



POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Krzysztof Kania

Nr albumu 15040

**Elastyczny system do zarządzania dokumentami
finansowymi**

Praca magisterska napisana

pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, wrzesień 2018

Spis treści

1	Cel pracy	6
1.1	Rozwiązania przyjęte w pracy	6
1.2	Rezultat pracy	7
2	Analiza rozwiązań dostępnych na rynku	8
2.1	Aplikacja „gdzieparagon.pl”	8
2.2	Aplikacja „Smart Paragon”	9
2.3	Aplikacja „Paragon - gwarancje pod ręką”	9
2.4	Aplikacja „Kwitki”	10
2.5	Aplikacja „ToCoMoje”	11
2.6	Podsumowanie	12
3	Koncepcja nowego systemu	15
3.1	Wizja systemu	15
3.2	Specyfikacja wymagań funkcjonalnych	16
3.2.1	Możliwość implementacji wtyczek do obsługi logowania oraz miejsca trzymania dokumentów	17
3.2.2	Możliwość implementacji generycznych dodatków uruchamianych w przez użytkownika	17
3.2.3	Możliwość implementacji generycznych dodatków uruchamianych podczas dodawania nowych dokumentów	18
3.2.4	Możliwość dodawania oraz edycji dokumentów	18
3.2.5	Możliwość tworzenia oraz edycji kategorii	19

3.2.6	Możliwość wyszukiwania dokumentów	19
3.2.7	Możliwość uruchamiania dodatków	19
3.2.8	Automatyczne przetwarzanie dokumentu podczas dodawania	20
3.2.9	Edycja danych profilu użytkownika	20
3.3	Specyfikacja wymagań pozafunkcyjnych	20
3.3.1	Środowisko uruchomieniowe	21
3.3.2	Wymaganie czytelności aplikacji	21
3.3.3	Wymaganie utrzymania aplikacji	22
3.3.4	Bezpieczeństwo aplikacji	22
3.4	Architektura aplikacji	23
3.4.1	Architektura REST API	24
3.4.2	Architektura Bazy danych	28
3.4.3	Architektura aplikacji front-end'owej	31
4	Zastosowane technologie, narzędzia i standardy	32
4.1	Użyte Technologie	32
4.1.1	ASP.NET Core 2.0	32
4.1.2	SQL Server 2017 Express	33
4.1.3	Angular 5.0 i TypeScript	33
4.1.4	Entity Framework Core	34
4.1.5	AutoMapper	34
4.1.6	Flurl	35
4.2	Użyte narzędzia	35

4.2.1	Visual Studio 2017 Community	35
4.2.2	ReSharper	35
4.2.3	Visual Studio Code	36
4.2.4	Manager paczek NuGet	36
4.2.5	Manager paczek npm	36
4.2.6	Google Cloud Vision API	37
4.2.7	Biblioteka ng-swagger-gen	37
4.2.8	Swagger	38
4.2.9	Narzędzie do kontroli wersji Git	38
4.3	Standard OAuth 2.0	39
5	Prototyp nowego rozwiązania	41
5.1	Istotne rozwiązania implementacyjne	41
5.1.1	Generyczny moduł do obsługi logowania	41
5.1.2	Generyczny moduł do obsługi przetwarzania dokumentów	44
5.1.3	Moduł realizujący OCR dokumentu	45
5.1.4	Generyczny moduł dodatków uruchamianych przez użytkownika	50
5.1.5	Dodatki zaimplementowane w prototypie jako dodatki uruchamiane przez użytkownika	51
5.2	Przykłady użycia aplikacji	56
5.2.1	Logowanie do aplikacji	56
5.2.2	Dodawanie nowego dokumentu finansowego	58
5.2.3	Zarządzanie kategoriami zdefiniowanymi przez użytkownika	61

5.2.4	Ekran służący do uruchamiania dodatków	61
5.2.5	Ekran edycji profilu zalogowanego użytkownika	62
6	Podsumowanie	64
6.1	Dalsze kierunki rozwoju prototypu	64
6.2	Napotkane problemy podczas implementacji	64
6.3	Zakończenie	65
	Bibliografia	66
	Wykaz rysunków	69

Streszczenie

Praca dotyczy utworzenia elastycznego systemu do zarządzania dokumentami finansowymi. W pracy zastosowano autorskie rozwiązania, dzięki którym prototyp jest modułowy. Programy obecne na rynku nie dają możliwości rozszerzania ich o dodatkowe funkcjonalności. Nie posiadają także otwartych interfejsów programistycznych, które pozwalałyby na integrację zewnętrznych systemów za pośrednictwem protokołu HTTP z wykorzystaniem REST API. Konkurencyjne rozwiązania nie dają także możliwości decydowania przez użytkownika gdzie będą trzymane jego dokumenty. Ta opcja może być szczególnie ważna dla osób przywiązujących szczególną uwagę do bezpieczeństwa swoich danych.

Dodatkowym atutem autorskiego systemu jest możliwość hostowania go na zewnętrznych serwerach (np. Azure, Amazon) lub na lokalnych maszynach użytkownika - nad którymi ma pełną kontrolę i możliwość własnego dbania o bezpieczeństwo systemu.

System został zaimplementowany z wykorzystaniem frameworków otwartoźródłowych takich jak ASP.NET Core 2.0 oraz Angular 5.0.

Słowa kluczowe: Wtyczki, Komponenty, Generyczność, Dokumenty finansowe, REST API, Angular, ASP.NET Core

1 Cel pracy

Celem pracy jest określenie potrzeb oraz funkcjonalności jakimi powinien charakteryzować się elastyczny system pozwalający na zarządzanie dowolnymi dokumentami finansowymi. Kolejnym etapem jest wykonanie projektu architektonicznego oraz implementacji systemu zgodnie z założeniami. Program ma zapewnić łatwość w rozszerzaniu jego możliwości o dodatkowe moduły. Dodatkowo użytkownik systemu powinien mieć możliwość wybrania gdzie będą trzymane są jego dokumenty finansowe.

1.1 Rozwiązania przyjęte w pracy

System został podzielony na dwie główne części:

1. **Front-end** - responsywna¹ aplikacja uruchamiana w przeglądarce internetowej, napisana przy użyciu języka TypeScript oraz frameworka Angular w wersji 5.1,
 2. **Back-end** - jest to aplikacja napisana w języku C#, oparta na frameworku² *ASP.NET Core 2.0*. Projekt ten ma za zadanie wystawić dla aplikacji front-end'owej WebAPI³ w technologii REST działający na protokole HTTP.
- Baza danych użyta w projekcie to **SQL Server 2017** w wersji Express Edition.
 - Dodatkowym założeniem jest możliwość uruchomienia systemu na dowolnej maszynie, zarówno wyposażonej w system operacyjny Windows, Linux lub macOS.

¹Technika projektowania stron WWW, aby jej wygląd dostosowywał się do rozmiaru urządzenia. Źródło:

Wikipedia

²Platforma programistyczna - szkielet do budowy aplikacji

³Interfejs komunikacyjny

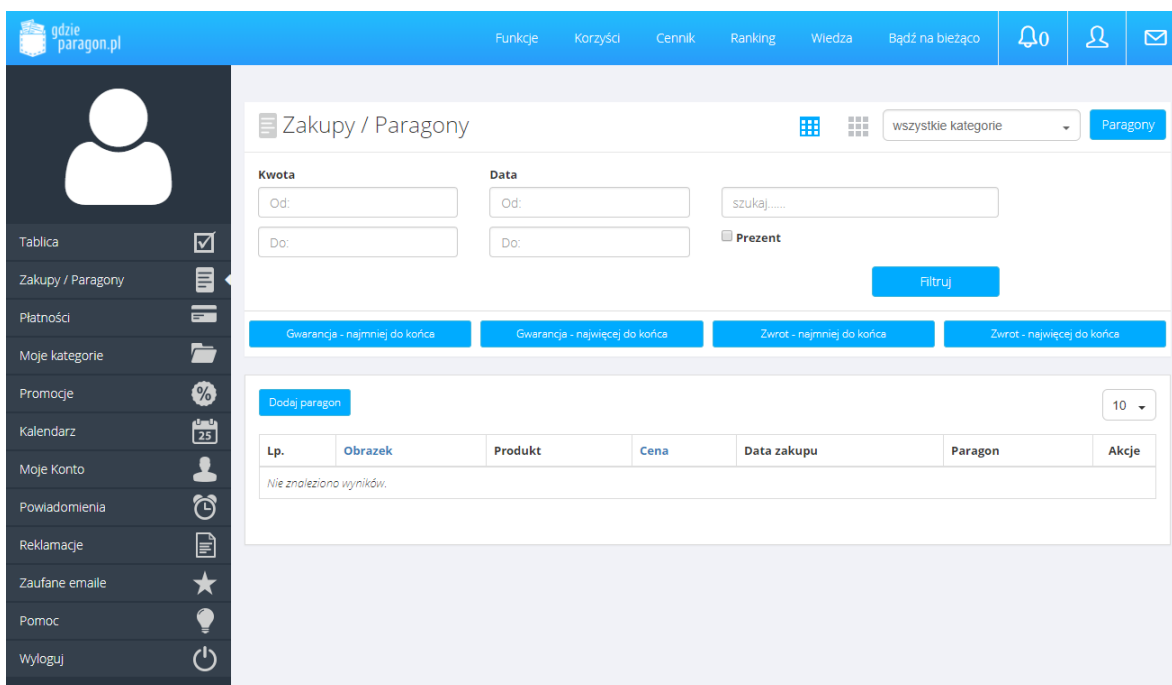
1.2 Rezultat pracy

Głównym efektem pracy jest gotowy projekt systemu do zarządzania dokumentami finansowymi wraz z jego działającą i przetestowaną implementacją. System ma za zadanie rozwiązać problem klasyfikacji oraz archiwizacji dokumentów w formie elektronicznej. Dodatkowym założeniem jest możliwość prostego rozszerzenia aplikacji o dodatkowe funkcjonalności poprzez dołożenie kolejnych modułów. Moduły powinny być proste w implementacji i dawać szerokie możliwości ingerencji w główny rdzeń systemu.

2 Analiza rozwiązań dostępnych na rynku

Przed przystąpieniem do fazy projektowania nowego systemu warto dowiedzieć się jakie gotowe rozwiązania dostępne są obecnie na rynku. W tym celu została przygotowana analiza porównawcza, która skupia się na systemach posiadających zbliżone możliwości zarządzania dokumentami finansowymi. Celem tego rozdziału jest przedstawienie różnic lub braków funkcjonalności w obiektywny i przejrzysty sposób. Analiza została przygotowana w oparciu o programy obecne na rynku. Jednocześnie została dokonana interpretacja tych rozwiązań pod kątem użyteczności wyżej wymienionych rozwiązań, wraz z dokładnym wyszczególnieniem napotkanych problemów lub braków funkcjonalnych w przypadku każdej z aplikacji.

2.1 Aplikacja „gdzieparagon.pl”



Rysunek 1: Główne okno aplikacji „gdzieparagon.pl”, Źródło: opracowanie własne

Aplikacja pozwala zarówno na dodawanie dokumentów finansowych jak i zarządzanie kategoriami. Dokumenty trzymane są na serwerach twórców aplikacji. Niestety nie została

przewidziana dla użytkowników opcja wykonania kopii bezpieczeństwa dokumentów. Dodatkowo warto zauważyć, że korzystanie z aplikacji jest **płatne** na zasadzie SaaS⁴.

Korzystając z tego rozwiązania nie mamy możliwości integracji z wystawionym przez firmę *gdzieparagon.pl* API⁵ lub też rozbudowania system, a tym samym większe dopasowanie go do własnych potrzeb. Aplikacja nie została także w pełni przystosowana do widoku mobilnego. Interfejs użytkownika dostępny z poziomu przeglądarki internetowej został zaprezentowany na *rysunku 1*.

2.2 Aplikacja „Smart Paragon”

Aplikacja została zaprojektowana zarówno dla systemów mobilnych działających pod kontrolą systemu iOS oraz Android. Brakuje tutaj wersji działającej w przeglądarce użytkownika.

Po krótkich testach aplikacje okazały się mało stabilne w działaniu, a dodane za jej pomocą dokumenty potrafią znikać z systemu co ma bezpośredni wpływ na utratę zaufania do trzymanych w jej bazie danych.

Autorzy nie zadbali także o utworzenie dodatkowych funkcjonalności jak np. informacje o zbliżającym się terminie końca gwarancji lub tworzeniu raportów na podstawie zgromadzonych danych.

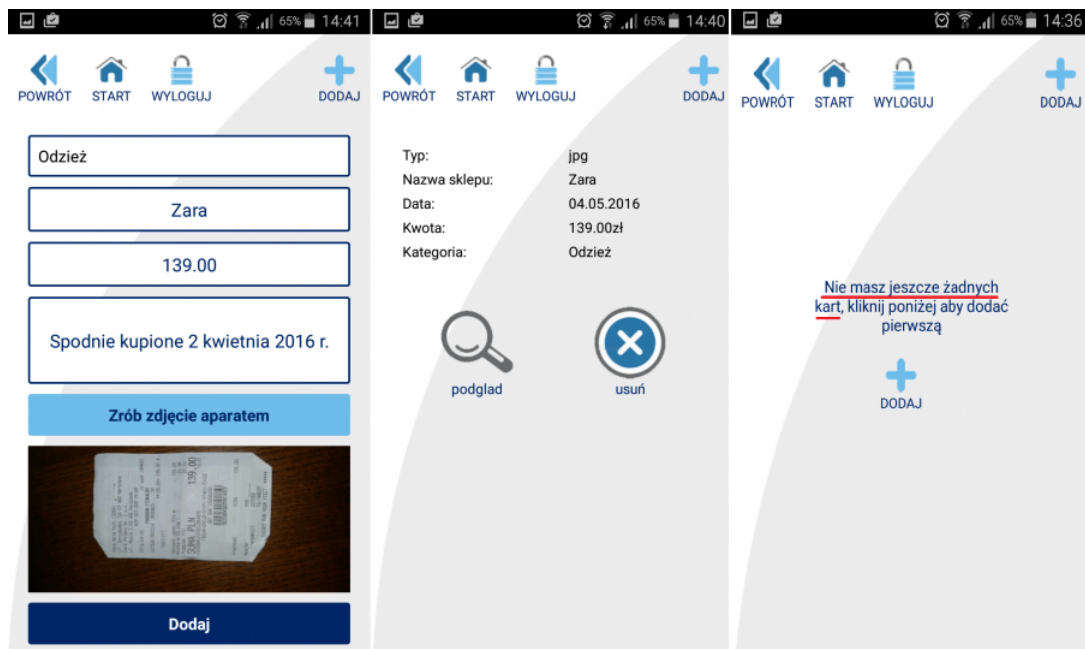
Na plus zasługuje czytelny interfejs graficzny. Przykładowe ekrany zostały zaprezentowane na *rysunku 2*.

