WOJSKOWA AKADEMIA TECHNICZNA
im. Jarosława Dąbrowskiego

WYDZIAŁ CYBERNETYKI

Bazy danych No SQL

Sprawozdanie z zadania laboratoryjnego

Autorzy:
sierż. pchor. Aneta PODSTAWKA
sierż. pchor. Michał ZIMOŃ
plut. pchor. Jan Andrzejewski
Grupa: **K9B1S4**

Prowadzący:
kpt. mgr inż. Maciej Szymczyk

Warszawa 2019

Spis treści

1 Zadanie **3**
1.1 Wybrany obszar dziedzinowy . . . 3

2 Opis baz **5**
2.1 MongoDB . . . 5
2.1.1 Kolekcja users . . . 5
2.1.2 Kolekcja accounts . . . 6
2.1.3 Kolekcja cards . . . 6
2.1.4 Przykładowe wypełnienie każdej kolekcji . . . 6
2.2 Cassandra . . . 7
2.2.1 transaction_by_id . . . 7
2.2.2 transaction_by_source . . . 8
2.2.3 transaction_by_target . . . 8
2.2.4 Przykładowe wypełnienie każdej tabeli . . . 8
2.3 Redis . . . 9
2.3.1 Przykładowe wartości w bazie, podczas działania aplikacji . . . 9
2.4 Elasticsearch . . . 10
2.4.1 Nowa rejestracja . . . 10
2.4.2 Nowa transakcja . . . 10
2.4.3 Nowe logowanie . . . 10
2.4.4 Przykładowe logi . . . 11
2.5 Wykorzystanie baz . . . 11

3 Implementacja **13**
3.0.1 MongoDB . . . 13
3.0.2 Cassandra . . . 14
3.0.3 Elasticsearch . . . 15
3.0.4 Redis . . . 16

4 Podsumowanie **18**
4.1 Wady i zalety . . . 18
4.1.1 MongoDB . . . 18
4.1.2 Cassandra . . . 18
4.1.3 Redis . . . 19
4.1.4 Elasticsearch . . . 19
4.2 Napotkane problemy i ich rozwiązanie . . . 20
4.3 Pomysły i idee . . . 20

Bibliografia **21**

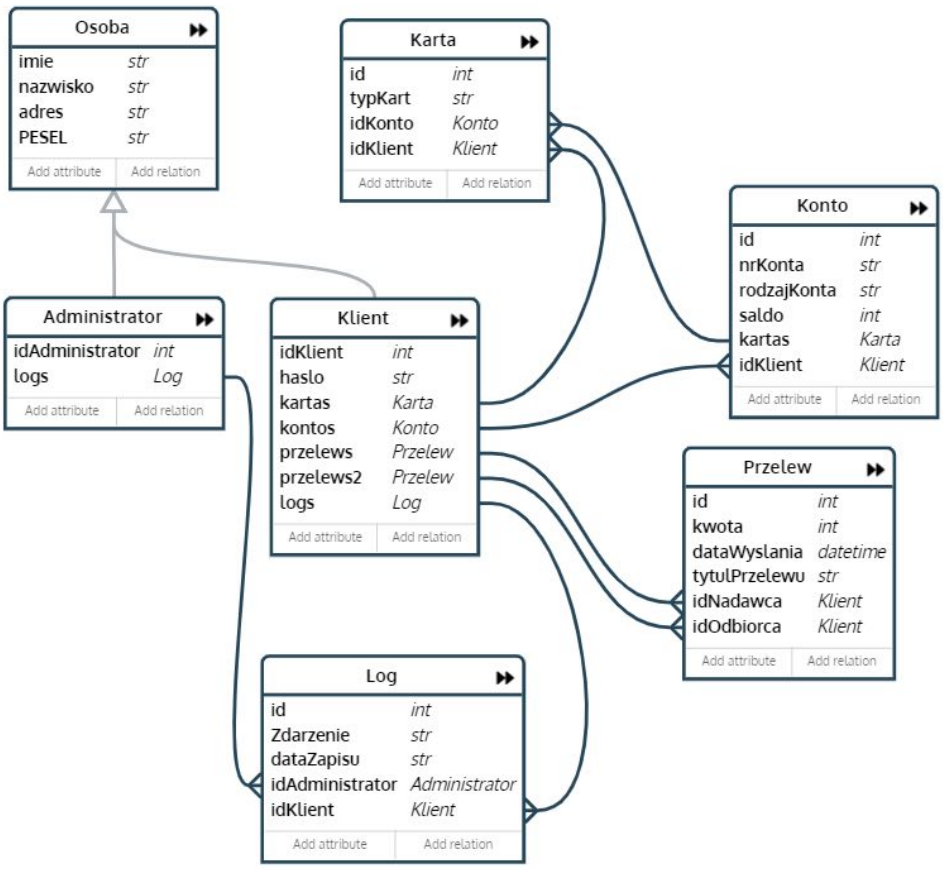
1 Zadanie

Celem zadania projektowego, było wybranie i przedstawienie obszaru dziedzinowego, z opisem systemu informatycznego i projektu relacyjnej bazy danych dla niego.

