**UNIWERSYTET EKONOMICZNY W KATOWICACH**

**Informatyka**

**Dominik JEDZINIAK**
**140510**

# Implementacja aplikacji do zarządzania domowym budżetem „BPlan”

## Implementation of an application helping on home budget management “BPlan”

Praca licencjacka
napisana w Katedrze Informatyki
pod kierunkiem
Dr hab. Artura Strzeleckiego

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem
i stwierdzam, że spełnia wymogi stawiane pracom dyplomowym

Pracę akceptuję

……
……

(data) (podpis promotora)

**KATOWICE 2021**

Dominik Jedziniak

Kolegium Informatyki i Komunikacji

Kierunek: Informatyka

Nr Albumu: 140510

Katowice, dnia 16.09.2021

# OŚWIADCZENIE

Świadom(a) odpowiedzialności prawnej oświadczam, że złożona praca licencjacka/inżynierska/magisterska pt.: „Implementacja aplikacji wspomagającej tworzenie harmonogramu pracy pracowników" została napisana przeze mnie samodzielnie.

Równocześnie oświadczam, że praca ta nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (tj. Dz. U. z 2018 r., poz. 1191, z późn. zm.) oraz dóbr osobistych chronionych prawem.

Ponadto praca nie zawiera informacji i danych uzyskanych w sposób niedozwolony i nie była wcześniej przedmiotem innych procedur związanych z uzyskaniem dyplomów lub tytułów zawodowych uczelni wyższej.

Wyrażam zgodę na nieodpłatne udostępnienie mojej pracy w celu oceny jej oryginalności przez Jednolity System Antyplagiatowy prowadzony przez Ministra Nauki i Szkolnictwa Wyższego oraz przechowywania jej w Ogólnopolskim Repozytorium Prac Dyplomowych oraz wewnętrznej bazie prac dyplomowych Uniwersytetu Ekonomicznego w Katowicach. Zostałem poinformowany o zasadach dotyczących oceny oryginalności pracy dyplomowej przez Jednolity System Antyplagiatowy.

Oświadczam także, że ostateczna wersja pracy przesłana przeze mnie drogą elektroniczną jest zgodna z plikiem poddanym ocenie w Jednolitym Systemie Antyplagiatowym.

Jednocześnie oświadczam, że jest mi znany przepis art. 233 § 1 Kodeksu karnego określający odpowiedzialność za składanie fałszywych zeznań.

Dominik Jedziniak

(podpis składającego oświadczenie)

# Spis treści

# 1. Wprowadzenie

## 1.1. Wprowadzenie i rys historyczny

Wymiana dóbr towarzyszy ludzkości od zarania dziejów a na jej podstawie wykształciły się pojęcia kupna i sprzedaży.

Pierwsza forma handlu polegała na wymianie towarów bez obecności środka pośredniczącego czyli barter. Barter stanowił jednak duży problem w dokładnym określeniu wartości produktu dlatego z czasem zaczęto używać produktów pośredniczących które spełniały założenia późniejszego pieniądza. Łatwość transportu, podzielność czy rzadkość występowania cechowały ówczesne środki płatnicze. Wraz z rozwojem cywilizacji funkcję tą przejęły metale. W celach ujednolicenia środka płatności zaczęło się dzielenie surowca na mniejsze kawałki i licencjonowanie poprzez wybijane na nich oznakowania co było wstępem do monetyzacji.

Wszystko to jednak miało formę jawną, wydawanie posiadanych środków skutkowało zmniejszeniem swoich aktywów w kontekście objętościowym czy wagowym. Wizualizacja aktywów pozwalała w prosty sposób planować i nadzorować stan budżetu domowego. Pomijając czeki, weksle i pierwsze karty kredytowe oraz bankowe dopiero wiek XXI przyniósł rewolucję jeżeli chodzi o możliwości płatności, w tym szeroko rozumiany pieniądz elektroniczny. Różnorodność form płatności oraz wprowadzenie pieniądza w formie wirtualnej wymusza na współczesnym człowieku posiadania minimum kilku aplikacji bankowych lub płatniczych. Rejestracja przychodów i rozchodów oraz planowanie budżetu w takiej formie nie jest zcentralizowane.

12 stycznia 2016 weszła w życie dyrektywa PSD2, czyli otwarta bankowość zmieniająca oblicze europejskiej i światowej bankowości. AIS czyli *Account Information Service* wprowadza nowy standard w rozumieniu bankowości. Założeniem funkcjonalności AIS ma być udostępnianie zagregowanych informacji o rachunku klienta od dostawców usług płatniczych, co stanowi kolosalną zmianę w przedstawianiu informacji o finansach klienta. Efekty zmian widać już teraz, kiedy za pomocą jednej aplikacji możliwe jest nadzorowanie stanu środków pieniężnych z rachunków prowadzonych u różnych dostawców. Wprowadzenie tej usługi mocno przyśpieszyło rozwój aplikacji mobilnych sektora

bankowości detalicznej ponieważ informacje o stanach kont dostępne są teraz w jednym a zarazem w każdym miejscu. Niewątpliwie AIS jest krokiem naprzód jeżeli chodzi o centralizację danych budżetu domowego, natomiast problem który nadal nie został do końca rozwiązany to śledzenie wydatków wraz ze zbieraniem danych analitycznych które ułatwią planowanie budżetu lub wybieranie najatrakcyjniejszych cenowo miejsc na zakupy.

Od roku 2020 w Polsce przedsiębiorcy mają możliwość wystawiania e-paragonów, czyli paragonów wystawianych i przesyłanych drogą elektroniczną. Brak jasnego standardu w przekazywaniu informacji, jak i niskie zainteresowanie przedsiębiorców powoduje, że rozwiązanie nie jest w powszechnym użytku. Dodatkowo mała popularność kas wirtualnych czy brak poparcia ludności w cyfryzacji paragonów nie pozwala na przyśpieszenie rozwoju projektu. Pojawiały się próby współpracy pomiędzy różnymi instytucjami mające na celu popularyzację paragonów elektronicznych, m.in. program pilotażowy E-Paragon realizowany przez PKO Bank Polski, PKN Orlen, KIR i eService jednak zazwyczaj mają one charakter czasowy.

Wszystko to skutkuje problemami w zarządzaniu domowym budżetem, śledzeniu swoich wydatków czy obserwacji wzrostu bądź spadku cen produktów przez zwykłego Kowalskiego. Poszerzanie wiedzy na temat swoich finansów mogłoby przekładać się na polepszenie statusu materialnego bądź lepsze rozeznanie w procesach rynkowych.

Poniższa praca zakłada zaplanowanie i stworzenie systemu który będzie wspomagał użytkownika w kwestiach śledzenia wydatków i wizualizacji aktywów finansowych. Całość projektu ma być nastawiona na praktyczność oraz łatwość obsługi. W pracy powstaną przykładowe integracje z podmiotami udostępniającymi informacje w formacie elektronicznym.

## 1.2. Przykłady dostępnego oprogramowania

Rynki aplikacji internetowych oraz mobilnych są jednymi z najszybciej rozwijających się gałęzi IT w związku z czym istnieje znaczna ilość oprogramowania, która pozwala na zbieranie informacji o wydatkach i finansach domowych. Duża konkurencja w tej dziedzinie jednak nie pozwoliła na stworzenie ogólnoświatowego ideału aplikacji, który z sukcesem mógłby opanować rynek aplikacji finansowych.

W pracy zostanie przedstawionych kilka rozwiniętych i bardzo popularnych aplikacji wraz z ich funkcjonalnościami czy modelem dystrybucji. Dodatkowo zostaną przedstawione aplikacje których funkcjonalnością poboczną jest śledzenie lub planowanie budżetu. Dogłębna analiza dostępnych aplikacji skoncentrowana będzie na przedstawieniu ich mocnych i słabych stron, tak aby mogły one później pomóc w dostosowaniu funkcjonalności BPlan.

### 1.2.1. Cozy.io

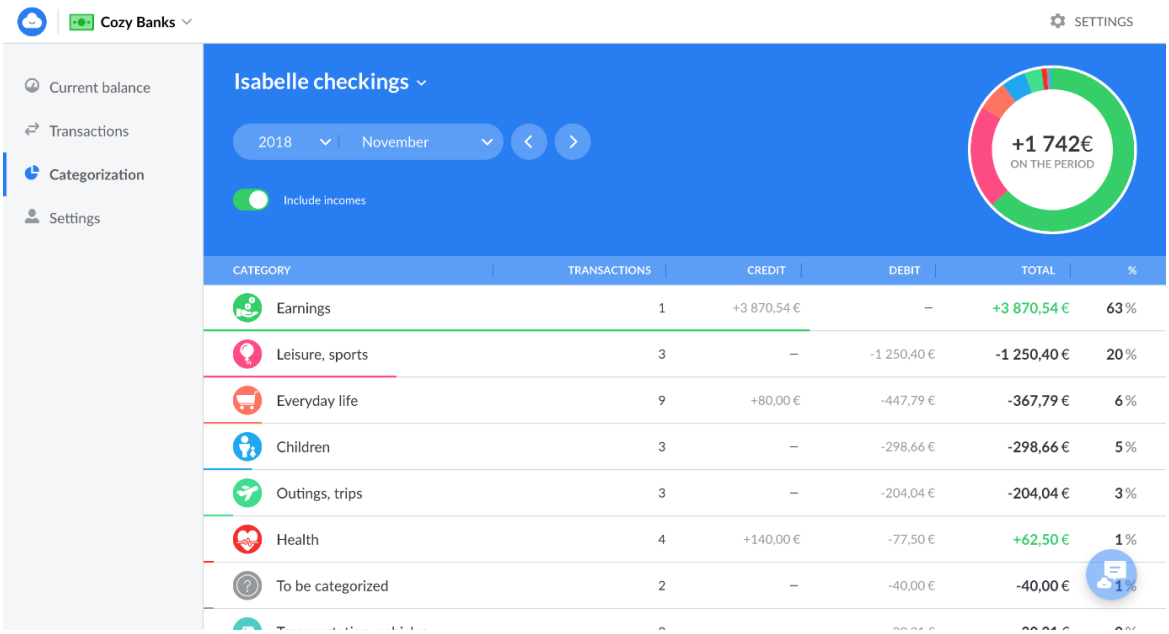
Aplikacja rozwijana przez francuską firmę Cozy Cloud, której celem jest stworzenie oprogramowania dostarczającego dużą ilość usług chmurowych ułatwiających codzienne czynności. W swojej ofercie posiadają między innymi zbieranie informacji odnośnie stanu konta wraz z przychodami i rozchodami, przechowywaniu plików w chmurze, zarządzanie dokumentami i notatkami czy funkcjonalność menadżera haseł. Całość utrzymywana jest na serwerach firmy OVH we Francji, co jest istotnym punktem w kwestiach marketingowych.

