
Password Manager

Wydanie 1.0.0

Patryk Jaworski, Jakub Jach

16 gru 2023

Contents:

1	backend	1
1.1	backend package	1
1.2	mainApp package	6
1.3	manage module	20
2	Indices and tables	23
	Indeks modułów Pythona	25
	Indeks	27

1.1 backend package

1.1.1 Submodules

1.1.2 backend.asgi module

ASGI config for projectBAI project.

It exposes the ASGI callable as a module-level variable named `application`.

For more information on this file, see <https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/>

```
"""
ASGI config for projectBAI project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'backend.settings')

application = get_asgi_application()
```

1.1.3 backend.settings module

Django settings for projectBAI project.

Generated by «django-admin startproject» using Django 4.2.4.

For more information on this file, see <https://docs.djangoproject.com/en/4.2/topics/settings/>

For the full list of settings and their values, see <https://docs.djangoproject.com/en/4.2/ref/settings/>

```
"""
Django settings for projectBAI project.

Generated by 'django-admin startproject' using Django 4.2.4.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""

from pathlib import Path
import environ

env = environ.Env()
environ.Env.read_env()

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = env('SECRET_KEY')

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'mainApp',
    'django_otp',
    'corsheaders',
    'django_cryptography',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```

'django.contrib.messages',
'django.contrib.staticfiles',
]

MIDDLEWARE = [
    #corsheaders.middleware.CorsMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'backend.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'backend.wsgi.application'

# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {

```

(ciąg dalszy na następnej stronie)

```
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'pl-pl'

TIME_ZONE = 'Europe/Warsaw'

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'

STATIC_ROOT = "/var/www/django/"

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

CORS_ALLOWED_ORIGINS = [
    "http://localhost:8000",
    "exp://127.0.0.1:8081"
]

ALLOWED_HOSTS = [*env('HOST').split(',')]
```

```
# SECURE_SSL_REDIRECT = True
# SECURE_PROXY_SSL_HEADER = ("HTTP_X_FORWARDED_PROTO", "https")
```


1.1.4 backend.urls module

URL configuration for projectBAI project.

The `urlpatterns` list routes URLs to views. For more information please see:

<https://docs.djangoproject.com/en/4.2/topics/http/urls/>

Examples: Function views

1. Add an import: from my_app import views
2. Add a URL to urlpatterns: `path(«», views.home, name=»home«)`

Class-based views

1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: `path(«», Home.as_view(), name=»home«)`

Including another URLconf

1. Import the `include()` function: from `django.urls` import `include`, `path`
2. Add a URL to urlpatterns: `path(«blog/», include(«blog.urls«))`

```
"""
URL configuration for projectBAI project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path("", views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path("", Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
from mainApp import views

urlpatterns = [
    path('main/', include('mainApp.urls')),
    path('admin/', admin.site.urls),
]
```

1.1.5 backend.wsgi module

WSGI config for projectBAI project.

It exposes the WSGI callable as a module-level variable named `application`.

For more information on this file, see <https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/>

```
"""
WSGI config for projectBAI project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'backend.settings')

application = get_wsgi_application()
```

1.1.6 Module contents

1.2 mainApp package

1.2.1 Subpackages

1.2.2 Submodules

1.2.3 mainApp.admin module

```
from django.contrib import admin
from .models import *

"""
Rejestruje modele Django w panelu admina.

Automatycznie rejestruje modele `UserProfile` i `CreditStorage` w panelu admina Django,
umożliwiając ich zarządzanie przez interfejs administracyjny.

:return: Brak zwracanej wartości.
"""
admin.site.register(UserProfile)
admin.site.register(CreditStorage)
```

1.2.4 mainApp.apps module

class mainApp.apps.MainappConfig(app_name, app_module)

Klasy bazowe: AppConfig

Konfiguracja aplikacji Django o nazwie «mainApp».

Klasa *MainappConfig* definiuje konfigurację głównej aplikacji Django o nazwie «mainApp». Zawiera parametr *default_auto_field*, który określa domyślne pole automatyczne dla modeli.

