

WYDZIAŁ
MATEMATYKI
I FIZYKI STOSOWANEJ
POLITECHNIKI RZESZOWSKIEJ

Analizy zachowań użytkowników na stronie internetowej przy użyciu Apache Kafka

Usługi sieciowe w biznesie - projekt

Krystian Pupiec

Rzeszów, 2024

Spis treści

1. Wstęp teoretyczny	2
1.1. Opis problemu	2
1.2. Apache Kafka	2
2. Implementacja rozwiązania	6
2.1. Opis struktury i ogólne działanie	6
2.2. Apache Kafka	7
2.3. Baza danych w języku MySQL	9
2.4. Strona internetowa	11
2.5. Aplikacja Flask	14
2.6. Serwer w aplikacji Node.js	15
2.7. Aplikacja tworząca raport	17
3. Podsumowanie	21
Literatura	22

1. Wstęp teoretyczny

1.1. Opis problemu

Analiza zachowań użytkowników na stronie internetowej jest jednym z kluczowych narzędzi dla firm dążących do optymalizacji doświadczenia użytkownika i zwiększenia skuteczności swoich działań marketingowych. Podczas gdy strony internetowe generują coraz większe ilości danych, kluczowym wyzwaniem staje się efektywne gromadzenie, przetwarzanie i analiza tych danych. Rozumienie, jak użytkownicy poruszają się po stronie, jakie treści przyciągają ich uwagę, a także gdzie napotykają trudności, pozwala na wprowadzanie precyzyjnych usprawnień. Skuteczna analiza tych zachowań umożliwia lepsze dostosowanie treści oraz personalizację ofert, co w rezultacie prowadzi do wyższych wskaźników konwersji i satysfakcji klientów.

1.2. Apache Kafka

Apache Kafka to platforma do przetwarzania strumieni danych (event streaming platform), która na pierwszy rzut oka może wydawać się zwykłym brokerem wiadomości, takim jak RabbitMQ czy Amazon SQS. Rzeczywiście, Kafka również umożliwia asynchroniczną wymianę wiadomości między producentami a konsumentami. Jednakże, szczegóły robią różnicę. Kafka kładzie duży nacisk na wydajność i skalowalność, oferując wiele opcji konfiguracyjnych, które pozwalają dostosować działanie platformy do specyfiki danych, z którymi pracujemy, ich wielkości oraz częstotliwości odczytu. Ponadto, Kafka umożliwia konfigurację czasu przechowywania wysyłanych wiadomości. W przeciwieństwie do typowych brokerów wiadomości, gdzie dane są usuwane zaraz po odebraniu przez adresata, w Kafka wiadomości mogą być przechowywane przez określony czas. Adresat nie musi być znany w momencie wysyłania wiadomości, co odróżnia Kafkę od tradycyjnych modeli publish/subscribe. W rzeczywistości, działanie Kafki bardziej przypomina bazę danych NoSQL.

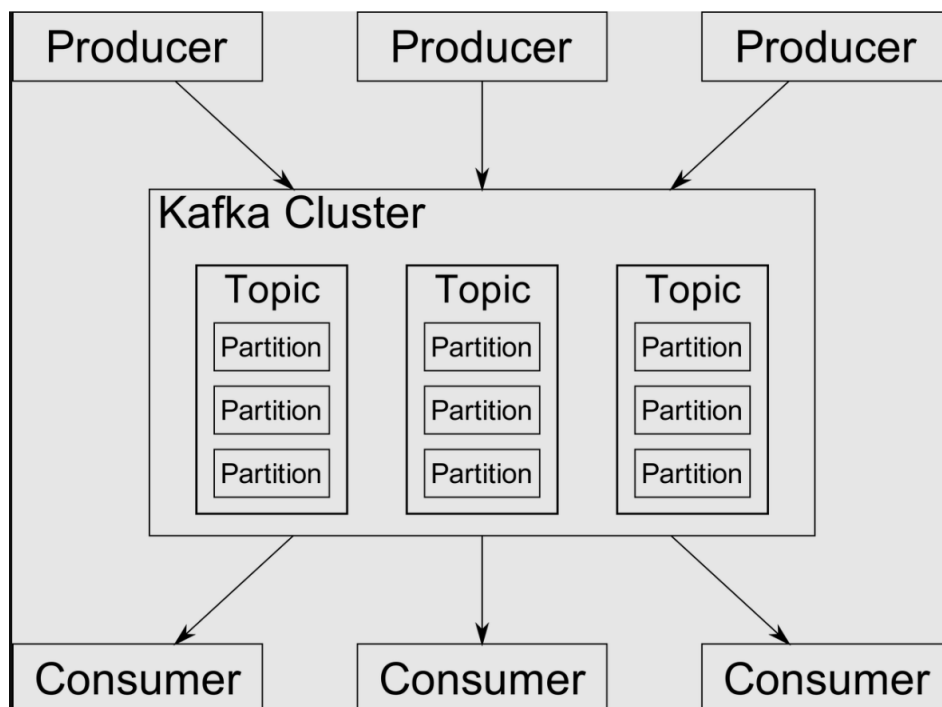
Potrzebujemy uruchomić co najmniej dwa serwery (w znaczeniu procesy), aby mieć działającą instancję Kafki. Zookeeper pełni kluczową rolę w serwerze Apache Kafka, przechowując aktualną konfigurację, listę brokerów i topiców oraz ich rozmieszczenie między brokerami, a także informacje o uprawnieniach (ACLs). Odpowiada za monitorowanie do-

stępności brokerów i w przypadku wykrycia problemu, informuje brokerów o zaistniałej sytuacji. Na przykład, gdy broker będący liderem dla danej partycji przestaje działać, jedna z replik (slave) przejmuje jego rolę. Zookeeper musi być uruchomiony przed startem Kafki. Konsumenci i producenci nie wchodzi z nim bezpośrednio w interakcje, jednak Zookeeper musi działać w klastrze, aby Kafka funkcjonowała prawidłowo. Zookeeper jest niezbędny dla prawidłowego działania klastra Kafki, dlatego w środowiskach produkcyjnych zaleca się posiadanie zapasowych instancji. Ze względu na mechanizm replikacji w Zookeeper, rekomenduje się nieparzystą liczbę węzłów, przy minimalnej sensownej liczbie trzech. W przypadku większej liczby brokerów, liczba węzłów Zookeeper może być zwiększona do 5 lub 7. Brokerzy to serwery odpowiedzialne za wysyłanie i odbieranie wiadomości. Przy konfiguracji klienta Kafka należy podać nazwy hostów i porty brokerów. Nie trzeba podawać pełnej listy, ponieważ klient Kafka pobierze ją od wskazanego brokera. W kodzie klienckim odnosimy się do topiku, a nie do konkretnego brokera. Aby zapewnić działającą replikację danych, potrzebujemy kilku brokerów. Rozsądnie jest zacząć od trzech, a w razie potrzeby ich liczba może być zwiększona nawet do kilkuset.