Następnie należało zidentyfikować i wyznaczyć obszary systemu, które mogą zostać przeniesione na bazy NoSQL.

1.1 Wybrany obszar dziedzinowy

Grupa wybrała bank jako obszar dziedzinowy. Stworzony diagram ERD przedstawia się nastepująco:



Rysunek 1: Diagram ERD. *Źródło: opracowanie własne*

Obszary systemu zostały przeniesione na bazy NoSQL w nasępujący sposób:

1. MongoDB
    a) Przechowywanie danych o klientach,
    b) Przechowywanie danych o kontach,
    c) Przechowywanie danych o kartach.
2. Cassandra
    a) Przechowywanie danych o przelewach.
3. Redis
    a) Sesja użytkownika,
    b) Kolejka podczas rejestracji.
4. Elasticsearch
    a) Przechowywanie logów.

2 Opis baz

W każdej bazie stworzono elementy, pozwalające jak najlepiej zrealizować postawione zadanie i pokazać możliwości każdej z nich.

Dla bazy MongoDB i Cassandra został stworzony skrypt, wypełniający je żądaną ilością danych. Pozwala to na dodatkowe wygenerowanie logów dla Elasticsearch oraz przedstawienie działania mechanizmu kolejki w Redis'ie.

2.1 MongoDB

Serwer został postawiony na maszynie lokalnej i na cele projektu utworzono zwykłą bazę, a w niej trzy kolekcje:

- users
- accounts
- cards

Dane do kolekcji *users* użytkownik podaje podczas rejestracji. Nie wszystkie z nich są niezbędne do działania aplikacji. Dane do kolekcji *accounts* i *cards* są generowane automatycznie. Podczas rejestracji użytkownikowi tworzone jest automatycznie konto główne, natomiast karę może dodać później.

2.1.1 Kolekcja users

Przechowywane są tutaj informacje o użytkowniku, umożliwiające realizację zadania. Można tu znaleźć następujące pola:

- **_id**: numer PESEL, który jest unikalny
- **name**: pole opcjonalne, zawierające imię użytkownika
- **surname**: pole opcjonalne, zawierające nazwisko użytkownika
- **gender**: pole opcjonalne, zawierające płeć użytkownika
- **password**: pole wymagane, zawierające hasło użytkownika
- **email**: pole wymagane, zawierające email użytkownika
- **accounts**: pole zawierające numery kont użytkownika
- **membership**: pole zawierające status użytkownika
- **phone**: pole opcjonalne, zawierające numer telefonu użytkownika
- **created_at**: pole generowane automatycznie, zawierające datę rejestracji

2.1.2 Kolekcja accounts

Przechowywane są tutaj informacje o kontach użytkowników. Można tu znaleźć następujące pola:

- **_id**: numer konta, który jest unikalny
- **type**: pole zawierające typ konta (główne lub oszczędnościowe)
- **balance**: pole, zawierające stan konta
- **cards**: pole zawierające numery kart, przypisane do konta
- **created_at**: pole generowane automatycznie, zawierające datę utworzenia konta

2.1.3 Kolekcja cards

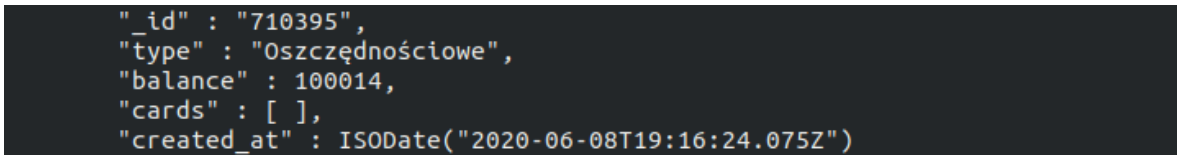
Przechowywane są tutaj informacje o kartach bankowych. Można tu znaleźć następujące pola:

- **_id**: numer karty, który jest unikalny
- **limit**: pole zawierające limit karty
- **activated**: pole zawierające informację, czy karta jest aktywowana
- **expiry_date**: pole zawierające datę ważności karty
- **CVV**: pole zawierające kod CVV

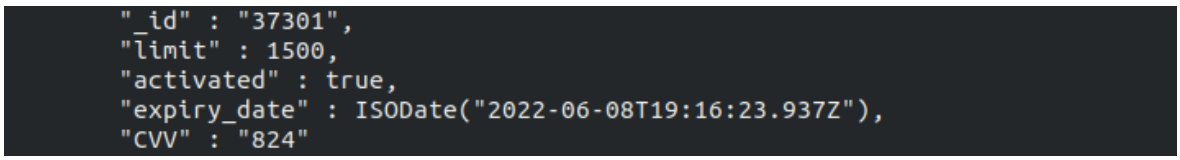
2.1.4 Przykładowe wypełnienie każdej kolekcji


```
"_id" : "72072027746",
"name" : "Deshawn",
"surname" : "BOUZIGARD",
"gender" : "Mężczyzna",
"password" : "rMSqfksr",
"email" : "JTrzA@mail.com",
"accounts" : [
        "760761"
],
"membership" : "silver",
"phone" : "817736377",
"created_at" : "2020-05-04"
```


