Dokumentacja deweloperska zespołu SushiBar

Grzegorz Nowak

Jan Nowak

Ziemowit Pałka

Wstęp

System SushiBar został stworzony dla klienta Yakitori Grill Bar, ul. Głogowska 158, Poznań w celu umożliwienia oraz ułatwienia restauracji pracę podczas przyjmowania i wydawania zamówień a także stworzenia systemu pozwalającego usamodzielnienie się klienta w Internecie.

Główne składowe systemu:

- Relacyjna baza danych Postgresql
- Framework Django oraz Django Rest Framework na backendzie
- Framework React oraz Bootstrap na frontendzie

Spis treści

1.Ustawienia systemu

System oparty jest o framework Django, zatem cała struktura projektu podąża zgodnie za wytycznymi tego frameworka. Wszelkich ustawień dotyczących jego konfiguracji dokonuje się w pliku settings.py.

1.1 Aplikacje

Wszystkie aplikacje oraz moduły zewnętrzne zawierane są w liście „INSTALLED_APPS:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
```

```
    'django.contrib.staticfiles',
    'news.apps.NewsConfig',
    'users.apps.UsersConfig',
    'crispy_forms',
    'contact.apps.ContactConfig',
    'about.apps.AboutConfig',
    'menu.apps.MenuConfig',
    'rest_framework',
    'ckeditor',
    'ckeditor_uploader',
    'django_user_agents',
    #allauth
    'django.contrib.sites',
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    'allauth.socialaccount.providers.google',
    'allauth.socialaccount.providers.facebook',
    'translation_manager',
]
```

1.2 Middleware

Middleware w Django są pluginami przetwarzanymi podczas wykonywania żądań (request) i odpowiedzi (response) do serwera zawierane w liście „MIDDLEWARE”

```
MIDDLEWARE = [
    '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',
    'django.middleware.locale.LocaleMiddleware',
    'django_user_agents.middleware.UserAgentMiddleware',
]
```

1.3 Szablony oraz context processors

Szablony w Django odpowiadają za prezentowanie treści gościom strony otrzymując informacje z bazy danych. Context processors są ta funkcje biorące za argument request zwracając dołączone do response dane w postaci słownika (klucz-wartość). Wykorzystywane by przekazywać globalnie pewne wartości do wszystkich szablonów w całym projekcie, zawierane w liście „TEMPALTES”

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
       #Określenie miejsca położenia globalnego folderu
        'DIRS': [(os.path.join(BASE_DIR, 'templates')),],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.template.context_processors.i18n',

'django.contrib.messages.context_processors.messages',
                'contact.context_processors.bar_informations',
                'contact.context_processors.opening_hours',
                'contact.context_processors.realization_hours',
                'menu.context_processors.user_cart',
            ],
            'debug': DEBUG,
        },
    },
]
```

1.4 Konfiguracja bazy danych

W liście „DATABASES” konfigurowane jest połącznie z bazą danych

```
DATABASES = {
    'default': {
       'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'xxxxxxx,
        'USER': 'xxxxxxx,
        'PASSWORD': 'xxxxxxx',
       'HOST': 'pcnet.home.pl',
       'PORT': '5432',
    }
}
```

1.5 Uwierzytelnianie użytkownika oraz walidatory hasła

W projekcie wykorzystywana jest walidacja oparta o wbudowany middleware Django „AuthenticationMiddleware”, który należy skonfigurować tworząc odpowiedni model użytkownika oraz wskazać sposób autentykacji

```
AUTH_USER_MODEL = 'users.CustomUser'
ACCOUNT_EMAIL_REQUIRED=True
ACCOUNT_USERNAME_REQURIED=True
ACCOUNT_AUTHENTICATION_METHOD = "email"

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityVali
dator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

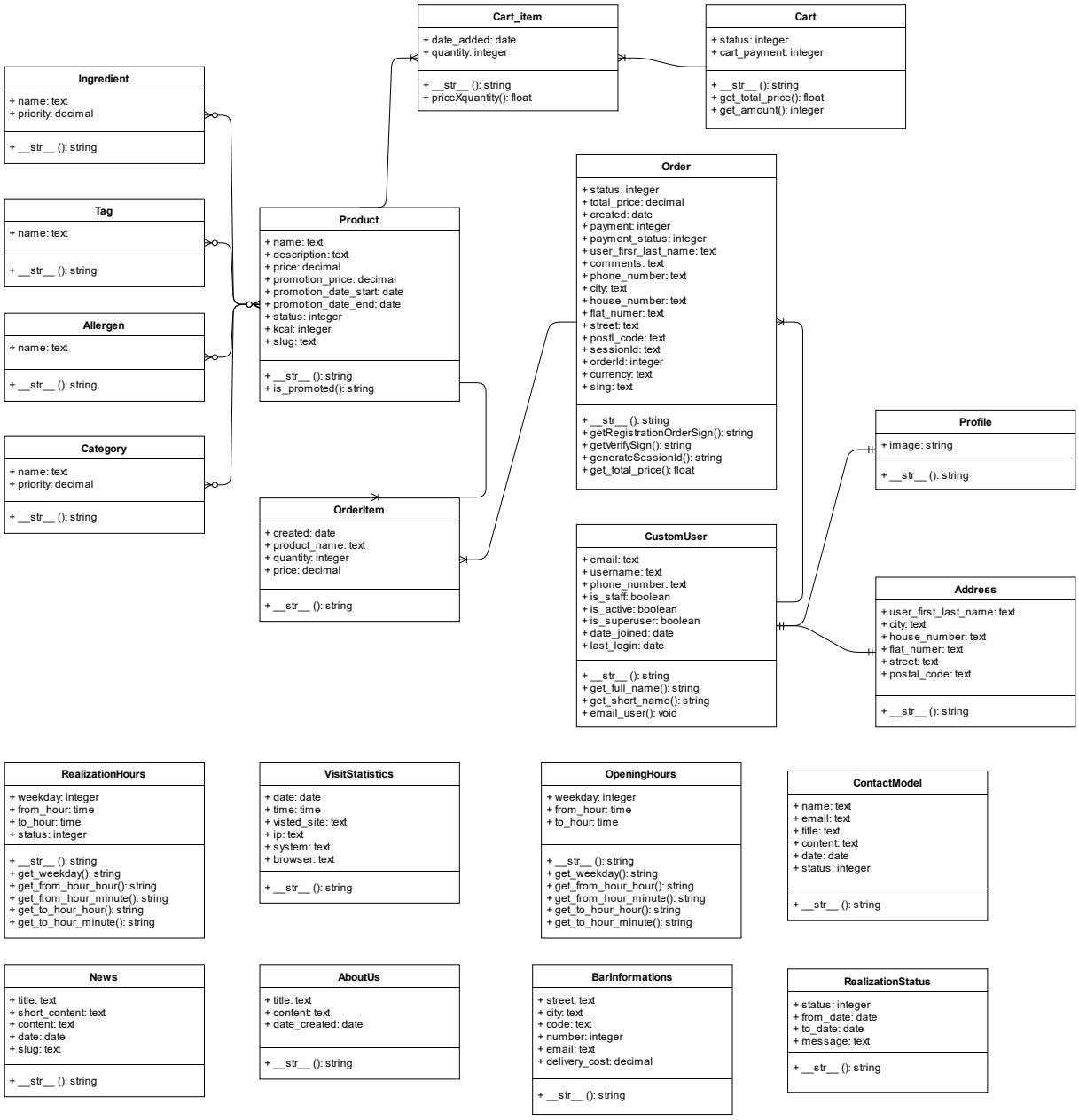
1.6 Konfiguracja poczty

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = xxxxxxxxxx.xx'
EMAIL_PORT = 465
EMAIL_USE_SSL = True
EMAIL_HOST_USER = 'sushibar@soft21.pl'
EMAIL_HOST_PASSWORD = 'xxxxxxx'
DEFAULT_FROM_EMAIL = 'sushibar@soft21.pl'
SERVER_EMAIL = 'sushibar@soft21.pl'
SERVER_EMAIL = 'kontakt@yakitori-grill.pl'
```

2. Modele

Bazę danych tworzą modele danych mapowane dzięki narzędziu ORM (mapowanie obiektowo0relacyjne). Modele są tworzone w odpowiadającym aplikacją (częścią składowych projektu w Django) plikach „models.py”.

2.1 Diagram klas projektu

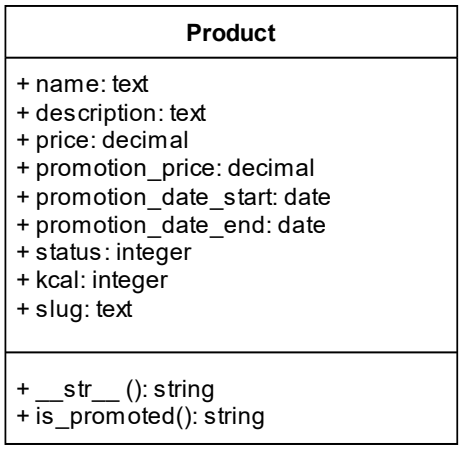


2.2 Menu

Aplikacja „Menu” składa się z dziewięciu modeli. Jest to miejsce kluczowe dla realizacji głównej funkcjonalności klienta – zarządzanie produktami.

