

UNIwersytet Gdański  
Wydział Matematyki, Fizyki i Informatyki

*Kierunek: INFORMATYKA*

*Specjalność: TECHNOLOGIE SIECIOWE I BAZY DANYCH*

# **Możliwości i zagrożenia wynikające z automatyzacji ludzkich zachowań w mediach społecznościowych**

**Autorzy:**

**Dariusz Gawęda – 255196**

**Patryk Kirszenstein – 260952**

Praca licencjacka napisana pod kierunkiem  
**dr inż. Arkadiusza Mirakowskiego**

**Gdańsk 2021**

## Spis treści

Wykaz pojęć.....	3
Wstęp.....	4
1. Opis problemu .....	6
1.1 Znaczenie internetowego bota .....	6
1.2 Rodzaje automatycznego oprogramowania oraz ich możliwości.....	6
1.3 Social Media Bot .....	7
1.4 Popularność w mediach .....	7
1.5 Oszczędność czasu dzięki automatyzacji .....	7
1.6 Zdobywanie popularności przez oprogramowanie vs przez człowieka .....	8
1.6.1 Eksperyment .....	9
1.7 Boty w grach MMO.....	10
1.8 Wpływy i walory używania automatyzacji w sieci Instagram .....	10
1.8.1 Generowanie ruchu w sieci .....	10
1.8.2 Ile zarabia influencer na Instagramie? .....	11
1.8.3 Metody zdobywania popularności .....	12
1.9 Koncepcja .....	14
2. Projekt Systemu.....	15
2.1. Diagram Przypadków Użycia.....	15
3. Implementacja .....	19
3.1 Architektura rozwiązania.....	19
3.2 Strona startowa.....	19
3.2.1 Walidacja .....	20
3.2.2 Komunikacja user - server .....	21
3.2.3 Uruchomienie IMAM .....	23
3.3 Komentarze.....	24
3.4 Selenium .....	26
3.5 InstaPy .....	26
3.6 Funkcjonalności bota w serwisie Instagram .....	26
3.6.1 Like By Tags.....	26
3.6.2 Following .....	28
3.7 Exit.....	29
3.8 Bazy Danych – SQL i NOSQL.....	30

3.8.3 MongoDB .....	31
3.8.4 Sposób działania .....	33
3.8.5 Inicjalizacja bazy.....	35
4. Testy .....	37
4.1 Testy Automatyczne .....	37
4.2 Testy manualne.....	37
4.3 Po co nam testy? .....	37
4.4 Metodyki pisania testów .....	38
4.5 Testy Jednostkowe - unittest.....	38
4.6 Testy Integracyjne - docker.....	39
4.7 Scenariusze testowe .....	40
Zakończenie .....	41
Podział Pracy .....	42
Bibliografia.....	43
Spis listingów .....	45
Spis rysunków .....	46
Spis tabel .....	47

## Wykaz pojęć

**fake konto** – fałszywe konto np. tworzone przez oprogramowanie

**user** – użytkownik

**hate** – rodzaj cyberprzemocy, obrażanie, ośmieszanie, poniżanie kogoś

**cyberstalking** – odmiana stalkingu, nękanie drugiej osoby przez internet

**post** – opublikowana treść w mediach społecznościowych, zawierająca film, grafikę lub tekst

**content** – opublikowana treść, zawartość

**software** – oprogramowanie

**tag** – oznaczenia tematyczne i identyfikacyjne tworzone przez użytkownika np. #mountains w opisie zdjęcia w górach

**lajkowanie** – (ang. like) lubić, w serwisach społecznościowych używane jako skrótowe określenie polubienia jakiegoś posta

**following** – słowo określające obserwowanie użytkownika w serwisie Instagram

**unfollowing** – słowo określające zaprzestanie obserwowania użytkownika w serwisie Instagram. Przeciwnie do following

**influencer** – osoba, posiadająca dużą grupę fanów, mająca realny wpływ na ich decyzje lub opinie

**frontend** – warstwa prezentacji, odpowiedzialna za wygląd i działanie interfejsu od strony użytkownika

**backend** – warstwa zawierająca logikę aplikacji

**słownik** – kolekcja przechowująca dane w parach – klucz : wartość

**postujący** – autor opublikowanego posta w serwisach społecznościowych

**pid** – identyfikator procesu

**os.kill(pid , akcja)** – metoda wysyłająca specjalny sygnał z akcją do podanego pidu procesu

**bug** – błąd, powodujący nieprawidłowe zachowanie programu

**code coverage** – jednostka określająca stopień pokrycia kodu naszej aplikacji przez testy

**przycisk submit** – przycisk odpowiedzialny za potwierdzenie i wysłanie formularza

## Wstęp

W obecnych czasach media społecznościowe cieszą się ogromną popularnością pośród wszystkich grup wiekowych. Patrząc na to, jaki ogrom możliwości nam oferują, nie ma w tym nic zaskakującego. Kontakt z innymi ludźmi, publikacje zdjęć, statusów, informacje ze świata – wszystko zebrane w jednym miejscu [27]. Media społecznościowe stały się nieodłącznym elementem życia większości ludzi na świecie.

Ile czasu dziennie spędzasz w mediach społecznościowych?	
Odpowiedzi	Liczba osób
1 do 3 godzin	63
mniej niż godzinę	82
pół dnia	23
ani minuty	32
nie wyobrażam sobie życia bez mediów społecznościowych	4

**Tabela 1. Wynik ankiety sprawdzającej aktywność użytkowników w mediach społecznościowych z dnia 2021-03-16**

Źródło: [29]

Z badania wynika, że 172 na 204 osoby, czyli 84% badanych korzysta z mediów społecznościowych codziennie. W zależności od tego, jakie treści udostępniamy narażamy się na pewnego rodzaju ryzyko. Człowiek zupełnie nam nieznany na podstawie naszego profilu może pozyskać wiele prywatnych informacji, między innymi adres zamieszkania, zainteresowania, dane osobowe. Kradzież danych, cyberstalking, cyberprzemoc czy hate to tylko wierzchołek góry lodowej. Wraz z rozwojem internetowej infrastruktury narodziły się również inne, poważne, mało zauważalne problemy.

Najbardziej podatną na te problemy grupą w social mediach są młodzi ludzie, którzy nie mają pełnej świadomości tego, jaki wpływ wywiera na nich Facebook, Instagram czy YouTube. Media społecznościowe stały się swego rodzaju trendem wśród młodzieży, co zwiększyło poziom uzależnienia od wirtualnej rzeczywistości. Dlatego oprogramowanie internetowe otwiera furtkę do wielu rozwiązań, nie tylko związanych z powszechnie znanym

promowaniem własnego wizerunku. Mogą mieć poważne konsekwencje użycia, wpływając na wiele sfer życia ludzkiego, a nawet gospodarki, co zostanie przedstawione w dalszej części niniejszej pracy.

Cała praca powstała w celach badawczych, naszym celem jest zobrazowanie możliwości i zagrożeń, jakie płyną z automatyzacji ludzkich zachowań w sieciach społecznościowych, jako przykład badań wybraliśmy serwis Instagram.

# **1. Opis problemu**

## **1.1 Znaczenie internetowego bota**

Bot internetowy to aplikacja, która odpowiada za automatyczne uruchamianie zadań w internecie. Zadania uruchamiane przez boty są zazwyczaj proste, powtarzalne i wykonywane z dużo większą częstotliwością niż akcje wykonywane przez człowieka [31]. Automatyzacja powtarzalnych procesów jest zatem swego rodzaju kupnem czasu.

Wyróżniamy wiele rodzajów botów, istniejących w różnych miejscach sieci internetowej - zaczynając od tworzenia fake kont w mediach społecznościowych, po pozyskiwanie informacji ze stron internetowych, w celu ich dalszego użycia. Najlepszymi przykładami są "GoogleBot" [32], który poprzez przeszukiwanie stron internetowych definiuje najbardziej trafną realizację naszego zapytania oraz dowolny social media bot imitujący zachowanie ludzi w mediach społecznościowych.

## **1.2 Rodzaje automatycznego oprogramowania oraz ich możliwości**

W obecnych czasach często spotykamy się z automatycznym oprogramowaniem wykorzystywanym do komunikacji - Chatboty. Jest to duże udogodnienie dla firm, które przede wszystkim pozwala obniżyć koszty. Boty komunikują się z klientami poprzez szereg zaaplikowanych komend. Ich funkcjonalności zazwyczaj kończą się na udzielaniu odpowiedzi na proste pytania konsumentów, jednak to wystarcza do udowodnienia ich użyteczności. Boty mogą być używane również w przeciwną stronę, czyli być źródłem hejtu [33] poprzez szerzenie mowy nienawiści wobec innych użytkowników lub służyć jako źródło propagandy i dezinformacji w mediach [34].

Bardzo często słyszy się o kradzieży danych osobowych ze stron internetowych, przez różnego rodzaju hakerów lub boty. Dzieje się tak przez pobieranie danych z formularzy kontaktowych lub stron logowania - jest to podstawowa funkcja botów rejestracyjnych. Oprogramowanie wykorzystuje zdobyte dane do zakładania fałszywych kont w innych serwisach internetowych.

Jednym z najgroźniejszych typów oprogramowania jest samo rozprzestrzeniający się złośliwy software, który za cel bierze nasz sprzęt. Pewne akcje użytkowników powodują, że użytkownik nieświadomie pobiera różnego rodzaju pliku powodujące rozprzestrzenienie działań bota. Skutkiem tego może być nawet kontrola nad sprzętem danego użytkownika [35].

Powyższe przykłady są tylko małą częścią ogromnej bazy. Istnieje jeszcze wiele innych typów oprogramowania, jak chociażby social media bot, będący połączeniem kilku innych rodzajów programów. Jako przykład posłuży nam autorska aplikacja IMAM (ang. Intelligent Media Auto Manager), która zostanie omówiona w dalszej części pracy.

### **1.3 Social Media Bot**

Jak sama nazwa wskazuje, są używane w mediach społecznościowych w celu zautomatyzowania zachowań ludzkich. Do konkretnych funkcjonalności możemy zaliczyć wysyłanie wiadomości, promowanie idei na różnego rodzaju grupach w postaci dodawania komentarzy pod postami, komentowanie zdjęć, wysyłanie powiadomień czy nawet sztuczne kreowanie wizerunku i zwiększenie popularności na przykładzie Instagrama, gdzie oprogramowanie followuje inne konta w celu uzyskania obserwujących na swoim profilu.

### **1.4 Popularność w mediach**

W obecnym czasie popularność w mediach stała się ważnym aspektem życia w społeczeństwie. Idziemy za trendami społeczności wprowadzając zachowania innych w nasze życie. Widząc informacje o kolejnych milionach nowych użytkowników na Instagramie, nie chcemy czuć się odosobnieni lub gorsi, więc zakładamy własne konta, które potem wypełniamy poprzez publikację postów, zdjęć lub relacji. Powodem tego jest instagratyfikacja. Reakcje pod zdjęciem pojawiają się od razu, bez czekania zostajemy zasypani nagrodami w postaci komentarzy i lajków, a chcąc wyróżnić się na tle innych wrzucamy jeszcze więcej contentu, gdyż to wiąże się z szybkimi pozytywnymi doświadczeniami, które po pewnym czasie mogą nawet zamienić się w nawyk.

Poza insta-gratyfikacją, media tworzą nam odpowiednie środowisko do łatwego monitorowania postępów. Codziennie widzimy jak licznik followersów, znajomych i lajków pod zdjęciami rośnie wraz w opublikowaną zawartością, co sprawia, że jeszcze chętniej korzystamy z dostępnych nam platform [36].

### **1.5 Oszczędność czasu dzięki automatyzacji**

Automatyzacja towarzyszy nam od setek lat, człowiek zawsze dążył do tego, aby zautomatyzować wszystkie powtarzalne czynności, które musi wykonywać. Powtarzanie tego samego zadania w dłuższym odcinku czasu zawsze w pewnym momencie staje się monotonne, a następnie męczące.