Serwis udostępniany jest w formule SaaS (ang. *Software as a service*) w kilku wariantach finansowych - zaczynając od planu darmowego, który cechuje się małą ilością dostępnej przestrzeni dyskowej, po plany mogące służyć jako serwery domowej kopii zapasowej z zasobami dyskowymi sięgającymi rozmiarów 1 terabajta. Oprócz interfejsu przeglądarkowego, Cozy udostępnia również aplikacje na platformy PC, Android oraz iOS. System zapewnia pobieranie informacji poprzez specjalne łączniki zwane dalej konektorami.

W przypadku pozyskiwania informacji finansowych, Cozy nawiązało partnerstwo z firmą Budget Insight, która na rynku francuskim jest liderem w kontekście narzędzi PFM (ang. *Personal Financial Management*).

Chmura Cozy jest oprogramowaniem o kodzie otwartoźródłowym (ang. *open source*), co ma stanowić potwierdzenie jakości i bezpieczeństwa przechowywania danych. Z tego względu, wydawana jest również wersja pozwalająca na utworzenie własnej instancji aplikacji i serwowaniu jej z własnego sprzętu (ang. *self-hosted*).

*Rys. 1 Panel kategorii Cozy Banks*




*Źródło: https://cozy.io/en/features/ - Cozy Cloud - Get the power of your data*

Transakcje są kategoryzowane według rodzaju odbiorcy, ale możliwe jest ich sortowanie względem czasu, wolumenu, czy udziału w wydatkach. Podobne mechanizmy z powodzeniem funkcjonują w popularnych aplikacjach dostawców usług bankowych jak PKO BP czy mBank. Istnieje również możliwość ręcznego dodania dokumentu potwierdzającego transakcję, jednak brakuje automatycznych mechanizmów pobierających dane z serwisów odbiorcy. Dodawane potwierdzenia nie są również później analizowane pod względem pozycji znajdujących się na nich.

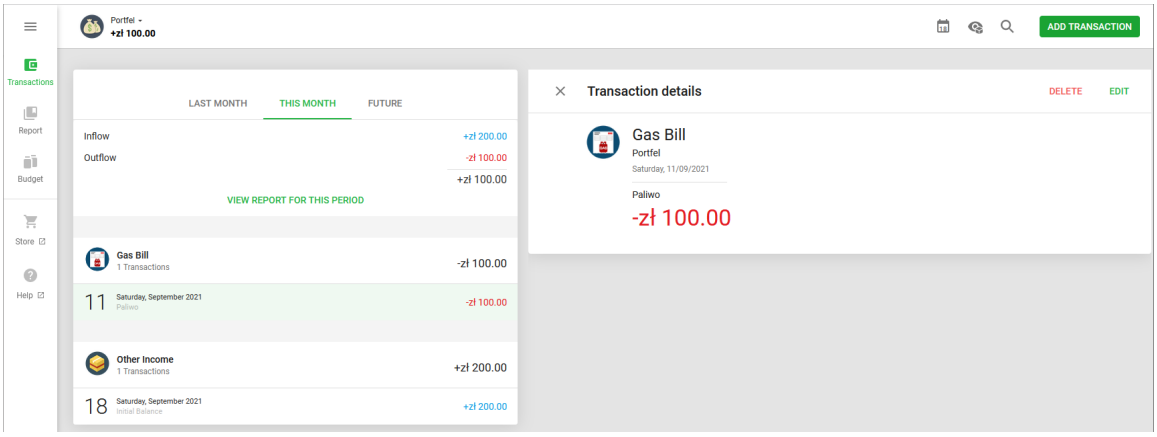
### 1.2.2. Money Lover

Serwis firmy Finsify Technology Co. Ltd. oferujący tylko i wyłącznie funkcjonalności mające na celu ułatwienie śledzenia transakcji finansowych.

Aplikacja dostępna jest na platformach PC, Android, iOS czy poprzez interfejs przeglądarki. Oprogramowanie zakłada, że informacje dotyczące transakcji będą wprowadzone ręcznie przez użytkownika, co może sprawiać duży problem osobom mało systematycznym. Aplikacja nie przewiduje prowadzenia spisu zakupionych produktów. Dodatkowe zastosowania Money Lover stanowią funkcjonalności pozwalające na wprowadzanie przypomnień dla cyklicznych płatności i celów finansowych czy przeliczanie transakcji w innych walutach celem wizualizacji wydatków w rodzimym środku pieniężnym.

Aplikacja dostarczana jest jako SaaS, niestety jej dostawca nie podaje informacji o tym, gdzie przechowywane są dane klienta. Pomimo angielsko brzmiącej nazwy, aplikacja jest rozwijana w Wietnamie, co w przypadku braku możliwości zweryfikowania miejsca przechowywania danych może budzić wątpliwości.

*Rys.2 Panel główny aplikacji wraz z dodanymi transakcjami*



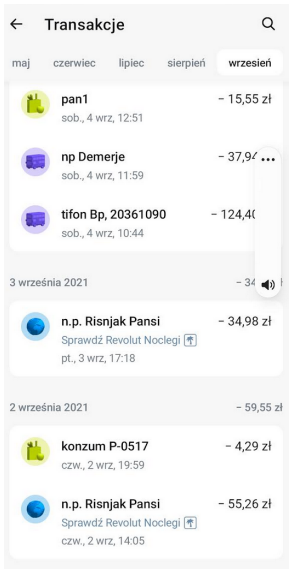

*Źródło: opracowanie własne*

### 1.2.3. Revolut

Startup posiadający europejską licencję bankową oferujący usługi wymiany walut, karty przedpłacone, handel kryptowalutami i papierami wartościowymi na kilku światowych giełdach czy płatności peer-to-peer. Możliwość wymiany walut po bardzo konkurencyjnych cenach pozwoliła na rozkwit fintechu m.in. w Polsce oraz w innych krajach Europy środkowo-wschodniej. Jako funkcje dodatkowe możemy wymienić programy cashback czy możliwość planowania budżetu i celów zbiórek pieniędzy.

Oprogramowanie dostępne jest tylko na urządzenia mobilne, przez co nie ma możliwości podglądu stanu konta przez interfejs przeglądarki.

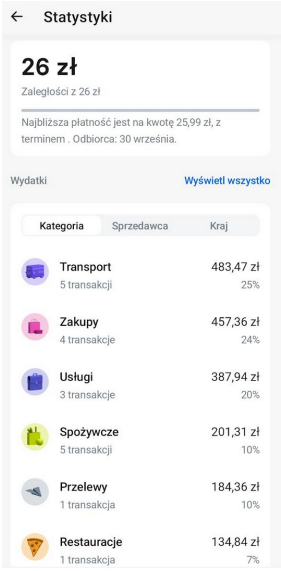
*Rys.3 Przykładowy ekran transakcji w Revolut*




*Źródło: opracowanie własne*

Revolut, często określany jako bank w telefonie, tworzy pełen ekosystem usług pozwalających na śledzenie wydatków oraz zarządzanie zarówno budżetem jak i bezpośrednio funduszami. Dodatkowa automatyczna kategoryzacja wydatków, pozwala na szybką analizę najbardziej obciążającej portfel dziedziny życia. Transakcje analizowane są również pod względem operacji cyklicznych - po wykryciu powtarzalności w obciążaniu rachunku, aplikacja przed terminem następnej płatności wyśle powiadomienie.

*Rys.4 Okno statystyk aplikacji Revolut*




*Źródło: opracowanie własne*

Minusem całego systemu jest duża hermetyczność względem innych instytucji finansowych. Dane prezentowane w Revolut dotyczą tylko i wyłącznie wydatków prowadzonych z własnego konta przedpłaconego, co nie jest dobrym przykładem centralizacji informacji finansowych.

Oprócz wspomagania planowania budżetu, umożliwione jest również tworzenie swoistego sejfu, do którego transferowana jest reszta z zaokrąglonej kwoty dokonywanej transakcji.

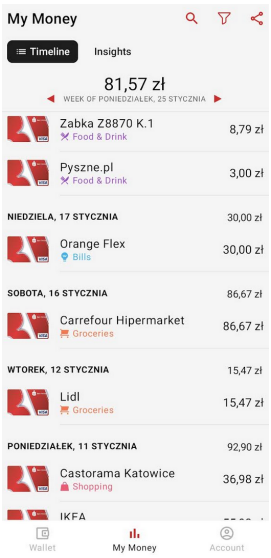
### 1.2.4. Curve

Mimo tego, że na pierwszy rzut oka aplikacja Curve wydaje się być bliźniaczo podobna do aplikacji Revolut, dzieli je znacząca różnica polegająca na tym, że Curve nie ma na celu oferowania funkcjonalności banku a działać jedynie jako agregator kart płatniczych.

Aplikacja działa jak proxy dla kart płatniczych oferując przy tym bardzo konkurencyjne stawki przewalutowania. Użytkownik dodaje do aplikacji karty kredytowe bądź debetowe, które są zamiennie używane według określonego schematu. Osoba eksploatująca aplikację ma do dyspozycji mechanizmy Google Pay, standardową kartę Mastercard lub też Apple Pay.

Oprogramowanie jest dostępne tylko na platformy mobilne, jednak oferuje szeroki zakres informacji dotyczących wydatków z danej karty. Transakcje są kategoryzowane według odbiorcy lub kategorii jakiej dotyczyła płatność. Możliwe jest sortowanie według różnych kryteriów oraz ręczne dodawanie paragonów w formie zdjęcia.

*Rys.5 Panel płatności Curve*




*Źródło: opracowanie własne*

## 1.3. Identyfikacja niszy w dostępnym oprogramowaniu

Przedstawione aplikacje mają wiele cech wspólnych w funkcjonalności, jednak każda z nich zawiera pewne luki czyniąc ją niekompletną bądź trudną w wykorzystaniu.

W obecnym świecie coraz więcej uwagi przywiązuje się do automatyzacji, która ma zapewnić dużą oszczędność czasu użytkownika, dlatego najważniejszym celem powinno być załatanie luki, która wymaga wprowadzania informacji ręcznie. Warto też wyróżnić inne podstawowe punkty, które aplikacja tego typu musi spełniać aby być użyteczną dla odbiorcy.

### 1.3.1. Automatyczne pobieranie danych

Jednym z najważniejszych punktów niniejszej pracy jest założenie, że dane finansowe i analityczne powinny być wprowadzane automatycznie bez znaczącej ingerencji użytkownika. Minimum potrzebne do funkcjonowania aplikacji to dane transakcyjne.