2.2.1 Produkt

Model „Product” zawiera wszystkie informacje dotyczące dań lub napojów oferowanych przez klienta.

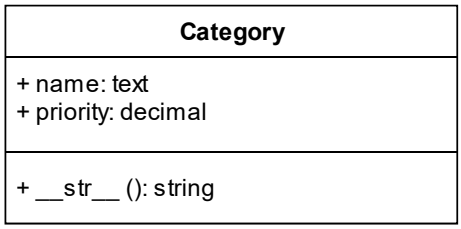


__str__() – Reprezentacja obiektu, w tym miejscu ustala się jak poszczególny obiekt klasy z bazy danych jest wyświetlany dla odbiorcy.

is_promoted() – Funkcja zwracająca wartość boolean w zależności od wyniku sprawdzenia czy dany produkt jest obecnie przeceniony.

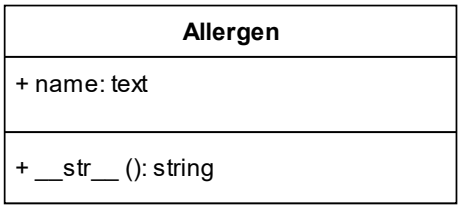
2.2.2 Kategoria

Model „Category” zawiera informacje o kategoriach utworzonych przez klienta oraz kolejność ich wyświetlania na stronie.



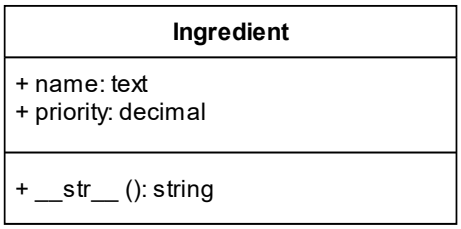
2.2.3 Alergen

Model „Allergen” jest miejscem w którym dodajemy alergeny występujące w potrawach.



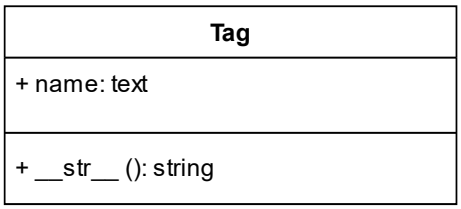
2.2.4 Składnik

Model „Ingredient” jest miejscem w którym dodajemy składniki występujące w potrawach.



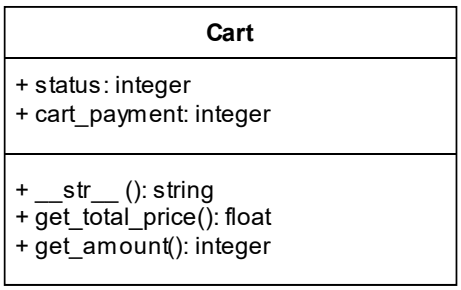
2.2.5 Tag

Model „Tag” jest miejscem w którym dodajemy etykiety, które można przypisać do produktów w celu ułatwienia ich znajdowania.



2.2.6 Koszyk

Model „Cart” jest miejscem w którym tworzymy prywatny koszyk użytkownika.



get_total_price() – funkcja zliczająca łączną kwotę wybranych produktów przez użytkownika

get_amount() – zwraca liczbę produktów znajdujących się w koszyku użytkownika

Koszyk jest tworzony przez wysłanie do serwera sygnału w momencie utworzenia konta przez użytkownika.

Sygnały wykonywane są w pliku signals.py:

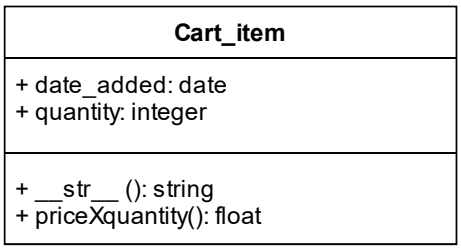
```
@receiver(post_save, sender=CustomUser)
def create_cart(sender, instance, created, **kwargs):
    if created:
        Cart.objects.create(user=instance)



@receiver(post_save, sender=CustomUser)
def save_cart(sender, instance, **kwargs):
    instance.profile.save()
```

2.2.7 Element koszyka

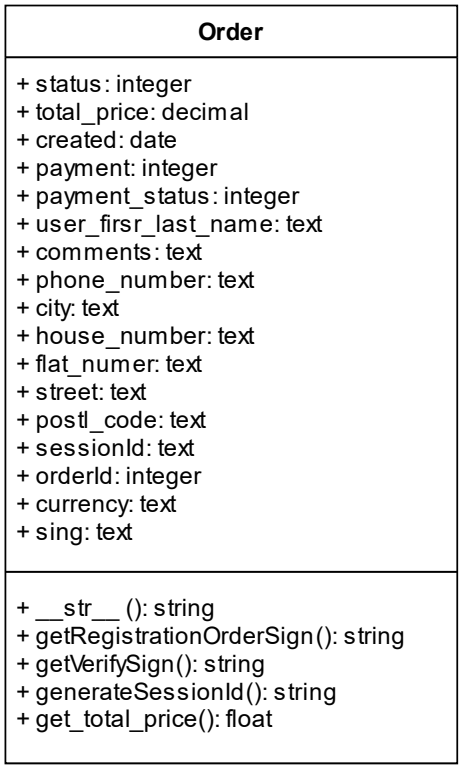
Model „Cart_Item" jest miejscem w którym tworzone są produkty dodawane przez użytkownika do jego koszyka.



priceXquantiy() – funkcja zwracająca ostateczna wartość za produkt w koszyku. W sytuacji gdy produkt jest przeceniony zostaje zwrócona wartość uwzględnia o przecenę.

2.2.8 Zamówienie

Model „Order” jest miejscem w którym tworzone jest zamówienie dokonane przez klienta.



getRegistrationOrderSign() – funkcja tworząca sumę kontrolną wymaganą przy tworzeniu płatności w systemie Przelewy24

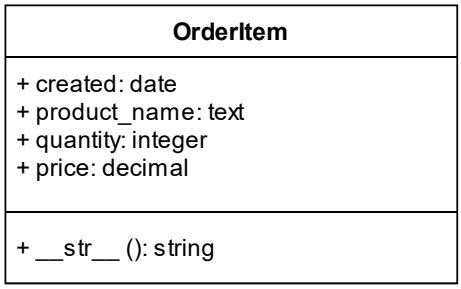
getVeryfySign() – funkcja tworząca sumę kontrolną wymaganą przy weryfikacji płatności w systemie Przelewy24

generateSessionId() – funkcja tworząca identyfikator sesji wymagany przy dokonywaniu płatności w systemie Przelewy24

get_total_price() – funkcja zwracająca pełną wartość kwoty zamówienia

2.2.9 Element zamówienia

Model „OrderItem” jest miejscem w którym tworzone są poszczególne elementy zamówienia.

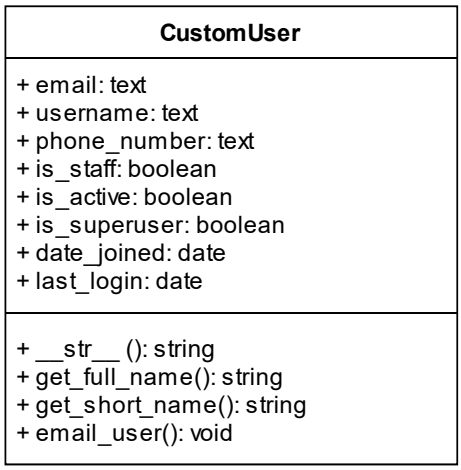


2.3 Użytkownicy

Aplikacja „Użytkownicy” składa się z czterech modeli. W tym miejscu zarządza się wszystkimi informacjami dotyczącymi kont tworzonych przez użytkowników lub właściciela.

2.3.1 Użytkownik

Model „CustomUser” jest miejscem w którym tworzone jest konto użytkownika.



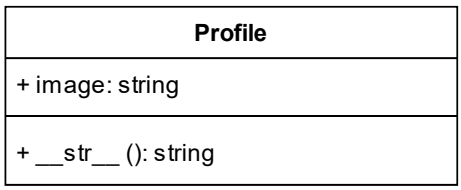
get_full_name() – Funkcja zwracająca email użytkownika wraz z nazwą

get_short_name() – Funkcja zwracająca email użytkownika

email_user() – Funkcja umożliwiająca wysłanie email na adres podany przez użytkownika

2.3.2 Profil

Model „Profile” jest miejscem w którym tworzony jest prywatny profil użytkownika. W aktualnej budowie przechowuje ikonę użytkownika.



