Django + Angular 16/15/14/13: CRUD example | Django Rest Framework

Last modified: May 25, 2023 (https://www.bezkoder.com/django-angular-13-crud/) bezkoder (https://www.bezkoder.com/author/bezkoder/) Angular (https://www.bezkoder.com/category/angular/), Django (https://www.bezkoder.com/category/django/), Full Stack (https://www.bezkoder.com/category/full-stack/)

In this tutorial, we will learn how to build a full stack Django + Angular 16/15/14/13 example with a CRUD App. The backend server uses Python 3/Django with Rest Framework for REST APIs. Frontend side is made with Angular 16/15/14/13, HttpClient & Router.

Other versions:
– using Angular 8 (https://www.bezkoder.com/django-angular-crud-rest-framework/)
– using Angular 10 (https://www.bezkoder.com/django-angular-10-crud-rest-framework/)
– using Angular 11 (https://www.bezkoder.com/django-angular-11-crud-rest-framework/)
– using Angular 12 (https://www.bezkoder.com/django-angular-12-crud-rest-framework/)

Contents [hide]

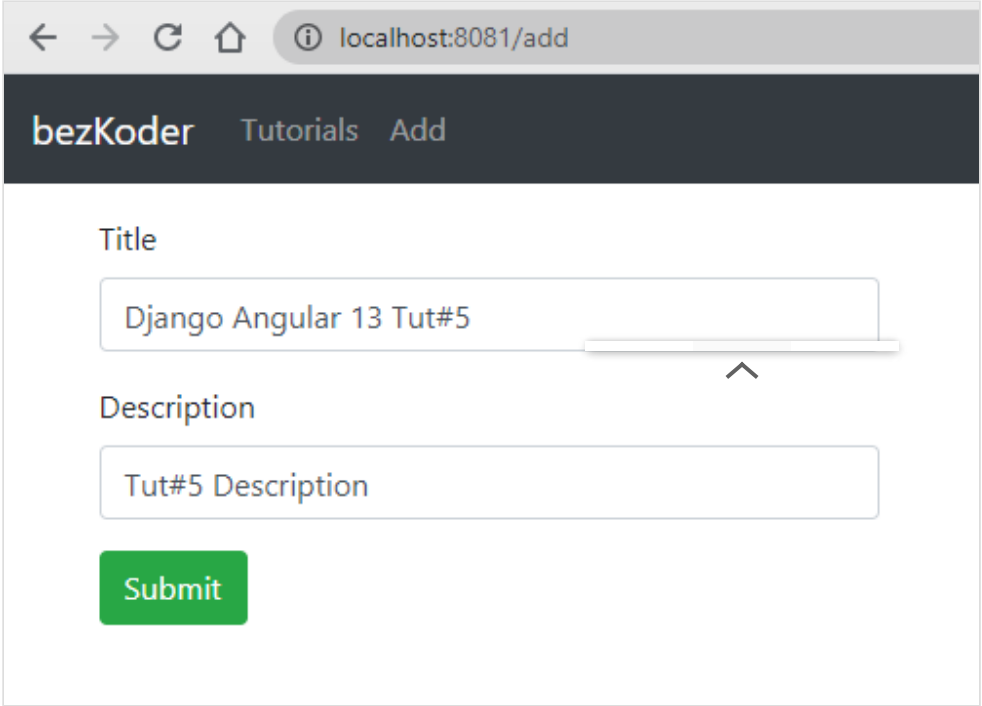
Django + Angular example Overview

We will build a full-stack Django and Angular 16/15/14/13 Tutorial Application in that:

- Each Tutorial has id, title, description, published status.
- We can create, retrieve, update, delete Tutorials.
- We can also find Tutorials by title.

The images below shows screenshots of our System.

