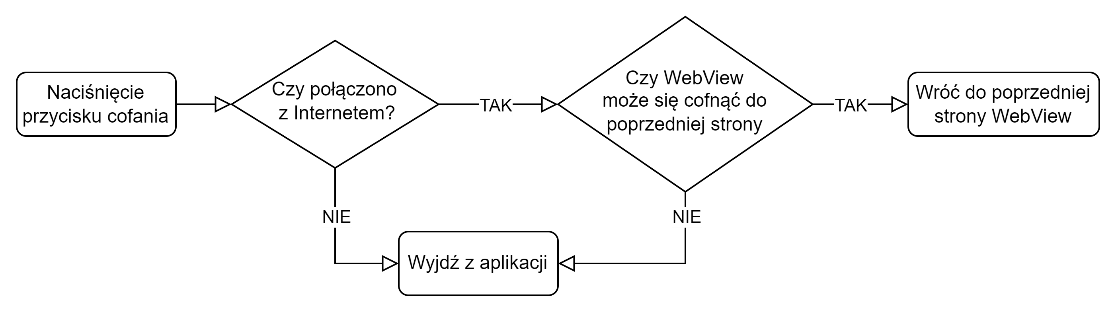
* **shouldOverrideUrlLoading** – funkcja języka Kotlin, która została nadpisana. Odpowiada za prawidłowe ładowanie się strony internetowej przez WebView, gdy Internet jest dostępny, a w przeciwnym razie wywołuje funkcję **refresh**, której sama jest częścią. Pseudokod funkcji został pokazany na Rys.22.

Obraz zawierający tekst, zrzut ekranu, Czcionka, Grafika

Opis wygenerowany automatycznie

Rys. 24 Pseudokod funkcji shouldOverrideUrlLoading

* **onShowFileChooser** i **onActivityResult** – to dwie funkcje języka Kotlin, które zostały nadpisane w celu naprawy błędu w odczycie plików, występującego przy próbie dodania pliku obrazu do bazy danych poprzez WebView w aplikacji. Polegał on na tym, iż obraz był wybierany, lecz dane nie były odczytywane. Pierwsza z funkcji znajduje się wewnątrz funkcji **refresh** i nadpisywana zostaje w obiekcie klienta chrome (*ang.WebChromeClient*). Jej zadaniem jest wykrywanie zdarzenia wgrania pliku oraz utworzenie intencji obiektu parametrów wyboru plików WebView (*ang. fileChooserParams*). Druga funkcja jest wywoływana pod koniec pierwszej i przeprowadza walidację wprowadzonych danych, które przypisywane są do intencji. Stamtąd, po przesłaniu formularza, przekazywane są do bazy danych.
* **onBackPressed** – nadpisanie funkcji dostępnej w języku Kotlin, która obsługuje zdarzenie naciśnięcia przycisku cofania na telefonie. W aplikacji została ona zmodyfikowana poprzez wprowadzenie logiki sprawdzania połączenia z Internetem i odpowiednie reakcji. Pseudokod funkcji przedstawiono na Rys.23.



Rys. 25 Pseudokod funkcji onBackPressed

# Testowanie aplikacji

## Biblioteka danych testowych

Testy zostały przeprowadzone w oparciu o własną bazę danych, zawierającą około 150 zdjęć i zeskanowanych paragonów oraz pliki pdf z potwierdzeniem płatności w języku polskim. Baza danych testowych obejmuje w głównej mierze paragony z popularnych sieci sklepów, takich jak chociażby Biedronka, Żabka, Kaufland, Carrefour czy Rossmann. Jakość paragonów jest zróżnicowana – zostały zgromadzone paragony o różnym stopniu zniszczenia oraz zastosowano zmienne warunki oświetlenia. Oprócz tego, niektóre z paragonów mają niestandardowy kolor tła lub format, zmienioną orientację czy też, zostały zeskanowane przy pomocy skanera.

