

# CookBook

## project documentation

### **Authors:**

*Adrian Ryt*

*Jakub Cichocki*

*Mateusz Kowalski*

## Spis Treści

<b>Database</b>	<b>1</b>
1. Review	1
2. Dish	1
3. Posts	2
<b>Endpoints functionality</b>	<b>3</b>
1. review/	3
2. dish/	4
3. dish_of_type/	4
4. ingredient/	4
5. post/	4
6. types/	5
<b>Endpoints implementation</b>	<b>6</b>
1. review	6
2. post	7
3. availableIngredient	8
4. dishOfType	9
5. dish	9
6. dishWithIngredients	11
<b>Models</b>	<b>11</b>
Review	11
Dish	12
Post	12
Ingredient and IngredientForm (abstract, embedded)	12
TextElement and TextForm (abstract, embedded)	13
<b>Frontend</b>	<b>14</b>
<b>Important places in the project:</b>	<b>14</b>

# Database

Database used for recipes, posts and other data storage is mongoDB, in this case located on the Atlas service. The database consists of 3 data collections:

## 1. Review

Review contains all user reviews

Example element of the Review collection:

```
{
  "id": 1,
  "text": "Example post text",
  "userName": "Test User",
  "title": "Post Title",
  "img": "https://via.placeholder.com/350x150"
}
```

## 2. Dish

Dish contains every recipe, with its name, reviews, types ingredients and text of the recipe itself

Example element of the Dish collection:

```
{
  "id": 1,
  "name": "Test Dish",
  "author": "Test Author",
  "types": [
```

```

        {"name": "Test Type 1"},
        {"name": "Test Type 2"}
    ],
    "description": "Test dish description for
example purposes",
    "Ingredients": [
        {
            "name": "egg",
            "amount": "2 eggs"
        },
        {
            "name": "milk",
            "amount": "2 l."
        }
    ],
    "img": "https://via.placeholder.com/350x150",
    "reviews":
    [
        {
            "id": 74,
            "text": "Example review text",
            "userName": "userName",
            "rating": 5
        }
    ]
}

```




On the DB side there are also additional tables created by the Django ORM framework and Django, which are necessary for the API to function and deliver data to the frontend.




### 3. Posts


Posts containing every post made by users

Example element of the Posts collection:

```
{
  "id": 1,
  "text": "Test text for Post element",
  "userName": "Test user",
  "title": "Test Title",
  "img":
  "https://via.placeholder.com/350x150"
}
```

Dish	
   id	{ Integer }
name	{ String }
author	{ String }
types	{ Array of Objects }
description	{ String }
Ingredients	{ Array of Objects }
reviews_id	{ Array of Objects }
img	{ String }

Post	
   id	{ Integer }
text	{ String }
userName	{ String }
title	{ String }
img	{ String }

Review	
   id	{ Integer }
text	{ String }
userName	{ String }
rating	{ Integer }



# Endpoints functionality

## 1. review/

For posting and acquiring reviews data from DB

Can use every model field as a query param, for example

```
http://localhost:8000/api/review/?userName=blabla&rating=5
```

When it comes to POST method it's really simple, just put the filled database model in your request's body

## 2. dish/

For posting and acquiring dishes data from DB

Can use every model field as a query param excluding array and object fields, for example:

```
http://localhost:8000/api/dish?name=testDish&author=testAuthor
```

When it comes to the POST method it's really simple, just put the filled database model in your request's body.

## 3. dish\_of\_type/

For posting and acquiring dishes data of specific type from DB

Can access dishes with query param "types". Every single type is separated by ",", for example:

```
http://localhost:8000/api/dish_of_type?types=testType1,testType3
```

#### 4. ingredient/

For acquiring every available ingredient data from DB  
You don't need any query params, example usage is:

```
http://localhost:8000/api/ingredient
```

#### 5. post/

For posting and acquiring user posts  
Can use every model field as a query param, for example

```
http://localhost:8000/api/post?title=tytul&userName=slazak
```

When it comes to POST method it's really simple, just put the filled database model in your request's body.