Rysunek 2: Przykładowe wypełnienie kolekcji users. *Źródło: opracowanie własne*



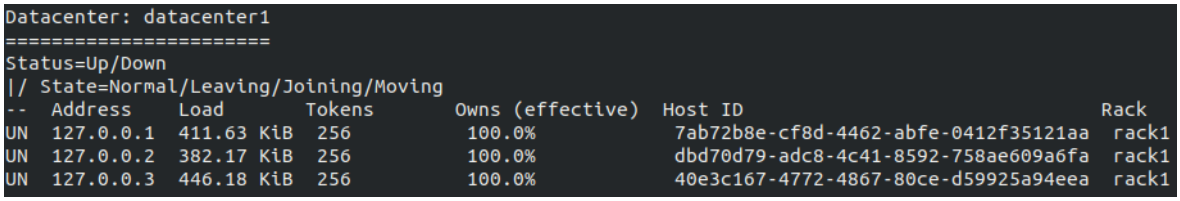
Rysunek 3: Przykładowe wypełnienie kolekcji accounts. *Źródło: opracowanie własne*



Rysunek 4: Przykładowe wypełnienie kolekcji cards. *Źródło: opracowanie własne*

2.2 Cassandra

Baza została postawiona na maszynie lokalnej. Aby zaprezentować dodatkowe możliwości, zostały stworzone trzy node'y i bazę stworzono ze współczynnikiem replikacji równym trzy. Można wtedy pokazać niezawodność tej bazy, jeśli jeden z node'ów "umrze", to dostęp do danych pozostaje na "żywych" node'ach.



Rysunek 5: Obecność 3 node'ów. *Źródło: opracowanie własne*

Aby wykorzystać szybkość Cassandry, zostały stworzone trzy tabele:

- transaction_by_id
- transaction_by_source
- transaction_by_target

2.2.1 transaction_by_id

W tabeli przechowywane są informacje na temat przelewów wykonywanych przez użytkowników. Składa się ona z:

- **transaction_id**: numer transakcji liczony od 1 (PRIMARY KEY)
- **amount**: kwota przelewu
- **source**: numer konta, z którego został wykonany przelew
- **target**: numer konta, na który został wykonany przelew

- **title**: tytuł przelewu
- **transaction_date**: data transakcji

2.2.2 TRANSACTION_BY_SOURCE

W celu szybszego wyszukiwania np. wszystkich transakcji wychodzących danego użytkownika w danym przedziale czasowym została utworzona omawiana tabela, która zawiera

- **source**: numer konta, z którego wykonano przelew (część PRIMARY KEY)
- **transaction_date**: data transakcji (część PRIMARY KEY)
- **transaction_id**: numer transakcji

2.2.3 TRANSACTION_BY_TARGET

W celu szybszego wyszukiwania np. wszystkich transakcji przychodzących danego użytkownika w danym przedziale czasowym została utworzona omawiana tabela, która zawiera

- **target**: numer konta, na który wykonano przelew (część PRIMARY KEY)
- **transaction_date**: data transakcji (część PRIMARY KEY)
- **transaction_id**: numer transakcji

2.2.4 Przykładowe wypełnienie każdej tabeli

transaction_id	amount	source	target	title	transaction_date
23	911	486821	149049	tRUhwd	2020-02-04
114	19	288757	887195	RLVMnV	2020-01-15
53	256	121856	691754	boKMOJ	2020-05-11
110	919	515520	879748	tlCHwb	2020-04-10
91	536	584063	458849	BtfLoJ	2020-02-06

Rysunek 6: Przykładowe wypełnienie tabeli transaction_by_id. *Źródło: opracowanie własne*

source	transaction_date	transaction_id
760761	2020-02-12	456
760761	2020-03-15	480
760761	2020-04-08	420
674952	2020-01-17	490

Rysunek 7: Przykładowe wypełnienie tabeli transaction_by_source. *Źródło: opracowanie własne*

```
target | transaction_date | transaction_id
-------+------------------+---------------
760761 |       2020-03-07 |            171
674952 |       2020-01-17 |            380
674952 |       2020-02-07 |            307
674952 |       2020-05-19 |            206
213571 |       2020-03-09 |            412
```

Rysunek 8: Przykładowe wypełnienie tabeli transaction_by_target. *Źródło: opracowanie własne*

2.3 Redis

Baza została postawiona na maszynie lokalnej. Została wykorzystana na dwa sposoby:

1. Sesja użytkownika po zalogowaniu, gdzie kluczem jest e-mail użytkownika, a maksymalny czas sesji (jeśli niewykonywane są żadne operacje) to 5 minut.
2. Mechanizm kolejek. W przypadku projektu, jeśli rejestracja jednego użytkownika jeszcze się nie zaczęła, a już kolejny użytkownik się zarejestrował, to rejestracja drugiego jest dodawana do kolejki Redis'a i oczekuje na zakończenie pierwszej rejestracji.