– Create a Tutorial:



– Retrieve Tutorials:

– Click on **Edit** button to view a Tutorial details:

On this Page, you can:

- change status to **Published** using **Publish** button
- remove the Tutorial from Database using **Delete** button
- update the Tutorial details on Database with **Update** button

If you want to implement Form Validation, please visit:
Angular Form Validation example (Reactive Forms)
(https://www.bezkoder.com/angular-16-form-validation/)

– Search items by title:

Architecture of Django Angular Tutorial example

This is the application architecture we're gonna build:

– Django Server exports REST Apis using Django Rest Framework & interacts with Database using Django Model.
– Angular 16/15/14/13 Client sends HTTP Requests and retrieve HTTP Responses using HttpClient Module, shows data on the components. We also use Angular Router for navigating to pages.

Django Rest Api Backend

Overview

These are APIs that Django App will export:

Methods	Urls	Actions
POST	/api/tutorials	create new Tutorial
GET	/api/tutorials	retrieve all Tutorials
GET	/api/tutorials/:id	retrieve a Tutorial by `:id`
PUT	/api/tutorials/:id	update a Tutorial by `:id`
DELETE	/api/tutorials/:id	delete a Tutorial by `:id`

Methods	Urls	Actions
DELETE	/api/tutorials	delete all Tutorials
GET	/api/tutorials?title=[keyword]	find all Tutorials which title contains `keyword`

Technology

- Python 3.7
- Django 2.1.15
- Django Rest Framework 3.11.0
- PyMySQL 0.9.3 (MySQL) / psycopg2 2.8.5 (PostgreSQL) / djongo 1.3.1 (MongoDB)
- django-cors-headers 3.2.1

Project Structure

This is our Django project structure:

– *tutorials/apps.py*: declares `TutorialsConfig` class (subclass of `django.apps.AppConfig`) that represents Rest CRUD Apis app and its configuration.
– *bzkRestApis/settings.py*: contains settings for our Django project: Database engine, `INSTALLED_APPS` list with Django REST framework, Tutorials Application, CORS and `MIDDLEWARE`.
– *tutorials/models.py*: defines Tutorial data model class (subclass of `django.db.models.Model`).
– *migrations/0001_initial.py*: is created when we make migrations for the data model, and will be used for generating database table/collection.
– *tutorials/serializers.py*: manages serialization and deserialization with `TutorialSerializer` class (subclass of `rest_framework.serializers.ModelSerializer`).
– *tutorials/views.py*: contains functions to process HTTP requests and produce HTTP responses (using `TutorialSerializer`).

– *tutorials/urls.py*: defines URL patterns along with request functions in the Views.
– *bzkRestApis/urls.py*: also has URL patterns that includes `tutorials.urls` , it is the root URL configurations.

Install Django REST framework

Django REST framework helps us to build RESTful Web Services flexibly.

To install this package, run command:
`pip install djangorestframework`

Setup new Django project

Let's create a new Django project with command:
`django-admin startproject bzkRestApis`

When the process is done, you can see folder tree like this:

Now we open *settings.py* and add Django REST framework to the `INSTALLED_APPS` array here.

```
INSTALLED_APPS = [
    ...
    # Django REST framework
    'rest_framework',
]
```

Setup Database engine

Open *settings.py* and change declaration of `DATABASES` :

```
DATABASES = {
    'default': {
        'ENGINE': ...,
        'NAME': '...',
        'USER': 'root',
        'PASSWORD': '123456',
        'HOST': '127.0.0.1',
        'PORT': ...,
    }
}
```

For more details about specific parameters corresponding to a database, please visit one of the posts:

- Django CRUD with MySQL example | Django Rest Framework (https://bezkoder.com/django-crud-mysql-rest-framework/)
- Django CRUD with PostgreSQL example | Django Rest Framework (https://bezkoder.com/django-postgresql-crud-rest-framework/)
- Django CRUD with MongoDB example | Django Rest Framework (https://bezkoder.com/django-mongodb-crud-rest-framework/)
- Django CRUD with Sqlite example | Django Rest Framework (https://bezkoder.com/django-rest-api/)

Setup new Django app for Rest CRUD Api

Run following commands to create new Django app **tutorials**:

```
cd bzkRestApis
python manage.py startapp tutorials
```

Refresh the project directory tree, you can see it now looks like:

Now open *tutorials/apps.py*, you can see `TutorialsConfig` class (subclass of `django.apps.AppConfig`).

