Uniwersytet Mikołaja Kopernika w Toruniu

Wydział Matematyki i Informatyki

Krzysztof Skibicki

nr albumu: 273235

Informatyka

Praca inżynierska

TwojeReferendum.pl - Serwis badający opinię publiczną

Opiekun pracy dyplomowej

dr Aleksandra Boniewicz

Toruń 2021

# 1. Wstęp i analiza problemu

Niniejsza praca prezentuje projekt pt. "TwojeReferendum.pl - Serwis badający opinię publiczną". Jest to serwis do zbierania danych od opinii publicznej na temat wprowadzonych i opublikowanych ustaw na stronie: https://isap.sejm.gov.pl. Celem projektu jest umożliwienie przeprowadzania referendum nad każdą ustawą, która została wprowadzona w życie bez konieczności wychodzenia z domu i wydawania milionów złotych na organizację głosowania. W kolejnych rozdziałach zaprezentowane zostaną narzędzia, ich funkcjonalności oraz sposób ich użycia w projekcie.

Motywacją do stworzenia tego projektu była sytuacja, która miała miejsce w Polsce w roku 2020. Mowa tu o wyborach korespondencyjnych, które miały się odbyć ze względu na pandemię Covid-19, a nie doszły do skutku. Biorąc pod uwagę koszty organizacji takiego przedsięwzięcia, ilość osób zaangażowanych w przygotowania oraz czas potrzebny do realizacji referendum, mój projekt redukuje potrzebne zasoby do minimum. Co więcej, pozwala on na badanie opinii publicznej na bieżąco. Dzięki mojemu rozwiązaniu ludzie mogliby wypowiedzieć się na tematy bieżącego prawa zaraz po jego wprowadzeniu.

Drugim czynnikiem, który podsunął mi ten pomysł, było korzystanie z profilu zaufanego. Jest to profil, który może być przypisany do dokładnie jednego obywatela (każdy Polak posiada swój unikatowy profil). Dzięki temu każdy mógłby za pośrednictwem Internetu potwierdzić swoją tożsamość i zagłosować w referendum online. Niestety, po bliższym zapoznaniu się z tematem okazało się, że do weryfikacji za jego pośrednictwem mają dostęp jedynie strony rządowe oraz urzędy. Zmusiło mnie to do zastosowania innych, prawie tak samo wydajnych środków bezpieczeństwa, które opiszę w dalszej części pracy.

Biorąc pod uwagę przedłużającą się pandemię, moje rozwiązanie mogłoby posłużyć równie skutecznie do przeprowadzenia nie tylko referendum, ale również wyborów jako alternatywna opcja głosowania dla osób np. przebywających na kwarantannie lub będących za granicą.

Podsumowując, z punktu widzenia ekonomicznego jest to wielka oszczędność środków które można bezpiecznie przeznaczyć na równie istotne cele. Ze strony organizacyjnej, projekt umożliwiłby zagłosowanie i wyrażenie swojej opinii każdemu Polakowi bez względu na to, gdzie mieszka, w Polsce czy za granicą. Ponadto, takie wybory byłyby dużo bardziej wiarygodne, gdyż pieczę nad nimi sprawować dedykowany zespół osób. Kolejnym atutem byłby brak konieczności ręcznego przeliczania głosów oraz powoływania komisji wyborczych i drukowania kart do głosowania. Środowisko również mogłoby na tym skorzystać. Wyniki byłyby natychmiastowe. Z punktu widzenia ochrony zdrowia, jeśli każdy mógłby zagłosować we własnym domu to ryzyko zarażenia się niebezpiecznym wirusem spada do zera.

Uważam, że powyższe argumenty są wystarczające, aby taki projekt miał rację bytu i aby warto było podjąć się jego stworzenia. W następnych rozdziałach zaprezentuję sposób działania programu oraz z jakich technologii skorzystałem podczas jego implementacji.

# 2. Istniejące rozwiązania

W poszukiwaniu podobnych istniejących już rozwiązań nie natknąłem się na projekt o zbliżonej tematyce. Udało mi się natomiast znaleźć wiele rozwiązań, które są analogiczne do składowych stworzonego przeze mnie serwisu i pojedynczo występują w poszczególnych, istniejących już, stronach internetowych.

Pierwszym z nich zebranie informacji o ustawach, które zostały wprowadzone w ostatnim czasie oraz jaka jest ich treść. Dane te można znaleźć w kilku miejscach. Ja posiłkowałem się wyżej wspomnianą już stroną: https://isap.sejm.gov.pl. Z tego serwisu pochodzą wszystkie dane dotyczące ustaw jakie widnieją w moim projekcie.

Podczas pisania modelu głosowania oraz sposobu wyświetlania wyników na stronie inspirowałem się narzędziem google.com do tworzenia własnych formularzy. Za jego pośrednictwem możemy stworzyć dowolną ankietę i udostępnić ją wszystkim za pomocą odnośnika albo nadać dostęp do niej konkretnym adresom mailowym. Mimo, iż w moim rozwiązaniu nie znalazła się możliwość dostosowywania i udostępniania formularzy (nie ma takiej konieczności) to był to wzór jak serwis powinien się prezentować dla użytkownika końcowego. Rozwiązanie to mogłoby posłużyć za sposób przeprowadzenia referendum, ale nie rozstrzyga ono problemu weryfikacji głosującego i nie blokuje ono możliwości wielokrotnego wyrażania opinii przez jedną osobę.

Kolejnym sposobem na zbadanie opinii publicznej jest analiza forów tematycznych i treści jakie się na nich znajdują. Jest to niestety sposób niemiarodajny. Wywnioskowanie z nich jednoznacznej opinii jest bardzo trudne, a czasem niemożliwe. Takie serwisy przyciągają zazwyczaj osoby o skrajnych poglądach, które ścierają się ze sobą w kolejnych postach. Ponadto, dostęp do takiego konta na forum może mieć każdy, więc bardzo łatwo można by manipulować informacjami. Jedna osoba jest w stanie założyć wiele kont i w ten sposób zaburzać kierunek opinii publicznej. Możemy tam również spotkać tak zwanych „trolli internetowych”. Są to osoby, które celowo prowokują innych poprzez kontrowersyjne wypowiedzi, co wprowadza chaos. Fora internetowe mają zasięg globalny, co sprawia, że nie mamy kontroli nad tym jakiej narodowości ludzie się do nich logują, a co za tym idzie nie jesteśmy w stanie stwierdzić, czy opinia pochodzi od osób z polskim obywatelstwem. Postanowiłem ograniczyć sposób wypowiedzi do twierdzenia: „za”, „przeciw”, „wstrzymał się”. W ten sposób można uzyskać bardziej wiarygodny obraz opinii publicznej.

Zakładanie konta w portalu internetowym prawie zawsze wiąże się z obowiązkiem autentykacji i potwierdzenia swojej tożsamości. Najczęściej wykorzystuje się do tego adres mail lub numer telefonu. W Polsce od kilku lat rozwijany jest portal ePUAP, z pomocą którego możemy potwierdzić swoją tożsamość. Jeśli przykładowo chcemy złożyć zawiadomienie o sprzedaży auta, wystarczy wypełnić wniosek na stronie urzędu i potwierdzić swoją tożsamość za pomocą profilu zaufanego. Jest to jedna z bezpieczniejszych metod weryfikacji obywatela, bo opiera się o numer PESEL. Niestety rozwiązanie to używane jest jedynie przez jednostki państwowe. Biorąc pod uwagę, że mój projekt jest bezpośrednio związany ze zbieraniem danych, które skutecznie mogą być wykorzystywane przez rząd, wpisywałby się idealnie do kryteriów korzystania z profilu zaufanego w celu autentykacji.

# 3.Użyte narzędzia

Główną koncepcją było stworzenie serwisu internetowego, do którego będzie mógł zalogować się każdy obywatel RP posiadający PESEL. Strona pozwoli wyrazić swoją opinię na temat wprowadzonych przez sejm ustaw. Podczas projektowania strony pracowałem nad tym, aby było to rozwiązanie proste w obsłudze i intuicyjne.

Najistotniejszym elementem pobierającym dane pochodzące z strony https://isap.sejm.gov.pl jest skrypt napisany w języku python 3.6, korzystający z biblioteki Beautiful Soup. Aby podkreślić, dlaczego warto było z niej skorzystać, przytaczam kilka funkcjonalności, które zapewnia:

* Biblioteka udostępnia kilka prostych metod i idiomów do nawigacji, wyszukiwania i modyfikowania parsowanych danych z pliku HTML i XML. Zapewnia zestaw narzędzi do analizy dokumentu i wyodrębnienia tego, czego potrzebujesz. Podczas korzystania z tej biblioteki do napisania aplikacji nie potrzeba dużo kodu.
* Beautiful Soup automatycznie konwertuje dokumenty przychodzące do standardu Unicode, a dokumenty wychodzące do UTF-8. Nie trzeba myśleć o kodowaniu znaków, chyba że dokument tego nie określa, a Beautiful Soup nie może sam tego wykryć.
* Beautiful Soup jest jednym z najbardziej popularnych parserów dostępnych dla języka Python. Łatwo jest znaleźć rozwiązania częstych problemów i przykłady użycia.

Cenne dane, które kiedyś były zamknięte w źle zaprojektowanych witrynach internetowych, są dzięki tej bibliotece w zasięgu ręki. Projekty, których napisanie pochłonęłyby całe dni, zajmują co najwyżej kilka godzin, co jest niewątpliwym atutem. Używając tego narzędzia, jestem w stanie prostym skryptem wygenerować plik csv, który zawiera wszystkie potrzebne rekordy do bazy danych.

Aby informacje były aktualne i nowe ustawy pojawiały się na bieżąco, skrypt pobierający dane za pomocą Beautiful Soup został zautomatyzowany. Aby wyjaśnić na czym to polega, przedstawię najpierw pozostałe elementy projektu, a następnie wrócę do aktualizacji danych w bazie.

Do stworzenia wizualnej części projektu, czyli tak zwanego „front endu”, niezbędna była znajomość języków powszechnie wykorzystywanych przez strony internetowe. Ilość narzędzi, języków oraz sposobów ich zastosowania jest spora, dlatego postaram się wspomnieć tylko o wybranych. Najbardziej podstawowym elementem jest HTML (Hyper Rext Markup Language). Definiuje on znaczenie i strukturę treści internetowych. Inne technologie bazują na HTML, oferując nakładki do opisania stron internetowych i wyglądu lub sposobu prezentacji. Taką technologią jest CSS. Aby zwiększyć interaktywność strony, użyłem również się języki skryptowe takie jak JavaScript i PHP.

HTML używa „znaczników”, które są potem odczytywane i interpretowane przez przeglądarki internetowe. Znaczniki HTML zawierają specjalne „elementy”, takie jak:

<head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, <div>, <span>, <img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video>, <ul>, <ol>, <li>

i wiele innych. Element HTML jest oddzielony od innego tekstu w dokumencie za pomocą „tagów”, które składają się z nazwy elementu otoczonej znakami „<” i „>”. W nazwie elementu wewnątrz tagu nie jest rozróżniana wielkość liter. Oznacza to, że można go pisać wielkimi, małymi literami lub ich mieszanką.

