feder Documentation

Release 0.1

Adam Dobrawy

Apr 29, 2024

SPIS TREŚCI

CHAPTER

ONE

WPROWADZENIE

1.1 Koncepcja aplikacji

Do najważniejszych problemów debaty publicznej niewątpliwie zalicza się częste opieranie argumentacji na niewerifikowalnych lub nieaktualnych danych.

Informacje posiadane przez organy administracji publicznej są skarbnicą wiedzy. Ich skuteczne wykorzystanie pozwoliłoby rozwiać wiele pojawiających się wątpliwości. Niestety, forma ich przechowywania i udostępniania często uniemożliwia łatwe ich odczytanie oraz intepretację, w szczególności przy użyciu technik cyfrowych.

Obywatelskie fedrowanie danych jest projektem powstałym w odpowiedzi na to wyzwanie. Ma na celu usystematyzowanie dużej liczby informacji spływających od instytucji publicznych oraz zapisanie ich w formie łatwej do analizy, zarówno przez człowieka, jak i maszynę. To obywatele-wolontariusze, którym zależy na jakości życia publicznego w Polsce, czytają udzielone przez organy odpowiedzi i uzupełniają bazę danych poprzez wykorzystanie intuicyjnego mechanizmu ankiet. Zebrane i uporządkowane dane służą następnie do przeprowadzania badań, które to przyczyniają się do wzrostu świadomości społecznej i poprawy jakości rządzenia.

1.1.1 Cele społeczne

- Szybka i zbiorowa analiza danych nieistniejących w żadnych repozytoriach i nie istniejących w formie możliwej do odczytu maszynowego. Do wykorzystywania w kampaniach rzeczniczych lub jako dowody w systemie tworzenia prawa.
- Masowa edukacja obywatelska na temat tego, jakie informacje można uzyskać od władz i jak wyglądają ich odpowiedzi oraz korespondencja z nimi, a także jak sprawdzać wypowiedzi pojawiające się w debacie publicznej.

1.1.2 Cele dodatkowe

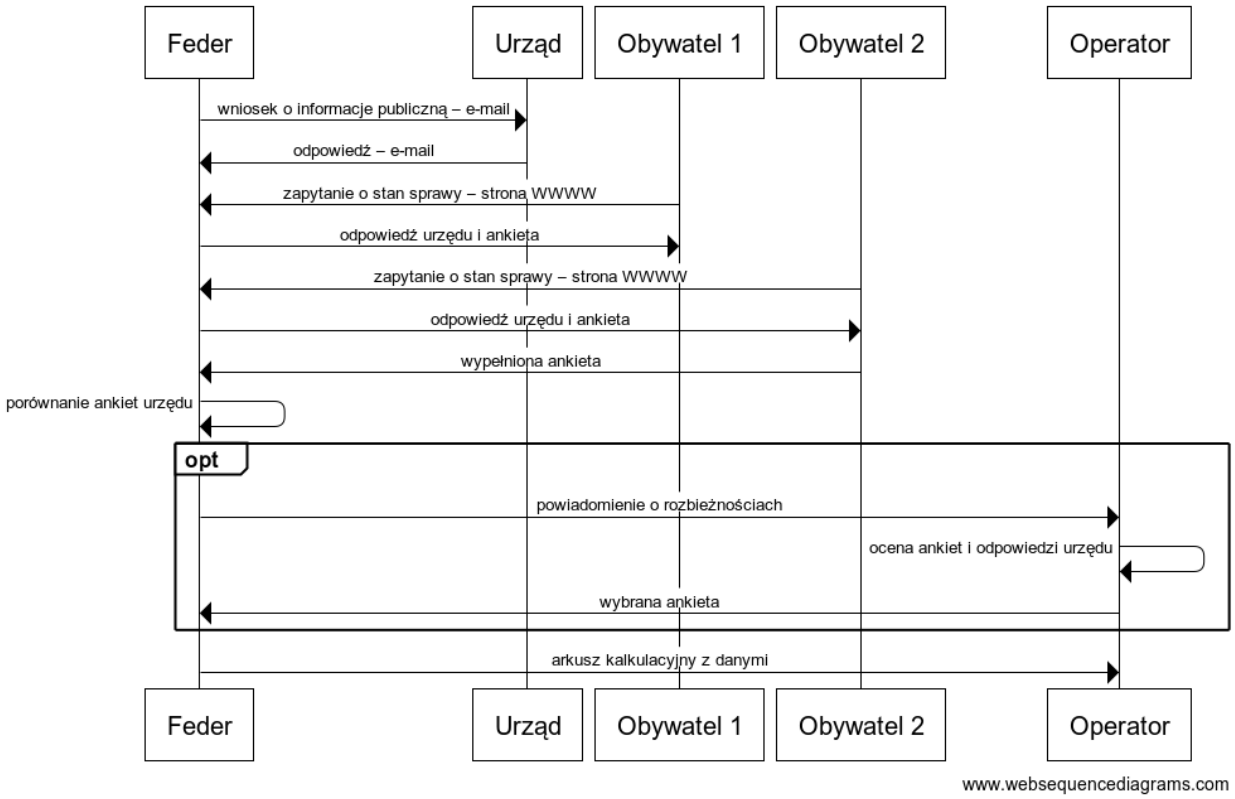
Aplikacja uniwersalna, do przystosowania dla różnych akcji przez różne organizacje.

1.1.3 Zasada działania

W pierwszej kolejności użycie aplikacji wymaga dostarczenia danych adresowych instytucji, co zapewnia moduł *Instytucje*. Po wypełnieniu bazy adresowej wymagane jest utworzenie monitoringu (zob. *Monitoringi*), co zapewnia jednocześnie utworzenie spraw (zob. *Sprawy*), a także wysłanie stworzenie w nich pierwszych listów (zob. *Listy*).

Wysyłka listów systemu odbywa się z wykorzystaniem poczty elektronicznej. Zazwyczaj jeden wzórw listu jest kierowany do kilkudziesięciu podmiotów. W odpowiedzi urząd ma obowiązek udzielenia nam odpowiedzi w formie i sposób określony w wniosku - na dedykowany adres e-mail. Wiadomośći są interpretowane i pobierane do systemu przez moduł *Listy*, który zapewnia również przypisanie korespondencji do danej sprawy (zob. *Sprawy*), która związana jest z konkretnym urzędem.

Po zarejestrowaniu odpowiedniej ilości odpowiedzi możliwe konieczne jest utworzenie w monitoringu stworzenia kwestionariusza (zob. questionaries), a następnie na jego podstawie w licznych sprawach utworzenie zadań (zob. tasks). Wówczas dane są przetwarznae przez użytkowników, co zostało przedstawione na diagramie:




```
Feder->Urząd: wniosek o informacje publiczną - e-mail
Urząd->Feder: odpowiedź - e-mail
Obywatel 1->Feder: zapytanie o stan sprawy - strona WWWW
Feder->Obywatel 1: odpowiedź urzędu i ankieta
Obywatel 2->Feder: zapytanie o stan sprawy - strona WWWW
Feder->Obywatel 2: odpowiedź urzędu i ankieta
Obywatel 2->Feder: wypełniona ankieta
Feder->Feder: porównanie ankiet urzędu

opt
    Feder->Operator: powiadomienie o rozbieżnościach
    Operator->Operator: ocena ankiet i odpowiedzi urzędu
    Operator->Feder: wybrana ankieta
end

Feder->Operator: arkusz kalkulacyjny z danymi
```

System wyposażony winien być w mechanizm weryfikacji rozbieżności w ankietach, gdyby dochodziło do rozbieżnych interpretacji udzielonych odpowiedzi. Wówczas operator dokonuje wyboru właściwej ankiety, albo zgłasza odpowiedzi własne.

Ankiety związane z danym kwestionariuszem mogą być wyeksportowane i analizowane z wykorzystaniem właściwych narzędzi.

1.1.4 Przykłady zastosowań

Przepisywanie skróconych informacji i dostarczanie danych liczbowych

W 2012 Sieć Obywatelska Watchdog Polska włączyła się w kampanię przeciw zmianom w ustawie o zgromadzeniach. Zmiany wprowadzano pod wpływem zamieszek przy okazji Dnia Niepodległości w Warszawie. Miały one zwiększać kontrolę i de facto nakładać duże obowiązki na organizatorów zgromadzeń. Celem zbierania danych było uzyskanie informacji możliwych do pokazania parlamentarzystom, o tym że zmiany które chcą wprowadzić dotkną także organizatorów zgromadzeń w małych miejscowościach. Przekaz miał uświadomić, że zmian prawa nie można dokonywać bez widzenia całości obszaru, którego będą one dotyczyły oraz że zmiany mogą zamrozić i tak niewielką aktywność obywatelską.

Aby dowiedzieć się jak wglądają zgromadzenia w małych miejscowościach (duże często mają rejestr w formie możliwej do odczytu w BIPie), Sieć Obywatelska Watchdog Polska wysłała wniosek o informację do wybranych urzędów gmin o:.

- skany wszystkich wniosków zgłaszających zgromadzenie za lata 2010-2012
- skany ewentualnych decyzji odmawiających zgłaszanie zgromadzenia za lata 2010-2012

Z otrzymanych odpowiedzi można było uzyskać głównie dane jakościowe: * jakie podmioty zgłaszają zgromadzenia (czy są to osoby indywidualne, związki zawodowe, kościoły, organizacje) * w jakiej sprawie są te zgromadzenia organizowane oraz dane ilościowe * ile rocznie zgłasza się zgromadzeń (zwłaszcza w mniejszych miejscowościach)

Dostarczanie danych liczbowych

W 2012 roku zwiększyła się nieco aktywność obywatelska w zakresie wnioskowania o informację. Był to wynik błędów rządu przy nowelizacji ustawy o dostępie do informacji publicznej i dużego nagłośnienia medialnego. Częściowo zapewne także wynik aktywności Sieci Obywatelskiej Watchdog Polska i innych organizacji. Nie bez znaczenia jest, że firmy zbierające dane, nauczyły się korzystać z prawa do informacji, co szczególnie oburza urzędników. Lobbing urzędników był i jest na tyle skuteczny, że coraz więcej szanowanych osób zabierających głos w debacie publicznej powtarza sformułowanie o „nadużywaniu prawa do informacji" Ponieważ może to skutkować realnymi zmianami w prawie, Sieć Obywatelska Watchdog Polska wysłała do wszystkich urzędów gmin (2500) wniosek, który miał zweryfikować jaki jest faktyczny stan wnioskowania i zbadać jakie dane są w ogóle dostępne. Wyniki pokazały, że realny poziom wnioskowania jest bardzo niski – od kilku do kilkudziesięciu wniosków rocznie (poza największymi miastami i ekstremalnymi sytuacjami), a wzrost pomiędzy 2011 i 2012 roku jest znikomy.

Aby uzyskać te informacje, Sieć Obywatelka Watchdog Polska zadała następujące pytania:

1. Ile wniosków o informację publiczną otrzymał urząd w 2011 roku
2. Ile wniosków o informację publiczną otrzymał urząd w 2012 roku
3. Udostępnienie ewidencji wniosków o informację publiczną za 2011 rok. Jeżeli ewidencja prowadzona jest w formie elektronicznej, żądamy udostępnienia w postaci pliku w formacie dokumentu tekstowego lub arkusza kalkulacyjnego. Jeżeli ewidencja/rejestr nie jest prowadzony w formie elektronicznej, wnosimy o udostępnienie informacji w postaci skanu, z dokonaniem niezbędnych wyłączeń dotyczących ochrony prywatności wnioskujących osób.
4. Udostępnienie ewidencji wniosków o informację publiczną za 2012 rok. Jeżeli ewidencja prowadzona jest w formie elektronicznej, żądamy udostępnienia w postaci pliku w formacie dokumentu tekstowego lub arkusza kalkulacyjnego. Jeżeli ewidencja/rejestr nie jest prowadzony w formie elektronicznej, wnosimy o udostępnienie informacji w postaci skanu, z dokonaniem niezbędnych wyłączeń dotyczących ochrony prywatności wnioskujących osób.