#### 6. types/

For acquiring all available dish types  
\* You don't need any query params, for example:

```
http://localhost:8000/api/types
```

#### 7. dish\_with\_ingredient/

For acquiring all dishes containing all ingredients in query set

Can access dishes with query param "ingredients". Every single ingredient is separated by ",", for example:

```
http://localhost:8000/api/dish_with_ingredient?ingredients=jajko,mleko
```

Every API request is handled by our Django based backend, and returns a JSON response which is structured like this:

```
{  
  data: [],  
  error: 0  
}
```



# Endpoints implementation

All API endpoints are defined in the urls.py file:

```
urlpatterns = [
    path('review/', views.review, name='review'),
    path('post/', views.post, name='post'),
    path('ingredient/', views.availableIngredient, name='ingredient'),
    path('dish/', views.dish, name='dish'),
    path('dish_of_type/', views.dishOfType, name='dish_of_type'),
    path('types/', views.availableTypes, name='types'),
    path('dish_with_ingredient/', views.dishWithIngredients,
name='dish_with_ingredient')
]
```

POST and GET methods on endpoints are handled by functions available in the file views.py.

## 1. review

Handling /review endpoint.

```
def review(request, *args, **kwargs):
    if request.method == 'GET':
        try:
            queryParams = dict(request.GET.items())
            result = list(Review.objects.filter(**queryParams).values())
            return JsonResponse({
                "data": result,
                "error": 0
            })
        except Exception as e:
            print(e)
            return JsonResponse({
                "error": str(e)
            }, status=400)
    if request.method == 'POST':
        try:
            body_unicode = request.body.decode('utf-8')
```

```

body = json.loads(body_unicode)

dishID = ''
if 'dishID' in body['data']:
    dishID = body['data'].pop('dishID')

toSend = Review(**body['data'])
toSend.save()

if 'dishID' != '':
    dish = Dish.objects.get(id = dishID)
    dish.reviews.add(toSend)

return JsonResponse({
    "data": body['data'],
    "error": 0
})
except Exception as e:
    print(e)
    return JsonResponse({
        "error": str(e)
    }, status=400)

```

## 2. post

Handling /post endpoint.

```

def post(request, *arg, **kwargs):
    if request.method == 'GET':
        try:
            queryParams = dict(request.GET.items())
            result = list(Post.objects.filter(**queryParams).values())
            return JsonResponse({
                "data": result,
                "error": 0
            })
        except Exception as e:
            print(e)
            return JsonResponse({
                "error": str(e)
            })

```

```

    }, status=400)
if request.method == 'POST':
    try:
        body_unicode = request.body.decode('utf-8')
        body = json.loads(body_unicode)
        toSend = Post(**body['data'])
        toSend.save()
        return JsonResponse({
            "data": body['data'],
            "error": 0
        })
    except Exception as e:
        print(e)
        return JsonResponse({
            "error": str(e)
        }, status=400)

```

### 3. availableIngredient

Handling /ingredient endpoint.

```

def availableIngredient(request, *arg, **kwargs):
    if request.method == 'GET':
        try:
            data = list(Dish.objects.filter().values())
            result = list(map(lambda x: list(map(lambda y: y['name'],
x['Ingredients'])), data))
            result = list(itertools.chain(*result))
            result = list(set(result))
            print(result)
            return JsonResponse({
                "data": result
            })
        except Exception as e:
            print(e)
            return JsonResponse({
                "error": str(e)
            }, status=400)

```

## 4. dishOfType

Handling /dish\_of\_type endpoint.

```
def dishOfType(request, *arg, **kwargs):
    if request.method == "GET":
        try:
            data = list(Dish.objects.filter().values())
            data = populateReviews(data)
            queryParams = dict(request.GET.items())
            print(queryParams['types'])
            if 'types' not in queryParams or queryParams['types'] == '':
                return JsonResponse({
                    "data": data
                })

            lookingForTypes = queryParams['types'].split(',')

            result = []
            for d in data:
                types = list(map(lambda x: x['name'], d['types']))
                if set(lookingForTypes).issubset(set(types)):
                    result.append(d)
            print(result)
            return JsonResponse({
                "data": result
            })
        except Exception as e:
            print(e)
            return JsonResponse({
                "error": str(e)
            }, status=400)
```