Zmienne

- **default_auto_field** (str) – Pole automatyczne używane jako domyślne dla modeli w aplikacji.
- **name** (str) – Nazwa aplikacji Django.

default_auto_field = 'django.db.models.BigAutoField'

name = 'mainApp'

```
from django.apps import AppConfig

class MainappConfig(AppConfig):
    """
    Konfiguracja aplikacji Django o nazwie 'mainApp'.

    Klasa `MainappConfig` definiuje konfigurację głównej aplikacji Django o nazwie
    ↪ 'mainApp'.
    Zawiera parametr `default_auto_field`, który określa domyślne pole automatyczne dla
    ↪ modeli.

    :ivar default_auto_field: Pole automatyczne używane jako domyślne dla modeli w
    ↪ aplikacji.
    :vartype default_auto_field: str
    :ivar name: Nazwa aplikacji Django.
    :vartype name: str
    """
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'mainApp'
```

1.2.5 mainApp.forms module

class mainApp.forms.UserForm(*args, **kwargs)

Klasy bazowe: UserCreationForm

Formularz rejestracji użytkownika.

Klasa *UserForm* dziedziczy po *UserCreationForm* i dodaje pole email. Używa modelu *User* i zawiera funkcję *save()*, która zapisuje użytkownika do bazy danych.

Zmienne

- **email** (forms.EmailField) – Pole email użytkownika.

class Meta

Klasy bazowe: object

fields = ('username', 'email', 'password1', 'password2')

model

alias of User

base_fields = {'email': <django.forms.fields.EmailField object>, 'password1': <django.forms.fields.CharField object>, 'password2': <django.forms.fields.CharField object>, 'username': <django.forms.fields.CharField object>}

declared_fields = {'email': <django.forms.fields.EmailField object>, 'password1': <django.forms.fields.CharField object>, 'password2': <django.forms.fields.CharField object>}

property media

Return all media required to render the widgets on this form.

save(commit=True)

Zapisuje użytkownika do bazy danych.

Funkcja *save()* zapisuje nowego użytkownika do bazy danych. Jeśli *commit* jest ustawione na *True*, użytkownik zostanie zapisany do bazy danych.

Parametry

commit (*bool*) – Flaga wskazująca, czy zapisać użytkownika do bazy danych. Domyślnie *True*.

Zwraca

Użytkownik zapisany w bazie danych.

Typ zwracany

User

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
from .models import UserProfile
from django_otp.plugins.otp_totp.models import TOTPDevice

class UserForm(UserCreationForm):
    """
    Formularz rejestracji użytkownika.

    Klasa `UserForm` dziedziczy po `UserCreationForm` i dodaje pole email.
    Używa modelu `User` i zawiera funkcję save(), która zapisuje użytkownika
    do bazy danych.

    :ivar email: Pole email użytkownika.
    :vartype email: forms.EmailField
    """
    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = ("username", "email", "password1", "password2")
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```

def save(self, commit=True):
    """
    Zapisuje użytkownika do bazy danych.

    Funkcja `save()` zapisuje nowego użytkownika do bazy danych.
    Jeśli `commit` jest ustawione na `True`, użytkownik zostanie zapisany
    do bazy danych.

    :param commit: Flaga wskazująca, czy zapisać użytkownika do bazy danych.
    ↪ Domyślnie True.
    :type commit: bool
    :return: Użytkownik zapisany w bazie danych.
    :rtype: User
    """
    user = super(UserForm, self).save(commit=False)
    user.email = self.cleaned_data['email']
    if commit:
        user.save()
    return user

```

1.2.6 mainApp.models module

class mainApp.models.CreditStorage(*args, **kwargs)

Klasy bazowe: Model

Przechowywanie danych związanego z zapisanymi serwisami.

Klasa *CreditStorage* definiuje model przechowujący dane związane z serwisami. Powiązany jest z modelem *User* reprezentującym użytkownika.

Zmienne

- **user** (*User*) – Powiązanie z modelem użytkownika Django.
- **name** (*str*) – Nazwa przechowywanych danych.
- **icon** (*dict*) – Ikona powiązana z danymi serwisu (w formacie JSON).
- **username** (*encrypt*) – Zaszyfrowane pole przechowujące nazwę użytkownika.
- **password** (*encrypt*) – Zaszyfrowane pole przechowujące hasło użytkownika.

exception DoesNotExist

Klasy bazowe: ObjectDoesNotExist

exception MultipleObjectsReturned

Klasy bazowe: MultipleObjectsReturned

icon

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

password

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

user_id**username**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class mainApp.models.**UserProfile**(*args, **kwargs)

Klasy bazowe: Model

Profil użytkownika aplikacji.

Klasa *UserProfile* definiuje profil użytkownika, powiązany z modelem wbudowanym *User*. Zawiera również pole *totp_device* reprezentujące urządzenie TOTP (Time-based One-Time Password).

Zmienne

- **user** (*User*) – Powiązanie z modelem użytkownika Django.
- **totp_device** (*TOTPDevice*) – Powiązanie z urządzeniem TOTP (opcjonalne).

exception DoesNotExist

Klasy bazowe: ObjectDoesNotExist

exception MultipleObjectsReturned

Klasy bazowe: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

totp_device

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

`totp_device_id`

user

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

`user_id`