This represents the Django app that we've just created with its configuration:

```
from django.apps import AppConfig


class TutorialsConfig(AppConfig):
    name = 'tutorials'
```

Don't forget to add this app to `INSTALLED_APPS` array in *settings.py*:

```
INSTALLED_APPS = [
    ...
    # Tutorials application
    'tutorials.apps.TutorialsConfig',
]
```

Configure CORS

We need to allow requests to our Django application from other origins.
In this example, we're gonna configure CORS to accept requests from `localhost:8081`.

First, install the *django-cors-headers* library:
`pip install django-cors-headers`

In *settings.py*, add configuration for CORS:

```
INSTALLED_APPS = [
    ...
    # CORS
    'corsheaders',
]
```

You also need to add a middleware class to listen in on responses:

```
MIDDLEWARE = [
    ...
    # CORS
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
]
```

Note: `CorsMiddleware` should be placed as high as possible, especially before any middleware that can generate responses such as `CommonMiddleware`.

Next, set *CORS_ORIGIN_ALLOW_ALL* and add the host to *CORS_ORIGIN_WHITELIST*:

```
CORS_ORIGIN_ALLOW_ALL = False
CORS_ORIGIN_WHITELIST = (
    'http://localhost:8081',
)
```

- *CORS_ORIGIN_ALLOW_ALL*: If `True` , all origins will be accepted (not use the whitelist below). Defaults to `False` .
- *CORS_ORIGIN_WHITELIST*: List of origins that are authorized to make cross-site HTTP requests. Defaults to `[]` .

Define the Django Model

Open **tutorials**/*models.py*, add `Tutorial` class as subclass of `django.db.models.Model` .
There are 3 fields: *title*, *description*, *published*.

```
from django.db import models


class Tutorial(models.Model):
    title = models.CharField(max_length=70, blank=False, default
    description = models.CharField(max_length=200,blank=False, d
    published = models.BooleanField(default=False)
```

Each field is specified as a class attribute, and each attribute maps to a database column.
id field is added automatically.

Migrate Data Model to the database

Run the Python script: `python manage.py makemigrations tutorials` .

The console will show:

```
Migrations for 'tutorials':
  tutorials\migrations\0001_initial.py
    - Create model Tutorial
```

Refresh the workspace, you can see new file *tutorials/migrations/0001_initial.py*.
It includes code to create `Tutorial` data model:

```
# Generated by Django 2.1.15

from django.db import migrations, models


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Tutorial',
            fields=[
                ('id', models.AutoField(auto_created=True, prima
                ('title', models.CharField(default='', max_lengt
                ('description', models.CharField(default='', max
                ('published', models.BooleanField(default=False)
            ],
        ),
    ]
```

The generated code defines `Migration` class (subclass of the `django.db.migrations.Migration`).
It has operations array that contains operation for creating Tutorial model table: `migrations.CreateModel()`.

The call to this will create a new model in the project history and a corresponding table in the database to match it.

To apply the generated migration above, run the following Python script:

```
python manage.py migrate tutorials
```

The console will show:

```
Operations to perform:
  Apply all migrations: tutorials
Running migrations:
  Applying tutorials.0001_initial... OK
```

At this time, you can see that a table/collection for `Tutorial` model was generated automatically with the name: ***tutorials_tutorial*** in the database.

Create Serializer class for Data Model

Let's create `TutorialSerializer` class that will manage serialization and deserialization from JSON.