## Miary stosowane do oceny wyników OCR

W fazie testowania oprogramowania niezbędne jest przeprowadzenie testów, mających na celu określenie skuteczności i ocenę działania wykorzystanego algorytmu OCR. Otrzymane wyniki są w stanie zobrazować, czy działa on poprawnie, czy też posiada błędy, na które należałoby zwrócić uwagę. Do oceny najczęściej stosowane są miary, które opisał w swoim skrypcie Wojciech Jaworski [31]. Są one powszechnie wykorzystywane przy zadaniach klasyfikacji i umożliwiają ocenę jakości danego algorytmu lub modelu w sposób miarodajny i obiektywny.

Jaworski porusza również tematykę powszechnie przyjmowanych konwencji. Pojawia się tam stwierdzenie, iż w OCR do oceny jakości wykorzystywane są znaki tekstu, traktowane jako odrębne klasy. To podejście znaleźć można chociażby przeglądając dokumentację [7] wykorzystywanego w aplikacji silnika OCR. Wielokrotnie pojawiają się tam dwie metryki – WER (*ang.* *Word Error Rate*) i CER (*ang.* *Character Error Rate*). Zostały one opisane między innymi w tegorocznym artykule [32] na portalu callzen.ai, w kontekście algorytmu ASR (*ang Automatic Sspeech Recognition*). Metryki te mogą być jednak wykorzystywane uniwersalnie, co pozwala na użycie ich przy ocenie jakości OCR.

CER to wskaźnik wyliczany na podstawie tego, ile minimalnie transformacji danego, wynikowego ciągu znaków musiałoby zostać wykonane, aby otrzymać ciąg pierwotny. Przedstawia się jako procentowy stosunek minimalnej liczby transformacji do sumy tych transformacji i poprawnie wykrytych znaków. Zależność ta została zobrazowana równaniem

|  |  |  |
| --- | --- | --- |
|  |  | (6.2.1) |

gdzie:

T – minimalna liczba transformacji tekstu

C – liczba poprawnie wykrytych znaków w tekście

WER to wskaźnik podobny do CER, jednak działający na poziomie słowa. Wyliczany jest na podstawie tego, ile minimalnie transformacji elementów danego, wynikowego zdania musiałoby zostać wykonane, aby otrzymać pierwotny tekst. Zależność ta została zobrazowana równaniem

|  |  |  |
| --- | --- | --- |
|  |  | (6.2.2) |

gdzie:

T – minimalna liczba transformacji tekstu

C – liczba poprawnie wykrytych znaków w tekście

W – liczba poprawnie wykrytych wyrazów w tekście

Dla wskaźników WER oraz CER wzory mogą również zostać dokładniej przedstawione, wstawiając w miejsce ogólnych transformacji T wartości S (liczby substytucji), D (delecji) oraz I (insercji). Zarówno WER jak i CER w kontekście badania jakości odczytu, z punktu widzenia aplikacji, mogłyby nie dostarczyć miarodajnych wyników. Algorytm mógłby bowiem wykryć istotne informacje, a tym samym osiągnąć bardzo wysokie wskazania WER i CER. Oba wskaźniki dostarczyłyby jedynie danych na temat działania OCR, a nie jest to tematem pracy. Dlatego zarówno WER i CER zostały przeze mnie pominięte w dalszych podrozdziałach.

Jeżeli jednak algorytm zostanie potraktowany jako klasyfikacja, wówczas do oceny mogą zostać wykorzystane wzory opisywane przez W. Jaworskiego w skrypcie odnośnie miar jakości. Opisane w nim metryki dostarczają kluczowych informacji na temat algorytmu umożliwiają dokonanie obiektywnej oceny. Do najważniejszych miar należą dokładność, precyzja, czułość, swoistość oraz wynik F1. Dla tych wzorów przyjęto następujące oznaczenia:

* TP – wykrycie prawdziwie pozytywne (*ang. True Positive*), oznaczające wykrycie słowa, które występowało w tekście,
* FP – wykrycie fałszywie pozytywne (*ang. False Positive*), czyli wykrycie słowa, gdy w rzeczywistości nie występuje ono w tekście,
* TN – wykrycie prawdziwie negatywne (*ang. True Negative*), czyli brak wykrycia słowa, gdy nie występowało ono w tekście,
* FN – wykrycie fałszywie negatywne (*ang. False Negative*), czyli brak wykrycia słowa, gdy występowało ono w tekście.