Koszyk jest tworzony przez wysłanie do serwera sygnału w momencie utworzenia konta przez użytkownika.

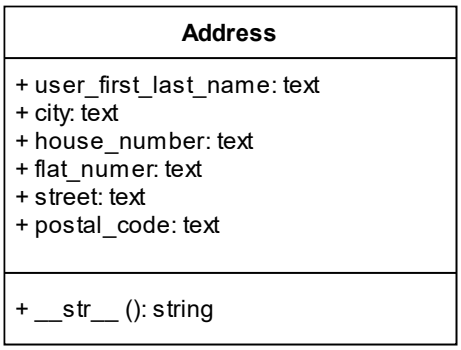
Sygnały wykonywane są w pliku signals.py:

```
@receiver(post_save, sender=CustomUser)
def create_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)


@receiver(post_save, sender=CustomUser)
def save_profile(sender, instance, **kwargs):
    instance.profile.save()
```

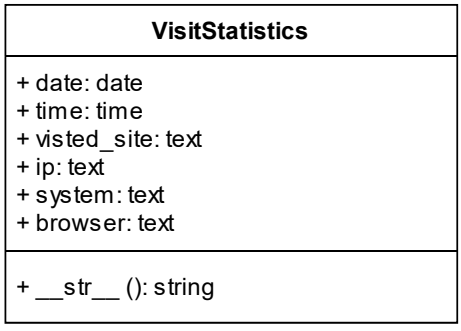
2.3.3 Adres

Model „Address" jest miejscem w którym tworzony jest adres użytkownika. Adres znajduje się na profilu użytkownika.



2.3.4 Statystyki

Model „VisitStatistics" jest miejscem w którym tworzone są statystyki odwiedzin poszczególnej podstorny w systemie przez użytkownika.

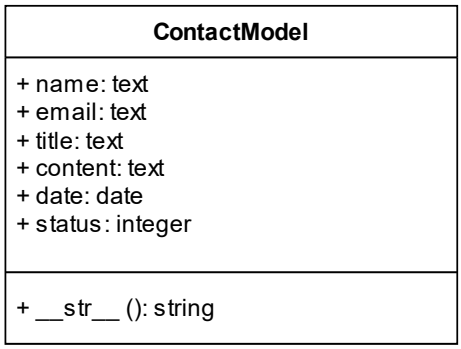


2.4 Kontakt

Aplikacja „Kontakt” składa się z pięciu modeli. Jest to miejsce w którym zarządza się wszelkimi informacjami dotyczycącymi restauracji między innymi godzinami realizacji zamówień.

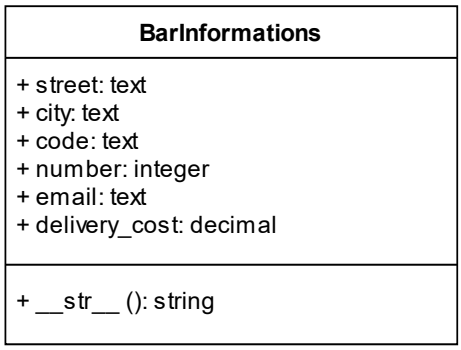
2.4.1 Kontakt

Model „Contact” jest miejscem w którym tworzone są zgłoszenia wysyłane przez użytkowników strony za pomocą formularza zgłoszeniowego w zakładce „Kontakt”.



2.4.2 Informacje o kliencie

Model „BarInformations” jest miejscem w którym tworzone są informacje dotyczące danych kontaktowych dotyczących firmy, adresu ulokowania oraz kosztu dostawy.

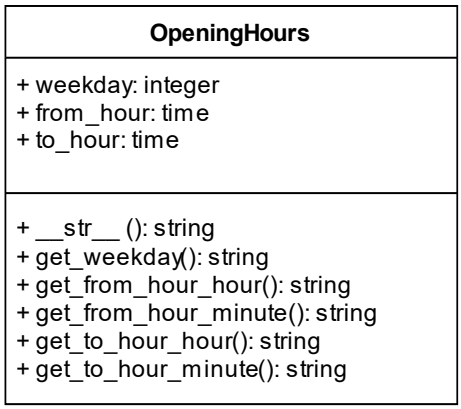


W modelu nadpisywana jest metoda „save” w celu u uniemożliwienia stworzenia więcej niż jednego obiektu w celu uniknięcia duplikatów.

```
def save(self, *args, **kwargs):
    if not self.pk and BarInformations.objects.exists():
        raise ValidationError('Może występować tylko jeden adres!')
    return super(BarInformations, self).save(*args, **kwargs)
```

2.4.3 Godziny otwarcia

Model „OpeningHours” jest miejscem w którym tworzone są informacje dotyczące godzin otwarcia restauracji, nie są to natomiast godziny realizacji zamówień a informacja dla klientów o której można spotkać otwartą restauracje na miejscu.



get_weekday() – Funkcja zwracająca dzień tygodnia

get_from_hour_hour() – Funkcja zwracająca godzinę otwarcia restauracji. Wykorzystywane w funkcjonalności scalowania wyświetlania godzin otwarcia.

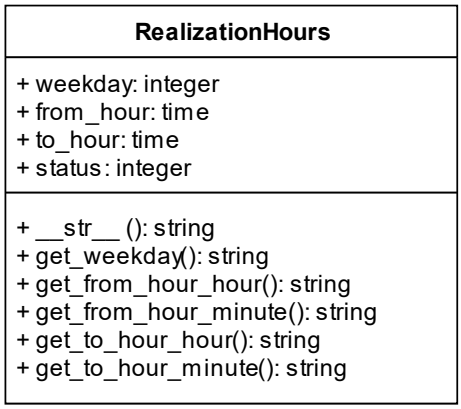
get_from_hour_minute() – Funkcja zwracająca minutę otwarcia restauracji. Wykorzystywane w funkcjonalności scalowania wyświetlania godzin otwarcia.

get_to_hour_hour() – Funkcja zwracająca godzinę zamknięcia restauracji. Wykorzystywane w funkcjonalności scalowania wyświetlania godzin otwarcia.

get_to_hour_minute() - Funkcja zwracająca minutę zamknięcia restauracji. Wykorzystywane w funkcjonalności scalowania wyświetlania godzin otwarcia.

2.4.4 Godziny realizacji zamówień

Model „RealizationHours” jest miejscem w którym tworzone są informacje dotyczące godzin realizacji zamówień przez restauracje.



get_weekday() – Funkcja zwracająca dzień tygodnia

get_from_hour_hour() – Funkcja zwracająca godzinę otwarcia przyjmowania zamówień przez restaurację. Wykorzystywane w funkcjonalności obliczania możliwości składania zamówień przez klientów.

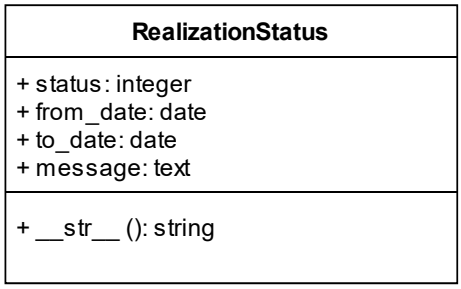
get_from_hour_minute() – Funkcja zwracająca minutę otwarcia przyjmowania zamówień przez restaurację. Wykorzystywane w funkcjonalności obliczania możliwości składania zamówień przez klientów.

get_to_hour_hour() – Funkcja zwracająca godzinę zamknięcia przyjmowania zamówień przez restaurację. Wykorzystywane w funkcjonalności obliczania możliwości składania zamówień przez klientów.

get_to_hour_minute() - Funkcja zwracająca minutę zamknięcia przyjmowania zamówień przez restaurację. Wykorzystywane w funkcjonalności obliczania możliwości składania zamówień przez klientów.

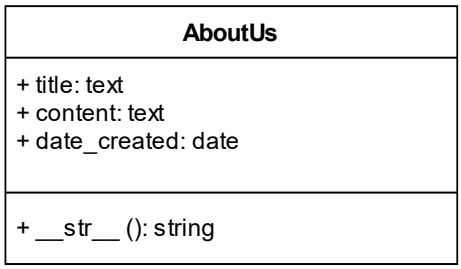
2.4.5 Status możliwości dokonywania zamówień

Model „RealizationStatus” jest miejscem w którym tworzone są informacje dotyczące możliwości dokonywania zamówień przez klientów.



2.5 O nas

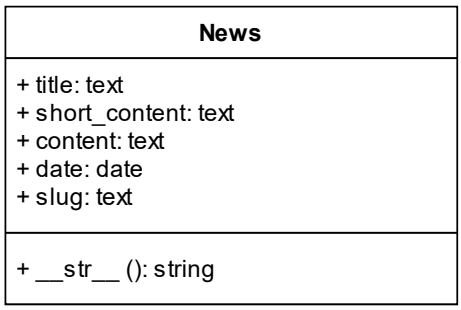
Aplikacja „O nas” jest miejscem, w którym klient ma możliwość przedstawienia się odbiorcy. Oparta jest o narzędzie „CKEditor”, dzięki któremu można wyświetlić informacje zgodnie z oczekiwaniami dzięki podążaniu metodyką WYSIWYG (What You See Is What You Get).