It inherit from `rest_framework.serializers.ModelSerializer` superclass which automatically populates a set of `fields` and default `validators`. We need to specify the model class here.

tutorials/*serializers.py*

```
from rest_framework import serializers
from tutorials.models import Tutorial


class TutorialSerializer(serializers.ModelSerializer):

    class Meta:
        model = Tutorial
        fields = ('id',
                  'title',
                  'description',
                  'published')
```

In the inner class `Meta`, we declare 2 attributes:

- `model` : the model for Serializer
- `fields` : a tuple of field names to be included in the serialization

Define Routes to Views functions

When a client sends request for an endpoint using HTTP request (GET, POST, PUT, DELETE), we need to determine how the server will response by defining the routes.

These are our routes:

- `/api/tutorials` : GET, POST, DELETE
- `/api/tutorials/:id` : GET, PUT, DELETE
- `/api/tutorials/published` : GET

Create a *urls.py* inside **tutorials** app with `urlpatterns` containing `url`s to be matched with request functions in the *views.py*:

```
from django.conf.urls import url
from tutorials import views

urlpatterns = [
    url(r'^api/tutorials$', views.tutorial_list),
    url(r'^api/tutorials/(?P<pk>[0-9]+)$', views.tutorial_detail
    url(r'^api/tutorials/published$', views.tutorial_list_publis
]
```

Don't forget to include this URL patterns in root URL configurations.

Open **bzkRestApis**/*urls.py* and modify the content with the following code:

```
from django.conf.urls import url, include

urlpatterns = [
    url(r'^', include('tutorials.urls')),
]
```

Write API Views

We're gonna create these API functions for CRUD Operations:

- `tutorial_list()` : GET list of tutorials, POST a new tutorial, DELETE all tutorials
- `tutorial_detail()` : GET / PUT / DELETE tutorial by 'id'
- `tutorial_list_published()` : GET all published tutorials

Open **tutorials**/*views.py* and write following code:

```
from django.shortcuts import render
...

@api_view(['GET', 'POST', 'DELETE'])
def tutorial_list(request):
    # GET list of tutorials, POST a new tutorial, DELETE all tut


@api_view(['GET', 'PUT', 'DELETE'])
def tutorial_detail(request, pk):
    # find tutorial by pk (id)
    try:
        tutorial = Tutorial.objects.get(pk=pk)
    except Tutorial.DoesNotExist:
        return JsonResponse({'message': 'The tutorial does not e

    # GET / PUT / DELETE tutorial


@api_view(['GET'])
def tutorial_list_published(request):
    # GET all published tutorials
```

You can continue with step by step to implement this Django Server (with Github) in one of the posts:

- Django CRUD with MySQL example | Django Rest Framework (https://bezkoder.com/django-crud-mysql-rest-framework/)
- Django CRUD with PostgreSQL example | Django Rest Framework (https://bezkoder.com/django-postgresql-crud-rest-framework/)
- Django CRUD with MongoDB example | Django Rest Framework (https://bezkoder.com/django-mongodb-crud-rest-framework/)
- Django CRUD with Sqlite example | Django Rest Framework (https://bezkoder.com/django-rest-api/)

Run the Django Rest Api Server

Run our Django Project with command: `python manage.py runserver 8080`.
The console shows:

```
Performing system checks...

System check identified no issues (0 silenced).
Django version 2.1.15, using settings 'bzkRestApis.settings'
Starting development server at http://127.0.0.1:8080/
Quit the server with CTRL-BREAK.
```

Angular Frontend

Overview

– The `App` component is a container with `router-outlet` . It has navbar that links to routes paths via `routerLink` .

– `TutorialsList` component gets and displays Tutorials.
– `TutorialDetails` component has form for editing Tutorials details based on `:id` .
– `AddTutorial` component has form for submission new Tutorial.

– These Components call `TutorialService` methods which use Angular `HTTPClient` to make HTTP requests and receive responses.