2.3 Aplikacja „Paragon - gwarancje pod ręką”

Aplikacja powstała na urządzenia mobilne działające pod kontrolą systemu iOS oraz Android. Możemy zarejestrować się zarówno za pomocą nazwy użytkownika i hasła lub za po-

⁴Software as a Service - z ang. oprogramowanie jako usługa

⁵Application Programming Interface – z ang. interfejs programistyczny aplikacji



Rysunek 2: Główne okna aplikacji „Smart Paragon”, Źródło: payu.pl/blog

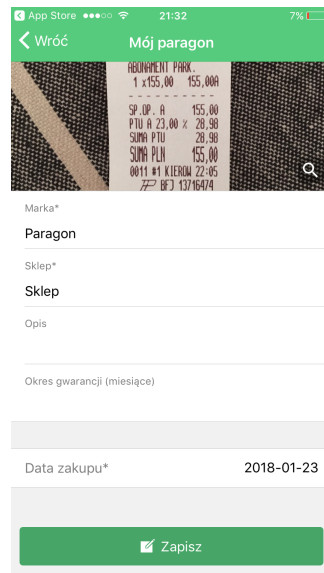
średnictwem portalu Facebook. Po zalogowaniu mamy możliwość jedynie przeglądania wcześniej dodanych paragonów lub dodania nowego.

W aplikacji brakuje czytelnej informacji gdzie trzymane są nasze dokumenty. Nie mamy także żadnych dodatkowych opcji zarządzania, katalogowania oraz uzupełnienia wcześniej dodanych informacji o dodatkowe informacje. Możemy więc powiedzieć, że aplikacja charakteryzuje się ubogimi funkcjonalnościami dostępnymi dla użytkownika. Główne okno aplikacji zostało zaprezentowane na *rysunku 3* oraz *rysunku 4*.

2.4 Aplikacja „Kwitki”

Tak samo jak w przypadku wyżej wymienionych konkurentów, tutaj także mówimy o aplikacjach jedynie mobilnych. Twórcy skupili się na systemie iOS oraz Android. W przypadku tej aplikacji na uwagę zasługuje mechanizm odczytu danych z dokumentu na podstawie przesłanego zdjęcia (Automatyczny OCR ⁶). Niestety rozwiązanie zaimplementowane w tej

⁶Optical Character Recognition - z ang. Optyczne rozpoznawanie znaków

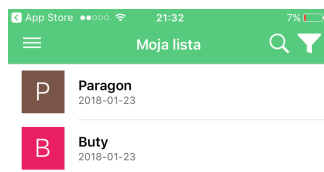


Rysunek 3: Okno dodawania nowego dokumentu finansowego w aplikacji „Paragon - gwarancje pod ręką”, Źródło: opracowanie własne

aplikacji nie działa zbyt precyzyjnie. Sprawia to, że korzystanie z niego nie jest wygodne, nie przyspiesza także procesu archiwizacji dokumentów. Wymaga także od użytkownika późniejszego sprawdzenia automatycznie wprowadzonych danych i dokonania licznych poprawek. Sprawia to, że aplikacja skutecznie zniechęca do korzystania z niej. Użytkownik nie ma także możliwości dopasowania funkcjonalności do własnych potrzeb. Brakuje także możliwości integracji z oferowanymi przez aplikację rozwiązaniami.

2.5 Aplikacja „ToCoMoje”

Jest to jedna z najbardziej popularnych aplikacji tego typu dostępnych na rynku na urządzeniach mobilnych (działających pod kontrolą systemu iOS i Android). Pozwala zarówno na skanowanie dokumentów, dodawanie dokładnego opisu oraz wzbogacanie dokumentów o tagi, które pozwolą w późniejszym czasie skrócić czas szukania interesującego nas dokumentu lub grupy dokumentów. Niestety także w przypadku tej aplikacji użytkownik nie ma możliwości na przechowywanie dokumentów w miejscu przez niego wybranym.



Rysunek 4: Główne okno aplikacji „Paragon - gwarancje pod ręką”, Źródło: opracowanie własne

Użytkownicy tej aplikacji często napotykają również problem z dostępem do swoich danych spowodowany brakiem możliwości zalogowania się na swoje konto. Problem ten połączony z brakiem wsparcia ze strony twórców skutkuje realnym ryzykiem utraty wszystkich przechowywanych dokumentów oraz innych danych. Sprawia to, że zaufanie do tego produktu wyraźnie spadło, ze względu na jego niestabilność oraz brak wsparcia w przypadku problemów od strony twórców.

2.6 Podsumowanie

Przytoczone aplikacje, które są obecne na rynku charakteryzowały się bardzo zróżnicowanym działaniem oraz stabilnością w działaniu. Żaden z przedstawionych powyżej systemów nie spełniał wszystkich wymagań funkcjonalnych, które zostały uwzględnione w tej pracy magisterskiej. Głównie brakowało możliwości wybrania przez użytkownika, czy chciałby on korzystać z systemu na zasadach subskrypcji SaaS ⁷, czy może mieć możliwość hostowania całego systemu na dowolnie wybranym przez siebie hostingu lub lokalnej maszynie. W przy-

⁷Software as a Service - z ang. oprogramowanie jako usługa

padku maszyny lokalnej ważna jest możliwość ograniczenia dostępności systemu tylko do sieci lokalnej. Może to być szczególnie ważne, kiedy zależy nam na dodatkowym bezpieczeństwie trzymanych w systemie danych.

Podsumowując główne cechy konkurencji:

- tylko jedna aplikacja (*gdzieparagon.pl*) dawała użytkownikowi możliwość wygodnego korzystania zarówno na urządzeniach mobilnych jak i za pośrednictwem przeglądarki internetowej,
- brak możliwości zapisu dokumentów w dowolnym miejscu,
- brak responsywności interfejsu graficznego na urządzeniach mobilnych - w przypadku systemów uruchamianych w przeglądarce internetowej,
- brak możliwości korzystania z wersji webowej, która działaniem oraz funkcjonalnością przypomina aplikację mobilną,
- brak możliwości tworzenia własnych modułów rozszerzających działanie porównywalnych gotowych systemów,
- brak możliwości integracji z wewnętrznym API systemów.

Na podstawie powyższej analizy można jasno stwierdzić, że w przypadku obecnych rozwiązań można mówić o braku podejścia modułowego do projektowanych aplikacji. Nie mamy możliwości ustawienia systemu w taki sposób, aby w pełni odpowiadał naszym wymaganiom. Nie jesteśmy także w stanie napisać własnego programu, który wykorzystywał by dane zgromadzone w zaprezentowanych powyżej systemach. Na rynku obecne są tak zwane systemy pudełkowe.

Alternatywa w postaci zaproponowanego w tej pracy magisterskiej systemu wprowadza innowację w zakresie klasyfikacji oraz archiwizacji dokumentów. Wolna jest o wad rozwiązań obecnych na rynku. Daje możliwość dowolnego rozszerzania działania kluczowych procesów

systemu, dzięki czemu nawet najbardziej wymagający użytkownicy będą mieli możliwość dopasowania systemu do swoich potrzeb.

3 Koncepcja nowego systemu

Dokładne przeanalizowanie rozwiązań obecnych na rynku pozwoliło odkryć ich słabe strony i jednocześnie zaproponować nowe narzędzie wolne od wcześniejszych niedogodności. W wyniku analizy powstała wizja nowego systemu wraz z listą najważniejszych funkcjonalności, które powinny zostać zaimplementowane.

3.1 Wizja systemu

W dzisiejszych czasach każdy z nas jest zobowiązany do posiadania wielu dokumentów finansowych. Niestety dokumenty w formie fizycznej narażone są na wiele zagrożeń, które mogą sprawić że ulegną zniszczeniu. Obecnie prawo traktuje dokumenty w formie elektronicznej na równi z ich oryginałami. Otwiera to nowe możliwości w zarządzaniu bardzo dużą ilością papierowych dokumentów, które do tej pory były przetrzymywane tylko w formie dokumentów fizycznych. Na pierwszym miejscu zalet związanych z przejściem na cyfrową archiwizację można wyróżnić oszczędność miejsca ich składowania. Dodatkowo posiadając elektroniczne kopie możemy w łatwy sposób dokonać ich katalogowania, analizy oraz późniejszego dużo szybszego wyszukiwania. Problem, który stoi przed każdym, kto chciałby przejść w stronę cyfrowej archiwizacji to dylemat, który system powinien wybrać. Na rynku aplikacji mobilnych znajduje się wiele pozycji, które pozwalają na archiwizację dokumentów. Nie zawsze natomiast wspomniane systemy w jasny i czytelny sposób określają zasady na których będą przetwarzane nasze dokumenty. Dbając o własną prywatność i poufność w wielu przypadkach nie będziemy ryzykowali wycieku swoich dokumentów, a co za tym idzie danych. Kiedy dodatkowo mamy do czynienia z płatnym rozwiązaniem to wielu z nas zastanawia się, czy subskrypcja na którą się decydujemy będzie atrakcyjna za np. rok. Jeśli nie, to w najlepszym przypadku będzie czekało nas przepisywanie wszystkich zgromadzonych w wybranej wcześniej aplikacji danych oraz ręczne pobieranie naszych dokumentów. Pozostaje jeszcze fakt ewentualnego nagłego zniknięcia aplikacji z rynku, a wraz z nią naszych wszystkich dokumentów.

Wszystkie powyższe argumenty potwierdzają, że wizja prototypu aplikacji do zarządzania dokumentami finansowymi zaproponowana w tej pracy będzie wolna od wyżej wymienionych problemów. Jego głównym założeniem jest modułowe podejście do przetwarzania dokumentów finansowych. Modułowość zastosowaną w prototypie można podzielić na trzy kategorie:

1. **Modułowość ze względu na miejsce trzymany dokumentów** - dokumenty są trzymane na koncie użytkownika. Miejscem są dobrze znani dostawcy usług np. Google Drive lub Dropbox. Jednocześnie wspomnieni dostawcy pełnią funkcję uwierzytelnienia. Użytkownik korzystając z systemu nie musi więc zakładać kolejnego konta, a jedynie korzysta z konta, które już posiada. Logowanie odbywa się za pośrednictwem standardu OAuth 2.0.
2. **Modułowość ze względu na przetwarzanie dokumentów podczas ich dodawania** - w tym przypadku każdy kto chciałby rozszerzyć możliwości automatyzacji dodawania nowych dokumentów może zaimplementować dodatek, który będzie uruchamiany podczas dodawania każdego dokumentu finansowego. Uruchomienie dodatków odbywa się na wyraźne żądanie użytkownika, po wybraniu dokumentu. Aplikacja pozwala na implementację oraz uruchomienie wielu dodatków w jednym momencie.
3. **Modułowość ze względu na dodatki uruchamiane w kontekście aplikacji** - aplikacja daje możliwość implementacji dodatków, które są uruchamiane w kontekście użytkownika lub całej aplikacji. Dodatki te mają za zadanie wspomaganie poprzez automatyzację całego procesu zarządzania dokumentami. Możliwości jakie mogą zostać zaimplementowane w ramach tej kategorii dodatków są bardzo szerokie i zostały opisane w kolejnych rozdziałach pracy.

3.2 Specyfikacja wymagań funkcjonalnych

Rozpoczęcie prac nad każdą aplikacją powinno być zapoczątkowane określeniem wymagań funkcjonalnych, jakie program powinien spełniać. Poniżej została przedstawiona lista

wymagań jakie zostały wyszczególnione dla systemu przeznaczonego do zarządzania dokumentami finansowymi.

3.2.1 **Możliwość implementacji wtyczek do obsługi logowania oraz miejsca trzymania dokumentów**

Użytkownicy powinni mieć możliwość implementacji generycznych wtyczek, które są używane podczas logowania oraz komunikacji z zewnętrznym serwisem, który odpowiada za bezpieczne trzymanie dokumentów. Implementacja nowej wtyczki powinna opierać się na implementacji interfejsu programistycznego napisanego w języku C# oraz utworzeniu paczki Nuget [22], która w późniejszym etapie jest użyta w projekcie. Po dodaniu nowo utworzonej paczki, aplikacja w sposób automatyczny powinna jej użyć, a po stronie aplikacji klienckiej dostępnej dla użytkowników powinna zostać wyświetlona opcja logowania za jej pomocą. Skanowanie w poszukiwaniu zaimplementowanych wtyczek powinno odbywać się z użyciem mechanizmu refleksji.

3.2.2 **Możliwość implementacji generycznych dodatków uruchamianych w przez użytkownika**

System powinien posiadać możliwość użycia generycznych dodatków, które powinny mieć możliwość automatyzacji zarządzania dokumentami finansowymi lub też innymi częściami systemu. Implementacja dodatku powinna odbywać się w sposób analogiczny jak w przypadku dodatków używanych do mechanizmu logowania oraz miejsca trzymania dokumentów. Dodatki powinny mieć możliwość uruchamiania w kontekście pojedynczego użytkownika lub też całej aplikacji. Programista odpowiedzialny za stworzenie nowego dodatku powinien mieć możliwość także dostępu do danych użytkownika trzymany w prototypie aplikacji w tym do danych kontaktowych, które są trzymane w bazie danych aplikacji.

3.2.3 **Możliwość implementacji generycznych dodatków uruchamianych podczas dodawania nowych dokumentów**

Podczas procesu dodawania nowych dokumentów finansowych istnieje możliwość automatycznego przetworzenia dodawanego dokumentu przez system. Dzięki temu na podstawie informacji wprowadzonych przez użytkownika system może dokonać automatycznego wypełnienia innych pól, które opisują dokument finansowy. System powinien dawać możliwość implementacji własnych dodatków, które na etapie dodawania dokumentów będą uruchamiane na żądanie i będą brały udział w przetwarzaniu dokumentów. Implementacja tego typu dodatku powinna odbywać się poprzez implementację interfejsu programistycznego, a następnie tak samo jak w przypadku innych dodatków opisanych w punktach powyżej, dodatek powinien zostać opakowany w paczkę Nuget [22] i dodany do głównej aplikacji.

3.2.4 **Możliwość dodawania oraz edycji dokumentów**

Użytkownicy aplikacji powinni mieć możliwość pełnego zarządzania swoimi dokumentami finansowymi. Pod pojęciem zarządzania można wyszczególnić takie akcje jak:

- Możliwość dodawania dokumentów,
- Możliwość edycji istniejących dokumentów,
- Możliwość usuwania dokumentów z systemu.

Każdy z dokumentów posiada listę atrybutów jakie mogą być użyte do jego opisu. Atrybuty można dodatkowo podzielić na te, które są wymagane w kontekście każdego dokumentu oraz na te, które są opcjonalne:

1. **Nazwa dokumentu** - atrybut wymagany
2. **Opis dokumentu** - atrybut opcjonalny
3. **Nazwa sprzedawcy** - atrybut opcjonalny

4. **Cena** - atrybut opcjonalny
5. **Data zakupu** - atrybut opcjonalny
6. **Data końca gwarancji** - atrybut opcjonalny
7. **Kategorie przypisane do dokumentu** - atrybut opcjonalny

Dodatkowo każdy dokument tworzony w systemie powinien posiadać załączony plik w formacie obrazka (*image/**) lub dokumentu PDF (*application/pdf*).

3.2.5 **Możliwość tworzenia oraz edycji kategorii**

Jedną z możliwości łatwego kategoryzowania dokumentów jest nadawanie im kategorii. Dzięki temu można w późniejszym czasie w szybszy sposób odszukać dokumenty właśnie z ich pomocą. System powinien dawać możliwość pełnego zarządzania kategoriami. Akcje jakie są dozwolone w kontekście kategorii to: dodawanie nowych kategorii, edycja istniejących kategorii oraz usuwanie wcześniej zdefiniowanych kategorii.