Dokładność (*ang. accuracy*) to miara określająca stosunek prawdziwych detekcji pozytywnych i negatywnych do wszystkich detekcji. Przekłada się to na to, ile detekcji słów w danym tekście zostało przeprowadzonych prawidłowo. Opisana została w postaci wzoru, jako

|  |  |  |
| --- | --- | --- |
|  |  | (6.2.3) |

Precyzja (*ang. precision*) określa liczbę prawdziwie pozytywnych detekcji słów w stosunku do wszystkich detekcji pozytywnych. Przekłada się to na ilość słów wykrytych, które faktycznie były obecne w tekście. Zależność pokazuje równanie

|  |  |  |
| --- | --- | --- |
|  |  | (6.2.4) |

Czułość (*ang. recall*) określa liczbę prawdziwie pozytywnych wykryć słów w stosunku do wszystkich słów, które powinny zostać wykryte pozytywnie, czyli do sumy znaków, które słusznie zostały wykryte oraz tych, które błędnie zostały niewykryte. Zależność ta została opisana równaniem

|  |  |  |
| --- | --- | --- |
|  |  | (6.2.5) |

Swoistość (*ang. Specificity*) to miara, która określa, ile słów niewystępujących w tekście, zostało prawidło oznaczonych jako niewystępujące, w stosunku do wszystkich niewystępujących w rzeczywistości słów. Opisuje ją równanie

|  |  |  |
| --- | --- | --- |
|  |  | (6.2.6) |

Wynik F1 (*ang. F1 score*) to średnia harmoniczna precyzji oraz czułości, która określa wydajność przeprowadzonych testów, łącząc właściwości obu cech. Zależność została zobrazowana równaniem:

|  |  |  |
| --- | --- | --- |
|  |  | (6.2.7) |

## Testowanie zaimplementowanego algorytmu OCR

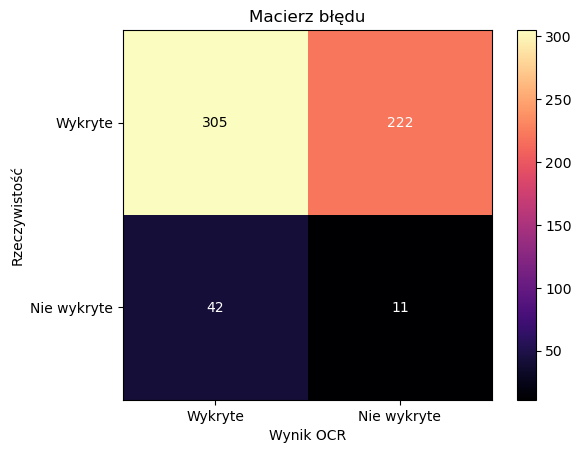
Testy funkcjonalności OCR zostały przeprowadzone jedynie na paragonach w języku polskim, gdyż aplikacja jest do niego przystosowana. Wyniki zaprezentowane przez algorytm porównano z danymi odczytanymi przez człowieka i zanotowanymi ręcznie w arkuszu kalkulacyjnym. Testowanie było skoncentrowane w głównej mierze na sprawdzeniu skuteczności zaimplementowanego przetwarzania wstępnego. W tym celu, przeprowadzono analizę wyników detekcji wybranych informacji dla obrazów oryginalnych bez uprzedniego przetwarzania wstępnego oraz tych samych obrazów, na których najpierw przeprowadzono przetwarzanie wstępne. W obu przypadkach wykorzystano te same ustawienia dla silnika OCR oraz to samo przetwarzanie końcowe. Do oceny otrzymanych wyników zostały wykorzystane miary z podrozdziału 6.2, z wyjątkiem miar CER (6.2.1) oraz WER (6.2.2).

Sprawdzanie rozpoczęto od określenia jakości odczytu i klasyfikacji bez uprzednio przeprowadzonego przetwarzania wstępnego. Testy zostały przeprowadzone na bazie 145 polskich paragonów, a ilość wykrytych oraz niewykrytych informacji na paragonach dla OCR zestawiono w Tab.2.

