Dokumentacja kodu

ROZDZIAŁ 1

accounts/apps.py

class accounts.apps.**AccountsConfig**(*app_name*, *app_module*)

Klasy bazowe: AppConfig

Konfiguracja aplikacji Django accounts.

Określa nazwę aplikacji i domyślny typ klucza głównego dla modeli.

Atrybuty:

default_auto_field (str): Domyślny typ pola dla automatycznego klucza głównego. name (str): Nazwa aplikacji Django.

default_auto_field = 'django.db.models.BigAutoField'

name = 'accounts'

ROZDZIAŁ 2

accounts/forms.py

2.1 Formularz rejestracji użytkownika

class accounts.forms.**UserRegistrationForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

Klasy bazowe: `ModelForm`

Formularz rejestracji nowego użytkownika.

Rozszerza klasę `ModelForm` opartą o model `User` i umożliwia wprowadzenie hasła oraz jego potwierdzenia. Zastosowano walidację siły hasła za pomocą *validate_password*.

Pola formularza: - `username` – nazwa użytkownika (z modelu User) - `email` – adres email użytkownika (z modelu User) - `password` – hasło główne (pole typu `PasswordInput`) - `password2` – powtórzenie hasła (pole typu `PasswordInput`)

Walidacja: - Sprawdza, czy oba hasła są zgodne - Weryfikuje siłę hasła za pomocą walidatora Django

class Meta

Klasy bazowe: `object`

fields = ('username', 'email')

model

alias of `User`

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

clean()

Sprawdza, czy hasło i jego powtórzenie są identyczne. W przeciwnym razie zgłasza błąd walidacji.

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

property media

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

2.2 Formularz logowania użytkownika

class accounts.forms.**LocalUserLoginForm**(*request=None*, **args*, ***kwargs*)

Klasy bazowe: `AuthenticationForm`

Formularz logowania dla lokalnych użytkowników.

Rozszerza wbudowany formularz `AuthenticationForm`, dodając własne etykiety dla pól logowania.

Pola formularza: - `username` – nazwa użytkownika - `password` – hasło użytkownika

base_fields = {'password': <django.forms.fields.CharField object>, 'username': <django.forms.fields.CharField object>}

declared_fields = {'password': <django.forms.fields.CharField object>, 'username': <django.forms.fields.CharField object>}

property media

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

ROZDZIAŁ 3

accounts/views.py

3.1 Widoki funkcjonalne

accounts.views.**register_view**(*request*)

Widok rejestracji nowego użytkownika z e-mailowym potwierdzeniem.

Jeśli formularz rejestracyjny jest poprawny: - Tworzy użytkownika w stanie nieaktywnym. - Wysyła e-mail aktywacyjny z linkiem zawierającym UID i token. - Wyświetla stronę potwierdzenia wysłania wiadomości.

Parametry
request – Obiekt żądania HTTP

Zwraca
Strona z formularzem lub komunikatem o wysłaniu maila

accounts.views.**activate_account**(*request*, *uid*, *token*)

Aktywuje konto użytkownika po kliknięciu w link z wiadomości e-mail.

Jeśli UID i token są poprawne: - Aktywuje konto (ustawia `is_active=True`). - Przekierowuje na stronę profilu.

Parametry

- **request** – Obiekt żądania HTTP
- **uid** – Zakodowany identyfikator użytkownika
- **token** – Token aktywacyjny

Zwraca
Przekierowanie na stronę profilu

accounts.views.**logout_view**(*request*)

Wylogowuje użytkownika i przekierowuje na stronę logowania.

Parametry
request – Obiekt żądania HTTP

Zwraca
Przekierowanie na login

accounts.views.**user_home_view**(*request*)

Widok dostępny tylko dla zalogowanych użytkowników.

Przekierowuje do profilu użytkownika po zalogowaniu.

Parametry
request – Obiekt żądania HTTP

Zwraca
Przekierowanie na stronę profilu

accounts.views.**confirm_disable_2fa**(*request*, *uid*, *token*)

Potwierdzenie dezaktywacji 2FA przez kliknięcie w link z e-maila.

Jeśli token jest poprawny: - Usuwa urządzenia OTP przypisane do użytkownika.