2.6 Aktualności

Aplikacja „Aktualności” umożliwia zarządzanie wiadomościami pojawiającymi się na stornie w celu przekazania odbiorcom informacji o zmianach, nowościach czy promocjach dotyczących restauracji. Umożliwia:

- Dodawanie nowych aktualności
- Edycje aktualności
- Usuwanie aktualności



3. Logowanie za pomocą portali społecznościowych

W projekcie wykorzystujemy moduł „django-allauth” nadpisując niektóre metody w celu usprawnienia działania systemu w pliku „adapter.py”.

```
class MyLoginAccountAdapter(DefaultAccountAdapter):
    def get_login_redirect_url(self, request):
        if request.user.is_authenticated:
            if request.user.username == 'user':
                User = get_user_model()
                users = User.objects.all().count()
                request.user.username = 'Użytkownik' + str(users+1)
                request.user.save()
            url = settings.LOGIN_URL
            return resolve_url(url)
        else:
            return "/"


#sytuacja gdy konto z facebooka oraz gmail mają ten sam mail
@receiver(pre_social_login)
def link_to_local_user(sender, request, sociallogin, **kwargs):
    try:
        email_address = sociallogin.account.extra_data['email']
    except:
        response = redirect('logout')
        return response
    User = get_user_model()
    users = User.objects.filter(email=email_address)
    if users:
        response =
redirect(settings.LOGIN_REDIRECT_URL.format(id=request.user.id))
        # allauth.account.app_settings.EmailVerificationMethod
        perform_login(request, users[0],
email_verification='optional')
        raise ImmediateHttpResponse(response)
```

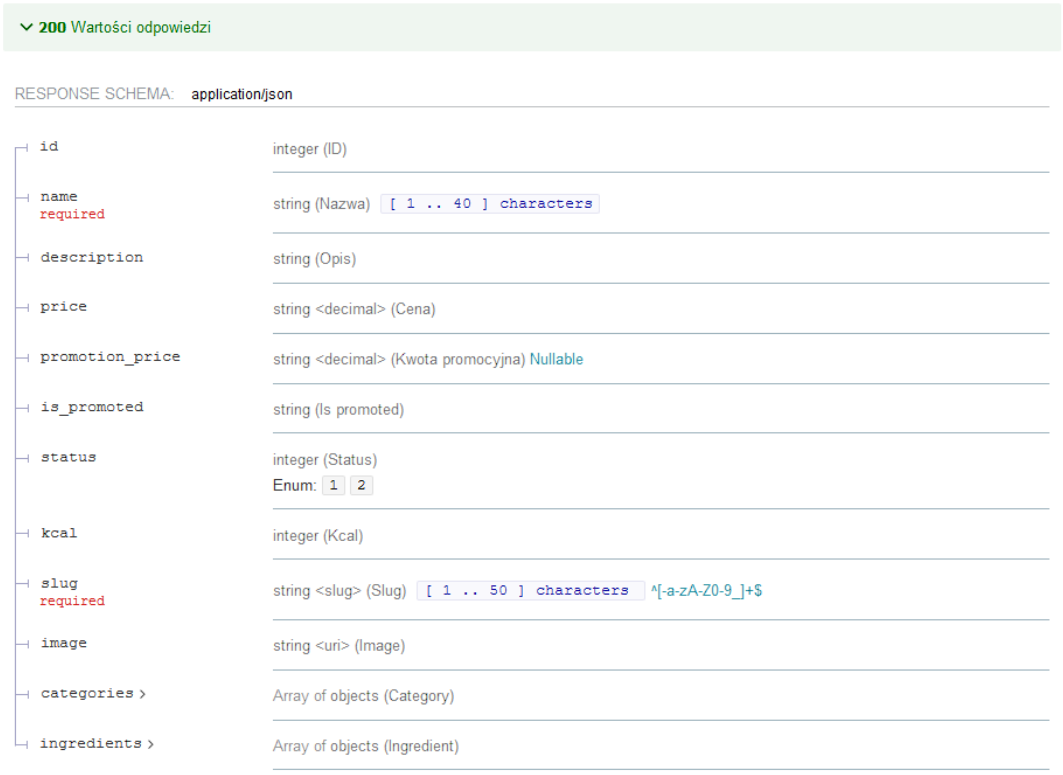
4. Interfejsy API REST

4.1. Interfejsy modelu MENU dostępne pod adresem https://sushibar.soft21.pl

4.1.1. GET /api/menu/ (Pobieranie listy produktów)

Odpowiada za pobranie wszystkich produktów dostępnych w ofercie.

Zwraca odpowiedź:



Rysunek 1 Schemat odpowiedzi GET https://sushibar.soft21.pl/api/menui

Content type
application/json


```
{
   "id": 0,
   "name": "string",
   "description": "string",
   "price": "string",
   "promotion_price": "string",
   "is_promoted": "string",
   "status": 1,
   "kcal": 0,
   "slug": "string",
   "image": "http://example.com",
 - "categories": [
    - {
         "name": "string"
      }
   ],
 - "ingredients": [
    - {
         "name": "string"
      }
   ]
}
```


Rysunek 2 Przykład odpowiedzi https://sushibar.soft21.pl/api/menu

4.1.2. PUT /api/menu/edit/{slug}/ (Aktualizacja danych o produkcie)

Odpowiada za edytowanie produktu w ofercie. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

Wymaga odpowiednich paramentów żądania:

REQUEST BODY SCHEMA: application/json

name required	string (Nazwa) `[ 1 .. 40 ] characters`
description	string (Opis)
price	string <decimal> (Cena)
promotion_price	string <decimal> (Kwota promocyjna) Nullable
status	integer (Status) Enum: `1` `2`
kcal	integer (Kcal)
slug required	string <slug> (Slug) `[ 1 .. 50 ] characters` ^[-a-zA-Z0-9_]+$

Rysunek 3 Schemat zawartości żądania PUT https://sushibar.soft21.pl/api/menu/edit/{slug}/

Content type

application/json


```
{
    "name": "string",
    "description": "string",
    "price": "string",
    "promotion_price": "string",
    "status": 1,
    "kcal": 0,
    "slug": "string"
}
```


Rysunek 4 Przykład zawartości żądania PUT https://sushibar.soft21.pl/api/menu/edit/{slug}/

Zwraca odpowiedź:

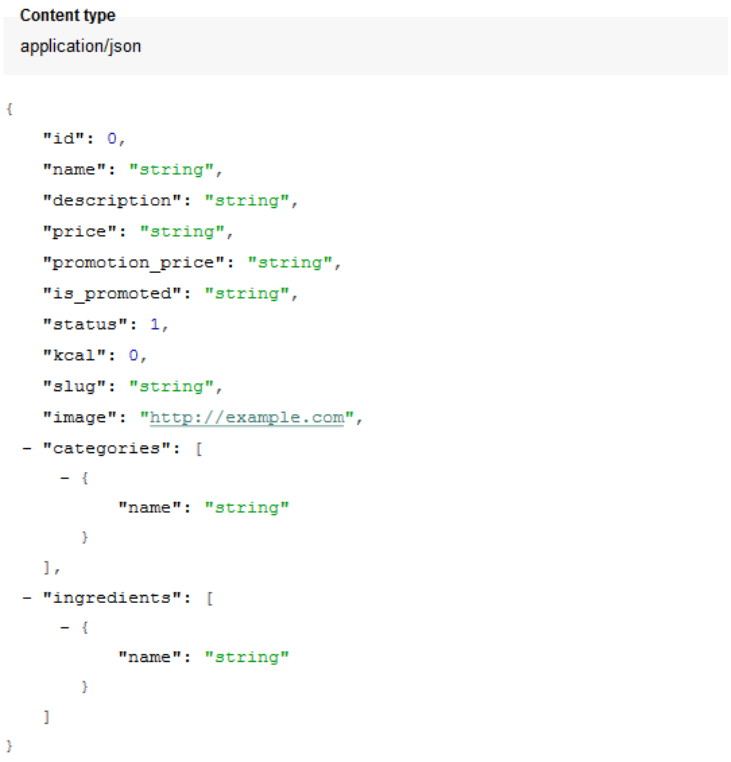
Responses

200

RESPONSE SCHEMA: application/json

id	integer (ID)
name required	string (Nazwa) `[ 1 .. 40 ] characters`
description	string (Opis)
price	string <decimal> (Cena)
promotion_price	string <decimal> (Kwota promocyjna) Nullable
is_promoted	string (Is promoted)
status	integer (Status) Enum: `1` `2`
kcal	integer (Kcal)
slug required	string <slug> (Slug) `[ 1 .. 50 ] characters` ^[-a-zA-Z0-9_]+$
image	string <uri> (Image)
categories >	Array of objects (Category)
ingredients >	Array of objects (Ingredient)

Rysunek 5 Schemat odpowiedzi PUT https://sushibar.soft21.pl/api/menu/edit/{slug}/



Rysunek 6 Przykład odpowiedzi PUT https://sushibar.soft21.pl/api/menu/edit/{slug}/

4.1.3. DELETE /api/menu/edit/{slug}/ (Usuwanie produktu)

Odpowiada za usuwanie produktu z oferty. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

Wymaga podania slugu produktu.