Dane, które można uzyskać dzięki masowej analizie obywatelskiej to:

- Ile wniosków wpłynęło w 2011 roku? LICZBA

- Ile wniosków wpłynęło w 2012 roku? LICZBA
- Czy załączona została ewidencja wniosków za 2011 rok? TAK/NIE
- Czy załączona została ewidencja wniosków za 2012 rok? TAK/NIE
- Kiedy wniosek został zrealizowany? FORMAT DATY
- Czy urząd twierdzi, że żądanie dotyczy informacji przetworzonej? odhaczenie jeśli tak
- Czy za przygotowanie informacji zażądano opłaty/sugerowano opłatę? odhaczenie jeśli tak
- Czy napisano, że konieczne jest przedłużenie czasu potrzebnego na odpowiedź? odhaczenie jeśli tak
- Czy w tej gminie wystąpiła sytuacja braku ewidencji, ale w zamian pojawiły się skany wniosków? odhaczenie jeśli tak
- Czy w tej gminie wystąpiła sytuacja braku ewidencji, ale w zamian w odpowiedzi pojawił się opis złożonych wniosków? odhaczenie jeśli tak

1.2 Architektura

Aplikacja została wykonana zaimplentowana w języku Python 2.7 z wsparciem frameworku Django 1.10. Została zaprojektowania do wykorzystania bazy danych MariaDB.

Zestawienie bibliotek Python wykorzystanych w projekcie:

```
# Production and staging Django
Django==4.2.11

# Database adapter
mysqlclient==2.2.4

# Configuration
django-environ==0.11.2

# Forms
django-braces==1.15.0
django-crispy-forms==1.14.0

# Models
django-model-utils==4.5.0

# Images
Pillow==10.3.0

# For user registration, either via email or social
# Well-built with regular release cycles!
#django-allauth[socialaccount]==0.62.1
# version 0.62 requires config change
django-allauth==0.61.1

# Unicode slugification
unicode-slugify==0.1.5
django-autoslug==1.9.9
```

(continues on next page)

(continued from previous page)

```
# Time zones support
pytz==2024.1

# Commands
lxml==5.2.1


# Your custom requirements go here
cryptography==42.0.5
django-filter==24.2
django-autocomplete-light==3.11.0
# django-ajax-datatable to be installed from github fork until
# https://github.com/morlandi/django-ajax-datatable/pull/111 is merged
# django-ajax-datatable==4.4.5
git+https://github.com/PiotrIw/django-ajax-datatable.git@05afe42

django-tinymce==3.7.1

# replacing unmaintained django-atom with updated fork
# django-atom==0.16.3
git+https://github.com/PiotrIw/django-atom.git@master#egg=django-atom

# django-tinycontent is not supported anymore so watchdog maintained fork is used
git+https://github.com/watchdogpolska/django-tinycontent.git@master

django-formtools==2.5.1
django-mptt==0.16.0
jsonfield==3.1.0
django-guardian==2.4.0
django-teryt-tree==0.18.4
cached-property==1.5.2

# replacing unmaintained django-bootstrap-pagination with updated fork
# django-bootstrap-pagination==1.7.1
git+https://github.com/PiotrIw/django-bootstrap-pagination.git@master#egg=django-
↪bootstrap-pagination

django-reversion==5.0.12
djangorestframework==3.15.1
djangorestframework-csv==3.0.2
unicodecsv==0.14.1
tqdm==4.66.2
django-github-revision==0.0.3
django-extensions==3.2.3
django-cleanup==8.1.0

# Ping commit due missing cached object in <=4.7.1
django-extra-views==0.13.0
# django-extra-views==0.14.0 -> py3.10 or dj3.2: cannot import name 'InlineFormSet' from
↪'extra_views'
django-sendfile2==0.7.1
virustotal-api==1.1.11
```

(continues on next page)

(continued from previous page)

```
https://github.com/ross/performant-pagination/archive/
↪5b537da95728d622792031071ecc4cb5154ec86f.zip
# not available on pypi - see https://github.com/ross/performant-pagination/issues/5

django4-background-tasks==1.2.9
django-cors-headers==4.3.1

django-rosetta==0.10.0
bleach==6.1.0
beautifulsoup4==4.12.3
html2text==2024.2.26
openpyxl==3.1.2

# LLM exvaluation
langchain==0.1.16
#langchain[llms]==0.1.5
openai==1.23.6
langchain-openai==0.1.4
tiktoken==0.6.0
```

Ponadto podczas pracy deweloperskiej są wykorzystane następujące biblioteki:

```
# Local development dependencies go here
-r base.txt
Sphinx==7.3.7
sphinx-rtd-theme==2.0.0
Werkzeug==3.0.2
django-debug-toolbar==4.3.0
ipdb==0.13.13
factory-boy==3.3.0
django-coverage-plugin==3.1.0
vcrpy==6.0.1
sphinxcontrib-programoutput==0.17
pre-commit==3.7.0
coveralls==3.3.1
```

1.3 Panel administracyjny

Dostęp do panelu administracyjnego, na których odbywać się będzie zarządzanie wszystkimi zasobami portalu jest tylko możliwy po autoryzacji i wyłącznie dla konkretnych osób. Tworzenie kont administracyjnych jest możliwe wyłącznie z poziomu administracyjnego, to znaczy, że konto administracyjne może założyć osoba zalogowana do panelu. Oprogramowanie portalu zapewnia rejestrowaną i skuteczną kontrolę dostępu.

CHAPTER

TWO

ADMINISTRACJA

2.1 Uruchomienie w środowisku deweloperskim

W celu uruchomienia aplikacji wykonaj:

```
$ make build
$ docker-compose up
```

2.2 Dane testowe

W celu szybkiego rozruchu aplikacji możliwe jest wygenerowanie lub wczytanie pewnych danych początkowych. Szczegółowe instrukcje zostały przedstawione w modułach właściwych modułów.

Zaleca się jednak następującą sekwencje:

1. *Jednostki podziału terytorialnego*
2. *Monitoringi*
3. *Instytucje*
4. *Sprawy*

Wykorzystywane w aplikacj są jednak także moduły, które nie wspierają automatycznego generowania swojej treści ze względu na wykorzystywanie złożonej strukturys danych. Ich wypełnienie danymi jest możliwe z poziomu interfejsu użytkownika. Każdy moduł jednak zawiera submoduł `fixtures`, który może stanowić źródło wiedzy o pożądanej strukturze.

CHAPTER

THREE

MODUŁY

3.1 Powiadomienia

3.1.1 Założenia

Moduł stanowi komponent powiadomień do operatora o konieczności podjęcia akcji w systemie. Takie powiadomienia mogą być kierowane m. in. w związku z zgłoszeniem spamu.

3.1.2 Dane testowe

Dla modułu nie możliwe jest w środowisku deweloperskim dynamicznie wygenerowanie generowanych danych testowych.

Todo: Opracować generowanie danych testowych.

3.1.3 Architektura

Model

class feder.alerts.models.**Alert**(*id*, *created*, *modified*, *monitoring*, *reason*, *author*, *solver*, *status*, *content_type*, *object_id*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Utworzony
- **modified** (*AutoLastModifiedField*) – Zmodyfikowany
- **monitoring_id** (ForeignKey to *feder.monitorings.models.Monitoring*) – Monitoring
- **reason** (*TextField*) – Przyczyna
- **author_id** (ForeignKey to feder.users.models.User) – Autor
- **solver_id** (ForeignKey to feder.users.models.User) – Rozwiązywacz/-ka
- **status** (*BooleanField*) – Status

- **content_type_id** (ForeignKey to django.contrib.contenttypes.models.ContentType) – Content type
- **object_id** (*PositiveIntegerField*) – Object id

exception DoesNotExist

exception MultipleObjectsReturned

class feder.alerts.models.**AlertQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Widoki

class feder.alerts.views.**AlertCreateView**(***kwargs*)

Parameters
url_name – alerts:create

form_class

alias of AlertForm

get_context_data(***kwargs*)

Insert the form into the context dict.

get_form_kwargs()

Return the keyword arguments for instantiating the form.

model

alias of *Alert*

class feder.alerts.views.**AlertDeleteView**(**args*, ***kwargs*)

Parameters
url_name – alerts:delete

get_success_url()

Return the URL to redirect to after processing a valid form.

model

alias of *Alert*

class feder.alerts.views.**AlertDetailView**(***kwargs*)

Parameters
url_name – alerts:details

model

alias of *Alert*

class feder.alerts.views.**AlertListView**(***kwargs*)

Parameters
url_name – alerts:list

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

model

alias of *Alert*

class feder.alerts.views.**AlertStatusView**(***kwargs*)

Parameters
url_name – alerts:status

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

class feder.alerts.views.**AlertUpdateView**(***kwargs*)

Parameters
url_name – alerts:update

form_class

alias of AlertForm

get_form_valid_message()

Validate that form_valid_message is set and is either a unicode or str object.

model

alias of *Alert*

3.2 Sprawy

3.2.1 Założenia

Moduł odpowiedzialny jest za mechanizm "wątków" odnoszących się do konkretnego zapytania skierowanego do konkretnego urzędu. Każda sprawa jest związana tylko z jednym monitoringiem i jednym zapytaniem. W obrębie sprawy mogą być agregowane informacje różnej kategorii.

3.2.2 Architektura

Model

class feder.cases.models.**Alias**(*id*, *case*, *email*)

Parameters

- **id** (*AutoField*) – Id
- **case_id** (ForeignKey to *feder.cases.models.Case*) – Sprawa
- **email** (*CharField*) – Email

exception DoesNotExist

exception MultipleObjectsReturned

class feder.cases.models.**Case**(*id*, *created*, *modified*, *name*, *slug*, *user*, *monitoring*, *institution*, *mass_assign*, *email*, *confirmation_received*, *response_received*, *is_quarantined*, *first_request*, *last_request*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Utworzony
- **modified** (*AutoLastModifiedField*) – Zmodyfikowany
- **name** (*CharField*) – Nazwa
- **slug** (*AutoSlugField*) – Adres url
- **user_id** (ForeignKey to feder.users.models.User) – User
- **monitoring_id** (ForeignKey to *feder.monitorings.models.Monitoring*) – Monitoring
- **institution_id** (ForeignKey to *feder.institutions.models.Institution*) – Instytucja
- **mass_assign** (*UUIDField*) – Mass assign id
- **email** (*CharField*) – Email
- **confirmation_received** (*BooleanField*) – Otrzymano potwierdzenie
- **response_received** (*BooleanField*) – Otrzymano odpowiedź
- **is_quarantined** (*BooleanField*) – Poddany kwarantannie
- **first_request_id** (ForeignKey to *feder.letters.models.Letter*) – Pierwszy wniosek
- **last_request_id** (ForeignKey to *feder.letters.models.Letter*) – Ostatni wniosek

exception **DoesNotExist**

exception **MultipleObjectsReturned**

class feder.cases.models.**CaseQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

get_mass_assign_uid()

Returns random UUID identifier ensuring it's unique.

class feder.cases.models.**GroupConcat**(**args*, ***kwargs*)

Widoki

class feder.cases.views.**CaseAutocomplete**(***kwargs*)

Parameters

url_name – cases:autocomplete

get_queryset()

Filter the queryset with GET['q'].

class feder.cases.views.**CaseCreateView**(***kwargs*)

Parameters
url_name – cases:create

form_class

alias of CaseForm

get_context_data(***kwargs*)

Insert the form into the context dict.

get_form_kwargs()

Return the keyword arguments for instantiating the form.

model

alias of *Case*

class feder.cases.views.**CaseDeleteView**(**args*, ***kwargs*)

Parameters
url_name – cases:delete

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Case*

class feder.cases.views.**CaseDetailView**(***kwargs*)

Parameters
url_name – cases:details

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Case*

class feder.cases.views.**CaseFindAutocomplete**(***kwargs*)

Parameters
url_name – cases:autocomplete-find

get_queryset()

Filter the queryset with GET['q'].

get_result_label(*result*)

Return the label of a result.

class feder.cases.views.**CaseListView**(***kwargs*)

Parameters
url_name – cases:list

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

model

alias of *Case*

class feder.cases.views.**CaseUpdateView**(***kwargs*)

Parameters
url_name – cases:update

form_class

alias of CaseForm

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Case*

3.3 Domeny

3.3.1 Założenia

Moduł stanowi mechanizm zarządzania domenami, które następnie mogą być wykorzystywane jako źródło dla *Monitoringu*.