2.3.1 Przykładowe wartości w bazie, podczas działania aplikacji

```
127.0.0.1:6379> ttl JTrzA@mail.com
(integer) 255
```

Rysunek 9: Czas sesji po zalogowaniu użytkownika JTrzA@mail.com. *Źródło: opracowanie własne*

```
INFO:rq.worker:Worker rq:worker:6a2de62f4cf748c2a3654a49fcaa5e7e: started, versi
on 1.3.0
INFO:rq.worker:*** Listening on default...
INFO:rq.worker:Cleaning registries for queue: default
INFO:rq.worker:default: database_code.functions_mongo.mongo_insert('97867586758'
, 'Gerard', 'Miloszewski', 'Mężczyzna', 'gerard123', 'gerard@gmail.com', '765766
586', '212312', datetime.datetime(2020, 6, 9, 21, 54, 17, 501170)) (d8762b15-561
b-4ffb-a7f0-46d853d7e09f)
Zaczynam czekac
Dodaje
INFO:rq.worker:default: Job OK (d8762b15-561b-4ffb-a7f0-46d853d7e09f)
INFO:rq.worker:Result is kept for 500 seconds
INFO:rq.worker:default: database_code.functions_mongo.mongo_insert('86758675864'
, 'Aneta', 'Podstawka', 'Kobieta', 'aneta123', 'aneta@gmail.com', '576856758', '
387234', datetime.datetime(2020, 6, 9, 21, 54, 43, 459846)) (8f3f799c-54d4-4ec2-
a3ed-89bedebf1607)
Zaczynam czekac
Dodaje
INFO:rq.worker:default: Job OK (8f3f799c-54d4-4ec2-a3ed-89bedebf1607)
INFO:rq.worker:Result is kept for 500 seconds
```

Rysunek 10: Wykorzystanie kolejki podczas rejestracji. *Źródło: opracowanie własne*

2.4 Elasticsearch

Baza została postawiona na maszynie lokalnej. Został stworzony *index* o nazwie *treebank*. Logi zbierane są dla trzech rodzajów operacji:

- rejestracja
- przelew
- logowanie

2.4.1 Nowa rejestracja

Log z nowych rejestracji zostaje przesłany do Elasticsearch'a z następującymi informacjami:

- **operation**: nazwa operacji
- **email**: nowo zarejestrowany e-mail
- **account_number**: numer konta, przypisany nowemu użytkownikowi
- **date**: data operacji

2.4.2 Nowa transakcja

Log z nowej transakcji zostaje przesłany do Elasticsearch'a z następującymi informacjami:

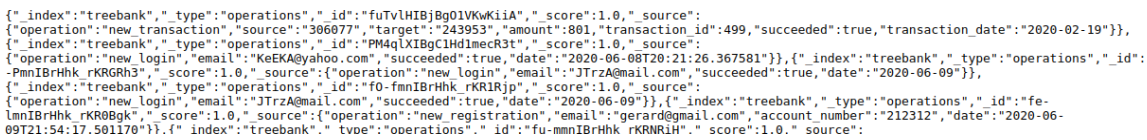
- **operation**: nazwa operacji
- **source**: numer konta, z którego został wysłany przelew
- **target**: numer konta, na który został wykonany przelew
- **amount**: kwota przelewu
- **succeeded**: czy przelew był udany
- **transaction_date**: data transakcji

2.4.3 Nowe logowanie

Log z logowania zostaje przesłany do Elasticsearch'a z następującymi informacjami:

- **operation**: nazwa operacji
- **email**: e-mail użytkownika, który się loguje
- **succeeded**: czy logowanie się powiodło
- **date**: data logowania

2.4.4 Przykładowe logi


{"_index":"treebank","_type":"operations","_id":"fuTvlHIBjBgO1VKwKiiA","_score":1.0,"_source":
{"operation":"new_transaction","source":"306077","target":"243953","amount":801,"transaction_id":499,"succeeded":true,"transaction_date":"2020-02-19"}},
{"_index":"treebank","_type":"operations","_id":"PM4qlXIBgC1Hd1mecR3t","_score":1.0,"_source":
{"operation":"new_login","email":"KeEKA@yahoo.com","succeeded":true,"date":"2020-06-08T20:21:26.367581"}},{"_index":"treebank","_type":"operations","_id":"e-
-PmnIBrHhk_rKRGRh3","_score":1.0,"_source":{"operation":"new_login","email":"JTrzA@mail.com","succeeded":true,"date":"2020-06-09"}},
{"_index":"treebank","_type":"operations","_id":"fO-fmnIBrHhk_rKR1Rjp","_score":1.0,"_source":
{"operation":"new_login","email":"JTrzA@mail.com","succeeded":true,"date":"2020-06-09"}},{"_index":"treebank","_type":"operations","_id":"fe-
lmnIBrHhk_rKR0Bgk","_score":1.0,"_source":{"operation":"new_registration","email":"gerard@gmail.com","account_number":"212312","date":"2020-06-
09T21:54:17.501170"}},{"_index":"treebank","_type":"operations","_id":"fu-mmnIBrHhk_rKRNRiH","_score":1.0,"_source":