Przychody i rozchody powinny być rejestrowane od wszystkich dostawców usług finansowych użytkownika w możliwie jak najprostszy sposób. Dodatkową wartość analityczną zapewni pobieranie paragonów i w miarę możliwości zapisywanie dostępnych na nich pozycji celem późniejszej analizy.

### 1.3.2. Łatwość implementacji

Ze względu na metodę dostarczenia aplikacji jaką jest praktyka self-hosted, bezpieczeństwo danych pośrednio zostanie zostawione w rękach użytkownika, dlatego też zostanie przedstawionych kilka technik pozwalających zabezpieczyć środowisko uruchomieniowe.

Zarówno instalacja jak i konfiguracja powinny być bezawaryjne oraz proste w obsłudze. Dodatkowo środowisko powinno być łatwo przenoszalne, tak aby użytkownik domowy mógł je bez problemu przenieść ze stacji domowej nawet na serwer zewnętrzny. Sposób implementacji powinien też zakładać mnogość dostępnych platform na których uruchomić można serwer aplikacyjny.

Dostęp do aplikacji przewidziany jest głównie przez przeglądarkę internetową a sam interfejs ma pozwalać na późniejsze przekształcenie aplikacji internetowej w aplikację progresywną.

### 1.3.3. Mnogość dostępnych rozszerzeń

Jednym z celów nadrzędnych jest również dostarczenie w prosty sposób rozszerzalności o kolejne instytucje bankowe czy podmioty udostępniające informacje o transakcjach. Śladem aplikacji Cozy, opracowany model aplikacji będzie zakładał możliwość dopisywania konkretnych konektorów i dodawanie ich do ekosystemu BPlan nawet przez

zwykłego użytkownika. Rozwiązanie użyte w tym temacie powinno być bardzo uniwersalne, proste i powszechnie stosowane w innych aplikacjach.

## 1.4. Cel pracy i plan wykonania

Celem pracy jest przedstawienie procesu realizacji aplikacji wraz z wyjaśnieniem użycia technologii bądź oprogramowania obsługującego daną część opracowywanego zagadnienia.

W kolejnych rozdziałach zostanie przedstawiona architektura, technologie, języki programowania i ich frameworki. Duży nacisk zostanie skierowany również na opis konfiguracji który zapewni ciągłą i bezawaryjną pracę oprogramowania.

Ważnym etapem podczas realizacji aplikacji będzie analiza dokumentacji technicznej dostarczonej przez producenta oprogramowania. Literaturą, która pozwoli na wybranie środowiska uruchomieniowego są artykuły a w nich m.in. „*Docker [Software engineering]*" autorstwa Charlesa Andersona oraz „*An Introduction to Docker and Analysis of its Performance*" wydany przez International Journal of Computer Science and Network Security, które w prosty sposób argumentują użycie konteneryzacji w nowoczesnych aplikacjach. Wybór frameworka argumentowany jest opracowaniem *„Comparative study on Python web frameworks: Flask and Django"* autorstwa Ghimire Devndra. Pomocą w opracowaniu mechanizmu pobierania danych będzie opracowanie pt. „*Using Python for Text Analysis in Accounting Research*" wydanego przez University of Miami Business School.

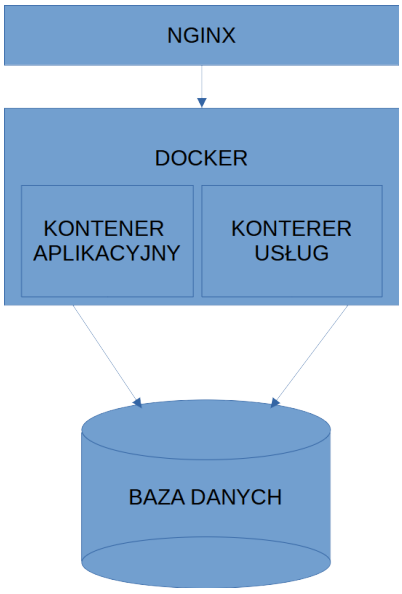
# 2. Opis implementacji

## 2.1. Architektura aplikacji

Koncept zakłada aplikację trzypoziomową (ang. *three tier*) zawierającą 3 podstawowe warstwy:

- Warstwa prezentacji (ang. *reverse proxy*)
- Warstwa aplikacji
- Warstwa bazy danych

*Rys.6 Architektura aplikacji*




*Źródło: opracowanie własne*

Charakterystyka aplikacji pozwoliłaby zamknąć się w modelu dwuwarstwowym, jednak ciekawszym rozwiązaniem okazuje się wersja wzbogacona dodatkowym poziomem, jakim jest warstwa reverse proxy, która w tym wypadku może być łączona również z warstwą prezentacji.

Podstawowym uzasadnieniem użycia dodatkowej warstwy, jest zastosowanie mechanizmu certbot, który automatycznie wygeneruje certyfikat SSL podpisany przez

centrum certyfikacji *Let's Encrypt,* co znacznie ułatwi i przyśpieszy pracę. Dodatkowo, zastosowanie NGINX jako odwróconego proxy pozwalać będzie na utworzenie pamięci podręcznej, za czym idzie odciążenie warstwy aplikacyjnej z serwowania materiałów statycznych.

Ze względu na niższe dwie warstwy, użycie dodatkowego serwera http zapewni zaimplementowanie mechanizmu fallback czy wprowadzenie statycznych dedykowanych stron kodów błędu. Wyjaśnienie użycia poszczególnych komponentów rozwijane będzie w kolejnych podrozdziałach rozdziału drugiego.

### 2.1.1. Nginx

Nginx, czyli serwer HTTP, serwer reverse proxy oraz serwer proxy protokołów TCP i UDP stworzony przez Igora Sosojewa, aktualnie rozwijany przez firmę Nginx wraz ze wsparciem jednostki macierzystej jaką jest F5 Networks. Serwer charakteryzuje się niskim zużyciem zasobów, wysoką wydajnością i łatwą rozszerzalnością za pomocą modułów. Dostępny w dwóch wariantach:

- Nginx wydawany na 2-punktowej licencji z rodziny BSD. Darmowy oraz wieloplatformowy
- Nginx+, czyli płatna wersja biznesowa wydawana z dużo większym zestawem modułów i komercyjnym zespołem wsparcia

Aktualnie na rynku istnieje duża różnorodność jeżeli chodzi o serwery HTTP - możemy wymienić między innymi Apache HTTP Server, który jeszcze do niedawna wiódł prym w tej dziedzinie, Cuddy2 czyli serwer z wbudowaną obsługą Let's Encrypt pozwalającą na szybkie uruchomienie serwisu działającego na protokole HTTPS, wspomniany w tytule nginx czy wiele innych serwerów HTTP.

Wybór serwera nginx był podyktowany prostotą konfiguracji oraz obszernym zestawem modułów dostarczanych w pakiecie instalacyjnym. Różnica w ilości linii potrzebnych do poprawnej konfiguracji jest znacząca, co zilustrują poniżej przedstawione przykłady realizacji pamięci podręcznej dla nginx oraz Apache HTTP Server.

*Rys. 7 Przykładowa konfiguracja disk cache dla nginx*

```
proxy_cache_path /path/to/cache levels=1:2 keys_zone=my_cache:10m max_size=10g inactive=60m use_temp_path=off;

server {
    location / {
        proxy_cache my_cache;
        proxy_pass http://my_upstream;
    }
}
```

*Źródło: https://www.nginx.com/blog/nginx-caching-guide/ - A Guide to Caching with NGINX and NGINX Plus - NGINX*

Warto zaznaczyć, że linie zaczynające się od server oraz location nie są składowymi konfiguracji cache. Są to podstawowe i obowiązkowe dyrektywy konfiguracji nginx.

*Rys.8 Przykładowa konfiguracja Apache HTTP Server*

```
LoadModule cache_module modules/mod_cache.so
<IfModule mod_cache.c>
    LoadModule cache_disk_module modules/mod_cache_disk.so
    <IfModule mod_cache_disk.c>
        CacheRoot "c:/cacheroot"
        CacheEnable disk  "/"
        CacheDirLevels 5
        CacheDirLength 3
    </IfModule>
</IfModule>
```

*Źródło: https://httpd.apache.org/docs/2.4/mod/mod_cache.html - mod_cache - Apache HTTP Server Version 2.4*

Linie zaczynające się od CacheDirLevels oraz CacheDirLenght nie są wymagane, jednak znajdują się w przykładowej konfiguracji cache. Należy zauważyć, że taka konstrukcja wymaga załadowania 2 dodatkowych modułów.

### 2.1.2. Python

Python jest językiem programowania wysokiego poziomu o szerokim spektrum zastosowania.

Stosowany jako język skryptowy i język implementacji aplikacji internetowych, w ostatnich latach szczególnie popularny w dziedzinie sztucznej inteligencji i data science. Prostota składni, dynamiczne typowanie zmiennych, duża gama bibliotek i prostota ich instalacji prowadzą do wybrania tego języka jako głównej technologii użytej do implementacji aplikacji wspomagającej zarządzanie budżetem. Dodatkowym atutem Pythona jest też jedna z największych społeczności i ogromny popyt na programistów tego języka.

Projekt zakłada użycie wersji 3 języka Python.

### 2.1.3. Flask

Flask jest frameworkiem języka python dedykowanym dla aplikacji webowych, określanym mianem micro-framework ze względu na jego stosowanie. Przedrostek micro- odnosi się do serca aplikacji, nazywanego dalej jądrem (ang. *core*), które później może być rozwijane zewnętrznymi rozszerzeniami.

Domyślnie Flask nie zawiera warstwy obsługi bazy danych - do tego celu używane są wspomniane rozszerzenia. Dodatkowo Flask cechuje się dużą prostotą tworzenia aplikacji, przekładającą się na czas poświęcony implementacji i niewielką ilość kodu, przez co idealnie wpasuje się w projekt Bplan.