3.3.2 Architektura

Model

class feder.domains.models.**Domain**(*id*, *created*, *modified*, *name*, *active*, *organisation*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Utworzony
- **modified** (*AutoLastModifiedField*) – Zmodyfikowany
- **name** (*CharField*) – Nazwa
- **active** (*BooleanField*) – Status aktywności
- **organisation_id** (ForeignKey to feder.organisations.models.Organisation) – Organizacja

exception DoesNotExist

exception MultipleObjectsReturned

class feder.domains.models.**DomainQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Panel administracyjny

class feder.domains.admin.**DomainAdmin**(*model*, *admin_site*)

actions = None

Widoki

3.4 Instytucje

3.4.1 Założenia

Moduł stanowi mechanizm gromadzenia danych adresowych o instytucjach i przedstawienie spraw w jakich dany urząd jest zaaganżowany.

3.4.2 Architektura

Model

class feder.institutions.models.**Institution**(*id*, *created*, *modified*, *name*, *slug*, *jst*, *regon*, *extra*, *email*, *archival*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Utworzony
- **modified** (*AutoLastModifiedField*) – Zmodyfikowany
- **name** (*CharField*) – Nazwa
- **slug** (*AutoSlugField*) – Adres url
- **jst_id** (ForeignKey to *feder.teryt.models.JST*) – Jednostka podziału terytorialnego
- **regon** (*CharField*) – Kod regon
- **extra** (*JSONField*) – Unorganized additional information
- **email** (*EmailField*) – Adres e-mail instytucji
- **archival** (*BooleanField*) – Instytucja archiwalna nie może być przypisana do monitorowania lub masowej wysyłki.

exception DoesNotExist

exception MultipleObjectsReturned

get_jst_tree()

Returns voivodeship JST list in order: voivodeship, county, community by searching over parents of related JST. This operation may be expensive, so in queries it should be used with with_jst manager method.

class feder.institutions.models.**InstitutionQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

class feder.institutions.models.**Tag**(*id*, *name*, *slug*)

Parameters

- **id** (*AutoField*) – Id
- **name** (*CharField*) – Nazwa
- **slug** (*AutoSlugField*) – Adres url

exception DoesNotExist

exception MultipleObjectsReturned

class feder.institutions.models.**TagQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Panel administracyjny

class feder.institutions.admin.**InstitutionAdmin**(**args*, ***kwargs*)

Admin View for Institution

get_actions(*request*)

Return a dictionary mapping the names of all actions for this ModelAdmin to a tuple of (callable, name, description) for each action.

class feder.institutions.admin.**TagAdmin**(*model*, *admin_site*)

actions = None

get_queryset(**args*, ***kwargs*)

Return a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

Widoki

class feder.institutions.views.**InstitutionAutocomplete**(***kwargs*)

Parameters
url_name – institutions:autocomplete

get_queryset()

Filter the queryset with GET['q'].

get_result_label(*result*)

Return the label of a result.

class feder.institutions.views.**InstitutionCreateView**(***kwargs*)

Parameters
url_name – institutions:create

form_class

alias of InstitutionForm

model

alias of *Institution*

class feder.institutions.views.**InstitutionDeleteView**(**args*, ***kwargs*)

Parameters
url_name – institutions:delete

model

alias of *Institution*

class feder.institutions.views.**InstitutionDetailView**(***kwargs*)

Parameters

- **url_name** – institutions:details
- **url_name** – institutions:details

static get_object_list(*obj*)

A method to return object list to additional list. This should be overriden.

Parameters
obj – The object the view is displaying.

Returns
A list of object to paginated

Return type
QuerySet

Raises
ImproperlyConfigured – The method was not overrided.

model

alias of *Institution*

class feder.institutions.views.**InstitutionListView**(***kwargs*)

Parameters
url_name – institutions:list

get_context_data(**args*, ***kwargs*)

Get the context for this view.

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

model

alias of *Institution*

class feder.institutions.views.**InstitutionUpdateView**(***kwargs*)

Parameters
url_name – institutions:update

form_class

alias of InstitutionForm

model

alias of *Institution*

class feder.institutions.views.**TagAutocomplete**(***kwargs*)

Parameters
url_name – institutions:tag_autocomplete

get_queryset()

Filter the queryset with GET['q'].

get_result_label(*result*)

Return the label of a result.

3.5 Listy

3.5.1 Założenia

Moduł odpowiedzialny za indywidualny komunikat wymieniony pomiedzy systemem, a urzędem. Zapewnia odbiór korespondencji w formie e-mailowej wraz z załącznikami i jej publikacji.

Odbiór korespondencji w formie e-mailowej realizowany jest z wsparciem aplikacji imap-to-webhook .

3.5.2 Dane testowe

Dla modułu nie możliwe jest w środowisku deweloperskim dynamicznie wygenerowanie generowanych danych testowych.

Todo: Opracować generowanie danych testowych.

3.5.3 Architektura

Model

class feder.letters.models.**Attachment**(*id*, *attachment*, *letter*, *text_content*, *text_content_update_result*)

Parameters

- **id** (*AutoField*) – Id
- **attachment** (*FileField*) – Plik
- **letter_id** (ForeignKey to *feder.letters.models.Letter*) – Letter
- **text_content** (*TextField*) – Treść tekstowa
- **text_content_update_result** (*TextField*) – Wynik aktualizacji zawartości tekstowej

exception DoesNotExist

exception MultipleObjectsReturned

class feder.letters.models.**AttachmentQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

class feder.letters.models.**Letter**(*id*, *created*, *modified*, *record*, *author_user*, *author_institution*, *title*, *body*, *html_body*, *quote*, *html_quote*, *email*, *email_from*, *email_to*, *note*, *ai_evaluation*, *normalized_response*, *is_spam*, *is_draft*, *message_type*, *mark_spam_by*, *mark_spam_at*, *message_id_header*, *eml*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*CreationDateTimeField*) – Utworzony
- **modified** (*ModificationDateTimeField*) – Zmodyfikowany
- **record_id** (OneToOneField to feder.records.models.Record) – Record
- **author_user_id** (ForeignKey to feder.users.models.User) – Autor (jeżeli użytkownik)
- **author_institution_id** (ForeignKey to *feder.institutions.models.Institution*) – Autor (jeżeli instytucja)
- **title** (*CharField*) – Temat
- **body** (*TextField*) – Treść
- **html_body** (*TextField*) – Treść w formacie html
- **quote** (*TextField*) – Cytat
- **html_quote** (*TextField*) – Cytat w formacie html
- **email** (*EmailField*) – E-mail
- **email_from** (*EmailField*) – Z adresu e-mail
- **email_to** (*EmailField*) – Na adres e-mail
- **note** (*TextField*) – Komentarz od redakcji
- **ai_evaluation** (*TextField*) – Ocena ai listu
- **normalized_response** (*JSONField*) – Znormalizowana odpowiedź na monitoring
- **is_spam** (*IntegerField*) – Oznaczony jako spam?
- **is_draft** (*BooleanField*) – Czy szkic?
- **message_type** (*IntegerField*) – Typ wiadomości
- **mark_spam_by_id** (ForeignKey to feder.users.models.User) – Osoba, która oznaczyła jako spam
- **mark_spam_at** (*DateTimeField*) – Czas kiedy list został oznaczony jako spam
- **message_id_header** (*CharField*) – Id wysłanej wiadomości e-mail “message-id”
- **eml** (*FileField*) – Plik

exception DoesNotExist

exception MultipleObjectsReturned

property allowed_recipient

Returns True if any of the recipients from Letter.get_recipients email domain is in monitoring domains.

generate_mass_letters()

Uses this letter as a template for generating mass message
(it has to be defined with "mass draft" message type). prepares and returns generated letters ready for sending.

get_recipients()

Returns a list of all email addresses from the "To" and "Cc" fields of the Letter eml file.

class feder.letters.models.**LetterEmailDomain**(*id*, *created*, *modified*, *domain_name*, *is_trusted_domain*, *is_monitoring_email_to_domain*, *is_spammer_domain*, *is_non_spammer_domain*, *email_to_count*, *email_from_count*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*CreationDateTimeField*) – Utworzony
- **modified** (*ModificationDateTimeField*) – Zmodyfikowany
- **domain_name** (*CharField*) – Domena adresu e-mail
- **is_trusted_domain** (*BooleanField*) – Czy zaufana (własna lub partnerska)?
- **is_monitoring_email_to_domain** (*BooleanField*) – Czy e-mail do jest z monitoringu?
- **is_spammer_domain** (*BooleanField*) – Jest spamerem
- **is_non_spammer_domain** (*BooleanField*) – Nie jest spamerem
- **email_to_count** (*IntegerField*) – Liczba adresów email do
- **email_from_count** (*IntegerField*) – Liczba adresów email od

exception DoesNotExist

exception MultipleObjectsReturned

save(**args*, ***kwargs*)

Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class feder.letters.models.**LetterQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

class feder.letters.models.**MassMessageDraft**(*id*, *created*, *modified*, *letter*, *monitoring*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*CreationDateTimeField*) – Utworzony
- **modified** (*ModificationDateTimeField*) – Zmodyfikowany
- **letter_id** (OneToOneField to *feder.letters.models.Letter*) – List
- **monitoring_id** (ForeignKey to *feder.monitorings.models.Monitoring*) – Monitoring

exception DoesNotExist

exception MultipleObjectsReturned

class feder.letters.models.**ReputableLetterEmailTLD**(*id*, *created*, *modified*, *name*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*CreationDateTimeField*) – Utworzony
- **modified** (*ModificationDateTimeField*) – Zmodyfikowany
- **name** (*CharField*) – Renomowana domena najwyższego poziomu z adresu e-mail

exception DoesNotExist

exception MultipleObjectsReturned

Panel administracyjny

class feder.letters.admin.**AttachmentInline**(*parent_model*, *admin_site*)

Stacked Inline View for Attachment

model

alias of *Attachment*

class feder.letters.admin.**LetterAdmin**(*model*, *admin_site*)

Admin View for Letter

class feder.letters.admin.**LetterEmailDomainAdmin**(*model*, *admin_site*)

Admin View for LetterEmailDomain

class feder.letters.admin.**ReputableLetterEmailTLDAdmin**(*model*, *admin_site*)

Admin View for ReputableLetterEmailTLD

Widoki

class feder.letters.views.**AssignLetterFormView**(***kwargs*)

Parameters

url_name – `letters:assign`

form_class

alias of `AssignLetterForm`

form_valid(*form*)

If the form is valid, redirect to the supplied URL.

get_context_data(***kwargs*)

Insert the form into the context dict.

get_form_kwargs()

Return the keyword arguments for instantiating the form.

get_success_url()

Return the URL to redirect to after processing a valid form.

model

alias of *Letter*

class feder.letters.views.**AttachmentRequestCreateView**(***kwargs*)

Parameters
url_name – letters:scan

get_object(**args*, ***kwargs*)

Return the object the view is displaying.