Największym przykładem tego jak ważna jest automatyzacja pokazuje rewolucja przemysłowa [9], która zapoczątkowała ten proces, przez zastąpienie pracy ręcznej człowieka na prace maszyn. Jak widać automatyzacja i mechanizacja są nieoderwalną częścią naszego życia. Sam pomysł rewolucji wynikał w głównej mierze z chęci zastąpienia człowieka w procesie twórczym maszynami. Pozwoliło to w dłuższym czasie ludziom zająć się czynnościami mniej niebezpiecznymi oraz ciekawszymi, takimi, którymi maszyna nie była się w stanie zająć ponieważ wymagała czynnika ludzkiego.

Społeczeństwo nieprzerwanie dąży do lepszego wykorzystania zasobów ludzkich, a co za tym idzie zlikwidowania powtarzalnej i nieprzyjemnej pracy. Co prawda nasz program nie zastąpi człowieka w hucie ani szwalni jednak też nie do tego został stworzony. Od rewolucji minęło już 250 lat a co za tym idzie cele i narzędzia automatyzacji uległy zmianom i postępowi. W tym okresie powstały nowe zawody takie jak influencer, których praca wciąż może zostać przyspieszona oraz ułatwiona. Praca influencera polega na mozolnym i powtarzalnym przeglądaniu setek zdjęć, pisaniu kilku powtarzalnych komentarzy czy rzucaniu obserwacjami na lewo i prawo w celu zwrócenia na siebie uwagi. Człowiek taki z każdym dniem spędza około 6 godzin na pracy, którą mógłby wykonać BOT, dodatkowo też byłby dużo bardziej wydajny. Jeżeli tą samą pracę można wykonać o wiele prościej za pomocą komputera, to zdecydowanie powinniśmy skorzystać z takich udogodnień.

## 1.6 Zdobywanie popularności przez oprogramowanie vs przez człowieka

Zdobywanie popularności na Instagramie wiąże się z systematyczną aktywnością. Aktywność przekłada się na liczbę minut lub godzin spędzonych w serwisie.

Średnia czasu spędzonego na Instagramie w ciągu jednego dnia	
Osoby Poniżej 25 roku życia	Osoby powyżej 25 roku życia
32 minuty	24 minuty

**Tabela 2. Średnia czasu spędzonego na Instagramie przez użytkowników**

Źródło: [39]

Systematyczna aktywność ma przelicznik również w liczbie obserwujących, co z kolei prowadzi do wzrostu popularności. Jednak jak wyglądają statystyki, gdy skorzystamy z Bota?

### 1.6.1 Eksperyment

Na podstawie eksperymentu przeprowadzonego przez jednego z użytkowników Instagrama, dotyczącego porównania zdobywania obserwujących przez oprogramowanie i przez człowieka, uzyskaliśmy pogląd, dlaczego software tego typu jest stosowany.

Eksperyment polegał na obserwacji, jaki wpływ na konto użytkownika ma użycie automatycznego oprogramowania i porównanie go do normalnej aktywności jaką owy użytkownik prowadził na swoim głównym koncie. W tym celu stworzył drugi profil, na którym używał bota, gdzie spędzał na nim nie więcej niż 5 minut tygodniowo, czyli 43 sekundy w ciągu dnia. Patrząc na spędzony czas można wysnuć pewne wnioski i odkryć jeden z głównych powodów stosowania bota, co zostanie wyjaśnione w dalszej części pracy.

Konto sterowane przez bota	Konto główne sterowane przez człowieka	
2 lata	4 lata	Czas prowadzenia konta
11 400	5 800	Ilość obserwujących (podana w przybliżeniu)
269	409	Ilość obserwowanych
365	696	Ilość opublikowanych postów

**Tabela 3. Porównanie statystyk**

Źródło: [10]

Dane pochodzą z 6 kwietnia 2017 roku i w tamtym czasie konto miało już 4 lata porównując do konta-bota, mającego 2 lata. Jak widać w powyższej tabeli ilość obserwujących na obu kontach niebotycznie się różni - 11,4 tysiąca do 5,8 tysięcy obserwujących. Warto dodać, że akcje oprogramowania polegały jedynie na dawaniu lajków i pisaniu komentarzy pod opublikowaną treścią innych osób. Według danych zawartych w artykule bot lajkował 30 tys. postów oraz dodawał około 7 tys. komentarzy miesięcznie.

Autor podczas swojego eksperymentu obserwował również inne konta-boty, które w tym samym czasie potrafiły zdobyć nawet do 50 tysięcy obserwujących, co obrazuje siłę działania automatycznego oprogramowania [10].

Jednak liczba obserwujących to nie wszystko. Aby w pełni wykorzystać możliwości konta z tak pokaźną ilością followersów nie możemy opierać się wyłącznie na

oprogramowaniu, gdyż jak autor słusznie zauważył - duża ilość followersów również była botami, więc widoczna liczba obserwujących na naszym profilu nie jest wiarygodną liczbą osób, które nas obserwują.

## **1.7 Boty w grach MMO**

W grach typu MMO (ang. Massive Multiplayer Online) tworzenie botów potrafi zniszczyć całą ekonomię, oraz doprowadzić grę do ruiny. Przykładów jest wiele, zwykle koncepcja ta sama - automatyzacja powtarzalnych zwykle nużących procesów.

Jednym z podstawowych botów na serwerach gier typu MMO są exp boty (ang. Experience), czyli oprogramowanie stworzone do zdobywania doświadczenia dla naszej postaci, bez ingerencji użytkownika. Jak łatwo się domyślić, w grach nastawionych na zdobywanie doświadczenia jest to wielki problem, gdyż niszczy równowagę wśród graczy - zaburza ranking użytkowników niszcząc zdrową i ważną dla tego typu gier rywalizację, co z kolei eliminuje sens z rozgrywki, gdyż nie każdy gracz ma równe szanse.

Ogólnie rzecz biorąc boty w grach MMO są tworzone głównie przez wzgląd na dużą ilość powtarzalnych czynności, nazywane przez graczy jako "farmienie", czyli zdobywanie nie tylko doświadczenia, ale i przedmiotów lub innych form waluty w grze. Farmienie zwykle wiąże się z długimi godzinami siedzenia przed komputerem przy wielu cyklach powtarzalnych czynności [11].

W 2005 roku można było zauważyć ciekawe zjawisko, gdzie blisko 100 000 graczy z Chin było zatrudnionych do "farmienia" przedmiotów lub doświadczenia w grach MMO, przez co w tamtym czasie można było zaobserwować wysoki wzrost wirtualnych walut, gdzie gracze wymieniali prawdziwe pieniądze na przedmioty w grach [12]. Porównanie tej sytuacji, czyli działania 100 000 osób, pomiędzy działaniem kilku botów tworzy pogląd, jak olbrzymi postęp technologiczny osiągnęliśmy oraz jak bardzo sztuczna inteligencja zastępuje ludzką aktywność.

## **1.8 Wpływy i walory używania automatyzacji w sieci Instagram**

### **1.8.1 Generowanie ruchu w sieci**

*"Według badań przeprowadzonych przez Imperva Incapsula (The Bot Traffic Report 2016) 48,2% ruchu w sieci generowane jest przez ludzi, a 51,8% to efekt działania botów, zarówno tych dobrych, jak i złych.*

*Dobre boty, które odpowiadają za 22,9% ruchu, przede wszystkim są odpowiedzialne za kompletowanie treści, na przykład w wyszukiwarkach. Indeksują także dane z sieci oraz sprawdzają, czy wszystkie funkcjonalności na stronie działają prawidłowo. Niespełna 3% ruchu generowanego przez boty to praca robotów wykorzystywanych w marketingu. Natomiast przeszło 24% to efekt działania botów podszywających się pod użytkowników. Ten rodzaj najczęściej wykorzystywany jest do ataków na serwery. Pod uwagę wzięto 100 000 losowo wybranych domen.”[13]*

Ruch w sieci			
Ludzie	Dobre Boty	Złe boty	Suma
48,2 %	22, 9 %	28, 9 %	100 %
	51,8 %		

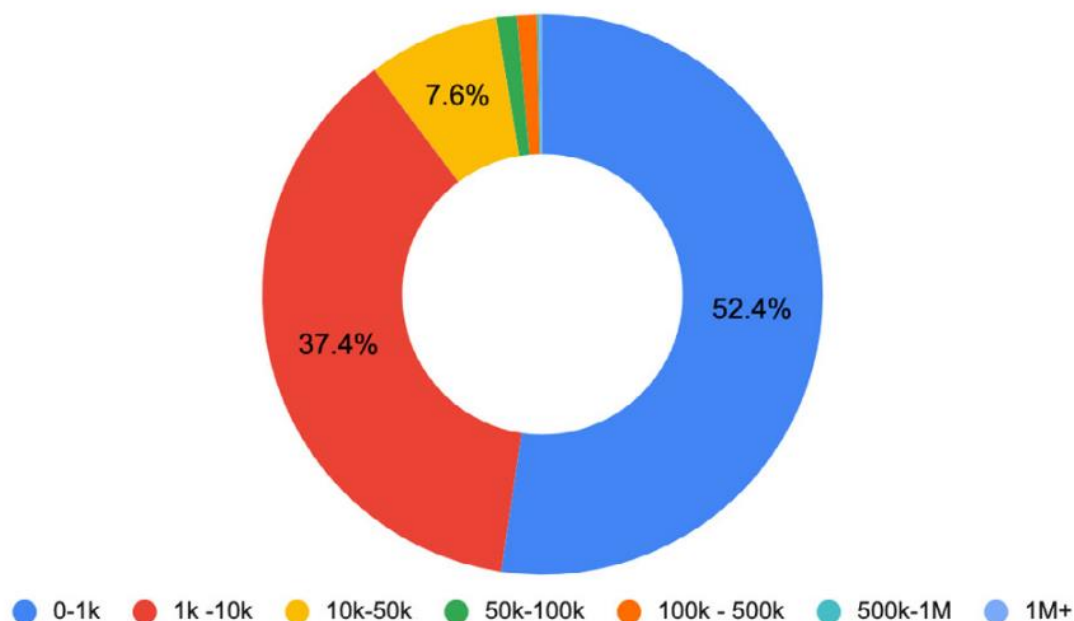
**Tabela 4. Zestawienie ruchu w sieci**

Źródło: [14]

### **1.8.2 Ile zarabia influencer na Instagramie?**

Według Instagram Engagement Raport przeciętny influencer posiadający 500 tysięcy fanów może liczyć na zarobek rzędu 4500 zł za post [15]. Zakładając tworzenie jednego postu dziennie możemy dojść do naprawdę niebagatelnych kwot rzędu setek tysięcy złotych. 500 tysięcy fanów to naprawdę dużo jednak najwięksi influencerzy posiadają ich dziesiątki milionów.

## Types of Influencers



**Rysunek 1. Podział użytkowników, ze względu na liczbę obserwujących.**

Źródło: [15]

Jak widać z wykresu nano-influencerzy (0-1k) oraz micro (1-10k) stanowią 90% wszystkich influencerów. Obie grupy są celem naszego programu, zatem bazując na powyższych danych, liczba kont-botów na instagramie sięgnęła 95 milionów na 1 bilion użytkowników, co daje około 9.5% kont-botów na całą społeczność Instagrama [16].

Oczywiście o ile automatyzacja zachowań sprawdzi się w skali mikro, tak niestety w makro może już tylko działać jako wsparcie. Rodzaje automatyzacji, jakie występują, mogą być różne. Zaczynając od spamowych zachowań, nachalnych botów, przez falsyfikacje masową, po umożliwienie przeciętnemu użytkownikowi uniknięcia powtarzania monotonnych i nużących czynności. Ostatnia opcja jest tą, którą zajmujemy się w naszej pracy.