Zwraca status operacji.

4.1.4. GET/api/menu/{slug}/ (Pobieranie danego produktu)

Odpowiada za pobranie danych o produkcie.

Zwraca odpowiedź:

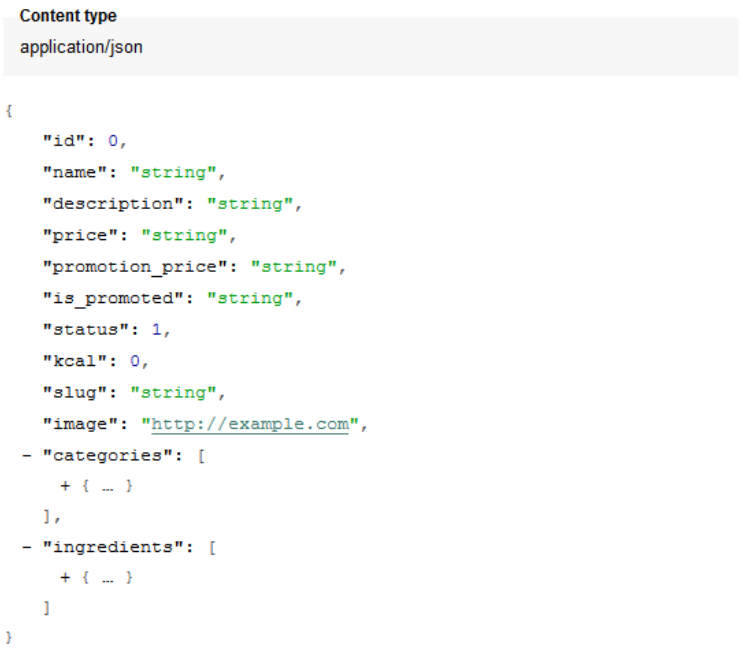
Responses

200

RESPONSE SCHEMA: application/json

id	integer (ID)
name required	string (Nazwa) `[ 1 .. 40 ] characters`
description	string (Opis)
price	string <decimal> (Cena)
promotion_price	string <decimal> (Kwota promocyjna) Nullable
is_promoted	string (Is promoted)
status	integer (Status) Enum: `1` `2`
kcal	integer (Kcal)
slug required	string <slug> (Slug) `[ 1 .. 50 ] characters` ^[-a-zA-Z0-9_]+$
image	string <uri> (Image)
categories >	Array of objects (Category)
ingredients >	Array of objects (Ingredient)

Rysunek 7 Schemat odpowiedzi GET https://sushibar.soft21.pl/api/menu/{slug}/

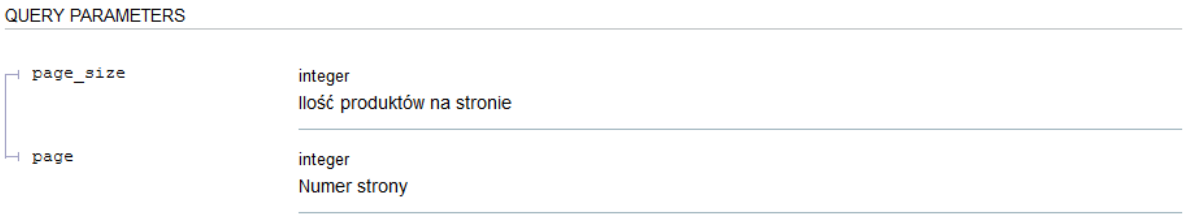


Rysunek 8 Przykładowa odpowiedź GET https://sushibar.soft21.pl/api/menu/{slug}/

4.1.5. GET/api/menu/orders/ (Pobieranie listy zamówień z paginacją)

Odpowiada za pobranie wszystkich zamówień. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

Wymaga określenia parametrów zapytania GET:



Rysunek 9 Wymagane parametry dla zapytania GET https://sushibar.soft21.pl/api/menu/orders/

Zwraca odpowiedź:

Responses

200 Wartości odpowiedzi

RESPONSE SCHEMA: application/json

id	integer (ID)
user	integer (Użytkownik) Nullable
status	integer (Status zamówienia) Enum: 1 2 3 4 5
total_price	number (Kwota zamówienia)
created	string <date-time> (Data zamówienia)
payment	integer (Sposób płatności) Value: 1
payment_status	integer (Status płatności) Enum: 0 1 2
user_first_last_name	string (Imię i Nazwisko) <= 60 characters
comments	string (Uwagi)
phone_number	string (Numer telefonu) <= 30 characters
city	string (Miejscowość) <= 30 characters
house_number	string (Numer domu) <= 5 characters
flat_number	string (Numer mieszkania) <= 5 characters
street	string (Ulica) <= 40 characters
postal_code	string (Kod pocztowy) <= 15 characters
sessionId	string (Id zamówienia P24) <= 100 characters
orderId required	string (Id zamówienia) [1 .. 100] characters
order_items	Array of objects (GetOrderItem)

Array () [

product_name required	string (Nazwa produktu) [1 .. 40] characters
created	string <date-time> (Data zamówienia)
quantity	integer (Ilość)
price	number (Cena)

]

Rysunek 10 Schemat odpowiedzi GET https://sushibar.soft21.pl/api/menu/orders/

Content type
application/json


```
{
   "id": 0,
   "user": 0,
   "status": 1,
   "total_price": 0,
   "created": "2019-08-24T14:15:22Z",
   "payment": 1,
   "payment_status": 0,
   "user_first_last_name": "string",
   "comments": "string",
   "phone_number": "string",
   "city": "string",
   "house_number": "strin",
   "flat_number": "strin",
   "street": "string",
   "postal_code": "string",
   "sessionId": "string",
   "orderId": "string",
 - "order_items": [
     - {
           "product_name": "string",
           "created": "2019-08-24T14:15:22Z",
           "quantity": 0,
           "price": 0
       }
   ]
}
```


Rysunek 11 Przykład odpowiedzi GET https://sushibar.soft21.pl/api/menu/orders/

4.1.6. GET/api/menu/orders/{id}/ (Pobieranie zamówienia o podanym id)

Odpowiada za pobranie danych o zamówieniu według podanego id. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

Wymaga podania parametru id.

Zwraca odpowiedź:

Responses

200 Wartości odpowiedzi

RESPONSE SCHEMA: application/json

id	integer (ID)
user	integer (Użytkownik) Nullable
status	integer (Status zamówienia) Enum: 1 2 3 4 5
total_price	number (Kwota zamówienia)
created	string <date-time> (Data zamówienia)
payment	integer (Sposób płatności) Value: 1
payment_status	integer (Status płatności) Enum: 0 1 2
user_first_last_name	string (Imię i Nazwisko) <= 60 characters
comments	string (Uwagi)
phone_number	string (Numer telefonu) <= 30 characters
city	string (Miejscowość) <= 30 characters
house_number	string (Numer domu) <= 5 characters
flat_number	string (Numer mieszkania) <= 5 characters
street	string (Ulica) <= 40 characters
postal_code	string (Kod pocztowy) <= 15 characters
sessionId	string (Id zamówienia P24) <= 100 characters
orderId required	string (Id zamówienia) [1 .. 100] characters
order_items	Array of objects (GetOrderItem)

Array () [

product_name required	string (Nazwa produktu) [1 .. 40] characters
created	string <date-time> (Data zamówienia)
quantity	integer (Ilość)
price	number (Cena)

]

Rysunek 12 Schemat odpowiedzi GET https://sushibar.soft21.pl/api/menu/orders/{id}/

Content type
application/json


```
{
    "id": 0,
    "user": 0,
    "status": 1,
    "total_price": 0,
    "created": "2019-08-24T14:15:22Z",
    "payment": 1,
    "payment_status": 0,
    "user_first_last_name": "string",
    "comments": "string",
    "phone_number": "string",
    "city": "string",
    "house_number": "strin",
    "flat_number": "strin",
    "street": "string",
    "postal_code": "string",
    "sessionId": "string",
    "orderId": "string",
  - "order_items": [
      + { … }
    ]
}
```


Rysunek 13 Przykład odpowiedzi GET https://sushibar.soft21.pl/api/menu/orders/{id}

4.1.7. PUT/api/menu/orders/{id}/ (Aktualizowanie danych o zmaówieniu)

Odpowiada za zaktualizowanie danych zamówienia o podanym id. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

Wymaga podania parametru id.