Parametry

- **request** – Obiekt żądania HTTP
- **uid** – Zakodowany identyfikator użytkownika
- **token** – Token potwierdzający

Zwraca
Przekierowanie na stronę profilu

3.2 Widoki klasowe

class accounts.views.**CustomLoginView**(***kwargs*)

Niestandardowy widok logowania.

Jeśli użytkownik jest już zalogowany, przekierowuje go na stronę profilu.

dispatch(*request*, **args*, ***kwargs*)

This method gets called by the routing engine. The first argument is *request* which contains a *HttpRequest* instance. The request is stored in *self.request* for later use. The storage instance is stored in *self.storage*.

After processing the request using the *dispatch* method, the response gets updated by the storage engine (for example add cookies).

class accounts.views.**CustomDisableView**(***kwargs*)

Widok formularza do wyłączenia 2FA z e-mailowym potwierdzeniem.

- Użytkownik wypełnia formularz wyłączenia.
- Na e-mail wysyłany jest link potwierdzający.

form_class

alias of `DisableForm`

form_valid(*form*)

Wysłanie wiadomości e-mail z linkiem potwierdzającym wyłączenie 2FA.

Parametry
form – Wypełniony formularz DisableForm

Zwraca
Strona z komunikatem o wysłaniu wiadomości

ROZDZIAŁ 4

accounts/urls.py

Plik definiuje ścieżki URL obsługujące logowanie, rejestrację i zarządzanie dwuetapową weryfikacją (2FA) w aplikacji użytkownika.

Dostępne ścieżki: - Logowanie (z 2FA) - Rejestracja nowego użytkownika - Wylogowanie - Aktywacja konta przez e-mail - Dezaktywacja uwierzytelnienia dwuskładnikowego

accounts.urls.**urlpatterns = [<URLPattern 'login/' [name='two_factor:login']>, <URLPattern 'register/' [name='register']>, <URLPattern 'logout/' [name='logout']>, <URLPattern 'activate/<uid>/<token>/' [name='activate']>, <URLPattern 'two_factor/disable/' [name='two_factor:disable']>, <URLPattern 'two_factor/disable-2fa-confirm/<uid>/<token>/' [name='disable_confirm']>]**

Lista ścieżek URL obsługiwanych przez aplikację accounts

4.1 Opis ścieżek

- **/login/** → widok logowania (z 2FA)
- **/register/** → formularz rejestracyjny
- **/logout/** → wylogowanie użytkownika
- **/activate/<uid>/<token>/** → aktywacja konta przez e-mail
- **/two_factor/disable/** → wyłączenie 2FA (krok 1)
- **/two_factor/disable-2fa-confirm/<uid>/<token>/** → potwierdzenie wyłączenia 2FA (krok 2)

ROZDZIAŁ 5

accounts/tests.py

5.1 Klasa AccountsViewsTest

class accounts.tests.**AccountsViewsTest**(*methodName='runTest'*)

Klasy bazowe: `TestCase`

Klasa testowa sprawdzająca poprawne działanie widoków w aplikacji `accounts`.

Używa wbudowanego klienta testowego Django do wykonywania żądań HTTP. Sprawdza między innymi proces rejestracji, logowania, wylogowywania oraz dostęp do stron wymagających uwierzytelnienia.

Zmienne

- **client** – Instancja `django.test.Client` używana do wykonywania żądań HTTP w trakcie testów
- **register_url** – Adres URL dla rejestracji użytkownika
- **login_url** – Adres URL dla logowania (w tym przypadku obsługiwany przez pakiet two_factor)
- **logout_url** – Adres URL dla wylogowania użytkownika
- **profile_url** – Adres URL do profilu użytkownika
- **user_home_url** – Adres URL do strony głównej użytkownika
- **admin_login_url** – Adres URL do logowania administratora przez Google OAuth2
- **user** – Użytkownik testowy utworzony na potrzeby testów

setUp()

Przygotowanie wstępne przed każdym testem.

Tworzy klienta testowego, podstawowe adresy URL oraz użytkownika testowego.

test_logout_view()

Test wylogowania użytkownika.