Require *self.queryset* and a *pk* or *slug* argument in the URLconf. Subclasses can override this to return any object.

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Attachment*

class feder.letters.views.**AttachmentXSendFileView**(***kwargs*)

Parameters

- **url_name** – letters:attachment
- **url_name** – letters:None

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Attachment*

class feder.letters.views.**LetterCaseAtomFeed**

feed_type

alias of Atom1Feed

subtitle(*obj*)

Encapsulate a function call and act as a proxy for methods that are called on the result of that function. The function is not evaluated until one of the methods on the result is called.

class feder.letters.views.**LetterCaseRssFeed**

description(*obj*)

Encapsulate a function call and act as a proxy for methods that are called on the result of that function. The function is not evaluated until one of the methods on the result is called.

model

alias of *Case*

title(*obj*)

Encapsulate a function call and act as a proxy for methods that are called on the result of that function. The function is not evaluated until one of the methods on the result is called.

class feder.letters.views.**LetterCommonMixin**

Defines get_queryset and get_permission_object methods. It should to be specified before permission related mixins.

class feder.letters.views.**LetterCreateView**(***kwargs*)

Parameters
url_name – `letters:create`

form_class

alias of `LetterForm`

get_context_data(***kwargs*)

Insert the form into the context dict.

get_form_kwargs()

Return the keyword arguments for instantiating the form.

model

alias of *Letter*

class feder.letters.views.**LetterDeleteView**(**args*, ***kwargs*)

Parameters
url_name – `letters:delete`

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

get_success_url()

Return the URL to redirect to after processing a valid form.

model

alias of *Letter*

class feder.letters.views.**LetterDetailView**(***kwargs*)

Parameters

- **url_name** – `letters:details`
- **url_name** – `letters:details`

get_context_data(***kwargs*)

Insert the single object into the context dict.

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Letter*

class feder.letters.views.**LetterListView**(***kwargs*)

Parameters
url_name – `letters:list`

get_context_data(***kwargs*)

Get the context for this view.

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

model

alias of *Letter*

class feder.letters.views.**LetterMarkSpamView**(***kwargs*)

Parameters
url_name – `letters:mark_spam`

get_object(**args*, ***kwargs*)

Return the object the view is displaying.

Require *self.queryset* and a *pk* or *slug* argument in the URLconf. Subclasses can override this to return any object.

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Letter*

class feder.letters.views.**LetterMessageXSendFileView**(***kwargs*)

Parameters
url_name – `letters:download`

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Letter*

class feder.letters.views.**LetterMonitoringAtomFeed**

feed_type

alias of `Atom1Feed`

subtitle(*obj*)

Encapsulate a function call and act as a proxy for methods that are called on the result of that function. The function is not evaluated until one of the methods on the result is called.

class feder.letters.views.**LetterMonitoringRssFeed**

description(*obj*)

Encapsulate a function call and act as a proxy for methods that are called on the result of that function. The function is not evaluated until one of the methods on the result is called.

model

alias of *Monitoring*

title(*obj*)

Encapsulate a function call and act as a proxy for methods that are called on the result of that function. The function is not evaluated until one of the methods on the result is called.

class feder.letters.views.**LetterReplyView**(***kwargs*)

Parameters
url_name – letters:reply

form_class

alias of ReplyForm

forms_valid(*form*, *inlines*)

If the form and formsets are valid, save the associated models.

get_context_data(***kwargs*)

Insert the form into the context dict.

get_form_kwargs()

Return the keyword arguments for instantiating the form.

get_form_valid_message()

Validate that form_valid_message is set and is either a unicode or str object.

model

alias of *Letter*

class feder.letters.views.**LetterReportSpamView**(***kwargs*)

Parameters
url_name – letters:spam

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Letter*

class feder.letters.views.**LetterResendView**(***kwargs*)

Parameters
url_name – letters:resend

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Letter*

class feder.letters.views.**LetterSendView**(***kwargs*)

Parameters
url_name – letters:send

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Letter*

class feder.letters.views.**LetterUpdateView**(***kwargs*)

Parameters
url_name – letters:update

form_class

alias of LetterForm

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Letter*

class feder.letters.views.**ReceiveEmail**(***kwargs*)

Parameters
url_name – letters:webhook

class feder.letters.views.**UnrecognizedLetterListView**(***kwargs*)

Parameters
url_name – letters:unrecognized_list

get_context_data(***kwargs*)

Get the context for this view.

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

model

alias of *Letter*

3.6 Dziennik listów

3.6.1 Założenia

Moduł przeznaczony jest do gromadzenia informacji na temat dostarczenia wiadomości przesłanych w monitoringu / sprawie, a docelowo także indywidualnych wiadomości.

Moduł dostarcza polecenie `python manage.py update_emaillabs`, który pobiera aktualne wpisy dziennika z Emaillabs , a następnie archiwizuje te, które dotyczą spraw zarejestrowanych w systemie.

Dostęp do dzienników jest możliwy przez użytkownika, który ma uprawnienie `view_logs` w danym monitoringu.

3.6.2 Dane testowe

Dla modułu istnieją stosowne fabryki w module `feder.letters.logs.factories`.

3.6.3 Architektura

Model

class feder.letters.logs.models.**EmailLog**(*id*, *created*, *modified*, *status*, *case*, *letter*, *email_id*, *to*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Utworzony
- **modified** (*AutoLastModifiedField*) – Zmodyfikowany
- **status** (*CharField*) – Status
- **case_id** (ForeignKey to *feder.cases.models.Case*) – Case
- **letter_id** (OneToOneField to *feder.letters.models.Letter*) – Letter
- **email_id** (*CharField*) – Message-id
- **to** (*CharField*) – Do

exception DoesNotExist

exception MultipleObjectsReturned

class feder.letters.logs.models.**EmailQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

class feder.letters.logs.models.**LogRecord**(*id*, *created*, *modified*, *email*, *data*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Utworzony
- **modified** (*AutoLastModifiedField*) – Zmodyfikowany
- **email_id** (ForeignKey to *feder.letters.logs.models.EmailLog*) – E-mail
- **data** (*JSONField*) – Data

exception DoesNotExist

exception MultipleObjectsReturned

class feder.letters.logs.models.**LogRecordQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Panel administracyjny

class feder.letters.logs.admin.**EmailLogAdmin**(*model*, *admin_site*)

Admin View for EmailLog

actions = None

has_add_permission(*request*, *obj=None*)

Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

has_change_permission(*request*, *obj=None*)

Return True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to change the *obj* model instance. If *obj* is None, this should return True if the given request has permission to change *any* object of the given type.

has_delete_permission(*request*, *obj=None*)

Return True if the given request has permission to delete the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to delete the *obj* model instance. If *obj* is None, this should return True if the given request has permission to delete *any* object of the given type.

class feder.letters.logs.admin.**LogRecordInline**(*parent_model*, *admin_site*)

Stacked Inline View for LogRecord

model

alias of *LogRecord*

Widoki

class feder.letters.logs.views.**EmailLogCaseListView**(***kwargs*)

Parameters
url_name – logs:list

get_context_data(***kwargs*)

Get the context for this view.

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

class feder.letters.logs.views.**EmailLogDetailView**(***kwargs*)

Parameters
url_name – logs:detail

model

alias of *EmailLog*

class feder.letters.logs.views.**EmailLogMonitoringCsvView**(***kwargs*)

Parameters
url_name – logs:export

class feder.letters.logs.views.**EmailLogMonitoringListView**(***kwargs*)

Parameters
url_name – logs:list

class feder.letters.logs.views.**ListMonitoringMixin**

model

alias of *EmailLog*

3.7 Monitoringi

3.7.1 Założenia

Moduł stanowi komponent, który agreguje sprawy związane z różnymi urzędami, które odnoszą się do zbiernia informacji tej samej kategorii. Zatem monitoringiem będzie np. zainteresowanie wysoką opłaty za śmieci w Polsce. Na tej postawie system tworzy liczne sprawy dla każdego urzędu, który ma być objęty badaniem.

3.7.2 Architektura

Model

class feder.monitorings.models.**Monitoring**(*id*, *created*, *modified*, *name*, *slug*, *user*, *description*, *subject*, *hide_new_cases*, *template*, *use_llm*, *responses_chat_context*, *normalized_response_template*, *results*, *email_footer*, *notify_alert*, *is_public*, *domain*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Utworzony
- **modified** (*AutoLastModifiedField*) – Zmodyfikowany
- **name** (*CharField*) – Nazwa
- **slug** (*AutoSlugField*) – Adres url
- **user_id** (ForeignKey to feder.users.models.User) – Użytkownik / użytkowniczka
- **description** (*TextField*) – Opis
- **subject** (*CharField*) – Temat

- **hide_new_cases** (*BooleanField*) – Czy ukrywać nowe sprawy przy przypisywaniu?
- **template** (*TextField*) – Użyj {{EMAIL}} aby umieścić adres odpowiedzi
- **use_llm** (*BooleanField*) – Użyj LLM do oceny odpowiedzi
- **responses_chat_context** (*JSONField*) – Kontekst odpowiedzi w monitoringu dla czatu AI
- **normalized_response_template** (*JSONField*) – Znormalizowany szablon odpowiedzi
- **results** (*TextField*) – Wyniki monitoringu i otrzymanych odpowiedzi
- **email_footer** (*TextField*) – Podpis w stopce e-maili, w tym w odpowiedziach na e-maile
- **notify_alert** (*BooleanField*) – Powiadom o nowych alertach osoby, które mogą je widzieć
- **is_public** (*BooleanField*) – Czy publicznie widoczny?
- **domain_id** (ForeignKey to *feder.domains.models.Domain*) – Domena użyta do wysłania wiadomości

exception DoesNotExist

exception MultipleObjectsReturned

generate_voivodeship_table()

Generate html table with monitoring voivodeships and their institutions and cases counts

class feder.monitorings.models.**MonitoringGroupObjectPermission**(*id*, *permission*, *group*, *content_object*)

Parameters

- **id** (*AutoField*) – Id
- **permission_id** (ForeignKey to django.contrib.auth.models.Permission) – Permission
- **group_id** (ForeignKey to django.contrib.auth.models.Group) – Group
- **content_object_id** (ForeignKey to *feder.monitorings.models.Monitoring*) – Content object

exception DoesNotExist

exception MultipleObjectsReturned

class feder.monitorings.models.**MonitoringQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

with_case_confirmation_received_count()

function to annotate with case count when case.confirmation_received field is True

with_case_quarantined_count()

function to annotate with case count when case.is_quarantined field is True

with_case_response_received_count()

function to annotate with case count when case.response_received field is True

class feder.monitorings.models.**MonitoringUserObjectPermission**(*id*, *permission*, *user*, *content_object*)

Parameters

- **id** (*AutoField*) – Id
- **permission_id** (ForeignKey to django.contrib.auth.models.Permission) – Permission
- **user_id** (ForeignKey to feder.users.models.User) – User
- **content_object_id** (ForeignKey to *feder.monitorings.models.Monitoring*) – Content object

exception **DoesNotExist**

exception **MultipleObjectsReturned**

Panel administracyjny

class feder.monitorings.admin.**MonitoringAdmin**(**args*, ***kwargs*)

Admin View for Monitoring

actions = None

Widoki

class feder.monitorings.views.**DraftListMonitoringView**(***kwargs*)

Parameters

- **url_name** – monitorings:drafts
- **url_name** – monitorings:drafts

get_context_data(***kwargs*)

Insert the single object into the context dict.

get_object_list(*obj*)

A method to return object list to additional list. This should be overriden.

Parameters
obj – The object the view is displaying.