Wiadomość w Kafka, równoznaczna z rekordem w bazie danych, składa się z klucza (opcjonalnego, jeśli kompaktowanie danych nie jest używane), nagłówków (opcjonalnych i mogących być wykorzystanych dowolnie), metadanych oraz wartości zawierających nasze dane. Metadane obejmują informacje o docelowym topiku. Gdy klucz nie jest zdefiniowany, partycja, do której trafia wiadomość, zostaje wybrana losowo. Jednak przy użyciu klucza, partycja zostaje wyznaczona przy użyciu domyślnego algorytmu round-robin lub innego, który sami zaimplementujemy. Dodatkowo, wiadomość zawiera znacznik czasowy (timestamp) określający czas wysłania. W przypadku braku wartości, zostanie on ustawiony przez brokera. W języku Java, wiadomość jest reprezentowana przez klasę `ProducerRecord`. Rozmiar wiadomości może wynosić do około 1 MB, a domyślne ustawienia związane z wysyłaniem są optymalizowane dla wiadomości o wielkości do 1 KB. Teoretycznie, w jednym topiku można przechowywać wiadomości o różnym formacie. W praktyce, korzystniej jest uzgodnić jeden schemat. Kafka nie przeprowadza walidacji danych, więc dane mogą przyjąć dowolną formę. Walidacja może być przeprowadzana podczas odbierania lub wysyłania danych w kodzie producenta lub konsumenta.

Topic jest kluczowym elementem w Kafce, którego zrozumienie jest niezbędne do zrozu-

mienia wszystkich innych koncepcji. Można go porównać do kolejki komunikatów z tradycyjnych brokerów wiadomości, gdzie nadawcy wysyłają dane, a konsumenci je odbierają. Jednak w Kafka topic działa nie jak typowa kolejka, ale raczej jak tablica dwuwymiarowa, składająca się z partycji i offsetów, co wyznacza jego specyficzną strukturę.



Rysunek 1.1: Działanie struktury Kafka Apache

Możemy sami ustalić liczbę partycji, która jest elastyczna. Dla większości przypadków wystarczy rozpocząć z około 10 partycjami. Dodanie nowych partycji nie niesie za sobą dodatkowych kosztów. Chociaż nie ma ściśle określonego limitu partycji, Kafka zacznie zwalniać w pewnym momencie. Rzeczywiste maksimum to zwykle kilka tysięcy partycji na jednym serwerze i kilkaset tysięcy na całym klastrze.

Producenci kierują wiadomość do konkretnej partycji w danym topiku. Offset jest przypisywany przez brokera i zawsze zwiększa się o jeden. Nie ma możliwości nadpisywania ani usuwania wiadomości z pozycji konsumenta. Jednak w kodzie producenta nie musimy sami wybierać partycji, na którą ma trafić wiadomość. Kod producenta automatycznie dobierze partycję na podstawie klucza wiadomości, a jeśli klucz nie istnieje, wiadomość zostanie wysłana do losowej partycji.

W Kafce dane w topikach nie są automatycznie usuwane po odczytaniu. Standardowym zachowaniem jest automatyczne usuwanie wpisów starszych niż tydzień, jednak tę wartość możemy dostosować w konfiguracji brokera lub konkretnego topiku. Ta elastyczność pozwala konsumentom na samodzielne wybieranie punktu, od którego chcą rozpocząć odczyt. Dzięki temu możliwe jest czytanie tych samych partycji równocześnie z różnych miejsc.

Jeśli nie ma możliwości skorzystania z bazy danych lub innego rozwiązania do komunikacji asynchronicznej, Apache Kafka może nie być odpowiednim wyborem dla mniejszych projektów. Na początku wymaga kilku serwerów, aby w pełni wykorzystać korzyści płynące z jej użycia, jednak w większych organizacjach jest to często łatwiejsze niż zarządzanie jedną ogromną bazą danych lub wieloma różnymi.

W wielu dużych korporacjach zaczęło brakować wolnych kluczy w tabelach baz danych, co skłoniło architektów do poszukiwania sposobów zarządzania danymi bez konieczności przeprowadzania skomplikowanych migracji i przepisów. W Apache Kafka dane są wyspecjalizowane, gdzie każdy zapisywany jest do niezależnego topicu. Dzięki temu dane z jednego lub wielu tematów mogą być przetwarzane i przesyłane dalej, co pozwala oszczędzić miejsce i uniknąć problemów z zakleszczeniem czy liczbą połączeń do bazy.

Jeśli priorytetem jest maksymalna przepustowość, istnieją alternatywy takie jak Apache Spark, Storm, Flink, Redis, RabbitMQ. Jednak żadna z tych alternatyw nie oferuje takich samych właściwości jak Kafka i są przeznaczone do nieco innych celów.

Jedną z głównych wad Kafki jest brak indeksowania. Nawet jeśli projekt nie zakłada analizy danych, wyszukiwania czy łączenia, istnieje potrzeba dostępu do danych w celach testowania i debugowania oprogramowania. Dodatkowo brakuje opcji filtrowania danych pobieranych z Kafki, co może znacznie utrudnić pracę.

W porównaniu do baz relacyjnych, Kafka ma ograniczenia, zwłaszcza jeśli chodzi o dostęp do konkretnych pól wiadomości i sposób pobierania danych. W przypadku prób weryfikacji nowej funkcjonalności w środowisku SIT/UAT, gdzie topiki mogą mieć dużą objętość, może to prowadzić do długiego oczekiwania na wyniki zapytań.

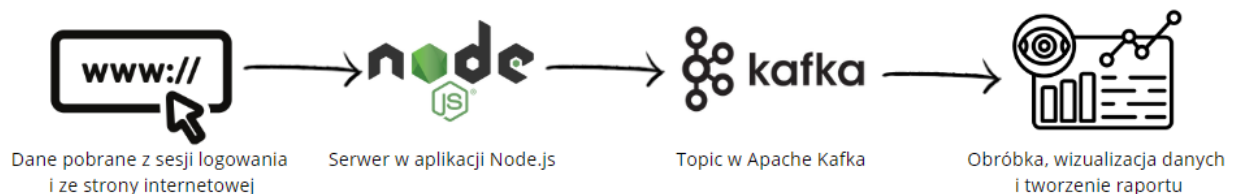
Apache Kafka to więc wysokowydajna platforma do przetwarzania strumieni danych, umożliwiająca niezawodne przesyłanie, przetwarzanie i przechowywanie dużych strumieni danych w czasie rzeczywistym. Dzięki swojej skalowalności, odporności i elastyczności,

Kafka jest szeroko stosowana w dużych organizacjach do budowy systemów analizy danych, przetwarzania zdarzeń oraz komunikacji asynchronicznej.

