## **Run the Django CRUD API Locally**

* Install the latest version of Python from [**https://www.python.org/downloads/**](https://www.python.org/downloads/).
* Visit [**https://github.com/wpcodevo/Django\_Crud\_Project**](https://github.com/wpcodevo/Django_Crud_Project) to download or clone the Django CRUD API project and open it with an IDE.
* In the integrated terminal of your IDE or text editor, run this command to create a virtual environment in the root project directory.
  + Windows OS – python -m venv venv
  + Mac or Linux OS – python3 -m venv venv
* Once the new virtual environment has been created, your IDE or code editor will prompt you to activate it for the workspace folder. Click “Yes” to accept the action.  
    
  Note: You need to close the previously opened terminal in your IDE and open a new terminal to activate the virtual environment in the terminal.  
    
  If your IDE or text editor didn’t prompt you to activate the new virtual environment, run the command below from the terminal of the root directory to manually activate it.
  + Windows OS (Command Prompt ) – venv\Scripts\activate.bat.
  + Windows OS (Git Bash) – venv/Scripts/activate.bat.
  + Mac or Linux OS – source venv/bin/activate
* Run pip install -r requirements.txt to install all the required modules.
* After that, migrate the database schema to the SQLite database withpython manage.py migrate.
* Start the Django development server by running python manage.py runserver
* Test the API endpoints from an API testing tool like [**Postman**](https://www.postman.com/) or [**Thunder Client**](https://marketplace.visualstudio.com/items?itemName=rangav.vscode-thunder-client) VS Code extension. You can also set up the React.js app to interact with the Django CRUD API.

## 

## 

## 

## **Run a React.js App with the Django API**

For a complete guide on how to build the React.js CRUD app see the post [Build a React.js CRUD App using a RESTful API](https://codevoweb.com/build-a-reactjs-crud-app-using-a-restful-api). Nevertheless, follow these steps to quickly spin up the React app.

* Download Node.js from [**https://nodejs.org/**](https://nodejs.org/) and run npm i -g yarn to install the Yarn package manager globally.
* Download or clone the React.js CRUD project from [**https://github.com/wpcodevo/reactjs-crud-note-app**](https://github.com/wpcodevo/reactjs-crud-note-app) and open it with an IDE.
* After that, open the integrated terminal in your IDE or code editor and run yarn or yarn install to install all the required dependencies.
* Start the Vite dev server by running yarn dev from the console of the root directory.
* Open http://localhost:3000/ in a new tab to perform the CRUD operations against the Django API. Note: When you visit the React app on http://127.0.0.1:3000 , you will get a site can’t be reached or CORS error. The Django API is configured to accept requests from only http://localhost:3000/

## **Setup Django**

The first step in creating any project is to create the project’s work directory. So navigate to your Desktop or any location on your machine and create a folder named Django\_Crud\_Project . After that, open the newly-created folder in an IDE or code editor (I will use [**Visual Studio Code**](https://code.visualstudio.com/) in this project)

mkdir Django\_Crud\_Project

cd Django\_Crud\_Project

Now let’s create a virtual environment to isolate dependencies. To do this, open the integrated terminal in your IDE and run this command.

* Windows OS – python -m venv venv

After that, run the command below to activate the virtual environment in the current workspace.

* Windows OS (Command Prompt ) – venv\Scripts\activate.bat.
* Windows OS (Git Bash) – venv/Scripts/activate.bat.

Because we are using both Django and Django REST framework, run this command in the console of the root directory to install them in the virtual environment.

pip install django djangorestframework

* [**django**](https://www.djangoproject.com/) – A Python web framework that follows the model–template–views architectural pattern.
* [**djangorestframework**](https://www.django-rest-framework.org/) – A powerful and flexible toolkit for building Web APIs.

At this point, we are ready to create the Django project. Run this command to create a Django project named notes in the current workspace folder. The dot (.) after the project’s name will tell Django-admin to generate the project in the current directory.

django-admin startproject notes .

After the project has been generated, run this command to activate the initial database migrations of the built-in user model.

python manage.py migrate

Now that we’ve generated the Django project and migrated the database schema, let’s start the dev server to make sure we did everything correctly. So, start the Django development server with this command.

python manage.py runserver

This will start the HTTP server on . When you visit it in a new tab, you will see the Django welcome page.

Django provides a reusability feature that we can leverage to organize our code into reusable apps. That means we will create an app to handle every request made to the “notes” endpoints. So, run this command to create a Django app called note\_api.

django-admin startapp node\_api

## **Create the Django Models**

In this section, you’ll create a database model that the API will use to perform the CRUD operations and a serializer model that the Django REST framework will use to convert the models into serialized JSON objects.

### **Database Model**

In Django, a model is a built-in feature that Django uses to generate the underlying database tables, their columns, relationships between the tables, and various constraints.

So, create a models.py file in the note\_api app and add the following model definitions. Instead of using incremental integers for the primary key of the table, we’ll use UUIDs to prevent attackers from scanning the table with a range of integers to explore leakages.