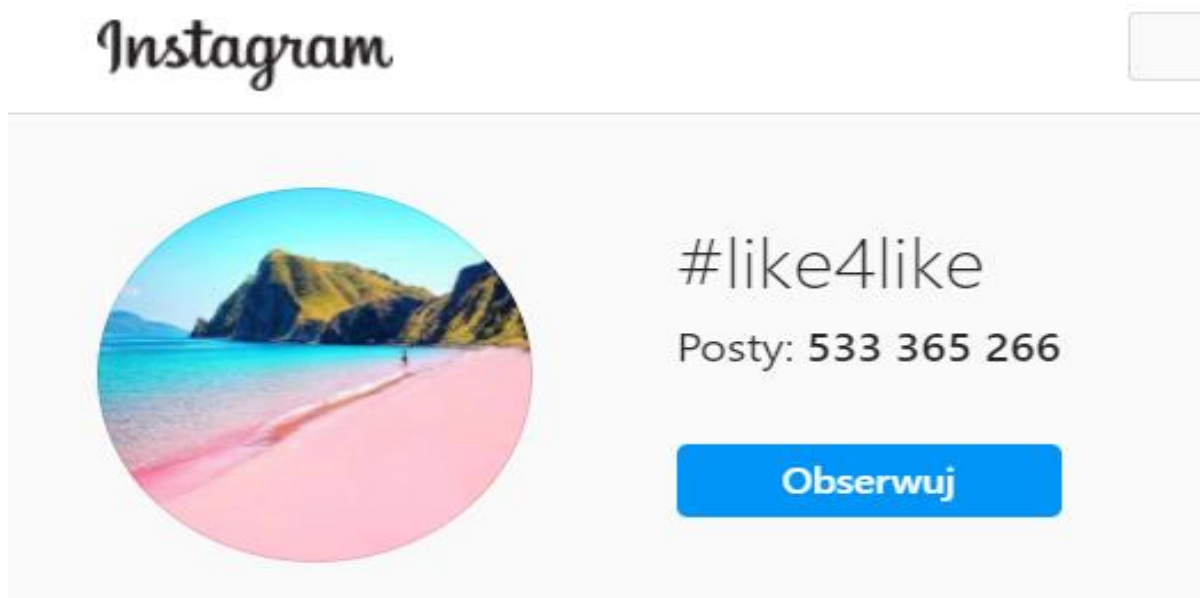
### 1.8.3 Metody zdobywania popularności

Na instagramie jest wiele dróg do zdobycia sławy, są pewne metody i zachowania, które należy i powinno się stosować aby do niej dojść. Jedną z nich jest komentowanie treści, każdy pozostawiony komentarz pod czyimś postem jest widziany przez osobę komentowaną, dodatkowo każda osoba odwiedzająca skomentowany post także ma szansę na spostrzeżenie naszego komentarza, co może skutkować odwiedzeniem naszego profilu.

*“Aby zdobyć popularność w IG, musisz być komentatorem, a nie tylko cichym obserwatorem. Nawet jeśli łatwiej jest coś polubić, idź o krok dalej i skomentuj zdjęcie, aby*

*osoba zobaczyła, że się angażujesz. To z kolei sprawi, że będą chcieli zrobić to samo dla Ciebie. Pozostaw znaczące komentarze na temat profilu, a najprawdopodobniej uzyskasz w zamian więcej” [17].*

Kolejną metodą tworzenia aktywności na naszym profilu jest “Like4like”. Jest to jedna z wielu metod, używanych do zdobycia dodatkowych polubień pod zdjęciem. Użytkownicy instagrama umieszczają pod swoimi zdjęciami tagi, które mają na celu określenie kategorii danego zdjęcia, wystarczy że przed frazą opisującą zdjęcie dodamy “#” (hash). Metoda L4L polega na wzajemnej wymianie lajków pod zdjęciami. Osoba która polubi zdjęcie z tagiem “#l4l” otrzyma polubienie zwrotne. Zwykle zamiarem takiego działania jest zdobycie polubień od osób, które nas nie obserwują. Z racji, że owy tag jest bardzo popularny nasze zdjęcie może trafić do szerokiego grona odbiorców, co skutkuje zwiększeniem liczby odbiorców naszego profilu. Jednak nie jest to zbyt dobra opcja na pozyskanie wartościowych odbiorców. Osoba polubiła post ze zdjęciem tylko ze względu na tag, co wygląda dobrze patrząc na liczbę polubień, jednak nie powoduje ogromnego zwiększenia liczby obserwujących [19].



**Rysunek 2. Liczba postów oznaczonych tagiem like4like**

Źródło: [18]

Podobnym przykładem do zdobycia popularności jest tag “Follow4Follow”, czyli obserwacja za obserwację. Mimo, że działa podobnie i powoduje faktyczne zwiększenie liczby obserwujących, nie jest to do końca dobra metoda, gdyż na naszym profilu przybywa

osób, którym raczej nie do końca zależy na publikowanych przez nas treściach, a raczej na zdobyciu dodatkowego obserwatora.

Podstawą zdobycia zasięgów w sieci jest regularność, czyli cykliczne wrzucanie treści na nasz profil. To spowoduje, że będziemy przyzwyczajać naszych obserwujących do zaglądania na nasz profil, co zwiększy aktywność naszego profilu.

Jest jeszcze kilka metod, które pozwolą naszemu profilowi się rozwinąć, jednak nie są one ściśle powiązane z działaniami naszego oprogramowania, dlatego skupimy się na powyżej wymienionych, gdyż te są podstawowym środkiem napędowym naszej aplikacji. Używanie odpowiednich hashtagów oraz komentowanie zdjęć innych jest w stanie zapewnić nam duży wzrost popularności w krótkim okresie czasu. Dowodem na to jest opisany wyżej eksperyment, gdzie używany bot stosuje podobne metody do zdobycia popularności.

## **1.9 Koncepcja**

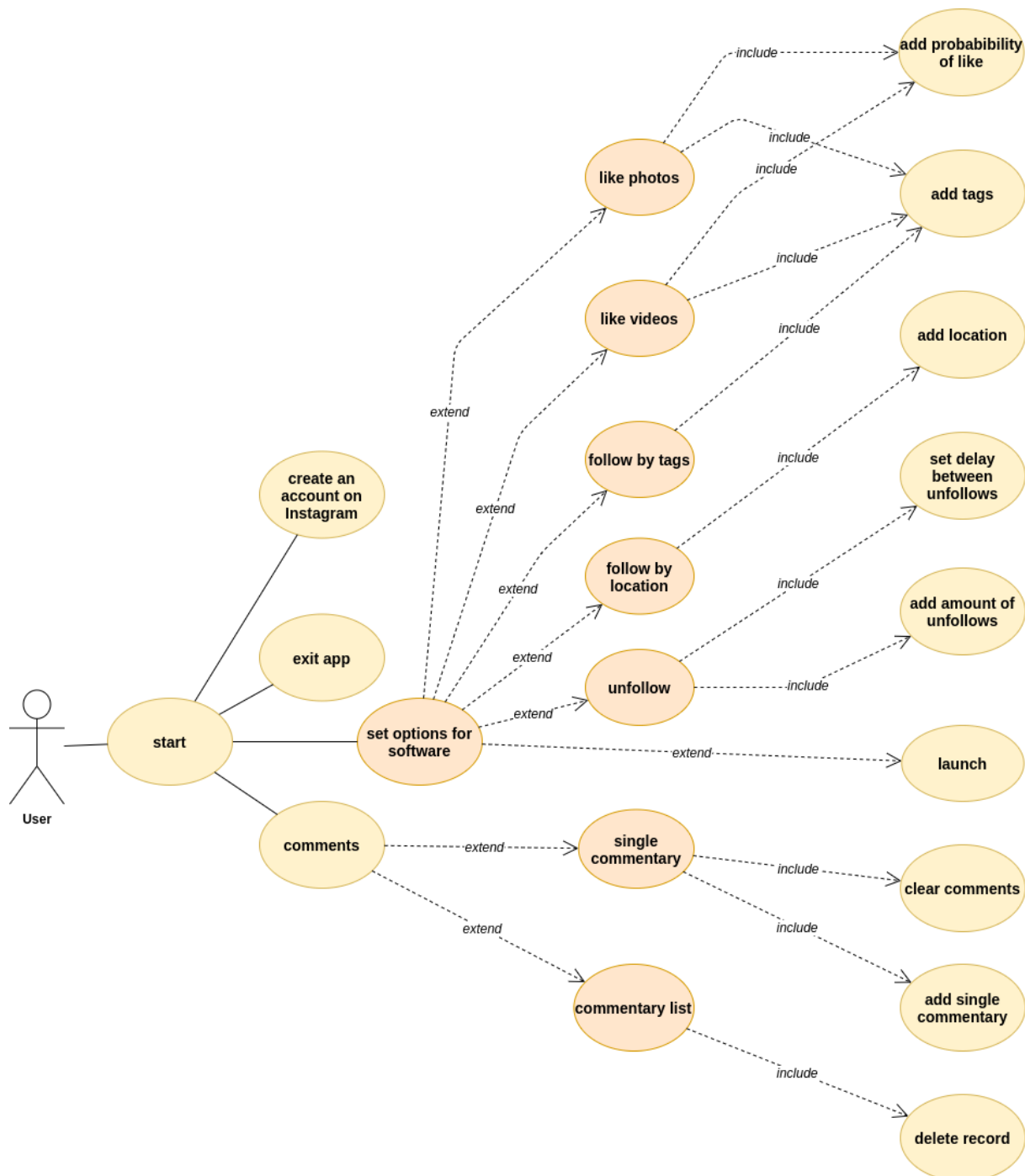
Powyższa część pracy przedstawia ogólną teoretyczną koncepcję automatycznego oprogramowania występującego w sieci - przykłady użycia oprogramowania i jego rodzaje zostały omówione na bazie ogólnych przykładów botów powszechnie stosowanych w internecie.

Przedstawiona do tej pory koncepcja miała na celu określenie poglądu na temat automatycznego oprogramowania. Posiadając wiedzę o zarówno dobrych jak i złych stronach omawianego software'u, jesteśmy w stanie przejść do głównego celu niniejszej pracy, czyli omówienia autorskiego automatycznego oprogramowania, jakim jest IMAM - Intelligent Media Auto Manager.

## 2. Projekt Systemu

Aplikacja IMAM jest oprogramowaniem służącym do automatyzacji procesów użytkownika w jednym z mediów społecznościowych, jakim jest Instagram. Jest to oprogramowanie zaliczane do grona jednej z wielu grup - Social Media Bot.

### 2.1. Diagram Przypadków Użycia



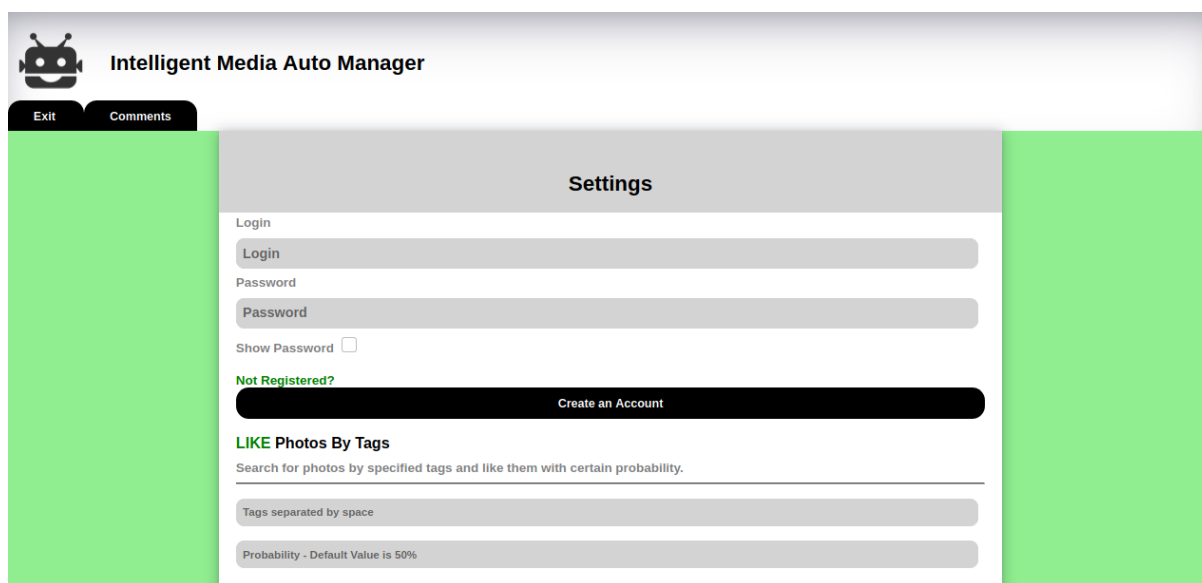
Rysunek 3. Diagram przypadków użycia strona pierwsza

Źródło: [20]



## 2.2. Interfejs użytkownika

Poniżej zostały przedstawione widoki interfejsu użytkownika.