2. Implementacja rozwiązania

2.1. Opis struktury i ogólne działanie

Sposobem w jaki będzie wykorzystana analiza zachowań użytkowników na stronie internetowej jest monitorowanie poszczególnych kliknięć użytkowników. Po kliknięciu w poszczególne łącze na stronie internetowej, informacja o tym zdarzeniu wraz z danymi aktualnie zalogowanego użytkownika trafia do serwera w aplikacji Node.js, z którego później, zebrane dane trafiają do topicu w Apache Kafka. Następnie co określony czas, automatycznie uruchamiany jest program w języku Python, który odpowiednio przekształca dane i tworzy z nich wizualizacje. Pozwala to na odczytanie, które łącze jest najczęściej odwiedzane oraz, dodatkowo, dla każdego łącza tworzone są wykresy z informacją o użytkownikach w nie klikających, a konkretniej, informacje o wieku, płci oraz mieście zamieszkania. Może to być wykorzystane do analizy, jaka grupa wiekowa lub płeć interesuje się danym tematem lub z jakiej części świata ludzie najczęściej odwiedzają zakładki o danym temacie.



Rysunek 2.2: Struktura implementacji rozwiązania

W dalszych punktach zostaną przedstawione dokładniejsze opisy poszczególnych plików potrzebnych do zastosowania tego rozwiązania. W projekcie zostały utworzone takie pliki jak:

- login_or_register.html - strona internetowa, której zadaniem jest rejestracja użytkownika lub przeprowadzenie procesu logowania,
- serwer.js - serwer napisany w aplikacji Node.js, który przekazuje dane ze strony internetowej do topicu w Apache Kafka,

- index.html - jest to strona internetowa zawierająca główną treść (kolejne podstrony tej strony internetowej są umieszczone w osobnych plikach html o nazwach wskazujących na nazwę podstrony),
- app.py - ten plik zawiera aplikację Flask, która obsługuje logowanie, rejestrację użytkowników, wyświetlanie danych sesji, harmonogramowanie zadań oraz komunikację z bazą danych MySQL,
- Raport.py - program odczytujący dane z topiców w Apache Kafka oraz tworzący raport wraz z wizualizacjami.

Dodatkowo, koniecznością było skonfigurowanie Apache Kafka, uruchomienie jego serwerów. Utworzona została również baza danych w języku MySQL, która była potrzebna do poprawnego funkcjonowania procesu rejestracji i logowania.

2.2. Apache Kafka

Aby cała struktura działała najpierw został uruchomiony serwer ZooKeeper. ZooKeeper jest centralną komponentą w ekosystemie Apache Kafka i pełni rolę koordynatora klastra Kafka, zarządzając m.in. konfiguracją, rozproszonymi zadaniami oraz synchronizacją między brokerami Kafka.

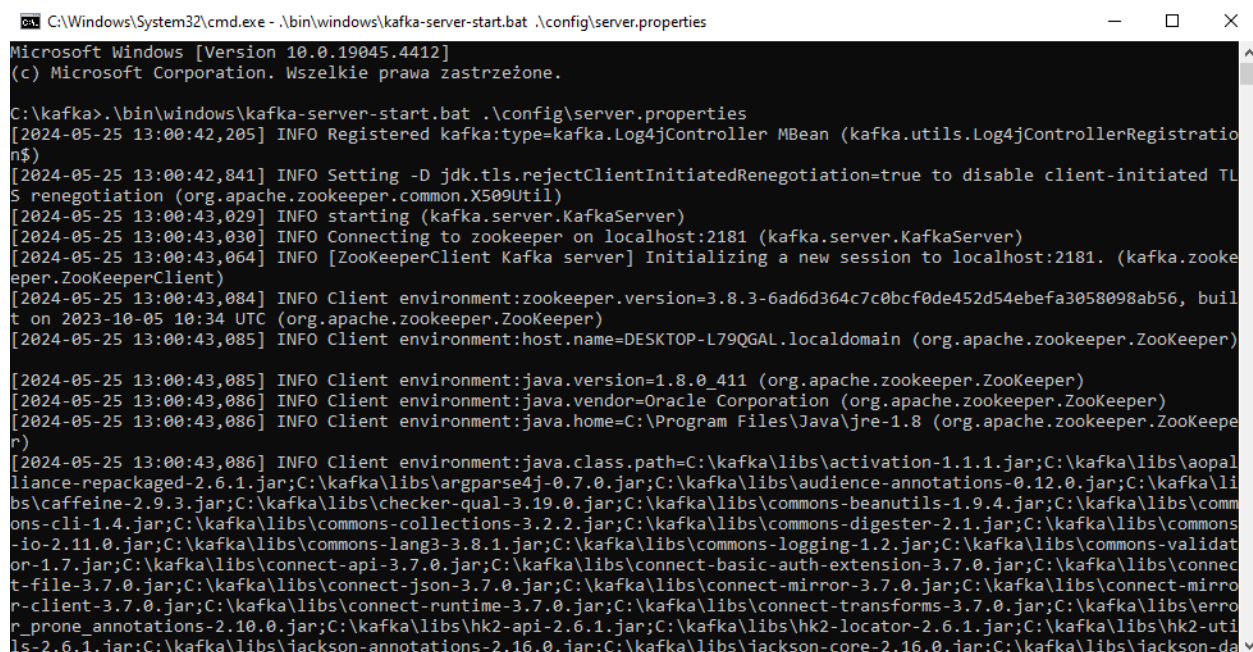
```

C:\Windows\System32\cmd.exe - .\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
Microsoft Windows [Version 10.0.19045.4412]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\kafka>.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
[2024-05-25 12:59:59,740] INFO Reading configuration from: .\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,755] WARN \tmp\zookeeper is relative. Prepend .\ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,755] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,755] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,771] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,771] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,771] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DataDirCleanupManager)
[2024-05-25 12:59:59,771] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DataDirCleanupManager)
[2024-05-25 12:59:59,771] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DataDirCleanupManager)
[2024-05-25 12:59:59,771] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)
[2024-05-25 12:59:59,771] INFO Log4j 1.2 jmx support not found; jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2024-05-25 12:59:59,771] INFO Reading configuration from: .\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,771] WARN \tmp\zookeeper is relative. Prepend .\ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,771] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,771] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,771] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,771] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-25 12:59:59,771] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2024-05-25 12:59:59,803] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider
  