Returns
A list of object to paginated

Return type
QuerySet

Raises
ImproperlyConfigured – The method was not overrided.

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Monitoring*

class feder.monitorings.views.**LetterListMonitoringView**(***kwargs*)

Parameters

- **url_name** – monitorings:letters
- **url_name** – monitorings:letters

get_context_data(***kwargs*)

Insert the single object into the context dict.

get_object_list(*obj*)

A method to return object list to additional list. This should be overriden.

Parameters
obj – The object the view is displaying.

Returns
A list of object to paginated

Return type
QuerySet

Raises
ImproperlyConfigured – The method was not overrided.

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Monitoring*

class feder.monitorings.views.**MassMessageView**(***kwargs*)

Parameters
url_name – monitorings:mass-message

form_class

alias of MassMessageForm

forms_valid(*form*, *inlines*)

If the form and formsets are valid, save the associated models.

get_context_data(***kwargs*)

Insert the form into the context dict.

get_form_kwargs()

Return the keyword arguments for instantiating the form.

get_success_url()

Return the URL to redirect to after processing a valid form.

model

alias of *Letter*

class feder.monitorings.views.**MonitoringAssignView**(***kwargs*)

Parameters
url_name – monitorings:assign

get_context_data(***kwargs*)

Get the context for this view.

get_filterset_kwargs(*filterset_class*)

Returns the keyword arguments for instantiating the filterset.

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

model

alias of *Institution*

class feder.monitorings.views.**MonitoringAutocomplete**(***kwargs*)

Parameters
url_name – monitorings:autocomplete

get_queryset()

Filter the queryset with GET['q'].

class feder.monitorings.views.**MonitoringCasesAjaxDatatableView**(***kwargs*)

View to provide table list of all Monitoring Cases with ajax data.

Parameters
url_name – monitorings:monitoring_cases_table_ajax_data

get_latest_by(*request*)

Override to customize based of request.

Provides the name of the column to be used for global date range filtering. Return either '', a fieldname or None.

When None is returned, in model's Meta 'get_latest_by' attributed will be used.

model

alias of *Case*

class feder.monitorings.views.**MonitoringCasesTableView**(***kwargs*)

View for displaying template with table of Monitoring Cases.

Parameters
url_name – monitorings:monitoring_cases_table

get_context_data(**args*, ***kwargs*)

Get the context for this view.

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringChatView**(***kwargs*)

get_context_data(***kwargs*)

Insert the single object into the context dict.

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringCreateView**(***kwargs*)

Parameters
url_name – monitorings:create

form_class

alias of MonitoringForm

form_valid(*form*)

If the form is valid, save the associated model.

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringDeleteView**(**args*, ***kwargs*)

Parameters
url_name – monitorings:delete

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringDetailView**(***kwargs*)

Parameters

- **url_name** – monitorings:details
- **url_name** – monitorings:details

get_context_data(***kwargs*)

Insert the single object into the context dict.

get_object_list(*obj*)

A method to return object list to additional list. This should be overriden.

Parameters
obj – The object the view is displaying.

Returns
A list of object to paginated

Return type
QuerySet

Raises
ImproperlyConfigured – The method was not overrided.

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringListView**(***kwargs*)

Parameters
url_name – monitorings:list

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringPermissionView**(***kwargs*)

Parameters
url_name – monitorings:perm

get_context_data(***kwargs*)

Insert the single object into the context dict.

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringReportView**(***kwargs*)

Parameters

- **url_name** – monitorings:report
- **url_name** – monitorings:report

get_context_data(***kwargs*)

Get the context for this view.

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

get_template_names()

Return a list of template names to be used for the request. Must return a list. May not be called if render_to_response is overridden.

model

alias of *Case*

class feder.monitorings.views.**MonitoringResponsesReportView**(***kwargs*)

Parameters
url_name – monitorings:responses_report

class feder.monitorings.views.**MonitoringResultsUpdateView**(***kwargs*)

Parameters
url_name – monitorings:results_update

form_class

alias of MonitoringResultsForm

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringResultsView**(***kwargs*)

Parameters
url_name – monitorings:results

get_context_data(***kwargs*)

Insert the single object into the context dict.

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringTemplateView**(***kwargs*)

Parameters
url_name – monitorings:template

get_context_data(***kwargs*)

Insert the single object into the context dict.

get_queryset()

Return the *QuerySet* that will be used to look up the object.

This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringUpdatePermissionView**(***kwargs*)

Parameters
url_name – monitorings:perm-update

form_class

alias of SaveTranslatedUserObjectPermissionsForm

form_valid(*form*)

If the form is valid, redirect to the supplied URL.

get_context_data(***kwargs*)

Insert the form into the context dict.

get_form_kwargs()

Return the keyword arguments for instantiating the form.

class feder.monitorings.views.**MonitoringUpdateView**(***kwargs*)

Parameters
url_name – monitorings:update

form_class

alias of MonitoringForm

form_valid(*form*)

Call the super first, so that when overriding get_form_valid_message, we have access to the newly saved object.

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringsAjaxDatatableView**(***kwargs*)

View to provide table list of all Monitorings with ajax data.

Parameters
url_name – monitorings:monitorings_table_ajax_data

model

alias of *Monitoring*

class feder.monitorings.views.**MonitoringsTableView**(***kwargs*)

View for displaying template with Monitorings table.

Parameters
url_name – monitorings:table

class feder.monitorings.views.**MultiCaseTagManagement**(***kwargs*)

Parameters
url_name – monitoring-case-tags-update

class feder.monitorings.views.**PermissionWizard**(***kwargs*)

Parameters
url_name – monitorings:perm-add

done(*form_list*, *form_dict*, **args*, ***kwargs*)

This method must be overridden by a subclass to process to form data after processing all steps.

get_context_data(**args*, ***kwargs*)

Returns the template context for a step. You can overwrite this method to add more data for all or some steps. This method returns a dictionary containing the rendered form step. Available template context variables are:

- all extra data stored in the storage backend
- *wizard* - a dictionary representation of the wizard instance

Example:

```
class MyWizard(WizardView):
    def get_context_data(self, form, **kwargs):
        context = super().get_context_data(form=form, **kwargs)
        if self.steps.current == 'my_step_name':
            context.update({'another_var': True})
        return context
```

get_form_kwargs(*step=None*)

Returns the keyword arguments for instantiating the form (or formset) on the given step.

class feder.monitorings.views.**UserMonitoringAutocomplete**(***kwargs*)

Parameters
url_name – monitorings:autocomplete_user

get_queryset()
Filter the queryset with GET['q'].

3.8 Jednostki podziału terytorialnego

3.8.1 Założenia

Moduł dostarcza informacje na temat podziału terytorialnego w Polsce. Zapewnia przegląd instytucji w danym regionie. Oparty jest o moduł django-teryt-tree dla którego istnieje odrębna dokumentacja.

3.8.2 Dane testowe

Dla modułu możliwe jest zaimportowanie automatyczne danych testowych. Mogą posłużyć do tego polecenia:

```
wget "http://www.stat.gov.pl/broker/access/prefile/downloadPreFile.jspa?id=1110" -O TERC.
↪xml.zip;
unzip TERC.xml.zip;
pip install lxml;
python manage.py load_teryt TERC.xml;
rm TERC.xml*;
```

W razie trudności - patrz sekcja "Quickstart" dokumentacji django-teryt-tree .

3.8.3 Architektura

Model

class feder.teryt.models.**JST**(*id*, *parent*, *name*, *category*, *slug*, *updated_on*, *active*, *lft*, *rght*, *tree_id*, *level*)

Parameters

- **id** (*CharField*) – Id
- **parent_id** (TreeForeignKey to teryt_tree.models.JednostkaAdministracyjna) – Parent
- **name** (*CharField*) – Nazwa
- **category_id** (ForeignKey to teryt_tree.models.Category) – Category
- **slug** (*AutoSlugField*) – Slug
- **updated_on** (*DateField*) – Updated date
- **active** (*BooleanField*) – Active
- **lft** (*PositiveIntegerField*) – Lft
- **rght** (*PositiveIntegerField*) – Rght
- **tree_id** (*PositiveIntegerField*) – Tree id

- **level** (*PositiveIntegerField*) – Level

exception DoesNotExist

exception MultipleObjectsReturned

Widoki

class feder.teryt.views.**CustomCommunityAutocomplete**(***kwargs*)

Parameters
url_name – teryt:community-autocomplete

get_base_queryset()

Refactored from CommunityAutocomplete view to use JST model instead of JednostkaAdministracyjna. additionally select_related "parent" and "parent__parent" has been added and filtered only the active records.

class feder.teryt.views.**JSTAutocomplete**(***kwargs*)

Parameters
url_name – teryt:jst-autocomplete

class feder.teryt.views.**JSTDetailView**(***kwargs*)

Parameters
url_name – teryt:details

get_context_data(***kwargs*)

Insert the single object into the context dict.

model

alias of *JST*

class feder.teryt.views.**JSTListView**(***kwargs*)

Parameters

- **url_name** – teryt:list
- **url_name** – teryt:voivodeship

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

model

alias of *JST*

class feder.teryt.views.**TerytViewSet**(***kwargs*)

serializer_class

alias of JednostkaAdministracyjnaSerializer

3.9 Skanowanie antywirusowe

3.9.1 Założenia

Moduł odpowiedzialny jest za skanowanie wybranych przez użytkowników plików z użyciem skanerów antywirusowych on-line.

Dostępne są następujące silniki skanowania:

- VirusTotal - limit 4 żądań / minutę, brak limitu plików, szczegóły
- MetaDefender Cloud - limit 10 żądań / minutę, 100 żądań / dzień, szczegóły: https://metadefender.opswat.com/licensing
- AttachmentScanner - brak limitów, niska skuteczność, szczegóły: https://www.attachmentscanner.com/pricing

3.9.2 Architektura

Model

class feder.virus_scan.models.**Request**(*id*, *created*, *modified*, *content_type*, *object_id*, *field_name*, *engine_name*, *engine_id*, *engine_report*, *engine_link*, *status*)

Parameters

- **id** (*AutoField*) – Id
- **created** (*AutoCreatedField*) – Utworzony
- **modified** (*AutoLastModifiedField*) – Zmodyfikowany
- **content_type_id** (ForeignKey to `django.contrib.contenttypes.models.ContentType`) – Content type
- **object_id** (*PositiveIntegerField*) – Object id
- **field_name** (*CharField*) – Field name
- **engine_name** (*CharField*) – Nazwa silnika
- **engine_id** (*CharField*) – Zewnętrzny identyfikator
- **engine_report** (*JSONField*) – Wynik silnika
- **engine_link** (*CharField*) – Link do wyniku silnika
- **status** (*IntegerField*) – Status

exception **DoesNotExist**

exception **MultipleObjectsReturned**

class feder.virus_scan.models.**RequestQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Widoki

class feder.virus_scan.views.**RequestWebhookView**(***kwargs*)

Parameters
url_name – virus_scan:webhook

Panel administracyjny

class feder.virus_scan.admin.**ScanRequestAdmin**(*model*, *admin_site*)

actions = None

get_queryset(**args*, ***kwargs*)

Return a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

Silniki

exception feder.virus_scan.engine.**NotFoundEngineException**

CHAPTER

FOUR

ROZWÓJ

W tym dokumencie opisujemy opis procesu rozwoju aplkacji. Ma on postać FAQ, aby utrzymywać dokument prostym.

4.1 Jak zgłosić usterkę?

Po prostu przejdź na https://github.com/watchdogpolska/feder/issues i utworz zgłoszenie.