Tab. 2 Zestawienie ilości informacji wykrytych i niewykrytych na wszystkich paragonach bez uprzedniego przetwarzania wstępnego

|  |  |  |  |
| --- | --- | --- | --- |
| Szukane informacje | Wykryto | Nie wykryto | Średnia |
| Suma | 61 | 84 | 42,1% |
| Data | 75 | 70 | 51,7% |
| Sklep | 94 | 51 | 64,8% |
| Kategoria | 86 | 59 | 59,3% |

Na Rys.24 uwidoczniono natomiast, jak algorytm poradził sobie z detekcją i klasyfikacją istotnych informacji.



Rys. 26 Macierz błędu detekcji istotnych informacji dla OCR bez przetwarzania wstępnego

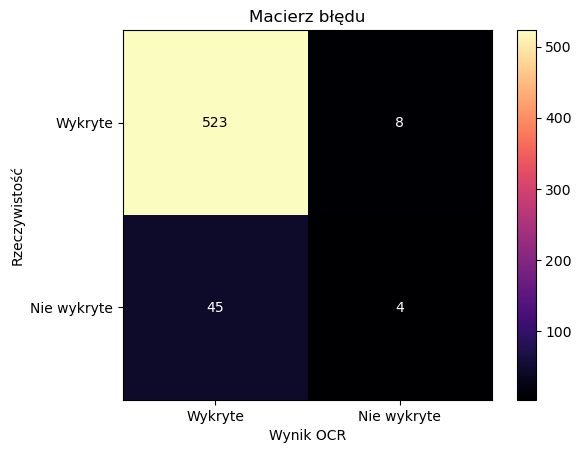
Analizują wyniki można zauważyć, że wiele elementów zostało wykrytych poprawnie, jednak znaczna część została niewykryta. Jakość ogólnej średniej detekcji uplasowała się na poziomie ok. 52,6%. Najgorzej wykrywalną informacją była suma ogólna z paragonu, a najlepiej – sklep.

Przeprowadzono również testy dla tych samych 145 paragonów, jednak tym razem przed przystąpieniem do OCR, przeprowadzono przetwarzanie wstępne obrazów. Rezultaty badań dla poszczególnych informacji można zobaczyć w Tab.3. Ogólne rezultaty detekcji są natomiast widoczne na Rys.25 w postaci macierzy błędu.

Tab. 3 Zestawienie ilości informacji wykrytych i niewykrytych na wszystkich paragonach z uprzednio przeprowadzonym przetwarzaniem wstępnym

|  |  |  |  |
| --- | --- | --- | --- |
| Szukane informacje | Wykryto | Nie wykryto | Średnia |
| Suma | 126 | 19 | 86,9% |
| Data | 124 | 21 | 85,5% |
| Sklep | 133 | 12 | 91,7% |
| Kategoria | 142 | 3 | 97,9% |

Przeprowadzono również testy dla tych samych 145 paragonów, jednak tym razem przed przystąpieniem do OCR, przeprowadzono przetwarzanie wstępne obrazów. Rezultaty badań dla poszczególnych informacji można zobaczyć w Tab.3. Ogólne rezultaty detekcji są natomiast widoczne na Rys.25 w postaci macierzy błędu.



Rys. 27 Macierz błędu detekcji istotnych informacji dla OCR z przetwarzaniem wstępnym

Tym razem wyniki detekcji dla każdego elementu okazały się znacznie lepsze i w każdym przypadku średnia detekcji przekroczyła poziom 85%. Na macierzy błędu dodatkowo uwidacznia się, iż powyżej 90% obiektów niewykrytych przed przetwarzaniem wstępnym zostało wykrytych po jego przeprowadzeniu.