```

Rysunek 2.3: Uruchomienie Zookeeper'a

Następnie został uruchomiony serwer Kafka. Jest to główny komponent w ekosystemie Apache Kafka, który odpowiada za przetwarzanie i przechowywanie strumieni danych. Po uruchomieniu serwera Kafka klastr staje się gotowy do przyjmowania wiadomości, przetwarzania ich i udostępniania konsumentom.



```
C:\Windows\System32\cmd.exe - .\bin\windows\kafka-server-start.bat .\config\server.properties
Microsoft Windows [Version 10.0.19045.4412]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\kafka>.\bin\windows\kafka-server-start.bat .\config\server.properties
[2024-05-25 13:00:42,205] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2024-05-25 13:00:42,841] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2024-05-25 13:00:43,029] INFO starting (kafka.server.KafkaServer)
[2024-05-25 13:00:43,030] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2024-05-25 13:00:43,064] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2024-05-25 13:00:43,084] INFO Client environment:zookeeper.version=3.8.3-6ad6d364c7c0bcf0de452d54ebefa3058098ab56, built on 2023-10-05 10:34 UTC (org.apache.zookeeper.ZooKeeper)
[2024-05-25 13:00:43,085] INFO Client environment:host.name=DESKTOP-L79QGAL.localdomain (org.apache.zookeeper.ZooKeeper)
[2024-05-25 13:00:43,085] INFO Client environment:java.version=1.8.0_411 (org.apache.zookeeper.ZooKeeper)
[2024-05-25 13:00:43,086] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2024-05-25 13:00:43,086] INFO Client environment:java.home=C:\Program Files\Java\jre-1.8 (org.apache.zookeeper.ZooKeeper)
[2024-05-25 13:00:43,086] INFO Client environment:java.class.path=C:\kafka\libs\activation-1.1.1.jar;C:\kafka\libs\aaliance-repackaged-2.6.1.jar;C:\kafka\libs\argparse4j-0.7.0.jar;C:\kafka\libs\audience-annotations-0.12.0.jar;C:\kafka\libs\caffeine-2.9.3.jar;C:\kafka\libs\checker-qual-3.19.0.jar;C:\kafka\libs\commons-beanutils-1.9.4.jar;C:\kafka\libs\commons-cli-1.4.jar;C:\kafka\libs\commons-collections-3.2.2.jar;C:\kafka\libs\commons-digester-2.1.jar;C:\kafka\libs\commons-io-2.11.0.jar;C:\kafka\libs\commons-lang3-3.8.1.jar;C:\kafka\libs\commons-logging-1.2.jar;C:\kafka\libs\commons-validator-1.7.jar;C:\kafka\libs\connect-api-3.7.0.jar;C:\kafka\libs\connect-basic-auth-extension-3.7.0.jar;C:\kafka\libs\connect-file-3.7.0.jar;C:\kafka\libs\connect-json-3.7.0.jar;C:\kafka\libs\connect-mirror-3.7.0.jar;C:\kafka\libs\connect-mirror-client-3.7.0.jar;C:\kafka\libs\connect-runtime-3.7.0.jar;C:\kafka\libs\connect-transforms-3.7.0.jar;C:\kafka\libs\error-prone-annotations-2.10.0.jar;C:\kafka\libs\hk2-api-2.6.1.jar;C:\kafka\libs\hk2-locator-2.6.1.jar;C:\kafka\libs\hk2-utils-2.6.1.jar;C:\kafka\libs\jackson-annotations-2.16.0.jar;C:\kafka\libs\jackson-core-2.16.0.jar;C:\kafka\libs\jackson-databind-2.16.0.jar
```

Rysunek 2.4: Uruchomienie serwera Kafka

Aby można było monitorować zachowania użytkowników na stronie internetowej bez ograniczonego przedziału czasowego, w pliku konfiguracyjnym brokera **server.properties**, został zmieniony parametr dotyczący określa minimalnego wieku pliku dziennika w godzinach, aby był uznany za kandydata do usunięcia ze względu na wiek. W tym przypadku ustawiony był on na 168 godzin, co oznacza 7 dni. Po upływie tego czasu starsze pliki dziennika mogły zostać usunięte. Parametr `log.retention.hours` został zmieniony na wartość `'-1'`, aby dane nigdy nie były usuwane automatycznie ze względu na wiek.

```
##### Log Retention Policy #####

# The following configurations control the disposal of log segments. The policy can
# be set to delete segments after a period of time, or after a given size has accumulated.
# A segment will be deleted whenever *either* of these criteria are met. Deletion always happens
# from the end of the log.

# The minimum age of a log file to be eligible for deletion due to age
log.retention.hours=-1

# A size-based retention policy for logs. Segments are pruned from the log unless the remaining
# segments drop below log.retention.bytes. Functions independently of log.retention.hours.
#log.retention.bytes=1073741824

# The maximum size of a log segment file. When this size is reached a new log segment will be created.
#log.segment.bytes=1073741824

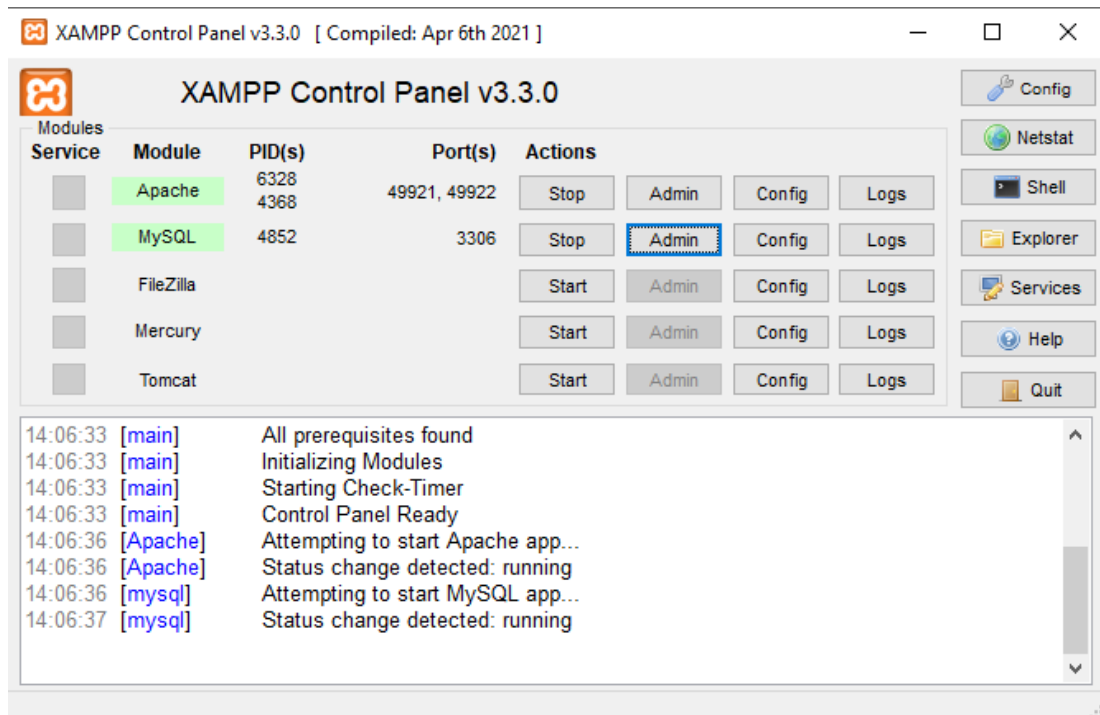
# The interval at which log segments are checked to see if they can be deleted according
# to the retention policies
log.retention.check.interval.ms=300000
```