The models.UUIDField() attribute will tell Django to generate a UUID for the ID field by evoking the uuid4() function.

We added a unique constraint on the title field to ensure that no two records in the table end up with the same title.

The db\_table property provided in the [**Meta Options**](https://docs.djangoproject.com/en/dev/ref/models/options/) will tell Django to change the table name to the specified value. Also, the ordering property will tell Django to order the lists of objects in ascending order of the createdAt timestamp.

### **Model Serializer**

Serializers in Django REST framework convert the object or list of objects returned by the database into data types that frontend frameworks and API testing tools can understand. Also, serializers can be used to deserialize data into complex types after the incoming request data has been validated.

Django REST framework comes with a handful of [**serializers**](https://www.django-rest-framework.org/api-guide/serializers/) but in this tutorial, we’ll only use the [**ModelSerializer**](https://www.geeksforgeeks.org/modelserializer-in-serializers-django-rest-framework/). The ModelSerializer class provides a shortcut to create a serializer whose fields correspond to the Model fields.

In the note\_api app directory, create a serializers.py file and add the serializer below.

Inheriting the ModelSerializer class will:

* Automatically generate a set of fields for the NoteSerializer
* Automatically generate validators for the serializer
* Create default implementations of .create() and .update()

## **Create the CRUD API Views in Django**

In this section, you’ll create five API views to implement the CRUD (Create, Read, Update, and Delete) functionalities. In Django, views are divided into two major types; function-based views (FBVs) and class-based views (CBVs).

* function-based views – Are functions that leverage the @api\_view decorator and return HTTP responses after processing the business logic of the API.
* class-based views – Python objects that allow us to create views from inherited classes.

These views handle the application logic of the API and return information as responses to the client or frontend application. As a matter of best practice, we always keep the application logic in a views.py file.

To simplify the process, we’ll use the [**GenericAPIView**](https://www.django-rest-framework.org/api-guide/generic-views/) class provided by the Django REST framework to create the views. Using the GenericAPIView class is quite similar to the regular View class since the incoming request will be delegated to an appropriate handler method such as .get() or .post().

We’ll group the API views under two classes, Notes and NoteDetail. The Notes API view class will handle requests made to the /api/notes endpoint and the NoteDetail API view class will handle requests made to the api/notes/<str:pk> endpoint.

Before the CRUD implementation, create a views.py file inside the note\_api app directory

### **GET and POST API Views**

Here, you’ll create a Notes class that inherits the GenericAPIView class to handle POST and GET requests. The Notes view will have two handler methods:

* get() – The get() handler will return a paginated list of records to the client. By default, the handler will only return the first 10 records to the client if the limit and page parameters are absent in the request URL.
* post() – This handler will be evoked by the REST framework to add the new record to the database when a POST request hits the server at /api/notes.

### **GET, PATCH, and DELETE API Views**

Here, you’ll create a NoteDetail class that inherits the GenericAPIView class to handle GET, PATCH, and DELETE requests. The NoteDetail view will have three handler methods:

* get() – This handler will be called to find a record by ID and return the found record to the client or frontend app.
* patch() – This handler will be evoked by the REST framework to find a record that matches an ID and update the fields of the found record.
* delete() – This handler will be evoked to find a record by ID and delete the found record from the database.

## **Add the CRUD Routes**

A request in Django first comes to urls.py in the main project and then goes to the urls.py of an app before the matching function in views.py will be called to process the request.

So, to make users interact with the API views defined in the note\_api app, we need to create URLs that map to them.

### **Add the CRUD API URLs**

To add the CRUD API views to the middleware stack, create a urls.py file in the note\_api app directory

**Add the Base URL of the CRUD App to the Project**

Now that we’ve defined the CRUD API routes in the note\_api/urls.py file, let’s add a base URL that references the URLs defined in the note\_api app to the notes project.

## **Setup CORS in Django**

Now that we’ve implemented the CRUD functionalities, let’s configure the Django project to enable CORS on the server. Adding CORS will ensure that the server accepts requests from specified cross-origin domains. Follow the steps below to enable CORS on the Django server.

Add this middleware class to listen in on responses. Note: The CorsMiddleware should be placed before any middleware that can generate responses.

Now let’s configure the CORS middleware’s behavior by specifying the domains for CORS. Also, add CORS\_ALLOW\_CREDENTIALS = True so that cookies can be allowed in cross-site HTTP requests.

After all the above configurations, your notes/settings.py file should look somewhat like this.

## **Create the Migration File and Start the Server**

Let’s create a database migration file for the NoteModel. The migration files will instruct Django on how to create the database tables. Run this command to generate the migration file.

python manage.py makemigrations

Now run the migrate command to push the migrations to the database

python manage.py migrate

After Django has pushed the schema to the database and synced the database schema with our model, run this command to start the HTTP server.

python manage.py runserver

## **Test the Django CRUD API**

It’s now time to test the Django CRUD API. To do this, we’ll use an API testing tool like Postman, Insomnia, or Thunder client VS Code extension. You can import the collection I used in testing the API into Thunder client or Postman by following the steps below.