**Rysunek 4. Widok interfejsu użytkownika zaraz po uruchomieniu aplikacji**

Źródło: Opracowanie własne

### LIKE Videos By Tags

Search for videos by specified tags and like them with certain probability.

car

100

### FOLLOW By Location

Follows users by specified locations

Locations separated by space

### FOLLOW By Tags

Follows users by specified tags.

Tags separated by space

### UNFOLLOW

Unfollow an amount of users

Amount

Time in minutes

### COMMENT

Allows to comment posts by specified list

AUTHOR → Darek DESCRIPTION → for bmw cars TAGS → #bmw COMMENTS → "love <3" " :) " "Bmw are t v

Launch

## Rysunek 5. Interfejs użytkownika, część druga

Źródło: Opracowanie własne

Comments Setting

Add Comment

Your comment...

Add Comment

Clear Comments

Comments to Add

Author

Place your name here...

Description

Your description of the comment setting...

Tags

Place your tags here...

Added

Author	Description	Tags	Comments	Action
Darek	Do testu	#Tag1 #tag2 #Tag3	"Przykładowy komentarz" "Przykładowy komentarz2"	DELETE
Darek	Do testu	#tag1	"Przykładowy3"	DELETE

Add Comment Set

Rysunek 6. Interfejs strony odpowiedzialnej za dodawanie zestawów komentarzy

Źródło: Opracowanie własne

18

## **3. Implementacja**

### **3.1 Architektura rozwiązania**

Cała architektura oprogramowania opiera się na języku python3 oraz jego frameworków, takich jak Django i Selenium.

Django biblioteka języka python odpowiada za skuteczne tworzenie aplikacji webowych. Szczególnie popularne przez tzw. “django admin”, które generuje cały fundament backendowy projektu, przez co z łatwością możemy ją rozbudowywać. To idealnie sprawdziło się w naszym projekcie - poprzez django byliśmy w stanie zapewnić prostą acz skuteczną wymianę informacji między użytkownikiem a serwerem. Poza dynamicznym elementem - takim jak przycisk wyjścia - użytkownik komunikuje się poprzez formularz. Jako miejsce składowania danych użyliśmy mongoDB, które okazało się najbardziej odpowiednią bazą danych dostosowaną do naszego projektu. Dalsze podrozdziały rozszerzą tematykę poruszoną powyżej.

Selenium, czyli framework wykorzystany przez nas do automatycznej obsługi przeglądarki internetowej. To nasz główny motor napędowy na backendzie - jest częścią narzędzia, które zostało wykorzystane w naszej aplikacji.

InstaPy, czyli narzędzie, do którego stworzenia zostało użyty framework Selenium. Wszelkie funkcjonalności instaPy zostały przeznaczone do obsługi popularnego serwisu społecznościowego Instagram. W związku z tematem niniejszej pracy owe narzędzie okazało się nieodłącznym elementem naszego projektu. Sposoby implementacji powyższych technologii zostaną zbliżone w dalszej części pracy.

### **3.2 Strona startowa**

Jest to epicentrum aplikacji zawierające formularz odpowiedzialny za działanie oprogramowania. Formularz stanowi podstawowy środek komunikacji między serwerem i użytkownikiem, wymaga jedynie ustawienia opcji, od których zależne jest działanie bota na Instagramie. Ustawianie to nic innego jak wypełnianie odpowiednio podpisanych pól tekstowych formularza lub wybieranie z listy gotowych rozwiązań - jak w przypadku opcji komentarzy. To sprawia, że aplikacja jest ukierunkowana dla każdego - jedynym wymogiem jest posiadanie konta w serwisie Instagram.

Bot zawiera kilka rodzajów aktywności, które zostają użyte w zależności od woli użytkownika - lajkowanie zdjęć lub filmów, obserwowanie użytkowników lub zaprzestanie

obserwacji użytkowników. Co więcej, każda z wymienionych powyżej opcji zawiera składowe odpowiedzialne za sposób działania.

Poza podstawowymi opcjami musimy wypełnić pola odpowiadające danym konta na Instagramie, czyli login i hasło. Jest to nieodłączny element działania, bez którego aplikacja nie będzie w stanie zidentyfikować konta na Instagramie, czego skutkiem będzie zakończenie działania programu z komunikatem o błędnych danych logowania.

### 3.2.1 Walidacja

Do stworzenia strony startowej wykorzystaliśmy HTML5, CSS wraz z JavaScript oraz jej biblioteką - jQuery. Są to główne elementy odpowiedzialne za frontend. Biblioteka jQuery przez swój funkcyjny charakter umożliwiła nam wygodną i prostą implementację animacji oraz walidację formularza zgłoszeniowego. Poniżej przykład metody napisanej przy pomocy jQuery:

```
$(document).ready(function(){
    $(".launch").on({
        mouseenter : function(){
            $(this).css({backgroundColor: "gray", color: "black"});
        },
        mouseleave : function(){
            $(this).css({backgroundColor: "black", color: "white"});
        },
        click : function(){
            $(this).css({"color": "green", "background-color": "gray"});
        }
    });
})
```

**Listing 1. Przykład metody w jQuery z animacją przycisków**

Źródło: Opracowanie własne

Aby uniknąć zbędnego obciążenia od strony serwera,

Zdecydowaliśmy się na walidację danych po stronie klienta. Implementując walidację tego typu, zapewniamy kompleksową obsługę błędów w czasie rzeczywistym, bez potrzeby korzystania z requestów serwera. W momencie wpisania danych niezgodnych z formatem opcji<sup>1</sup>, przycisk submit staje się nieaktywny oraz pod polem, w którym popełniliśmy błąd pojawia się stosowny komunikat naprowadzający użytkownika na poprawne dane.

---

<sup>1</sup> np. do Probability wpiszemy litery

**LIKE Photos By Tags**  
Search for photos by specified tags and like them with certain probability.

++

Tags can only contain letters, numbers, and underscores

test

Probability must be a number between 1 and 100

**LIKE Videos By Tags**  
Search for videos by specified tags and like them with certain probability.

TAG1

Probability - Default Value is 50%

This field can't be empty

**Rysunek 7. Przykład działania walidacji**

Źródło: Opracowanie własne

Każde pole zostało odpowiednio obsługane, również parami, czyli jeśli jedno z pól zostało poprawnie wypełnione do działania wymagana jest również druga opcja, w innym wypadku przycisk uruchomienia bota pozostanie nieaktywny. Taki sposób badania poprawności wprowadzanych danych uniemożliwia wystąpienie błędu.

### 3.2.2 Komunikacja user - server

Dane formularza są przesyłane przy użyciu jednego z dwóch metod HTTP, czyli metodą POST, gdzie są uprzednio kodowane i przesyłane do serwera. Od strony serwera widoki odpowiednio wychwycą treść komunikatu, przez kierunek nadany w formularzu.

```
<form action="/Start/" method="post" id="starting_form">
```

**Listing 2. Kierunek akcji formularza**

Źródło: Opracowanie własne

Dane formularza mają określoną ścieżkę, po której informacje są odczytywane - każda ścieżka, musi mieć przypisaną akcję, czyli widok, który ją obsłuży i zwróci wartość do szablonu aplikacji, czyli strony html.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', csrf_exempt(views.main), name = "Main"),
    path('Start/', csrf_exempt(views.botSettings)),
    path('Exit/', views.exit, name = "Exit"),
    path('Comments/', views.comments, name = "Comments"),
    path('CommentsAdd/', views.add_comment_record, name = "Comments_Setting"),
    path('AddSingleComment/', views.add_single_comment, name =
"Single_Comment"),
    path('ClearComments/', views.clear_comments, name = "Clear"),
    path('Delete/<slug:_id>', views.delete_record, name = "Delete")
]
```

**Listing 3. Kod źródłowy urls projektu - ścieżki oraz przypisane do nich widoki**  
Źródło: Opracowanie własne

Widoki operują na różnego rodzaju obiektach do zwracania odpowiedzi w kierunku użytkownika. W IMAM najczęściej korzystaliśmy z obiektu render, który generuje zawartość szablonu, przekazując przetworzone dane do warstwy interfejsu, tym sposobem użytkownik widzi wynik swojego żądania.

Aby jeszcze bardziej przybliżyć sposób działania naszej aplikacji skupimy się na konkretnych przykładach ustawień, użytych do działania oprogramowania. Wiemy już, że wysłanie formularza do serwera skutkuje przesłaniem danych do widoku, jednak w jaki sposób dane zostają przechwycone?

```
def get(request):
    settings = {
        'username': request.POST.get('username'),
        'password': request.POST.get('password'),
        'photo_tags': request.POST.get('like_photo_tag'),
        'photo_prob': request.POST.get('like_photo_probability'),
        'video_tags': request.POST.get('video_tag'),
        'video_prob': request.POST.get('like_video_probability'),
        'location': request.POST.get('location'),
        'follow_tags': request.POST.get('follow_tags'),
        'non_followers_amount': request.POST.get('non_followers_amount'),
        'non_followers_delay': request.POST.get('non_followers_delay'),
        'new_followers_amount': request.POST.get('new_followers_amount'),
        'new_followers_delay': request.POST.get('new_followers_delay'),
    }
    if check_if_null(settings):
        settings = True
    return settings
```

**Listing 4. Przykład przechwytywania danych z formularza**  
Źródło: Opracowanie własne

Powyższy listing doskonale obrazuje sposób przechwycenia informacji - metoda GET, czyli druga metoda protokołu HTTP wykorzystywana do pobierania danych. Serwer wykorzystuje ową metodę, aby uzyskać wartości każdego pola formularza. Jak możemy zauważyć pola zostały umieszczone w słowniku, co ma swoje dwa powody, czyli łatwość przesyłu i odczytu między metodami oraz obiekt render, który wykorzystuje słowniki do przesyłu informacji. Jest to bardzo wygodna metoda, gdyż każdy rekord słownika zawiera unikalny klucz, dzięki czemu identyfikacja pola w szablonie jest bardzo prosta.

```
<h3 class="error">{{ login_info }}</h3>
```

**Listing 5. Przykład pola umieszczonego w szablonie aplikacji z kluczem słownikowym**

Źródło: Opracowanie własne

Ważnym aspektem identyfikacji są nawiasy klamrowe między którymi umieszczamy klucz, dzięki czemu użytkownik zobaczy znajdującą się pod nim wartość w zależności od żądania jakie wysła do serwera (w tym przypadku zawartość kryjąca się pod kluczem zostanie wyświetlona w momencie błędu logowania).

```
<input type="text" name="author_name" id="author_name">
```

**Listing 6. Przykład nazwanego pola**

Źródło: Opracowanie własne

Tu z kolei mamy przykład pola, z którego dane są pobierane - identyfikacja odbywa się przy pomocy atrybutu nazwy - zawartość tego pola zostanie wysłana do serwera i ten metodą GET przechwytuje żądanie.

### 3.2.3 Uruchomienie IMAM

W poprzednich podrozdziałach wyjaśniliśmy sposoby komunikacji użytkownika z serwerem, jednak co dzieje się po przekazaniu i przetworzeniu owych danych? Aby bot rozpoczął działanie, poza przekazaniem danych musimy go odpowiednio uruchomić - w tym celu skorzystaliśmy z pomocy narzędzia InstaPy, które zostanie omówione w dalszych rozdziałach.

Aby rozpocząć działanie IMAM uruchamiamy sesję z parametrami przekazanymi od użytkownika, które zostaną wykorzystane podczas logowania do serwisu Instagram. Następnie po pomyślnym zalogowaniu, reszta pozyskanych parametrów z formularza zostaje rozdysponowana według funkcjonalności - jeśli jakieś pole formularza pozostało puste, bot



nie będzie korzystał z tej funkcjonalności, zatem wykorzystuje jedynie poprawnie wypełnione właściwości odpowiadające konkretnym funkcjom. Po stworzeniu sesji, InstaPy uruchomi przeglądarkę w nowym oknie i otworzy serwis Instagram ze stroną logowania wykorzystując uzyskane dane od użytkownika. Warto wspomnieć, że nie musimy doglądać widoku przeglądarki, ponieważ program może sam działać w tle, wystarczy mała zmiana parametru sesji InstaPy, co zostało przedstawione poniżej.