Rysunek 2.5: Zmiana parametru `log.retention.hours` na wartość **-1**

Tworzenie topiców nie będzie wykonywane bezpośrednio w nowym terminalu, proces ten jest częścią kodu w pliku *serwer.js*, który będzie opisany w późniejszym etapie.

2.3. Baza danych w języku MySQL

Aby funkcjonalność rejestracji oraz logowania użytkowników była możliwa, konieczna była baza danych. Na potrzeby tego projektu, za pomocą pakietu XAMPP, została stworzona baza danych w języku MySQL. Na kolejnych zrzutach ekranu zostaną przedstawione panel pakietu XAMPP, struktura tabeli oraz przykładowe dane z dostępnej bazy danych.



Rysunek 2.6: Panel pakietu XAMPP

Serwer: 127.0.0.1 » Database: all_users » Tabela: users

Table structure Widok relacyjny

#	Nazwa	Typ	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	id	int(11)			Nie	Brak		AUTO_INCREMENT	Change Drop Więcej
<input type="checkbox"/> 2	name	varchar(50)	utf8mb4_general_ci		Nie	Brak			Change Drop Więcej
<input type="checkbox"/> 3	email	varchar(100)	utf8mb4_general_ci		Nie	Brak			Change Drop Więcej
<input type="checkbox"/> 4	age	int(11)			Nie	Brak			Change Drop Więcej
<input type="checkbox"/> 5	gender	enum('Mężczyzna', 'Kobieta')	utf8mb4_general_ci		Nie	Brak			Change Drop Więcej
<input type="checkbox"/> 6	city	varchar(50)	utf8mb4_general_ci		Nie	Brak			Change Drop Więcej
<input type="checkbox"/> 7	password	varchar(255)	utf8mb4_general_ci		Nie	Brak			Change Drop Więcej

Rysunek 2.7: Struktura tabeli users z bazy danych all_users

✓ Pokazano wiersze 0 - 5 (6 total, Wykonanie zapytania trwało 0,0004 sekund(y).)

```
SELECT * FROM `users`
```

☐ Profilowanie [[Edytuj w linii](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

☐ Show all | Liczba wierszy: 25 | Filter rows: Sortuj wg klucza:

Extra options

					id	name	email	age	gender	city	password
<input type="checkbox"/>					1	Krystian	krystian@wp.pl	22	Mężczyzna	Rzeszów	password1
<input type="checkbox"/>					4	Jan	jan@onet.pl	25	Mężczyzna	Warszawa	password1
<input type="checkbox"/>					5	Daniel	d@df.pl	43	Mężczyzna	Lublin	password1
<input type="checkbox"/>					6	Ewa	ewa@wp.pl	27	Kobieta	Kraków	password1
<input type="checkbox"/>					7	Damian	damian@onet.pl	32	Mężczyzna	Gdańsk	password1
<input type="checkbox"/>					8	Marta	marta@onet.pl	19	Mężczyzna	Poznań	password1

☐ Check all Z zaznaczonymi: Edit Copy Delete Export

Rysunek 2.8: Przykładowe dane z tabeli users

Połączenie z bazą danych następuje w pliku **app.py**.

2.4. Strona internetowa

Na potrzeby projektu została stworzona prosta strona internetowa, aby tylko zaprezentować możliwości przesyłania danych do Apache Kafka.

Strona *login_or_register.html* pełni rolę interfejsu umożliwiającego użytkownikom logowanie się lub rejestrowanie na stronie internetowej. Strona została zbudowana przy użyciu HTML5, co zapewnia solidną strukturę dokumentu oraz semantyczne znaczenie poszczególnych elementów. Wbudowany CSS definiuje styl i rozmieszczenie elementów takich jak przyciski, formularze i nagłówki. Do dynamicznego wyświetlania formularzy logowania i rejestracji użyto prostego skryptu JavaScript. Strona wykorzystuje również szablon Jinja2, który jest renderowany przez framework Flask w Pythonie. Dzięki temu możliwe jest wyświetlanie komunikatów zwrotnych dla użytkownika, takich jak błędy lub potwierdzenia.

Rysunek 2.9: Formularz rejestracji użytkownika

Rysunek 2.10: Formularz logowania użytkownika

Strona *index.html* jest główną stroną aplikacji internetowej. Zawiera ona menu nawigacyjne, które umożliwia użytkownikowi poruszanie się pomiędzy różnymi sekcjami strony, takimi jak GitLab, JupyterHub, Git, OpenLDAP, Google Kubernetes, Odoo, RabbitMQ, ActiveMQ, Camel, Cassandra, Kafka i ZooKeeper. Podstrony te są umieszczone w osobnych plikach html o nazwach wskazujących na nazwę łącza. Struktura dokumentu jest zbudowana przy użyciu HTML5, co zapewnia czytelność i semantyczność kodu. Do