Wspomniany wcześniej CCS (ang. Cascading Style Sheets) jest językiem używanym do opisywania warstwy prezentacji za pośrednictwem tak zwanych stylów. Język ten współpracuje z dokumentami napisanymi w HTML lub XML, wyłączając ich pochodne takie jak: SVG czy XHTML. Głównym zadaniem CCS jest przedstawienie jak dany element na stronie powinien się prezentować. Zaliczamy go do podstawowych języków otwartej sieci. Pracuje w standardzie W3C oraz jego składnia i funkcjonalności są stale rozwijane.

Aby umieścić interaktywne element na stronie, często używa się JavaScript (w skrócie JS). Jest to skryptowy język programowani, w którym funkcje grają kluczową rolę. Można przechowywać je w zmiennych jako referencje i przesyłać jak każdy inny obiekt. JS, mimo iż znany jest jako skryptowy język wspierający strony internetowe to znajduje zastosowanie w wielu środowiskach, nie tylko w przeglądarce internetowej. Przykładem jest np. Apache CouchDB, Adobe Acrobat lub Node.js.

W moim projekcie zastosowanie znalazł HTML, CSS oraz pojedyncze skrypty napisane w JS.

Do stworzenia wizualnej części strony powyższe narzędzia wystarczą w zupełności, lecz aby w bezpieczny oraz uporządkowany sposób połączyć je z częścią serwerową, potrzebny jest dodatkowy framework. W zależności od języka w jakim chcemy napisać część serwerową, można znaleźć różne technologie upraszczające proces tworzenia serwisu webowego. Zdecydowałem się na skorzystanie z języka programowania Pythona i frameworku Django. Głównym założeniem tej biblioteki jest, aby częste web-deweloperskie zadania były wykonywane w sposób szybki i prosty. Mój projekt używa Django w wersji 3.2.

Jednym z atutów tego frameworku jest wbudowany mapper obiektowo-relacyjny. Za jego pośrednictwem jest możliwe stworzenie bazy danych z poziomu kodu. Wbudowane klasy w Django umożliwiają pokaźną liczbę kombinacji rodzajów tabel, pól i ich przedstawienia. Po zadeklarowaniu modeli tabel wystarczy skorzystać z dwóch poleceń:

$ python manage.py makemigrations

$ python manage.py migrate

Pierwsza komenda - “makemigrations“ sprawdza czy zadeklarowane tabele istnieją w bazie i zgadzają się z modelami. Druga część - “migrate“ uruchamia migrację i tworzy nowe tabele. W ten sposób Django w łatwy i przystępny sposób pozwala na operacje bazodanowe z poziomu serwera.

Kolejnym z atutów tego narzędzia jest możliwość całkowitej personalizacji adresów URL bez zbędnych wstawek typu .asp lub. php. Do zaprojektowania URL-ów potrzebny jest dodatkowy moduł Pythona – URLconf. Jest to swego rodzaju spis treści dla aplikacji, który opisuje mapowanie pomiędzy wzorcami URL a funkcjami callback napisanymi w Pythonie. Moduł ten służy także do rozdzielenia adresów URL od kodu zapisanego w Pythonie. Wyrażania opisujące ścieżki korzystają z tagów, aby pobrać wartości zawarte w URL. Po wysłaniu przez klienta żądania strony z konkretnym adresem URL, Django przeszukuje wszystkie dostępne ścieżki i gdy znajdzie pierwszą pasującą do wzorca wywołuje przypisany mu widok. Jeśli jest to funkcja, zostanie wczytana odpowiednia strona, w przeciwnym przypadku zwracana jest odpowiedź z kodem błędu “404”. Cały proces jest bardzo szybki, ponieważ ścieżki URL są reprezentowane przez wyrażenia regularne. Wszystkie widoki opatrzone są parametrem „request”, który mieści w sobie metadane żądania oraz wartości zebrane ze wzorca.

Django posiada również komponent zwany „widokiem”. Ma on za zadanie zwrócenie obiektu HttpResponse, w której mieści się zawartość oczekiwanej strony lub wyświetlenie kodu błędu odpowiedzi HTTP. Dodatkowo widok pobiera obiekty podane w parametrach, ładuje szablony i wyświetla je z załadowanymi danymi. Dane te można wyświetlać lub wykonywać na nich operacje po stronie wizualnej

Przedstawione elementy to tylko ułamek możliwości jaki oferuje Django. Możliwości tego frameworku w zupełności pokrywają potrzeby mojego projektu, dlatego zdecydowałem się na użycie go.

Mimo tego, że Django udostępnia możliwość skorzystania z wbudowanego modelu bazy danych, zdecydowałem się na skorzystanie z PostgreSQL do przechowywania informacji oraz zarządzania bazą. Jest to obiektowo-relacyjny system do administrowania bazami danych. PostgreSQL umożliwia wiele przydatnych operacji, takich jak: stworzenie nowego typu danych, funkcji, operatora, funkcji agregującej, metod indeksowania, a nawet użycie języków proceduralnych. Warto podkreślić, że rozwiązanie jest w pełni darmowe bez względu na sposób użycia. Ponadto oferuje bogatą dokumentację, wiele materiałów i kursów dostępnych za darmo, a także ogromną społeczność, która służy pomocą za pośrednictwem forów internetowych. Do zarządzania bazą PostgreSQL wystarczy terminal, ale współpracuje on również ze środowiskiem pgAdmin, który za pomocą przeglądarki znacznie ułatwia zarządzanie bazą danych.

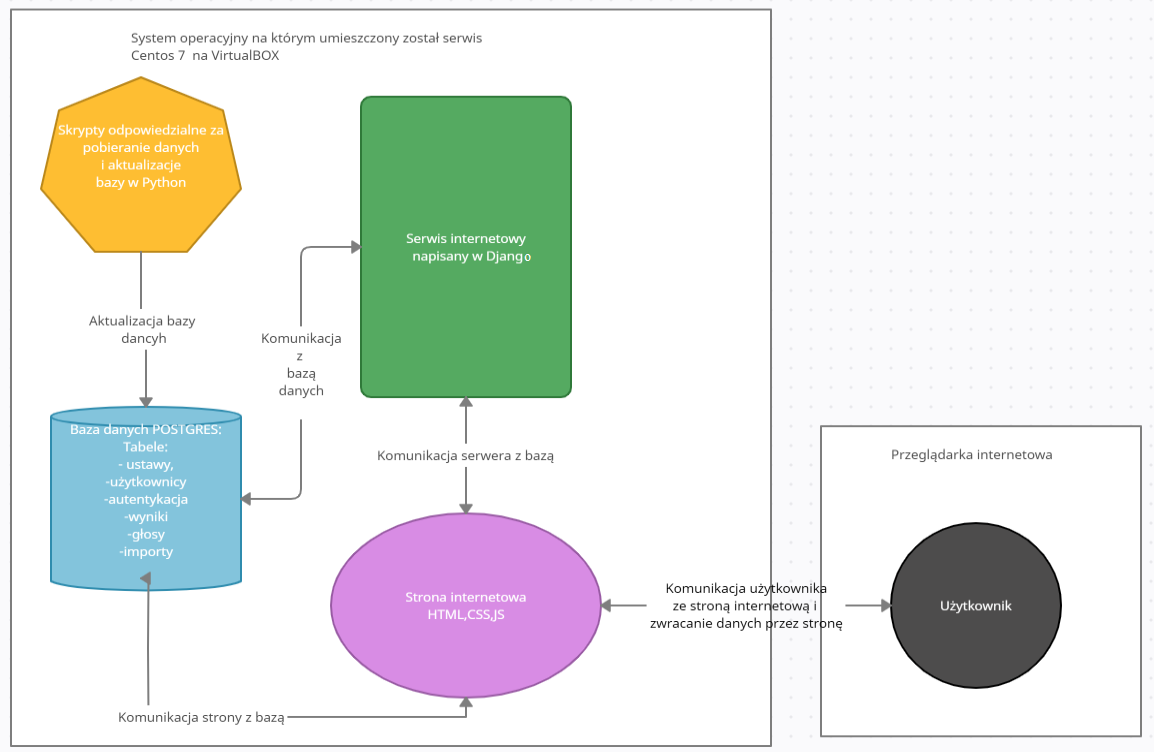
Wszystkie te komponenty podczas budowy rozwiązania umieszczone zostały na systemie operacyjnym Centos 7. Jest to system oparty o kod źródłowy Red Hat Enterprise Linux. Wykorzystanie systemu Linux jest ważne, ponieważ do automatycznej aktualizacji bazy danych o nowe ustawy został wykorzystany program cron. Jest to specjalny agent w systemie Linux przeglądający co minutę tabele z listami programów i zadaniami, które mają zostać uruchomione o podanej godzinie. Korzystając z tej funkcjonalności nie musiałem martwić się o ręczne uruchamianie skryptu pobierającego dane. Cały proces automatycznie dodaje informacje do bazy danych.

Wszystkie wyżej wymienione narzędzia w przyjazny sposób współpracują z językiem Python, co było istotnym kryterium podczas dobierania ich.

Jak wspomniałem wcześniej istotne jest, aby każdy obywatel miał możliwość zagłosowania tylko raz. Do osiągnięcia tego celu najlepsza byłaby weryfikacja oparta na profilu zaufanym. Niestety w komercyjnych projektach rozwiązanie to nie jest dostępne. Gdyby projekt uzyskał akredytację rządu, zamiana sposobu potwierdzania tożsamości byłaby ostatnim elementem znacząco podnoszącym wartość użytkową projektu. Ze względu na brak takiej możliwości musiałem rozpatrzeć inne dostępne środki bezpieczeństwa. Drugim w miarę skutecznym i prostym do wprowadzenia rozwiązaniem jest weryfikacja sms. Numery telefonów są przechowywane w bazie danych i nie ma możliwości utworzenia drugiego konta z tym samym numerem telefonu. Ponadto, każdy numer telefonu w Polsce musi mieć zarejestrowanego właściciela. Oznacza to, że weryfikacja takiego numeru mogłaby potwierdzić częściowo tożsamość posiadacza konta i utrudnić tworzenie wielu kont na serwerze. Po przeanalizowaniu zagadnienia, znalazłem jedynie serwisy wysyłające SMS, które wymagają podania danych firmy do założenia konta. Wychodzi na to, że nie ma możliwości, aby osoba prywatna skorzystała z tego typu rozwiązania. Udało mi się odnaleźć jedno darmowe rozwiązanie, ale to po bliższym zapoznaniu się okazało się być pozbawione zabezpieczeń. Serwis używał pozyskane adresy email do wysyłania reklam oraz spamu. Dodatkowo, natrafiłem w sieci na kilka opinii wskazujących na to, że po otwarciu e-maila, urządzenia podejrzanie zwalniały. Ponadto, strona główna łączyła się przez ftp nie szyfrując danych. Z powyższych powodów, metody te nie zostały dodane do projektu. Zaimplementowanie tych funkcjonalności na pewno poprawiłoby bezpieczeństwo oraz wiarygodność serwisu. Jako zastępstwo dodałem system uwierzytelnienia konta za pomocą podanego adresu mail. Jeśli użytkownik nie potwierdził swojego adresu, nie będzie miał możliwości głosowania.

# 4.Opis projektu

Poniżej przedstawiony został koncepcyjny model projektu.



Rys. nr. 1 Rysunek koncepcyjny projektu

# 4.1 Specyfikacja wymagań od strony użytkownika:

Użytkownik może założyć konto podając: imię, nazwisko, nazwę użytkownika, hasło, datę urodzenia, e-mail, numer telefonu i PESEL.