4.2 Jak diagnozować funkcjonowanie poczty elektronicznej?

W środowisku deweloperskim wiadomości e-mail są domyślnie wypisywane na konsole w oknie serwera WWW. Jeżeli chcesz zweryfikować np. formatowanie wiadomości zaleca się wykorzystanie maildump, który możliwy jest do zainstalowania i uruchomienia poprzez:

```
$ pip install maildump
$ maildump
```

Następnie należy ponownie uruchomić serwer WWW w następujący sposób `EMAIL_URL=smtp://localhost:1025/ python manage.py runserver`. Wiadomości będą dostępne przez interfejs WWW pod adresem `http://localhost:1080`.

4.3 Jak uruchomić automatyczne testy?

Do prawidłowego uruchomienia automatycznych testów bezwzględnie wymagane jest zainstalowanie wszystkich deweloperskich pakietów. Można to osiągnąć poprzez:

```
$ make build
```

Następnie należy wywołać:

```
$ make test
```

4.4 Jak wygenerować dokumentacje?

Do prawidłowego zbudowania dokumentacji bezwzględnie wymagane jest zainstalowanie wszystkich deweloperskich pakietów. Można to osiągnąć poprzez:

```
$ make build
```

Następnie wywołać:

```
$ make docs
```

Warto zaznaczyć, że aktualna dokumentacja jest budowana automatycznie i publikowana na Read the Docs.

4.5 Jak pozyskać testowe dane?

W celu utworzenia danych testowych należy zaimportować podział terytorialny zgodnie z instrukcją biblioteki django-teryt-tree. Następnie można wykorzystać poniższy kod:

```
from feder.letters.factories import SendOutgoingLetterFactory
SendOutgoingLetterFactory()
```

Jeżeli otrzymujesz:

```
IntegrityError: (1062, "Duplicate entry '1' for key 'PRIMARY'")
```

oznacza to w bazie danych istnieją rekordy, które kolidują z istniejącymi danymi. Możesz ponownie uruchomić `SendOutgoingLetterFactory()`, a licznik powinien wzrosnąć, co pozwoli uniknąć kolizji, albo usuń zgromadzone dane.

Ręcznie taka procedura polega na dodaniu instytucji, potem stworzeniu monitoringu i przypisaniu do niego instytucje. Wówczas powinny istnieć także list w sprawie.

4.6 Jakie jest hasło dla automatycznie utworzonego użytkownika?

Domyślnym hasłem utworzonych *Jak pozyskać testowe dane?* to `pass`. Zostało ono określone w klasie `feder.users.UserFactory`.

4.7 Jak utworzyć konto administratora?

Konto administratora może zostać utworzone poprzez polecenie `python manage.py creatsuperuser`. Szczegółowe parametry są przedstawione na odpowiedniej podstronie dokumentacji Django.

CHAPTER

FIVE

PODRĘCZNIK UŻYTKOWNIKA

5.1 Wprowadzenie

Zapoznaj się z wprowadzeniem przedstawiającym podstawowe założenia aplikacji - *Koncepcja aplikacji* .

5.2 Słownik

5.2.1 Obiekty

Monitoring
: zbiór powiązanych tematycznie spraw dotyczących jednolitego tematu i jednolitej grupy podmiotów

Zadanie
: przyporządkowanie konkretnego kwestionariusza do monitoringu

Kwestionariusz
: szablon ankiety, która po utworzeniu zadania w sprawie będzie wymagała uzupełnienia przez użytkownika, definiuje także format zebranych danych, składający się z pytań określające informacje, które chcemy przypisać w ramach wykonania zadania

Ankieta
: wypełniony kwestionariusz zawierający odpowiedzi na zadane pytania

Sprawa
: zbiór korespondencji wymienionej w konkretnej sprawie do konkretnej instytucji, a także zadania i ankiety danej sprawy

List
: dowolny skrawek informacji, który jest wymieniony w postaci pisemnej pomiędzy instytucją a organizatorem monitoringu

5.2.2 Czynności

Przypisanie do sprawy instytucji
wybranie "Przypisz" na ekranie montioringu skutkujące utworzeniem sprawy, wysłaniem wniosku i zarejestrowaniem pierwszego pisma w sprawie

5.3 Utworzenie monitoringu

W przypadku użytkowania oprogramowania powtarzającym się cyklem będzie utworzenie monitoringu, następnie utworzenie spraw wraz z wysłaniem listów. Odczekanie na nadejście odpowiedzi - listów z urzędów. Następnie określany jest kwestionariusz, a następnie jest on przypisywany do spraw, które są gotowe do tego, aby użytkownicy dokonywali ich analizy.

Utworzenie monitoringu należy rozpocząć od weryfikacji kompletności bazy instytucji, która podlegać będą monitoringowi. Należy dokonać oceny informacji dotychczas zgromadzonych w instancji aplikacji. Warto także dokonać weryfikacji jej aktualności, w miarę dostępnych zasobów.

W przypadku braku określonych podmiotów w bazie adresowej instytucji konieczne jest opracowanie bazy adresowej instytucji. Szczegółowe wymagania w tym zakresie zostały przedstawione w *Przygotowanie bazy adresowej* .

W celu wgrania bazy adresowej instytucji możesz wykorzystać API REST (dostępne pod adresem `http://example.com/api/institutions/` , które wspiera metody GET a po uwierzytelnieniu kontem użytkownika PUT, POST i PATCH). Inne formy zmian bazy adresowej systemu pozostają w trakcie opracowania Eksport instytucji do CSV .

W kolejnym kroku przechodzimy do zakładki Szukaj -> Monitoringi -> Dodaj monitoring. Wypełniamy formularz podstawowymi informacjami na jego temat. W tym kroku nie określamy instytucji, które będą adresatami zapytań, ale określamy szablon zapytania, które zostanie do nich przesłane.

> **Caution:** Jeżeli opcja „Dodaj monitoring" jest niedostępna - skontaktuj się z operatorem systemu, aby uzyskać stosowne uprawnienia.

Po przejściu na stosunkowo pustą stronę monitoringu wybieramy przycisk „Przypisz". Uzyskujesz ekran na którym masz dostępny wykaz instytucji a także możliwość ich odfiltrowania.

Dokonaj starannego wyboru instytucji, które mają być adresatami pytań. Możesz dokonać to przez zaznaczenie każdej z nich indywidualnie. Możesz także umiejętnie operować filtrowaniem, aby uzyskać w zestawieniu tylko podmioty, które mają być adresatami petycji. Pomocne mogą być w tym celu zwłaszcza tagi. Po uzyskaniu takiego zestawienia możesz w 1 wierszu (wierszu nagłówkowym) tabeli zaznaczyć pole, które będzie równoważne oznaczeniu wszystkich podmiotów na wszystkich stronach zestawienia.

Po dokonaniu wyboru instytucji wybierz przycisk „Przypisz (...) i wyślij wniosek", aby utworzyć sprawy dla każdej z instytucji, a także skierować do niej list.

> **Warning:** Operacja przypisania do sprawy jest nieodwracalna, bowiem automatycznie po przypisaniu do sprawy jest wysyłany wniosek zgodny z szablonem ustalonym dla danego monitoringu.

Po nadejściu odpowiedzi z urzędów będą one automatycznie opublikowane na stronie monitoringu.

Todo: Przedstawić tworzenie kwestionariusza, a następnie formułowania zadań dla niego.

5.4 Przygotowanie bazy adresowej

Rzetelne funkcjonowanie systemu Obywatelskiego Fedrowania Danych rozumiane w szczególności jako przystępna nawigacja, pomimo rozrostu systemu, wymaga odpowiedniej jakości bazy danych. Wymaganymi danymi dla każdej instytucji jest co najmniej:

- pełna i kompletna nazwa instytucji - identyfikacji instytucji przez użytkowników strony,
- adres e-mail instytucji - korespondencji z instytucją,
- numer identyfikacyjny REGON - weryfikacji unikalności instytucji, a także weryfikacji zmian w strukturze instytucji (likwidacja, przekształcenie itd.)

Gromadzone są także:

- tagi, które pozwalają na przypisywanie jednej lub kilku kategorii do instytucji np. marszałkowie, lasy państwowe, dyrekcja lasów państwowych, sądy,
- wyciąg z rejestru REGON,
- odwołanie do instytucji nadrzędnych - do potencjalnie przyszłego wykorzystania,
- kod TERC (rejestr podziału terytorialnego) z bazy TERYT - nawigacja według regionu.
- inne informacje, które mogą zostać w przyszłości wykorzystane.

Wskazuje, że Stowarzyszenie Sieć Obywatelska Watchdog Polska ma zapewniony dostęp do rejestru REGON na podstawie numeru REGON/NIP/KRS poprzez API. Kluczami dostępowymi dysponuje Administrator Bezpieczeństwa Informacji.

Nie mamy możliwości przeszukiwania tej bazy danych na podstawie regionów, ani nazw instytucji, ani ich kategorii. Stowarzyszenie ma jednak możliwość wyszukania nazwy instytucji w Google, pobrania wszystkich cyfr z nadzieją, że wśród nich będzie numer REGON, co pozwoli uzupełnić lokalną kopie rejestru REGON, a następnie przeszukiwania według nazwy wcześniej pobrane instytucje, które były przedmiotem odrębnych analiz. Pozwoli to opracować częściowe zestawienie, które może okazać się pomocne w dalszych pracach.

Rejestr REGON pozwala na wyszukiwanie po numerze NIP i KRS, zatem kod REGON nie wymaga uzupełnienia w przypadku obecności kodu NIP/KRS. Mogą one być stosowane wymiennie.

Mając na względzie powyższe:

- pełna i kompletna nazwa instytucji - rejestr REGON pozwala na dostarczenie podpowiedzi w tym zakresie na podstawie numeru REGON, lecz rejestr REGON nie jest w tym zakresie wystarczający, gdyż w wielu wypadkach wprowadzone nazwy nie spełniają poniżej przedstawionych wymogów,
- adres e-mail instytucji - rejestr REGON pozwala czasem na dostarczenie informacji w tym zakresie, nie zawsze aktualnych i kompletnych,
- numer REGON - rejestr REGON pozwala na uzupełnienie informacji na podstawie numerów NIP/KRS i można dostarczyć podpowiedzi na podstawie wcześniej pobranych rekordów bazy REGON,
- numer TERC (rejestr podziału terytorialnego) z bazy TERYT - rejestr REGON pozwala w pełni na uzupełnienie tych informacji.

5.4.1 Format nazw instytucji

Nazwy instytucji powinny być możliwie jednoznacznie identyfikować instytucje i być zapisane zgodnie z zasadami języka polskiego. Nie akceptowalna w polu nazwa instytucji jest występowanie samych nazw miejscowości, ani sama nazwa rodzajowa instytucji. Nazwa instytucji regionalnych winna w miarę możliwości składać się z oznaczenia `[kategoria / nazwa] [w/we] [miejscowość odmieniony jako miejscownik]`. Niedopuszczalne jest stosowanie w nazwie danej instytucji nazw innych instytucji np. instytucji nadrzędnej.

5.4.2 Uwagi na temat rejestru REGON

Rejestr REGON:

- służy osiągnięciu spójności identyfikacyjnej podmiotów gospodarki narodowej wpisywanych do innych urzędowych rejestrów i systemów informacyjnych administracji publicznej,
- służy jednolitości opisów stosowanych w nomenklaturze pojęciowej i klasyfikacyjnej we wszystkich urzędowych rejestrach i systemach informacyjnych administracji publicznej,
- dostarcza ogólnej charakterystyki działających w gospodarce narodowej podmiotów w przekrojach: terytorialnym, własnościowym, rodzajów działalności, form prawnych itp.,
- umożliwia sporządzanie wykazu adresów działających podmiotów,
- jest podstawą do tworzenia baz i banków danych o podmiotach gospodarki narodowej,
- stanowi główne źródło zasilania bazy jednostek wybieranych do badań statystycznych.

Dostęp do rejestru REGON możliwy jest poprzez oficjalną internetową wyszukiwarkę.