*Rys.9. Przykładowy kod małej aplikacji zwracającej predefiniowany tekst*

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "<p>Hello World</p>"
```

*Źródło: opracowanie własne*

Po uruchomieniu aplikacji i wysłaniu zapytania GET / poprzez cURL otrzymamy poniższy wynik:

```
* Expire in 0 ms for 6 (transfer 0x1dcc8b0)
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Expire in 200 ms for 4 (transfer 0x1dcc8b0)
* Connected to 127.0.0.1 (127.0.0.1) port 5000 (#0)
> GET / HTTP/1.1
> Host: 127.0.0.1:5000
> User-Agent: curl/7.64.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 18
< Server: Werkzeug/0.14.1 Python/3.7.3
< Date: Sun, 19 Sep 2021 14:41:24 GMT
<
* Closing connection 0
<p>Hello World</p>
```

### 2.1.4. Docker

Docker, czyli platforma o otwartym kodzie źródłowym, służąca do tworzenia, dostarczania i uruchamiania aplikacji w specjalnie przygotowanych wyizolowanych środowiskach uruchomieniowych zwanych kontenerami. Kontenery domyślnie są oddzielnymi bytami, dopiero świadome skonfigurowanie elementów pozwala na łączenie poszczególnych elementów tworząc rozbudowane systemy.

Samo pojęcie konteneryzacji można przyrównać w większości do pojęcia wirtualizacji z jedną zasadniczą różnicą - kontenery działają w obrębie jądra systemu, tworząc dużo lżejszą konstrukcję  która mniej obciąża zasoby serwera.

Popularność rozwiązania zaowocowała bogatym zbiorem predefiniowanych obrazów, które można z powodzeniem implementować w budowanych aplikacjach. Oprócz predefiniowanych obrazów istnieje możliwość zbudowania własnego obrazu na podstawie pliku DOCKERFILE.

Plik dockerfile to zbiór poleceń opisujących proces budowy obrazu. Możemy wyróżnić kilka najczęściej używanych poleceń.

- FROM - Definiuje bazowy obraz na którym oparte będą dalsze akcje
- MAINTAINER - Dane osoby utrzymującej obraz
- COPY - Pozwala na kopiowanie plików z dysku lokalnego do obrazu
- ADD - Rozszerzona wersja COPY potrafiąca wypakowywać archiwa oraz ma możliwość obsługi protokołu HTTP
- RUN - Wykonuje podane jako argument polecenie wewnątrz kontenera
- USER - Definiuje domyślnego użytkownika
- WORKDIR - Definiuje katalog aplikacji
- CMD - Domyślna komenda wykonywana przez kontener

*Rys.10. Przykładowy plik dockerfile*

```
FROM alpine:latest
CMD ["echo", "Hello World"]
```

*Źródło: opracowanie własne*

Budowa obrazu w katalogu gdzie znajduje się plik dockerfile:

`docker build . -t „testowy”`

Uruchomienie obrazu powinno wyświetlić tekst „Hello World”:

`docker run testowy`

*Rys.11 Zrzut ekranu budowy i wykonania testowego obrazu*

```
pi@rpi:~ $ docker build . -t testowy
Sending build context to Docker daemon  1.819GB
Step 1/2 : FROM alpine:latest
latest: Pulling from library/alpine
a14774a5a62e: Already exists
Digest: sha256:e1c082e3d3c45cccac829840a25941e679c25d438cc8412c2fa221cf1a824e6a
Status: Downloaded newer image for alpine:latest
 ---> 3e8172af00ce
Step 2/2 : CMD ["echo", "Hello World!"]
 ---> Running in 380cafb90635
Removing intermediate container 380cafb90635
 ---> 0c9c14d909f7
Successfully built 0c9c14d909f7
Successfully tagged testowy:latest
pi@rpi:~ $ docker run testowy
Hello World!
```

*Źródło: opracowanie własne*

### 2.1.5. Biblioteka re

Wyrażenia regularne często określano jako regex lub regexp, czyli skrót od regular expression. Zbiór znaków opisujący konkretny łańcuch symboli, pozwalający na określenie podzbioru pasujących ciągów znaków bądź wyszczególnienie szukanej części ciągu znaków.

W przypadku projektowanej aplikacji wyrażenia regularne będą stosowane do wyciągania informacji z powiadomień mailowych banków, celem zapisu ich w bazie danych. Do tego celu zostanie wykorzystana biblioteka języka python - re.

Wyróżnić możemy podstawowe kwantyfikatory dla wyrażeń regularnych pozwalające na budowanie skomplikowanych wzorców:

- ^Tekst - Wyszukuje linie tekstu zaczynające się od słowa „Tekst”
- Tekst$ - Wyszukuje linie tekstu kończące się słowem „Tekst”
- K* - Ciąg składający się z zera lub więcej „K”
- K? - Ciąg składający się z zera lub jednego wystąpienia „K”
- K+ - Ciąg składający się z jednego lub więcej wystąpień „K”
- K{2} - Dokładnie dwa wystąpienia „K”
- K{,2} - Do dwóch wystąpień „K”
- K{2,} - Minimum dwa wystąpienia „K”

*Rys.12 Kod wykorzystujący bibliotekę re pozwalający na potwierdzenie wystąpienia danego ciągu znaku*

```
import re

ciagZnakow = "Lorem ipsum dolor sit amet"
ciagSzukany = "^.*dolor.*$"

if re.search(ciagSzukany,ciagZnakow):
    print("Znaleziono")
else:
    print("Brak wynikow")
```

*Źródło: opracowanie własne*

Wynikiem powyższego skryptu będzie słowo „Znaleziono”.

### 2.1.6. Requests

Requests to zewnętrzna biblioteka języka python, obsługująca połączenia protokołu http.

Biblioteka ma na celu uproszczenie wykonywania zapytań co znacząco skraca kod aplikacji i przyśpiesza pisanie aplikacji.

*Rys.13 Przykład wykorzystania biblioteki requests*

```
import requests
r = requests.get('https://pue.zus.pl:8001/ws/zus.channel.gabinetoweV3.zla?wsdl', auth=('ezla_ag','ezla_ag'))
print(r.status_code)
```

*Źródło: opracowanie własne*

Przykład prezentuje wykonanie żądania GET dla adresu aplikacji gabinetowych udostępnianych przez ZUS. Parametr zawiera żądanie autoryzacji metodą „Basic". Wywołanie tak skonstruowanego skryptu powinno zwrócić kod 200 a w wyniku znalazłby się plik definicji webservice.

### 2.1.7. SQLite

SQLite, czyli biblioteka o kodzie otwartoźródłowym implementująca mały, szybki i niezależny silnik relacyjnej bazy danych, który nie wymaga uruchamiania innych procesów. Biblioteka dostarcza w pełni funkcjonalny silnik SQL. Dane zawarte w bazie danych przechowywane są w jednym pliku na dysku, który nie jest zależny od architektury serwera bądź środowiska. Umożliwia to łatwe przenoszenie danych względem dostępnych platform. Rozwiązanie idealnie pasuje do małych projektów, gdzie dostęp jest przyznawany jednemu użytkownikowi. Dodatkowo nie istnieje żadna procedura instalacji bądź konfiguracji bazy SQLite co znacznie usprawnia proces instalacji.

Istnieje możliwość uruchomienia bazy danych bezpośrednio w pamięci RAM, przez co dane mogą być dostępne dużo szybciej. Takie rozwiązanie niestety wiąże się z łatwą ulotnością danych podczas przełączania aplikacji.

Warto nadmienić, że SQLite obsługuje tylko cztery typy danych:

- INTEGER - służy do przechowywania liczb całkowitych
- TEXT - przechowuje ciągi znaków
- REAL - obsługuje liczby zmiennoprzecinkowe
- BLOB - z angielskiego *binary large object* czyli typ obsługujący dane binarne

## 2.2. Implementacja

Implementacja aplikacji zawiera również podstawową konfigurację środowiska oraz dobre praktyki w zabezpieczaniu serwerów aplikacyjnych. Dalsze podrozdziały będą skupiać się konfiguracji bądź propozycji realizacji danego zagadnienia. Omówiona zostanie również rola danego komponentu w ekosystemie aplikacji i ewentualne możliwości modyfikacji w przyszłości.

### 2.2.1. Reverse proxy realizowane przez nginx

Odwrotne proxy, czyli dodatkowa warstwa między serwowaną aplikacją a klientem, mająca zapewnić balansowanie ruchu, pamięć podręczną, udostępnianie zawartości po protokole HTTPS, mechanizmy fallback czy zabezpieczyć przed bezpośrednim dostępem do aplikacji.

Zasada działania reverse proxy polega na przekazywaniu zapytań od klienta z zewnątrz do serwera aplikacyjnego i odsyłaniu odpowiedzi aplikacji do klienta zapewniając brak bezpośredniego kontaktu klienta z serwerem aplikacyjnym.

*Rys.14 Schemat przedstawiający zasadę działania reverse proxy*


*Źródło: opracowanie własne*

Konfiguracja zamieszczona w pracy zakłada posiadanie wykupionej domeny internetowej która kieruje na serwer udostępniający aplikację (w tym wypadku nginx). Istnieje możliwość uruchomienia aplikacji na serwerze bez wykupionej domeny bądź w sieci wewnętrznej, jednak wymagać to będzie wygenerowania ręcznie certyfikatu self-signed oraz dostarczenia go do konfiguracji nginx.

Prezentowana aplikacja była uruchamiana na serwerze VPS działającym pod systemem Ubuntu. Po zainstalowaniu pakietów python3-certbot-nginx oraz nginx została wykonana pierwsza konfiguracja.

*Rys.15 Konfiguracja nginx*

```
worker_processes  5;
error_log  logs/error.log;
pid        logs/bplan_nginx.pid;
worker_rlimit_nofile 8192;

events {
  worker_connections  4096;
}

http {
proxy_cache_path /tmp/bplan levels=1:2 keys_zone=bplan:10m max_size=1g inactive=30d use_temp_path=off;

  server {
    listen       82;
    server_name  pomido.re www.pomido.re;
    access_log   /var/log/pomido_re.access.log;
    error_page 404 /app/404.html;


    location ~ ^/images/*  {
      proxy_cache bplan;
      proxy_pass      http://127.0.0.1:5000;
    }

    location ~ ^/css/*  {
      proxy_pass      http://127.0.0.1:5000;
      expires 30d;
    }

    location / {
      proxy_pass      http://127.0.0.1:5000;
    }
  }
}
```

*Źródło: opracowanie własne*

Tak uzupełniona konfiguracja pozwoli na uruchomienie serwera HTTP. Dodatkowo, po wykonaniu poniższego polecenia, konfiguracja zostanie uzupełniona o wpisy dotyczące serwowania ruchu po protokole HTTPS.