Pola te są sprawdzane pod względem poprawności, to znaczy:

* Treść pola nazwa użytkownika nie może się powtarzać w bazie danych i musi należeć tylko do jednej osoby.
* Pole hasło jest sprawdzane pod względem długości, jeśli ma mniej niż 8 znaków to zwróci błąd. Funkcja analizuje też czy zawiera ono literę małą, wielką, cyfrę i znak specjalny. Jeśli zawiera minimum trzy z wymienionych składników to hasło jest zaakceptowane jako silne.
* Zawartość pola e-mail nie może być wcześniej zapisana w bazie danych. Prawdziwość adresu jest weryfikowana po pierwszym zalogowaniu się użytkownika. Jeśli się nie powiedzie, nie ma możliwości oddania głosu.
* Wartość pola numer telefonu musi być unikatowa w bazie danych. Funkcja walidacji sprawdza czy pole składa się z samych cyfr i czy jest ich dziewięć.
* PESEL również nie może się powtarzać w bazie danych. Zgodnie z wytycznymi jak interpretować liczby w numerze PESEL (opisane na stronie <https://www.gov.pl/web/gov/czym-jest-numer-pesel>), sprawdzana jest zgodność z datą urodzenia oraz suma kontrolna. Z punktu widzenia projektu płeć nie odgrywa żadnej roli, dlatego nie jest to sprawdzane i zapisywane.

Po założeniu konta użytkownik ma możliwość zalogowania się za pomocą swojej nazwy użytkownika oraz hasła, po czym otrzymuje możliwość zweryfikowania swojego adresu e-mail. Po potwierdzeniu tego, wszystkie składowe strony są dla niego dostępne.

Użytkownik ma możliwość skorzystania ze wszystkich zawartych w projekcie funkcji. W zakładce „Roczniki”, gdzie znajduje się podział na lata od 2013 do 2021, użytkownik ma dostęp do listy ustaw, która zawiera linki do dokumentów opublikowanych przez rząd. Na stronie tej, przy każdej ustawie znajdują się przyciski „Zagłosuj” oraz „Sprawdź”. Po naciśnięci przycisku „Zagłosuj” użytkownik, jeśli jeszcze nie oddał głosu na daną ustawę, ma możliwość wyrażenia jednej z następujących opinii:

-Za.

-Przeciw.

-Wstrzymał się.

Po zatwierdzeniu odpowiedzi nie ma możliwości zagłosowania ponownie lub zmiany swojego głosu.

Po naciśnięci przycisku „Sprawdź” użytkownik może sprawdzić jakie są aktualne wyniki głosowania dla poszczególnej ustawy. Nie ma możliwości sprawdzenia czy i w jaki sposób głosowali inny użytkownicy.

W dowolnym momencie użytkownik może wylogować się z serwisu poprzez naciśnięcie przycisku „wyloguj” w prawym górnym rogu ekranu.

# 4.2 Architektura projektu

Serwis internetowy został stworzony w oparciu o architekturę klient-serwer. Warstwa użytkownika, której funkcjonalności zostały opisane w podpunkcie 4.1 działa w przeglądarce internetowej, która służy w tym przypadku za klienta i wysyła zapytania do serwera.

Serwer działa na wcześniej wspomnianym już systemie operacyjnym Centos 7. Zostały tam zainstalowane narzędzia niezbędne do uruchomienia projektu:

* Serwer i klient PostgreSQL
* Psycopg2
* Python3
* Django
* BeautifulSoup
* Numpy
* Python 3.6 requests

Skrypty służące do pobierania danych oraz aktualizacji tabel w bazie są zautomatyzowane za pomocą programu cron. Poniżej można zobaczyć wylistowane wpisy z tabeli crontab użytkownika serwisowego:

Rys.nr. 2 Zrzut ekranu przedstawiający tabele crontab

Wpisy te składają się z informacji o czasu uruchamiania i zadań do wykonania. W powyższym przypadku pierwsza linia oznacza: uruchomiaj skrypt vote\_v.1.0.py za pomocą python3 codziennie o 4:00, a drugi oznacza uruchamiaj skrypt export\_to\_database.py za pomocą python3 każdego dnia o 5:00. Rolą vote\_v.1.0.py jest zebranie danych ze strony [https://isap.sejm.gov.pl](https://isap.sejm.gov.pl/) oraz zapisanie ich do pliku csv. Przeznaczeniem skryptu export\_to\_database.py jest przeczytanie poprzednio przygotowanego pliku i zaimportowanie nowych rekordów do bazy danych. Dokument w standardzie CSV zawiera następujące dane:

* Unikatowy numer indeksu, który jest generowany przez skrypt na bazie daty oraz numeru ustawy w danym roku.
* Rok w jakim ustawa została procedowana.
* Numer ustawy.
* Tytuł ustawy.
* Link do dokumentu.

Skrypt export\_to\_database.py wykonuje również dodatkowe operacje na bazie danych. Więcej informacji na ten temat można znaleźć w podpunkcie 4.3, który opisuje sposób przechowywania danych.

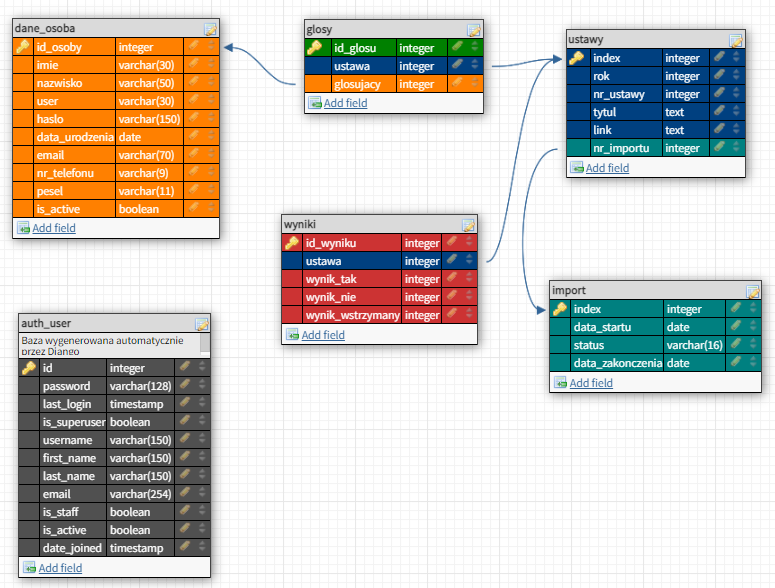
Za pomocą komendy: sudo systemctl start postgresql uruchamiany jest serwer PostgreSQL, Szczegóły tabel, relacje między nimi oraz skrypty wykorzystane do stworzenia ich są opisane w podpunkcie 4.3.

Wszystkie wspomniane komponenty komunikują się z częścią serwerową, dzięki projektowi stworzonemu w Django. Serwer łączy się bezpośrednio z bazą danych. Ma możliwość pobierania, zapisywania informacji oraz aktualizacji ich. Odpowiada za wyświetlanie części wizualnej po stronie użytkownika, oznacza to, że generuje kod strony w przeglądarce. Oddziela dane wrażliwe, tak aby w przeglądarce użytkownik widział tylko informacje przeznaczone bezpośrednio dla niego. Podczas tworzenia konta to serwer odpowiada za walidację wszystkich pól. Sprawdza poprawność wpisywanych danych za pomocą napisanych przeze mnie funkcji, tworzy zapytania do bazy i odrzuca błędnie wypełnione formularze wyświetlając użytkownikowi komunikaty z opisem błędów. Zapewnia autoryzację podczas logowania i jest odpowiedzialny za potwierdzanie adresu e-mail.

Serwer udostępnia niezalogowanemu użytkownikowi tylko stronę do logowania oraz stronę do zakładania konta. Aby w pełni korzystać z serwisu, należy potwierdzić adres e-mail wpisując kod otrzymany na podanego maila. Niepotwierdzony użytkownik dalej ma dostęp do wyników głosowania oraz listy ustaw, lecz nie ma możliwości oddania głosu. W tym przypadku strona będzie pełniła tylko funkcję informacyjną. Jeśli zalogowany użytkownik nie ukończył 18 lat, strona również uniemożliwi mu głosowanie. Operacje dodawania użytkownika lub oddawania głosu wymagają zapisywania do bazy danych, co jest realizowane przez serwer Django. Informacje wyświetlane na stronie internetowej przekazywane są w większości w HTML i dodatkowo wzbogacone wizualnie za pomocą stylów w CSS. Na stronie użyty został również JavaScript w celu wizualnego przedstawienia wyników głosowania. Dzięki Django, serwer ma możliwość przekazywania zmiennych bezpośrednio do stron internetowych oraz wykonywania na nich części kodowej. Informacje, które są przetwarzane to między innymi nazwa użytkownika, ustawy pochodzące z konkretnego rocznika, wyniki głosowania dla poszczególnych tabel. Więcej informacji na temat przekazywanych informacji do poszczególnych tabel znajduje się w podrozdziale „Dokumentacja techniczna”.

# 4.3 Koncepcja przechowywania danych

Do przechowywania danych użyty został PostgreSQL. Oto diagram przedstawiający tabele oraz relacje między nimi:



Rys. nr.3 Schemat bazy danych

Tabela auth\_user została wygenerowana bezpośrednio po stworzeniu projektu przez Django.

Tabele auth\_user oraz dane\_osoba są ze sobą powiązane po stronie serwera web. W bazie danych nie istnieje bezpośrednie połączenie między nimi.

Ponadto, tabele te częściowo współdzielą ze sobą te same dane na potrzeby użytkowania wbudowanych funkcji Django oraz możliwości wykonywania operacji na danych użytkownika. Bardzo istotnym faktem jest to, że tabele przechowują zahashowane hasła w sposób bezpieczny. Oznacza to, że zostały one wcześniej zakodowane i odczytane będą mogły być jedynie przez serwer web, który posiada do tego odpowiedni klucz. Dzięki temu w razie wycieku danych hasła dalej pozostaną bezpieczne.

Kolejnym elementem jest tabela ustawy, która jest bezpośrednio powiązana ze skryptami wykorzystującymi bibliotekę beautifulsoup. To właśnie do niej trafiają dane o ustawach, które wyświetlane są na stronie internetowej. Po każdym zaimportowaniu danych do tabeli importy trafia rekord z informacją o tym, kiedy on miał miejsce. Każda zaimportowana ustawa posada informacje o tym, kiedy była importowana poprzez dodawanie do niej klucza obcego pochodzącego z tabeli importy. Jeśli jakaś ustawa była już wcześniej zaimportowana do tej tabeli, rekord jest pomijany i sprawdzana jest następny. Zabieg ten pomaga uniknięcia wpisania do tabeli dwa razy tej samej informacji.

Ponadto, z tabelą ustawy oraz z skryptami importującymi dane do bazy powiązana jest tabela wyniki. To w niej przechowywane są informacje na temat głosowania na poszczególne ustawy. Za każdym razem, gdy do tabeli ustawy trafia nowy rekord, tabela wyniki również dodaje wpis z takim samym kluczem. Taki wiersz w polach wynik\_tak, wynik\_nie, wynik\_wstrzymany ma wartość ustawianą na zero. Dodawanie tych rekordów odbywa się za pomocą skryptu export\_to\_database.py.