Rysunek 11: Przykładowe logi przesłane do Elasticsearch'a. *Źródło: opracowanie własne*

2.5 Wykorzystanie baz

W celu realizacji założeń projektu, została stworzona aplikacja łącząca się z wcześniej wspomnianymi bazami. Jej możliwości, wraz z wykorzystanymi bazami przedstawiają sie nastepująco:

1. Rejestracja użytkownika:
   a) sprawdzenie, czy użytkownik z wprowadzonym mailem lub peselem nie jest już zarejestrowany (MongoDB),
   b) ustawienie procesu rejestracji na 60 sek., aby wykorzystać mechanizm kolejki w Redisie (Redis),
   c) podczas rejestracji tworzone jest nowe konto bankowe dla użytkownika, również sprawdzane jest czy nie jest już w bazie (MongoDB),
   d) utworzone konto zostaje dopisane do listy kont użytkownika (użytkownik może mieć max 2 konta) (MongoDB),
   e) poprawna rejestracja zostaje zapisana w logu i dodana do wykonanych operacji w Elasticsearch'u (Elasticsearch).
2. Logowanie na konto użytkownika:
   a) sprawdzenie, czy wprowadzony e-mail do logowania istnieje w bazie, czyli czy użytkownik istnieje (MongoDB),
   b) sprawdzenie, czy wprowadzone hasło zgadza się z tym w bazie dla danego konta (MongoDB),
   c) jeśli wprowadzone dane do logowania są poprawne to następuje logowanie i przesłanie logu o poprawnym logowaniu do Elasticsearch'a (Elasticsearch),
   d) jeśli wprowadzone hasło jest niepoprawne, do Elasticsearcha zostaje przesłany log o próbie logowania, zakończonej niepowodzeniem, poprzez wprowadzenie złego hasła (Elasticsearch).
3. Panel użytkownika:
   a) wyświetlanie czasu sesji, czyli ile czasu zostało użytkownikowi zanim zostanie wylogowany, poprzez brak aktywności na koncie (Redis)
   b) każdorazowa akcja na koncie, poprzez naciśnięcie jakiegoś guzika odświeża czas sesji (Redis),
   c) wyświetlenie ostatnich sześciu transakcji danego użytkownika (Cassandra).

4. Wykonywanie przelewów:
    a) sprawdzenie, czy użytkownik wykonujący przelew ma wystarczająco środków na koncie (MongoDB),
    b) jeśli jest wystarczająco środków na koncie przelew zostaje zapisany w Cassandrze (Cassandra),
    c) stany kont są odpowiednio aktualizowane na obu kontach (MongoDB),
    d) powodzenie lub niepowodzenie przelewu zostaje zapisane w logach Elasticsearcha (Elasticsearch).
5. Wyświetlanie historii przelewów:
    a) możliwość wyświetlenia dowolnej liczby ostatnio wykonanych przelewów (wychodzące) (Cassandra),
    b) możliwość wyświetlenia dowolnej liczby ostatnio wykonanych przelewów (przychodzące) (Cassandra),
    c) możliwość wyświetlenia dowolnej liczby ostatnio wykonanych przelewów (wszystkie) (Cassandra),
    d) możliwość nałożenia filtrów na historię, np. przedział czasowy, zakres kwot (Cassandra).
6. Dodawanie konta bankowego do konta użytkownika:
    a) utworzenie nowego konta oszczędnościowego, jeśli użytkownik jeszcze go nie posiada (użytkownik może mieć maksymalnie 2 konta) i przypisanie go do konta użytkownika (MongoDB),
    b) przesłanie logów o dodaniu konta do Elasticsearch'a (Elasticsearch).
7. Dodawanie kart dla kont bankowych:
    a) przypisanie karty do odpowiedniego konta (MongoDB),
    b) dodanie karty zostaje odnotowane w logach i zapisane w Elasticsearch'u (Elasticsearch).
8. Panel administratora:
    a) możliwość odczytu logów z Elasticsearch'a z różnymi filtrami (Elasticsearch),
    b) jeśli administrator pomyli się w e-mailu który chce wyszukać, to zostaną wyświetlone ewentualne podpowiedzi (Elasticsearch).