Wraz ze wzrostem średniej detekcji, minimalnie wzrosło także błędne wykrycie elementów, których w rzeczywistości nie ma na paragonie. Warto jednak zwrócić uwagę, że najlepiej wykrywana jest kategoria, która opiera się na detekcji sklepu oraz przygotowanej bazie produktów. Zaraz po niej plasuje się informacja o sklepie, która bazuje również na wcześniej przygotowanej bazie sklepów i ich danych, takich jak chociażby numer NIP. Największe trudności pojawiają się przy sumie i dacie, które ciężko znormalizować. Informacja o sumie może być ciężka do wykrycia, ze względu na częste podobieństwo cyfr oraz brak możliwości pełnego zweryfikowania wyniku. Z tej przyczyny, często wykrywane są wyniki podobne do oczekiwanych, tak jak przedstawiono Tab.4.

Tab. 4 Zestawienie przykładowych wyników oczekiwanych i otrzymanych w wyniku detekcji sumy

|  |  |
| --- | --- |
| Wynik oczekiwany | Wynik otrzymany |
| 91,51 | 1,51 |
| 18,00 | 16,00 |
| 25.00 | 25.06 |
| 26,66 | 28,66 |

Często jednak poprawny wynik znajduje się na dalszym miejscu listy wykrytych wartości, na podstawie których, metodą większościową, wybierany jest ostateczny wynik. Dobrym rozwiązaniem jest więc udostępnienie użytkownikom listy wszystkich wykrytych możliwości, w celu przyspieszenia pracy i umożliwienie edycji, aby kontrolować niewielkie zmiany w odczycie.

Podobnie sytuacja wygląda przy odczytywaniu daty, co można zauważyć analizując Tab.5. W przypadku daty istnieje sposób na kontrolę formatu daty i wprowadzenie systemu naprawczego dla dni miesiąca, roku czy też dni. Problematyczna jest jednak długość odczytywanego tekstu oraz różnorodność jego rozmieszczenia i zapisu. Może to być powodem niższej średniej detekcji.

Tab. 5 Zestawienie przykładowych wyników oczekiwanych i otrzymanych w wyniku detekcji daty

|  |  |
| --- | --- |
| Wynik oczekiwany | Wynik otrzymany |
| 2023-06-18 | 2023-08-18 |
| 2023-03-01 | 2022-03-01 |
| 2023-07-28 | 2003-07-28 |

Porównując wyniki przed przeprowadzeniem przetwarzania wstępnego i po nim, można stwierdzić, iż ma ono znaczny wpływ na odczytywanie wyników. Analizując zestawienie miar, przedstawionych w Tab.6, niemal wszystkie z nich zanotowały wzrost, co zdecydowanie świadczy na korzyść preprocessingu. Jedynie swoistość odnotowała procentowy spadek wartości, co świadczy o tym, że utworzony algorytm ma tendencję do wyszukiwania informacji tam, gdzie ich w rzeczywistości nie ma. Dobrze radzi sobie jednak tam, gdzie informacje te występują.

Tab. 6 Zestawienie miar dla wyników OCR bez i z wykorzystaniem przetwarzania wstępnego

|  |  |  |  |
| --- | --- | --- | --- |
| Porównywana miara | Bez przetwarzania wstępnego | Z przetwarzaniem wstępnym | Różnica |
| Dokładność | 54,8% | 90,7% | 35,9 p.p. |
| Precyzja | 87,2% | 92,1% | 4,9 p.p. |
| Czułość | 58,5% | 98,5% | 40,0 p.p. |
| Swoistość | 21,1% | 8,2% | -12,9 p.p. |
| Wynik F1 | 70,0% | 95,2% | 25,2 p.p. |

Zestawienie wyników detekcji wszystkich elementów dla wszystkich paragonów zostało przedstawione na Rys.26. Jako „Częściowo poprawne” zostały oznaczone wszystkie paragony, na których co najmniej jeden element nie został odczytany prawidłowo. Powyżej 80% tych wyników stanowiły pojedyncze, nieprawidłowo wykryte elementy.

Obraz zawierający tekst, zrzut ekranu, krąg, Czcionka

Opis wygenerowany automatycznie

Rys. 28 Porównanie detekcji wszystkich elementów na paragonie bez i z wykorzystaniem przetwarzania wstępnego

## Testowanie manualne – wybrane funkcjonalności

Ze względu na formę aplikacji większość funkcjonalności i zależności była testowana ręcznie poprzez sprawdzanie reakcji aplikacji na działanie użytkownika.

