BAI_PRO_DEMO

Wydanie 1.0.0

Patryk Jaworski, Jakub Jach

24 lis 2023

Contents:

ROZDZIAŁ 1

backend

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 = 'django-insecure-emp8g=$&%83^zjr1#4uip$ujoh0j3kv)&khfj$_*1cc#0^wht@'

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

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'mainApp',
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
    'corsheaders',
    'django_otp',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    '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 = {
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
    '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 = [
    {
        '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/'
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
# 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_ALLOW_CREDENTIALS = True

CORS_ALLOW_HEADERS = [
    'access-control-allow-origin',
    'access-control-allow-credentials',
    'content-type',
    'accept',
    'x-csrftoken',
]

CSRF_TRUSTED_ORIGINS = [
    "http://localhost:5173"
]

CORS_ALLOWED_ORIGINS = [
    "http://localhost:5173"
]

EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = env('EMAIL_HOST')
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = env('EMAIL_HOST_USER')
EMAIL_HOST_PASSWORD = env('EMAIL_HOST_PASSWORD')
```

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/
"""
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
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 Submodules

1.2.2 mainApp.admin module

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

admin.site.register(Comment)
admin.site.register(UserProfile)

"""
Konfiguracja panelu administracyjnego.

Rejestruje modele `Comment` i `UserProfile` w panelu administracyjnym Django.
"""
```

1.2.3 mainApp.apps module

class mainApp.apps.**MainappConfig**(*app_name*, *app_module*)

Klasy bazowe: AppConfig

default_auto_field = 'django.db.models.BigAutoField'

name = 'mainApp'

```
from django.apps import AppConfig
"""
    Klasa konfiguracyjna aplikacji głównej.
"""
class MainappConfig(AppConfig):
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'mainApp'
```

1.2.4 mainApp.decorators module

mainApp.decorators.**require_login**(*func*, **args*, ***kwargs*)

Sprawdza, czy użytkownik jest zalogowany. Zwraca 401 z komunikatem błędu JSON, jeśli użytkownik nie jest zalogowany.

```
from django.http import HttpRequest, JsonResponse
def require_login(func, *args, **kwargs):
    """
    Sprawdza, czy użytkownik jest zalogowany. Zwraca 401 z komunikatem błędu␣
↪JSON, jeśli użytkownik nie jest zalogowany.

    """
    def _wrapper(request:HttpRequest):
        if request.user.is_authenticated:
            return func(request, *args, **kwargs)
        return JsonResponse({"error": "Please log in first!"}, status=401)
    return _wrapper
```

1.2.5 mainApp.forms module

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

Klasy bazowe: UserCreationForm

Formularz rejestracji użytkownika z możliwością włączenia uwierzytelniania dwuetapowego.

Dziedziczy po klasie UserCreationForm i dodaje pole *email* oraz *enable_2fa* do standardowego formularza rejestracji.

class Meta

Klasy bazowe: object

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

model

alias of User

base_fields = {'email': <django.forms.fields.EmailField object>, 'enable_2fa': <django.forms.fields.BooleanField 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>, 'enable_2fa': <django.forms.fields.BooleanField 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 oraz, jeśli włączono uwierzytelnianie dwuetapowe, tworzy i zapisuje urządzenie TOTP dla tego użytkownika.

Args: commit (bool): Określa, czy operacja zapisu ma zostać wykonana natychmiastowo (domyślnie True).

Returns: User: Obiekt użytkownika, który został zapisany.

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

class UserForm(UserCreationForm):
    """
    Formularz rejestracji użytkownika z możliwością włączenia uwierzytelniania␣
↪dwuetapowego.

    Dziedziczy po klasie UserCreationForm i dodaje pole `email` oraz `enable_2fa`␣
↪do standardowego formularza rejestracji.
    """
    email = forms.EmailField(required=True)
    enable_2fa = forms.BooleanField(
        required=False,
        widget=forms.CheckboxInput(attrs={'class': 'form-check-input'}),
        label='Enable Two-Factor Authentication'
    )

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

    def save(self, commit=True):
        """
        Zapisuje użytkownika oraz, jeśli włączono uwierzytelnianie dwuetapowe,
        tworzy i zapisuje urządzenie TOTP dla tego użytkownika.

        Args:
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
        commit (bool): Określa, czy operacja zapisu ma zostać wykonana␣
↪natychmiastowo (domyślnie True).