3.2.6 **Możliwość wyszukiwania dokumentów**

Użytkownicy prototypu powinni mieć możliwość łatwego wyszukiwania wcześniej dodanych dokumentów za pomocą pola tekstowego, w które można wpisać jakąkolwiek informację, która została uzupełniona podczas dodawania dokumentu finansowego.

Dodatkowo system powinien mieć możliwość pokazania wszystkich dokumentów, które zostały przypisane do danej kategorii.

3.2.7 **Możliwość uruchamiania dodatków**

W prototypie poza możliwością implementacji własnych generycznych dodatków, powinna być także możliwość ich uruchamiania przez użytkownika na żądanie. Dodatki powinny być uruchamiane na wyraźne polecenie użytkownika. Każde uruchomienie powinno

być poprzedzone komunikatem, który będzie informował użytkownika o akcji, którą zamierza wykonać.

3.2.8 Automatyczne przetwarzanie dokumentu podczas dodawania

Aplikacja powinna dawać możliwość skorzystania z automatycznego wstępnego przetwarzania dokumentu podczas jego dodawania. Funkcjonalność ta została może zostać zaimplementowana jako jeden z typów generycznych dodatków. Skorzystanie z tej funkcjonalności powinno odbywać się poprzez naciśnięcie przycisku „*Przetwórz dokument*” znajdującego się na ekranie dodawania nowych dokumentów.

3.2.9 Edycja danych profilu użytkownika

Prototyp systemu powinien posiadać funkcjonalność pełnej edycji danych znajdujących się w profilu użytkownika. Każdy użytkownik powinien posiadać możliwość uzupełnienia takich danych o sobie jak:

- Imię,
- Nazwisko,
- Adres e-mail,
- Numer telefonu.

3.3 Specyfikacja wymagań pozafunkcyjnych

Na wymagania pozafunkcyjne składają się wymagania, które nie definiują bezpośrednio „co” aplikacja ma wykonywać, a definiują istotne szczegóły techniczne, które muszą zostać spełnione, aby aplikacja mogła działać w prawidłowy sposób.

Przykładami grup wymagań pozafunkcyjnych mogą być:

- Wymagania środowiska uruchomieniowego aplikacji,
- Wymagania użyteczności,
- Wymagania dotyczące późniejszego utrzymania aplikacji,
- Wymagania związane z bezpieczeństwem aplikacji,
- Wymagania związane z wydajnością aplikacji.

W nawiązaniu do powyższych grup można wyszczególnić następujące wymagania pozafunkcjonalne dla aplikacji do zarządzania dokumentami finansowymi.

3.3.1 Środowisko uruchomieniowe

Aplikacja działa w przeglądarce internetowej zarówno na komputerze i urządzeniu mobilnym. Minimalna szerokość ekranu jaka jest wspierana przez aplikację to 320px. Poniżej tej rozdzielczości następują zniekształcenia w wyświetlanej treści. Dodatkowo sposób prezentacji różni się znacząco od tego w jakiej rozdzielczości uruchamiamy aplikację, tak aby użytkownik otrzymał treść w możliwie przejrzystej postaci.

Wybranie przeglądarki internetowej sprawia, że pomijamy w tym przypadku problem wdrażania aplikacji bezpośrednio na maszynach klientów. Zostaje także znacząco uproszczony proces aktualizacji, którą w przypadku aplikacji webowych można wykonać tylko na serwerze. Klient przy każdym wejściu na stronę będzie korzystał w najnowszej wersji.

3.3.2 Wymaganie czytelności aplikacji

Interfejs graficzny aplikacji został zaprojektowany w sposób czytelny. Został zastosowany podział na menu boczne oraz główną część aplikacji, gdzie prezentowane są aktualnie wybrane informacje. W wersji mobilnej menu boczne zostało zastąpione przez rozwijane menu górne. Wszystkie ekrany posiadają spójne style graficzne, a przyciski akcji posiadają taki

sam wygląd i kolor. Dzięki temu użytkownik końcowy już po chwili dokładnie wie w jaki sposób korzystać z aplikacji.

Dodatkowo komunikaty skierowane do użytkownika informujące zarówno o powodzeniach jak i niepowodzeniach, zostały zaprojektowane w sposób czytelny i jednolity dla każdego z ekranów. Komunikaty wymagające specjalnej uwagi wymagają od użytkownika dodatkowego potwierdzenia wykonywanej akcji. Dzięki temu mamy pewność świadomości wykonania konkretnych działań.

Aplikacja nie posiada wymagań, które określały by jej użycie przez osoby z wadą wzroku. W obecnej wersji jest możliwość zwiększenia rozmiaru czcionki z poziomu przeglądarki internetowej, co będzie skutkowało znacznym zwiększeniem komfortu użytkowania przez osoby z dysfunkcją narządu wzroku.

3.3.3 Wymaganie utrzymania aplikacji

W przypadku każdego programu komputerowego możemy stwierdzić z dużym prawdopodobieństwem, że w niedalekiej przyszłości będzie on wymagał implementacji nowych funkcjonalności lub poprawienia ewentualnych błędów, które mogą się uwydatnić podczas użytkowania. Kod programu powinien zostać napisany w taki sposób, aby późniejsze zmiany nie powodowały nadmiernej pracy przy utrzymaniu [1]. Szczególnie pomocna może okazać się tutaj zasada **SOLID** [6] odnosząca się bezpośrednio do programowania obiektowego.

3.3.4 Bezpieczeństwo aplikacji

Obecnie każda projektowana aplikacja powinna dużą uwagę poświęcać jej bezpieczeństwu. W prototypie aplikacji do zarządzania dokumentami finansowymi zostało założonych kilka ważnych wymagań pozafunkcyjnych, które podnoszą bezpieczeństwo całej aplikacji:

1. Aplikacja nie przechowuje kont użytkowników. Został zaimplementowany mechanizm generycznych dostawców, który wykorzystywany jest podczas logowania. Całość oparta

jest o standard OAuth [7]. Dzięki temu użytkownicy mogą wykorzystywać swoje konta z popularnych serwisów takich jak np. Google, Dropbox w celu zalogowania do prototypu.

2. Dokumenty użytkowników są przechowywane na serwerach dostawców do których logowanie odbywa się za pośrednictwem standardu OAuth. Dzięki temu kwestie odpowiedniego zabezpieczenia przechowywanych danych są po stronie dobrze znanych serwisów.
3. Każde żądanie wysłane z aplikacji front-end'owej do back-endu musi zawierać token, który jest generowany podczas logowania. Token ma postać zaszyfrowaną (za pomocą algorytmu szyfrującego AES [8]). Po odszyfrowaniu każdy token posiada zbiór informacji dotyczących użytkownika, a także token, który jest używany do łączenia i pobierania informacji o dokumentach od zewnętrznego dostawcy. Przyjęcie takiego rozwiązania sprawia, że użytkownik jest w stanie zobaczyć jedynie token, jaki został wygenerowany do komunikacji z wewnętrznym API. Nie jest natomiast w stanie samodzielnie korzystać z tokenu przeznaczonego do komunikacji z dostawcą.

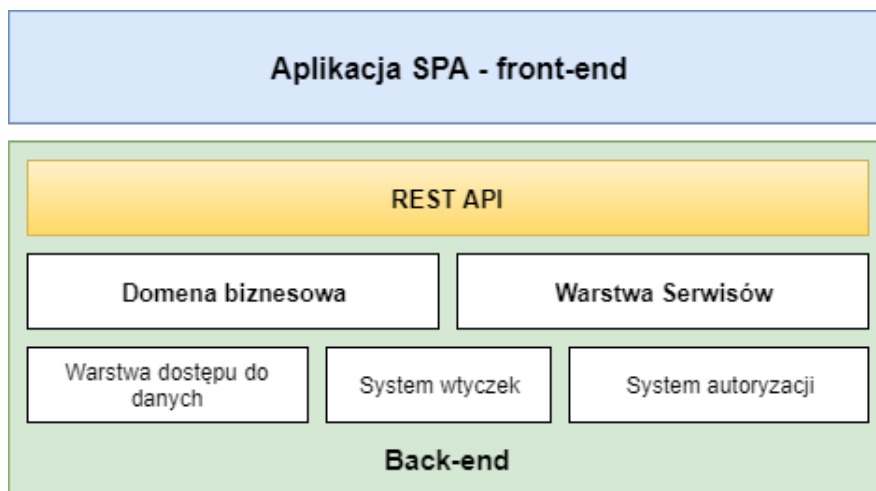
3.4 Architektura aplikacji

Podczas projektowania aplikacji jedną z ważniejszych rzeczy jest dokładne przemyślenie, a w późniejszym etapie zaimplementowanie jej architektury. Od decyzji podjętych na początku może zależeć działanie całego systemu. Kompozycja kodu ma bezpośredni wpływ na:

- Wydajność aplikacji,
- Możliwość jej późniejszej rozbudowy,
- Czystość tworzonego kodu, a co za tym idzie modyfikację aplikacji w przyszłości,
- Testowanie kodu.

Prototyp aplikacji powstał w architekturze wielowarstwowej, z wyraźnym podziałem na część **front-end** oraz **back-end**. Dzięki temu już na początku można wyraźnie oddzielić

prace poświęcone na tworzenie części widocznej dla użytkownika od REST'owego API, z którego korzysta aplikacja front-end'owa. Dodatkowo w części back-end'u także można wyróżnić warstwy z których ta część została zbudowana. Diagram uproszczonej architektury systemu został zaprezentowany na *Rysunku 5*.



Rysunek 5: Diagram architektury aplikacji, Źródło: opracowanie własne

3.4.1 Architektura REST API

Jednym z elementów architektonicznych aplikacji back-end'owej jest REST API [2]. Jest to główny punkt komunikacyjny aplikacji działającej po stronie klienta (front-end) z naszym systemem. Na etapie projektowania warto zapoznać się ze standardem komunikacji oraz zbiorem reguł jakich należy się trzymać aby stworzony system był spójny ze standardem [11].

Poniżej w tabelach od 1 do 5 zostały przedstawione dokładne informacje odnośnie metod, jakie zostały udostępnione w ramach REST API.

Tabela 1: Wykaz metod do zarządzania kategoriami udostępnionych w ramach REST API

Metoda	Typ metody	Opis metody
<i>api/Categories</i>	GET	Metoda do pobierania kategorii
<i>api/Categories</i>	POST	Metoda do dodawania nowych kategorii
<i>api/Categories/Update</i>	POST	Metoda do aktualizacji kategorii
<i>api/Categories/Delete</i>	DELETE	Metoda do usuwania wcześniej zdefiniowanych kategorii

Tabela 2: Wykaz metod do zarządzania dokumentami udostępnionych w ramach REST API

Metoda	Typ metody	Opis metody
<i>api/Documents/get</i>	GET	Metoda do pobierania wszystkich dokumentów w kontekście użytkownika
<i>api/Documents/Get/id</i>	GET	Metoda do pobrania konkretnego dokumentu na podstawie wprowadzonego id dokumentu
<i>api/Documents/Create</i>	POST	Metoda do tworzenia nowego dokumentu finansowego
<i>api/Documents/Edit</i>	POST	Metoda do edycji dokumentu finansowego

<i>api/Documents/Delete/id</i>	DELETE	Metoda do usuwania wcześniej zdefiniowanego dokumentu finansowego. Jako parametr przyjmuje document id .
<i>api/Documents/Download/id</i>	GET	Metoda służąca do pobierania od zewnętrznego dostawcy dokumentu. Jako parametr przyjmuje document id .
<i>api/Documents/ProcessDocument</i>	POST	Metoda służąca do generycznego przetwarzania dokumentów za pomocą wtyczek.

Tabela 3: Wykaz metod do dodatkami uruchamianymi w kontekście użytkownika lub aplikacji udostępnionych w ramach REST API

Metoda	Typ metody	Opis metody
<i>api/Extensions/GetExtensions</i>	GET	Metoda do pobierania listy zaimplementowanych dodatków
<i>api/Extensions/Run</i>	POST	Metoda do uruchamiania wybranego dodatku.

Tabela 4: Wykaz metod używanych w procesie uwierzytelnienia udostępnionych w ramach REST API

Metoda	Typ metody	Opis metody
<i>api/oauth/providers</i>	GET	Metoda do pobierania listy dostawców, z których można korzystać podczas logowania do aplikacji oraz trzymania dokumentów finansowych
<i>/api/oauth</i>	GET	Metoda, która jest uruchamiana podczas zwrotnego żądania wysłanego od wybranego wcześniej dostawcy. Jest to jedna z metod, które biorą udział w procesie uwierzytelnienia standardu OAuth.
<i>/api/oauth/validateToken</i>	GET	Metoda pomocnicza służąca do walidacji ważności tokena.

Tabela 5: Wykaz metod do zarządzania profilem użytkownika udostępnionych w ramach REST API

Metoda	Typ metody	Opis metody
<i>api/Users/Details</i>	GET	Metoda do pobierania informacji o imieniu oraz nazwisku wprowadzonym w profilu użytkownika

<i>api/Users/Details/Save</i>	POST	Metoda do aktualizacji informacji o imieniu i nazwisku wprowadzonym w profilu użytkownika
<i>api/Users/ContactDetails</i>	GET	Metoda do pobierania informacji o generycznych danych kontaktowych zapisanych w profilu użytkownika.
<i>api/Users/ContactDetails/Save</i>	POST	Metoda do aktualizacji generycznych danych kontaktowych w kontekście profilu użytkownika.

Dodatkowo wysyłając żądanie do REST API należy dodać dodatkowy nagłówek *Authorization*, który powinien zawierać token otrzymany podczas procesu uwierzytelnienia. Dzięki temu REST API jest w stanie po jego odszyfrowaniu odczytać w kontekście jakiego użytkownika nastąpiło żądanie. Dodatkowo zalogowany użytkownik nie ma możliwości wykonania zapytania w kontekście innego konta, na które nie jest zalogowany.

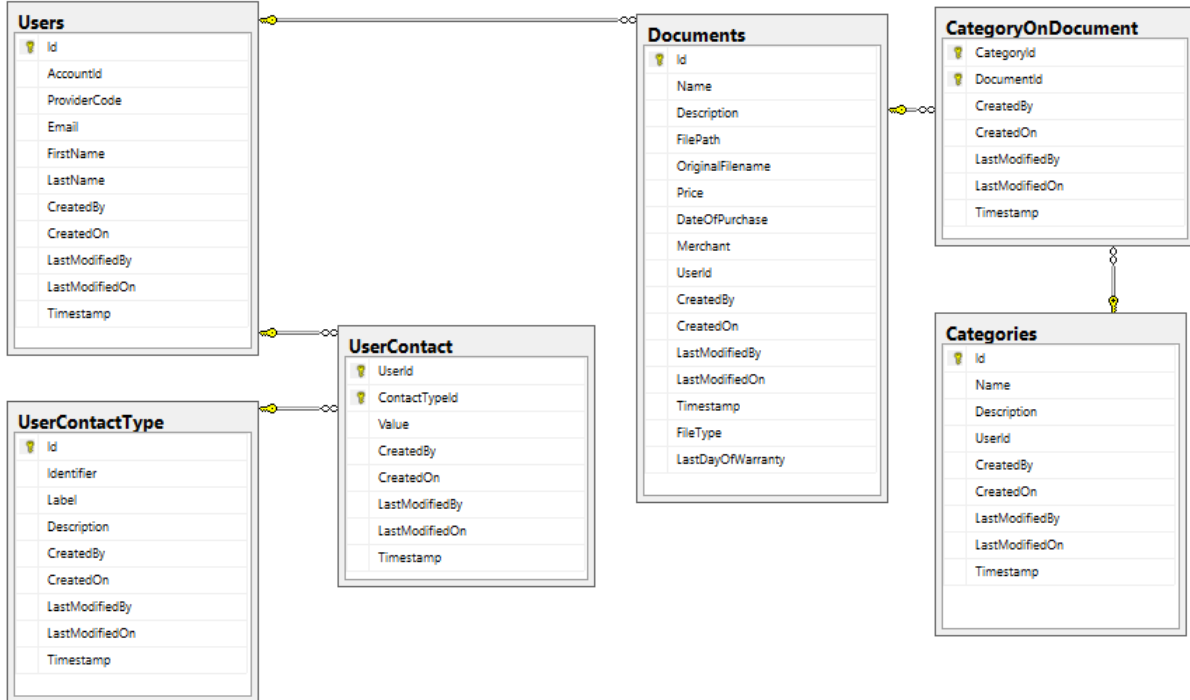
3.4.2 Architektura Bazy danych

Projektując każdą aplikację równie ważnym etapem jest zaprojektowanie wydajnego schematu bazy danych. Podczas projektowania należy zwrócić szczególną uwagę na wiele kwestii takich jak:

1. **Specyfikę aplikacji** - każda aplikacja posiada unikatową specyfikę działania. Niektóre operacje będą wykonywane częściej, inne natomiast bardzo rzadko. Na etapie projektowania bazy danych należy dokładnie przemyśleć tworzony schemat pod kątem późniejszego użycia.