Ostatnia opisana przeze mnie tabela o nazwie głosy, przechowuje informacje o tym czy użytkownik głosował na konkretną ustawę. Tabela ta przechowuje identyfikator głosu, numery ustaw oraz id użytkownika, powiązując te dane ze sobą i w ten sposób kontrolując kto oddał głos. Serwer odwołuje się do niej by zadecydować czy wczytać formularz i po każdym oddaniu głosu doda wpis do tabeli o tym kto głosował i na co. Dodatkowo, podczas głosowania na ustawę, jedno z następujących pól w rekordzie tabeli wyniki: wynik\_tak, wynik\_nie, wynik\_wstrzymany jest zwiększane o jeden, w zależności od oddanego głosu. Ten sposób uniemożliwia pozyskanie informacji nt. poglądów i treści oddanych głosów dla konkretnego użytkownika.

# 4.4 Projekt interfejsu użytkownika

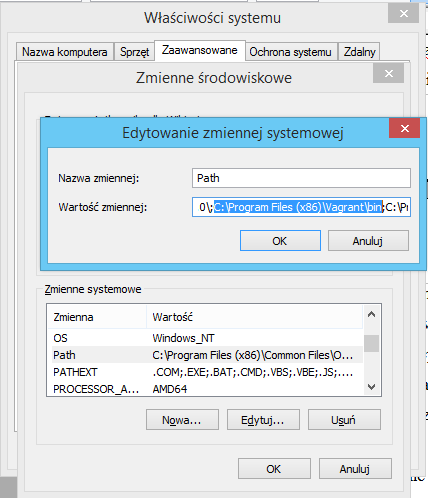
Interfejs użytkownika zachowuje pełną przejrzystość. Użytkownik może skorzystać z niewielu opcji, ale jest to spowodowane faktem, że do przeprowadzenia głosowania nie jest wymagane zbyt wiele. Główne funkcjonalności dostępne dla użytkownika to zakładanie konta, logowanie i wylogowanie się, autoryzacja maila, głosowanie na ustawę oraz sprawdzanie wyników.

# 4.5 Dokumentacja techniczna

Do zainstalowania i uruchomienia projektu potrzebne będą narzędzia:

* Vagrant
* VirtualBox
* Opcjonalnie Git
* Opcjonalnie GIT BASH (można skorzystać z cmd)
  + 1. Uruchomienie środowiska oraz konfiguracja.

Podczas instalacji Vagrant powinna zostać dodana ścieżka programu do zmiennej środowiskowej „PATH”. Jeżeli nie została dodana, należy zrobić to ręcznie. Aby sprawdzić wartości zapisane w zmiennej „PATH”, należy w menu start uruchomić okno „Edytuj zmienne środowiskowe dla systemu”. W zakładce „zaawansowane” trzeba przejść przyciskiem „Zmienne Środowiskowe” do odpowiedniej sekcji, a następnie znaleźć rekord zawierający zmienną „Path”. Wewnątrz powinna znajdować się ścieżka do instalacji programu Vagrant oddzielona od pozostałych średnikami. Obraz poniżej prezentuje konfigurację:



Rys. nr 4. Ustawienia zmiennych środowiskowych dla systemu Windows

Następnie uruchamiamy cmd i wpisujemy komendę:

vagrant plugin install vagrant-vbguest

Spowoduje to instalację wtyczki, która automatycznie konfiguruje funkcjonalność vbguest na maszynie wirtualnej. Dzięki temu możliwe będzie udostępnienie folderu z hosta do maszyny wirtualnej jako mount.

Pierwsze uruchomienie serwera, bazy danych oraz serwisu:

1. Należy uruchomić Git i sklonować repozytoriom ze strony lub skorzystać z paczki instalacyjnej dołączonej do projektu.
2. Następnie trzeba przejść do katalogu zawierającego rozpakowany projekt, wszystkie kolejne ścieżki będą podane względem tej lokalizacji.
3. Po rozpakowaniu projektu, należy uruchomić GIT BASH lub cmd i przejść do katalogu TwojeReferendum\_projekt\vagrant\.
4. Uruchamiamy komendę: vagrant up (pierwsze uruchomienie komendy może zwrócić błąd dotyczący udostępnianie katalogu, zostanie on naprawiony kolejnymi komendami). Komenda ta uruchamia skrypt bash ze ścieżki TwojeReferendum\_projekt\vagrant\provide.sh, który wykonuje kolejne zadania:
   * Instaluje aktualizację aktualne dostępne dla systemu,
   * Instaluje program vim
   * Dodaje repozytorium epel-release do yum, zapewnia dostęp do dodatkowego oprogramowania open sorce, wykorzystywanego w projekcie.
   * Ponownie aktualizuje oprogramowanie,
   * Instaluje ansible
   * Uruchamia playbook ansible, który sprawdza czy poniższe moduły są zainstalowane, jeśli nie to je dodaje:
5. git
6. serwer PostgreSQL
7. dodatkowe biblioteki potrzebne do obsługi bazy danych
8. python3
9. django
10. beautiful soup
11. numpy
12. request

* Aktualizuje django do najnowszej wersji (aktualnie projekt działa w wersji 3.2.0)
* Wykonuje migrację dla django
* Startuje serwer

1. Gdy maszyna wirtualna zostanie stworzona, wykonujemy następujące komendy:

* vagrant ssh -c 'sudo yum -y update'- (instaluje aktualizacje dostępne dla system maszyny wirtualnej)
* vagrant ssh -c 'sudo shutdown now' – (wyłącza maszynę wirtualną)

1. W razie problemów z narzędziami zainstalowanymi na serwerze można uruchomić polecenie: vagrant provision. Spowoduje to przegląd aktualnej konfiguracji środowiska i dodanie wymaganych narzędzi.

Po wykonaniu raz tej procedury, możemy uruchamiać nasz serwer zgodnie z opisem przedstawionym w podrozdziale 6.3. Należy dodatkowo zadbać o bazę danych opisaną w następnym podrozdziale.

# 4.5.2 Konfiguracja i import danych do bazy.

Proces tworzenia tabeli zgodnie z tym, co zostało przedstawione w podpunkcie 4.3 odbywa się za pomocą playbooka ansible znajdującego się w pliku TwojeReferendum\_projekt\ansible\configure-database.yml. Skrypt ten konfiguruje:

* ustawienia bazy danych
* umożliwienie zalogowania się za pomocą hasła z localhost dla wszystkich użytkowników
* umożliwienie zalogowania się za pomocą autentykacji peer z localhost dla użytkownika postgres
* umożliwienie zalogowania się bez autentykacji z prywatnej sieci host-only tworzonej przez VirtualBox
* IP na jakim nasłuchuje serwer bazodanowy,
* restart bazy danych, jeśli konfiguracja bazy została zmieniona
* sprawdzenie czy serwer jest uruchomiony, jeśli nie to wystartowanie usługi
* tworzenie nowego klastra bazodanowego o nazwie projekt
* tworzenie użytkownika serwisowego django w nowej bazie danych

Na samym końcu skrypt ten uruchamia plik SQL ze ścieżki: TwojeReferendum\_projekt\baza\DB\sql-fina.sql, który dodaje do bazy danych tabele oraz relacje pomiędzy nimi.

# 4.5.3 Obsługa środowiska po konfiguracji

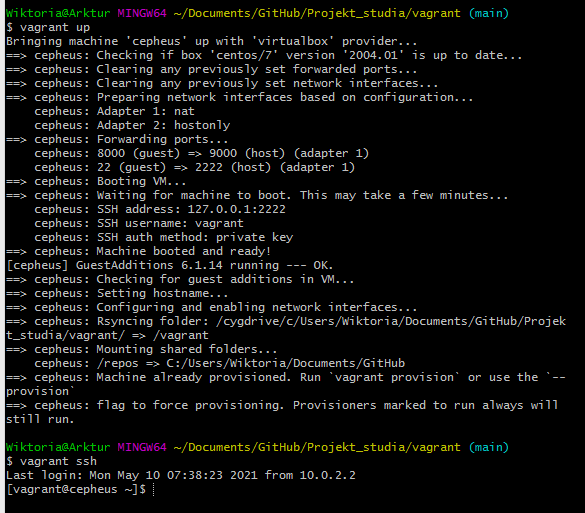
Aby uruchomić środowisko testowe, należy uruchomić aplikację GIT BASH (zastępczo cmd) i przejść do katalogu zawierającego projekt, a następnie do katalogu vagrant. Następnie, trzeba wywołać komendę:

vagrant up

po czym zalogować się do serwera poleceniem:

vagrant ssh

Poniżej przedstawiam zrzut ekranu z terminala przestawiający powyższe operacje:



Rys. Nr. 5 Start środowiska

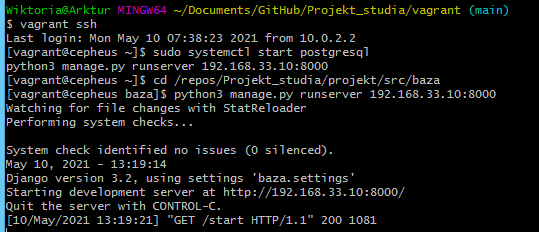
Po zalogowaniu się do serwera musimy wykonać jeszcze dwie czynności: uruchomić serwer bazodanowy oraz uruchomić serwis internetowy. W celu wykonania tych zadań należy wpisać poniższe komendy:

sudo systemctl start postgresql

cd /repos/TwojeReferendum\_projekt/projekt/src/baza

python3 manage.py runserver 192.168.33.10:8000

Pierwsza z nich uruchamia serwer, druga służy do przejścia do odpowiedniego katalogu zawierającego skrypt manage.py. Ostatnia linijka uruchamia serwis internetowy. Od tego momentu osoby w tej samej sieci będą mogły wejść na stronę za pośrednictwem przeglądarki używając adresu <http://192.168.33.10:8000/start> . Po wykonaniu komend i przejściu na adres strony terminal GIT BASH powinien wyglądać następująco:



Rys. nr. 6 Uruchomienie bazy danych i serwisu internetowego.

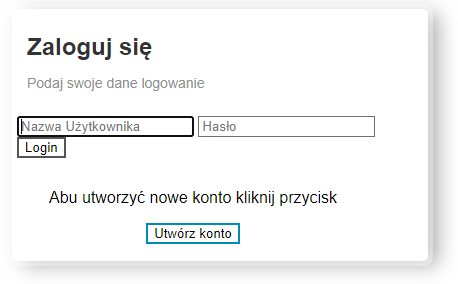
Teraz za każdym razem gdy będziemy przemieszczać się na stronie, serwer będzie zwracał kody odpowiedzi http.

# 4.5.4 Projekt Django

Po uruchomieniu serwera, bazy danych oraz serwisu internetowego pod adresem <http://192.168.33.10:8000/start> mamy dostęp do strony startowej.

# 4.5.4.1 Strona startowa

Strona startowa umożliwia dwie akcje: zalogowanie się i utworzenie konta. Jeśli użytkownik poda niepoprawne dane, wyświetlony zostanie odpowiedni komunikat. W przeciwnym razie, zostanie zalogowany do serwisu i przeniesiony do strony głównej. Wyjątkiem jest pierwsze logowanie i każde kolejne w przypadku, gdy adres e-mail nie został potwierdzony.