```
from django.db import models
from django.contrib.auth.models import User
from django_otp.plugins.otp_totp.models import TOTPDevice
from django_cryptography.fields import encrypt

class UserProfile(models.Model):
    """
    Profil użytkownika aplikacji.

    Klasa `UserProfile` definiuje profil użytkownika, powiązany z modelem wbudowanym
    ↪ `User`.
    Zawiera również pole `totp_device` reprezentujące urządzenie TOTP (Time-based One-
    ↪ Time Password).

    :ivar user: Powiązanie z modelem użytkownika Django.
    :vartype user: User
    :ivar totp_device: Powiązanie z urządzeniem TOTP (opcjonalne).
    :vartype totp_device: TOTPDevice
    """
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    totp_device = models.OneToOneField(TOTPDevice, null=True, blank=True, on_
    ↪ delete=models.CASCADE)

class CreditStorage(models.Model):
    """
    Przechowywanie danych związanego z zapisanymi serwisami.

    Klasa `CreditStorage` definiuje model przechowujący dane związane z serwisami.
    Powiązany jest z modelem `User` reprezentującym użytkownika.

    :ivar user: Powiązanie z modelem użytkownika Django.
    :vartype user: User
    :ivar name: Nazwa przechowywanych danych.
    :vartype name: str
    :ivar icon: Ikona powiązana z danymi serwisu (w formacie JSON).
    :vartype icon: dict
    :ivar username: Zaszyfrowane pole przechowujące nazwę użytkownika.
```

(ciąg dalszy na następnej stronie)

```

:var type username: encrypt
:ivar password: Zaszyfrowane pole przechowujące hasło użytkownika.
:var type password: encrypt
"""

user = models.ForeignKey(User, on_delete=models.CASCADE)
name = models.CharField(max_length=32)
icon = models.JSONField(null=False)
username = encrypt(models.CharField(max_length=128))
password = encrypt(models.CharField(max_length=150))

```

1.2.7 mainApp.serializers module

class mainApp.serializers.CreditSerializer(*args, **kwargs)

Klasy bazowe: ModelSerializer

Serializer dla modelu *CreditStorage*.

Klasa *CreditSerializer* służy do serializacji modelu *CreditStorage*. Konwertuje dane modelu na format JSON do wykorzystania w interfejsach API.

class Meta

Klasy bazowe: object

fields = ['pk', 'name', 'icon', 'username', 'password']

model

alias of *CreditStorage*

```

from rest_framework.serializers import ModelSerializer
from .models import CreditStorage

class CreditSerializer(ModelSerializer):
    """
    Serializer dla modelu `CreditStorage`.

    Klasa `CreditSerializer` służy do serializacji modelu `CreditStorage`.
    Konwertuje dane modelu na format JSON do wykorzystania w interfejsach API.

    """
    class Meta:
        model = CreditStorage
        fields = ['pk', 'name', 'icon', 'username', 'password']

```


1.2.8 mainApp.tests module

```
from django.test import TestCase

# Create your tests here.
```

1.2.9 mainApp.urls module

```
from django.urls import path

from . import views

urlpatterns = [
    path("register/", views.register_view, name="register"),
    # Widok rejestracji nowego użytkownika

    path("login/", views.login_view, name="login"),
    # Widok logowania użytkownika

    path("logout/", view=views.user_logout, name="logout"),
    # Widok wylogowania użytkownika

    path("status/", view=views.status, name="status"),
    # Widok zwracający status uwierzytelnienia użytkownika

    path("authenticate/", views.authenticate_view, name="authenticate"),
    # Widok uwierzytelniania użytkownika z dwuetapową weryfikacją

    path("service/add/", view=views.add_service, name="add_service"),
    # Widok dodawania nowej usługi