2. **Możliwości rozbudowy modelu danych** - projektując model danych należy założyć, że aplikacja będzie rozbudowywana o nowej funkcjonalności w przyszłości. Model danych także nie powinien wprowadzać ograniczeń w tym zakresie. Szczególnie pomocne podczas analizy tego punktu mogą okazać się **postacie normalne bazy danych** [10].
3. **Wydajność bazy danych** - do tego punktu można zaliczyć zarówno kwestie wyboru odpowiedniego silnika bazy danych oraz jego wersji wraz z licencją. Od licencji często zależy jakie funkcjonalności optymalizacyjne zostały udostępnione. Dodatkowym czynnikiem, który może przyspieszyć działanie bazy danych jest nadanie odpowiednich indeksów w samej bazie danych. Nie można także pominąć maszyny, na której działa nasza baza wraz z parametrami łącza (jeśli mówimy o bazie znajdującej się fizycznie na innej maszynie niż główna aplikacja - back-end).

Na *rysunku 6* został zaprezentowany schemat bazy danych dla prototypu aplikacji do zarządzania dokumentami finansowymi.



Rysunek 6: Diagram struktury bazy danych aplikacji, Źródło: opracowanie własne

W przedstawionej strukturze bazy danych można wyróżnić następujące modele danych:

- **tabela *Users*** - zawiera informacje o użytkowniku.
- **tabela *UserContactType*** - zawiera informacje o typach danych kontaktowych, jakie mogą być uzupełnione przez każdego z użytkowników. Domyślnie aplikacja posiada zdefiniowane typy dla numeru telefonu oraz adresu email.
- **tabela *UserContact*** - tabela odpowiedzialna jest za trzymanie danych kontaktowych dla typów zdefiniowanych generycznie w tabeli *UserContactType*.
- **tabela *Documents*** - odpowiedzialna jest za trzymanie informacji o dokumentach zapisanych w systemie. Posiada także referencje do dostawcy, u którego trzymany jest oryginał dokumentu oraz nazwę pod jakim dokument występuje.

- **tabela *Categories*** - tabela zawiera informacje dotyczące kategorii zdefiniowanych w kontekście użytkownika.
- **tabela *CategoryOnDocument*** - tabela typu intersekcja. Zawiera przypisanie kategorii do dokumentu. Powstała w celu umożliwienia przypisania wielu kategorii do jednego dokumentu.

3.4.3 Architektura aplikacji front-end'owej

W ramach prototypu powstała także osobna aplikacja front-end'owa. Jej zadaniem jest prezentacja danych dla użytkowników w sposób graficznie czytelny. Aplikacja komunikuje się z częścią back-end'ową za pośrednictwem protokołu HTTP przy użyciu REST API.

Jako framework SPA⁸ do stworzenia tej warstwy aplikacji został użyty Angular 5.0. Koncept tworzenia aplikacji SPA posiada szereg zalet, a do głównych z nich można zaliczyć:

- separację warstwy prezentacji od części uruchamianej po stronie serwera,
- aplikacja jest ładowana tylko przy pierwszym wejściu użytkownika na stronę,
- nawigowanie po kolejnych ekranach aplikacji jest realizowane poprzez podmianę tylko części informacji prezentowanych na stronie.

⁸SPA - Single Page Application

4 Zastosowane technologie, narzędzia i standardy

Obecnie na rynku dostępnych jest bardzo wiele technologii i narzędzi pozwalających na wytwarzanie oprogramowania. Ich wybór jest ściśle powiązany z tym do czego aplikacja ma być używana oraz na jakich urządzeniach powinna być możliwa do uruchomienia. Dodatkowym czynnikiem, który powinien brać udział podczas decydowania się na konkretną technologię lub narzędzie jest pytanie, czy potrzebujemy dodatkowych licencji do jego użycia. Jest to istotny argument jeśli bierzemy pod uwagę dodatkowe koszty projektu. Biorąc pod uwagę wspomniane powyżej argumenty zdecydowałem się na technologie otwartoźródłowe oraz biblioteki programistyczne oferowane w ramach licencji MIT [12].

4.1 Użyte Technologie

Jednym z najpopularniejszych obecnie możliwości uruchomienia aplikacji jest użycie w tym celu przeglądarki internetowej. Ten sposób został wybrany także w przypadku aplikacji do zarządzania dokumentami finansowymi. Przeglądarka sieci web dostępna zarówno na komputerach klasy PC, tabletach oraz urządzeniach mobilnych takich jak smartphoney. Dokładne technologie wybrane do stworzenia prototypu zostały przedstawione w tym podrozdziale.

4.1.1 ASP.NET Core 2.0

ASP.NET Core 2.0 jest otwartoźródłowym frameworkiem do tworzenia między innymi aplikacji webowych. Środowiskiem uruchomieniowym dla aplikacji jest platforma .NET Core. Jest to rozwiązanie darmowe i działa zarówno na platformie Windows, Linux oraz MacOS [13].

Wybrany framework pozwala na tworzenie aplikacji w językach C#, F# lub VisualBasic. W przypadku prototypu aplikacji do zarządzania dokumentami finansowymi został użyty język C#.

Dodatkową przewagą nowego środowiska uruchomieniowego stworzonego przez firmę Microsoft w porównaniu do wcześniejszej wersji .NET Framework jest dużo większa modularność bibliotek/pakietów, z którym programista może skorzystać podczas tworzenia aplikacji. Znaczącej poprawie uległa także szybkość działania aplikacji napisanych z użyciem nowej wersji frameworka.

Aplikacje stworzone z użyciem ASP.NET Core mogą być uruchamiane zarówno na dedykowanym serwerze Microsoftu - IIS, ale także na systemach linux (np. używając serwera Apache lub nginx) lub też jako self-hosting⁹. Możliwe jest także uruchomienie jako Windows Service, jednak ta opcja dostępna jest jedynie na platformie systemów Windows. Minimalną wersją systemu wspierającą tą metodę jest Windows 7.

Wszystkie powyższe argumenty sprawiają, że wybrana technologia idealnie wpasowuje się do projektu aplikacji, która z założenia ma posiadać liczne możliwości jej późniejszego wdrożenia, a następnie działania.

4.1.2 SQL Server 2017 Express

SQL Server w wersji Express [14] jest darmową relacyjną bazą danych stworzoną na potrzeby aplikacji, której wielkość bazy danych nie przekracza 10GB. W pełni wspiera transakcje, posiada możliwość optymalizacji np. poprzez nadawanie indeksów. W każdym momencie daje także możliwość przejścia na wyższe, płatne wersje serwera SQL Server.

4.1.3 Angular 5.0 i TypeScript

Aplikacja front-end'owa działająca po stronie przeglądarki została stworzona przy użyciu frameworka Angular 5.0 [3] jako SPA¹⁰. Kod aplikacji został napisany w języku TypeScript [4]. Przewagą TypeScriptu jest jego silne typowanie oraz dodanie możliwości korzystania z klas oraz typów. Dzięki temu już na etapie wstępnej kompilacji programista jest w stanie

⁹aplikacja będzie uruchomiona jako aplikacja działająca w tle i udostępniona na wybranym porcie

¹⁰Single Page Application

dowiedzieć się o ewentualnych problemach związanych z niezgodnością typów danych. Kod napisany w języku TypeScript jest kompilowany do języka JavaScript.

Projekty stworzone za pomocą frameworka Angular posiadają własną strukturę organizacyjną plików w projekcie. Dzięki temu korzystanie z zewnętrznych bibliotek tworzonych przez społeczność i udostępnionych w ramach licencji MIT jest o wiele prostrze do zaimplementowania niezależnie od projektu w którym pracujemy. Angular jest obecnie jednym z trzech¹¹ najbardziej popularnych frameworków służących do tworzenia zarówno aplikacji webowych jak i hybrydowych aplikacji mobilnych. Dodatkowo proces programowania z użyciem frameworka ułatwia szeroka społeczność użytkowników oraz liczne artykuły pozwalające w łatwy sposób rozpocząć implementację nowego projektu.

4.1.4 Entity Framework Core

Entity Framework Core [16] jest darmowym otwartoźródłowym frameworkiem, służącym do mapowania modeli stworzonych w aplikacji serwerowej na tabele w relacyjnej bazie danych. Dzięki temu, możemy w łatwy sposób, także korzystając z wyrażeń LinQ¹² budować zaawansowane zapytania do bazy danych, dodawać nowe dane, a także je modyfikować i usuwać. Biblioteka została użyta po stronie aplikacji serwerowej.

4.1.5 AutoMapper

AutoMapper [18] to niewielka biblioteka pozwalająca na mapowanie modeli zgodnie z wcześniej ustalonymi oraz zaprogramowanymi regułami. Dzięki tej bibliotece unikamy wielu linii kodu związanych tylko z przypisywaniem danych pomiędzy polami. Biblioteka została użyta po stronie aplikacji serwerowej.

¹¹Równie popularne są obecnie React oraz Vue.js [15]

¹²Language Integrated Query [17]

4.1.6 Flurl

Flurl [19] jest biblioteką służącą do budowania zapytań HTTP w sposób dużo bardziej czytelny, niż w przypadku standardowego użycia klasy *HttpClient* dostępnej w przypadku frameworka .NET. Jego dodatkową zaletą jest możliwość testowania stworzonych zapytań. Biblioteka została użyta po stronie aplikacji serwerowej.

4.2 Użyte narzędzia

Podczas implementacji prototypu zostały użyte liczne narzędzia programistyczne. Dzięki wykorzystaniu ich w pracy udało się zredukować czas potrzebny na stworzenie prototypu aplikacji. Poniżej zostały opisane najważniejsze w nich.

4.2.1 Visual Studio 2017 Community

Visual Studio jest jednym z najbardziej popularnych IDE¹³ służące do tworzenia aplikacji z wykorzystaniem języka C# na platformę .NET Core. Wersja Community jest wersją w pełni darmową także dla zastosowań komercyjnych. Limity, które zostały przewidziane przez producenta są na tyle wysokie, że prototyp aplikacji do zarządzania dokumentami finansowymi może w pełni korzystać z tego oprogramowania.

Visual Studio 2017 zostało użyte w projekcie podczas implementacji serwerowej części aplikacji (back-end'u). Decyzję o użyciu tego IDE została poparta bogatym wsparciem dla konfiguracji projektów wykorzystujących platformę .Net Core.

4.2.2 ReSharper

ReSharper [20] jest dodatkiem do środowiska programistycznego Visual Studio 2017 stworzonym przez firmę JetBrains. Posiada liczne funkcjonalności, które w znaczący sposób poprawiają jakość pracy nad kodem. Pozwala automatyzować często powtarzające się

¹³Integrated Development Environment - z ang. Zintegrowane Środowisko Programistyczne

akcje, przez co programista może skupić się na tworzeniu nowych funkcjonalności, zamiast bezpośrednio nad jego pisaniem oraz organizacją. ReSharper oferuje darmową licencję dla studentów, dzięki czemu mógł zostać użyty w projekcie.

4.2.3 Visual Studio Code

Visual Studio Code [21] jest w pełni darmowym otwartoźródłowym edytorem stworzonym także przez firmę Microsoft. Działa na systemach Windows, Linux oraz macOS. Edytor został zaprezentowany w roku 2015 i od tego czasu jego popularność stale rośnie, wraz z liczbą wtyczek pozwalających rozszerzać jego możliwości. W bieżącym projekcie edytor był wykorzystywany do implementacji części aplikacji działającej po stronie klienta, napisanej z użyciem języka TypeScript. Decyzja o użyciu tego edytora została uzasadniona szybkością jego działania oraz bogatym wsparciem dla wykorzystywanego w projekcie języka programowania.

4.2.4 Manager paczek NuGet

Manager paczek NuGet [22] pozwala na dużo prostsze zarządzanie zależnościami w projekcie. Wszystkie paczki przez niego zarządzane posiadają wersje. Sprawia to, że niezależnie od maszyny na której uruchamiany jest projekt, zostanie on zbudowany w dokładnie ten sam sposób, przy użyciu dokładnie tych samych wersji zależności. Użycie managera paczek zwalnia także programistę z obowiązku ręcznego dodawania plików zależności do projektu, a w późniejszym etapie dbania o spójność pomiędzy użytymi bibliotekami zewnętrznymi. Informacje o tym jakie paczki zostały użyte w projekcie znajdują się w pliku projektu **.csproj* w sekcji *PackageReference*.

4.2.5 Manager paczek npm

Manager paczek npm [23] jest odpowiednikiem managera paczek NuGet. Różnica polega na rodzaju paczek, jakie są przez niego zarządzane. W przypadku npm są to paczki napisane

z użyciem języka JavaScript lub pochodnych np. TypeScript. Odbiorcami paczek są zatem głównie aplikacje webowe. Do zarządzania paczkami w projekcie służy terminal. Informacje o tym jakie paczki zostały użyte oraz w jakiej wersji znajdują się w pliku *package.json* w głównym katalogu projektu.

4.2.6 Google Cloud Vision API

Usługa Google Cloud Vision [28] pozwala na skorzystanie z licznych algorytmów służących do analizy obrazów. Jedną z możliwości dostępnej dla programistów jest rozpoznawanie tekstu na podstawie przesłanych zdjęć. Serwis posiada także wiele innych funkcjonalności takich jak kategoryzowanie zdjęć, analize pod kątem treści zamieszczonych na zdjęciach lub analize kolorów zdjęcia. Podczas tworzenia prototypu usługa świadczona przez firmę Google została użyta w celu zaimplementowania funkcjonalności automatycznego rozpoznawania danych umieszczonych na dokumencie finansowym.