```
sudo certbot --nginx -d pomido.re -d www.pomido.re
```

Dla lokalizacji images została włączona obsługa cache dyskowego. Zgodnie z ustawieniami w dyrektywie proxy_cache_path, pamięć podręczna będzie utrzymywana przez 30 dni. Lokalizacja css, w przeciwieństwie do images, nie posiada obsługi pamięci podręcznej, natomiast użyta tam dyrektywa expires sprawia, że do odpowiedzi dodawany jest nagłówek, który zapewnia utrzymywanie cache po stronie przeglądarki przez 30 dni. Reszta

stron nie podpada pod pamięć podręczną ze względu na dynamiczny charakter danych prezentowanych użytkownikowi.

Dyrektywa proxy_pass przekierowuje ruch do serwera 127.0.0.1 na porcie 5000.

### 2.2.2. Integracja pobierania informacji o transakcjach

Jednym z podstawowych założeń aplikacji jest automatyczne pobieranie informacji odnośnie transakcji. Należy wiedzieć, że dyrektywa PSD2 wymusza udostępnianie API poprzez instytucje finansowe dla podmiotów trzecich w ramach AISP (ang. *Account Information Service Providers*). W przypadku aplikacji rozwijanej w niniejszej pracy nie jest możliwym otrzymanie statusu TPP (ang. *Third Party Providers*), dlatego też rozwiązanie pobierania informacji nie może korzystać z API powstałego dzięki PSD2. Opracowywane rozwiązanie musi bazować na środkach dostępnych dla zwykłego Kowalskiego i być na tyle uniwersale, aby dotyczyło dużego zbioru instytucji finansowych. Po dogłębnej analizie dostępnych źródeł danych najbardziej pasującym rozwiązaniem okazuje się analizowanie maili użytkownika pod kątem notyfikacji mailowych o transakcji. Do tego celu zostały przygotowane specjalne wyrażenia regularne zbierające dane z otrzymanych wiadomości.

```python
import imaplib
import re
import email
import dateutil.parser
from email.header import decode_header

#Sprawdzanie wiadomości młodszych niż (dana pobierana z bazy)
#Wprowadzenie limitu czasowego (wartość będzie pobierana z bazy)
tsLimit = 1632152941
regex = {
    'PKO Bank Polski <powiadomienia@pkobp.pl> ': {
        '=?UTF-8?Q?Transakcja_kart=C4=85?=' : (r'Kwota: ([0-9]+\,[0-9]+)', r'Miejsce: (.*)','OUTCOME'),
        '=?UTF-8?Q?Wp=C5=82yw_na_konto?=' : (r'kwota \*\+(.*) ', r'w tym:\n\n(.*)','INCOME')
    }
}
#Utworzenie obiektu imaplib
poczta = imaplib.IMAP4_SSL("imap.gmail.com",993)
#Zalogowanie się do poczty
poczta.login('bplan.agg@gmail.com','HASLO_!_!.')
#Wybranie skrzynki
status, wiadomosciLiczba = poczta.select("INBOX")
#Ograniczenie w pobieraniu adresow
ograniczenie = 25
# Liczba wiadomosci email w skrzynce
wiadomosciLiczba = int(wiadomosciLiczba[0])

for i in range(wiadomosciLiczba-ograniczenie, wiadomosciLiczba+1):
    if i > 0:
        #Pobierz wiadomości
        res, wiadomosci = poczta.fetch(str(i), "(RFC822)")
        for wiadomosc in wiadomosci:
            if isinstance(wiadomosc, tuple):
                #Dekodowanie wiadomości
                msg = email.message_from_bytes(wiadomosc[1])
                date = decode_header(msg["Date"])[0][0]
                if (int(dateutil.parser.parse(date).timestamp())) >= tsLimit:
                    if msg["From"] in regex:
                        if msg["Subject"] in regex[msg["From"]]:
                            mail = msg.get_payload()[0]
                            kwota = re.findall(regex[msg["From"]][msg["Subject"]][0], str(mail))[0]
                            dane = re.findall(regex[msg["From"]][msg["Subject"]][1], str(mail))[0]
                            typ = regex[msg["From"]][msg["Subject"]][2]
```

*Rys.16 Uproszczony kod zbierania informacji z skrzynki email*

*Źródło: opracowanie własne*

Powyższy wycinek kodu jest uproszczoną wersją mechanizmu pobierającego dane z adresu mailowego. Słownik regex zawiera wyrażenia regularne przypisane do konkretnego adresu mailowego, przez co dodawanie kolejnych organizacji ograniczać będzie się do aktualizacji słownika. Dzięki użyciu biblioteki imaplib następuje logowanie, wybranie folderu skrzynki mailowej, pobranie i odkodowanie wiadomości. Po dopasowaniu adresu email oraz tematu wiadomości za pomocą wyrażeń regularnych wyciągane są informacje o zdarzeniu. Wynikowo otrzymujemy 3 informacje odnośnie transakcji. Jakiej kwoty dotyczy powiadomienie, jaki jest opis zdarzenia oraz czy transakcja miała charakter obciążeniowy czy była wpływem środków na konto.

Docelowo taki mechanizm wykonywać się będzie cyklicznie co kilka minut dzięki zastosowaniu prostej pętli.

### 2.2.3. Pobieranie informacji na temat nabytych towarów

Drugim znaczącym punktem aplikacji BPlan jest pobieranie informacji na temat przedmiotu transakcji. Ze względu na znikomą dostępność e-paragonów możliwości są bardzo ograniczone. Opracowanie usługi pobierania informacji na temat paragonu ogranicza się do przedsiębiorstwa Lidl oraz serwisów przez nich udostępnianych.

Analizując metodą reverse engineering ruch sieciowy pomiędzy aplikacją LidlPlus a serwerami udostępniającymi informacje, udało się opracować mechanizm wymiany informacji i zaimplementować go za pomocą języka python w aplikacji.

*Rys.17 Klasa Lidl*

```python
import json
import requests

class Lidl:

    __lidlAuth = auth=('LidlPlusNativeClient','secret')
    __tokenUrl = 'https://accounts.lidl.com/connect/token'
    __headers = {
        'App-Version': '999.99.9',
        'Operating-System': 'iOS',
        'App': 'com.lidl.eci.lidl.plus',
        'Accept-Language': 'PL'
        }

    def __init__(self,lidlToken):
        """Wyciągnij acces_token"""
        self.lidlToken = lidlToken
        payload='refresh_token=' + self.lidlToken + '&grant_type=refresh_token'
        headers = {
            'Content-Type': 'application/x-www-form-urlencoded'
            }
        resp = requests.request("POST", self.__tokenUrl, headers=headers, data=payload, auth=self.__lidlAuth)
        self.__headers["authorization"] = "Bearer " + resp.json()["access_token"]

    def refreshToken(self):
        """Wykonaj ponownie inicjator"""
        self.__init__(self.lidlToken)

    def zakupy(self,page=1):
        """Pobierz liste ostatnich 25 paragonow"""
        __zakupy =  "https://appgateway.lidlplus.com/tickets/api/v1/PL/list/" + str(page)
        resp = requests.request("GET", __zakupy, headers=self.__headers)
        try:
            resp.raise_for_status()
        except:
            self.refreshToken()
        finally:
            self.zakupy
        return json.dumps(resp.json())

    def paragon(self,idTransakcji):
        """Pobierz liste zakupow, idTransakcji = id paragonu"""
        __paragon = 'https://appgateway.lidlplus.com/app/V24/PL/tickets/' + idTransakcji
        resp = requests.request("GET", __paragon, headers=self.__headers)
        try:
            resp.raise_for_status()
        except:
            self.refreshToken()
        finally:
            self.paragon
        return json.dumps(resp.json())
```

*Źródło: opracowanie własne*

Powyżej przedstawiona została klasa odpowiedzialna za połączenie z danymi aplikacji LidlPlus, gdzie zaimplementowane zostały trzy metody odpowiedzialne kolejno za: ponowne wykonanie inicjacji biletu dostępu, pobranie 25 ostatnich transakcji, pobranie paragonu wraz z detalami. Z racji żywotności tokenu dostępu w wypadku kodu błędu większego lub równego 400 zostanie wykonana ponowna inicjalizacja, czyli ponowne pobranie tokenu.

Klasa zostanie użyta do kolejnego modułu, który podobnie jak mechanizm do analizy wiadomości email co kilka minut będzie analizował za pomocą wyrażeń regularnych wykonane transakcje i przypisywał je do odpowiadających nim paragonów. Przykładowe odpowiedzi metod zostały przedstawione poniżej.

*Rys.18 Odpowiedź metody Lidl.zakupy()*


```
{
    "page":1,
    "size":25,
    "totalCount":2,
    "records":[
        {
            "id":"24001623520201219650547",
            "isFavorite":false,
            "date":"2020-12-19T15:41:45+00:00",
            "totalAmount":"85,05",
            "storeCode":"PL1623",
            "currency":{
                "code":"PLN",
                "symbol":"z\u0142"
            },
            "articlesCount":21,
            "couponsUsedCount":0,
            "hasReturnedItems":false,
            "returnedAmount":"0",
            "invoiceRequestId":null,
            "invoiceId":null
        },
        {
            "id":"24001623120201209785075",
            "isFavorite":false,
            "date":"2020-12-09T12:56:38+00:00",
            "totalAmount":"61,06",
            "storeCode":"PL1623",
            "currency":{
                "code":"PLN",
                "symbol":"z\u0142"
            },
            "articlesCount":20,
            "couponsUsedCount":0,
            "hasReturnedItems":false,
            "returnedAmount":"0",
            "invoiceRequestId":null,
            "invoiceId":null
        }
    ]
}
```


*Źródło: opracowanie własne*

*Rys.19 Odpowiedź metody Lidl.paragon()*


```
{
    "id":"2400162312020090972618l",
    "barCode":"8881623726181001090920",
    "sequenceNumber":"726181",
    "workstation":"01",
    "itemsLine":[
        {
            "currentUnitPrice":"2,45",
            "quantity":"2",
            "isWeight":false,
            "originalAmount":"4,90",
            "extendedAmount":"4,90",
            "description":"Ser Moz. bez GMO",
            "taxGroup":"3",
            "taxGroupName":"C",
            "codeInput":"4056489091752",
            "discounts":[

            ],
            "deposit":null,
            "giftSerialNumber":null
        },
        {
            "currentUnitPrice":"2,49",
            "quantity":"1",
            "isWeight":false,
            "originalAmount":"2,49",
            "extendedAmount":"2,49",
            "description":"Cukier kryszta\u0142",
            "taxGroup":"2",
            "taxGroupName":"B",
            "codeInput":"20624477",
            "discounts":[

            ],
            "deposit":null,
            "giftSerialNumber":null
        }
    ],
```


*Źródło: opracowanie własne*

Zgodnie z wcześniejszymi informacjami, wykorzystując atak MITM (ang. *Man in the middle*) został podsłuchany ruch sieciowy pomiędzy aplikacją LidlPlus a serwerami udostępniającymi dane. Tak przygotowane dane pozwoliły na przeanalizowanie mechanizmu logowania aplikacji i odtworzenie go później w języku python tworząc swoistą nakładkę realizującą logowanie.

*Rys.20 Inicjalizator klasy OpenID*