```
session = InstaPy(username=settings['username'],  
                  password=settings['password'],  
                  headless_browser=True)
```

**Listing 7. Parametry ustawień sesji InstaPy aby działała w tle**  
Źródło: [21]

Główną komendą odpowiedzialną za działania w tle jest `headless_browser` ustawione na `True`. Dalszą obsługą żądania użytkownika zajmuje się `instaPy`. Narzędzie korzystając z biblioteki Selenium oraz języka Python, kolejno realizuje funkcjonalności bota wcześniej zdefiniowane przez użytkownika.

### 3.3 Komentarze

Jako drugi środek komunikacji w mediach społecznościowych wykorzystujemy komentarze, czyli krótkie opisy pod postami innych użytkowników stanowiące prolongację kontaktów między ludźmi. Każdy komentarz pod zdjęciem lub filmem może doprowadzić do dalszej ciekawej rozmowy z twórcą posta, przez co ten zainteresuje się Twoim profilem. To z kolei sprowadza się do zwiększenia popularności, gdyż zdobyliśmy dodatkowego obserwatora - przez jeden komentarz. Oczywiście ilość komentarzy nigdy nie będzie wprost proporcjonalna do zdobywania obserwacji, lecz jest to dobry sposób na zwrócenie czyjejs uwagi.

Na potrzeby powyższej funkcjonalności stworzyliśmy dodatkowy moduł, który umożliwia użytkownikowi działanie na rekordach bazy danych przechowujących informacje o zestawach komentarzy. Każdy rekord to inny zbiór komentarzy dotyczący innej tematyki posta. Dokładny widok rekordu bazy został przedstawiony poniżej.

Author	Description	Tags	Comments	Action
Darek	Do testu	#Tag1 #tag2 #Tag3	"Przykładowy komentarz" "Przykładowy komentarz2"	DELETE
Darek	Do testu	#tag1	"Przykładowy3"	DELETE

**Rysunek 8. Przykładowy rekord bazy danych**

Źródło: Opracowanie własne

Jak w przypadku każdego rekordu bazy, musi on posiadać informacje, dzięki której jesteśmy w stanie zidentyfikować i odróżnić rekordy między sobą, dlatego poza podanymi powyżej danymi istnieje jeszcze specjalne pole z numerem identyfikacyjnym. Pole jest podstawą do operowania na zasobach bazy danych, gdyż po id jesteśmy w stanie odnaleźć każdy rekord.

Dodawanie zestawu komentarzy to odpowiednie wypełnienie pól formularza - komunikacja odbywa się na podobnych zasadach jak w przypadku strony startowej. Aby dodać zbiór komentarzy, najpierw należy dodać treści komentarzy.

#### Comments to Add

"Sample" "Sample2"

**Rysunek 9. Przykładowe komentarze do dodania**

Źródło: Opracowanie własne

Ta metodyka dodawania komentarzy znacząco ułatwia obsługę zarówno dla serwera jak i użytkownika, ze względu na możliwość dodania kilku komentarzy oraz niemożność popełnienia błędu przez użytkownika, gdyż ten nie musi się martwić o stosowanie odpowiednich znaków, np. cudzysłowie, będący swoistą granicą rozdzielającą ciągi - to byłoby niepraktyczne.

### **3.4 Selenium**

Selenium to zbiór narzędzi do automatyzacji działań w przeglądarkach internetowych, jednak nie jest to jedyne przeznaczenie jakie posiada. Biblioteki możemy użyć również do tworzenia testów aplikacji webowych na różnego rodzaju przeglądarkach, web scrapingu, czyli pobieraniu danych ze stron internetowych, przeglądanie stron internetowych i wiele innych. Ciekawostką jest, że selenium jest również używane do tworzenia działań dla web crawlerów, o których wspominaliśmy we wcześniejszych podrozdziałach [3].

### **3.5 InstaPy**

Jest to główne narzędzie w naszej aplikacji zapewniające obsługę żądań użytkownika na Instagramie, począwszy od stworzenia sesji, aż do jej zakończenia, czyli po wykonaniu wszystkich planowanych akcji usera. InstaPy zawiera zbiór aktywności i różnego rodzaju działań zapożyczonych z frameworka selenium - otwieranie przeglądarki, wyszukiwanie elementu na stronie, akcja na przycisku i wiele, wiele innych. Wszelkie pomniejsze aktywności Selenium zostały złożone w duże funkcjonalności typowe dla Instagrama, czyli lajkowanie, komentowanie postów oraz following i unfollowing użytkowników. Samo narzędzie używa GraphQL - jest to język zapytań stworzony przez firmę Facebook, pozwala na pozyskiwanie oraz udostępnianie danych protokołami HTTP [22]. Wszystko złożone w logiczną i spójną całość jako jedno kompleksowe narzędzie.

### **3.6 Funkcjonalności bota w serwisie Instagram**

IMAM zapewnia użytkownikowi kilka rodzajów aktywności stosowanych przez zwykłych użytkowników. Jest to celowy zabieg, aby upodobnić działania aplikacji do działań człowieka, w celu zyskania aprobaty innych userów, czego skutkiem jest zwiększenie popularności profilu.

#### **3.6.1 Like By Tags**

W aplikacji zaimplementowaliśmy dwa rodzaje powyższej aktywności. Lajkowanie postów jest jedną z podstawowych możliwości mediów społecznościowych, dlatego jest tak ważna dla naszego profilu.

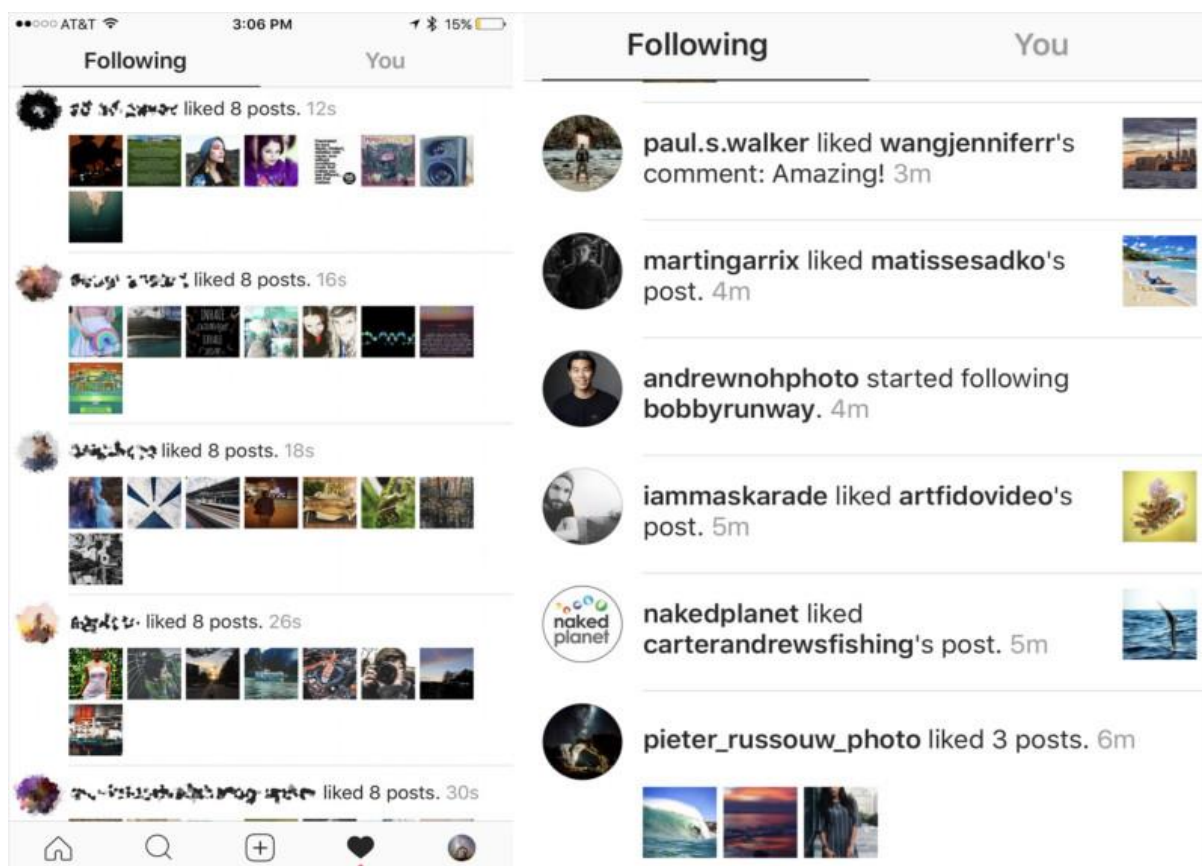
Opcja like by tags działa na zasadzie wyszukiwania różnego rodzaju zdjęć lub filmów w zależności od podanego przez użytkownika tagu, czyli oznaczenia tematycznego. Kolejnym krokiem jest znalezienie 3 zdjęć osoby postującej, której aktywność polubiliśmy,

co sprawia, że zostawiamy po sobie duży ślad działania na profilu usera skutkujący większą szansą na odwiedzenie naszego profilu. Jak już wiemy tagi są bardzo często używane przez użytkowników Instagrama, przez co łatwo zidentyfikować tematykę danego zdjęcia. To rozwiązanie sprawia, że wyszukiwanie potencjalnego zdjęcia do lajkowania jest banalnie proste.

```
def likePhotosByTags(session, tagList, probability):  
    if (tagList != '' and probability != ''):  
        session.set_user_interact(amount=3, randomize=True,  
percentage=probability, media='Photo')  
        session.like_by_tags(tagList, amount=10)
```

**Listing 8. Uproszczona wersja funkcjonalności**  
Źródło: Opracowanie własne

Tagowanie nie jest jedynym parametrem przekazywanym do działania powyższej opcji. Aby uzyskać pewien realizm dotyczący naszych działań korzystamy z prawdopodobieństwa polubienia - określamy procentową ilość szans polubienia posta. Jest to doskonały zabieg, aby nasza historia aktywności była zupełnie przypadkowa, co sprawia, że bot jest trudny do wykrycia i nasze działania są bliższe ludzkim. We wcześniejszych rozdziałach wspominaliśmy o wyborze funkcjonalności przez bota - na powyższym zdjęciu mamy prostą instrukcję warunkową zabezpieczającą przed pobieraniem pustych danych, co sprawia, że oprogramowanie nie uruchamia funkcjonalności, które nie zostały zdefiniowane.



**Rysunek 10. Porównanie działań bota (po lewej) i człowieka (po prawej)**  
Źródło: [10]

### 3.6.2 Following

Obserwacja użytkowników to kolejna z podstawowych możliwości Instagrama - zapewnia nam dostęp zarówno do profilu użytkownika jak i publikowanych treści, które po zaobserwowaniu będą pojawiać na naszej stronie głównej.

W IMAM zaimplementowaliśmy dwa typy obserwacji użytkowników - poprzez już wyżej omawiane tagowanie oraz lokalizacje. W przypadku tagów system jest bardzo podobny do wspomnianego wcześniej lajkowania. Jediną zasadniczą różnicą jest efekt działania, czyli zaobserwowanie profilu użytkownika.

Wyszukiwanie przez lokalizację działa na podobnej zasadzie jak powyższe lajkowanie przez tagi, jednak w tym wypadku obiektem wyszukiwania jest lokalizacja danego posta. W wielu przypadkach użytkownicy Instagrama z różnych powodów dodają oznaczenie do opublikowanych treści. Oznaczenia te mogą dotyczyć absolutnie wszystkiego, pod tym względem nie ma ograniczeń. Jedną z możliwości oznaczenia jest lokalizacja, czyli przypisanie posta do konkretnego miejsca niezależnie czy jest to zdjęcie czy film, czy innego

rodzaju aktywność - nie ma to żadnego znaczenia. Nie wszystkie treści posiadają tagi, dlatego tak ważne jest, aby nie ograniczać się wyłącznie do jednej możliwości.