## 5. dish

Handling /dish endpoint.

```
def dish(request, *arg, **kwargs):
    if request.method == 'GET':
        try:
            queryParams = dict(request.GET.items())
            result = list(Dish.objects.filter(**queryParams).values())
            result = populateReviews(result)
```

```

        return JsonResponse({
            "data": result
        })
    except Exception as e:
        print(e)
        return JsonResponse({
            "error": str(e)
        }, status=400)

if request.method == 'POST':
    try:
        body_unicode = request.body.decode('utf-8')
        body = json.loads(body_unicode)

        reviewsLen = len(body['data']['reviews'])
        tempList = []

        for i in range(reviewsLen):
            tempReview = Review(**body['data']['reviews'][i])
            tempReview.save()
            tempList.append(tempReview)

        body['data'].pop('reviews')
        toSend = Dish(**body['data'])
        toSend.save()
        for review in tempList:
            toSend.reviews.add(review)

        return JsonResponse({
            "data": body['data'],
            "error": 0
        })
    except Exception as e:
        print(e)
        return JsonResponse({
            "error": str(e)
        }, status=400)

```

## 6. dishWithIngredients

Handling /dishWithIngredients endpoint.

```
def availableTypes(request, *arg, **kwargs):
    if request.method == 'GET':
        try:
            data = list(Dish.objects.filter().values())
            result = list(map(lambda x: list(map(lambda y: y['name'],
x['types']))), data))
            result = list(itertools.chain(*result))
            result = list(set(result))

            return JsonResponse({
                "data": result
            })
        except Exception as e:
            print(e)
            return JsonResponse({
                "error": str(e)
            }, status=400)
```

# Models

To be able to map data acquired from the mongoDB database to objects we use Django models.

- Review

```
class Review(models.Model):
    text = models.CharField(max_length=1000)
    userName = models.CharField(max_length=50)
    rating = models.IntegerField()
```

- Dish

```
class Dish(models.Model):
    name = models.CharField(max_length=40)
    author = models.CharField(max_length=40)
    types = models.ArrayField(
        model_container=TextElement,
        model_form_class=TextForm
    )
    description = models.CharField(max_length=1000)
    Ingredients = models.ArrayField(
        model_container=Ingredient,
        model_form_class=IngredientForm
    )
    reviews = models.ArrayReferenceField(
        to=Review,
        on_delete=models.CASCADE,
    )
    img = models.CharField(max_length=100)
```

- Post

```
class Post(models.Model):
    text = models.CharField(max_length=1000)
    userName = models.CharField(max_length=50)
    title = models.CharField(max_length=50)
    img = models.CharField(max_length=100)
```

- Ingredient and IngredientForm (abstract, embedded)

```
class Ingredient(models.Model):
    name = models.CharField(max_length=500)
    amount = models.CharField(max_length=100)

    class Meta:
        abstract = True

class IngredientForm(forms.ModelForm):
    class Meta:
        model = Ingredient
        fields = (
            'name', 'amount'
        )
```

- TextElement and TextForm (abstract, embedded)

```
class TextElement(models.Model):
    name = models.CharField(max_length=50)

    class Meta:
```



```
        abstract = True

class TextForm(forms.ModelForm):
    class Meta:
        model = TextElement
        fields = (
            'name',
        )
```

Where error returns an error code signifying the success or failure of the request, and data returns an array of suitable objects to be parsed by frontend.

# Frontend

React components:

AddPost - Form to add posts

AddRecipe - Form to add recipes

DishCard - Displaying dish preview

DishDetails - Displaying dish with all details and reviews

Fridge - To show all dishes you can make with ingredients in your fridge

NavBar - Lets us use React routing between components

Posts - Shows every user post available on the database

Recipes - Shows every recipe of chosen type

ReviewCard - Shows a single review

ReviewForm - Used to add a new review to the chosen recipe

## Important places in the project:

1. *CookBook/RestApi/models.py - models*
2. *CookBook/RestApi/urls.py - urls*
3. *CookBook/RestApi/views.py - views*
4. *frontend/src/components - all components*