Sprawdza, czy po wylogowaniu użytkownik zostaje przekierowany na stronę logowania (kod statusu 302).

`test_register_view_email_send()`

Test wysłania emaila aktywacyjnego po rejestracji.

Sprawdza:

- czy użytkownik po rejestracji jest nieaktywny (*is_active=False*),
- czy w skrzynce nadawczej pojawiła się wiadomość e-mail,
- czy temat oraz treść wiadomości zawierają oczekiwane informacje, takie jak link aktywacyjny i identyfikator użytkownika.

`test_register_view_invalid()`

Test rejestracji z błędnymi danymi.

Sprawdza, czy w przypadku niepoprawnego uzupełnienia formularza rejestracyjnego (np. brak nazwy użytkownika, różne hasła), użytkownik nie zostanie utworzony w bazie.

`test_register_view_password_validation()`

Test walidacji hasła podczas rejestracji.

Przykład weryfikujący odrzucenie zbyt słabych haseł: - krótkich - zbyt oczywistych, jak `password` - używających nazwy użytkownika

`test_register_view_success()`

Test poprawnej rejestracji użytkownika.

Sprawdza, czy użytkownik z prawidłowymi danymi rejestracyjnymi zostaje zapisany w bazie danych oraz czy otrzymuje właściwy kod statusu (200).

`test_user_home_view_authenticated()`

Test dostępu do strony użytkownika po zalogowaniu.

Sprawdza, czy zalogowany użytkownik zostanie przekierowany na stronę profilu (kod statusu 302).

`test_user_home_view_unauthenticated()`

Test dostępu do strony użytkownika bez zalogowania.

Sprawdza, czy niezalogowany użytkownik zostanie przekierowany na stronę logowania i czy w parametrach przekierowania znajduje się właściwy adres docelowy (`next=...`).

ROZDZIAŁ 6

accounts/pipeline.py

6.1 Funkcje pipeline

accounts.pipeline.**verify_staff_status**(*backend*, *user*, *response*, **args*, ***kwargs*)

Zezwala tylko użytkownikom z uprawnieniami administracyjnymi na logowanie przez Google OAuth.

Sprawdza, czy użytkownik jest oznaczony jako `is_staff` i `is_superuser`. W przeciwnym razie następuje przekierowanie na stronę profilu (lub logowania) bez dostępu.

Parametry

- **backend** – Obiekt backendu autoryzacji (np. GoogleOAuth2)
- **user** – Obiekt użytkownika Django
- **response** – Surowa odpowiedź z serwera OAuth (zawiera dane o użytkowniku)

Zwraca

`redirect()` w przypadku braku uprawnień lub `None`

accounts.pipeline.**social_uid**(*backend*, *details*, *response*, **args*, ***kwargs*)

Zwraca niestandardowy identyfikator użytkownika w zależności od użytego providera.

Obsługiwani providerzy: - Microsoft Graph: zwraca `userPrincipalName` - Facebook: zwraca `email` - Domyślnie: używa metody `get_user_id` z backendu

Parametry

- **backend** – Obiekt backendu autoryzacji
- **details** – Dane podstawowe o użytkowniku
- **response** – Surowa odpowiedź od providera

Zwraca

Słownik z kluczem `uid` zawierającym identyfikator użytkownika jako string

ROZDZIAŁ 7

bezpieczenstwo_chmurowe/urls.py

Plik odpowiada za główną konfigurację URL projektu Django.

Zawiera: - przekierowanie do logowania admina przez 2FA, - podłączenie ścieżek z aplikacji `accounts`, - integrację z `django-two-factor-auth` oraz `social-auth-app-django`, - ścieżkę domyślną i rezerwową (catch-all).

bezpieczenstwo_chmurowe.urls.**redirect_admin**(*request*)

Przekierowuje próbę logowania do panelu admina na stronę logowania z 2FA.