        Returns:
        User: Obiekt użytkownika, który został zapisany.
        """
        user = super(UserForm, self).save(commit=False)
        user.email = self.cleaned_data['email']
        if commit:
            user.save()

        if self.cleaned_data['enable_2fa']:
            totp_device = TOTPDevice.objects.create(user=user, confirmed=False)
            totp_device.save()
            profile = UserProfile.objects.get_or_create(user=user)
            profile.totp_device = totp_device
            profile.save()

        return user
```

1.2.6 mainApp.models module

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

Klasy bazowe: `Model`

Model reprezentujący komentarze.

Atrybuty: - title (str): Tytuł komentarza. - comment (str): Treść komentarza. - date_time (datetime): Data i czas utworzenia komentarza (domyślnie aktualna data i czas).

exception DoesNotExist

Klasy bazowe: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Klasy bazowe: `MultipleObjectsReturned`

comment

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

date_time

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

get_next_by_date_time(*, *field=<django.db.models.fields.DateTimeField: date_time>*, *is_next=True*, ***kwargs*)

get_previous_by_date_time(*, *field=<django.db.models.fields.DateTimeField: date_time>*, *is_next=False*, ***kwargs*)

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>

title

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`

Model reprezentujący profil użytkownika.

Atrybuty: - user (User): Odwołanie do modelu użytkownika (User). - totp_device (TOTPDevice): Odwołanie do modelu TOTPDevice (uwierzytelnianie dwuetapowe). - enable_2fa (bool): Flaga określająca, czy użytkownik włączył uwierzytelnianie dwuetapowe.

exception DoesNotExist

Klasy bazowe: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Klasy bazowe: `MultipleObjectsReturned`

enable_2fa

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.

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.utils.timezone import now
from django.contrib.auth.models import User
from django_otp.plugins.otp_totp.models import TOTPDevice

class Comment(models.Model):
    """
    Model reprezentujący komentarze.

    Atrybuty:
    - title (str): Tytuł komentarza.
    - comment (str): Treść komentarza.
    - date_time (datetime): Data i czas utworzenia komentarza (domyślnie␣
↪aktualna data i czas).
    """
    title = models.CharField(max_length=128)
    comment = models.TextField()
    date_time = models.DateTimeField(default=now)


class UserProfile(models.Model):
    """
    Model reprezentujący profil użytkownika.

    Atrybuty:
    - user (User): Odwołanie do modelu użytkownika (User).
    - totp_device (TOTPDevice): Odwołanie do modelu TOTPDevice (uwierzytelnianie␣
↪dwuetapowe).
    - enable_2fa (bool): Flaga określająca, czy użytkownik włączył␣
↪uwierzytelnianie dwuetapowe.
    """
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    totp_device = models.OneToOneField(TOTPDevice, null=True, blank=True, on_
↪delete=models.CASCADE)
    enable_2fa = models.BooleanField(default=False)
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
    def __str__(self):
        """
        Zwraca czytelny opis profilu użytkownika.
        """
        return f"UserProfile for {self.user.username}"
```

1.2.7 mainApp.serializers module

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

Klasy bazowe: ModelSerializer

Serializer dla modelu Comment.

Mapuje pola modelu Comment na format JSON i obsługuje walidację danych wejściowych.

Fields: - title (str): Tytuł komentarza. - date_time (datetime): Data i czas utworzenia komentarza. - comment (str): Treść komentarza.

class Meta

Klasy bazowe: object

fields = ['title', 'date_time', 'comment']

model

alias of *Comment*

```
from rest_framework.serializers import ModelSerializer
from .models import Comment

class CommentSerializer(ModelSerializer):
    """
    Serializer dla modelu Comment.

    Mapuje pola modelu Comment na format JSON i obsługuje walidację danych␣
↪wejściowych.

    Fields:
    - title (str): Tytuł komentarza.
    - date_time (datetime): Data i czas utworzenia komentarza.
    - comment (str): Treść komentarza.
    """
    class Meta:
        model = Comment
        fields = ['title', 'date_time', 'comment']
```

1.2.8 mainApp.tests module

```
from django.test import TestCase
```

1.2.9 mainApp.urls module

mainApp.urls.**urlpatterns = [<URLPattern 'register/' [name='register']>, <URLPattern 'login/' [name='login']>, <URLPattern 'logout/' [name='logout']>, <URLPattern 'islogin/'>, <URLPattern 'comments/'>, <URLPattern 'comments/gen/'>, <URLPattern 'js/'>, <URLPattern 'sqlgen/'>, <URLPattern 'sqldemo/'>, <URLPattern 'subscribe/'>]**

Moduł zawierający punkty końcowe (endpoints) dla aplikacji.