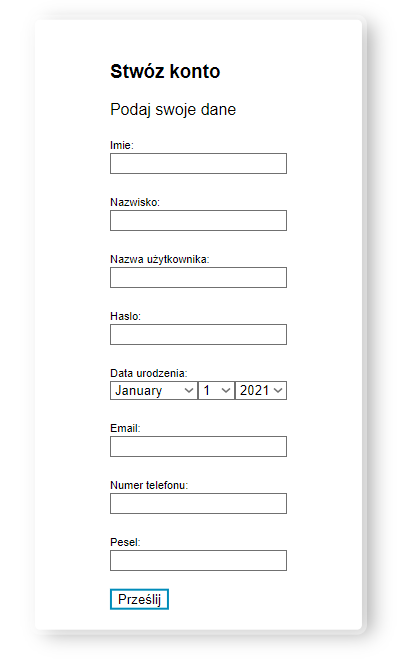
Rys. Nr. 7 Strona logowania

# 4.5.4.2 Tworzenie konta

Po wciśnięciu przycisku „Utwórz konto” możemy stworzyć nowego użytkownika. W tym celu musimy podać:

* Imię,
* Nazwisko,
* Nazwę użytkownika – nazwa użytkownika musi być unikatowa. Jeśli podamy nazwę, która już istnieje, formularz przekieruje nas do strony ze zwróconym błędem.
* Hasło – Hasło musi składać się z przynajmniej 8 znaków i zawierać minimum po jednym znaku z co najmniej trzech z następujących grup: litery małe, litery wielkie, liczby, symbole specjalne: @ # $ % ^ & \* ().
* Data urodzenia – należy wybrać ją z list rozwijanych.
* E-mail – musi zawierać znak @. Adres musi być unikatowy. Jeśli podamy e-mail już istnieje w bazie danych, formularz przekieruje nas do strony komunikującej odpowiedni błąd.
* Numer telefonu – strona akceptuje jedynie numery składające się z 9 cyfr. Numer telefonu musi być unikatowy. Jeśli podany numer telefonu już istnieje w bazie danych, formularz przekieruje nas do strony komunikującej odpowiedni błąd.
* PESEL – składa się z 11 cyfr. PESEL musi mieć poprawną formę oraz być zgodny z datą urodzenia. Numer musi być unikatowy. Jeśli podamy PESEL która już istnieje w bazie danych, formularz przekieruje nas do strony komunikującej odpowiedni błąd..

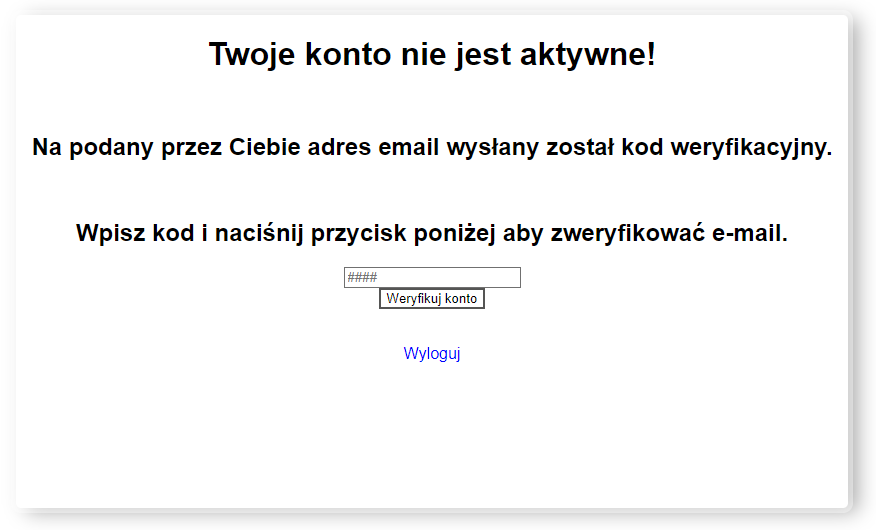
Jeżeli wszystkie informacje będą poprawne, po naciśnięciu przycisku „Prześlij” konto zostaje utworzone, o czym poinformuje nas komunikat na stronie. Teraz jesteśmy gotowi do zalogowania się po raz pierwszy.



Rys. Nr.8 Zakładanie konta

# 4.5.4.3 Weryfikacja konta.

Weryfikacja konta odbywa się za pomocą kodu wysyłanego na wcześniej podany adres e-mail. Po każdym zalogowaniu się użytkownik ma możliwość aktywowania swojego konta. Jeśli tego nie zrobi, nie będzie miał możliwości oddawania głosów. Na adres email wysyłany jest 4 cyfrowy kod, który należy wpisać zgodnie z instrukcją wyświetlaną przez stronę. Po autoryzacji użytkownik ma możliwość pełnego korzystania ze strony.



Rys. Nr.9 Autoryzacja konta

# 4.5.4.4 Strona główna i podstrony

Po zweryfikowaniu adresu e-mail, otrzymujemy pełen dostęp do strony. Ze strony głównej możemy przejść do zakładek:

* Home- przenosi nas do strony głównej, zawiera ogólne informacje na temat strony.
* Roczniki – przenosi nas do strony rocznikami, która zawiera listę ustaw z podanego rocznika.
* Statystyki – przenosi nas do strony prezentującej przykładowe wyniki głosowana na poszczególną ustawę. Jest to jedynie przykład wizualny.
* Kontakt - przenosi nas do strony, która zawiera informacje o autorze i dane kontaktowe.
* Regulamin - przenosi nas do strony regulaminem.

Powyższe zakładki i linki są dostępne z poziomu każdej podstrony, gdyż zawarte są w stworzonym przeze mnie szablonie.

Dodatkowo w prawym górnym rogu widnieje informacja o nazwie aktualnie zalogowanego użytkownika oraz przycisk „Wyloguj”, który kończy sesję użytkownika i przenosi do strony startowej.



Rys. Nr.10 Strona Główna

# 4.5.4.5 Przegląd ustaw.

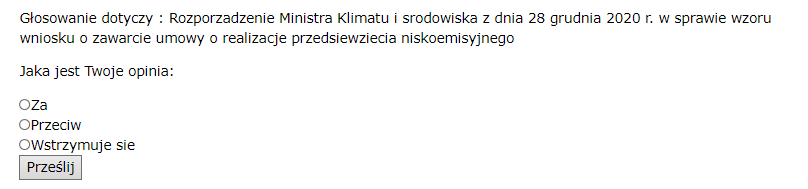
Strona pod adresem <http://192.168.33.10:8000/Roczniki> udostępnia listę ustaw od roku 2013 do bieżących. Po przejściu na wybrany rocznik, wyświetla się nam tabela zawierające następujące informacje: pozycje w dzienniku ustaw, link przenoszący bezpośrednio do aktu prawnego, tytuł ustawy oraz przycisk „Głosuj” i „Wyniki”. Zakładki zawierające ustawy od 2013 do 2020 roku mają charakter archiwalny, na podstronach tych nic się nie zmienia. Część z ustawami z 2021 roku jest na bieżąco aktualizowana.



Rys. Nr. 11 Lista ustaw

# 4.5.4.6 Głosowanie

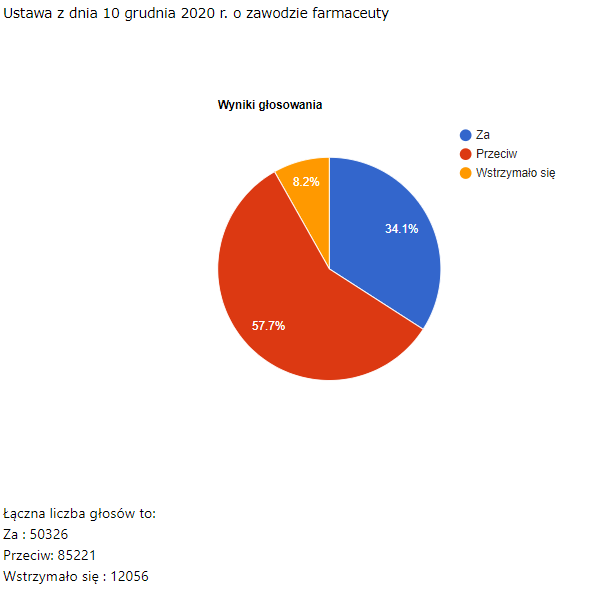
Głosowanie możliwe jest po wciśnięciu przycisku „Zagłosuj”, gdy znajdujemy się w jednej z podstron opisywanych w punkcie 4.4.5. Jeżeli konto użytkownika zostało aktywowane, serwer sprawdza na podstawie daty urodzenia czy użytkownik ma ukończone 18 lat. Jeśli nie, to dostępne będą jedynie informacje o aktach prawnych. Jeżeli użytkownik ukończył 18 lat, serwer sprawdza, czy został już oddany głos na daną ustawę. Jeśli tak, to nie będzie możliwości zagłosować ponownie ani zmienić głosu. Jeśli nie, to można oddać głos na jedną z 3 opcji: „Za”, „Przeciw”, „Wstrzymuję się”. Formularz wygląda następująco:

Rys. Nr. 12 Formularz głosowania

Serwis zapewnia pełną anonimowość głosowania. Oznacza to, że administrator serwisu otrzymuje jedynie informacje o tym, kto i na co głosował (aby uniknąć możliwości wielokrotnego głosowania na tą samą ustawę). Nie posiada informacji o tym jaki został oddany głos przez danego użytkownika. Dzięki temu wybory cały czas są tajne.

# 4.5.4.7 Sprawdzanie wyników

Analogicznie jak w poprzednim podpunkcie, aby sprawdzić wyniki głosowania dotyczące danej ustawy, należy wcisnąć przycisk „Wyniki”. Wyniki wyświetlane są bez względu na to, czy osoba korzystająca ze strony jest pełnoletnia i czy potwierdziła adres mailowy. Rezultat głosowania prezentowany jest na procentowym diagramie kołowym, a poniżej niego wyświetla się informacja o liczbach głosów. Przykład można zobaczyć poniżej:



Rys. nr 13 Przykładowe wyniki głosowania.

# 4.6 Część programistyczna projektu Django

Projekt Django składa się z kilku elementów wymienionych poniżej. Każdy z nich zostanie opisany w kolejnych podpunktach:

* plik forms.py
* plik models.py
* plik settings.py
* plik urls.py
* plik views.py
* folder strona - zawierający bezpośrednio wszystkie pliki .html
* folder static - zawierający katalogi: css, który przechowuje plik style.css oraz photo zawierający obrazy wyświetlane na stronie.

Każdy plik .py zaczyna się listy zaimportowanych bibliotek, które są wymagane do zapewnienia funkcjonalności na stronie.

# 4.6.1 Forms.py

Plik ten zawiera on formularze, które są wyświetlane na stronie internetowej oraz pośredniczą przy operacjach bazodanowych. Plik ten zawiera definicje:

* ContactForm – formularz odpowiedzialny za dodawanie użytkownika. Opisuje sposób wyświetlania daty urodzenia w formularzu. Określa typ niektórych pól wyświetlanych w formularzu (słownik widgets).
* GlosyForm - formularz odpowiedzialny za dodawanie informacji od tym kto głosował na jaką ustawę.

# 4.6.2 Models.py

Moduł ten zawiera modele, które są bezpośrednim opisem obiektów w bazie danych. Dzięki temu, baza danych posiada reprezentację pól i ich wartości za pomocą zmiennych. Modele jakie zostały stworzone w odniesieniu do baz w tym pliku to:

* Ustawy – bezpośrednio powiązane z tabelą ustawy.
* Dane\_osoba - bezpośrednio powiązane z tabelą dane\_osoba.
* Glosy - bezpośrednio powiązane z tabelą glosy.
* Wyniki - bezpośrednio powiązane z tabelą wyniki.

Modele te pozwalają na wyświetlanie obiektów znajdujących się w tabelach na stronie internetowej za pośrednictwem nawiasów klamrowych, przykładowo: {{wyswietlany\_obiekt}}. Umożliwiają także pobieranie danych z bazy i wykonywanie na nich operacji.