# Podsumowanie

Głównym założeniem niniejszej pracy dyplomowej było utworzenie projektu aplikacji do zarządzania paragonami oraz jej implementacja. W projekcie należało uwzględnić wykorzystanie OCR do automatycznego odczytu danych z paragonów, a także tworzenie statystyk na ich podstawie, w celu umożliwienia użytkownikowi ich szybkiej analizy. Oprócz tego, aplikacja miała powiadamiać o zbliżającym się czasie gwarancji czy terminu zwrotu, a także oferować inne, atrakcyjne dla potencjalnego użytkownika, funkcjonalności.

Wszystkie przyjęte początkowo cele zostały zrealizowane oraz poszerzone o dodatkowe funkcje, zainspirowane rozwiązaniami dostępnymi na rynku. W ramach pracy zapoznano się z ich wadami i zaletami, a następnie przedstawiono własną propozycję aplikacji. Ostateczna forma, jaka została przyjęta, umożliwia nie tylko zakładaną możliwość korzystania z aplikacji na urządzeniach mobilnych, ale także na wszystkich urządzeniach z dostępem do Internetu. Na ten moment, aplikacja w postaci mobilnej dostosowana jest tylko do systemu Android, jednak w przyszłości istnieje możliwość dostosowania rozwiązania również pod inne systemy.

Funkcjonalność aplikacji opiera się w głównym stopniu na prostych operacjach matematycznych, tworzeniu odpowiednich zapytań pod kątem otrzymania prawidłowych wyników z bazy danych, webscrapingu, a także na automatyzacji procesu analizy danych i przedstawieniu ich na odpowiednich wykresach. Najbardziej złożonym algorytmem jest ten, odpowiedzialny za odczytywanie informacji z paragonów. Wykorzystano w nim ogólnodostępny silnik OCR, rozwijany przez Google – Tesseract, a także techniki przetwarzania obrazów w preprocessingu oraz wyrażenia regularne w postprocessingu. Podjęte działania miały na celu podnieść jakość detekcji istotnych z punktu widzenia użytkownika informacji do tego stopnia, aby w znacznym stopniu zredukować nakład pracy przy manualnym poprawianiu błędnych wartości.

Po przeprowadzeniu testów z wykorzystaniem miarodajnych i powszechnie stosowanych metryk, wyszczególnionych w podrozdziale 6.2, otrzymano wyniki, które potwierdziły wzrost jakości detekcji po przeprowadzeniu postprocessingu. Zanotowany wzrost średniej dokładności wyniósł 37,6 punktu procentowego. Przyczyniło się to do poprawy pierwotnej detekcji na poziomie 52,6% do poziomu 90,2%. Jednocześnie, liczba paragonów, na których początkowo nie było wykrywalne nic, zmniejszyła się do zera.

Analizując otrzymane miary można zauważyć, że model posiada przede wszystkim problem ze swoistością, co przekłada się na to, iż zdarza mu się odnaleźć słowo, którego tak naprawdę nie ma. Zdarza się to zazwyczaj przy detekcji sklepu, gdy jego nazwa jest krótsza niż 5 znaków. Wówczas system może wychwycić inne słowo, które zostało upodobnione do poprawnego w wyniku błędu detekcji. Rozwiązaniem tego może być zmniejszenie wagi krótszych wyrazów, a także zmniejszenie tolerancji algorytmu na literówki w odczytywanych słowach. Poprawę jakości można także osiągnąć poprzez zwiększanie bazy sklepów i produktów w aplikacji, gdyż to na nich, na zasadzie leksykonu, oparty jest postprocessing.

Aplikacja daje stosunkowo dobre wyniki detekcji, a także zapewnia użytkownikowi dostęp do wielu funkcjonalności, gwarantując dodatkowo prostotę w obsłudze. Dodatkowo, złożoność obliczeniowa na większości stron nie jest duża, co umożliwia użytkownikowi komfortowe korzystanie z aplikacji. Największą złożoność obliczeniową posiada algorytm odczytu tekstu z obrazu. Jego wykonanie może trwać od 30 sekund aż do kilku minut. W przyszłości planowane jest rozszerzenie tej funkcjonalności w taki sposób, aby detekcja była prowadzona w tle.