```python
from sre_constants import error
import requests, string, random, hashlib, base64, time, re
from urllib import parse
from seleniumwire import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities

class OpenID:
    __meta = '/.well-known/openid-configuration'
    __login = "adres@email.com"
    __code_verif = str(''.join(random.choices(string.ascii_uppercase + string.ascii_lowercase + string.hexdigits, k = 43)))
    __code_challenge = base64.b64encode(hashlib.sha256(__code_verif.encode('utf-8')).digest()).decode('UTF-8').replace('/','_').replace('=','').replace('+','-')
    __pass = "Tutaj_Haslo"
    __config = {
        'code_verification' : __code_verif,
        'code_challenge' : __code_challenge,
        'login_country' : 'PL',
        'login_lang' : 'PL-PL',
        'client_id': 'LidlPlusNativeClient',
        'redirect_uris': 'com.lidlplus.app://callback',
        'response_types': 'code',
        'scope': 'openid profile offline_access lpprofile lpapis',
        'code_challenge_method': 'S256'
    }
    def __init__(self,issuer):
        self.__config["issuer"] = issuer
        self.ctx = requests.request('GET',self.__config["issuer"] + self.__meta, verify=None).json()
        self.authUrl = self.ctx["authorization_endpoint"] + '?client_id=' + self.__config["client_id"] + '&scope=' + self.__config["scope"]
        self.authUrl += '&response_type=' + self.__config["response_types"] +'&redirect_uri=' + self.__config["redirect_uris"] +'&code_challenge='
        self.authUrl += self.__config["code_challenge"] + '&code_challenge_method=' + self.__config["code_challenge_method"] +'&Country='
        self.authUrl += self.__config["login_country"] +'&language=' + self.__config["login_lang"]
```

*Źródło: opracowanie własne*

Klasa OpenID implementuje OpenID Relying Party. Podczas inicjalizacji pobierane są parametry konfiguracyjne OpenID a następnie tworzony jest adres punktu autoryzacji wraz z zaszyfrowanym SHA256 ciągiem 43 znaków zwanym dalej code challenge.

*Rys.21 Metoda OpenID.getURL()*

```python
def getURL(self):
    chrome_options = Options()
    d = DesiredCapabilities.CHROME
    d['goog:loggingPrefs'] = { 'browser': 'ALL'}
    chrome_options.add_argument('--headless')
    chrome_options.add_argument('--log-level=3')
    driver = webdriver.Chrome(options=chrome_options, desired_capabilities=d)
    driver.delete_all_cookies()
    driver.get(self.authUrl)
    time.sleep(1)
    zalogujBtn = driver.find_element_by_id("button_welcome_login")
    zalogujBtn.click()
    time.sleep(1)
    wpiszLogin = driver.find_element_by_id("field_EmailOrPhone")
    wpiszLogin.click()
    time.sleep(1)
    wpiszLogin.send_keys(self.__login)
    time.sleep(1)
    step2Btn = driver.find_element_by_id("button_btn_submit_email")
    step2Btn.click()
    time.sleep(1)
    wpiszHaslo = driver.find_element_by_id("field_Password")
    wpiszHaslo.click()
    wpiszHaslo.send_keys(self.__pass)
    time.sleep(1)
    self.step3Btn = driver.find_element_by_id("button_submit")
    time.sleep(1)
    self.step3Btn.click()
    time.sleep(1)
    self.j = False
    while self.j == False:
        for request in driver.requests:
            if re.match('.*lidlplus.*', request.url):
                if request.response.headers["location"]:
                    try:
                        self.sessionState(request.response.headers["location"])
                    except:
                        time.sleep(1)
    driver.close()
```

*Źródło: opracowanie własne*

Kolejnym etapem autoryzacji jest zasymulowanie działania przeglądarki internetowej tak aby wprowadziła dane na temat logowania. W tym punkcie zostało użyte Selenium, czyli platforma umożliwiająca wykonywania zautomatyzowanych akcji poprzez przeglądarkę. Zagłębiając się w kod widać, że tym wypadku został użyty Selenium Webdriver czyli biblioteka zapewniająca komunikację z przeglądarką i wykonywanie na niej zdalnych akcji.

Korzystając z identyfikatorów elementów znajdujących się na stronie została wykonana sekwencja wprowadzania danych w formularzu logowania po której „naciśnięty" został przycisk wyślij. Wysłanie formularza skutkuje sprawdzeniem przesyłanych nagłówków HTTP, wybrany zostaje ten który spełnia warunki przedstawione w metodzie sessionState.

*Rys.22 Metody sessionState oraz getToken*

```
def sessionState(self, location):
    if re.match('.*session_state.*', str(location)):
        self.__config["code"] = parse.parse_qs(parse.urlsplit(location).query)["code"][0]
        self.__config["session_state"] = parse.parse_qs(parse.urlsplit(location).query)["session_state"][0]
        self.j = True
    else:
        raise error

def getToken(self):
    naglowki = {
        'Authorization': 'Basic TGlkbFBsdXNOYXRpdmVDbGllbnQ6c2VjcmV0',
        'Content-Type': 'application/x-www-form-urlencoded'
        }
    payload = {
        'grant_type': 'authorization_code',
        'code' : self.__config["code"],
        'redirect_uri' : self.__config["redirect_uris"],
        'code_verifier' : self.__config["code_verification"]
    }
    r = requests.post(self.ctx['token_endpoint'], data=payload, headers=naglowki)
    return(r.json()["refresh_token"])
```

*Źródło: opracowanie własne*

Nagłówek przekierowujący *Location* zawierający w adresie frazę *session_state* zostaje później przekształcony w słownik z którego wyciągany jest klucz *code* zawierający kod autoryzacyjny. Ten punkt kompletuje nam wszystkie dane potrzebne do uzyskania tokenu który później zostanie wykorzystany przy tworzeniu obiektu klasy Lidl.

Metoda getToken wysyła żądanie do punktu przyznającego tokeny, w ładunku znajduje się między innymi niezaszyfrowany kod utworzony w punkcie pierwszym.

### 2.2.4. Konteneryzacja warstwy aplikacyjnej.

W założeniach implementacji zawarta została łatwość przenoszenia aplikacji, pomocą w wykonaniu tego celu jest konteneryzacja. Aby aplikacja mogła działać w kontenerze musi zostać utworzony jej obraz według instrukcji zawartych we wcześniej wspomnianym dockerfile.

Sekwencja wykonanych poleceń zawartych w dockerfile utworzy obraz aplikacji, który później będzie mógł być przenoszony i uruchamiany na innych platformach.

*Rys.23 Plik dockerfile modułu grab_mails*

```
#Deklaracja bazowego obrazu
FROM python:3
#Ustawienie katalogu aplikacji
WORKDIR /app
#Skopiowanie pliku zależności
COPY requirements.txt ./
#Zainstalowanie zależności
RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt
#Kopiowanie pliku skryptu
COPY . .
CMD [ "python", "./grab_mails.py" ]
```

*Źródło: opracowanie własne*

Obrazem bazowym jest python w wersji 3, a domyślną ścieżką dla aplikacji jest katalog /app. Skopiowane zostają dwa pliki z dysku lokalnego, pierwszy zawierający listę modułów python potrzebną do uruchomienia skryptu, drugi to moduł do pobierania informacji z skrzynki pocztowej. Podczas wykonania komendy pip następuje aktualizacja oprogramowania oraz zainstalowanie modułów zawartych w pliku txt. Na koniec deklarowana jest domyślna komenda uruchamiana przez kontener.

Plik dockerfile, oraz wszystkie pliki które są zdefiniowane w pliku powinny znajdować się w katalogu w którym zostanie wykonane budowanie obrazu. Po zbudowaniu można przystąpić do testowego uruchomienia kontenera.

Aplikacja zakłada dodatkowe parametry uruchomieniowe które muszą zostać uwzględnione podczas startu. Moduł będzie komunikował się z plikiem bazy danych

umieszczonym na dysku lokalnym oraz dodatkowo przekazane muszą zostać między innymi dane logowania do serwera pocztowego.

Dane logowania zostaną przekazane jako zmienne środowiskowe, natomiast dla pliku bazy danych zostanie utworzone specjalne dowiązanie, co ilustruje poniższy przykład :

```
docker run -e BP_USER=adresemail -e BP_PASS=password -v
/home/app/database.db:/app/database.db grabmail
```

Zmienne środowiskowe BP_USER oraz BP_PASS będą przekazywane do środowiska wykonawczego, po stronie modułu natomiast musi znaleźć się odwołanie do podanych wartości. Dokonać można tego za pomocą biblioteki os oraz obiektów mapujących zmienne.

*Rys.24 Mapowanie zmiennych środowiskowych*

```
import os

username = os.environ['BP_USER']
password = os.environ['BP_PASS']
```

*Źródło: opracowanie własne*

Dowiązanie pliku /home/app/database.db z zasobu lokalnego będzie natomiast widoczne w kontenerze pod adresem /app/database.db.

Do parametrów uruchomieniowych naszego kontenera warto dodać również zalecenia restartu w konkretnych sytuacjach. Flaga `--restart unless-stopped` wykona automatyczny restart kontenera w momencie gdy zakończy on niespodziewanie działanie.

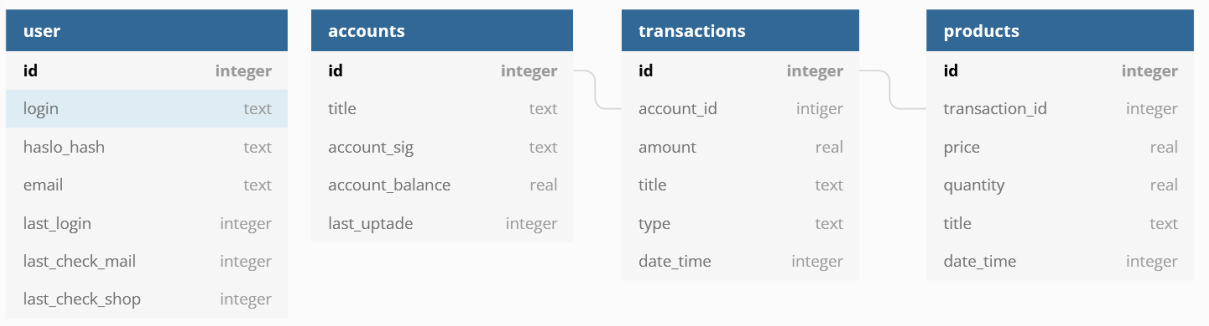
### 2.2.5. Baza danych SQLite

Ze względu na charakter aplikacji, gdzie docelowo dostęp ma mieć tylko jeden użytkownik oraz nie jest przewidywane wykonywanie jednoczesnych zapytań modyfikujących dane, została użyta baza danych typu SQLite. Dodatkową zaletą rozwiązania jest łatwość w przenoszeniu danych zawartych w bazie z racji przechowywania wszystkich informacji w jednym pliku.

Dane techniczne umieszczone są w tabeli *user,* gdzie oprócz loginu, hasła czy adresu email znajdziemy również informacje o ostatnim logowaniu czy 2 parametry wspomagające sprawdzanie wiadomości w skrzynce email czyli last_check_mail oraz last_check_shop. Z racji braku dedykowanego typu dla przechowywania czasu, wszystkie daty i godziny zapisywane są w postaci unix timestamp.

Tabla account zawiera informacje o stanie konta konkretnego dostawcy i jest odpowiednio aktualizowana w momencie przeprowadzenia transakcji. Tabela transactions zawiera informacje o przychodach i rozchodach z danego rachunku, natomiast tabela products zawiera informacje na temat produktów bądź usług realizowanych w ramach transakcji.

*Rys.25 Schemat bazy danych*




*Źródło: opracowanie własne*

### 2.2.6. Interfejs użytkownika

Realizacja interfejsu użytkownika opierała się na użyciu podstawowych dodatkowych bibliotek Flask, pozwalających na utworzenie systemu logowania, obsługi formularzy czy wyświetlania informacji odnośnie transakcji.

Wykonanie poszczególnych podstron bazowało na tym samym schemacie. Pierwszym elementem było utworzenie szablonu bazowego który później używany był na poszczególnych podstronach. Szablony są wykonane używając języka znaczników HTML wraz z udziałem symboli zastępczych {{ }} używanych do prezentowania zawartości dynamicznej.

*Rys.26 Szablon bazowy*