Parametry
request – Obiekt żądania HTTP

Zwraca
Przekierowanie do widoku logowania (two_factor:login)

bezpieczenstwo_chmurowe.urls.**urlpatterns = [<URLPattern '' [name='home']>, <URLPattern 'admin/login/' [name='admin_redirect']>, <URLResolver <URLPattern list> (admin:admin) 'admin/'>, <URLResolver <module 'social_django.urls' from '/home/franciszek/Dropbox/franciszek-pc/home/Python/sggw_py/bezpieczenstwo_chmurowe/.venv/lib/python3.13/site-packages/social_django/urls.py'> (social:social) 'oauth/'>, <URLResolver <module 'accounts.urls' from '/home/franciszek/Dropbox/franciszek-pc/home/Python/sggw_py/bezpieczenstwo_chmurowe/accounts/urls.py'> (None:None) 'account/'>, <URLResolver <URLPattern list> (two_factor:two_factor) ''>, <URLPattern '^.*$' [name='catch_all']>]**

Lista głównych ścieżek URL projektu Django

7.1 Opis ścieżek

- **/** → strona domowa po zalogowaniu użytkownika (przekierowuje do profilu)
- **/admin/login/** → przekierowanie logowania admina do logowania z 2FA
- **/admin/** → panel administracyjny Django
- **/oauth/** → logowanie społecznościowe (Google, Microsoft, Facebook itp.)

- **/account/** → podłączenie ścieżek z aplikacji „accounts”
- **ścieżki z django-two-factor-auth** → logowanie z 2FA, konfiguracja urządzeń itp.
- **catch-all (`^.*$`)** → dowolna inna ścieżka przekierowuje na stronę domową użytkownika

ROZDZIAŁ 8

bezpieczenstwo_chmurowe/settings.py

Plik zawiera konfigurację projektu Django, w tym:

- ustawienia środowiskowe z pliku *.env*,
- ścieżki do aplikacji i middleware,
- integrację z systemem uwierzytelniania społecznościowego (Google, Microsoft, Facebook),
- konfigurację django-two-factor-auth,
- ustawienia poczty, baz danych i statycznych plików.

Zmienne środowiskowe pobierane są przy użyciu *python-dotenv*.

8.1 Opis ustawień

Środowisko i bezpieczeństwo - `SECRET_KEY` – klucz aplikacji, ładowany z *.env* - `DEBUG` – tryb debugowania (`True`/`False`) - `ALLOWED_HOSTS` – lista dozwolonych hostów (oddzielone przecinkami)

Aplikacje Django - `INSTALLED_APPS` – zawiera aplikację *accounts*, *two_factor*, *social_django* oraz wtyczki OTP

Middleware - `MIDDLEWARE` – middleware Django + `OTPMiddleware` do obsługi 2FA

Szablony - `TEMPLATES['DIRS']` – katalog z szablonami (np. *templates/*) - `context_processors` – standardowy zestaw + request context

Baza danych - `DATABASES` – SQLite (można łatwo podmienić na PostgreSQL)

Walidacja haseł - Lista validatorów w `AUTH_PASSWORD_VALIDATORS` zgodnie z best practices Django

Międzynarodowość - `LANGUAGE_CODE = 'pl'`, `TIME_ZONE = 'UTC'`, `USE_TZ = True`

Ścieżki plików statycznych - `STATIC_URL`, `MEDIA_URL`

Uwierzytelnianie i logowanie - `AUTHENTICATION_BACKENDS` – obsługa Google, Microsoft, Facebook i Django - `LOGIN_URL`, `LOGOUT_REDIRECT_URL` – ścieżki po zalogowaniu/wylogowaniu

Social Auth – konfiguracja - `SOCIAL_AUTH_GOOGLE_OAUTH2_KEY` / `SECRET` - `SOCIAL_AUTH_FACEBOOK_KEY` / `SECRET` - `SOCIAL_AUTH_MICROSOFT_GRAPH_KEY` / `SECRET` - `SOCIAL_AUTH_PIPELINE` – niestandardowe kroki z aplikacji *accounts.pipeline*

E-mail - `EMAIL_BACKEND` – SMTP (Gmail) - `EMAIL_HOST_USER` – domyślny nadawca - `EMAIL_HOST_PASSWORD` – hasło z *.env* - `DEFAULT_FROM_EMAIL` – używane w *send_mail*

Inne - `BASE_URL` – używane w wiadomościach e-mail (do generowania linków aktywacyjnych)

Indeks

U

V