Dokładna prezentacja wykorzystania baz jak ich działanie, zostały przedstawione na filmiku umieszczonym na serwisie YouTube [1]

3 Implementacja

Cały kod źródłowy wraz z dokładnym opisem opisem każdej z funkcji znajduje się w repozytorium [2] na GitHub'ie:

- w plikach rozpoczynających się na *functions* oraz *generate* zawarte są kody źródłowe odpowiadające za komunikację aplikazji z każdą z baz oraz kody źródłowe do generacji danych, aby wypełnić bazy.
- w *gui.py* zawarty jest kod źródłowy, odpowiadający za GUI oraz wykorzystanie powyższych funkcji w aplikacji

Poniżej zostaną przedstawione wybrane funkcje, dla każdej z baz.

3.0.1 MongoDB

Poniższy kod przedstawia funkcję, odpowiadającą za rejestrację nowego użytkownika. Do funkcji dostarczany jest PESEL, imię, nazwisko, płeć, hasło, e-mail oraz numer telefonu, które podaje użytkownik podczas wypełniania formularza rejestracyjnego.

Jako że PESEL i e-mail muszą byc unikalne, funkcja sprawdza przede wszystkim, czy żane z nich nie istnieje już w bazie. Jeśli któraś z wartości istnieje, to odpowiednia wartość zostaje zwrócona, aby poinformować o tym uzytkownika. Jeśli nie, to można zlecić dodanie nowych danych do bazy.

Został tu wykorzystany mechanizm kolejki Redis'a. Zostaje wywołana funkcja *mongo_insert*, która odpowiada za wprowadzenie danych do bazy. Proces rejestracji został ustawiony na minutę, aby zaprezentować działanie kolejki. Oznacza to, że jeżeli jeden użytkownik się zarejestruje, jego dane zaczną być wprowadzane, a w tym czasie drugi użytkownik dokona rejestracji, to dane drugiego zostaną wprowadzone dopiero po zwolnieniu miejsca w kolejce.

Na koniec zostaje przesłany log do Elasticsearch'a o nowej rejestracji, i zostaje zwrócona odpowiednia wartość, aby poinformować użytkownika o pomyślnej rejestracji.

Kod 1: Funkcja *register* z *functions_mongo.py*. *Źródło: opracowanie własne*

```
"""
Rejestracja nowego uzytkownika
"""
def register(pesel, name, surname, gender, password,
    email, phone):
      # sprawdzenie, czy uzytkownik o podanym PESELu lub
         emailu jest juz w bazie
      pesel_exists = bool(users_collection.find_one({"_id
         ": pesel}))
      email_exists = bool(users_collection.find_one({"
         email": email}))

      # jesli nie to mozna zlecic wprowadzenie
         uzytkownika do bazy
      if not pesel_exists and not email_exists:
          created_at = datetime.now()
          account_number = add_account("Glowne")
```

```
        # wykorzystanie kolejki w Redisie
        q.enqueue(mongo_insert, pesel, name, surname,
            gender, password,
                    email, phone, account_number,
                        created_at)

        # przeslanie logow do Elasticsearcha
        es_new_registration(created_at, email,
            account_number)
        return True
    else:
        return False
```

3.0.2 Cassandra

Poniższy kod przedstawia funkcję zwracającą dane z zastosowanym filtrze. Jeśli Wyniki mają mieścić się w podanym przedziale kwot, to wykonywana jest odpowiednia operacja, sprawdzająca czy wyniki pobrane z bazy spełniają zadany warunek. Jeśli nie został zastosowany żaden filtr, to są zwracane wyniki, bez żadnej obróbki.

Końcowe wyniki są zapisywane do zmiennej oraz sortowane według dat, a następnie zwracane w celu ich wyświetlenia.

Kod 2: Funkcja *get_rows* z *functions_cassandra.py*. *Źródło: opracowanie własne*

```
"""
Pobranie danych dla funkcji wyswietlajacej historie z
    filtrami
"""
def get_rows(amount1, amount2, records):
    amounts = []
    sources = []
    targets = []
    titles = []
    transaction_dates = []
    types = []

    for record in records:
        transaction_id = record.transaction_id
        # sprawdzenie czy wyniki maja miescic sie w
            przedziale kwot
        if amount1:
            row = session.execute("SELECT * FROM
                transaction_by_id where transaction_id="
                + str(transaction_id))

            # jesli tak, to szukane sa tylko te, ktore
                spelniaja warunek
```

```
            if int(amount1) <= int(row[0].amount) <=
               int(amount2):
                amounts.append(row[0].amount)
                sources.append(row[0].source)
                targets.append(row[0].target)
                titles.append(row[0].title)
                transaction_dates.append(row[0].
                   transaction_date)
                types.append('incoming')
        else:
            # w przeciwnym razie, dodawane sa wszystkie
               wyniki
            row = session.execute("SELECT * FROM
               transaction_by_id where transaction_id="
               + str(transaction_id))
            amounts.append(row[0].amount)
            sources.append(row[0].source)
            targets.append(row[0].target)
            titles.append(row[0].title)
            transaction_dates.append(row[0].
               transaction_date)
            types.append('incoming')

    # zapisanie wynikow do zmiennej
    transactions = {"source": sources, "targets":
       targets, "amounts": amounts,
                    "titles": titles, "
                       transaction_dates":
                       transaction_dates,
                    "types": types}
    df = pd.DataFrame(transactions)
    # posortowanie wynikow wedlug dat
    transaction_df = df.sort_values(by=['
       transaction_dates'], ascending=False)

    return transaction_df
```