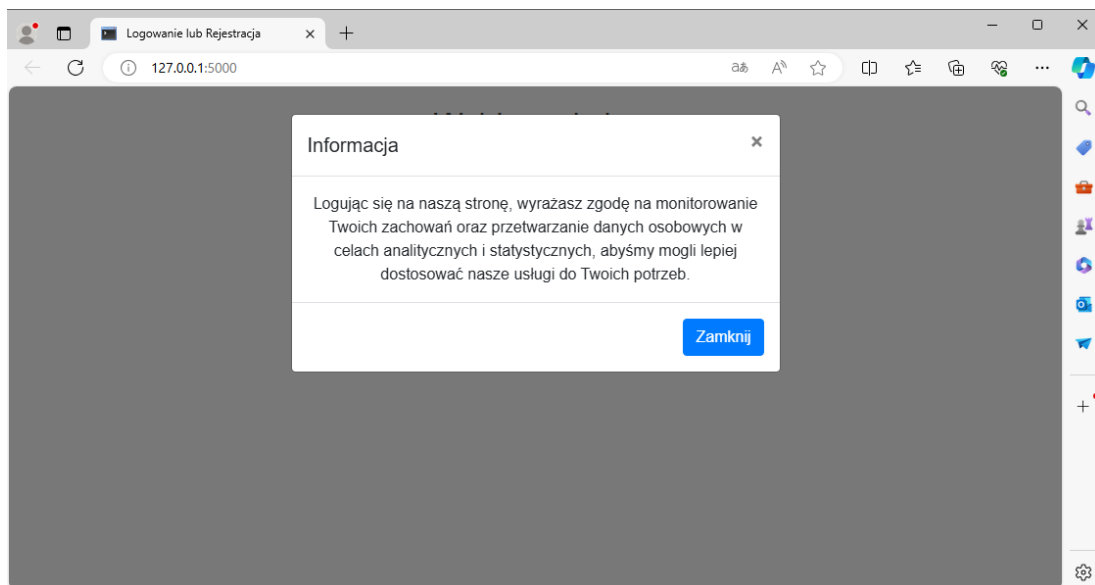
stylizacji strony użyto Bootstrap 4.5.2, co zapewnia responsywność i estetyczny wygląd. Skrypt jQuery obsługuje dynamiczne wyświetlanie sekcji strony na podstawie kliknięć użytkownika. Dodatkowo, skrypt w języku javascript, wysyła dane do serwera za pomocą funkcji fetch, aby przekazać informacje o wieku, mieście, płci i wybranym temacie do serwera, stworzonego w aplikacji Node.js, pośredniczącego z Apache Kafka, co umożliwia śledzenie interakcji użytkowników. Strona *index.html* korzysta z szablonów Jinja2 do dynamicznego generowania treści na podstawie zmiennych sesji. Flask, będący frameworkiem webowym używanym do obsługi serwera backend, renderuje szablony Jinja2 i zarządza logiką aplikacji.



Rysunek 2.11: Strona główna

Wszystkie inne podstrony mają taki sam szablon, różnią się jedynie opisem danej usługi.

Podsumowując, *login_or_register.html* i *index.html* są stronami internetowymi zbudowanymi przy użyciu HTML, CSS, JavaScript, Bootstrap oraz Flask z szablonami Jinja2. Pierwsza z nich umożliwia logowanie i rejestrację użytkowników, podczas gdy druga jest główną stroną aplikacji z menu nawigacyjnym i dynamicznymi sekcjami treści, integrującymi się z serwerem Node.js, który wysyła dane do Apache Kafka, do monitorowania interakcji użytkowników. Dodatkowo, po otwarciu strony przez użytkownika pojawia się informacja o monitorowaniu jego zachowania.



Rysunek 2.12: Informacja o monitorowaniu zachowania użytkownika na stronie internetowej

2.5. Aplikacja Flask

Ten plik zawiera kompletną aplikację internetową zbudowaną w frameworku Flask, która obsługuje procesy logowania i rejestracji użytkowników, wyświetlanie danych związanych z sesją użytkownika, a także realizuje harmonogramowanie zadań oraz komunikację z bazą danych MySQL.

Na początku pliku zainicjalizowano aplikację Flask i ustawiono klucz sesji, który jest niezbędny do zarządzania sesjami użytkowników w aplikacji. Następnie skonfigurowano połączenie z bazą danych MySQL, definiując parametry takie jak host, użytkownik, hasło oraz nazwę bazy danych.

Aplikacja zawiera również konfigurację logowania, która pozwala na rejestrowanie zdarzeń w aplikacji dla celów debugowania i monitorowania.

Istotnym elementem pliku jest mechanizm harmonogramowania zadań. Importowana jest funkcja odpowiedzialna za generowanie raportów, która uruchamiana jest co określony czas za pomocą biblioteki `schedule`. Aby zapewnić, że harmonogram działa równolegle z główną aplikacją, uruchomiono go w oddzielnym wątku.

Główna część aplikacji to definicje tras, które obsługują różne funkcjonalności:

- Trasa domyślna renderuje stronę logowania lub rejestracji.
- Trasa odpowiedzialna za stronę główną sprawdza, czy użytkownik jest zalogowany i, jeśli tak, wyświetla stronę główną z informacjami pobranymi z sesji użytkownika.
- Trasa do logowania użytkownika obsługuje przesyłane dane logowania, sprawdza je w bazie danych i, jeśli są poprawne, zapisuje dane użytkownika w sesji.
- Trasa wylogowania usuwa dane użytkownika z sesji i przekierowuje na stronę logowania.
- Trasa rejestracji użytkownika obsługuje przesyłane dane rejestracyjne, zapisuje je w bazie danych i przekierowuje na stronę logowania po pomyślnej rejestracji.
- Trasa do pobierania danych sesji zwraca informacje o użytkowniku w formacie JSON.

Na końcu pliku znajduje się kod uruchamiający aplikację Flask w trybie debugowania, co umożliwia testowanie i monitorowanie aplikacji podczas jej rozwoju.

```
* Serving Flask app 'app'
* Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 912-117-031
```

Rysunek 2.13: Uruchomienie aplikacji w Flask

2.6. Serwer w aplikacji Node.js

Ten kod definiuje serwer napisany w języku Node.js, który używa frameworka Express.js do obsługi zapytań HTTP. Na początku importowane są wymagane moduły, takie jak express, kafkajs i axios, które są niezbędne do działania aplikacji. Następnie tworzony jest obiekt aplikacji Express, który będzie obsługiwał żądania HTTP.

Kolejnym krokiem jest inicjalizacja klienta Kafka za pomocą biblioteki kafkajs. Klient ten jest używany do komunikacji z brokerami Apache Kafka. Podczas inicjalizacji klienta ustawiany jest jego identyfikator oraz adresy brokerów Kafka.

Po zainicjalizowaniu klienta Kafka, tworzone są obiekty admina i producenta Kafka. Admin Kafka służy do wykonywania operacji administracyjnych, takich jak tworzenie tematów, podczas gdy producent Kafka umożliwia wysyłanie wiadomości do określonych tematów.

Następnie definiowana jest funkcja *createTopicIfNotExists*, która służy do sprawdzania istnienia tematu w Kafka i ewentualnego jego utworzenia, jeśli nie istnieje. Ta funkcja jest wywoływana w obsłudze zapytania GET, gdy przychodzi nowe zapytanie.