Endpoints: - /register/ : Endpoint do rejestracji użytkownika. - /login/ : Endpoint do logowania użytkownika. - /logout/ : Endpoint do wylogowania użytkownika. - /islogin/ : Endpoint do sprawdzenia stanu zalogowania. - /comments/ : Endpoint do wyświetlania komentarzy. - /comments/gen/ : Endpoint do generowania komentarzy. - /js/ : Endpoint do obsługi kompromitacji DOM (niebezpieczne). - /sqlgen/ : Endpoint do tworzenia próbek SQL (niebezpieczne). - /sqldemo/ : Endpoint do pobierania przykładowego SQL (niebezpieczne). - /subscribe/ : Endpoint do renderowania maila subskrypcyjnego.

```
from django.urls import path

from . import views

urlpatterns = [
    path("register/", views.register_view, name="register"),
    path("login/", views.login_view, name="login"),
    path("logout/", views.logout_view, name="logout"),
    path("islogin/", views.islogin),
    path("comments/", views.comments),
    path("comments/gen/", views.gen_comments),
    path("js/", views.domcompromise),
    path("sqlgen/", views.create_sample),
    path("sqldemo/", views.getsql),
    path("subscribe/", views.render_mail)
]

"""
Moduł zawierający punkty końcowe (endpoints) dla aplikacji.

Endpoints:
- /register/ : Endpoint do rejestracji użytkownika.
- /login/ : Endpoint do logowania użytkownika.
- /logout/ : Endpoint do wylogowania użytkownika.
- /islogin/ : Endpoint do sprawdzenia stanu zalogowania.
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
- /comments/ : Endpoint do wyświetlania komentarzy.
- /comments/gen/ : Endpoint do generowania komentarzy.
- /js/ : Endpoint do obsługi kompromitacji DOM (niebezpieczne).
- /sqlgen/ : Endpoint do tworzenia próbek SQL (niebezpieczne).
- /sqldemo/ : Endpoint do pobierania przykładowego SQL (niebezpieczne).
- /subscribe/ : Endpoint do renderowania maila subskrypcyjnego.
"""
```

1.2.10 mainApp.views module

mainApp.views.**domcompromise**(*request*)

Endpoint do demonstracji kompromitacji DOM.

Parametry
request – HttpRequest object

Zwraca
HttpResponse z kodem HTML

mainApp.views.**islogin**(*request*)

Sprawdza stan zalogowania użytkownika.

Parametry
request – HttpRequest object

Zwraca
JsonResponse z informacją o zalogowaniu

mainApp.views.**login_view**(*request*)

Loguje użytkownika.

Parametry
request – HttpRequest object

Zwraca
JsonResponse z komunikatem o sukcesie/błędzie

mainApp.views.**logout_view**(*request*)

Wylogowuje użytkownika.

Parametry
request – HttpRequest object

Zwraca
JsonResponse z komunikatem o wylogowaniu

mainApp.views.**register_view**(*request*)

Rejestruje nowego użytkownika.