# 4.6.3 Settings.py

W tym pliku znajdują się wszystkie ustawienia projektu. Konfiguracja zawiera:

* Informacje o katalogu domyślnym.
* Kod zabezpieczający (ważne, aby był tajny i nie przekazywany nikomu). Jest to jedno z głównych zabezpieczeń aplikacji.
* Zmiany trybu debugowania, do wyboru off/on.
* IP na jakich można uruchomić serwer.
* Listę zainstalowanych aplikacji w projekcie.
* Rozdaj używanych zabezpieczeń.
* Informacje o nazwie domeny.
* Schematy.
* Informacje dotyczące połączenia z bazą danych – Jest to bardzo istotny punkt, ponieważ projekt korzysta z zewnętrznej bazy danych. Konfiguracja wymaga podania silnika bazy danych, nazwy bazy, użytkownika, hasła, adresu IP i postu na jakim nasłuchuje serwer bazy danych.
* Informacje o sposobach walidacji i autoryzacji hasła.
* Informację o używanym języku.
* Informacje o strefie czasowej.
* Ścieżki domyślne, skąd pobierane są pliki do projektu (css, JavaScript, obrazy).

# 4.6.4 Urls.py

Dzięki temu plikowi mamy możliwość sformułowania sposobu wyświetlania poszczególnych adresów URL w przeglądarce. Plik ten umożliwia tworzenie dynamicznych ścieżek i podstron. Każda ustawa posiada swój unikatowy adres strony wygenerowany dynamicznie. W wygenerowanym URL zawiera się id ustawy, które jest przechowywane w bazie co zapewnia unikatowość adresu.

# 4.6.5 Views.py

Moduł pozwala zdefiniować, jakie pliki mają zostać otwarte podczas wywołania adresu strony zapisanego w urls.py. To właśnie tu zdefiniowane są walidacje po stronie serwera. Plik ten zawiera widoki i funkcje opisane poniżej:

* Widok Start – jest odpowiedzialny za proces logowania do strony i wyświetlania komunikatów, gdy dane logowania są nieprawidłowe.
* Widok wyloguj –wylogowuje użytkownika i przechodzi na stronę startową.
* Widok StronaGlowna – zwraca stronę główną.
* Widok zakladanie\_konta – Tu mieści się walidacja po stronie serwera, która jest odpowiedzialna za poprawność wpisywanych danych podczas zakładania konta oraz za sprawdzenie czy wpisane dane już istnieją. Jeśli wszystkie informacje są poprawne, do tabeli „dane\_osoba” dodawany jest użytkownik oraz bliźniaczo, z tym samym id, dodawany jest użytkownik do tabeli „auth\_user”..
* Widok blad – zwraca błąd z przekazywany z poszczególnych podstron. Wykorzystywany przez inne widoki.
* Widok Weryfikacja – Widok odpowiedzialny za weryfikację konta. Po uruchomieniu tego widoku na zadany adres email wysyłany jest kod autentykacji. Po wpisaniu go na stronie i wciśnięciu „potwierdź”, widok sprawdza poprawność podanego kodu z tym wysłanym na mail. Wykorzystywane jest do tego pole haslo z tabeli dane\_user, które jest nadpisywane tym kodem. Hasło do logowania jest przechowywane w tabeli auth\_user, więc nie tracimy go.
* Widok Utworzono – zwraca komunikat o prawidłowym utworzeniu użytkownika.
* Widok Statystyki – zwraca stronę Statystyki.html. Jest to strona poglądowa.
* Widok Roczniki – zwraca stronę zawierającą listę roczników z ustawami.
* Widok Regulamin – zwraca stronę z regulaminem.
* Widok Kontakt – zwraca stronę z informacjami kontaktowymi.
* Widok roczniki – zwraca unikalną podstronę dla każdej ustawy. Adres jest tu generowany dynamicznie.
* Widok glosowanie – zwraca stronę do głosowania. Widok ten waliduje czy użytkownik jest pełnoletni i czy jego konto jest aktywne oraz czy nie głosował już wcześniej. Jeśli wszystkie warunki są prawidłowe, to umożliwia głosowanie i wywołuje zapytanie do bazy zwiększające stosowny wynik oraz dodaje informacje o oddanym głosie do tabeli glosy.
* Widok Wyniki\_głosowania - Zwraca strony z wynikami dla każdej ustawy. Adresy URL generowane są dynamicznie za pomocą numeru id danej ustawy.
* Funkcja sql\_check – Jest to funkcja sprawdzająca dane wejściowe dla formularza zakładającego konta. Funkcja sprawdza czy podane dane zawierają znaki niedozwolone, czyli spacja,” =” oraz “ ” ”. Jest to zabezpieczenie przed sql injection.

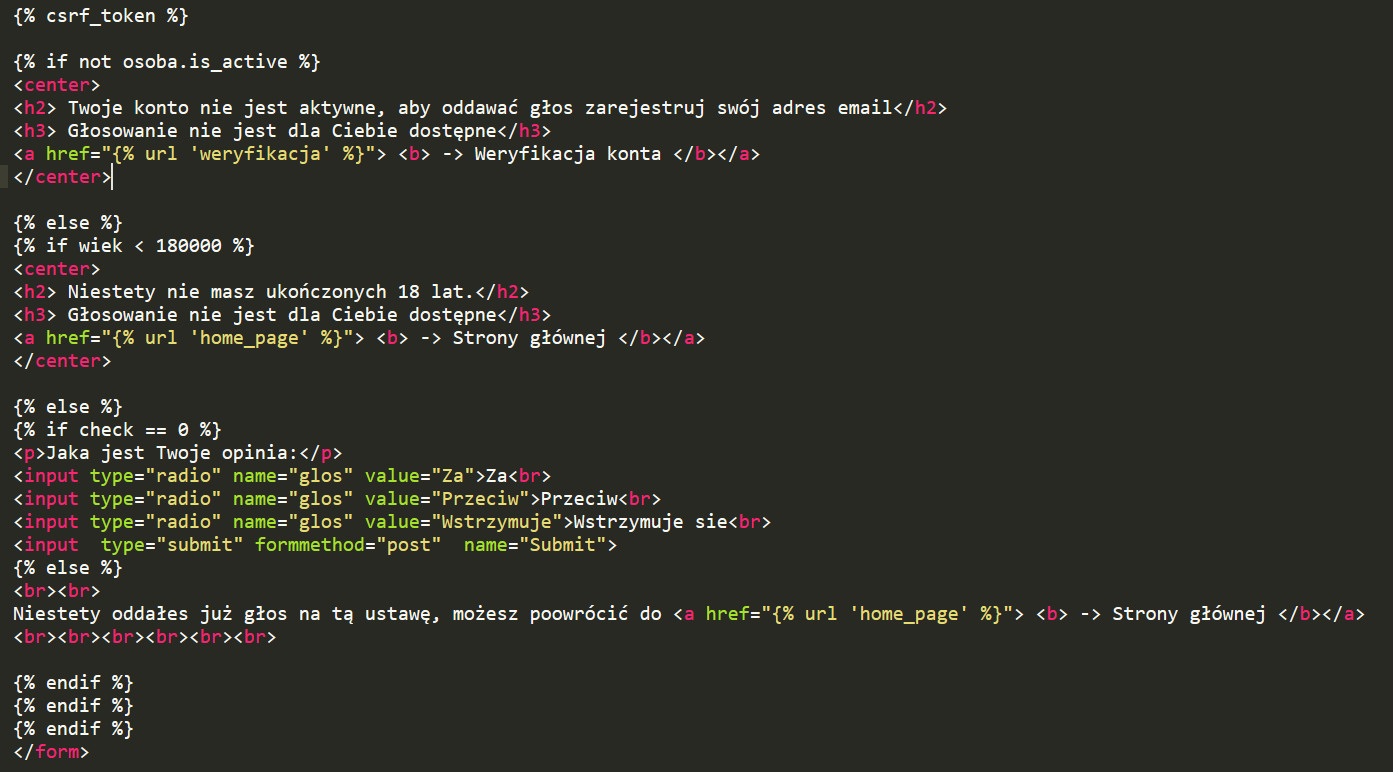
# 4.7 Opis skryptów pobierających dane ze strony isap.sejm.gov.pl i importujących je do bazy danych

Są to dwa skrypty podzielone na role. Pierwszy - vote\_v.1.0.py pobiera dane ze strony <https://isap.sejm.gov.pl> i zapisuje je w pliku .csv (format zapisywanych danych opisany został w podpunkcie 4.2, gdzie omawiana była architektura systemu). Skrypt ten opiera się w głównej mierze na bibliotece Beautiful Soup. Funkcje z tej biblioteki analizują dane pobrane ze strony i przeszukują je, wyłuskując potrzebne informacje.

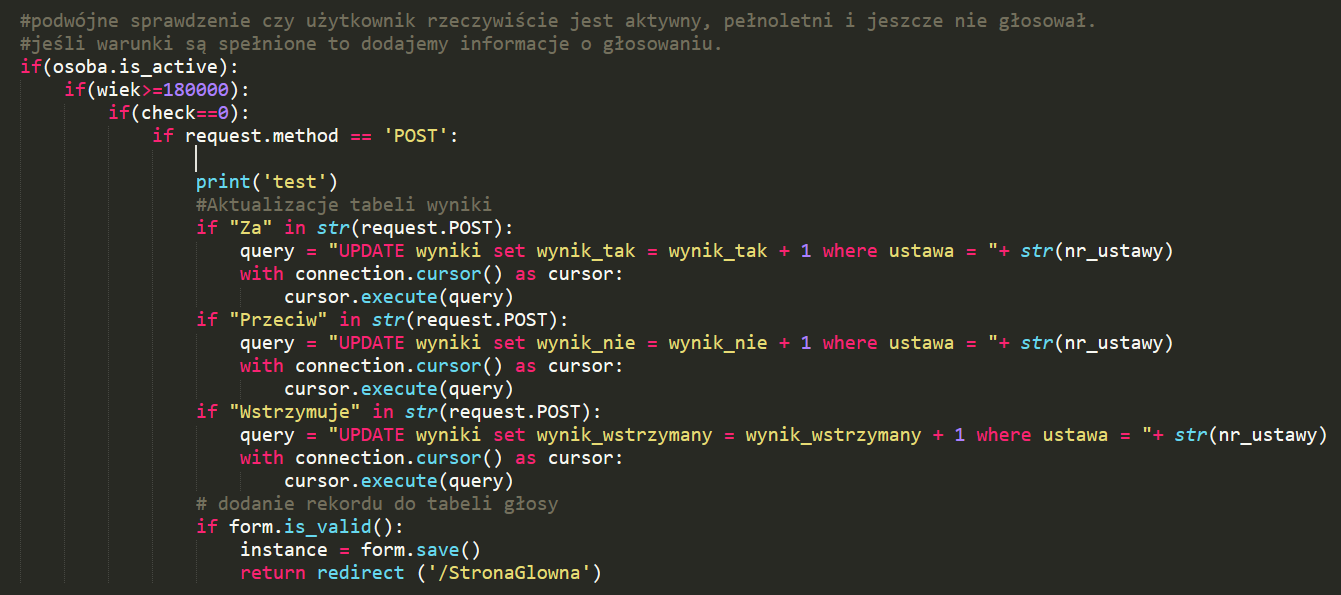
Drugi skrypt - export\_to\_database.py używa zapisanych danych i wykonuje operacje na bazie danych. Skrypt ten tworzy rekordy do tabeli ustawy i równolegle do tabeli wyniki.