```
class InstagramFunctions:
    def followByLocation(session, location):
        if (location != ''):
            session.follow_by_locations(location,
amount=100)exOptions.CultureInvariant,
            TimeSpan.FromMilliseconds(100)
        )
        .ToLowerInvariant();
```

**Listing 9. Kod źródłowy funkcji obserwacji użytkowników po lokalizacji**

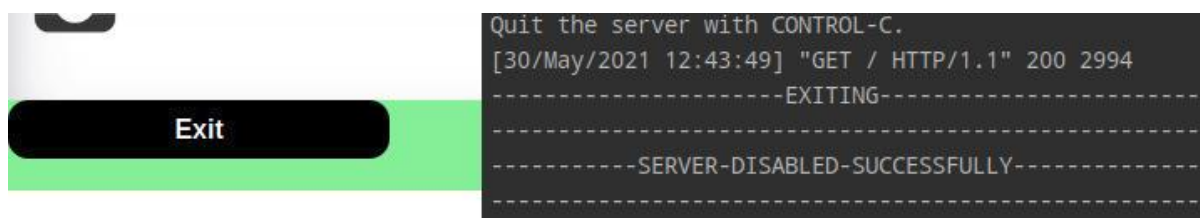
Źródło: [21]

Jak widać na powyższym zdjęciu mechanizm działania funkcji jest podobny do wcześniejszych, również podajemy listę - w tym wypadku lokalizacji - oraz dopuszczalną liczbę obserwacji. Te składowe stanowią podstawę do przeszukiwania serwisu Instagram.

Zdjęcie przedstawia tylko część logiki biznesowej zawartej w kodzie projektu.

### 3.7 Exit

Wszystko musi mieć swój początek i koniec, nie inaczej jest w przypadku naszej aplikacji. Jako dodatek dodaliśmy przycisk “Exit” odpowiedzialny za zakończenie działania programu. Jest to bardzo wygodne rozwiązanie, które pozwala nam zakończyć pracę programu w dowolnym momencie bez żadnych konsekwencji w działaniu.



**Rysunek 11. Proces wyłączania aplikacji**

Źródło: Opracowanie własne

Przycisk wykorzystuje bibliotekę systemową dostępną w pythonie - os. Zapewnia ona wszelkie potrzebne funkcjonalności wykonywane w systemie poprzez kod w języku python dlatego zakończenie procesów naszej aplikacji nie mogło się odbyć bez jej wykorzystania.

```
def kill_browser():
    try:
        if (os.name == 'posix'):
            browser = subprocess.Popen(['pgrep', 'firefox'],
            stdout=subprocess.PIPE)
            for pid in browser.stdout:
                os.kill(int(pid), signal.SIGTERM)
        if (os.name == 'nt'):
            os.popen("tskill chrome")
    except:
        print("An error occurred while exiting browser.")
```

**Listing 10. Metoda odpowiedzialna za zakończenie procesu przeglądarki**

Źródło: Opracowanie własne

Metoda obsługuje zarówno platformę Linuxa jak i Windowsa. Jak widać są to proste komendy implementowane do systemu z poziomu skryptu pythona kończące proces przeglądarki - w tym przypadku przeglądarki Mozilla Firefox przez użycie komendy `os.kill`.

### 3.8 Bazy Danych – SQL i NOSQL

Baza danych SQL (ang. Structured Query Language) jest oparta o strukturyzowaną relację między wierszami i kolumnami w tabeli. Każdy wiersz jest odrębnym obiektem a każda kolumna jego parametrem. Bazy SQL są stosowane ze względu na zgodność z ACID[38] compliance:




- read uncommitted – jedna transakcja może odczytywać wiersze, na których działają inne transakcje (najniższy poziom izolacji)
- read committed – transakcja może odczytywać tylko wiersze zapisane
- repeatable read – transakcja nie może czytać ani zapisywać na wierszach odczytywanych lub zapisywanych w innej transakcji
- serializable (szeregowalne) – wyniki współbieżnie realizowanych zapytań muszą być identyczne z wynikami tych samych zapytań realizowanych szeregowo (pełna izolacja).

NOSQL (ang. Not Only SQL) jest dużo elastyczniejszy od języka sql, pozwala na zdecydowanie zwinniejszą pracę z kodem ze względu na brak wcześniejszego dokładnego definiowania struktury dokumentów. Najlepiej nadaje się do przechowywania prostych danych, idealnie wpasowuje się w trzymanie plików typu json czy tekstowych. Główna myśl, za którą została stworzona było przede wszystkim nadanie jej elastyczności i skalowalności. Aby tego dokonać oparto ją na zasadach BASE [37]:

- Podstawowa dostępność (Basic Availability) - baza gwarantuje dostęp do danych nawet w przypadku kiedy może nie uzyskać wymaganych danych lub dane mogą być niespójne
- Stan miękkiej (Soft State) - stan bazy danych może się zmieniać z czasem.
- Ostateczna spójność (Eventual consistency) - baza danych ostatecznie stanie się spójna, a dane będą propagowane wszędzie w dowolnym momencie w przyszłości.

### 3.8.3 MongoDB

Jedną z najbardziej rozwijanych baz danych ostatnich lat, jednocześnie będącą jedyną takim nie relacyjnym silnikiem bazodanowym, znajdującym się w czołówce rozwiązań aplikacji biznesowych na całym świecie.

Rank			DBMS	Database Model	Score		
May 2021	Apr 2021	May 2020			May 2021	Apr 2021	May 2020
1.	1.	1.	Oracle 	Relational, Multi-model 	1269.94	-4.98	-75.50
2.	2.	2.	MySQL 	Relational, Multi-model 	1236.38	+15.69	-46.26
3.	3.	3.	Microsoft SQL Server 	Relational, Multi-model 	992.66	-15.30	-85.64
4.	4.	4.	PostgreSQL 	Relational, Multi-model 	559.25	+5.73	+44.45
5.	5.	5.	MongoDB 	Document, Multi-model 	481.01	+11.04	+42.02

**Rysunek 12. Ranking silników bazodanowych**

Źródło: [22]

MongoDB [4] posiada bardzo elastyczną strukturę przetwarza różnego rodzaju danych i w odróżnieniu do relacyjnych baz danych nie wymaga kreowania tabel ze stałymi obiektami, gdyż dane MongoDB są zapisywane w dokumentach w formacie JSON (JavaScript Object Notation) - lekki format wymiany informacji, łatwy zarówno do generacji jak i odczytu [24]. Dane mogą mieć różnie ustalone w zależności od rodzaju informacji typy.



```

{
  "id": 1,
  "authorName": "Patryick",
  "describe": "Cool set of comments for nature",
  "commentsSet": [
    {
      "mandatory_words": [
        "icecave",
        "ice_cave"
      ],
      "comments": [
        "Nice shot. Ice caves are amazing",
        "Cool. Aren't ice caves just amazing?"
      ]
    }
  ]
}

```

**Listing 11. Przykład formatu Json - przykładowy komentarz**

Źródło: Opracowanie własne

Komentarze są ustalane przez kluczowe słowa zawarte w opublikowanych postach, aby tematyka komentarza zgadzała się z tematyką posta. To nie wzbudzi żadnych podejrzeń wśród innych użytkowników, gdyż komentarze będą bezpośrednio nawiązywać do zdjęcia.

Ze względu na to, że nasza aplikacja nie wymaga tworzenia uporządkowanego schematu tabel – w odróżnieniu do relacyjnych baz danych - postawiliśmy na proste elastyczne rozwiązanie. Szczególnie ważnym aspektem jest fakt, że potrzebujemy bazy wyłącznie do przechowywania przykładowych komentarzy dla użytkownika, aby ten mógł w prosty sposób z nich korzystać, więc baza mongoDB była oczywistym wyborem, przez swoją prostotę zarówno w implementacji jak i obsłudze. To dobry krok ku zdobyciu nowego doświadczenia oraz odejściu od tradycyjnej metody implementacji relacyjnej bazy danych, które wśród aplikacji webowych stanowią nieodzowną większość<sup>2</sup>.

Dostęp do danych odbywa się przez wybór opcji z listy rozwijanej formularza na stronie głównej oprogramowania. To skutkuje pobraniem komunikatu w formacie Json z bazy danych do pola formularza. Dalsza część procesu została opisana w powyższych podrozdziałach.

---

<sup>2</sup>Jest to spowodowane organizacją danych w relacyjnym silniku bazodanowym. Każdy obiekt ma swoją referencję, przez co dane są schematycznie podobne i łatwe do przetworzenia (Każda tabela ma z góry określone stałe obiekty - kolumny, które łączą się z innymi tabelami poprzez unikalne klucze)

### 3.8.4 Sposób działania

```
class MongoClientService:
    def __init__(self, db_name, collection_name, host, port, username, password):
        try:
            client = MongoClient(host=host,
                                port=int(port),
                                username=username,
                                password=password
                                )
            self.db = client[db_name][collection_name]
        except:
            print("Server had problem while trying connecting with mongodb.")
            print("Oops!", sys.exc_info()[0], "occurred.")
```

**Listing 12. Serwis komunikacji z bazą MongoDB**

Źródło: Opracowanie własne

Klasa MongoClientService jest odpowiedzialne za tworzenie połączenia między naszym serwisem a bazą danych MongoDB. Podczas inicjalizacji klasy, podawane są takie parametry jak:

- db\_name - nazwa bazy danych
- collection\_name - nazwa używanej kolekcji
- host - lokalizacja pod jaką znajduje się baza danych
- port - Port wystawiony przez bazę danych
- username - nazwa użytkownika posiadającego uprawnienia do bazy danych
- password - hasło użytkownika

Baza zostaje utworzona poprzez wywołanie powyższej klasy.

```
def create(self, userName, description, commentsSet):
    self.db.insert_one({'author_name': userName, 'description': description,
                        'commentsSet': commentsSet})
```

**Listing 13. MongoDB Zapis danych**

Źródło: Opracowanie własne

Create odpowiada za stworzenie rekordu w bazie danych. Celowo w rekordzie nie zostało umieszczone pole identyfikacyjne, gdyż jest on samodzielnie generowany przez MongoDB. Rekord w bazie wygląda następująco:

- userName - jest to nazwa użytkownika który stworzył rekord
- description - jest opisem danych zawartych w komentarzach

- commentsSet - jest tablicą danych zawierającą dwie tablice opisane poniżej
- comments - przygotowany przez użytkownika zestaw komentarzy do powtarzalnego używania przez aplikację
- tags - zestaw tagów określających tematykę komentarzy

```
def readOneById(self, id):
    return self.db.find_one({'_id': id})

def readOneByAuthor(self, author):
    return self.db.find_one({'author_name': author})
```

#### Listing 14. MongoDB parametryzowany odczyt danych

Źródło: Opracowanie własne

W aplikacji potrzebowaliśmy trzech opcji odczytu, dwie z nich opierają się na odczycie za pomocą parametrów. W pierwszym przypadku wyszukujemy rekordu po jego numerze identyfikacyjnym. W drugim po nazwie autora danego zestawu komentarzy.

```
def readAll(self):
    return self.db.find()
```

#### Listing 15. MongoDB odczyt danych

Źródło: Opracowanie własne

Ostatnią metodą odczytu jest możliwość pobrania wszystkich rekordów. To pozwala w prosty sposób wypisać wszystkie informacje zawarte w naszej bazie danych.

Dodatkowo zaimplementowaliśmy metody do zmiany wartości zapisanych w rekordach. Każda z tych metod zmienia jeden parametr. Możemy podmienić komentarze, opisy i nazwę autora.