Wymaga odpowiednich paramentów żądania:

REQUEST BODY SCHEMA: application/json

user	integer (Użytkownik) Nullable
status	integer (Status zamówienia) Enum: 1 2 3 4 5
total_price	number (Kwota zamówienia)
payment	integer (Sposób płatności) Value: 1
payment_status	integer (Status płatności) Enum: 0 1 2
user_first_last_name	string (Imię i Nazwisko) <= 60 characters
comments	string (Uwagi)
phone_number	string (Numer telefonu) <= 30 characters
city	string (Miejscowość) <= 30 characters
house_number	string (Numer domu) <= 5 characters
flat_number	string (Numer mieszkania) <= 5 characters
street	string (Ulica) <= 40 characters
postal_code	string (Kod pocztowy) <= 15 characters
sessionId	string (Id zamówienia P24) <= 100 characters
orderId required	string (Id zamówienia) [1 .. 100] characters

Rysunek 14 Schemat zawartości żądania PUT https://sushibar.soft21.pl/api/menu/orders/{id}/

Content type

application/json

```
{
    "user": 0,
    "status": 1,
    "total_price": 0,
    "payment": 1,
    "payment_status": 0,
    "user_first_last_name": "string",
    "comments": "string",
    "phone_number": "string",
    "city": "string",
    "house_number": "strin",
    "flat_number": "strin",
    "street": "string",
    "postal_code": "string",
    "sessionId": "string",
    "orderId": "string"
}
```

Rysunek 15 Przykład zawartości żądania PUT https://sushibar.soft21.pl/api/menu/orders/{id}/

Zwraca odpowiedź:

Responses

200

RESPONSE SCHEMA: application/json

id	integer (ID)
user	integer (Użytkownik) Nullable
status	integer (Status zamówienia) Enum: 1 2 3 4 5
total_price	number (Kwota zamówienia)
created	string <date-time> (Data zamówienia)
payment	integer (Sposób płatności) Value: 1
payment_status	integer (Status płatności) Enum: 0 1 2
user_first_last_name	string (Imię i Nazwisko) <= 60 characters
comments	string (Uwagi)
phone_number	string (Numer telefonu) <= 30 characters
city	string (Miejscowość) <= 30 characters
house_number	string (Numer domu) <= 5 characters
flat_number	string (Numer mieszkania) <= 5 characters
street	string (Ulica) <= 40 characters
postal_code	string (Kod pocztowy) <= 15 characters
sessionId	string (Id zamówienia P24) <= 100 characters
orderId required	string (Id zamówienia) [1 .. 100] characters
order_items	Array of objects (GetOrderItem)

Array () [

product_name required	string (Nazwa produktu) [1 .. 40] characters
created	string <date-time> (Data zamówienia)
quantity	integer (Ilość)
price	number (Cena)

]

Rysunek 16 Schemat odpowiedzi PUT https://sushibar.soft21.pl/api/menu/orders/{id}/

Content type

application/json


```
{
    "id": 0,
    "user": 0,
    "status": 1,
    "total_price": 0,
    "created": "2019-08-24T14:15:22Z",
    "payment": 1,
    "payment_status": 0,
    "user_first_last_name": "string",
    "comments": "string",
    "phone_number": "string",
    "city": "string",
    "house_number": "strin",
    "flat_number": "strin",
    "street": "string",
    "postal_code": "string",
    "sessionId": "string",
    "orderId": "string",
  - "order_items": [
      - {
            "product_name": "string",
            "created": "2019-08-24T14:15:22Z",
            "quantity": 0,
            "price": 0
        }
    ]
}
```


Rysunek 17 Przykład odpowiedzi PUT https://sushibar.soft21.pl/api/menu/orders/{id}/

4.1.8. DELETE/api/menu/orders/{id}/ (Usuwanie zamówienia o danym id)

Odpowiada za usunięcia zamówienia o podanym id. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

Wymaga podania parametru id.

Zwraca status operacji.

4.2. Interfejsu modelu NEWS (Aktualności)

4.2.1. GET/api/news/ (Pobieranie wszystkich aktualności)

Odpowiada za pobranie wszystkich aktualności.

Zwraca odpowiedź:

Responses

200 Wartości odpowiedzi

RESPONSE SCHEMA: application/json

id	string (Id)
title required	string (Tytuł) `[ 1 .. 30 ] characters`
short_content	string (Opis) `[ 1 .. 60 ] characters`
content	string (Treść) `non-empty`
slug required	string <slug> (Slug) `[ 1 .. 50 ] characters` ^[-a-zA-Z0-9_]+$
image	string <uri> (Zdjęcie)

Rysunek 18 Schemat odpowiedzi GET https://sushibar.soft21.pl/api/news

Content type
application/json

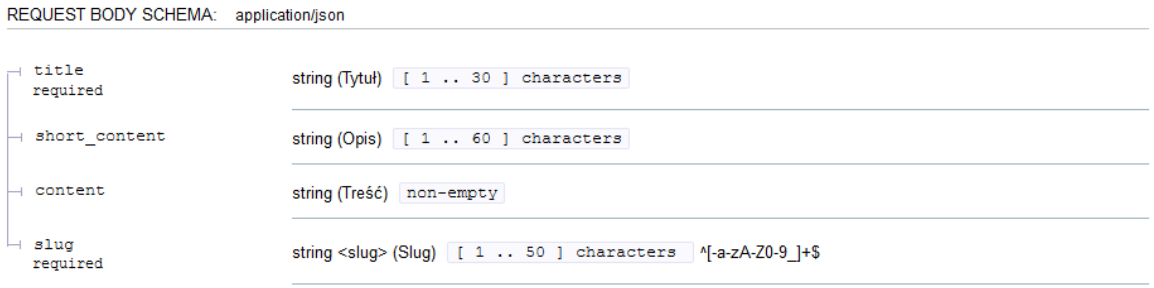

```
{
  "id": "string",
  "title": "string",
  "short_content": "string",
  "content": "string",
  "slug": "string",
  "image": "http://example.com"
}
```


Rysunek 19 Przykład odpowiedzi GET https://sushibar.soft21.pl/api/news

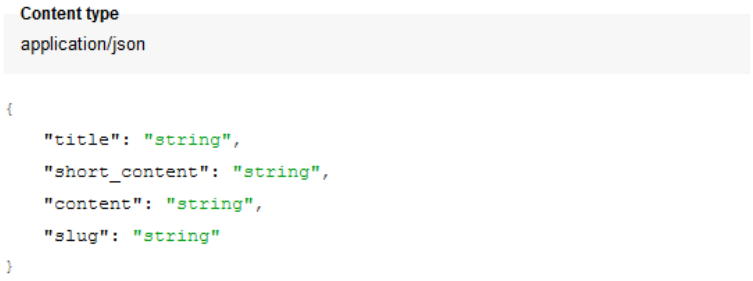
4.2.2. POST/api/news/create/ (Tworzenie aktualności)

Odpowiada za stworzenie nowej aktualności. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

Wymaga odpowiednich paramentów żądania:

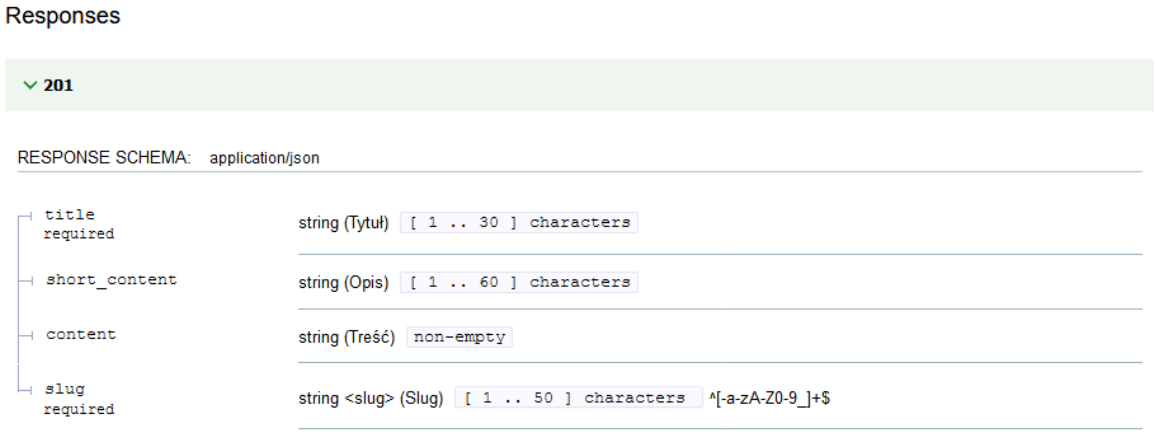


Rysunek 20 Schemat zawartości żądania POST https://sushibar.soft21.pl/api/news/create/



Rysunek 21 Przykład zawartości żądania POST https://sushibar.soft21.pl/api/news/create/

Zwraca odpowiedź:



Rysunek 22 Schemat odpowiedzi POST https://sushibar.soft21.pl/api/news/create/

Content type

application/json


```
{
    "title": "string",
    "short_content": "string",
    "content": "string",
    "slug": "string"
}
```


Rysunek 23 Przykład odpowiedzi POST https://sushibar.soft21.pl/api/news/create/

4.2.3. GET/api/news/{slug}/ (Pobranie aktualności o podanym identyfikatorze sług)

Odpowiada za pobranie aktualności o podanym identyfikatorze sług.