Technology

- Angular 16/15/14/13
- Angular HttpClient
- Angular Router
- Bootstrap 4

Project Structure

– `tutorial.model.ts` exports the main class model: `Tutorial` .

– There are 3 components: `tutorials-list` , `tutorial-details` , `add-tutorial` .

– `tutorial.service` has methods for sending HTTP requests to the Apis.

– **app-routing.module.ts** defines routes for each component.

– `app` component contains router view and navigation bar.

– `app.module.ts` declares Angular components and import necessary modules.

Setup Angular Project

Let's open cmd and use Angular CLI to create a new Angular Project as following command:

```
ng new angular-16-crud
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
```

We also need to generate some Components and Services:

```
ng g s services/tutorial

ng g c components/add-tutorial
ng g c components/tutorial-details
ng g c components/tutorials-list

ng g class models/tutorial --type=model
```

Set up App Module

Open *app.module.ts* and import `FormsModule` , `HttpClientModule` :

```
...
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [ ... ],
  imports: [
    ...
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Define Routes for Angular AppRoutingModule

There are 3 main routes:

- `/tutorials` for `tutorials-list` component
- `/tutorials/:id` for `tutorial-details` component
- `/add` for `add-tutorial` component

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { TutorialsListComponent } from './components/tutorials-l
import { TutorialDetailsComponent } from './components/tutorial-
import { AddTutorialComponent } from './components/add-tutorial/

const routes: Routes = [
  { path: '', redirectTo: 'tutorials', pathMatch: 'full' },
  { path: 'tutorials', component: TutorialsListComponent },
  { path: 'tutorials/:id', component: TutorialDetailsComponent }
  { path: 'add', component: AddTutorialComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Define Model Class

Our main model class `Tutorial` will be exported in *tutorial.model.ts* with 4 fields:

- `id`
- `title`
- `description`
- `published`

models/*tutorial.model.ts*

```
export class Tutorial {
  id?: any;
  title?: string;
  description?: string;
  published?: boolean;
}
```

Create Data Service

This service will use Angular `HttpClient` to send HTTP requests.

You can see that its functions includes CRUD operations and finder method.

services/*tutorial.service.ts*

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Tutorial } from '../models/tutorial.model';

const baseUrl = 'http://localhost:8080/api/tutorials';

@Injectable({
  providedIn: 'root'
})
export class TutorialService {

  constructor(private http: HttpClient) { }

  getAll(): Observable<Tutorial[]> {
    return this.http.get<Tutorial[]>(baseUrl);
  }

  get(id: any): Observable<Tutorial> {
    return this.http.get(`${baseUrl}/${id}`);
  }

  create(data: any): Observable<any> {
    return this.http.post(baseUrl, data);
  }

  update(id: any, data: any): Observable<any> {
    return this.http.put(`${baseUrl}/${id}`, data);
  }

  delete(id: any): Observable<any> {
    return this.http.delete(`${baseUrl}/${id}`);
  }

  deleteAll(): Observable<any> {
    return this.http.delete(baseUrl);
  }

  findByTitle(title: any): Observable<Tutorial[]> {
    return this.http.get<Tutorial[]>(`${baseUrl}?title=${title}`
  }
}
```

Create Angular Components

As you've known before, there are 3 components corresponding to 3 routes defined in `AppRoutingModule` .

- Add new Item Component

- List of items Component
- Item details Component

You can continue with step by step to implement this Angular App (with Github) in the post:
– Angular 13 CRUD example with Web API (https://www.bezkoder.com/angular-13-crud-example/)
– Angular 14 CRUD example with Web API (https://www.bezkoder.com/angular-14-crud-example/)
– Angular 15 CRUD example with Web API (https://www.bezkoder.com/angular-15-crud-example/)
– Angular 16 CRUD example with Web API (https://www.bezkoder.com/angular-16-crud-example/)

Run the Angular App

You can run this App with command: `ng serve --port 8081`.
If the process is successful, open Browser with Url: `http://localhost:8081/` and check it.