```
def updateComment(self, id, comment):
    self.db.updateOne({'id': id}, {'$set': {'comment': comment}})

def updateDesc(self, id, describe):
    self.db.updateOne({'id': id}, {'$set': {'describe': describe}})

def updateAuthor(self, id, userName):
    self.db.updateOne({'id': id}, {'$set': {'authorName': userName}})
```

#### Listing 16. MongoDB modyfikacja danych

Źródło: Opracowanie własne

### 3.8.5 Inicjalizacja bazy

Naszą bazę danych inicjujemy za pomocą dockera plikiem docker-compose.yml. Compose to narzędzie do definiowania i uruchamiania wielokontenerowych aplikacji Docker. Dzięki Compose [25] użyty jest jeden plik YAML do konfigurowania usług aplikacji, pozwala to na uruchomienie wielu kontenerów za pomocą prostego zestawu danych. Następnie za pomocą jednego polecenia (docker-compose up) docker tworzy i uruchamia wszystkie usługi ze swojej konfiguracji.

```
version: '3'
services:
  mongodb:
    image: mongo:4.0.8
    ports:
      - "27017:27017"
    container_name: mongodb
    restart: unless-stopped
    command: mongod --auth
    environment:
      MONGO_INITDB_ROOT_USERNAME: mongodbuserr
      MONGO_INITDB_ROOT_PASSWORD: mongoPassword
      MONGO_INITDB_DATABASE: comments
      MONGODB_DATA_DIR: /data/db
      MONGODB_LOG_DIR: /dev/null

  mongo-seed:
    build: ./mongo_seed
    ports:
      - "80:80"
    links:
      - mongodb
```

**Listing 17. Plik YAML konfigurujący Docker Compose**

Źródło: Opracowanie własne

Korzystanie z Compose to zasadniczo trzyetapowy proces:

1. Definiowanie środowiska aplikacji za pomocą dockerfile
2. Definiowanie usług, aby mogły być uruchamiane wraz docker-compose
3. Uruchom docker compose up, a polecenie docker compose uruchomi i uruchomi całą aplikację. Alternatywnie można uruchomić docker-compose up przy użyciu pliku binarnego docker-compose.

Poza samym mongo przygotowaliśmy także plik dockerfile, który zapewnił nam automatyczną inicjalizację bazy. Pliki dockerfile tworzą kontenery - jednostki

oprogramowania zawierające kod oraz wszystkie jego zależności, czego wynikiem jest niezawodne, szybkie i elastyczne działanie (kontenery mogą zostać użyte w wielu środowiskach programistycznych).

```
# set base image (host OS)
FROM python:3.8

# set the working directory in the container
WORKDIR /mongo_seed

# install dependencies
RUN pip install pymongo

# copy the content of the local src directory to the working directory
COPY . .

# command to run on container start
CMD [ "python", "database.py" ]
```

**Listing 18. Dockerfile startujący skrypt wykonujący plik python**

Źródło: Opracowanie własne

Jest pythonowy skrypt, który łączy się z bazą danych, tworzy kolekcję a następnie wpisuje w nią jeden podstawowy testowy rekord w formacie Json [8].

```
with open('mongoSeed.json') as data:
    data_dct = json.load(data)

result = mongo.insert_one(data_dct)
print('Inserted post id %s ' % result.inserted_id)
```

**Listing 19. Fragment kodu wpisujący zawartość pliku jsonowego do bazy danych**

Źródło: Opracowanie własne

## 4. Testy

Testy są nieodłączną częścią każdego projektu biznesowego, gdyż w jaki sposób jesteśmy w stanie sprawdzić czy nasza aplikacja spełnia oczekiwane wymagania? Za najbardziej podstawową formę testowania możemy uznać zwyczajne uruchomienie aplikacji, gdzie oczekujemy określonych danych wynikowych. Oczywiście środowisko testowe nie zamyka się wyłącznie na aplikacje webowe, jednak inne środowiska testowe nie dotyczą niniejszej pracy.[5]

### 4.1 Testy Automatyczne

Testy automatyczne jak sama nazwa wskazuje wykonują się automatycznie, zwykle to przygotowane fragmenty kodu zawierające jedną lub więcej funkcjonalności z aplikacji. Zaletą testów automatycznych jest fakt, iż te testy wykonują się same, bądź wykonuje je maszyna, w przeciwieństwie do testów manualnych gdzie potrzebny jest człowiek jako czynnik sprawdzający. Ten fakt pozwala pisać setki testów, które za każdym razem sprawdzają naszą aplikację, zapewniając nam stały komfort i bezpieczeństwo pracy.

### 4.2 Testy manualne

Drugim rodzajem są testy manualne. Testy automatyczne, mimo iż są ważne, nie zawsze są możliwe lub opłacalne do napisania. Systemy opierające się na widokach HTML<sup>3</sup> są dużo trudniejsze w testowaniu automatycznym, toteż często stosuje się testerów automatycznych. Każda większa dynamika jest często bardzo trudna do przetestowania. W naszym programie wielokrotnie wykonywaliśmy testy manualne na wszelkich płaszczyznach. Istotą problemu naszego systemu było wykorzystanie frameworka InstaPy, który w znacznym stopniu skomplikował możliwość wykonania testów automatycznych.

### 4.3 Po co nam testy?

Pisanie testów to nie tylko dobra praktyka, ale jest zwyczajnie opłacalne zarówno dla programisty jak i pracodawcy, gdyż pozwala zaoszczędzić ogrom czasu i pieniędzy. Pisząc testy mamy większy procent szans na wcześniejsze wykrycie błędu w aplikacji, szczególnie gdy testy sprawdzają poprawność działania dla pojedynczych modułów, co przy nieudanym teście automatycznie określa nam miejsce powstania buga. W przypadku, gdy nie

---

<sup>3</sup>Systemy te nie zwracają z funkcji czystych wartości a służą do zmiany stanu widoku. Widoki są często niemożliwe do testowania ze strony backendu

sprawdzamy bieżącego stanu działania naszej aplikacji, jesteśmy narażeni na długofalowe poszukiwanie błędów. O ile kompilator jest w stanie znaleźć niepoprawność w konstrukcjach, tak czasem nie ma wpływu na poprawność działania, gdyż kod źródłowy został skompilowany, a wynik pozostaje błędny [6]. To jest kwintesencja testów, gdyż chcemy dostarczyć jak najlepszy jakościowo produkt, a im większy code coverage, tym większa szansa, że produkt będzie działał zgodnie z naszymi oczekiwaniami [26]. Dobrze napisane testy zapewniają zaufanie do aplikacji i łatwość w znajdowaniu błędów, przez co pozwalają pracować niezależnie innym deweloperom [7].

#### **4.4 Metodyki pisania testów**

W świecie testów istnieją dwie metodyki pracy, jedną jest TDD (ang. Test Driven Development) - polega na pisaniu testów jeszcze przed rozpoczęciem implementacji części funkcjonalnej. Rozwiązanie to jest bardzo wygodne i wydajne. Dzięki takiemu podejściu twórca oprogramowania jest w stanie bardzo szybko odnaleźć błędy w kodzie oraz zna ukierunkowanie danej aplikacji biznesowej. To sprawia, że testy tworzą nam plan na implementację funkcjonalności w projekcie, więc automatycznie zaoszczędzimy ogrom czasu. Z kolei minusem tego rozwiązania jest bardzo duże wymaganie względem znajomości oraz świadomości naszej aplikacji. Sam test nie będzie mógł być ponownie zmieniony, a co za tym idzie nasza wartość końcowa zapisana w teście musi być konkretna i bezwzględna, gdyż to właśnie testy są naszym planem działania [1].

Drugą metodą pisania testów jest BDD (ang. Behavior Driven Development) [28], polegające na tworzeniu testów po zakończeniu implementacji funkcjonalności lub podczas implementacji. Jest to bardzo zwinne podejście, w TDD ze względu na stałe wyniki, często format zwracany przez funkcję musiał być dodatkowo poprawiany, co generowało niezbędny nakład pracy. To z kolei, że sprawiało lepsze rozwiązania były pomijane, aby dopasować się do testów. Ze względu na strukturę naszego projektu, zdecydowaliśmy się właśnie na tą metodykę. Pozwoliła ona dostosować testy do potrzeb, jakie generował kod oraz zmiennie adaptować wyniki, aby nie zaburzały płynności pracy i nie blokowały przepływu informacji między funkcjami [2].

#### **4.5 Testy Jednostkowe - unittest**

Do testów jednostkowych użyliśmy biblioteki unittest, która pozwoliła nam w wygodny oraz przystępny sposób pisać testy jednostkowe na wszystkie najważniejsze funkcjonalności, jakie były zawarte w naszej aplikacji [30].

```

def test_toJsonFormattedDictFunction(self):
    setting = {
        'author_name': "Patrick",
        'description': "Test case",
        'commentsSet': [{
            'mandatory_words': ["words", "are", "ducks"],
            'comments': ["My favorite place is Zalesie"]
        }]
    }
    result = toJsonFormattedDict(setting)
    expected = {'author_name': 'Patrick',
                'commentsSet': [{'comments': ['My favorite place is
Zalesie'],
                                'mandatory_words': ['words', 'are',
'ducks']}]},
                'description': 'Test case'}
    self.assertEqual(expected, result)

```

**Listing 20. Przykładowy test jednostkowy**

Źródło: Opracowanie własne

## 4.6 Testy Integracyjne - docker

Testy integracyjne pozwalają użytkownikowi na sprawdzenie zachowania aplikacji podczas użytkowania realnych narzędzi zewnętrznych. Testy integracyjne uzupełniają testy jednostkowe, pozwalając zobaczyć szerszy obraz aplikacji oraz zapewniając pokrycie możliwych błędów, jakie by wynikały ze względu na przepływ danych między punktem A oraz B.

Do testów integracyjnych użyliśmy dockera, który pozwolił nam na realną symulację użytkowania bazy danych mongoDb. W tym celu wykorzystaliśmy wcześniej przygotowany skrypt docker-compose, który stworzył nam lokalny kontener z bazą danych. W testach sprawdziliśmy wszystkie funkcjonalności oparte o bazę, które były wymagane przez IMAM. Dzięki zastosowaniu dockera testy zyskały na bezpieczeństwie, docker pozwala na konteneryzację, a co za tym idzie izolację serwisów.