Parametry
request – HttpRequest object

Zwraca

JsonResponse z komunikatem o sukcesie/błędzie

```
from django.http import JsonResponse, HttpResponse, HttpRequest
from django_otp.plugins.otp_totp.models import TOTPDevice
from django.views.decorators.csrf import csrf_exempt, ensure_csrf_cookie
from django.contrib.auth import login, authenticate, logout
from django.views.decorators.http import require_http_methods
from django.utils.dateparse import parse_datetime
from .forms import UserForm
from django.template.loader import get_template
from django.core.mail import EmailMultiAlternatives
from backend import settings
from traceback import format_exc
from .serializers import CommentSerializer
from .models import Comment, UserProfile
from .decorators import require_login
import sqlite3
import qrcode
from io import BytesIO
from base64 import b64encode

@require_http_methods(['POST'])
def register_view(request):
    """
    Rejestruje nowego użytkownika.

    :param request: HttpRequest object
    :return: JsonResponse z komunikatem o sukcesie/błędzie
    """
    try:
        username = request.POST.get('username')
        email = request.POST.get('email')
        password1 = request.POST.get('password1')
        password2 = request.POST.get('password2')
        enable_2fa = request.POST.get('enable_2fa') == 'true'

        form = UserForm({
            'username': username,
            'email': email,
            'password1': password1,
            'password2': password2
        })

        if form.is_valid():
            user = form.save()
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
            if enable_2fa:
                profile, created = UserProfile.objects.get_or_create(user=user)

                if not profile.totp_device:
                    totp_device = TOTPDevice.objects.create(user=user,␣
↪confirmed=False)
                    totp_device.save()
                    profile.totp_device = totp_device
                    profile.enable_2fa = enable_2fa
                    profile.save()

                    qr = qrcode.QRCode(
                        version=1,
                        error_correction=qrcode.constants.ERROR_CORRECT_L,
                        box_size=10,
                        border=0
                    )
                    qr.add_data(totp_device.config_url)
                    qr.make(fit=True)
                    qr_code_img = qr.make_image(fill_color=(66, 184, 131), back_
↪color=(18,18,18))
                    buffer = BytesIO()
                    qr_code_img.save(buffer)
                    buffer.seek(0)
                    encoded_img = b64encode(buffer.read()).decode()
                    qr_code_data = f'{encoded_img}'

                    response_data = {
                    'message': 'User registered successfully',
                    'image_data': qr_code_data
                    }
                else:
                    response_data = {'message': 'User registered successfully␣
↪(2FA was already enabled)'}
            else:
                response_data = {'message': 'User registered successfully (2FA␣
↪is disabled)'}
        else:
            return JsonResponse({'error': form.errors}, status=400)

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


@require_http_methods(['POST'])
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
def login_view(request):
    """
    Loguje użytkownika.

    :param request: HttpRequest object
    :return: JsonResponse z komunikatem o sukcesie/błędzie
    """
    try:
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)

        if user is not None:
            requires_2fa = False

            try:
                user_profile = UserProfile.objects.get(user=user)
                if user_profile.enable_2fa:
                    requires_2fa = True
            except UserProfile.DoesNotExist:
                pass

            if requires_2fa:
                totp_code = request.POST.get('totp_code')
                if totp_code:
                    try:
                        totp_device = TOTPDevice.objects.get(user=user)
                        if totp_device.verify_token(totp_code):
                            login(request, user)
                            return JsonResponse({'message': 'User logged in␣
↪successfully'})
                        else:
                            return JsonResponse({'error': 'Invalid 2FA code'},␣
↪status=400)
                    except TOTPDevice.DoesNotExist:
                        return JsonResponse({'error': '2FA device not found'},␣
↪status=400)
                else:
                    return JsonResponse({'requires_2fa': True})
            else:
                login(request, user)
                return JsonResponse({'message': 'User logged in successfully'})
        else:
            return JsonResponse({'error': 'Invalid username or password'},␣
↪status=400)
    except Exception as e:
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
        return JsonResponse({'error': str(e)}, status=500)


@require_http_methods(["POST"])
def logout_view(request):
    """
    Wylogowuje użytkownika.

    :param request: HttpRequest object
    :return: JsonResponse z komunikatem o wylogowaniu
    """
    logout(request)
    return JsonResponse({'message':'User logged out'})


@require_http_methods(['GET'])
@ensure_csrf_cookie
def islogin(request):
    """
    Sprawdza stan zalogowania użytkownika.

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


@require_http_methods(['GET'])
@require_login
def comments(request:HttpRequest):
    """
    Zwraca listę komentarzy.

    :param request: HttpRequest object
    :return: JsonResponse z listą komentarzy
    """
    return JsonResponse({"comments": CommentSerializer(Comment.objects.all(),␣
↪many=True).data})


@require_http_methods(['POST'])
@csrf_exempt
@require_login
def gen_comments(request):