Further Reading

- Django Rest Framework quich start (https://www.django-rest-framework.org/tutorial/quickstart/)
- Django Model (https://docs.djangoproject.com/en/2.1/topics/db/models/)
- Angular HttpClient (https://angular.io/guide/http)
- Angular Template Syntax (https://angular.io/guide/template-syntax)

– Django + Angular + MySQL example (https://bezkoder.com/django-angular-mysql/)
– Django + Angular + PostgreSQL example (https://bezkoder.com/django-angular-postgresql/)
– Django + Angular + MongoDB example (https://bezkoder.com/django-angular-mongodb/)

If you want to implement Form Validation, please visit:
Angular 16 Form Validation example (Reactive Forms) (https://www.bezkoder.com/angular-16-form-validation/)

Conclusion

Now we have an overview of Angular 16/15/14/13 + Django example when building a CRUD App that interacts with database. We also take a look at client-server architecture for Django backend REST API using Django Rest Framework (Python 3), as well as Angular 16/15/14/13 frontend project structure for making HTTP requests and consuming responses.

Next tutorials show you more details about how to implement the system (including Github source code):
– Back-end:

- with MySQL (https://bezkoder.com/django-crud-mysql-rest-framework/)
- with PostgreSQL (https://bezkoder.com/django-postgresql-crud-rest-framework/)
- with MongoDB (https://bezkoder.com/django-mongodb-crud-rest-framework/)
- with Sqlite (https://bezkoder.com/django-rest-api/)

– Front-end:

- Using Angular 8 (https://www.bezkoder.com/angular-crud-app/)
- Using Angular 10 (https://www.bezkoder.com/angular-10-crud-app/)
- Using Angular 11 (https://www.bezkoder.com/angular-11-crud-app/)
- Using Angular 12 (https://www.bezkoder.com/angular-12-crud-app/)
- Using Angular 13 (https://www.bezkoder.com/angular-13-crud-example/)
- Using Angular 14 (https://www.bezkoder.com/angular-14-crud-example/)
- Using Angular 15 (https://www.bezkoder.com/angular-15-crud-example/)
- Using Angular 16 (https://www.bezkoder.com/angular-16-crud-example/)

angular (https://www.bezkoder.com/tag/angular/)

angular 13 (https://www.bezkoder.com/tag/angular-13/)

crud (https://www.bezkoder.com/tag/crud/) django (https://www.bezkoder.com/tag/django/)

django rest framework (https://www.bezkoder.com/tag/django-rest-framework/)

rest api (https://www.bezkoder.com/tag/rest-api/)

One thought to “Django + Angular 16/15/14/13: CRUD example | Django Rest Framework”

Garrett Mitchener
April 13, 2022 at 6:55 pm (https://www.bezkoder.com/django-angular-13-crud/#comment-21023)

Thanks for writing these tutorials! They’ve been so helpful. I’m writing a Django / Angular web system and your posts have helped me put all the pieces together.

Comments are closed to reduce spam. If you have any question, please send me an email.

MEAN Stack example: CRUD with Angular 13, Node, MongoDB (https://www.bezkoder.com/mean-stack-crud-example-angular-13/)

Angular 13 Pagination example (server side) with ngx-pagination (https://www.bezkoder.com/angular-13-pagination-ngx/)

Search...

FOLLOW US

(https://www.youtube.com/channel/UCp0mx9RH0JxaFsmMvKXA86Q)

(https://facebook.com/bezkoder)

(https://github.com/bezkoder)

TOOLS

Json Formatter (https://www.bezkoder.com/json-formatter/)

. Privacy Policy (https://www.bezkoder.com/privacy-policy/)

Contact (https://www.bezkoder.com/contact-us/) About (https://www.bezkoder.com/about/)