# 4.8 Bezpieczeństwo.

Bezpieczeństwo w projektach zajmujących się przechowywaniem danych wrażliwych jest bardzo istotne. Projekt jest zabezpieczony przed wyciekami informacji. Narzędzia wbudowane w Django bardzo skutecznie radzą sobie z tym zagadnieniem. Aby na stronie wyświetlały się tylko te dane do których powinien mieć dostęp jedynie użytkownik, Django pozwala wykonywać instrukcje warunkowe, który w zależności od danych przypisanych do użytkownika wyświetlą właściwy kod (przykład poniżej). Część wizualna poparta jest w tym przypadku częścią kodową. Oznacza to, że nawet jeśli zmienimy kod wejściowy strony to walidacja i tak nie pozwoli nam na przesłanie lub odebranie danych:



Rys Nr. 14 Kod strony oddajgłos.html



Rys Nr. 15 Kod pliku views.py odpowiedzialny za walidację głosowania.

Kolejną istotną kwestią jest sposób zapisywania i przechowywania haseł w bazie danych. W tym przypadku Django udostępnia nam funkcję make\_password(haslo). Funkcja ta haszuje hasło i zapisuje je w takiej postaci do bazy danych. Dla przykładu, gdy podczas tworzenia użytkownika podamy hasło: “T3st123!”, w bazie danych zostanie zapisany rekord:

pbkdf2\_sha256$260000$9yXZ3jqOrqme8VzvGHDkfj$FiqjRs53qph7m2AYeHS4vioCqU5j8YzmOM23Z00hCMQ=

Aby je odszyfrować, Django używa do tego specjalnego klucza zadeklarowanego w settings.py w zmiennej SECRET\_KEY.

Aby zabezpieczyć się przed sql injection, stworzona została specjalna funkcja, która sprawdza przed wykonaniem zapytania do bazy czy dane zawierają znaki: spacja, = oraz cudzysłów ( “ ). Jeśli tak, to strona wyświetli odpowiedni komunikat i zapytanie nie zostanie wywołane.

# 5. Testy

Aby zabezpieczyć się prze SQL injection, pola przyjmujące wartości w formularzu są w pierwszej kolejności walidowane, aby sprawdzić czy nie zawierają nieodpowiednich znaków. Zdecydowałem się na rozwiązanie tego problemu za pomocą „whitelist”. Rozwiązanie to jest jedno z prostszych do zaimplementowania i polega na sprawdzeniu czy dane wejściowe zawierają niedozwolone znaki. W moim przypadku program sprawdza czy używane są znaki :

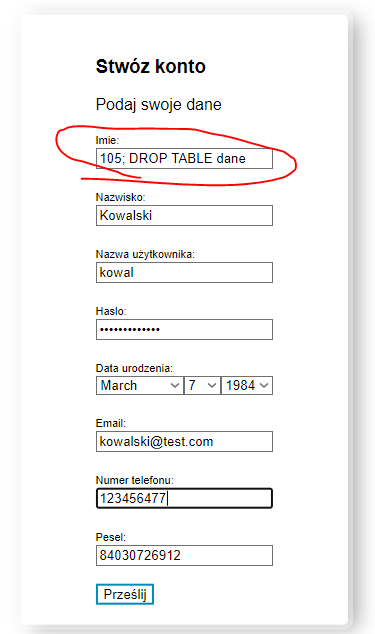
-spacji

-apostrofu „ ″”

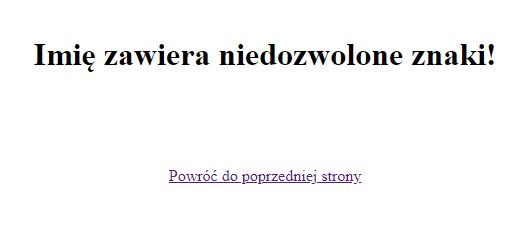
- równości „=”

- średnika „;”

Wycięcie tych znaków spowodowało wykluczenie możliwości stworzenia sql injection. Oto jak zabezpieczenie zadziałało w praktyce:

Rys Nr. 16 Próba wywołania sql injection

Próba przetworzenia takiego formularza skończyła się następującym komunikatem :

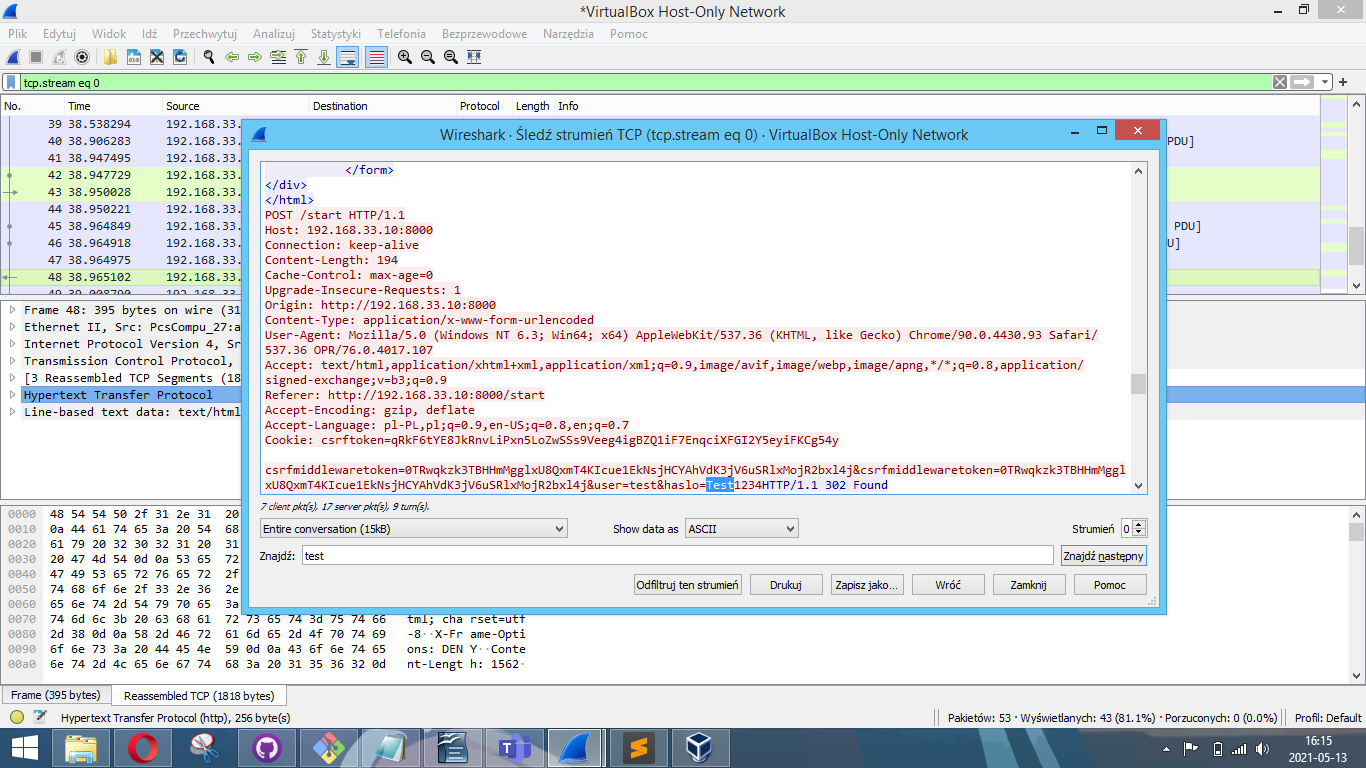


Rys Nr. 17 Zabezpieczenie dla pola Imię

Każda wartość, która wykonuje zapytanie do bazy danych została odpowiednio zabezpieczona.

Sprawdzenie czy przechwytywanie danych podczas logowania jest możliwe.

Do wykonania tego testu wykorzystałem program Wireshark który pozwala na śledzenie danych przesyłanych przez sieć do której jest połączony. Podczas prac nad projektem strona dostępna była poprzez sieć lokalną i połączenie do niej odbywało się z hosta z tej samej podsieci. Za pomocą programu Wireshark udało mi się przechwycić sygnał i niestety połączenie okazało się nie być zaszyfrowane:

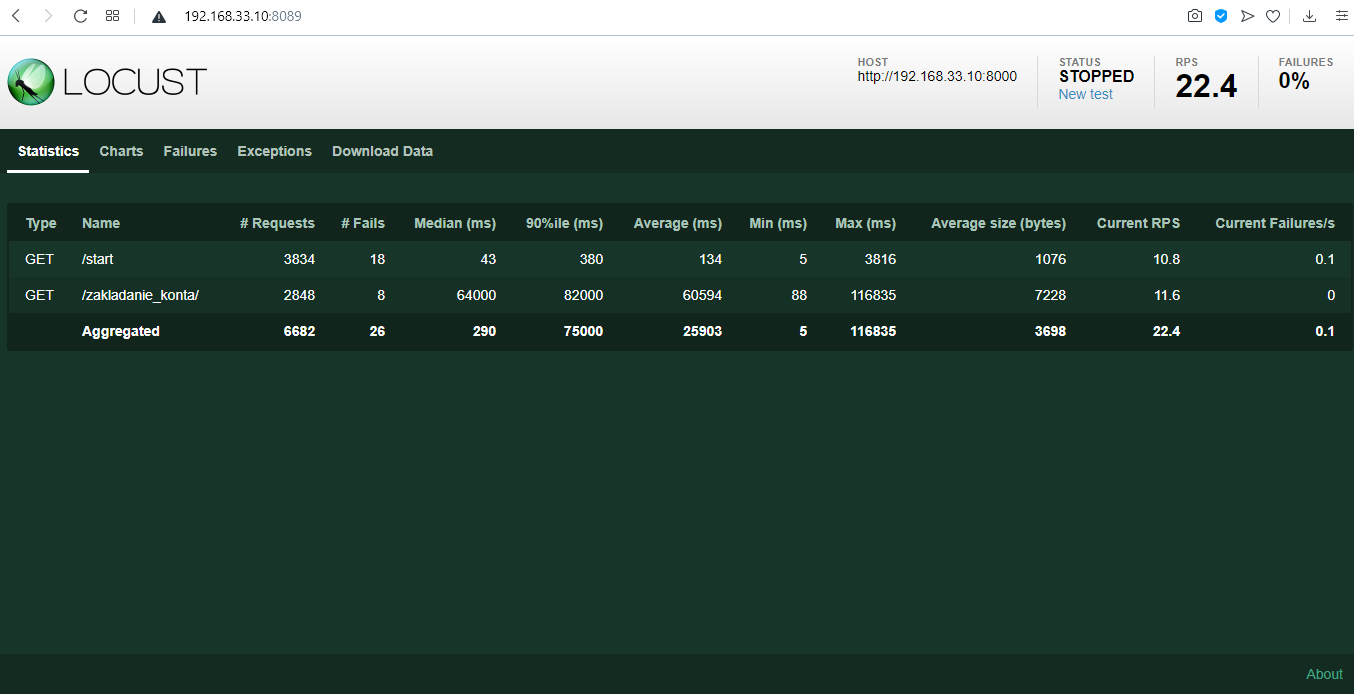
Rys Nr. 18 Wiresshark - podsłuchiwanie

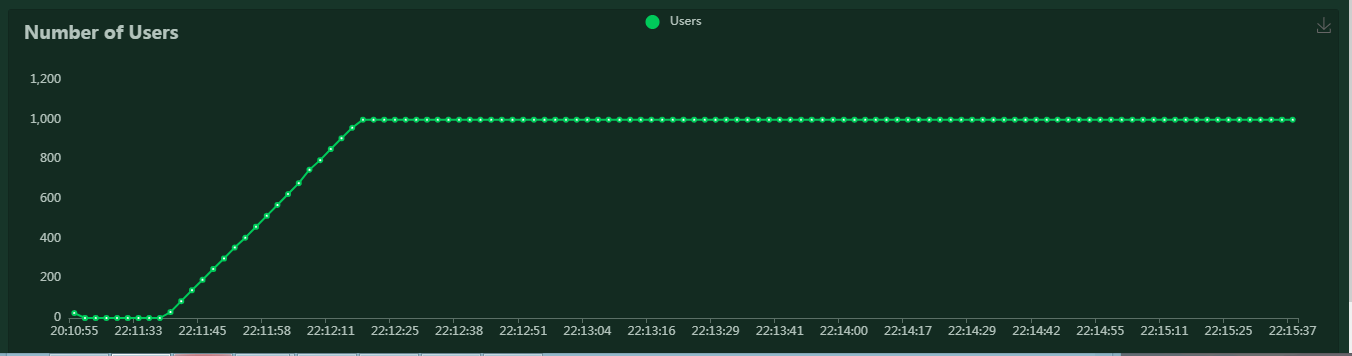
Powiększając istotną część tego zrzutu ekranu :