    """
    Generuje przykładowe komentarze.
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
    :param request: HttpRequest object
    :return: JsonResponse z komunikatem o wygenerowanych komentarzach
    """
    if Comment.objects.count() >= 5:
        return JsonResponse({"message": "Already created"})
    Comment(
        title = 'Simple comment',
        comment = 'This is a great example of normal comment',
        date_time = parse_datetime('2023-06-12T18:22:00+02:00')
    ).save()
    Comment(
        title = 'Unharmful comment',
        comment = 'This is a great example of unharmful <strong>comment</strong>␣
↪with injected HTML &lt;strong&gt; tag.',
        date_time = parse_datetime('2023-08-15T15:32:10+02:00')
    ).save()
    Comment(
        title = 'Somewhat dangerous comment',
        comment = 'This is a great example of dangerous comment <iframe width=
↪"560" height="315" src="https://www.youtube.com/embed/dQw4w9WgXcQ?
↪si=O7ynaNQw9GcNtXud" title="YouTube video player" frameborder="0" allow=
↪"accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-
↪in-picture; web-share" allowfullscreen></iframe> with injected HTML &lt;iframe&
↪gt; tag.',
        date_time = parse_datetime('2023-09-08T22:40:31+02:00')
    ).save()
    Comment(
        title = 'Pretty dangerous comment',
        comment = 'This is a great example of dangerous comment with <img src␣
↪onerror="alert(\'injected <script> tag\')"/>',
        date_time = parse_datetime('2023-10-09T00:20:15+02:00')
    ).save()
    Comment(
        title = 'Extremly dangerous comment',
        comment = '<img src onerror="fetch(\'http://localhost:8000/main/js/\').
↪then(response => response.text()).then( text => document.write(text)).
↪catch(error => { })">',
        date_time = parse_datetime('2023-10-12T21:37:00+02:00')
    ).save()
    return JsonResponse({"message": "Comments generated successfully"})


@require_http_methods(["GET"])
# No @require_login -> We simulate hackers website where no login is required.
def domcompromise(request):
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
    """
    Endpoint do demonstracji kompromitacji DOM.

    :param request: HttpRequest object
    :return: HttpResponse z kodem HTML
    """
    page = get_template("dom.html")
    return HttpResponse(page.render())


@require_http_methods(["POST"])
@require_login
def create_sample(request):
    """
    Tworzy przykładową bazę danych.

    :param request: HttpRequest object
    :return: JsonResponse z komunikatem o sukcesie/błędzie
    """
    try:
        file = open("start.sql", "r")
        conn = sqlite3.connect('sample.sqlite3')
        cursor = conn.cursor()
        cursor.executescript(file.read())
        cursor.close()
        conn.close()
        file.close()
        return JsonResponse({"message": "Sample database created successfully"})
    except Exception as e:
        return JsonResponse({"error": str(e)}, status=500)


@require_http_methods(["GET"])
@require_login
def getsql(request:HttpRequest):
    """
    Zwraca dane z bazy danych.

    :param request: HttpRequest object
    :return: JsonResponse z danymi z bazy
    """
    try:
        conn = sqlite3.connect('sample.sqlite3')
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        result = None
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
        name = request.GET['name'] if 'name' in request.GET else ''
        if 'safe' in request.GET and request.GET['safe'] == 'true':
            result = cursor.execute("""
            SELECT
                name,
                price,
                thumbnail
            FROM
                products
            WHERE
                name LIKE ?
            """,
                (f'%{name}%',)
            )
        else:
            result = cursor.execute(f"SELECT name, price, thumbnail FROM␣
↪products WHERE name LIKE '%{name}%'")
        # IMPORTANT #
        # WE ASSUME DEV IS LAZY AND WANTED TO GET EVERY COLUMN FROM QUERY AND␣
↪RETURN IT TO USER #
        products = [{item: row[item] for item in row.keys()} for row in result]
        cursor.close()
        conn.close()
        return JsonResponse({"products": products, "query": f"SELECT name, price,
↪ thumbnail FROM products WHERE name LIKE '%{name}%'"})
    except Exception as e:
        print(e)
        return JsonResponse({"error": str(e), "stack": format_exc()}, status=500)


# @require_http_methods(['POST']) # LAZY DEV DIDN'T ADD REQUIRED METHOD
@require_login # CSRF attack will be successfull - web browser sends cookies by␣
↪default.
def render_mail(request:HttpRequest):
    """
    Renderuje i wysyła maila subskrypcyjnego.

    :param request: HttpRequest object
    :return: JsonResponse z komunikatem o wysłaniu maila
    """
    if not request.user.is_authenticated:
        return JsonResponse({"error": "Unauthorized. Please log in first!"},␣
↪status=401)
    try:
        html = get_template('email.html')
        txt = get_template('email.txt')
```

(ciąg dalszy na następnej stronie)

(kontynuacja poprzedniej strony)

```
        subject, _from, to = 'Important information', settings.EMAIL_HOST_USER,␣
↪request.user.email
        html_content = html.render({'username': request.user.username})
        text_content = txt.render({'username': request.user.username})
        msg = EmailMultiAlternatives(subject, text_content, _from, [to])
        msg.attach_alternative(html_content, "text/html")
        msg.send()
    except Exception as e:
        print(e)
        return JsonResponse({"error": "Unable to send email!"}, status=500)

    return JsonResponse({"message": "Email sent successfully!"})
```

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."""
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()
```

ROZDZIAŁ 2

Indices and tables

- genindex
- modindex
- search

Indeks modułów Pythona

b

m

Indeks

N

O

R

S

T

U