4.2.7 Biblioteka ng-swagger-gen

Biblioteka ng-swagger-gen [24] została użyta w projekcie w celu generowania klienta do komunikacji pomiędzy aplikacją działającą po stronie klienta (*front-end*), a aplikacją serwerową (*back-end*). Klient jest generowany na podstawie informacji otrzymanych z narzędzia *Swagger*. Biblioteka jest darmowa w użyciu, a jej instalacja odbywa się poprzez menedżera paczek npm. Dzięki jej zastosowaniu programista unika konieczności ręcznego generowania, a w późniejszym czasie aktualizowania kontraktu do komunikacji pomiędzy aplikacjami. Wykluczony zostaje także element pomyłki podczas ręcznej implementacji. Biblioteka ta jest szczególnie polecana w przypadku większej liczby programistów, którzy równolegle rozwijają zarówno część serwerową oraz aplikację działającą po stronie klienta.

4.2.8 Swagger

Swagger [25] jest otwartym standardem służącym do opisu REST API. Pozwala prezentować informacje zarówno w sposób zrozumiały dla ludzi oraz dla innych bibliotek np. generatorów kodu. Przykładem biblioteki użytej w projekcie, która wykorzystuje informacje udostępnione przez Swaggera jest *ng-swagger-gen*. Swagger udostępnia także wygodny interfejs dostępny przez przeglądarkę dla programisty. Pozwala on dokładnie zapoznać się z metodami, które są udostępnione przez API. Dodatkowo informacje wzbogacone są o modele, które biorą udział w komunikacji.

4.2.9 Narzędzie do kontroli wersji Git

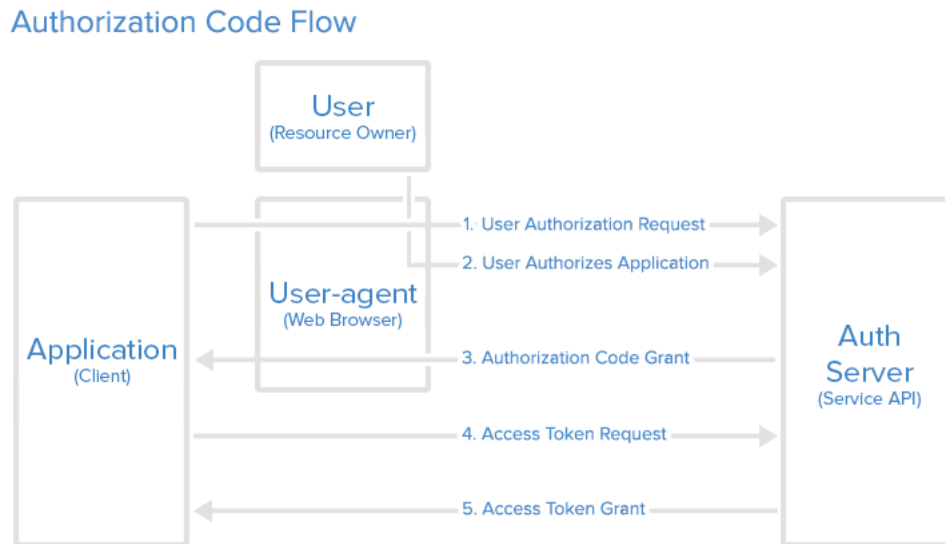
Obecnie korzystanie z kontroli wersji w przypadku projektów informatycznych stało się standardem. Jednym z najbardziej popularnych systemów kontroli wersji jest *GIT* [5]. Pozwala on w bardzo wygodny sposób zarządzać wersjami kodu, śledzić zmiany dokonane na plikach oraz ich autorów. Repozytoria (projekty) utworzone i zarządzane przez system GIT są rozproszone. Oznacza to, że kod programu trzymany jest na każdym z komputerów, na którym wytwarzane jest oprogramowanie oraz dodatkowo na zdalnym repozytorium. Całość sprawia, że kooperacja pracy pomiędzy programistami może być dużo wygodniejsza, a możliwość utraty jakiejś części pracy zniwelowana do minimum.

W bieżącym projekcie system kontroli wersji *Git* został użyty w celu zabezpieczenia kodu aplikacji przez ewentualną awarię maszyny, na której aplikacja powstawała. Dodatkowo wykonywanie małych *commitów*¹⁴ sprawiało, że w każdym momencie była możliwość powrotu do kodu z dowolnego implementacji prototypu.

¹⁴commit - zapis do repozytorium bieżącego efektu swojej pracy

4.3 Standard OAuth 2.0

W prototypie aplikacji został wykorzystany standard OAuth 2.0. Jest to otwarty protokół, który pozwala na tworzenie mechanizmów uwierzytelnienia. Zapewnia on możliwość korzystania obsługi zewnętrznych dostawców z wykorzystaniem konta użytkownika na którego odbyło się uwierzytelnienie.



Rysunek 7: Diagram przedstawiający proces korzystania ze standardu OAuth 2.0, Źródło: digitalocean.com [26]

Na podstawie diagramu (rysunek 7) można wyszczególnić konkretne akcje, które należy wykonać aby prawidłowo wykonać proces uwierzytelnienia użytkownika:

1. Użytkownik odwiedza stronę aplikacji, która korzysta ze standardu uwierzytelnienia OAuth 2.0.
2. Po wybraniu opcji zalogowania zostaje przeniesiony na stronę zewnętrznego dostawcy, gdzie wprowadza swoje dane logowania i akceptuje niezbędne zgody do wykorzystania swojego konta przez inne aplikacje.
3. Po poprawnym zalogowaniu następuje przekierowanie na stronę aplikacji, którą pierwotnie odwiedził użytkownik. W adresie powrotnym zostaje umieszczona informacja o

statusie logowania wraz z tymczasowym kodem, służącym do późniejszego wygenerowania tokenu.

4. Aplikacja po stronie serwerowej używając tymczasowego kodu otrzymanego w adresie zwrotnym oraz sekretne go kodu (który zostaje nadany w momencie rejestracji aplikacji u zewnętrznego dostawcy), wysyła żądanie wygenerowania tokenu.
5. Wygenerowany token jest poświadczeniem prawidłowego uwierzytelnienia użytkownika. Może być także używany w kontekście komunikacji z wybranym dostawcą np. w celu zarządzania plikami dokumentów.

Standard OAuth posiada także wiele dodatkowych możliwości, w tym także definiowania ról w aplikacjach. W przypadku prototypu systemu do zarządzania dokumentami finansowymi nie jest konieczne jego użycie, ze względu na specyfikę aplikacji.

5 Prototyp nowego rozwiązania

Głównym efektem pracy jest prototyp elastycznego systemu do zarządzania dokumentami finansowymi. W bieżącym rozdziale zostały opisane najważniejsze rozwiązania implementacyjne, które zostały zaprojektowane w ramach pracy. Dodatkowo w kolejnym podrozdziale zostały zaprezentowane przykłady użycia aplikacji wraz z dokładnym opisem jej działania oraz możliwości.

5.1 Istotne rozwiązania implementacyjne

Podczas implementacji prototypu zostały zaprojektowane liczne rozwiązania programistyczne, które umożliwiły powstanie generycznych części systemu. Proces rozszerzania funkcjonalności aplikacji o nowe moduły został dokładnie przemyślany. W bieżącym podrozdziale zostały opisane najważniejsze z elementów systemu.

5.1.1 Generyczny moduł do obsługi logowania

W aplikacji został zaimplementowany generyczny moduł pozwalający na użycie zewnętrznych dostawców kont użytkowników wspierających standard OAuth 2.0. Dzięki temu możliwe jest skorzystanie z usług takich dostawców jak np. Google, Dropbox, OneDrive i innych.

W bieżącej wersji systemu **została zaimplementowana obsługa dwóch dostawców** - *Dropbox* i *Google Drive*.

Mechanizm generycznych modułów opiera się na użyciu kontenera IoC¹⁵, który skanuje kod projektu *FinanceManager.SPA* w celu znalezienia klas, które implementują interfejs *IOAuthProvider*. Wszystkie klasy implementujące wspomniany wcześniej interfejs są automatycznie rejestrowane i użyte w projekcie. Kod interfejsu, który należy zaimplementować w celu dodania kolejnego dostawcy znajduje się na listingu 1.

¹⁵Inversion of Control [29]

Listing 1: Interfejs *IOAuthProvider*

```
public interface IOAuthProvider
{
    string ProviderCode { get; }
    string RedirectUrl { get; }
    string PrividerLabel { get; }
    string FontAwesomeIcon { get; }
    Task<object> GetToken(string code);
    Task<FileModel> GetFile(string token, string fileName);
    Task<UploadFileResponse> UploadFile(string token, FileModel file);
    Task DeleteFile(string token, string fileName);
}
```

Interfejs został zaprojektowany w taki sposób, aby jego implementacja w pełni umożliwiała skorzystanie z niego także po stronie aplikacji klienckiej (*front-end'u*). Stąd też pojawiły się na nim takie pola jak:

- *ProviderLabel* - jest to nazwa dostawcy wyświetlana w aplikacji klienckiej,
- *FontAwesomeIcon* - ikona, która wyświetlana jest obok nazwy w aplikacji klienckiej,
- *ProviderCode* - unikalny kod dostawcy,
- *RedirectUrl* - adres na który należy przekierować użytkownika w celu wykonania uwierzytelnienia przez zewnętrznego dostawcę.

Dodatkowo interfejs posiada także metody, które są odpowiedzialne za komunikację z dostawcą w kontekście trzymanyh tam plików dokumentów oraz pobierania tokena:

- metoda *GetToken* - metoda pozwala na pobranie tokena na podstawie tymczasowego kodu, który został wygenerowany podczas procesu uwierzytelnienia użytkownika.
- metoda *GetFile* - służy do pobierania pliku od dostawcy.
- metoda *UploadFile* - służy do wysyłania pliku do dostawcy.
- metoda *DeleteFile* - służy do usuwania pliku u dostawcy.

Za skanowanie kodu w poszukiwaniu klas, które implementują konkretny interfejs odpowiada biblioteka *Scrutor* [27]. Dodatkowo w całym procesie implementacji modułów został użyty wbudowany w ASP.NET Core system DI¹⁶. Dzięki zastosowaniu obu mechanizmów możliwe jest najpierw skanowanie, a następnie zarejestrowanie wszystkich istniejących klas w projekcie. Przykład implementacji mechanizmu rejestrującego klasy implementujące interfejs *IOAuthProvider* znajduje się na listingu 2.

Listing 2: Przykład konfiguracji skanowania klas implementujących interfejs *IOAuthProvider*

```
public static class OAuthProvidersConfiguration
{
    public static void ConfigureOAuthProviders(this IServiceCollection
        services)
    {
        services.AddTransient<IOAuthResolverService,
            OAuthResolverService>();

        services.Scan(s => s.FromApplicationDependencies()
            .AddClasses(c => c.AssignableTo<IOAuthProvider>())
            .AsImplementedInterfaces()
            .WithTransientLifetime());
    }
}
```

Następnie w celu pobrania wszystkich zarejestrowanych klas implementujących konkretny interfejs zostaje użyty *IServiceProvider*, który jest częścią frameworka i znajduje się w przestrzeni nazw *System*. Jego użycie ogranicza się do wstrzyknięcia go do Serwisu, który odpowiedzialny jest za pobieranie wszystkich zarejestrowanych dostawców. Przykład serwisu znajduje się na listingu 3.

Listing 3: Przykład serwisu odpowiedzialnego za pobieranie wcześniej zarejestrowanych generycznych modułów do obsługi logowania

```
public class OAuthResolverService : IOAuthResolverService
{
    private readonly IServiceProvider _serviceProvider;
```

¹⁶Dependency Injection - z ang. Wstrzykiwanie zależności

```

public OAuthResolverService(IServiceProvider serviceProvider)
{
    _serviceProvider = serviceProvider;
}

public IEnumerable<IOAuthProvider> GetAllProviders()
{
    return _serviceProvider.GetServices<IOAuthProvider>();
}

public IOAuthProvider GetProvider(string providerName)
{
    return GetAllProviders().Single(x => x.ProviderCode ==
        providerName);
}
}

```

OAuthResolverService posiada dwie metody. Pierwsza z nich jest używana w celu pobrania wszystkich zarejestrowanych dostawców w prototypie. Druga jest używana, kiedy wiadomo już z którego dostawcy skorzystał użytkownik. W takim przypadku można pobrać go za pomocą jego unikalnej nazwy.

5.1.2 Generyczny moduł do obsługi przetwarzania dokumentów

Kolejnym z generycznych rozwiązań, które zostały zaimplementowane w pracy jest moduł do przetwarzania dokumentów w trakcie ich dodawania. Analogicznie jak w przypadku modułu do uwierzytelnienia oraz trzymania plików u zewnętrznych dostawców, aby napisać własny dodatek do przetwarzania dokumentów należy zaimplementować interfejs - w tym przypadku *IDocumentProcessor*.

Definicja interfejsu jest stosunkowo prosta (została zaprezentowana na *listingu 4*). ogranicza się do jednej metody *ProcessDocument*, która jest uruchamiana w przypadku chęci przetworzenia dokumentu przez użytkownika.

Listing 4: Interfejs *IDocumentProcessor*

```

public interface IDocumentProcessor

```

```
{  
    Task<Document> ProcessDocument(Document document);  
}
```

Metoda *ProcessDocument* jako argument przyjmuje model, który jest całym dokumentem w rozumieniu biznesowym. Zawiera wszystkie pola, które są dostępne dla użytkownika po stronie aplikacji front-end'owej. Metoda jest uruchamiana asynchronicznie, a jej rezultatem jest ten sam model, który został podany jako parametr wejściowy do metody. Całość sprawia, że możliwe jest jego uzupełnienie w trakcie przetwarzania oraz przekazanie do kolejnego dodatku, który także może dokonać modyfikacji na zbiorze danych dokumentu.

5.1.3 Moduł realizujący OCR dokumentu

W prototypie został zaimplementowany moduł realizujący funkcjonalność OCR¹⁷ dokumentów dodawanych w formacie *image/**. Jest to przykład modułu implementujący interfejs *IDocumentProcessor*, który został dokładnie opisany w poprzednim podrozdziale. Dzięki niemu po dodaniu dokumentu użytkownik nie musi samodzielnie uzupełniać wszystkich pól dokumentu. Część z nich zostaje uzupełniona na podstawie dokumentu, który został załączony.

Do rozpoznawania tekstu z obrazów został użyty silnik udostępniony przez firmę Google - Cloud Vision [28]. Pozwala on na rozpoznanie wielu aspektów przesłanych obrazów - w tym także na wykonanie bardzo dokładnego OCR'u dokumentu.

Po przesłaniu dokumentu przedstawionego do rozpoznania otrzymujemy odpowiedź w formacie *json*. Jednym z pól z otrzymanego wyniku jest cały tekst jaki udało się rozpoznać z dokumentu finansowego.

Kolejnym etapem, który został zaimplementowany w pracy było wyciągnięcie z odczytanego przez firmę Google tekstu dokładnych informacji na temat:

- daty dokonanego zakupu,

¹⁷Optical character recognition - z ang. Optyczne rozpoznawanie znaków

- całkowitej kwoty zakupu,
- sklepu/punktu w jakim dokonany był zakup.

Rozpoznawanie wyżej wymienionych informacji z surowego tekstu dokumentu zostało zrealizowane głównie z wykorzystaniem wyrażeń regularnych. Dzięki nim możliwe jest znalezienie wszystkich dat występujących w dokumencie, kwot oraz nazwy sprzedawcy. Przykład metody, która odpowiedzialna jest za pobranie całkowitej kwoty na podstawie odczytanego wcześniej tekstu została przedstawiona na listingu 5.