    path("service/edit/<int:id>/", view=views.edit_service, name="edit_service"),
    # Widok edycji usługi

    path("services/list/", view=views.list_services, name="list"),
    # Widok listowania usług użytkownika

    path("services/delete/<int:id>/", view=views.delete_service, name="delete"),
    # Widok usuwania usługi
]
```

1.2.10 mainApp.views module

`mainApp.views.add_service(request: HttpRequest)`

Widok dodawania nowej usługi.

Dodaje nową usługę do bazy danych, wymaga uwierzytelnienia. Otrzymuje dane usługi (nazwę, użytkownika, hasło, ikonę) i zapisuje je do bazy danych.

Parametry

`request` – `HttpRequest`

Zwraca

JsonResponse z komunikatem o sukcesie lub błędzie

`mainApp.views.authenticate_view(request)`

Widok uwierzytelniania użytkownika z dwuetapową weryfikacją.

Obsługuje żądania typu POST zawierające dane uwierzytelniania i kod dwuetapowy TOTP. Dokonuje uwierzytelniania użytkownika i kodu TOTP, zwraca komunikat o sukcesie lub błędzie.

Parametry

request – HttpRequest

Zwraca

JsonResponse z komunikatem o sukcesie lub błędzie

`mainApp.views.delete_service(request: HttpRequest, id: int)`

Widok usuwania usługi.

Usuwa usługę z bazy danych na podstawie przekazanego identyfikatora. Wymaga uwierzytelnienia.

Parametry

- **request** – HttpRequest
- **id** – Identyfikator usługi do usunięcia

Zwraca

JsonResponse z komunikatem o sukcesie lub błędzie

`mainApp.views.edit_service(request: HttpRequest, id: int)`

Widok edycji usługi.

Zapisuje nowe dane do istniejącej usługi. Wymaga uwierzytelnienia.

Parametry

- **request** – Zapytanie HttpRequest
- **id** – Identyfikator usługi do modyfikacji

Zwraca

Informacja o powodzeniu operacji.

`mainApp.views.index(request)`

Widok wyświetlający prosty komunikat „Hello there from index!”.

Parametry

request – HttpRequest

Zwraca

HttpResponse z komunikatem

`mainApp.views.list_services(request: HttpRequest)`

Widok listowania usług użytkownika.

Zwraca listę usług użytkownika z bazy danych. Wymaga uwierzytelnienia.

Parametry

request – HttpRequest

Zwraca

JsonResponse z listą usług lub komunikatem o błędzie

`mainApp.views.login_view(request)`

Widok logowania użytkownika.

Obsługuje żądania typu POST zawierające dane logowania. Zwraca komunikat po pomyślnym zalogowaniu lub błąd przy nieudanym logowaniu.

Parametry

request – HttpRequest

Zwraca

JsonResponse z komunikatem o sukcesie lub błędzie

`mainApp.views.register_view(request)`

Widok rejestracji nowego użytkownika.

Obsługuje żądania typu POST zawierające dane niezbędne do rejestracji nowego użytkownika. Po sukcesie zwraca komunikat o rejestracji oraz wygenerowany sekret TOTP.

Parametry

request – HttpRequest

Zwraca

JsonResponse z komunikatem o sukcesie lub błędzie

`mainApp.views.status(request)`

Widok zwracający status uwierzytelnienia użytkownika.

Parametry

request – HttpRequest

Zwraca

JsonResponse z informacją o stanie uwierzytelnienia

`mainApp.views.user_logout(request)`

Widok wylogowania użytkownika.

Wylogowuje użytkownika z systemu.

Parametry

request – HttpRequest

Zwraca

JsonResponse z komunikatem o wylogowaniu

```
from django.http import JsonResponse, HttpResponseRedirect
from django.views.decorators.csrf import csrf_exempt
from django.views.decorators.http import require_http_methods, require_GET
from django.core.exceptions import ObjectDoesNotExist
from django.db.models import Q
from .models import UserProfile, CreditStorage
from .serializers import CreditSerializer
from django.contrib.auth.models import User
from django.contrib.auth import login, authenticate, logout
from django_otp.plugins.otp_totp.models import TOTPDevice

from base64 import b32encode

def index(request):
    """
```

(ciąg dalszy na następnej stronie)

```

Widok wyświetlający prosty komunikat "Hello there from index!".