Numer identyfikacyjny REGON podmiotu gospodarki narodowej skreślonego z rejestru REGON jest przechowywany w zbiorze historycznym i nie jest wykorzystywany do identyfikacji innego podmiotu.

Numer identyfikacyjny REGON podmiotu gospodarki narodowej składa się z dziewięciu cyfr, które nie mogą mieć ukrytego lub jawnego charakteru znaczącego, określającego pewne cechy podmiotu, przy czym osiem pierwszych cyfr stanowi liczbę porządkową, a dziewiąta – cyfrę kontrolną.

Numer identyfikacyjny REGON jednostki lokalnej składa się z czternastu cyfr, przy czym dziewięć pierwszych cyfr jest tożsame z numerem identyfikacyjnym REGON osoby prawnej, jednostki organizacyjnej niemającej osobowości prawnej lub osoby fizycznej prowadzącej działalność gospodarczą, cztery kolejne cyfry są liczbą porządkową przypisaną jednostce lokalnej, a czternasta cyfra – cyfrą kontrolną.

Dopuszczalna jest także notacja dla podmiotu gospodarki narodowej, który tradycyjnie ma 9 cyfrowy numer REGON poprzez uzupełnienie 5 cyframi zero z prawej strony do czternastu cyfr. Zatem numery REGON `11001690600000` i `110016906` są równoważne.

Szczegółowe informacje w zakresie funkcjonowania rejestru są przedstawione w:

- Informacje ogólne – Biuletyn Informacji Publicznej Głównego Urzędu Statystycznego
- Rozporządzenie Rady Ministrów z dnia 30 listopada 2015 r. w sprawie sposobu i metodologii prowadzenia i aktualizacji krajowego rejestru urzędowego podmiotów gospodarki narodowej, wzorów wniosków, ankiet i zaświadczeń (Dz. U. poz. 2009, z późn. zm.)

5.5 Zarządzanie korespondencja

Każda sprawa w systemie ma nadany unikalny adres e-mail. Pozwala on na automatyczną rejestracje przychodzącej wiadomości do konkretnej sprawy. Każda przychodząca wiadomość może odnosić się wyłącznie do jednej sprawy.

System odpowiedzialny jest za podstawowe operacje związane z zarządzanie korespondencją. W przypadku wysyłki jakiejkolwiek wiadomości z systemu w danej sprawie jest wykorzystywany unikalny adres e-mail.

Pole daty wiadomości identyfikuje czas rejestracji odpowiedzi przez system Stowarzyszenia. Nie jest to równoważne z czasem doręczenia do Stowarzyszenia, a także z czasem wysłania wiadomości przez instytucje ze względu na sposób działania poczty, która pozwala na dużą manipulacje tymi danymi. W przypadku chęci poznania czasu odpowiedzi przez urząd masz możliwość pobrania kopii wiadomości e-mail i otwarcia jej w programie pocztowym np. Thunderbird.

5.5.1 Uprawnienia

System Obywatelskiego Fedrowania Danych umożliwia zarówno wysyłkę korespondencji, sporządzenia na nią odpowiedzi, a także podstawowe mechanizmy recenzji projektów wiadomości poprzez pozostawienie wiadomości do przejrzenia przez osobę uprawnioną do wysyłki wiadomości.

Istnieją następujące uprawnienia związane z obiegiem korespondencji:

Może odpowiadać
: uprawnia do bezpośredniej wysyłki wiadomości do urzędu, a także do zatwierdzenia istniejących projektów wiadomości

Może dodawać szkic odpowiedzi
: uprawnia do tworzenia w danym monitoringu w ramach spraw szkiców odpowiedzi, które nie są wysyłane do urzędu, ale mogą zostać zatwierdzone przez osobę o stosownych uprawnieniach

Może usuwać list
: uprawnia do usuwania dowolnych wiadomości zarejestrowanych w systemie, które może być także wykorzystane do odrzucania szkiców wiadomości

Może edytować list
: uprawnienia do edycji dowolnej wiadomości zarejestrowanej w systemie, także zarejestrowanej i pochodzącej od urzędu.

5.5.2 Sporządzanie odpowiedzi

Aby udzielić odpowiedzi na list zarejestrowany w systemie przejdź na stronę sprawy, gdzie został on opublikowany. Następnie kliknij w jego tytuł, aby przejdź na stronę danego listu. U góry dostępny jest przycisk “Odpowiedź”, który pozwala sporządzić tekstową odpowiedź do urzędu. Formularz (stosownie do uprawnień) może posiadać następujące przyciski zatwierdzania:

- Zapisz szkic - pozostawia daną wiadomość do przejrzenia przez użytkownika i nie wysyła jej do urzędu, jednak publikuje ją na stronie
- Wyślij odpowiedź - wysyła wiadomość do urzędu, który jest właściwy w danej sprawie.

Hint: Jeżeli opcja “Odpowiedź” jest niedostępna - skontaktuj się z operatorem monitoringu, aby uzyskać stosowne uprawnienia.

5.5.3 Dziennik wiadomości

W przypadku wychodzących wiadomości poczty elektronicznej rejestrowane są dane na temat transmisji wiadomości pochodzące z interfejsu programistycznego dostawcy usług Emaillabs.pl. Dane te są przez system automatycznie aktualizowane raz dziennie, co pozwala uzyskać informacje o stanie wiadomości wysłanych w Fedrowaniu.

Dzienniki możliwe są do przejrzenia z poziomu monitoringu i z poziomu sprawy. Aby się z nimi zapoznać przejdź do zakładki "Zobacz dzienniki" na odpowiedniej podstronie. Uzyskasz zestawienie wiadomości, które zawiera takie kolumny jak:

- ID - identyfikator wiadomości nadany przez E-maillabs,
- Sprawa - odwołanie do sprawy w jakiej dana wiadomość została wysłana,
- Status - ostatni poglądowy zarejestrowany status wiadomości,
- List - odwołanie do listu, który jest związany z daną wiadomością (jeżeli wykryto),
- Liczba wpisów - licznik wskazujący ile zmian dziennika odnotowano dla danej wiadomości.

Hint: Jeżeli opcja "Zobacz dzienniki" jest niedostępna - skontaktuj się z operatorem monitoringu, aby uzyskać stosowne uprawnienia.

Po wybraniu identyfikatora wiadomości prezentowany są surowe dane odnoszące się do przebiegu doręczenia danej wiadomości stanowiące odpowiedź API. Te dane mogą kilkakrotnie ulegać zmianie, gdyż operator pocztowy w przypadku chwilowych trudności może ponowić wysyłkę w późniejszym terminie.

Hint: Jeżeli zamierzasz wykorzystać dane dziennika wiadomości do celów dowodowych np. w sądzie zwróć się do administratora systemu o sporządzenie indywidualnej opinii na temat przebiegu doręczenia konkretnej wiadomości.

Wiadomości mogą uzyskać następujący status:

- Odrzucony z powodu spamu - nie dostarczona, gdyż serwer odmówił przyjęcia wiadomości z powodu zakwalifikowania jej jako spam,
- Dostarczony - skutecznie dostarczona do serwera pocztowego adresata,
- Miękko odrzucony - odrzucona z przyczyn przejściowych np. przepełniona skrzynka, a system ponowi wysyłkę,
- Odroczone - odrzucona z przyczyn tymczasowych np. wykorzystania graylistingu,
- Porzucone - nie udało się doręczyć z powodu utrzymujących się problemów tymczasowych,
- Otwarte - uzyskano potwierdzenie poprawnego otwarcia wiadomości np. poprzez wczytanie niewidocznego obrazka z wiadomości,
- Twardo odrzucony - nie udało się doręczyć wiadomości z powodu permanentnych problemów np. skrzynka pocztowa nie istnieje, domena internetowa nie istnieje,
- Nieznany - nie udało się poprawnie wykryć stanu wiadomości.

5.6 Rozpatrywanie zgłoszenia nadużyć

System gromadzi dane wprowadzone nie tylko przez administratorów i uprawnionych redaktorów, ale także przez podmioty zewnętrzne, w szczególności przez urzędy. Stwarza to możliwość zamieszczenia na infrastrukturze Stowarzyszenia treści, które naruszają prawa osób trzecich. Dotyczy to w szczególności sytuacji, gdy urżąd udostępnia dane, których nie poddaje skutecznej anonimizacji.

Każde przesłane zgłoszenie winno być wiarygodne tj. winna być możliwość ustalenia osoby od jakiej ono pochodzi. Forma przekazania takich informacji jest dowolna.

Po wpłynięciu takiego zawiadomienia należy dokonać identyfikacji jakie sprawy ono dotyczy. W przypadku plików wyszukujemy jego nazwę na stronie, a wtedy odnajdujemy stosowny list. Następnie przekazuje do podmiotu wnoszącego zgłoszenia stanowisko:

> Szanowny Panie,
>
> Serdecznie dziękuje za wiadomość.
>
> Wyjaśniam, że dane w takiej postaci zostały przekazane przez {{NazwaInstytucji}} ({{LinkDoSprawy}}). Stowarzyszenie publikuje dane urzędowe w sposób automatyczny i obiektywnie nie ma możliwości weryfikacji anonimizacji dokonanej przez urzędy, ani czy osoby, które są wskazane w dokumentach wyraziły na to zgodę. Procesu anonimizacji powinny dokonać podmioty, które dane udostępniają i dane chronione nie powinny podlegać przekazaniu Stowarzyszenia. Wszelkie uwagi w tym zakresie proszę kierować do właściwego urzędu, w tej sytuacji do {{NazwaUrzedu}}, aby zapobiec tego rodzaju sytuacjom w przyszłości.
>
> Oczywiście, rozumiem Pańską sytuacje i wskazane informacje zostaną wycofane DZIŚ z publikacji wraz z odpowiednią adnotacją o ingerencji w treść przekazaną przez Urząd. Naszym celem jest jawność życia publicznego, ale z poszanowaniem innych praw, także pańskiego prawa do prywatności.
>
> O wycofaniu publikacji zostanie Pan poinformowany w odrębnej korespondencji DZIŚ. Postaram się także doprowadzić, aby najszybciej jak to możliwe dokument w takiej postaci został usunięty z wyników wyszukiwarki Google.
>
> W razie dodatkowych pytań - pozostaje do dyspozycji.
>
> Z poważaniem, {{ImieNazwisko}} {{PelnionaFunkcja}}

Następnie przystępujemy do usuwania skutków zaistniałej sytuacji. Procedura usuwania jest następująca:

- odnotować ścieżki do plików, które zawierają kwestionowane dane,
- pobrać pliki i dokonać ich modyfikacji,
- dokonać edycji listu jako uprawniony użytkownik monitoringu, w zakresie:
  - podmienienie załączników z wykorzystaniem opcji “Zmień” lub usunięcia ich
  - usunięciu publikacji oryginalnej wiadomości,
  - uzupełnienie adnotacji o dokonanej anonimizacji.
- dokonać weryfikacji czy pierwotne adresy do plików nie wskazują już więcej na pliki bez anonimizacji,
- zgłosić adres do usunięcia w Google przy użyciu “Narzędzie do usuwania adresów URL”.

Po zakończeniu ww. czynności należy przesłać do podmiotu wnoszącego zgłoszenie stanowisko:

> Szanowny Panie,
>
> Zgodnie z wcześniejszą korespondencją uprzejmie informuje, że:
>
> - odpowiedzi zostały zanonimizowane wraz z stosownym oznaczeniem zmian,
> - stare pliki nie są dostępne ({{URL}}),

- wystąpiliśmy do Google o wyindeksowanie wcześniejszej odpowiedzi, jednak nie mam wpływu z jakim terminem zostanie to dokonane.

Proszę o potwierdzenie czy zaspokaja to Pana oczekiwania wobec Stowarzyszenia w związku z publikacją.