Po zdefiniowaniu funkcji i inicjalizacji klienta Kafka, konfigurowane są nagłówki CORS. Nagłówki te są ustawiane w celu umożliwienia komunikacji między różnymi domenami. Nagłówki CORS (Cross-Origin Resource Sharing) są istotne w kontekście bezpieczeństwa przeglądarek internetowych i polityki bezpieczeństwa. W przypadku tego kodu, ustawienie nagłówków CORS za pomocą funkcji *app.use()* zapewnia, że aplikacja Express będzie akceptować żądania HTTP od dowolnej domeny ("*"), co oznacza, że żądania mogą być wykonywane zarówno z tej samej domeny, jak i z innych domen.

Następnie serwer łączy się z Kafka oraz uruchamiany jest serwer Express, który nasłuchuje na porcie 3000. Główna obsługa zapytania GET znajduje się na ścieżce */sendToKafka*. Po otrzymaniu zapytania GET, sprawdzane jest, czy wartość *id* jest równa *#home*. Jeśli nie, tworzone są tematy w Kafka, a następnie wysyłane są wiadomości do odpowiednich tematów. W kodzie zastosowano również funkcję *createTopicIfNotExists*, która sprawdza, czy dany temat już istnieje w klastrze Kafka. Jeśli temat nie istnieje, jest on tworzony za pomocą klienta admina Kafka. Nazwy tematów są tworzone na podstawie wartości parametru *topic* pobranego z zapytania GET. Na przykład, jeśli parametr *topic* to "example", to tematy będą miały nazwy *example_age*, *example_city*, *example_gender*, *example_clicks*, co jest związane z pewną logiką aplikacji i strukturą danych, w której wiek, miasto, płeć i kliknięcia są zbierane i przetwarzane.

```
C:\Users\kryst\Desktop\Uslugi sieciowe w biznesie - projekt>node server.js
{"level":"WARN","timestamp":"2024-05-27T21:15:20.812Z","logger":"kafkajs","message":"KafkaJS v2.0.0 switched default partitioner. To retain the same partitioning behavior as in previous versions, create the producer with the option \"createPartitioner: Partitioners.LegacyPartitioner\". See the migration guide at https://kafka.js.org/docs/migration-guide-v2.0.0#producer-new-default-partitioner for details. Silence this warning by setting the environment variable \"KAFKAJS_NO_PARTITIONER_WARNING=1\""}
Serwer działa na porcie 3000
```

Rysunek 2.14: Uruchomienie serwera w aplikacji Node.js

```
C:\Windows\System32\cmd.exe - node serwer.js
Wiek: 22
Miasto: Rzeszów
Płeć: Mężczyzna
Topic: gitlab
Otrzymane dane:
Wiek: 22
Miasto: Rzeszów
Płeć: Mężczyzna
Topic: activemq
Otrzymane dane:
Wiek: 22
Miasto: Rzeszów
Płeć: Mężczyzna
Topic: kafka
Otrzymane dane:
Wiek: 27
Miasto: Kraków
Płeć: Kobieta
Topic: kafka
Otrzymane dane:
Wiek: 25
Miasto: Warszawa
Płeć: Mężczyzna
Topic: kafka
Otrzymane dane:
Wiek: 32
Miasto: Gdańsk
Płeć: Mężczyzna
Topic: kafka
```

Rysunek 2.15: Wyświetlanie informacji (podgląd) o przesłanych danych przez serwer w aplikacji Node.js

2.7. Aplikacja tworząca raport

Program napisany w języku Python dotyczy tworzenia systemu, który odczytuje wiadomości z tematów Kafka, agreguje te dane, a następnie generuje raport w formie pliku PDF zawierającego wizualizacje tych danych. Na początku skrypt importuje niezbędne biblioteki, takie jak *KafkaConsumer* i *KafkaAdminClient* do obsługi komunikacji z Kafka, *matplotlib* do tworzenia wykresów, *collections* do agregacji danych, *reportlab* do generowania dokumentów PDF, a także biblioteki systemowe do obsługi czasu i plików. Następnie skrypt definiuje konfigurację połączenia z Kafka, wskazując serwery Kafka, z którymi będzie się łączyć.

Pierwszą funkcją jest *get_all_topics*, która ma za zadanie pobranie listy wszystkich tematów z Kafka. Inicjalizuje ona klienta administracyjnego Kafka, a następnie pobiera listę

tematów. W przypadku wystąpienia błędu podczas tej operacji, funkcja zwraca pustą listę i wyświetla odpowiedni komunikat błędu.

Kolejna funkcja, *read_kafka_messages*, ma za zadanie odczytywanie wiadomości z Kafka. Najpierw pobiera wszystkie tematy za pomocą funkcji *get_all_topics*. Jeśli nie ma żadnych tematów, funkcja kończy działanie. Inicjalizuje ona konsumenta Kafka, który subskrybuje wszystkie dostępne tematy. W pętli z limitem czasowym 10 sekund konsument odczytuje wiadomości z Kafka. Jeśli nie ma nowych wiadomości, czeka na nie. Odebrane wiadomości są dekodowane z formatu bajtowego na string i dodawane do słownika *all_data*, który grupuje wiadomości według tematów. Po zakończeniu pętli konsument jest zamykany, a zebrane dane są zwracane.

Funkcja *aggregate_data* agreguje dane zebrane z Kafka. Przyjmuje ona słownik zawierający wiadomości z różnych tematów i dla każdego tematu tworzy obiekt *Counter*, który zlicza wystąpienia poszczególnych wartości. Zagregowane dane są zwracane jako nowy słownik.

Funkcja *is_file_open* sprawdza, czy plik o podanej ścieżce jest otwarty przez jakiś proces. Wykorzystuje ona bibliotekę *psutil* do iteracji przez procesy i sprawdzania otwartych plików.

Główna funkcja do generowania raportów to *save_plots_to_pdf*. Tworzy ona plik PDF o nazwie "raport.pdf" i dodaje do niego różne elementy. Sprawdza ona, czy plik PDF jest aktualnie otwarty i pomija operację, jeśli plik jest w użyciu. Następnie inicjalizuje dokument PDF i rejestruje niestandardową czcionkę obsługującą polskie znaki. Definiuje nowe style z tą czcionką.