:param request: HttpRequest
:return: HttpResponse z komunikatem
"""

return HttpResponse("Hello there from index!")

@csrf_exempt
def register_view(request):
    """
    Widok rejestracji nowego użytkownika.

    Obsługuje żądania typu POST zawierające dane niezbędne do rejestracji nowego
    ↪użytkownika.
    Po sukcesie zwraca komunikat o rejestracji oraz wygenerowany sekret TOTP.

    :param request: HttpRequest
    :return: JsonResponse z komunikatem o sukcesie lub błędzie
    """
    if request.method == 'POST':
        try:
            username = request.POST.get('username')
            email = request.POST.get('email')
            password1 = request.POST.get('password1')
            password2 = request.POST.get('password2')

            # check password
            if password1 != password2:
                return JsonResponse({'error': 'Passwords do not match', 'status': 400})

            # check username
            if User.objects.filter(username=username).exists():
                return JsonResponse({'error': 'Username is already taken', 'status': 401}
            ↪)

            # Create the user
            user = User.objects.create_user(username=username, email=email, ↪
            ↪password=password1)
            totp_device = TOTPDevice.objects.create(user=user, confirmed=False)
            totp_device.save()
            device = TOTPDevice.objects.get(user=user)
            secret_key = b32encode(device.bin_key).decode('utf-8')
            profile, created = UserProfile.objects.get_or_create(user=user)
            profile.totp_device = totp_device
            profile.save()
            return JsonResponse({'message': 'User registered successfully.', 'secret
            ↪':secret_key, 'status':200})

        except Exception as e:
            return JsonResponse({'error': str(e), 'status':500})

    else:

```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```

        return JsonResponse({'error': 'Invalid request method', 'status':405})

@csrf_exempt
def login_view(request):
    """
    Widok logowania użytkownika.

    Obsługuje żądania typu POST zawierające dane logowania.
    Zwraca komunikat po pomyślnym zalogowaniu lub błąd przy nieudanym logowaniu.

    :param request: HttpRequest
    :return: JsonResponse z komunikatem o sukcesie lub błędzie
    """
    if request.method == 'POST':
        try:
            username = request.POST.get('username')
            password = request.POST.get('password')

            user = authenticate(request, username=username, password=password)

            if user is None:
                return JsonResponse({'error':'Invalid username or password'}, status=400)

            return JsonResponse({'message':'All good in sucessfully'})
        except Exception as e:
            return JsonResponse({'error':str(e)}, status=500)
    return JsonResponse({'error': 'Invalid request method'},status=405)

@csrf_exempt
def authenticate_view(request):
    """
    Widok uwierzytelniania użytkownika z dwuetapową weryfikacją.

    Obsługuje żądania typu POST zawierające dane uwierzytelniania i kod dwuetapowy TOTP.
    Dokonuje uwierzytelniania użytkownika i kodu TOTP, zwraca komunikat o sukcesie lub
    ↪błędzie.

    :param request: HttpRequest
    :return: JsonResponse z komunikatem o sukcesie lub błędzie
    """
    if request.method == 'POST':
        try:
            username = request.POST.get('username')
            password = request.POST.get('password')
            totp_code = request.POST.get('password2FA')
            user = authenticate(request, username=username, password=password)
            totp_device = TOTPDevice.objects.get(user=user)
            if totp_device.verify_token(totp_code):
                login(request, user)
                return JsonResponse({'message': 'User logged in successfully', 'status':
                ↪200})
            else:

```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```

        return JsonResponse({'error': 'Invalid 2FA code', 'status':400})
    except Exception as e:
        return JsonResponse({'error':str(e), 'status':500}, )
    return JsonResponse({'error': 'Invalid request method', 'status':405})

@require_GET
def status(request):
    """
    Widok zwracający status uwierzytelnienia użytkownika.

    :param request: HttpRequest
    :return: JsonResponse z informacją o stanie uwierzytelnienia
    """
    return JsonResponse({"authenticated": request.user.is_authenticated})

@require_http_methods(["POST"])
@csrf_exempt
def add_service(request:HttpRequest):
    """
    Widok dodawania nowej usługi.

    Dodaje nową usługę do bazy danych, wymaga uwierzytelnienia.
    Otrzymuje dane usługi (nazwę, użytkownika, hasło, ikonę) i zapisuje je do bazy
    ↪ danych.