These steps are for Postman but the process should be similar for the Thunder client extension.

Step 1: Click on the Import button and then click the Choose Files button.

Step 2: In the file explorer, navigate to the Note App.postman\_collection.json file in the Django project and choose it.

Step 3: Click on the Import button in the Import Elements tab to add the collection.

### **Create Note**

To create a new note, make a POST request to the http://localhost:8000/api/notes/ endpoint with the JSON object.

{

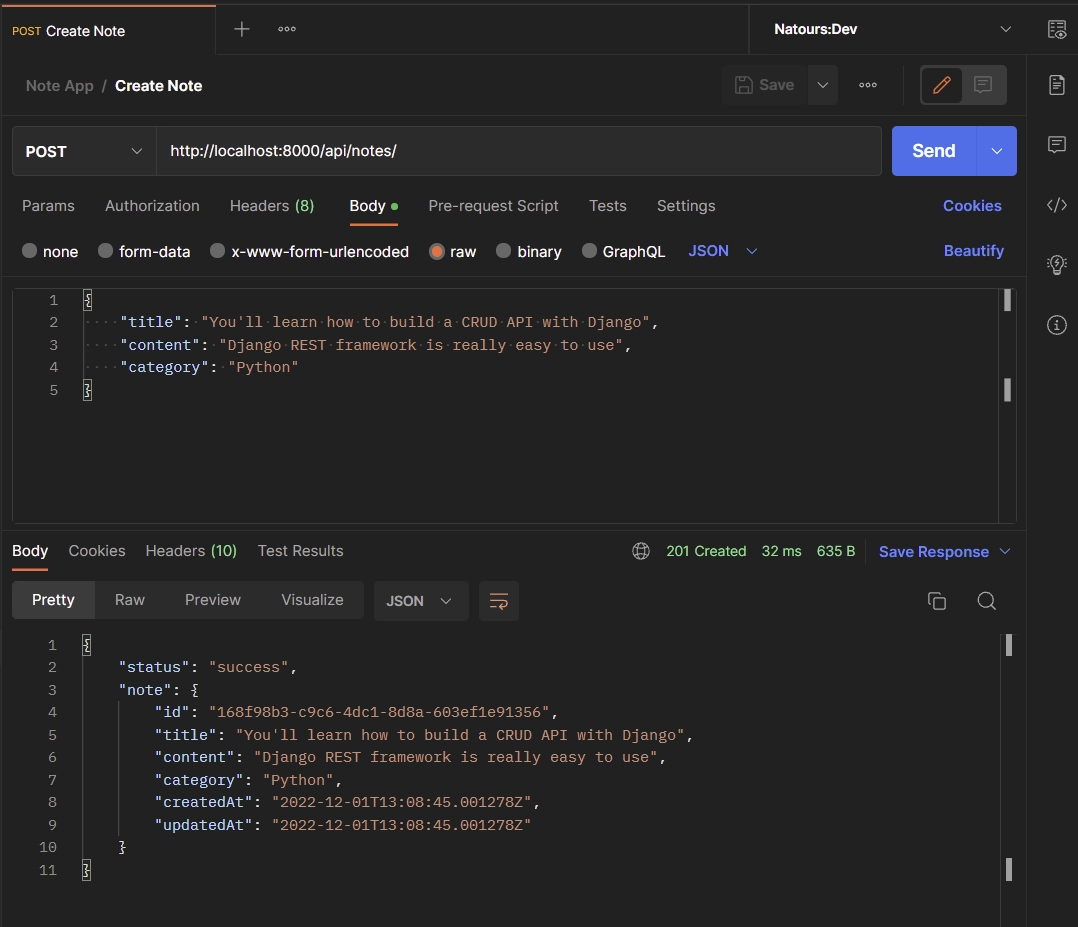
"title": "You'll learn how to build a CRUD API with Django",

"content": "Django REST framework is really easy to use",

"category": "Python"

}

The Django API will then validate the request body, add the new record to the database, and return the newly-created record to the API client or frontend app.



### **Update Note**

To edit an existing note in the database, make a PATCH request to the http://localhost:8000/api/notes/{noteId} endpoint with the new data provided in the JSON object.

{

"title": "This article will teach Django REST framework"

}

The Django API will validate the request payload, query the database to find the record that matches the provided ID, and update the fields in the found record.

### **Get All Notes**

To retrieve a list of records, make a GET request to the http://localhost:8000/api/notes/ endpoint. The Django API has a pagination feature so you can add a limit and page parameters in the URL to get a selected number of records.

If you don’t provide the page and limit parameters in the URL, the Django API will only return the first 10 records.

### **Delete Note**

To delete a note item in the database, make a DELETE request to the http://localhost:8000/api/notes/{noteId} endpoint.

The Django API will query the database to find the record that matches the provided ID and remove the found record from the database.

After that, a 204 status code will be returned to the client or frontend app to indicate that the record was deleted.