Z poważaniem,

5.7 Obsługa spamu

W okresie funkcjonowania systemu mogą wystąpić niepożądane sytuacje związane z dostarczaniem przez systemy informatyczne urzędu, albo – ze względu na publikacje adresów e-mail – inne podmioty niezamawianych informacji takich jak informacje handlowe, kartki świąteczne, których wartość informacyjna w konkretnej sprawie jest znikoma.

W celu obsługi tego rodzaju korespondencji został wprowadzony mechanizm zgłaszania spamu i oznaczania wiadomości jako spam.

5.7.1 Uprawnienia

System - w celu efektywnego rozłożenia zadań – wyposażony jest w mechanizm uprawnień. Osoba, która utworzyła monitoring ma możliwość zarządzania nim i dysponuje wszelkimi uprawnieniami do tego. Może nadawać i odbierać uprawnienia użytkownikom w danym monitoringu, a także nadawać im uprawnienia do takiego samego zarządzania.

Istnieją następujące uprawnienia związane z obsługą spamu:

Może widzieć dzienniki – `can_view_log`
uprawnia do zapoznania się z dziennikiem zgłoszeń w monitoringu

Może oznaczyć spam – `spam_mark`
uprawnia zapewniające dostęp do przycisku “Zgłoś spam” poprzez natychmiastowe ukrycie wiadomości

5.7.2 Proces obsługi

Na ekranie dowolnej wiadomości dostępny jest przycisk “Zgłoś spam”. Po jego wybraniu przez użytkownika niezalogowanego wiadomości trafiają do dziennika zgłoszeń.

Użytkownik, który posiada uprawnienie `can_view_log` otrzymuje powiadomienie o nowym wpisie w dzienniku zgłoszeń.

Użytkownik zalogowany, który posiada uprawnienie `mark_spam` po wybraniu przycisku “Oznacz spam” może ukryć wiadomość oznaczoną jako spam. Ewentualnie wiadomość zostanie oznaczona jako prawidłowa, a wówczas nie będzie możliwe ponowne zgłoszenie wiadomości jako spam. W obu przypadkach wpisy w dzienniku dotyczące danej wiadomości zostaną oznaczone jako załatwione.

Wiadomość oznaczona jako spam – ze względów dowodowych i potencjalnego przyszłego wykorzystania np. uczenie maszynowego automatycznego oznaczania podejrzanych wiadomości – nie jest całkowicie z systemu usunięta. Jest ona wyłącznie wycofywana z publikacji. Z tego też względu nie należy wprowadzać niezgodne z stanem faktycznym oznaczenia wiadomości jako spam, gdyż może to w przyszłości zakłócić maszynowe wnioskowanie.

5.7.3 Analiza bezpieczeństwa

Wiadomości, które są publikowane w systemie mogą zawierać złośliwe oprogramowanie, albowiem pochodzą od niezaufanych, zewnętrznych dostawców. Na dzień 10 lutego 2017 roku wiadomości są publikowane bez żadnej analizy antywirusowej.

Nawet w przypadku wprowadzenia takich mechanizmów - ze względu na niedoskonałość oprogramowania antywirusowego - będziemy w stanie wykryć wyłącznie wirusy poznane przez konkretny silnik antywirusowy.

W przypadku wiadomości zawierającej podejrzany załącznik:

1. **nie pobieraj, ani nie otwieraj pliku na komputerze**,
2. skopiuj adres URL pliku,
3. przejdź na VirusTotal do zakładki "URL"
4. wprowadź adres pliku i zatwierdź, aby uzyskać raport z badania adresu URL, który odnosi się do historii wiarygodności strony, ale nie treści,
5. w sekcji "Downloaded file" wybierz odnośnik, aby uzyskać raport z skanowania pliku przez wiele, niezależnych silników antywirusowych:
    1. w przypadku wyniku negatywnego - plik prawdopodobnie nie jest wirusem lub nie jest jeszcze znany oprogramowaniu antywirusowemu,
    2. w przypadku wyniku pozytywnego - niezwłocznie poinformuj Administratora Bezpieczeństwa Informacji oraz Administratora Systemu.

Administrator powinien:

1. zweryfikować nadesłane zgłoszenie,
2. zachować adres załączników i adres wiadomości,
3. podjąć działania, które uniemożliwią zapoznanie się z treścią i ochronią systemy informatyczne:
    1. przeanalizować nagłówki wiadomości w celu oceny celowości zgłoszenia do źródła przypadku nadużycia,
    2. usunąć wiadomość w ramach modułu `django_mailbox.Messages` wraz z załącznikiem `.eml` i załącznikami binarnymi,
    3. usunąć wiadomość w ramach modułu `letters.Letter` wraz z plikiem `.eml` i załącznikami binarnymi,
    4. zweryfikować np. programem `curl` czy pierwotny adres z załącznikami został skutecznie usunięty.

Należy zaznaczyć, że zagrażającym wiarygodności Stowarzyszenia jest sytuacja, gdy rozpowszechnia ono złośliwe oprogramowanie. W takim przypadku przeglądarki internetowe i firmy antywirusowe mogą oznaczyć wszystkie strony danego podmiotu jako niebezpieczne i uniemożliwić dostęp użytkownikom, co poważnie zakłóci realizacje podstawowych celów Stowarzyszenia.

5.8 Uprawnienia

System Obywatelskie Fedrowanie Danych był projektowany z założeniem wysokiej granulacji uprawnień w celu umożliwienia możliwie skutecznego powierzenia uprawnień użytkownikom o różnym stopniu odpowiedzialności.

Istnieją następujące globalne atrybuty użytkownika:

W zespole
: Określa czy użytkownik może zalogować się do panelu admina.

Status superużytkownika
: Oznacza, że ten użytkownik ma wszystkie uprawnienia bez jawnego ich przypisywania.

Jak również następujące globalne uprawnienia:

Can add Monitoring - `monitorings.add_monitoring`

Oznacza, że ten użytkownik może samodzielnie utworzyć nowy monitoring

- `letters.recognize_letter`
  Określa, że użytkownik może ręcznie rozpoznać list, który nie został przypisany do sprawy

5.8.1 Monitoring

Osoba, która utworzyła monitoring ma możliwość zarządzania nim i dysponuje niezbędnymi uprawnieniami do tego. Może nadawać i odbierać uprawnienia użytkownikom w danym monitoringu, a także nadawać im uprawnienia do takiego samego zarządzania.

'change_monitoring', 'delete_monitoring', 'add_case', 'change_case', 'delete_case', 'reply', 'add_draft', 'view_alert', 'change_alert', 'delete_alert', 'manage_perm',

Poszczególne uprawnienia są szczegółowo opisane w ramach właściwych części podręcznika użytkownika.

- **`monitorings.add_case` (domyślne dla autora monitoringu)**
  Określa, że użytkownik może dodawać nową sprawę
- **`monitorings.add_draft` (domyślne dla autora monitoringu)**
  Określa, że użytkownik może dodać szkic listu (bez wysyłania)
- **`monitorings.add_letter`**
  Określa, że użytkownik może dodać nowy list (bez wysyłania)
- **`monitorings.change_alert` (domyślne dla autora monitoringu)**
  Określa, że może edytować wpis w dzienniku
- **`monitorings.change_case`**
  Określa, że może dodać nową sprawę
- **`monitorings.change_letter`**
  Określa, że może edytować listy
- **`monitorings.change_monitoring` (domyślne dla autora monitoringu)**
  Określa, że może edytowac monitoring (jego opis, szablon wniosku itp.)
- **`monitorings.delete_alert` (domyślne dla autora monitoringu)**
  Określa, że może usuwać zadanie
- **`monitorings.delete_case` (domyślne dla autora monitoringu)**
  Określa, że może usuwać sprawę
- **`monitorings.delete_letter`**
  Określa, że może usuwać listy
- **`monitorings.delete_monitoring` (domyślne dla autora monitoringu)**
  Określa, że może usuwać monitoringi
- **`monitorings.manage_perm` (domyślne dla autora monitoringu)**
  Określa, że zarządzać uprawnieniami w monitoringu
- **`monitorings.reply` (domyślne dla autora monitoringu)**
  Określa, że wysyłać monitoring do wnioskodawcy
- **`monitorings.view_alert` (domyślne dla autora monitoringu)**
  Określa, że wyświetlać jeden wpis w dzienniku

- `monitorings.view_log`
  Określa, że wyświetlać dziennik wysyłki korespondencji e-mailowej

5.8.2 Instytucje

Osoba, która posiada właściwe uprawnienia może zarządzać katalogiem instytucji.

Powiązane z katalogiem uprawnienia to:

- `institutions.add_institution`
  Określa, że użytkownik może dodawać instytucje
- `institutions.change_institution`
  Określa, że użytkownik może usuwać instytucje
- `institutions.delete_institution`
  Określa, że użytkownik może edytować instytucje

5.9 Aktywowanie obsługi domeny

Niniejszy opis ma na celu przedstawienie w jaki sposób - bazując na stanie infrastruktury z dnia 26.11.2019 - należy aktywować obsługę nowej domeny do systemu Fedrowania.

Do poprawnego doręczenia wiadomości do Fedrowania konieczne jest:

- dodanie domeny do systemu Fedrowania,
- skonfigurowanie serwera pocztowego do obsługiwania domeny Fedrowania
- działanie procesu do importowania wiadomości z serwera pocztowego do systemu Fedrowania.
- skonfigurowanie serwera nazw do obsługiwania domeny

5.9.1 Dodanie domeny

Aby dodać nową domenę do systemu Fedrowania należy:

- zalogować się do systemu Fedrowania
- skorzystać z Panelu administracyjnego dostępnego pod adresem */admin/*
- przejść do zakładki *Domeny*
- dodać i oznaczyć jako aktywną nową domenę

5.9.2 Konfiguracja serwera pocztowego

Na dzień 26 listopada 2019 roku obsługę serwera pocztowego dla potrzeb Fedrowania zapewnia Zenbox. Realizowane jest to poprzez gromadzenie wiadomości w jednej skrzynce pocztowej.

Aby skonfigurować serwer pocztowego do obsługiwania domeny Fedrowania należy:

- zalogować się do Panelu Zarządzania Zenbox,
- dodać domenę do konta Zenbox,
- skonfigurować przekierowanie catch-all na adres *main@…*, zgodnie z domeną *siecobywatelska.pl*

5.9.3 Proces importowania wiadomości

Na dzień 26 listopada 2019 roku wykorzystywany jest współdzielony proces importowania wiadomości dla wszystkich domen Fedrowania, gdyż wykorzystywana jest - w poprzedniej sekcji - współdzielona skrzynka pocztowa.

Aby skonfigurować proces importowania należy zmodyfikować parametrykontenera i zastosować odpowiednią rolę Ansible ponownie.

Aby zmodyfikować proces importowania należy zmodyfikować kod_zrodlowy importera. Kod źródłowy z gałęzi branch jest automatycznie wdrażany.

5.9.4 Konfiguracja serwera nazw

Na dzień 26 listopada 2019 roku wiadomości są odbierane z wykorzystaniem serwera pocztowego zapewnionego przez Zenbox. Należy zapewnić poprawne skonfigurowanie rekordów MX w strefie DNS domeny, aby wiadomość docierała do serwera pocztowego Zenbox. Na dzień 26 listopada 2019 roku wiadomości są wysyłane z wykorzystaniem E-maillabs. Należy zapewnić poprawne skonfigurowanie DKIM, DMARC i SPF, aby wiadomości zostały uznane za wiarygodne.

Aby zarządzać rekordami DNS w strefach DNS Stowarzyszenia należy zmodyfikować repozytorium infra_terraform zgodnie z regułami repozytorium.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

G

H

I

J

L

M

N

P

R

S

T

U

W