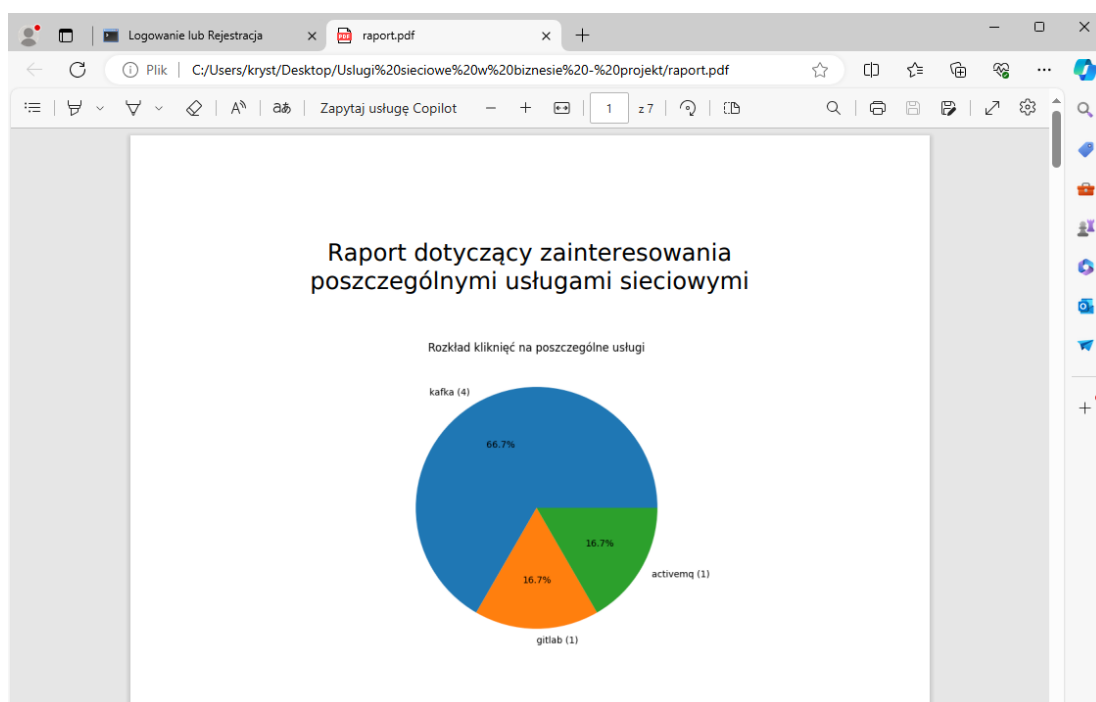
Raport zaczyna się od tytułu i nagłówka. Następnie dane kliknięć są agregowane według podstawowego tematu. Jeśli są dostępne dane kliknięć, tworzy wykres kołowy, który pokazuje rozkład kliknięć na poszczególne usługi. Plik wykresu jest tymczasowo zapisywany i dodawany do dokumentu PDF.

Dla każdego tematu (oprócz tematu "home") skrypt tworzy wykresy rozkładu danych dla wieku, miasta i płci. Wykresy te są również zapisywane jako pliki tymczasowe i dodawane do dokumentu PDF. Po zakończeniu tworzenia wykresów dokument PDF jest zapisywany, a tymczasowe pliki wykresów są usuwane.

Ostatnia funkcja, *run_report_script*, to główna pętla skryptu. Uruchamia ona odczytywanie danych z Kafka, agreguje te dane, a następnie zapisuje je w formie raportu PDF.

Pętla jest wykonywana co 60 sekund, jednak można ten czas dostosować do własnych potrzeb.

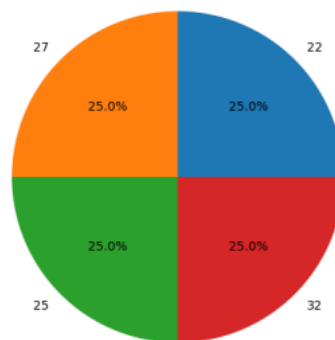
Skrypt kończy się wywołaniem funkcji `run_report_script` w bloku `if __name__ == "__main__"`, co powoduje jego uruchomienie, gdy skrypt jest bezpośrednio wywołany.



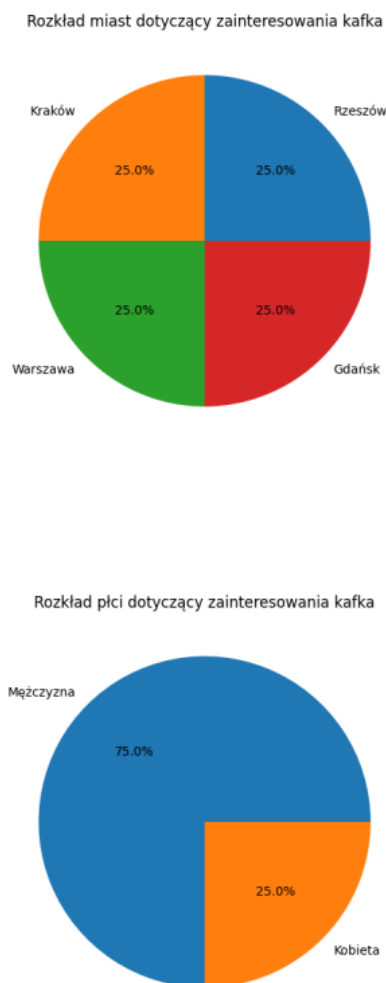
Rysunek 2.16: Wygenerowany raport cz.1

Zainteresowanie usługą kafka

Rozkład wieku dotyczący zainteresowania kafka



Rysunek 2.17: Wygenerowany raport cz.2 (wykresy dotyczące tylko tematu związanego z kafka)



Rysunek 2.18: Wygenerowany raport cz.3 (wykresy dotyczące tylko tematu związanego z kafka)

3. Podsumowanie

Projekt dotyczący analizy zachowań użytkowników na stronie internetowej przy użyciu Apache Kafka, jest kompleksowym rozwiązaniem umożliwiającym zbieranie i analizę danych o użytkownikach w czasie rzeczywistym. Integracja różnych technologii, głównie takich jak Express.js, Apache Kafka, matplotlib i reportlab, pozwala na skuteczne gromadzenie, przetwarzanie i wizualizację danych. Dzięki temu możliwe jest uzyskanie cennych informacji na temat preferencji i zachowań użytkowników, co może być wykorzystane do lepszego dostosowania usług i oferty serwisu do ich potrzeb. Rozwiązanie to umożliwia monitorowanie aktywności użytkowników w czasie rzeczywistym, ale także po-

zwala na identyfikację trendów i wzorców zachowań poprzez analizę historycznych danych. Projekt ten pokazuje, jak nowoczesne technologie mogą być używane do analizy dużych ilości danych w celu uzyskania praktycznych i użytecznych wniosków.

Literatura

- [1] <https://bulldogjob.pl/readme/apache-kafka-opis-dzialania-i-zastosowania>. (Dostęp: 25.05.2024).
- [2] <https://kafka.apache.org/quickstart> (Dostęp: 17.04.2024).
- [3] <https://www.w3schools.com/nodejs/default.asp> (Dostęp: 10.05.2024).
- [4] <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs> (Dostęp: 10.05.2024).
- [5] <https://kafka-python.readthedocs.io/en/master/usage.html> (Dostęp: 10.05.2024).