    :param request: HttpRequest
    :return: JsonResponse z komunikatem o sukcesie lub błędzie
    """
    if not request.user.is_authenticated:
        return JsonResponse({"error": "Please log in first"}, status=401)
    try:
        name = request.POST.get("name")
        username = request.POST.get("username")
        password = request.POST.get("password")
        icon = request.POST.get("icon")
        user = User.objects.filter(username=request.user.username).first()
        CreditStorage(
            user=user,
            name=name,
            username=username,
            password=password,
            icon=icon
        ).save()
        return JsonResponse({"message": "Credentials saved successfully!"})
    except Exception as e:
        print(e)
        return JsonResponse({"error": "Something went wrong when processing request"}, ↪
        ↪ status=500)

@require_http_methods(["POST"])
@csrf_exempt

```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```

def edit_service(request:HttpRequest, id:int):
    """
    Widok edycji usługi.

    Zapisuje nowe dane do istniejącej usługi.
    Wymaga uwierzytelnienia.

    :param request: Zapytanie HttpRequest
    :param id: Identyfikator usługi do modyfikacji
    :return: Informacja o powodzeniu operacji.
    """
    if not request.user.is_authenticated:
        return JsonResponse({"error": "Please log in first"}, status=401)
    try:
        name = request.POST.get("name")
        username = request.POST.get("username")
        password = request.POST.get("password")
        icon = request.POST.get("icon")

        credits = CreditStorage.objects.get(Q(pk=id) & Q(user=request.user))

        credits.name = name
        credits.username = username
        credits.password = password
        credits.icon = icon
        credits.save()
        return JsonResponse({"message": "Credentials changed successfully!"})
    except ObjectDoesNotExist:
        return JsonResponse({"error": "Requested credentials not found!"}, status=404)
    except Exception as e:
        print(e)
        return JsonResponse({"error": "Something went wrong!"}, status=500)

@require_http_methods(["POST"])
@csrf_exempt
def delete_service(request:HttpRequest, id:int):
    """
    Widok usuwania usługi.

    Usuwa usługę z bazy danych na podstawie przekazanego identyfikatora.
    Wymaga uwierzytelnienia.

    :param request: HttpRequest
    :param id: Identyfikator usługi do usunięcia
    :return: JsonResponse z komunikatem o sukcesie lub błędzie
    """
    if not request.user.is_authenticated:
        return JsonResponse({"error": "Please log in first"}, status=401)
    try:
        user = User.objects.filter(username=request.user.username).first()
        CreditStorage.objects.get(Q(pk=id) & Q(user=user.pk)).delete()

```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```

        return JsonResponse({"message": "Data deleted successfully"})
    except Exception as e:
        print(e)
        return JsonResponse({"error": "Something went wrong when processing request"},
↪ status=500)

@require_GET
def list_services(request:HttpRequest):
    """
    Widok listowania usług użytkownika.

    Zwraca listę usług użytkownika z bazy danych.
    Wymaga uwierzytelnienia.

    :param request: HttpRequest
    :return: JsonResponse z listą usług lub komunikatem o błędzie
    """
    if not request.user.is_authenticated:
        return JsonResponse({"error": "Please log in first"}, status=401)
    data = CreditStorage.objects.all().filter(user=request.user.pk)
    return JsonResponse({"services": CreditSerializer(data, many=True).data})

@require_http_methods(["POST"])
@csrf_exempt
def user_logout(request):
    """
    Widok wylogowania użytkownika.

    Wylogowuje użytkownika z systemu.

    :param request: HttpRequest
    :return: JsonResponse z komunikatem o wylogowaniu
    """
    logout(request)
    return JsonResponse({"message": "Logout successfull"})

```

1.2.11 Module contents

1.3 manage module

Django's command-line utility for administrative tasks.

`manage.main()`

Run administrative tasks.