Listing 5: Metoda odpowiedzialna za pobranie całkowitej kwoty z dokumentu finansowego.

```
private decimal GetTotalPrice(string text)
{
    List<decimal> allPrices = new List<decimal>();

    Regex moneyRegex = new Regex(@"\s(\d+) ([\.,,]{1}) (\d{1,2})");

    MatchCollection prices = moneyRegex.Matches(text);
    if (prices.Cast<Match>().Any())
    {
        foreach (var price in prices)
        {
            decimal tempPrice;
            var priceInCorrectFormat =
                price.ToString().Replace(",", ".");
            var result = Decimal.TryParse(priceInCorrectFormat,
                out tempPrice);
            if (result)
            {
                allPrices.Add(tempPrice);
            }
        }

        return allPrices.Max();
    }

    return 0;
}
```


Podczas implementacji zostało przyjęte założenie, że kwota całkowita jest najwyższą kwotą, która znalazła się na dokumencie finansowym. Dzięki temu możliwe jest znalezienie wszystkich kwot występujących w odczytanym tekście, a w późniejszym etapie wybranie tej z najwyższą wartością.

Metoda odpowiedzialna za rozpoznanie nazwy sprzedawcy bazuje na założeniu, że nazwa sprzedawcy znajduje się przed jego adresem. Adresy posiadają zazwyczaj stały format (zawierają np. informację o ulicy przy której mieści się punkt sprzedaży) więc ich znalezienie również nie stanowi problemu. W przypadku wyszukiwania adresu dodatkowym ważnym założeniem był fakt, że adres sprzedawcy zazwyczaj znajduje się na samym początku dokumentu finansowego. Dzięki temu udało się zaimplementować metodę, która z bardzo dużą dokładnością zwraca nazwę sprzedawcy na podstawie całego rozpoznanego tekstu dokumentu.

Ostatnim etapem była implementacja metody do rozpoznawania daty sprzedaży. Tak samo jak w powyższych przykładach metod, do rozpoznania użyto wyrażenia regularnego. Daty znajdujące się na dokumentach zazwyczaj przedstawiane są w formie kilku znanych formatów. Po znalezieniu ciągu znaków, który może odpowiadać dacie sprzedaży następuje próba jej konwersji na obiekt *DateTime*. Jeśli próba się powiedzie, można z dużym prawdopodobieństwem uznać, że data jest prawidłowa.

Dodatkowo zaprojektowany moduł posiada założenie, że w przypadku braku możliwości znalezienia którejkolwiek z informacji na dokumencie finansowym takie pole jest pomijane. Dzięki temu nawet w przypadku bardzo niestandardowych dokumentów załączanych w systemie, możemy maksymalnie wykorzystać funkcjonalność rozpoznawania informacji za pomocą systemu OCR.


Cały proces został przedstawiony na przykładzie dokumentu finansowego przedstawionego na rysunku 8:

1. Dokument finansowy zostaje przesłany do Google Vision Api w celu rozpoznania tekstu, który się na nim znajduje.

PETROSTAR
 SKLEP NR. 3 WARSZAWA BIELANY
 UL. Cegkowska 19
 01-803 Warszawa
 www.petrostar.pl
 sklep3@petrostar.pl
 TEL. 22 864 46 43
 NIP PL 113-199-05-35

2017-06-30 56203

PARAGON FISKALNY
 BOSCH 3 397 007 187 A1875 600/450
 1szt. x91,50 91,50A

SPRZEDAZ OPODATKOWANA A	91,50
PTU A 23,00 %	17,11
SUMA PTU	17,11
SUMA PLN	91,50
00066 #001 KIEROWNIK	15:20
6CB3E30B6D73E89E2BB8CCD1F1BA35B6D649DB70	
 BOS 12165282	

Karta 91,50 PLN

Rysunek 8: Przykładowy dokument finansowy użyty do rozpoznania przez autorski moduł OCR, Źródło: opracowanie własne

2. Aplikacja back-end otrzymuje odpowiedź przedstawioną na listingu 6 zawierającą surowy rozpoznany tekst.

Listing 6: Surowy tekst pozpoznanany przez Google Vision Api na podstawie dokumentu finansowego z rysunku 8.

```
PETROSTAR SKLEP NR. 3 WARSZAWA BIELANY UL. Cegkowska 19 01-803
Warszawa\www.petrostar.pl sklep3@petrostar.pl TEL: 22 864 46 43
NIP PL 113-199-05-35 2017-06-30 56203 PARAGON FISKALNY BOSCH 3 397
007 187 A1875 600/450 1szt. x91,50 91,50A -----
SPRZEDAZ OPODATKOWANA A 91,50 PTU A 23,00 %17,11 SUMA PTU SUMA PLN
00066 #001 KIEROWNIK 15:20 6CB3E30B6073E89E2BB8CCD1F1BA35B606490870
P B05 12165282 17,11 91,50 Karta 91,50 PLN
```

3. Z całego rozpoznanego tekstu zostają oznaczone takie informacje jak *cena*, *nazwa sklepu*, *data zakupu*. Oznaczenie następuje z użyciem autorskich algorytmów.
4. Model dokumentu zostaje uzupełniony o informacje oznaczone we wcześniejszym punk-

cie i odesłany w formie odpowiedzi do aplikacji front-end'owej w formacie json - listing 7. Widać, że zostały uzupełnione takie pola jak:

- *price* - cena,
- *merchant* - nazwa sklepu,
- *dateOfPurchase* - data zakupu

Pozostałe pola posiadają wartość *null*¹⁸, ponieważ nie zostały uzupełnione przez użytkownika w aplikacji front-end. Pola dotyczące samego pliku zostają uzupełnione automatycznie w momencie wyboru dokumentu finansowego.

Listing 7: Odpowiedź w formacie json z aplikacji back-end'owej na żądanie przetworzenia dokumentu finansowego.

```
{
  "id": 0,
  "name": null,
  "description": null,
  "price": 91.50,
  "merchant": "petrostar sklep nr. 3 warszawa bielany",
  "dateOfPurchase": "2017-06-30T00:00:00",
  "lastDayOfWarranty": null,
  "fileName": "paragon.png",
  "originalFilename": "paragon.png",
  "fileType": "image/png",
  "categories": null
}
```


5. Formularz dodawania nowych dokumentów finansowych w aplikacji front-end zostaje uzupełniony o dane rozpoznane przez autorski moduł OCR działający po stronie aplikacji back-end. Przykład na rysunku 9.

Na podstawie powyższego przykładu widać, że autorski moduł OCR w przypadku dokumentu przedstawionego z rysunku 8 zadziałał ze 100% dokładnością. Wszystkie informacje zostały prawidłowo znalezione i zostały przekazane do formularza w aplikacji front-end.

¹⁸null - wartość nie określona, pusta

+ Dodawanie nowego dokumentu

paragon.png



↑ Zmień dokument

👁 Przetwórz dokument

Nazwa

Opis

Sprzedawca

Cena

Data zakupu

Data końca gwarancji

Kategorie

Wybierz kategorię

Rysunek 9: Formularz dodawania nowego dokumentu finansowego z rozpoznanymi informacjami z dokumentu finansowego z rysunku 8. Źródło: opracowanie własne

5.1.4 Generyczny moduł dodatków uruchamianych przez użytkownika

Kolejnym generycznym modułem, który sprawia, że cały system jest elastyczny jest możliwość implementacji dodatków wspomagających zarządzanie dokumentami znajdującymi się na koncie użytkownika lub też stanem całej aplikacji. Dodatki są uruchamiane przez użytkownika na żądanie z poziomu aplikacji front-end'owej. Możliwości ingerencji dodatków w system są bardzo szerokie.

Aby utworzyć nowy dodatek należy zaimplementować interfejs *IFireAndForgetExtension*, ktorego definicja została przedstawiona na listingu 8.

Listing 8: Kod interfejsu *IFireAndForgetExtension*

```
public interface IFireAndForgetExtension
{
    string Name { get; }
    string Label { get; }
```

```
    string Description { get; }
    string FontAwesomeIconName { get; }
    Task PerformAction(int userId);
}
```

Główną metodą dodatku jest *PerformAction*. Każdy z dodatków posiada też kilka pól, które pomagają na poprawne wyświetlenie dodatku po stronie aplikacji front-end:

- pole *Name* - jest to unikalna nazwa dodatku,
- pole *Label* - jest to nazwa dodatku wyświetlana w aplikacji front-end,
- pole *Description* - jest opisem dodatku wyświetlanym w aplikacji front-end,
- pole *FontAwesomeIcon* - jest nazwą ikony dodatku, która także jest wyświetlana w aplikacji front-end.

Zastosowanie wyżej wymienionych pól pozwala wykluczyć konieczność ręcznego rejestrowania dodatków w aplikacji działającej po stronie użytkownika (front-end). Wszystkie informacje o zarejestrowanych dodatkach pobierane są z aplikacji działającej po stronie serwera (back-end).

Mechanizm tego typu dodatków bazuje na założeniu, że każdy z nich jest wywoływany asynchronicznie. Dodatki nie zwracają więc rezultatu swojego wykonania, a jedynie działają na danych zgromadzonych już w systemie. Parametr *userId* jest przekazywany do dodatku za każdym razem, kiedy wywoływana jest metoda *PerformAction*. Dzięki temu to dodatek decyduje, czy akcja, którą implementuje zostanie wykonana w kontekście użytkownika, czy też może całej aplikacji. Daje to duże możliwości podczas planowania nowych funkcjonalności.

5.1.5 Dodatki zaimplementowane w prototypie jako dodatki uruchamiane przez użytkownika

W ramach pracy nad prototypem zostały zaimplementowane trzy dodatki bazujące na interfejsie *IFireAndForgetExtension* - który został opisany w poprzednim podrozdziale:

1. **Dodatek do usuwania dokumentów, które nie posiadają już wsparcia gwarancyjnego** - implementacja dodatku została umieszczona w projekcie *FireAndForgetExtension.RemoveAfterWarranty*. Dodatek ma zadanie usunąć wszystkie dokumenty, dla których wsparcie gwarancyjne oferowane przez sprzedawcę zostało już zakończone. Pozwala to w prosty sposób dokonać redukcji dokumentów przetrzymywanych w systemie w kontekście użytkownika. Z założenia dokumenty te nie będą dłużej istotne dla tego użytkownika.
2. **Dodatek służący do tworzenia kopii bezpieczeństwa wszystkich dokumentów na infrastrukturze zewnętrznego dostawcy** - kopia bezpieczeństwa tworzona przez dodatek jest w formacie *json*. Zawiera wszystkie informacje o dokumentach trzymany w systemie (w bazie danych) w raz z kategoriami do których zostały przydzielone. Dzięki tej opcji użytkownik nie tylko ma stały dostęp do samych dokumentów, które są trzymane po stronie zewnętrznego dostawcy, ale także w każdej chwili może wykonać kopię danych trzymany po stronie aplikacji i zapisać je równoległe z samymi dokumentami. Implementacja dodatku została umieszczona w projekcie *FireAndForgetExtension.BackupDocuments*. Przykład implementacji dodatku został umieszczony na listingu 9.

Listing 9: Kod dodatku służącego do tworzenia kopii bezpieczeństwa wszystkich dokumentów trzymany w programie.

```
public class BackupAllDocumentsExtension : IFireAndForgetExtension
{
    private readonly IDocumentsService _documentsService;
    private readonly IOAuthResolverService _authResolverService;
    private readonly IIdentityProvider _identityProvider;
    private readonly IOAuthProvider _authProvider;

    public BackupAllDocumentsExtension(IDocumentsService
        documentsService,
        IOAuthResolverService authResolverService,
        IIdentityProvider identityProvider)
    {
        _documentsService = documentsService;
        _authResolverService = authResolverService;
    }
}
```

```

        _identityProvider = identityProvider;
        var providerName =
            _identityProvider.ProviderName.ValueOrDefault();
        _authProvider =
            _authResolverService.GetProvider(providerName);
    }

    public string Name => "BackupDocuments";
    public string Label => "Utworz kopie dokumentow";
    public string Description => "Dodatek sluzy do utworzenie kopii
        wszystkich szczegolow dokumentow w chmurze w formacie json.";
    public string FontAwesomeIconName => "fa fa-cloud-upload";

    public async Task PerformAction(int userId)
    {
        IEnumerable<Document> documents = await
            _documentsService.GetDocuments(new
                DocumentsSearchQuery(), userId);

        string backup = JsonConvert.SerializeObject(documents);
        var backupAsByteArray =
            System.Text.Encoding.UTF8.GetBytes(backup);
        var token = _identityProvider.Token.ValueOrDefault();
        await _authProvider
            .UploadFile(token, new FileModel()
            {
                Name =
                    $"backup-{DateTime.UtcNow.ToFileTimeUtc()}.json",
                Data = backupAsByteArray
            });
    }
}

```

3. Dodatek służący do tworzenia raportu wydatków z podziałem na kategorie - ten typ raportu może być wykorzystywany do śledzenia kosztów jakie są przeznaczone na poszczególne kategorie. Raport tworzony jest w formacie *csv*. Dzięki temu można w łatwy sposób dokonać jego późniejszej analizy np. w programie **Microsoft Excel**. Kod stworzonego dodatku został zaprezentowany na listingu 10.

Listing 10: Kod dodatku służącego do tworzenia raportu wydatków z podziałem na

przypisane do nich kategorie.

```
public class CategoryReport : IFireAndForgetExtension
{
    private readonly IDocumentsService _documentsService;
    private readonly IOAuthResolverService _authResolverService;
    private readonly IIdentityProvider _identityProvider;
    private readonly IOAuthProvider _authProvider;

    public CategoryReport(IDocumentsService documentsService,
        IOAuthResolverService authResolverService,
        IIdentityProvider identityProvider)
    {
        _documentsService = documentsService;
        _authResolverService = authResolverService;
        _identityProvider = identityProvider;
        var providerName =
            _identityProvider.ProviderName.ValueOrFailure();
        _authProvider =
            _authResolverService.GetProvider(providerName);
    }

    public string Name => "CategoryReport";
    public string Label => "Utworz raport wydatkow z podzialem na
        kategorie";
    public string Description => "Dodatek sluzy do tworzenia
        raportu wydatkow z podzialem na kategorie, do ktorych
        dokumenty zostaly przypisane.";
    public string FontAwesomeIconName => "fa fa-eur";
    public async Task PerformAction(int userId)
    {
        IEnumerable<FinanceManager.Domain.Models.CategoryReport>
            report = await
                _documentsService.GetCategoryReport(userId);

        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.AppendLine("L.P,Nazwa kategorii,Opis
            kategorii, Wydatki");
        int i = 0;
        foreach (var categoryReport in report)
        {
            i++;
            stringBuilder.AppendLine(BuildRow(i,
                categoryReport));
        }
    }
}
```



```

    }
    var backupAsByteArray =
        Encoding.UTF8.GetBytes(stringBuilder.ToString());
    var fileName = $"raport-finansowy-" +
        $"{DateTime.UtcNow.ToFileTimeUtc()}.csv";
    var token = _identityProvider.Token.ValueOrFailure();
    await _authProvider
        .UploadFile(token, new FileModel()
            {
                Name = fileName,
                Data = backupAsByteArray
            });
}

private string BuildRow(int lp,
    FinanceManager.Domain.Models.CategoryReport categoryReport)
{
    return $"{lp}," +
        $"{categoryReport.Category.Name}," +
        $"{categoryReport.Category.Description}," +
        $"{decimal.Round(categoryReport.Amount, 2,
            MidpointRounding.AwayFromZero)}";
}
}

```

Przykładowy raport utworzony przez aplikację i otwarty w programie *Microsoft Excel* ma strukturę przedstawioną na rysunku 10.