Myśląc w kontekście przyszłości i dalszego rozwoju aplikacji będącej sercem całej pracy, można dostrzec wiele potencjalnych dróg rozwoju. Jedną z nich jest z pewnością dalsze ulepszanie algorytmu detekcji istotnych informacji. Można to zrealizować chociażby przez nauczenie silnika OCR nowych czcionek, które są często spotykane przy analizie polskich paragonów. Istnieją również plany zaimplementowania algorytmu obsługującego kamerę i przeprowadzającego binaryzację w czasie rzeczywistym, co mogłoby ułatwić użytkownikom zrozumienie, jak paragon jest „postrzegany” z perspektywy urządzenia i tym samym zwiększyć szansę na poprawną detekcję.

Oprócz wskazanych ulepszeń, zawsze istnieje możliwość dodania możliwości obsługi większej ilości ofert promocyjnych, a także polepszenia wyglądu i płynności aplikacji poprzez wykorzystanie dedykowanych języków programowania do utworzenia frontendowej części aplikacji, dostosowanej do konkretnego systemu operacyjnego urządzenia.

# Bibliografia

1. Artykuł na temat aplikacji do zarządzania paragonami dostępnych na rynku,

adres: <https://subiektywnieofinansach.pl/aplikacje-do-zarzadzania-paragonami-robimy-zdjecia-paragonow-a-aplikacje-wszystko-odczytuja-i-udostepniaja-nam-statystyki-wydatkow-maja-tez-inne-dodatkowe-funkcje-ktora-z-nich-wybrac-i-czy-w-ogole-war/>,

data uzyskania dostępu [16.10.2023]

1. Dokumentacja silnika OCR Tesseract, adres: [https://tesseract-ocr.github.io/](https://tesseract-ocr.github.io/docs/) ,

data uzyskania dostępu [16.10.2023]

1. Strona aplikacji Pan Paragon, adres: <https://panparagon.pl/>,

data uzyskania dostępu [16.10.2023]

1. Ryszard Tadeusiewicz, Przemysław Korohoda, Komputerowa analiza i przetwarzanie obrazów, Kraków 1997
2. Porównanie dostępnych silników OCR, adres: <https://francescopochetti.com/easyocr-vs-tesseract-vs-amazon-textract-an-ocr-engine-comparison/>, data dostępu [7.12.2023]
3. Dokumentacja biblioteki Python-tesseract, adres: <https://pypi.org/project/pytesseract/>, data dostępu [7.12.2023]
4. Ray Smith, An Overview of the Tesseract OCR Engine, adres: <https://static.googleusercontent.com/media/research.google.com/pl//pubs/archive/33418.pdf>, data uzyskania dostępu [11.12.2023]
5. Ray Smith, A quick tour of the Tesseract Code, adres: <https://tesseract-ocr.github.io/docs/das_tutorial2016/2ArchitectureAndDataStructures.pdf>, data uzyskania dostępu [11.12.2023]
6. Transformacja Hougha, adres: <https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html>, data uzyskania dostępu [15.12.2023]
7. Lecture – Image Processing, adres: <https://wuecampus.uni-wuerzburg.de/moodle/mod/book/tool/print/index.php?id=958001>, data uzyskania dostępu [29.10.2023]
8. Strona aplikacji MintCloud, adres: <https://mintcloud.me/>,

data uzyskania dostępu [16.10.2023]

1. Strona sklepu z aplikacją Billy,

adres: <https://play.google.com/store/apps/details?id=money.inout.billy&hl=pl&gl=US>,

data uzyskania dostępu [16.10.2023]

1. Dokumentacja języka Python, adres: <https://docs.python.org/3/>, data uzyskania dostępu [16.10.2023]
2. Dokumentacja biblioteki Django, adres: <https://docs.djangoproject.com/en/4.2/>, data uzyskania dostępu [16.10.2023]
3. Pandas documentation, <https://pandas.pydata.org/docs/>,

data uzyskania dostępu [16.10.2023]