Wymaga podania identyfikatora sług.

Zwraca odpowiedź:

Responses

200 Wartości odpowiedzi

RESPONSE SCHEMA: application/json

id	string (Id)
title required	string (Tytuł) `[ 1 .. 30 ] characters`
short_content	string (Opis) `[ 1 .. 60 ] characters`
content	string (Treść) `non-empty`
slug required	string <slug> (Slug) `[ 1 .. 50 ] characters` ^[-a-zA-Z0-9_]+$
image	string <uri> (Zdjęcie)

Rysunek 24 Schemat odpowiedzi GET https://sushibar.soft21.pl/api/news/{slug}/

Content type

application/json


```
{
    "id": "string",
    "title": "string",
    "short_content": "string",
    "content": "string",
    "slug": "string",
    "image": "http://example.com"
}
```

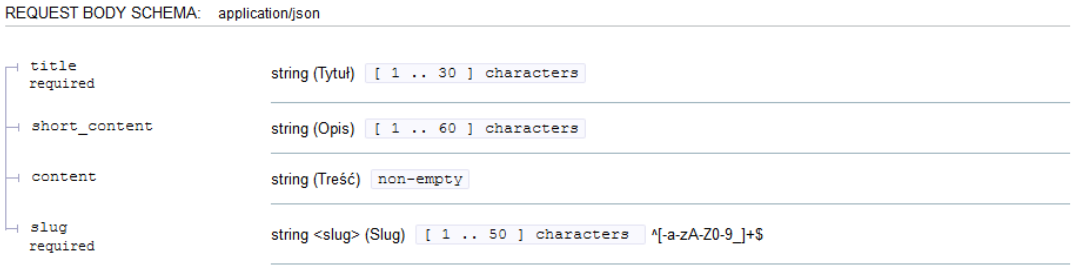

Rysunek 25 Przykład odpowiedzi GET https://sushibar.soft21.pl/api/news/{slug}/

4.2.4. PUT/api/news/{slug}/ (Aktualizacja aktualności o odpowiednim identyfikatorze)

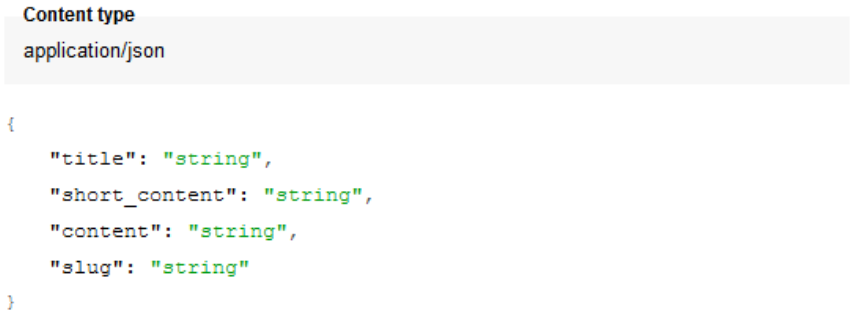
Odpowiada za aktualizowanie aktualności. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

Wymaga podania identyfikatora sług.

Wymaga odpowiednich paramentów żądania:

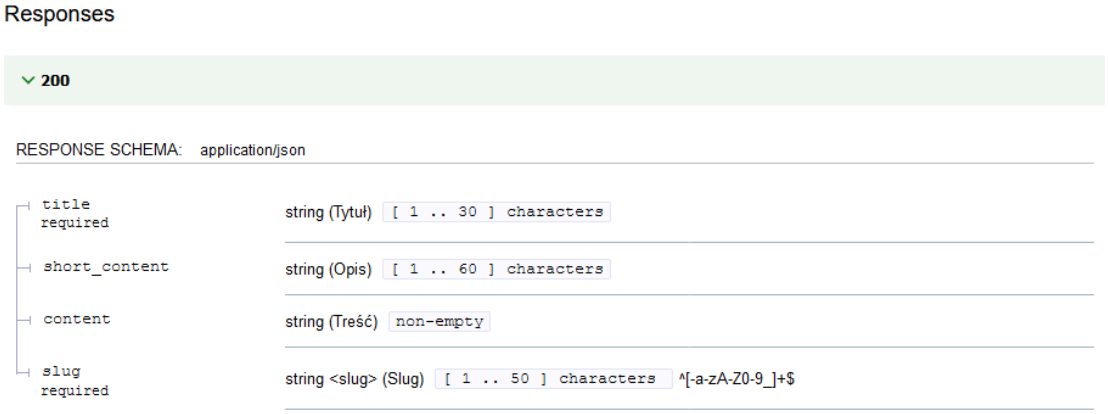


Rysunek 26 Schemat zawartości żądania PUT https://sushibar.soft21.pl/api/news/{slug}/

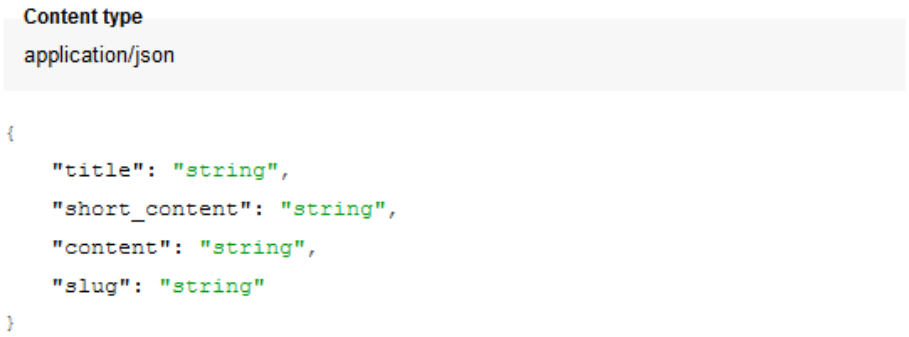


Rysunek 27 Przykład zawartości żądania PUT https://sushibar.soft21.pl/api/news/{slug}/

Zwraca odpowiedź



Rysunek 28 Schemat odpowiedzi PUT https://sushibar.soft21.pl/api/news/{slug}/



Rysunek 29 Przykład odpowiedzi PUT https://sushibar.soft21.pl/api/news/{slug}/

4.2.5. DELETE/api/news/{slug}/ (Usuwa aktualność o podanym identyfikatorze slug)

Odpowiada za usunięcie aktualność o podanym identyfikatorze sług. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

Wymaga podania identyfikatora slug.

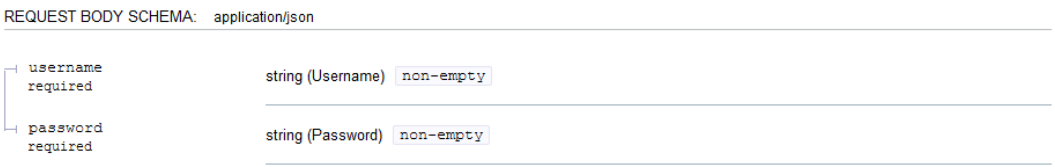
Zwraca status operacji.

4.3. Interfejsy modelu Users (Użytkownicy)

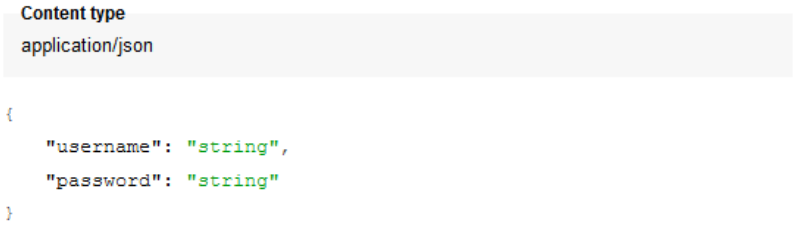
4.3.1. POST/api/token/obtain/ (Generuje token JWT)

Odpowiada za wygenerowanie tokenu JWT.