```

def setUp(self):
    self.userName = "mongodbuserr"
    self.dbName = "testDB"
    self.hostName = "localhost"
    self.port = 27017
    self.collectionName = "comments"
    self.password = "mongoPassword"
    self.mongo = MongoClientService(db_name=self.dbName,
collection_name=self.collectionName,
                                host=self.hostName,
                                port=self.port,
                                username=self.userName,
                                password=self.password
                                )

    self.raw_client = MongoClient(
        host=self.hostName,
        port=self.port,
        username=self.userName,
        password=self.password
    )
    self.raw_client["testDB"]["comments"].drop()
    with open('mongoSeed.json') as data:
        data_dct = json.load(data)

    result = self.raw_client["testDB"]["comments"].insert_one(data_dct)
    print('Inserted post id %s ' % result.inserted_id)

```

**Listing 21. Przykładowy test integracyjny**

Źródło: Opracowanie własne

Aby zainicjować połączenie z bazą danych potrzebowaliśmy odpowiednich ustawień bazy danych, w tym celu powstała oddzielna metoda setUp. Metoda niweluje efekt redundancji kodu źródłowego - nie musimy wielokrotnie podawać tych samych parametrów w funkcjach testowych, co sprawia że kod jest bardziej czytelny i przejrzysty.

## 4.7 Scenariusze testowe

Ze względu na mechanikę budowy naszego bota, funkcjonalności odpowiedzialne za wykonywanie akcji na Instagramie byłyby bardzo trudne do przetestowania automatycznego. Wybraliśmy przygotowanie zestawu scenariuszy operujących na wyekstrahowanych funkcjonalnościach [8].

## **Zakończenie**

Celem niniejszej pracy było przedstawienie możliwości oraz zagrożeń, wynikających z automatyzacji ludzkich zachowań w mediach społecznościowych, na podstawie serwisu Instagram. Do jego realizacji stworzyliśmy autorskiego Social Media Bota, naśladującego działania użytkowników tego medium. Funkcjonalność programu obejmuje obserwowanie, komentowanie i lajkowanie postów innych użytkowników, co pozwala między innymi na promowanie wybranych treści, czy zwiększenie popularności przez pozyskiwanie obserwatorów i polubień. IMAM może znaleźć zastosowanie np. w branży reklamowej lub zostać wykorzystane przez influencerów.

Automatyzacja ludzkich działań w internecie ma zarówno pozytywne, jak i negatywne konsekwencje, w zależności od sposobu wykorzystania i intencji użycia. Pozytywną stroną jest niewątpliwie oszczędność czasu i pracy człowieka, poprzez zastąpienie powtarzalnych i schematycznych czynności technologią, czyli przez użycie automatycznego oprogramowania - botów. Dodatkowo przedstawiliśmy porównanie pracy bota i człowieka, w którym większą efektywność wykazało oprogramowanie, co jednoznacznie udowadnia ich użyteczność. Natomiast zagrożenia wynikające z używania botów to promowanie szkodliwych treści, generowanie spamu, rozprzestrzenianie dezinformacji na skalę lokalną, krajową a nawet światową. Boty mogą również posłużyć do działań takich jak hate, cyberprzemoc, stalking, niekiedy są używane do kradzieży danych użytkownika.

Stworzony projekt w głównej mierze operuje na narzędziu instapy, które odpowiada za większość działań. Mając na uwadze zmienną architekturę serwisu Instagram, należy zdać sobie sprawę z wątpliwej niezawodności w działaniu oprogramowania. Takie sytuacje mogą się szczególnie pojawiać świeżo po aktualizacji serwisu, czego efektem będzie brak określonych funkcji. Powodem tego jest framework selenium, który operuje na ścieżkach dostępu po atrybutach html danej strony, w tym przypadku serwisu Instagram. Aktualizacje medium zazwyczaj zmieniają nieco jego wygląd, więc zmienia się też kod źródłowy, na którym operuje selenium. Pisząc tą aplikację napotkaliśmy wiele błędów z tym związanych, które do tej pory pozostały nierozwiązane, co zmusiło nas do rezygnacji z niektórych funkcjonalności.

Temat okazał się bardzo złożony i dynamiczny w nawiązaniu do naszej aplikacji, gdzie rozwiązanie było zależne od serwisu, na którym operuje. Projekt można rozwijać dodając kolejne funkcjonalności, jednakże należy mieć świadomość bieżących zmian zachodzących w serwisie Instagram.

## Podział Pracy

Praca licencjacka została zrealizowana w zespole składającym się z 2 osób. W jego skład wchodził: Dariusz Gawęda oraz Patryk Kirszenstein.

Podział pracy przedstawia się następująco:

Dariusz Gawęda 50% pracy:

### 1. Implementacja serwera Django:

- Implementacja funkcjonalności dodawania zestawów komentarzy do bazy danych
- Stworzenie szablonów przyjmujących i wysyłających dane z serwera
- Implementacja obsługi zakończenia procesu przeglądarki
- Stworzenie widoków obsługujących żądania użytkownika
- Stworzenie możliwości wyświetlania wyników z bazy danych na stronie przy pomocy MongoDB

### 2. Implementacja warstwy frontend:

- Stworzenie strony startowej oraz podstrony przy pomocy HTML5 / CSS / jQuery / JavaScript
- Stworzenie walidatora formularza przy pomocy jQuery / JavaScript
- Stworzenie animacji przy pomocy jQuery

Patryk Kirszenstein 50% pracy:

### 1. Implementacja InstaPy

- Implementacja funkcjonalności związanych z botem

### 2. Implementacja docker

- a. Stworzenie skryptu generującego początkowe dane
- b. Stworzenie docker-compose

### 3. Implementacja Testów Integracyjnych

### 4. Implementacja Testów Jednostkowych

- sprawdzenie poprawności działania części funkcjonalnej oprogramowania

### 5. Implementacja Bazy danych

- Implementacja MongoDB
- Stworzenie połączenia z bazą danych

## Bibliografia

- [1] K. Beck, *TDD. Sztuka tworzenia dobrego kodu* str. 7-12
- [2] J. F. Smart, *BDD w działaniu. Sterowanie zachowaniem w rozwoju aplikacji*, Helion str. 12-13
- [3] P. Sams, *Selenium Essentials*, Pack Publishing, 2015 str. 53 - 60
- [4] S. Bradshaw, E. Brazil, K. Chodorow, *Przewodnik po MongoDB. Wydajna i skalowalna baza danych. Wydanie III*, Helion, 2020, str. 25-26
- [5] R. C. Martin, *Czysty Kod. Podręcznik dobrego programisty*, Helion, 2009, str 320-326
- [6] R. C. Martin, *Mistrz czystego kodu. Kodeks postępowania profesjonalnych programistów*, Helion, 2013, str. 115-136
- [7] D. M. Beazley, Brian K. Jones, *Python. Receptury. Wydanie III*, O'reilly, 2014, str. 175-217
- [8] Al Sweigart, *Automate the Boring Stuff with Python, 2nd Edition: Practical*, No Starch Press 2015, str. 76-102
- [9] <https://encyklopedia.pwn.pl/haslo/rewolucja-przemyslowa;3967502.html> [dostęp z dnia 12.05.2021]
- [10] <https://petapixel.com/2017/04/06/spent-two-years-botting-instagram-heres-learned/> [dostęp z dnia 13.05.2021]
- [11] <https://pl.wikipedia.org/wiki/Farming> [dostęp z dnia 15.05.2021]
- [12] [https://1raindrop.typepad.com/1\\_raindrop/chinese\\_farmers/](https://1raindrop.typepad.com/1_raindrop/chinese_farmers/) [dostęp z dnia 15.05.2021]
- [13] <https://www.imperva.com/blog/bot-traffic-report-2016/> [dostęp z dnia 16.05.2021]
- [14] <https://www.blacksoft.pl/boty-internetowe-a-ruch-w-sieci/> [dostęp z dnia 16.05.2021]
- [15] <https://info.mention.com/instagram-engagement-report-2020> [dostęp z dnia 17.05.2021]
- [16] <https://backlinko.com/instagram-users> [dostęp z dnia 18.05.2021]
- [17] <https://socialmedia.biz.pl/sposoby-zdobywania-popularnosci-na-instagramie/> [dostęp z dnia 21.05.2021]
- [18] <https://www.instagram.com/explore/tags/like4like/> [dostęp z dnia 23.05.2021]
- [19] <https://szybkielajki.pl/jak-zdobyc-lajki-na-instagramie/> [dostęp z dnia 26.05.2021]
- [20] <https://app.diagrams.net/> [dostęp z dnia 26.05.2021]
- [21] <https://instapy.org/> [dostęp z dnia 28.05.2021]
- [22] <https://graphql.org/> [dostęp z dnia 01.06.2021]
- [23] <https://db-engines.com/en/ranking> [dostęp z dnia 02.06.2021]
- [24] <https://www.json.org/json-en.html> [dostęp z dnia 02.06.2021]

- [25] <https://docs.docker.com/compose/> [dostęp z dnia 03.06.2021]
- [26] <https://www.indiumsoftware.com/blog/why-software-testing/> [dostęp z dnia 06.06.2021]
- [27] <https://sjp.pwn.pl/sjp/media-spolecznosciowe;5579207.html> [dostęp z dnia 23.04.2021]
- [28] <https://www.agilealliance.org/glossary/bdd/#q=~> [dostęp z dnia 07.06.2021]
- [29] <https://swpanel.pl/baza-wynikow/202867/media-spolecznosciowe> [dostęp z dnia 25.04.2021]
- [30] <https://docs.python.org/3/library/unittest.html>. [dostęp z dnia 03.05.2021]
- [31] <https://www.imperva.com/learn/application-security/what-are-bots/> [dostęp z dnia 03.05.2021]
- [32] <https://harbingers.io/definicje/crawler> [dostęp z dnia 05.05.2021]
- [33] <https://116111.pl/czym-jest-hejt/> [dostęp z dnia 07.05.2021]
- [34] <https://encyklopedia.pwn.pl/haslo/propaganda;3962718.html> [dostęp z dnia 07.05.2021]
- [35] <https://trafficwatchdog.pl/pl/articles/32/czym-sa-dobre-i-zle-boty> [dostęp z dnia 07.05.2021]
- [36] <https://www.youtube.com/watch?v=xp0O2vi8DX4> - [dostęp z dnia 07.05.2021]
- [37] <https://www.freecodecamp.org/news/nosql-databases-5f6639ed9574/> [dostęp z dnia 05.06.2021]
- [38] <https://database.guide/what-is-acid-in-databases/> [dostęp z dnia 10.06.2021]
- [39] <https://backlinko.com/instagram-users#instagram-stats-top-picks> [dostęp z dnia 13.05.2021]

## Spis listingów

Listing 1. Przykład metody w jQuery z animacją przycisków .....	20
Listing 2. Kierunek akcji formularza .....	21
Listing 3. Ścieżki oraz przypisane do nich widoki.....	22
Listing 4. Przykład przechwytywania danych z formularza .....	22
Listing 5. Przykład pola w szablonie podpisanego kluczem słownikowym .....	23
Listing 6. Przykład nazwanego pola .....	23
Listing 7. Parametry ustawień sesji InstaPy aby działała w tle.....	24
Listing 8. Uproszczona wersja funkcjonalności.....	27
Listing 9. Kod źródłowy funkcji obserwacji użytkowników po lokalizacji .....	29
Listing 10. Metoda odpowiedzialna za zakończenie procesu przeglądarki .....	30
Listing 11. Przykład formatu Json - przykładowy komentarz.....	32
Listing 12. Serwis komunikacji z bazą mongodb .....	33
Listing 13. Zapis rekordu do bazy mongodb.....	33
Listing 14. Mongodb parametryzowany odczyt danych .....	34
Listing 15. Mongodb odczyt danych.....	34
Listing 16. Mongodb modyfikacja danych .....	34
Listing 17. Plik YAML konfigurujący Docker Compose .....	35
Listing 18. Dockerfile startujący skrypt wykonujący plik python .....	36
Listing 19. Fragment kodu wpisujący zawartość pliku jsonowego do bazy danych .....	36
Listing 20. Przykładowy test jednostkowy .....	39
Listing 21. Przykładowy test integracyjny .....	40

## Spis rysunków

Rysunek 1. Podział użytkowników, ze względu na liczbę obserwujących .....	12
Rysunek 2. Liczba postów oznaczonych tagiem like4like .....	13
Rysunek 3. Diagram przypadków użycia - strona pierwsza.....	15
Rysunek 4. Widok interfejsu użytkownika zaraz po uruchomieniu aplikacji .....	16
Rysunek 5. Interfejs użytkownika - część druga .....	17
Rysunek 6. Interfejs strony odpowiedzialnej za dodawanie zestawów komentarzy .....	18
Rysunek 7. Przykład działania walidacji.....	21
Rysunek 8. Przykładowy rekord bazy danych.....	25
Rysunek 9. Przykładowe komentarze do dodania .....	25
Rysunek 10. Porównanie działań bota (po lewej) i człowieka (po prawej) .....	28
Rysunek 11. Proces wyłączania aplikacji .....	29
Rysunek 12. Ranking silników bazodanowych.....	31

## **Spis tabel**

Tabela 1. Wynik ankiety sprawdzającej aktywność użytkowników w mediach .....	4
Tabela 2. Średnia czasu spędzonego na Instagramie przez użytkowników .....	8
Tabela 3. Porównanie statystyk .....	9
Tabela 4. Zestawienie ruchu w sieci.....	11