```

#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""

```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
import os
import sys

def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'backend.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()
```

Indices and tables

- `genindex`
- `modindex`
- `search`

b

- `backend`, 6
- `backend.asgi`, 1
- `backend.settings`, 2
- `backend.urls`, 5
- `backend.wsgi`, 6

m

- `mainApp`, 20
- `mainApp.admin`, 6
- `mainApp.apps`, 7
- `mainApp.forms`, 7
- `mainApp.models`, 9
- `mainApp.serializers`, 12
- `mainApp.tests`, 13
- `mainApp.urls`, 13
- `mainApp.views`, 13
- `manage`, 20

A

`add_service()` (w module `mainApp.views`), 13
`authenticate_view()` (w module `mainApp.views`), 14

B

`backend`
 moduł, 6
`backend.asgi`
 moduł, 1
`backend.settings`
 moduł, 2
`backend.urls`
 moduł, 5
`backend.wsgi`
 moduł, 6
`base_fields` (`mainApp.forms.UserForm` atrybut), 8

C

`CreditSerializer` (klasa w module `mainApp.serializers`), 12
`CreditSerializer.Meta` (klasa w module `mainApp.serializers`), 12
`CreditStorage` (klasa w module `mainApp.models`), 9
`CreditStorage.DoesNotExist`, 9
`CreditStorage.MultipleObjectsReturned`, 9

D

`declared_fields` (`mainApp.forms.UserForm` atrybut), 8
`default_auto_field` (`mainApp.apps.MainappConfig` atrybut), 7
`delete_service()` (w module `mainApp.views`), 14

E

`edit_service()` (w module `mainApp.views`), 14

F

`fields` (`mainApp.forms.UserForm.Meta` atrybut), 8

`fields` (`mainApp.serializers.CreditSerializer.Meta` atrybut), 12

I

`icon` (`mainApp.models.CreditStorage` atrybut), 9
`id` (`mainApp.models.CreditStorage` atrybut), 9
`id` (`mainApp.models.UserProfile` atrybut), 10
`index()` (w module `mainApp.views`), 14

L

`list_services()` (w module `mainApp.views`), 14
`login_view()` (w module `mainApp.views`), 14

M

`main()` (w module `manage`), 20
`mainApp`
 moduł, 20
`mainApp.admin`
 moduł, 6
`mainApp.apps`
 moduł, 7
`mainApp.forms`
 moduł, 7
`mainApp.models`
 moduł, 9
`mainApp.serializers`
 moduł, 12
`mainApp.tests`
 moduł, 13
`mainApp.urls`
 moduł, 13
`mainApp.views`
 moduł, 13
`MainappConfig` (klasa w module `mainApp.apps`), 7
`manage`
 moduł, 20
`media` (`mainApp.forms.UserForm` property), 8
`model` (`mainApp.forms.UserForm.Meta` atrybut), 8

`model` (*mainApp.serializers.CreditSerializer.Meta* *atrybut*), 12

`moduł`

- `backend`, 6
 - `backend.asgi`, 1
 - `backend.settings`, 2
 - `backend.urls`, 5
 - `backend.wsgi`, 6
- `mainApp`, 20
 - `mainApp.admin`, 6
 - `mainApp.apps`, 7
 - `mainApp.forms`, 7
 - `mainApp.models`, 9
 - `mainApp.serializers`, 12
 - `mainApp.tests`, 13
 - `mainApp.urls`, 13
 - `mainApp.views`, 13
- `manage`, 20

N

`name` (*mainApp.apps.MainappConfig* *atrybut*), 7

`name` (*mainApp.models.CreditStorage* *atrybut*), 9

O

`objects` (*mainApp.models.CreditStorage* *atrybut*), 10

`objects` (*mainApp.models.UserProfile* *atrybut*), 10

P

`password` (*mainApp.models.CreditStorage* *atrybut*), 10

R

`register_view()` (*w module mainApp.views*), 15

S

`save()` (*mainApp.forms.UserForm* *metoda*), 8

`status()` (*w module mainApp.views*), 15

T

`totp_device` (*mainApp.models.UserProfile* *atrybut*), 10

`totp_device_id` (*mainApp.models.UserProfile* *atrybut*), 11

U

`user` (*mainApp.models.CreditStorage* *atrybut*), 10

`user` (*mainApp.models.UserProfile* *atrybut*), 11

`user_id` (*mainApp.models.CreditStorage* *atrybut*), 10

`user_id` (*mainApp.models.UserProfile* *atrybut*), 11

`user_logout()` (*w module mainApp.views*), 15

`UserForm` (*klasa w module mainApp.forms*), 7

`UserForm.Meta` (*klasa w module mainApp.forms*), 7

`username` (*mainApp.models.CreditStorage* *atrybut*), 10

`UserProfile` (*klasa w module mainApp.models*), 10

`UserProfile.DoesNotExist`, 10

`UserProfile.MultipleObjectsReturned`, 10