3.0.3 Elasticsearch

Poniższy kod przedstawia funkcję służącą do przeglądania wybranych transakcji w Elasticsearch'u dla danego użytkownika.

Na początku pobierane są numery kont przypisanych danemu użytkownikowi, a następnie szukane są transakcje z nimi powiązane. Dzięki możliwościom Elasticsearch'a można szukać numeru konta w kilku polach, zo pozwala na szukanie przelewów wychodzących jak i przychodzących za jednym razem.

Jeśli znaleziono jakieś wyniki to zostają zapisane do zmiennej i przesłane w celu ich wyświetlenia, a jeśli nie, to zostaje zwrócony odpowiedni komunikat.

Kod 3: Funkcja *look_for_transaction* z *functions_elasticsearch.py*. *Źródło: opracowanie własne*

```
"""
Wyszukanie w logach transakcji danego uzytkownika
"""
def look_for_transactions(email):

    # pobranie numerow kont i ich sald dla wybranego
       uzytkownika
    accounts, balances = get_account_number_and_balance
       (email)

    data = []

    for account in accounts:
        search = {
                "query": {
                        "multi_match" : {
                          "query":    account,
                          "fields": [ "source", "target
                             " ]
                        }
                    }
            }

        # wyszukanie przelewow uzytkownika zarowno
           przychodzacych jak i
        # wychodzacych ("fields": [ "source", "target"
           ])
        res = es.search(index=INDEX_NAME, body=search,
           size=9999)
        if not res['hits']['hits']:
            # jesli nie ma wynikow, zostaja zwrocone
               odpowiednie wartosci
            return False, False
        else:
            # jesli sa wyniki, to zostaja zapisane do
               zmiennej
            for record in res['hits']['hits']:
                data.append(record['_source'])

    return data
```

3.0.4 Redis

Poniższy kod przedstawia funkcję zwracającą pozostały czas sesji dla danego użytkownika.

Po zalogowaniu, czas sesji użytkownika zostaje ustawiony na 5minut. Oznacza to, że jeśli przez 5minut nie będzie żadnej aktywności na jego koncie, to po tym czasie zostanie automatycznie wylogowany.

Kod 4: Kod programu. *Źródło: opracowanie własne*

```
"""
Pobranie TTL dla danego uzytkownika
"""
def get_user_ttl(email):
    time_left = redis.ttl(email)
    time_left = str(datetime.timedelta(seconds=
       time_left))

    if time_left[2] == '0':
        time_left = time_left[3:]
    else:
        time_left = time_left[2:]

    return time_left
```

4 Podsumowanie

Projekt został zrealizowany pomyślnie. Podczas realizacji zrodziło się wiele pomysłów, wystąpiły problemy, ale dzięki temu zdobyta wiedza jest bardzo duża. Poniżej zostaną opisane wady i zalety całego projektu, jak i użytych baz, napotkane problemy i sposób ich rozwiązania, a także pomysły, których nie udało się zaimplementować.

4.1 Wady i zalety

Projekt był dużym wyzwaniem, ponieważ wcześniej nie mieliśmy do czynienia z bazami NoSQL. Początkowo ciężko było przenieść wybrany obszar dziedzinowy na bazy NoSQL, ale wraz z zapoznawaniem się z każdą z baz i ich możliwościami stawało się to coraz łatwiejsze. Każda baza może być wykorzystywana w inny sposób, a połączenie ich tworzy spójną i wydajną całość.

Główną wadą projektu było to, że uczelnia nie zapewnia żadnej chmury czy zewnętrznego serwera, gdzie możnaby umieścić bazy. Sprawia to, że osoby pracujące nad projektem mają problem z wymianą danych między sobą, ponieważ konieczne było stawianie baz na maszynach lokalnych.

Kolejną wadą była konieczność stworzenia aplikacji, która zaprezentuje wyniki łączenia sie z bazami. Było to nieuniknione, lecz czas poświęcony na tworzenie aplikacji mógł zostać przeznaczony na lepsze zgłębianie wiedzy o bazach NoSQL. Czasem funkcjonalność którą można było zrealizować na bazie, ciężko było przenieść na obszar dziedzinowy, tak aby miało to sensowne zastosowanie i nie było wstawione "na siłę".