```
<!doctype html>
<html>
    <head>
        {% if title %}
        <title>{{ title }} - BPlan</title>
        {% else %}
        <title>BPlan</title>
        {% endif %}
    </head>
    <body>
        <div>
            {% if current_user.is_anonymous %}
            <a href="{{ url_for('login') }}">Zaloguj</a>
            {% else %}
            <a href="{{ url_for('index') }}">Home</a>
            <a href="{{ url_for('add) }}">Dodaj Transakcje</a>
            <a href="{{ url_for('recipt') }}">Paragony</a>
            <a href="{{ url_for('logout') }}">Logout</a>
            {% endif %}
        </div>
        {% block content %}{% endblock %}
    </body>
</html>
```

*Źródło: opracowanie własne*

Szablon bazowy będzie elementem każdej podstrony. W pierwszych liniach widzimy instrukcję warunkową która na podstawie przekazywanego podczas wykonywania parametru title ustawia tytuł strony. Kolejnym elementem jest wyświetlenie prostego menu w zależności od tego, czy użytkownik jest zalogowany. Ostatnim polem jest rezerwacja miejsca na blok zawartości który w późniejszych etapach będzie rozwijany o kolejne funkcjonalności.

*Rys.27 Szablon logowania*

```
{% extends "base.html" %}

{% block content %}
    <h1>Strona logowania</h1>
    <form action="" method="post" novalidate>
        {{ form.hidden_tag() }}
        <p>
            {{ form.login.label }}<br>
            {{ form.login(size=24) }}<br>
            {% for error in form.login.errors %}
            <span style="color: rgb(255, 0, 0);">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.haslo_hash.label }}<br>
            {{ form.haslo_hash(size=24) }}<br>
            {% for error in form.haslo_hash.errors %}
            <span style="color: rgb(255, 0, 0);">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>{{ form.submit() }}</p>
    </form>
{% endblock %}
```

*Źródło: opracowanie własne*

Kolejnym etapem będzie utworzenie formularza logowania do strony. Pierwsza linia informuje jaki szablon będzie rozszerzany. Oprócz standardowych znaczników HTML wyświetlających tytuł czy określających typ formularza możemy znaleźć odwołania do klasy Logowanie będącą rozszerzeniem FlaskForm która wprowadza nam podstawowe elementy formularza oraz ich walidację.

*Rys.28 Klasa Logowanie*

```
class Logowanie(FlaskForm):
    login = StringField('Login', validators=[DataRequired()])
    haslo_hash = PasswordField('Hasło', validators=[DataRequired()])
    wyslij = SubmitField('Zaloguj')
```

*Źródło: opracowanie własne*

Warto zwrócić uwagę na wywołanie form.hidden_tag(), jest to wbudowane zabezpieczenie przeciwko atakom CSRF (Cross-Site Request Forgery) powodujące dodanie do formularza losowego ciągu znaków powiązanego z obecną sesją.

*Rys.29 Obsługa URL /login*

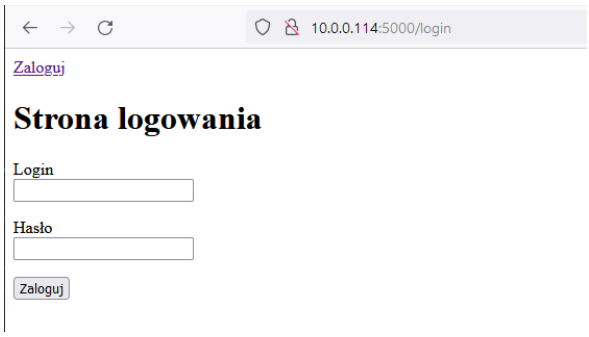
```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    #Użytkownik wchodzi na stronę login ale jest już zalogowany
    if current_user.is_authenticated:
        #Przenieś na index
        return redirect(url_for('index'))
    #Utwórz obiekt
    form = Logowanie()
    #Walidacja formularza
    if form.validate_on_submit():
        user = User.query.filter_by(login=form.login.data).first()
        #Jeżeli brak usera albo hash nie jest prawidłowy
        if user is None or not user.check_password(form.haslo_hash.data):
            flash('Nieprawidłowe doane')
            #Wróć na login
            return redirect(url_for('login'))
        #else zaloguj
        login_user(user)
        #Skąd user wchodził (podstrona czy na czysto)
        referer = request.args.get('next')
        if not referer or url_parse(referer).netloc != '':
            referer = url_for('index')
        #Przesnieś go tam gdzie zaczynał
        return redirect(referer)
    return render_template('login.html', title='Zaloguj', form=form)
```

*Źródło: opracowanie własne*

Opublikowanie podstrony polega na zastosowaniu dekoratora jakim jest @app.route. Dekorator wskazuje jakie akcje mają być wykonywane podczas uruchomienia podanego adresu URL. Dekorator zawiera również metody obsługiwane poprzez dany URL, w powyższym przykładzie użyte są metody GET oraz POST odpowiednio do wyświetlenia strony logowania jak i wysłania danych autoryzujących.

Dzięki zaimportowanej bibliotece flask_login mamy możliwość sprawdzenia czy użytkownik jest już zalogowany lub przeprowadzenia weryfikacji konta i zalogowania. W wyrażeniu *return* została użyta funkcja render_template w której wskazany został szablon, tytuł i obiekt formularza. Wynikiem tego wywołania będzie strona logowania.

*Rys.20 Strona logowania*




*Źródło: opracowanie własne*

Aby funkcjonalność logowania działała poprawnie, inne podstrony muszą zostać oznaczone dekoratorem @login_required aby wymusić sprawdzenie czy użytkownik jest zalogowany.

*Rys.31 Obsługa URL /index*

```python
@app.route('/')
@app.route('/index')
@login_required
def index():
    transactions = [
        {
            'bank': 'PKO BP',
            'type': 'INCOME',
            'title': 'Zakupy Lidl',
            'amount': '22,43',
        }

    ]
    return render_template('index.html', title='Strona Główna', transactions=transactions)
```

*Źródło: opracowanie własne*

Poprawnie zalogowany użytkownik zostanie przeniesiony na podstronę dla której wysłał żądanie, co zapewnia wynik przypisany do zmiennej *referer*. Dla przykładu URL index powinna zostać wygenerowana strona, w której zostaną pokazane dane występujące w specjalnie spreparowanej liście słowników *transactions*. Gwarantowane jest to poprzez szablon index w którym znajduje się pętla *for* która iteruje po dostarczonej w funkcji render_template liście.

*Rys.32 Szablon index*

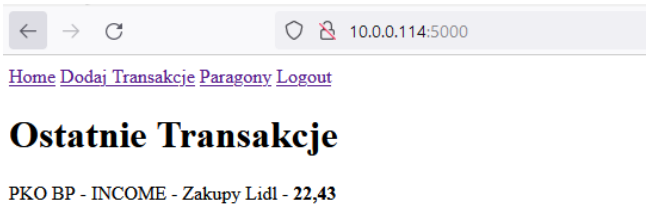
```
{% extends "base.html" %}