Wymaga odpowiednich paramentów żądania:



Rysunek 30 Schemat zawartości żądania POST https://sushibar.soft21.pl/api/token/obtain/



Rysunek 31 Przykład zawartości żądania POST https://sushibar.soft21.pl/api/token/obtain/

Zwraca odpowiedź:

Responses

201

RESPONSE SCHEMA: application/json

refresh required	string (Refresh token)	non-empty
access required	string (Access token)	non-empty
username required	string (Username)	non-empty
userId required	string (User Id)	non-empty
admin required	bool (Is super user)	non-empty

Rysunek 32 Schemat odpowiedzi POST https://sushibar.soft21.pl/api/token/obtain/

Content type

application/json


```
{
    "refresh": "string",
    "access": "string",
    "username": "string",
    "userid": "string",
    "admin": bool
}
```


Rysunek 33 Przykład odpowiedzi POST https://sushibar.soft21.pl/api/token/obtain/

4.3.2. POST/api/token/refresh/ (Odświeżenie tokenu JWT)

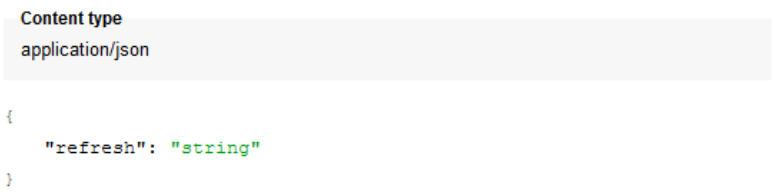
Odpowiada za odświeżenie tokenu.

Wymaga odpowiednich paramentów żądania (wartość tokenu refresh przekazany w odpowiedzi zapytania /api/token/obtain/ lub /api/token/refresh/):

REQUEST BODY SCHEMA: application/json

refresh required	string (Refresh)	non-empty

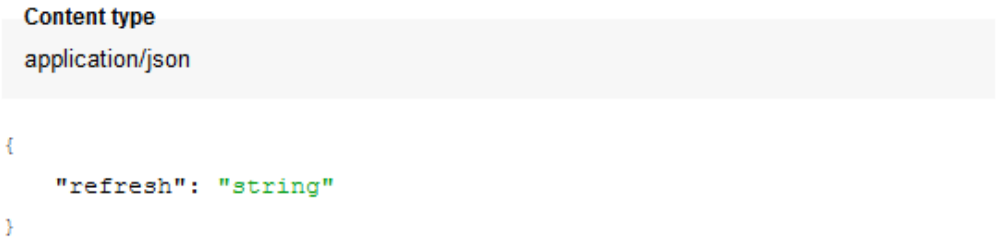
Rysunek 34 Wymagane parametry dla zapytania POST https://sushibar.soft21.pl/api/token/refresh/



Rysunek 35 Przykład wymaganych parametrów dla zapytania POST https://sushibar.soft21.pl/api/token/refresh/

Zwraca odpowiedź:

Rysunek 36 Schemat odpowiedzi POST https://sushibar.soft21.pl/api/token/refresh/

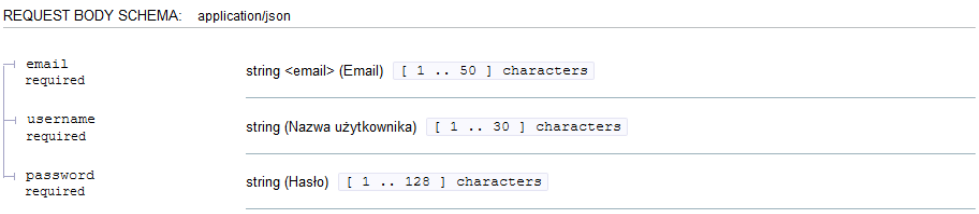


Rysunek 37 Przykład odpowiedzi POST https://sushibar.soft21.pl/api/token/refresh/

4.3.3. POST/api/users/create/ (Utworzenie nowego użytkownika)

Odpowiada za utworzenie użytkownika.

Wymaga odpowiednich paramentów żądania:



Rysunek 38 Schemat zawartości żądania POST https://sushibar.soft21.pl/api/users/create/

Content type

application/json


```
{
    "email": "user@example.com",
    "username": "string",
    "password": "string"
}
```


Rysunek 39 Przykład zawartości żądania POST https://sushibar.soft21.pl/api/users/create/

Zwraca odpowiedź:

Responses

201

RESPONSE SCHEMA: application/json

id	integer (ID)
email required	string <email> (Email) `[ 1 .. 50 ] characters`
username required	string (Nazwa użytkownika) `[ 1 .. 30 ] characters`
password required	string (Hasło) `[ 1 .. 128 ] characters`

Rysunek 40 Schemat odpowiedzi POST https://sushibar.soft21.pl/api/users/create/

Content type

application/json


```
{
    "id": 0,
    "email": "user@example.com",
    "username": "string",
    "password": "string"
}
```

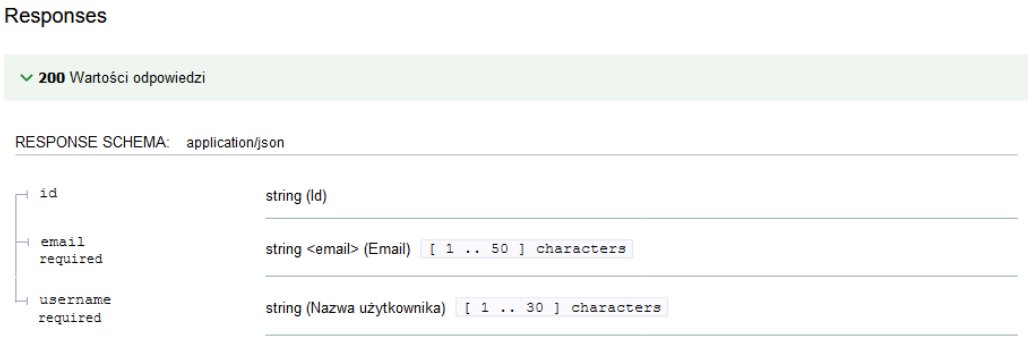

Rysunek 41 Przykład odpowiedzi POST https://sushibar.soft21.pl/api/users/create/

4.3.4. GET/api/users/{id}/ (Pobieranie danych użytkownika)

Odpowiada za pobranie danych użytkownika o podanym identyfikatorze. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

Wymaga podania identyfikatora id.

Zwraca odpowiedź:



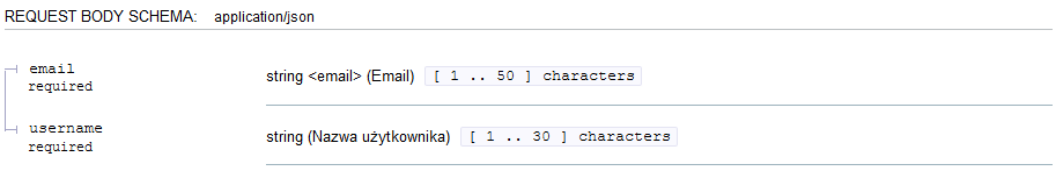
Rysunek 42 Schemat odpowiedzi GET https://sushibar.soft21.pl/api/users/{id}/

Rysunek 43 Przykład odpowiedzi GET https://sushibar.soft21.pl/api/users/{id}/

4.3.5. PUT/api/users/{id}/ (Aktualizowanie danych użytkownika)

Odpowiada za aktualizację danych użytkownika. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

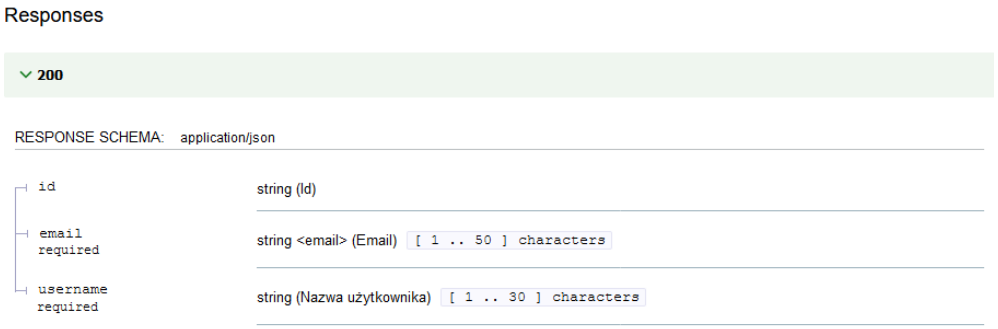
Wymaga podania identyfikatora id.



Rysunek 44 Schemat zawartości żądania PUT https://sushibar.soft21.pl/api/users/{id}/

Rysunek 45 Przykład zawartości żądania PUT https://sushibar.soft21.pl/api/users/{id}/

Zwraca odpowiedź:



Rysunek 46 Schemat odpowiedzi PUT https://sushibar.soft21.pl/api/users/{id}/

Rysunek 47 Przykład odpowiedzi PUT https://sushibar.soft21.pl/api/users/{id}/

4.3.6. DELETE/api/users/{id}/ (Usuwanie użytkownika)

Odpowiada za usuwanie użytkownika o podanym identyfikatorze. Wymaga uwierzytelnienia za pomocą tokena JWT lub Auth Basic oraz odpowiednich uprawnień.

Wymaga podania identyfikatora id.

Zwraca status operacji.

5. Spis ilustracji