4.1.1 MongoDB

- Zalety:
  - brak z góry zdefiniowanej struktury, co było główną różnicą od dotychczas poznanych baz SQL
  - dokumenty zamiast tabel, pozwalają na przyjemniejsze wprowadzanie danych
  - brak konieczności definiowania relacji między dokumentami, można jednak je zasymulować lub zastąpić, jeśli istnieje taka potrzeba
  - bez większych problemów można podzielić bazę na różne serwery
  - łatwe wykorzystanie z wieloma językami programowania typu Python, C/C++
  - obszerna dokumentacja i dostępne materiały
  - prosta instalacja i konfiguracja
- Wady:

  Na potrzeby projektu nie stwierdzono żadnych wad. Efektywna, prosta w obsłudze i nie sprawiająca większych problemów.

4.1.2 Cassandra

- Zalety:
  - wykorzystanie node'ów zapobiega utracie danych, jeśli jeden node jest nieaktywny, to dostęp do danych można uzyskać z aktywnego
  - jako że jest to kolumnowa baza danych, można było uzyskać szybki dostęp do danych, po dobrym stworzeniu tabel
  - polecenia podobne do SQL
  - obszerna dokumentacja na temat CQL
- Wady:
  - trudna instalacja i konfiguracja w przypadku większej ilości node'ów
  - już dla trzech node'ów potrzebna jest duża moc obliczeniowa
  - podczas tworzeniu tabel należy pamiętać na dobrym stworzeniu klucza
  - złe dobranie klucza tabeli może spowodować brak pełnej wydajności

4.1.3 Redis

- Zalety:
  - bardzo łatwe zastosowanie do stworzenia sesji użytkownika itp.
  - mechanizm kolejki, pozwalający na utrzymanie synchronizacji
  - bardzo wydajna baza
  - łatwe wykorzystanie z wieloma językami programowania
- Wady:

  Na potrzeby projektu nie stwierdzono żadnych wad.

4.1.4 Elasticsearch

- Zalety:
  - bardzo dobrze sprawdza się jako wyszukiwarka
  - łatwe wykorzystanie mechanizmu podpowiedzi, w większości baz źle wpisana wartość wiąże się z brakiem wyniku, w tej bazie można otrzymać wynik nawet jeśli popełni sie literówkę
  - bardzo łatwa i dobra wizualizacja danych w Kibanie
  - bardzo wydajna, nawet w przypadku bardzo dużej ilości danych
- Wady:
  - skromna dokumentacja na temat wykorzystania za pomocą różnych języków programowania, np. Python'a
  - lepiej sprawdza się tworząc requesty http

Projekt pozwolił poznać wiele możliwości wspomnianych baz, nawet jeśli nie zostały one wprost wykorzystane w projekcie. Czasem było tak, że można było pokazać lepiej zastosowanie jakiejś bazy, ale niekoniecznie pasowało to do projektu. Na przykład mechanizm podpowiedzi Elasticsearch'a. Gdyby np. tematem projektu była wyszukiwarka piosenek, można by lepiej pokazać tą funkcję, ale z kolei zastosowanie innych baz mogłoby stać się kłopotliwe.

4.2 Napotkane problemy i ich rozwiązanie

Projekt nie sprawił większych problemów. Przede wszystkim należało zapoznać się z dokumentacją i materiałami dostępnymi w internecie.

Jednym z głównych problemów był fakt, że bazy były postawione na maszynie lokalnej, przez co zewnętrzni użytkownicy nie mieli dostępu do wszystkich danych a czasem, żeby wykonać jakąś operację, trzeba było połączyć się z jedną bazą i przesłać dane do drugiej.

Problematyczne okazało się też zastosowanie większej ilości możliwości baz, ponieważ ciężko było odnieść je do obszaru dziedzinowego stworzonej aplikacji. Trzeba było się też skupić na aplikacji, a nie samych bazach, może gdyby było więcej czasu, udałoby się zastosować więcej funkcji z każdej z bazy.

4.3 Pomysły i idee

Kilku funkcji nie udało się zaimplementować, z powodu braku czasu, a mogłyby one przedstawić więcej funkcjonalności baz. Funkcje, których nie udało się zrobić:

- możliwość usunięcia konta bankowego lub użytkownika i przesłanie odpowiednich logów do Elasticsearch'a (Elasticsearch + MongoDB)
- możliwość edycji danych osobowych użytkownika w jego panelu (MongoDB)
- mechanizm płacenia kartą, z możliwością odczytu gdzie zapłacono kartą (Cassandra, MongoDB)
- podczas wykonywania przelewu, wykorzystanie Elasticsearch'a w przypadku pomylenia numeru (trochę niepraktyczne ze względów bezpieczeństwa) (Elasticsearch)
- możliwość przeglądu kont użytkowników z nakładaniem różnych filtrów z panelu administratora (MongoDB + Cassandra)

Bibliografia

[1] https://youtu.be/x-Nh0xxa6UE

[2] https://github.com/Darianek/BDnoSQL