{% block content %}
    <h1>Ostatnie Transakcje</h1>
    {% for t in transactions %}
    <div><p>{{t.bank}} - {{t.type}} -  {{ t.title }} -  <b>{{ t.amount }}</b></p></div>
    {% endfor %}
{% endblock %}
```

*Źródło: opracowanie własne*

Wynikowo użytkownik dostanie prostą stronę HTML z wyświetlonymi informacjami jak i pełnym menu dla zalogowanego użytkownika.

*Rys.33 Strona główna aplikacji*




*Źródło: opracowanie własne*

### 2.2.7. Zabezpieczenia

Podstawowym zabezpieczeniem danych użytkownika jest system logowania gdzie w momencie autoryzacji porównywany jest skrót SHA256 hasła z tym zapisanym w bazie danych. Aplikacja nie przechowuje haseł w postaci jawnej, co zapobiega wrogiemu przejęciu danych.

Nginx dodatkowo zapewnia nam szyfrowanie ruchu protokołem HTTPS gdzie dla większego bezpieczeństwa możemy wymusić wykorzystywanie protokołu TLS 1.2 dodając dyrektywę ssl_protocols TLSv1.2, certyfikat SSL dla strony jest podpisany przez odpowiedni urząd certyfikacji co zapewnia nam certbox oraz usługa Let's Encrypt.

Oprócz zabezpieczenia samej aplikacji wymagane jest odpowiednie przygotowanie środowiska uruchomieniowego. Bez względu na system operacyjny zdefiniować można podstawy, które pozwolą na bezpieczne udostępnianie aplikacji. Przed wystawieniem aplikacji na świat administrator powinien zwrócić uwagę na poniższe aspekty:

- Zmiana portów domyślnych - bez względu na to jaka usługa służy nam do komunikacji z serwerem używanie domyślnych portów przyśpiesza zlokalizowanie dostępnych usług na serwerze. Używanie numerów portów powyżej 10000 utrudni osobie atakującej zlokalizowanie usług.
- Zamknięcie wszystkich portów z wyłączeniem tylko tych używanych do wystawienia aplikacji na świat. Istnieją różnego typu ataki blokujące przepływ sieciowy czy nieupoważniony dostęp do serwera. Blokada portów usług niemających wpływu na działanie aplikacji pozwoli na bezpieczniejsze serwowanie zasobów. Przykładem ataku na jedną z podstawowych usług systemowych blokującego ruch sieciowy jest atak NTP flood wysyłający ogromne ilości pakietów UDP z dowolnych serwerów czasu.
- Włączenie autoryzacji dwuetapowej do serwera z wykorzystaniem fizycznego klucza fizycznego FIDO U2F bądź wyłączenie autoryzacji bazującej na haśle dostępu i użycia kluczy RSA. Wraz w udostępnieniem pakietu OpenSSH w wersji 8.2 dla serwerów linuxowych oraz unixowych możliwe jest użycie fizycznego klucza FIDO U2F do logowania po protokole SSH co stanowi jedną z najbezpieczniejszych metod autoryzacji.

# 3. Podsumowanie

Głównym celem pracy było przedstawienie procesu implementacji aplikacji złożonej z kilku komponentów wraz z wyjaśnieniem ich roli i zaprezentowaniem rozwiązania. W tym celu został przedstawiony wykonany wstępny projekt w którym pojawiły się technologie i oprogramowanie pozwalające na zbudowanie aplikacji.

Praca zawiera się w 3 rozdziałach, gdzie pierwszy z nich stanowi wstęp w którym zostały przedstawione aktualnie dostępne narzędzia na rynku wraz z ich analizą pod kątem użyteczności, w dalszej części rozdziału zostały wyróżnione założenia które musi spełnić aplikacja mogąca być konkurencyjna względem innych. Rozdział drugi stanowi opis technologii użytych do implementacji jak i prezentacje rozwiązań użytych w modułach. Trzeci rozdział jest podsumowaniem wykonanych prac a w jego dalszej części wykonana zostanie weryfikacja założeń funkcjonalnych przedstawionych na początku pracy jak i przemyśleniom dotyczącym dalszego rozwoju aplikacji. Koniec pracy ukaże problemy napotkane podczas implementacji bądź projektowania.

## 3.1. Weryfikacja założeń

Wraz z rozpoczęciem projektu jako cel zostały obrane 3 rzeczy. Pierwsza z nich, czyli możliwość automatycznego pobierania danych została zrealizowana wykorzystując powiadomienia wysyłane pocztą elektroniczną. Łatwość dodawania schematów pobierania danych czyni to rozwiązanie dość uniwersalnym natomiast wymaga na użytkowniku włączenia tej opcji dla swojego konta bankowego. Znaczącym utrudnieniem jest dodatkowa autoryzacja dla klienta poczty u największych dostawców poczty internetowej taka jak SASL XOAUTH2, która musi zostać wyłączona aby móc połączyć się bezpośrednio do serwera IMAP.

Kolejnym założeniem była łatwość instalacji i przenoszenia aplikacji względem różnych platform. Realizacja oparta na konteneryzacji pozwala w łatwy sposób zapewnić tą funkcjonalność jednak wymaga na użytkowniku podstawowej znajomości obsługi środowiska docker co stanowi sukces połowiczny.

Ostatni cel narzucony aplikacji dotyczył dużej ilości rozszerzeń pozwalających na pobieranie informacji o transakcjach. Wraz z wdrożeniem modułu obsługującego

powiadomienia mailowe użytkownik może opracować kolejne wyrażenia regularne, które posłużą do pobierania informacji o przepływach pieniężnych. Zbieranie danych na temat przedmiotu transakcji zostało zaprojektowane jako osobny moduł analizujący transakcje pod kątem źródła, uruchamiany cyklicznie i dopisujący znalezione informacje do bazy. Ze względu na znikomą popularność e-paragonów praca ogranicza się tylko do jednego dostawcy, jednak pozwala na późniejsze rozwijanie rozwiązań poprzez kolejne moduły działające na tej samej zasadzie poprzez uruchomienie dodatkowego kontenera i dodaniu kolumny znacznika czasu w tabeli users.

## 3.2. Plan rozbudowy

Pierwszym punktem rozbudowy aplikacji jest jej interfejs który w pracy został zaprezentowany jedynie w minimalnej formie. Wymagać to będzie zapoznania się z dodatkowymi bibliotekami takimi jak flask_bootstrap lub opracowaniem własnych nowych szablonów HTML. Odległym planem jest również przekształcenie aplikacji przeglądarkowej w aplikację progresywną jednak to wymagać będzie większego nakładu czasu i nauki.

Drugim punktem który powinien zostać zrealizowany aby dopełnić założenia aplikacji jest utworzenie obrazów nginx oraz certbox certbox wraz z poprawną obsługą Let’s Encrypt, pozwoliło by to na całkowite skonteneryzowanie aplikacji oraz uproszczenie procesu instalacji dzięki narzędziom do uruchamiania aplikacji multi-kontenerowych jakim jest między innymi docker compose. Wykonanie aplikacji w taki sposób pozwoli na uproszczenie większości elementów konfiguracji oraz zapewni lepszą dostępność aplikacji jak i poprawi jej bezpieczeństwo.

Ostatni punkt rozbudowy stanowić będzie wykonanie kolejnych modułów zbierających dane o produktach lub usługach nabytych u różnych dostawców, jest to jednak bardzo mocno uzależnione od samych dostawców jak i technologii przez nich używanych.

## 3.3. Problemy i ograniczenia

Główny problem stanowił brak popularności rozwiązania jakim są e-paragony bądź ich realizacja. Sposób udostępniania informacji poprzez aplikację LidlPlus pozwalały w prosty sposób na wyciągnięcie interesujących nas danych. Przeciwieństwem były e-paragony

firm obsługujących przewozy osobowe, gdzie pomimo udostępniania e-paragonu były one wysyłane w postaci obrazu wymagającego dodatkowego przetwarzania aby móc wprowadzić te dane do tabeli w postaci tekstu. Umieszczanie paragonów jako obiekt BLOB znacząco utrudniałoby weryfikowanie wydatków.

Kolejnym ograniczeniem jest dostęp do danych bankowych gdzie zwykły użytkownik nie jest w stanie pobrać czystych danych na temat swojego budżetu, jest to oczywiście podyktowane względami bezpieczeństwa.

# Bibliografia


1. Wray, L. Randall. "Powstanie pieniądza i społeczeństwa klasowego: pisma Johna F. Henry'ego." Czas Kultury, 2021 pp. 14-22.
2. Lewiński, Jan Stanisław. "Powstanie pieniądza." (1930).
3. Frączek, Bożena. "Pieniądz elektroniczny-próba zdefiniowania i sklasyfikowania." Bank i Kredyt 4 (2004): 91-95.
4. Szpringer, Włodzimierz, oraz Mariusz Szpringer. "Nowe zjawiska w regulacji rynku usług płatniczych (wybrane problemy na tle projektu noweli do dyrektywy PSD)." e-mentor 4 (56) (2014): 73-83.
5. Paduszyńska, Marta, oraz Bartłomiej Pawlak. "Rynek usług płatniczych w Polsce w świetle zmian prawnych implementujących postanowienia dyrektywy PSD2." Studia Prawno-Ekonomiczne 114 (2020): 333-349., DOI: 10.26485/SPE/2020/114/18
6. Sayal, M. A., M. H. Alameady, and S. A. Albermany. "The Use of SSL and TLS Protocols in Providing a Secure Environment for e-commerce Sites." Webology 17.2 (2020). DOI: 10.14704/WEB/V17I2/WEB17048
7. Anderson, Charles. "Docker [software engineering]." Ieee Software 32.3 (2015): 102-c3, DOI: 10.1109/MS.2015.62
8. Fortunato, David, and Jorge Bernardino. "Progressive web apps: An alternative to the native mobile Apps." 2018 13th Iberian Conference on Information Systems and Technologies (CISTI). IEEE, 2018, DOI: 10.23919/CISTI.2018.8399228
9. Rad, Babak Bashari, Harrison John Bhatti, and Mohammad Ahmadi. "An introduction to docker and analysis of its performance." International Journal of Computer Science and Network Security (IJCSNS) 17.3 (2017): 228
10. Ghimire, Devndra. "Comparative study on Python web frameworks: Flask and Django." (2020).
11. Anand, Vic, et al. "Using Python for text analysis in accounting research." Vic Anand, Khrystyna Bochkay, Roman Chychyla and Andrew Leone (2020)," Using Python for Text Analysis in Accounting Research", Foundations and Trends® in Accounting 14.3-4 (2020): 128-359, DOI: 0.1561/1400000062

# Bibliografia internetowa oraz data dostępu:


1. Ułatwienia w e-paragonach, https://www.gov.pl/web/gov/ulatwienia-w-e-paragonach, dostęp 2021-09-10
2. Cozy Cloud - le domicile numérique pour réunir toutes vos données, https://cozy.io/fr/, dostęp 2021-09-10
3. Money Lover, https://moneylover.me/, dostęp 2021-09-10
4. A better way to handle your money | Revolut PL, https://www.revolut.com/pl-PL, dostęp 2021-09-10
5. Revolut podbija Polskę. Nasz kraj jest dla fintechu drugim najszybciej rosnącym rynkiem w Europie, https://businessinsider.com.pl/technologie/nowe-technologie/revolut-w-polsce-liczba-uzytkownikow/rz9fr3m, Business Insider, 11 wrz 18, dostęp 2021-09-10
6. Nginx, http://nginx.org/en/, dostęp 12.09.2021
7. NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy, https://www.nginx.com/products/nginx/, dostęp 12.09.2021
8. Nginx vs. Apache usage statistics, September 2021, https://w3techs.com/technologies/comparison/ws-apache,ws-nginx, dostęp 15.09.2021
9. Docker Overview | Docker Documentation, https://docs.docker.com/get-started/overview/, dostęp 15.09.2021
10. SQLite Home Page, https://www.sqlite.org/index.html, dostęp 15.09.2021
11. WebDriver | Selenium, https://www.selenium.dev/documentation/webdriver/, dostęp 15.09.2021
12. NTP amplification DDoS attack, https://www.cloudflare.com/learning/ddos/ntp-amplification-ddos-attack/, dostęp 15.09.2021
13. Welcome to Flask - Flask Documentation 2.0.x, https://flask.palletsprojects.com/en/2.0.x/, dostęp 15.09.2021
14. The Flask Mega-Tutorial Part I: Hello, World!, https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world, dostęp 15.09.2021
15. IMAP, POP and SMTP | Gmail for Developpers, https://developers.google.com/gmail/imap/imap-smtp, dostęp 15.09.2021
16. Overview od docker copose | Docker documentation, https://docs.docker.com/compose/, dostęp 15.09.2021

# Spis rysunków

29. Obsługa URL /login - opracowanie własne
30. Strona logowania - opracowanie własne
31. Obsługa URL /index - opracowanie własne
32. Szablon index - opracowanie własne
33. Strona główna aplikacji - opracowanie własne