L.P	Nazwa kategorii	Opis kategorii	Wydatki
1	Samochodowe	Wydatki związane z eksploatacją samochodu	183.00
2	Sport		767.40
3	Zakupy	spożywcze	91.50
4	Buty		199.00

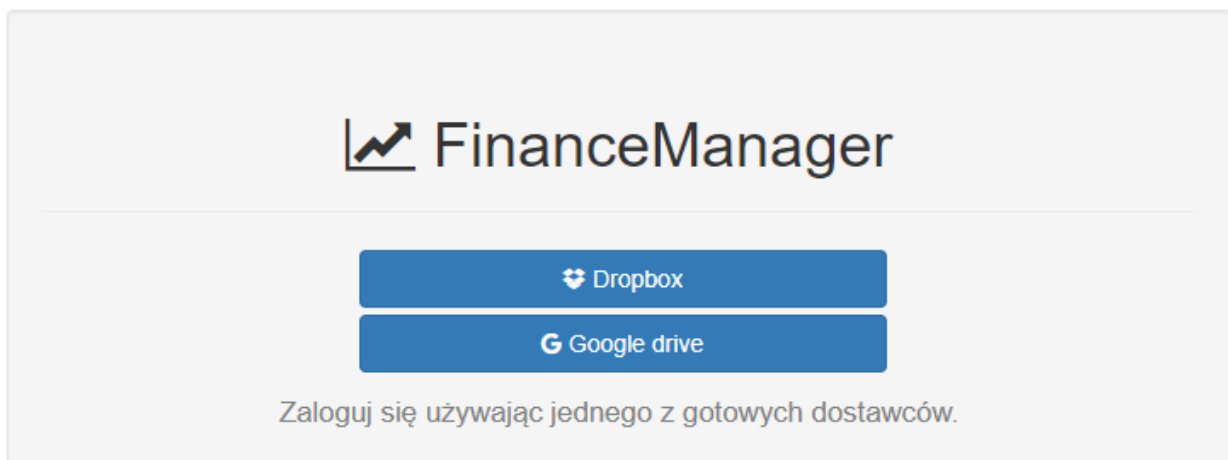
Rysunek 10: Przykładowy raport kosztów z podziałem na kategorie wygenerowany za pomocą dodatku z poziomu aplikacji - wizualizacja w programie Microsoft Excel, Źródło: opracowanie własne

5.2 Przykłady użycia aplikacji

Bieżący rozdział został poświęcony na prezentację przykładowego użycia prototypu aplikacji. W kolejnych podrozdziałach zostanie przedstawiona także różnica pomiędzy wyglądem aplikacji, kiedy zostaje uruchomiona urządzeniu mobilnym, a komputerze posiadającym wyższą rozdzielczość ekranu.

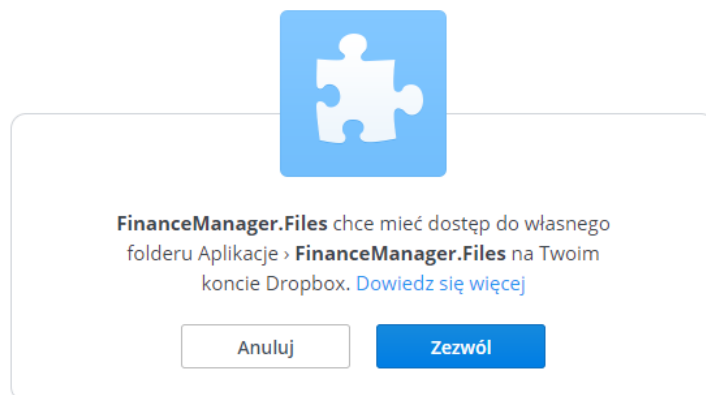
5.2.1 Logowanie do aplikacji

Wchodząc do aplikacji mamy możliwość wyboru dostawcy, u którego trzymane będą nasze dokumenty. Ekran logowania został przedstawiony na rysunku 11.



Rysunek 11: Ekran logowania do aplikacja i jednoczesny wybór dostawcy, Źródło: opracowanie własne

Po wybraniu jednego z dwóch dostawców, zgodnie ze specyfikacją standardu OAuth 2.0 jesteśmy przeniesieni na stronę tego dostawcy w celu podania swoich danych logowania i zaakceptowania faktu udostępniania części przestrzeni dysku na pliki, które będą tworzone w ramach zewnętrznej aplikacji (w tym przypadku *FinanceManagera*). Przykładowy ekran został zaprezentowany na rysunku 12. Jest to ekran, który zobaczy użytkownik, po wybraniu usługi *Dropbox*.

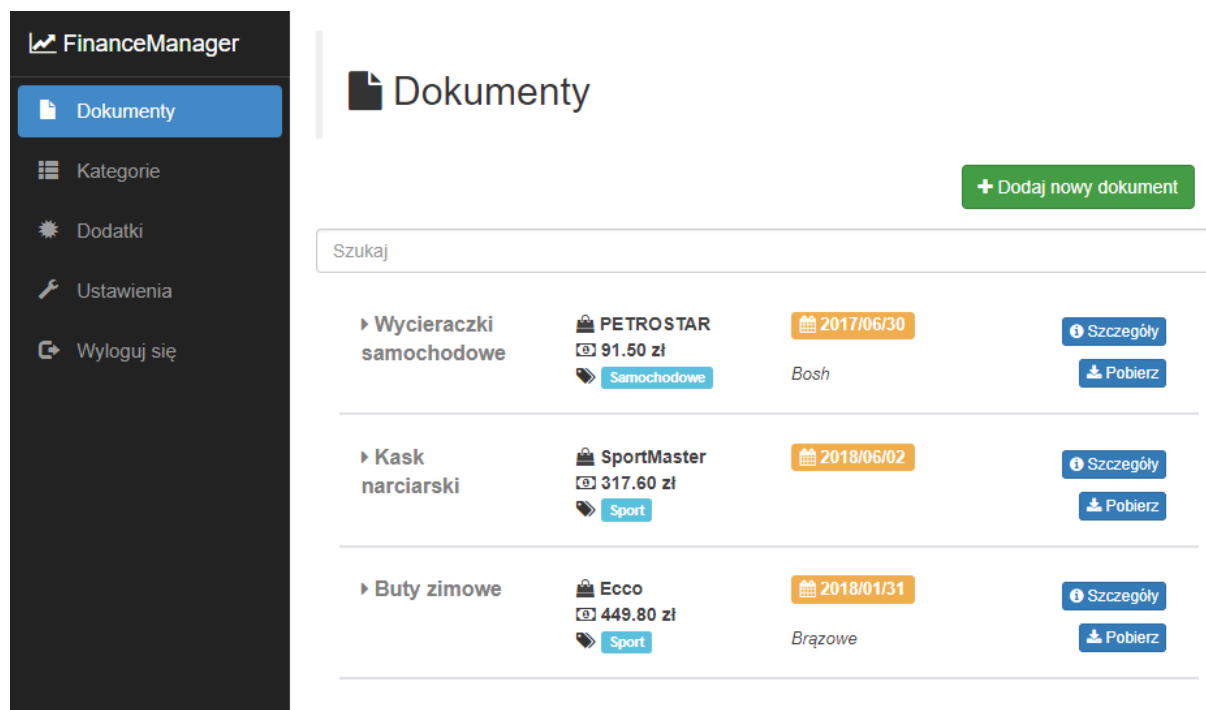


Rysunek 12: Ekran akceptacji wykorzystania własnego folderu aplikacji do trzymania dokumentów finansowych, Źródło: opracowanie własne

Po poprawnej weryfikacji danych logowania oraz zaakceptowaniu warunków trzymania zewnętrznych plików użytkownik zostaje przeniesiony z powrotem na główną stronę prototypu, która została zaprezentowana na rysunku 13.

W przypadku, kiedy użytkownik zdecyduje się odwiedzić aplikację korzystając z urządzenia mobilnego zobaczy widok dostosowany do rozdzielczości urządzenia z którego aktualnie korzysta. Zamiast tradycyjnej tabelki informacje prezentowane będą w formie kafelków. Dzięki temu uniknie się konieczności przesuwania strony horyzontalnie. Obecnie responsywny widok aplikacji jest jedną z najbardziej popularnych metod podczas tworzenia serwisów internetowych. Dodatkowo menu boczne jest zastępowane przez menu górne. Menu górne jest wyświetlane na żądanie użytkownika. Przykład wyglądu aplikacji na urządzeniach mobilnych został przedstawiony na rysunku 14.

Aplikacja uruchomiona za pośrednictwem urządzenia mobilnego ma dokładnie taką samą funkcjonalność jak ta uruchamiana z poziomu komputera PC z wyższą rozdzielczością.

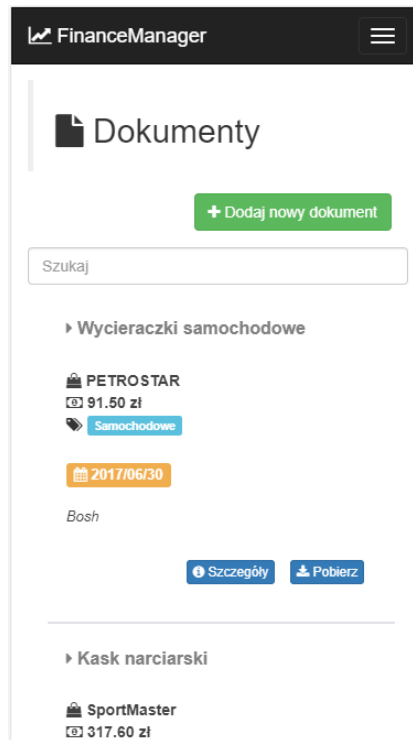


Rysunek 13: Główny ekran aplikacji z listą dokumentów finansowych widoczny po zalogowaniu, Źródło: opracowanie własne

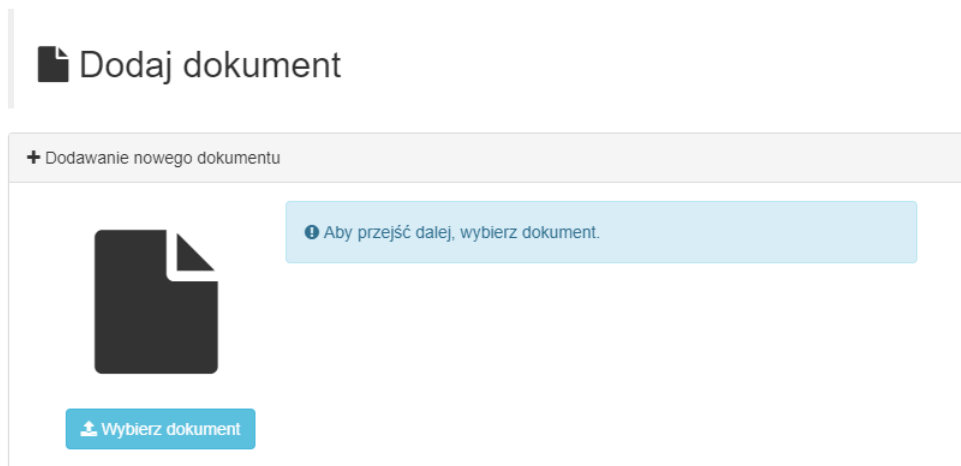
5.2.2 Dodawanie nowego dokumentu finansowego

Dodawanie każdego dokumentu finansowego rozpoczyna się do wybrania pliku z dokumentem (rysunek 15). Plik może być w formacie zdjęcia lub też w tradycyjnym formacie *PDF*. Dodatkowo na urządzeniach mobilnych została udostępniona także opcja zrobienia zdjęcia dokumentu. Jest to duże ułatwienie, kiedy chcemy dodać dokument, a nie posiadamy np. skanera.

Następnie użytkownik ma możliwość wybrania czy chciałby skorzystać z zaimplementowanego systemu OCR lub też wprowadzić wszystkie informacje ręcznie. Aby skorzystać z automatycznego rozpoznawania dokumentu należy kliknąć w przycisk *Przetwórz dokument*. W takim przypadku zostanie wysłane żądanie do aplikacji działającej po stronie serwera aby uruchomiła odpowiednie dodatki (w tym przypadku zaimplementowany dodatek służący do przeprowadzania OCR'a). Wynik przetwarzania zostaje odesłany do aplikacji klienckiej, a



Rysunek 14: Główny ekran aplikacji z listą dokumentów finansowych widoczny po zalogowaniu, Źródło: opracowanie własne



Rysunek 15: Ekran wyboru dokumentu finansowego podczas jego dodawania, Źródło: opracowanie własne

pola w formularzu zostaną automatycznie uzupełnione o dane, które udało się odczytać z dokumentu. Ekran widoczny dla użytkownika został przedstawiony na rysunku 16.

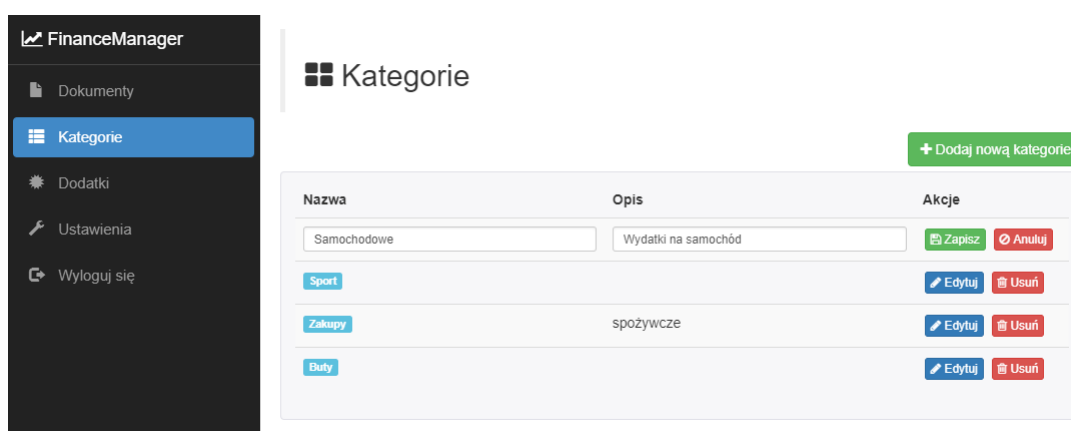
Rysunek 16: Ekran aplikacji służący do uzupełnienia informacji o dodawanym dokumencie, Źródło: opracowanie własne

Po uzupełnieniu wszystkich pól użytkownik powinien kliknąć przycisk *Zapisz*. Sprawia to, że zostanie wysłane kolejne żądanie do serwera, a sam dokument zostanie zapisany na dysku u wybranego podczas logowania dostawcy. Po zakończonej akcji użytkownik zostanie przeniesiony automatycznie na stronę z listą wszystkich swoich dokumentów. Tam też powinien zobaczyć nowo utworzony dokument.

Stan zapisanego dokumentu można także sprawdzić logując się bezpośrednio do wybranego podczas logowania dostawcy i przeglądając zapisane pliki na swoim koncie.