Rys Nr. 18 Wyłuskane dane

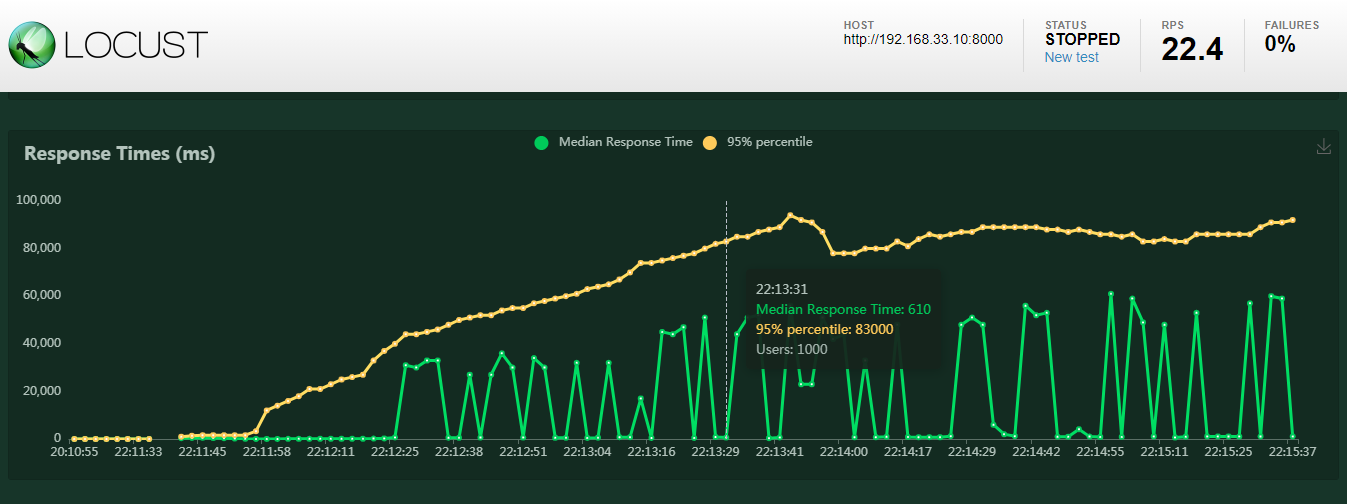
Nazwa użytkownika oraz hasło przekazywane były zwykłym teksem. Test odbywał się w sieci host-only, a więc bez dostępu na świat zewnętrzny. Gdyby strona została upubliczniona, wymagało by to dodatkowego zabezpieczenia. Protokół http powinien zostać zastąpiony protokołem https, który szyfruje połączenie. W tym przypadku można by skorzystać z narzędzia jakim jest snapcraft. Jest to darmowe narzędzie pozwalające skonfigurować połączenie sieciowe dla domeny i zabezpieczyć je.

Wykonane zostały również testy obciążeniowe strony. Do przeprowadzenia ich wykorzystałem narzędzie Locust. Wyniki uzależnione są od zasobów, jakie przeznaczone zostały na maszynę wirtualną. Poniższy test został przeprowadzony dla 1000 jednocześnie wysyłających zapytania użytkowników. Ilość nowych użytkowników na sekundę został ustawiony na 28.0. Maszyna wirtualna miała do dyspozycji 4 GB ramu oraz mogła skorzystać z 4 rdzeni procesora (każdy rdzeń 2,20GHz).

Rys Nr. 19 Locust testy wydajności



Rys Nr. 20 Ilość użytkowników

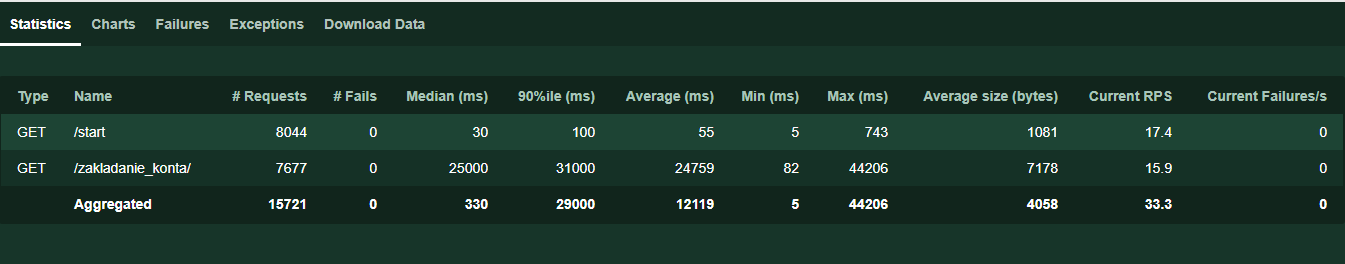
Rys Nr. 21 Locust czas odpowiedzi.

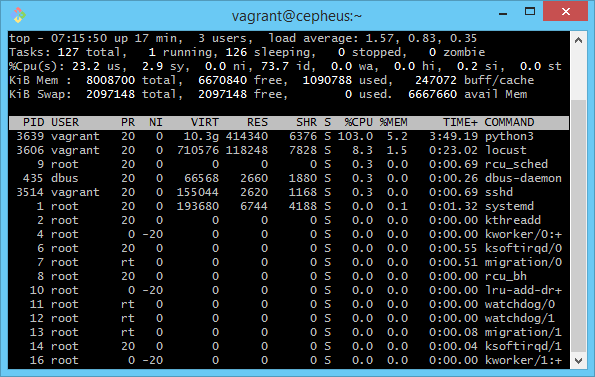
Rys Nr. 22 Locust zapytania na sekundę.

Jak widać strona działała stabilnie, lecz czas odpowiedzi serwera był długi.

Po zwiększeniu zasobów pamięci ram wykresy wyszły identycznie. Analizując wyniki zużycia zasobów na serwerze linux odkryłem, że głównym czynnikiem opóźniającym stronę jest procesor. Pamięć ram mimo zwiększenia nie wpłynęła na działanie strony.

Podsumowując, wydajność strony zależy bezpośrednio od parametrów serwera udostępniającego stronę. Maszyna mająca do dyspozycji procesor o mocy 2,2 GHz i 4 wątkach jest w stanie pracować przy obciążeniu 1000 użytkowników bez przerwania pracy serwera. Niestety, nie jest to komfortowe korzystanie ze strony, ponieważ czas odpowiedzi jest niedopuszczalnie długi. Przy obciążeniu rzędu 400 użytkowników strona pracuje dość wolno, ale jest w stanie spełnić swoje zadanie.

Rys Nr. 23 Locust test wydajności 2



Rys Nr. 24 Zużycie zasobów Centos

# 6. Podsumowanie

Stworzony projekt jest świetnym narzędziem do zbierania opinani publicznej na temat wprowadzanych ustaw. Strona umożliwia głosowanie nad każdą wprowadzoną ustawą. Wszystkie założenia jakie zostały postawione zostały zrealizowane, za wyjątkiem autoryzacji. Był to największy problem do zrealizowania ze względu na ograniczony dostęp do narzędzi. Gdyby strona otrzymała akredytację rządu, można by znacznie podnieść poziom jej zabezpieczeń oraz uzyskać pewność że głosowania są w pełni uczciwe. Analizując badania, bardzo istotna jest kwestia odpowiedniego zabezpieczenia strony przed wyciekami danych i podsłuchami. Znaczenie ma również poziom zasobów serwera udostępniającego stronę użytkownikom.

Po wykonaniu całej pracy i przeanalizowaniu problemu nasuwa mi się jedno pytanie. Czy gdyby stworzone narzędzie stało się faktycznym narzędziem używanym przez wszystkich obywateli, rozwiązywało by problemy i pomagało nam, czy wręcz przeciwnie?

Spis Treści

[1. Wstęp i analiza problemu 1](#_Toc71878532)

[2. Istniejące rozwiązania 2](#_Toc71878533)

[3.Użyte narzędzia 3](#_Toc71878534)

[4.Opis projektu 7](#_Toc71878535)

[4.1 Specyfikacja wymagań od strony użytkownika: 7](#_Toc71878536)

[4.2 Architektura projektu 9](#_Toc71878537)

[4.3 Koncepcja przechowywania danych 10](#_Toc71878538)

[4.4 Projekt interfejsu użytkownika 12](#_Toc71878539)

[4.5 Dokumentacja techniczna 12](#_Toc71878540)

[4.5.2 Konfiguracja i import danych do bazy. 14](#_Toc71878541)

[4.5.3 Obsługa środowiska po konfiguracji 15](#_Toc71878542)

[4.5.4 Projekt Django 17](#_Toc71878543)

[4.5.4.1 Strona startowa 17](#_Toc71878544)

[4.5.4.2 Tworzenie konta 18](#_Toc71878545)

[4.5.4.3 Weryfikacja konta. 19](#_Toc71878546)

[4.5.4.4 Strona główna i podstrony 20](#_Toc71878547)

[4.5.4.5 Przegląd ustaw. 21](#_Toc71878548)

[4.5.4.6 Głosowanie 22](#_Toc71878549)

[4.5.4.7 Sprawdzanie wyników 23](#_Toc71878550)

[4.6 Część programistyczna projektu Django 23](#_Toc71878551)

[4.6.1 Forms.py 24](#_Toc71878552)

[4.6.2 Models.py 24](#_Toc71878553)

[4.6.3 Settings.py 24](#_Toc71878554)

[4.6.4 Urls.py 25](#_Toc71878555)

[4.6.5 Views.py 25](#_Toc71878556)

[4.7 Opis skryptów pobierających dane ze strony isap.sejm.gov.pl i importujących je do bazy danych 26](#_Toc71878557)

[4.8 Bezpieczeństwo. 26](#_Toc71878558)

[5. Testy 28](#_Toc71878559)

[6. Podsumowanie 33](#_Toc71878560)

Źródła:

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

<https://docs.djangoproject.com/en/3.2/>

<https://www.w3schools.com>

<https://isap.sejm.gov.pl>

<https://docs.centos.org/en-US/docs/>

<https://docs.python.org/3/>