1. Dokumentacja biblioteki Beautiful Soup, adres: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>, data uzyskania dostępu [16.10.2023]
2. OpenCV documentation, <https://docs.opencv.org/4.x/>, data uzyskania dostępu [16.10.2023]
3. Dokumentacja biblioteki Plotly, adres: <https://plotly.com/python/>,

data uzyskania dostępu [16.10.2023]

1. Dokumentacja biblioteki pdf2image, adres: <https://pypi.org/project/pdf2image/>, data uzyskania dostępu [16.10.2023]
2. Dokumentacja biblioteki numpy, adres: <https://numpy.org/>, data uzyskania dostępu [16.10.2023]
3. Dokumentacja imutils, adres: <https://github.com/PyImageSearch/imutils>, data uzyskania dostępu [16.10.2023]
4. Dokumentacja języka programowania Javascript, adres: <https://developer.mozilla.org/en-US/docs/Web/JavaScript?retiredLocale=pl>, data uzyskania dostępu [16.10.2023]
5. Dokumentacja języka znaczników HTML, adres: <https://developer.mozilla.org/en-US/docs/Web/HTML>, data uzyskania dostępu [16.10.2023]
6. Dokumentacja arkusza stylów CSS, adres: <https://developer.mozilla.org/en-US/docs/Web/CSS>, data uzyskania dostępu [16.10.2023]
7. Dokumentacja biblioteki Bootstrap, adres: https://getbootstrap.com/docs/5.3/getting-started/introduction/, data uzyskania dostępu [16.10.2023]
8. Dokumentacja Android Studio, adres: https://developer.android.com/docs, data uzyskania dostępu [16.10.2023]
9. Dokumentacja języka programowania Kotlin, adres: https://kotlinlang.org/docs/home.html, data uzyskania dostępu [16.10.2023]
10. Dokumentacja PostgreSQL, adres: https://www.postgresql.org/files/documentation/pdf/16/postgresql-16-A4.pdf, data uzyskania dostępu [16.10.2023]
11. Filip Zelic, Anuj Sable, OCR Unlocked: A Guide to Tesseract in Python with Pytesseract and OpenCV, adres: https://nanonets.com/blog/ocr-with-tesseract/, data uzyskania dostępu [11.12.2023]
12. Aleksandra Skalska, Rodzaje aplikacji – webowa, mobilna, natywna, hybrydowa, PWA, adres: <https://udigroup.pl/blog/rodzaje-aplikacji-webowa-mobilna-natywna-hybrydowa-pwa/>, data uzyskania dostępu [16.10.2023]
13. Aplikacje mobilne wykorzystujące Django, adres: https://www.netguru.com/blog/django-apps-examples, data uzyskania dostępu [16.10.2023]
14. Wojciech Jaworski, Miary jakości, adres: https://www.mimuw.edu.pl/~wjaworski/SU/SU04\_miary\_jakosci.pdf, data uzyskania dostępu [29.10.2023]
15. Miary oceny OCR – WER i CER, adres: https://callzen.ai/blog/resources/technical/wer-cer-for-measuring-performance-of-automatic-speech-recognition/, data uzyskania dostępu [16.10.2023]
16. Ben Shaw, Saurabh Badhwar, Andrew Bird, Bharath Chandra K S, Chris Guest: Django: tworzenie nowoczesnych aplikacji internetowych w Pythonie, Gliwice, 2022.
17. Jon Duckett: JavaScript i JQuery: interaktywne strony WWW dla każdego: podręcznik Front-End Developera, Gliwice,2018.
18. Antonio Melé, Django 3: praktyczne tworzenie aplikacji sieciowych, Gliwice 2021.
19. Jon Duckett: HTML i CSS: zaprojektuj i zbuduj witrynę WWW: podręcznik Front-End Developera, Gliwice 2018
20. Dokumetacja ngrok, adres: https://ngrok.com/docs, data uzyskania dostępu [16.10.2023]

# Załączniki

1. Kod i baza danych programu,

https://github.com/Julnowak/receipt-management-app