5.2.3 Zarządzanie kategoriami zdefiniowanymi przez użytkownika

Każdy z użytkowników ma możliwość definiowania własnych kategorii, które później są używane podczas dodawania oraz edycji dokumentów finansowych. Do zarządzania kategoriami został przewidziany osobny ekran w aplikacji (rysunek 17). Dodatkowym atutem tego ekranu jest brak przeładowań strony w momencie dodawania lub edycji kategorii. Wszystkie operacje są wykonywane asynchronicznie, użytkownik może skupić się tylko na tym aby możliwie wygodnie zdefiniować kategorie, z których chciałby korzystać.

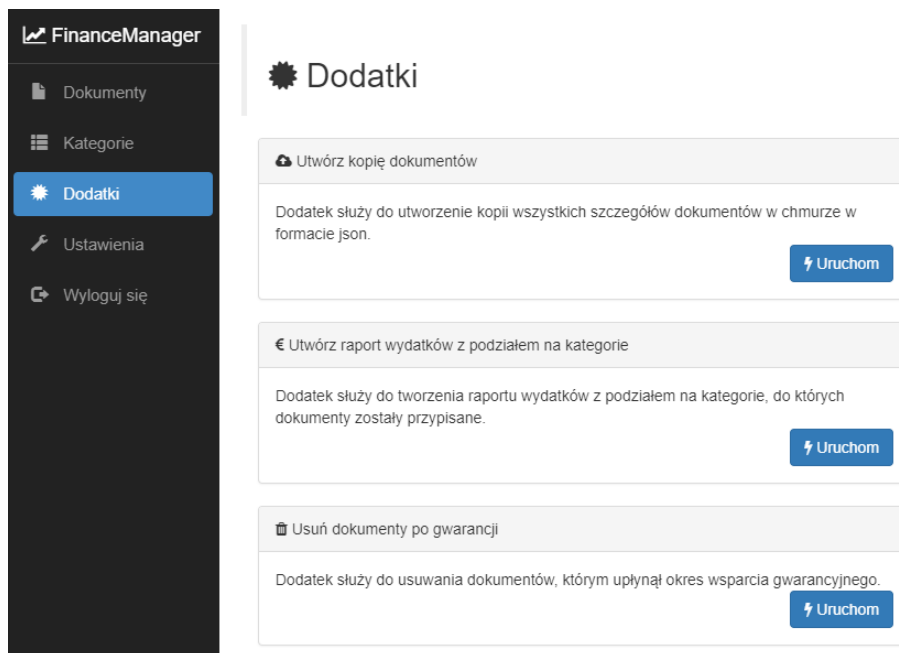


Rysunek 17: Ekran aplikacji służący do zarządzania kategoriami użytkownika, Źródło: opracowanie własne

5.2.4 Ekran służący do uruchamiania dodatków

Kolejnym ekranem dostępnym dla zalogowanego użytkownika w aplikacji jest ekran (rysunek 18) służący do uruchamiania generycznych dodatków. Dodatki dostępne w prototypie zostały dokładnie opisane we wcześniejszym podrozdziale.

Uruchomienie wybranego dodatku odbywa się po naciśnięciu przycisku *Uruchom*, znajdującego się obok każdego z dostępnych dodatków. System następnie wyświetli pytanie, czy użytkownik jest pewny chęci skorzystania z wybranego dodatku. Zapobiega to sytuacją przypadkowego kliknięcia w przycisk. W przypadku wyrażenia zgody na użycie dodatku następuje



Rysunek 18: Ekran aplikacji służący do uruchamiania wcześniej zaimplementowanych generycznych dodatków, Źródło: opracowanie własne

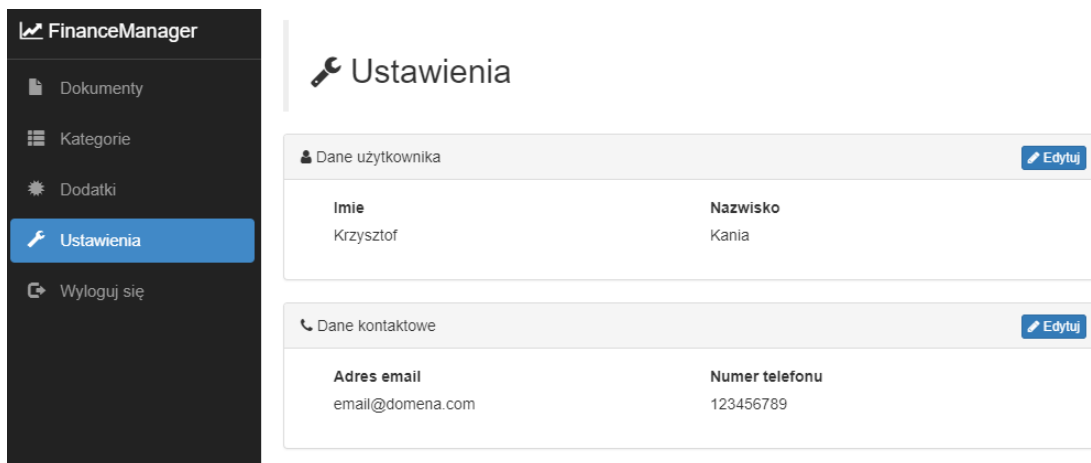
wysłanie żądania do serwera, a fakt powodzenia uruchomienia wybranego dodatku zostaje zaprezentowany użytkownikowi aplikacji w osobnym komunikacie.

5.2.5 Ekran edycji profilu zalogowanego użytkownika

Ostatnim z ekranów do których ma dostęp zalogowany użytkownik, jest ekran do ustawień profilu użytkownika. Dostęp do tego ekranu uzyskuje się poprzez wybranie z menu opcji *Ustawienia*. Każdy z użytkowników może uzupełnić swój profil o takie informacje prezentowane na rysunku 19.

Dane wykorzystywane w sekcji *Dane kontaktowe* pochodzą z aplikacji działającej po stronie serwera (back-end). Jednocześnie są to dane generycznie zdefiniowane w bazie danych. Dzięki temu w przyszłości możliwe jest ich dowolne rozszerzanie o kolejne możliwości kontaktu z użytkownikiem.

Na tym ekranie także został zaimplementowany mechanizm edycji danych pozbawiony



Rysunek 19: Ekran aplikacji służący do uzupełnienia danych kontaktowych użytkownika,
Źródło: opracowanie własne

przeładowania strony. Po kliknięciu w przycisk *Edytuj* dane prezentowane na formularzu w formie tekstu zostają zastąpione przez formularz zawierający pola typu *text input*¹⁹. Podmiana następuje w sposób płynny i bardzo czytelny dla użytkownika.

¹⁹Są to pola umożliwiające wprowadzanie danych

6 Podsumowanie

Bieżący rozdział został poświęcony na prezentację dalszych możliwych kierunków rozwoju prototypu oraz krótkiemu omówieniu problemów jakie zostały napotkane podczas tworzenia rozwiązania.

6.1 Dalsze kierunki rozwoju prototypu

Każda dobrze zaprojektowana aplikacja pozostawia duże pole do jej dalszego rozwoju. W tym podrozdziale zostały zabrane możliwe kierunki, w których w przyszłości będzie można rozwijać prototyp:

- Stworzenie dodatków przyjmujących generyczne parametry wejściowe, które będą uruchamiana przez użytkownika na żądanie. Będzie to rozszerzeniem funkcjonalności obecnych dodatków, które nie dają możliwości przekazywania do nich jakichkolwiek parametrów.
- Dodanie możliwości definiowania generycznych pól, jakie będą opisywały każdy z dokumentów finansowych. Do definiowania pól może służyć osobny kreator, dostępny przez aplikację front-end'ową. Implementacja tej funkcjonalności sprawi, że prototyp stanie się dużo bardziej elastyczny.
- Dodanie możliwości współdzielenia dokumentów pomiędzy użytkownikami.
- Dodanie funkcjonalności powiadamiania sms lub email o ważnych akcjach wykonanych na koncie użytkownika.

6.2 Napotkane problemy podczas implementacji

Proces implementacji prototypu wymagał poradzenia sobie z kilkoma problemami:

- Jednym z głównych problemów był proces integracji z zewnętrznymi dostawcami. Problematyczna okazała się nie zawsze w pełni czytelna dokumentacja opisująca kroki, jakie należy wykonać w celu skorzystania z dostępnych metod oferowanych przez dostawców.
- Kolejnym etapem pracy, na jaki poświęcono dużo czasu było stworzenie własnego dodatku odpowiedzialnego za rozpoznawanie informacji (dodatek *OCR*) z przesłanych przez użytkowników dokumentów finansowych. System *Google Vision* pomimo dobrej dokumentacji technicznej, nie zawsze zachowywał się dokładnie tak jak zostało w niej opisane.
- Problemem okazało się także korzystanie z *Google Drive* (jako zewnętrznego dostawcy), który wymagał wysłania niestandardowych zapytań podczas wysyłania i aktualizowania dokumentów użytkownika. Przykłady umieszczone na stronie producenta nie były wystarczająco czytelne.
- Jednym z wyzwań było także sprawienie, aby aplikacja działała bardzo stabilnie oraz była odporna na wszelkiego rodzaju niestandardowe akcje użytkownika.

6.3 Zakończenie

Aplikacje wspomagające zarządzanie dokumentami finansowymi stają się coraz bardziej popularne. W ramach tej pracy powstało kolejne narzędzie, które oferuje szereg zalet w porównaniu do tych obecnych na rynku. Stworzony prototyp może być alternatywą dla osób oczekujących większych możliwości dostosowania systemu do swoich potrzeb, a co za tym idzie realną alternatywą do aplikacji istniejących już na rynku.

Bibliografia

- [1] R. C. Martin. *Czysty kod*, Helion, ISBN: 978-83-283-1401-6
- [2] Joydip Kanjilal *ASP.NET Web Api* Build RESTful web applications and services on the .NET framework, ISBN: 978-1-84968-974-8
- [3] Mathieu Nayrolles, Rajesh Gunasundaram, Sridhar Rao *Expert Angular* ISBN 978-1-78588-023-0
- [4] Nathan Rozentals *Mastering TypeScript, Second Edition* ISBN 978-1-78646-871-0
- [5] Jakub Narebski *Mastering Git* ISBN 978-1-78355-375-4
- [6] *SOLID* <https://stackify.com/solid-design-principles/>, dostęp 2018-08-10
- [7] *oAuth 2.0* <https://tools.ietf.org/html/rfc6749>, dostęp 2018-06-10
- [8] *Advanced Encryption Standard (AES)* <https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard>, dostęp 2018-07-28
- [9] *Nuget* <https://docs.microsoft.com/en-us/nuget/what-is-nuget>, dostęp 2018-07-28
- [10] *Postacie normalne w bazach danych* <https://www.sqlpedia.pl/projektowanie-i-normalizacja-bazy-danych/>, dostęp 2018-08-01
- [11] *REST API* <https://www.oreilly.com/learning/how-to-design-a-restful-api-architecture-from-a-human-language-spec>, dostęp 2018-08-03
- [12] *Licencja typu MIT* <https://opensource.org/licenses/MIT>, dostęp 2018-08-10
- [13] *.NET CORE* <https://www.microsoft.com/net/learn/what-is-dotnet>, dostęp 2018-08-11
- [14] *SQL Server 2017 Express* <https://www.microsoft.com/en-us/sql-server/sql-server-2017-editions>, dostęp 2018-08-12

- [15] *Popularność frameworka Angular na tle konkurencji* <https://itnext.io/angular-5-vs-react-vs-vue-6b976a3f9172>, dostęp 2018-08-12
- [16] *Entity Framework Core* <https://docs.microsoft.com/en-us/ef/core/>, dostęp 2018-07-26
- [17] *Language Integrated Query (LINQ)* <https://docs.microsoft.com/en-us/dotnet/csharp/linq/>, dostęp 2018-08-03
- [18] *Automapper* <https://automapper.org/>, dostęp 2018-07-12
- [19] *Flurl* <https://flurl.io/>, dostęp 2018-07-17
- [20] *ReSharper* <https://www.jetbrains.com/resharper/>, dostęp 2018-06-24
- [21] *Visual Studio Code* <https://code.visualstudio.com/>, dostęp 2018-06-25
- [22] *Manager paczek NuGet* <https://www.nuget.org/>, dostęp 2018-06-25
- [23] *Manager paczek npm* <https://www.npmjs.com/>, dostęp 2018-06-25
- [24] *Biblioteka ng-swagger-gen* <https://github.com/cycloproject/ng-swagger-gen>, dostęp 2018-04-12
- [25] *Swagger* <https://swagger.io/specification/>, dostęp 2018-07-29
- [26] *Diagram przedstawiający proces skorzystania ze standardu OAuth* <https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>, dostęp 2018-08-25
- [27] *Scrutor* <https://github.com/khellang/Scrutor>, dostęp 2018-08-01
- [28] *Google Cloud Vision* <https://cloud.google.com/vision/>, dostęp 2018-08-10
- [29] *Inversion of Control* <https://martinfowler.com/articles/injection.html>, dostęp 2018-08-16

Spis rysunków

1	Główne okno aplikacji „gdzieparagon.pl”, Źródło: opracowanie własne	8
2	Główne okna aplikacji „Smart Paragon”, Źródło: payu.pl/blog	10
3	Okno dodawania nowego dokumentu finansowego w aplikacji „Paragon - gwarancje pod ręką”, Źródło: opracowanie własne	11
4	Główne okno aplikacji „Paragon - gwarancje pod ręką”, Źródło: opracowanie własne	12
5	Diagram architektury aplikacji, Źródło: opracowanie własne	24
6	Diagram struktury bazy danych aplikacji, Źródło: opracowanie własne	30
7	Diagram przedstawiający proces korzystania ze standardu OAuth 2.0, Źródło: digitalocean.com [26]	39
8	Przykładowy dokument finansowy użyty do rozpoznania przez autorski moduł OCR, Źródło: opracowanie własne	48
9	Formularz dodawania nowego dokumentu finansowego z rozpoznanymi informacjami z dokumentu finansowego z rysunku 8. Źródło: opracowanie własne	50
10	Przykładowy raport kosztów z podziałem na kategorie wygenerowany za pomocą dodatku z poziomu aplikacji - wizualizacja w programie Microsoft Excel, Źródło: opracowanie własne	55
11	Ekran logowania do aplikacja i jednoczesny wybór dostawcy, Źródło: opracowanie własne	56
12	Ekran akceptacji wykorzystania własnego folderu aplikacji do trzymania dokumentów finansowych, Źródło: opracowanie własne	57
13	Główny ekran aplikacji z listą dokumentów finansowych widoczny po zalogowaniu, Źródło: opracowanie własne	58

14	Główny ekran aplikacji z listą dokumentów finansowych widoczny po zalogowaniu, Źródło: opracowanie własne	59
15	Ekran wyboru dokumentu finansowego podczas jego dodawania, Źródło: opracowanie własne	59
16	Ekran aplikacji służący do uzupełnienia informacji o dodawanym dokumencie, Źródło: opracowanie własne	60
17	Ekran aplikacji służący do zarządzania kategoriami użytkownika, Źródło: opracowanie własne	61
18	Ekran aplikacji służący do uruchamiania wcześniej zaimplementowanych generycznych dodatków, Źródło: opracowanie własne	62
19	Ekran aplikacji służący do uzupełnienia danych kontaktowych użytkownika, Źródło: opracowanie własne	63