# Requirements

To be able to go through this set of activities, you should:
- Have at least python 3.9 installed on your local machine
- Have pip installed

Python includes a lightweight database called SQLite so you won't need to set up a database just yet.

To check if you have python installed, open your terminal and run the following command

**> python**

Or

**> python3**

This will open the python interactive shell and specify which version of python is installed.

Note: To exit from the python interactive shell: **crtl + D**

To check if you already have pip installed, run the following command:
**> pip -V**

Or

**> pip3 -V**

# Install Django

https://docs.djangoproject.com/en/5.1/topics/install/#installing-official-release

1. Install pip. The easiest is to use the standalone pip installer. If your distribution already has **`pip`** installed, you might need to update it if it's outdated. If it's outdated, you'll know because installation won't work.
2. Take a look at venv. This tool provides isolated Python environments, which are more practical than installing packages systemwide. It also allows installing packages without administrator privileges. The contributing tutorial walks through how to create a virtual environment.
3. After you've created and activated a virtual environment, or if you have decided to install is systemwide, enter the command:

**> python -m pip install Django**

# Verify your installation

# the first command will open in interactive shell and the remaining commands should be executed in that shell
> python
> import Django
> print(django.get_version())
5.1

Or run the command
> python -m django --version

# Install Django rest-framework

> pip install djangorestframework
> pip install markdown       # Markdown support for the browsable API.
> pip install django-filter  # Filtering support

# Start your first Django project

Choose a folder on you computer where you want to store your new project.

**Create the project folder:**
> mkdir <my project name>
NB: fill <my project name> with the name you want for your project

Or create you project folder using you computer interface if you want to avoid using the terminal

Now you should be in the terminal in the folder you've just created
> cd <my project name>

**Start a Django project**
Run the command below. This will create a project called <my API name> inside the <my project name> directory
**>** django-admin startproject <my API name> <my project name>

**Django project structure**
At this step you will find your self with the following structure
<my project name>/
    manage.py
    <my API name>/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py

These files are:

- **`manage.py`**: A command-line utility that lets you interact with this Django project in various ways. You can read all the details about **`manage.py`** in django-admin and manage.py.
- **`<my API name>/`**: A directory that is the actual Python package for your project. Its name is the Python package name you'll need to use to import anything inside it (e.g. **`<my API name>.urls`**).
- **`<my API name>/__init__.py`**: An empty file that tells Python that this directory should be considered a Python package. If you're a Python beginner, read more about packages in the official Python docs.
- **`<my API name>/settings.py`**: Settings/configuration for this Django project. Django settings will tell you all about how settings work.
- **`<my API name>/urls.py`**: The URL declarations for this Django project; a "table of contents" of your Django-powered site. You can read more about URLs in URL dispatcher.
- **`<my API name>/asgi.py`**: An entry-point for ASGI-compatible web servers to serve your project. See How to deploy with ASGI for more details.
- **`<my API name>/wsgi.py`**: An entry-point for WSGI-compatible web servers to serve your project. See How to deploy with WSGI for more details.

**Run the Django server**

> python manage.py runserver

You will get a message like this one:

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly
until you apply the migrations for app(s): admin, auth, contenttypes,
sessions.
Run 'python manage.py migrate' to apply them.
November 05, 2024 - 16:35:36
Django version 5.1.2, using settings 'django_api.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

NB: Note that you have the link to access your server through http: http://127.0.0.1:8000/

**Congratulations, you have started you Django server development !!**

**Run migration**
Note that when launching the server, you have in red: Run 'python manage.py migrate' to apply them.

Quit the server and run the command:
> python manage.py migrate

The output of the command will be like:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Notice that there's a new file in your project directory: **db.sqlite3.** It is the default database used by Django

Run migrate when:

- You've created or modified models and generated migrations using python manage.py makemigrations.
- You're setting up a new database, and migrations need to be applied to match your project's models.
- You're deploying an updated version of your project with schema changes.

**Create a Django application**
Now that your environment – a "project" – is set up, you're set to start doing work.

Each application you write in Django consists of a Python package that follows a certain convention. Django comes with a utility that automatically generates the basic directory structure of an app, so you can focus on writing code rather than creating directories.

Projects vs. apps
What's the difference between a project and an app? An app is a web application that does something – e.g., a blog system, a database of public records or a small poll app. A project is a collection of configuration and apps for a particular website. A project can contain multiple apps. An app can be in multiple projects.

To create your app, make sure you're in the same directory as **`manage.py`** and type this command:

> python manage.py startapp <my application name>
NB: fill <my application name> with the name of your application

That will create an application folder with the following structure

```
<my application name>/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

**Write the first view**

**Urls**
To define a URLconf for the `the application` app, create a file <my application name>/**`urls.py`** with the following content:

```
from django.urls import path

from . import views

urlpatterns = [
    path("test_json_view", views.test_json_view,
name="test_json_view"),
]
```

The content will look like

```
from django.urls import path

from . import views

urlpatterns = [
    path("test_json_view/", views.test_json_view, name="test_json_view"),
]
```

The next step is to configure the global URL
Open the urls file located at project level to include the url of your API. This is the routing mechanism to locate the right service.
**<my project name>/ <my API name>/urls.py**
In urlPatterns, add:
NB: male sure the path you are entering is correct and does exist otherwise it won't work

```
path("<my project name>/", include("<my project name>.urls"))
```

Do not forget to:

```
from django.urls import include, path
```

You will have a content similar to

```
from django.contrib import admin
from django.urls import path
from django.urls import include, path

urlpatterns = [
    path("data_lake_api/", include("data_lake_api.urls")),
    path('admin/', admin.site.urls),
]
```

**In the views.py file, add the following code**

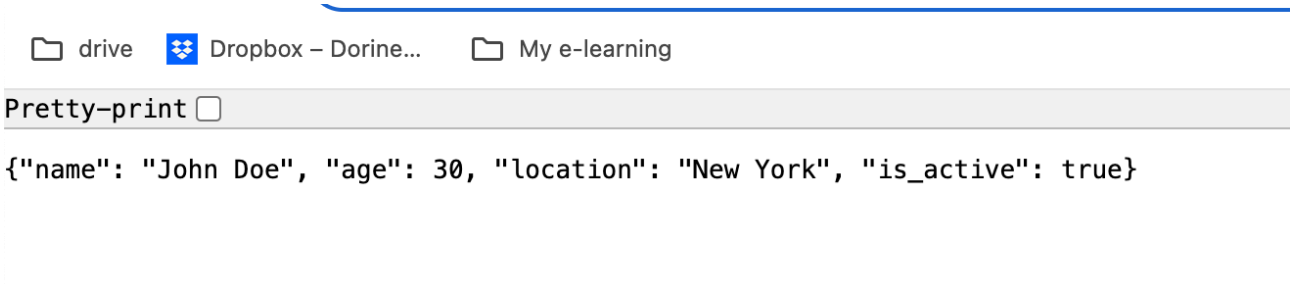
```
from django.shortcuts import render
import json

from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt

def test_json_view(request):
    data = {
        'name': 'John Doe',
        'age': 30,
        'location': 'New York',
        'is_active': True,
    }
    return JsonResponse(data)
```

run your server
> python manage.py runserver

Test the new route: open you browser: http://127.0.0.1:8000/data_lake_api/test_json_view/
Where data_lake_api is the name of the application and test_json_view is specified in the route as the name of the view

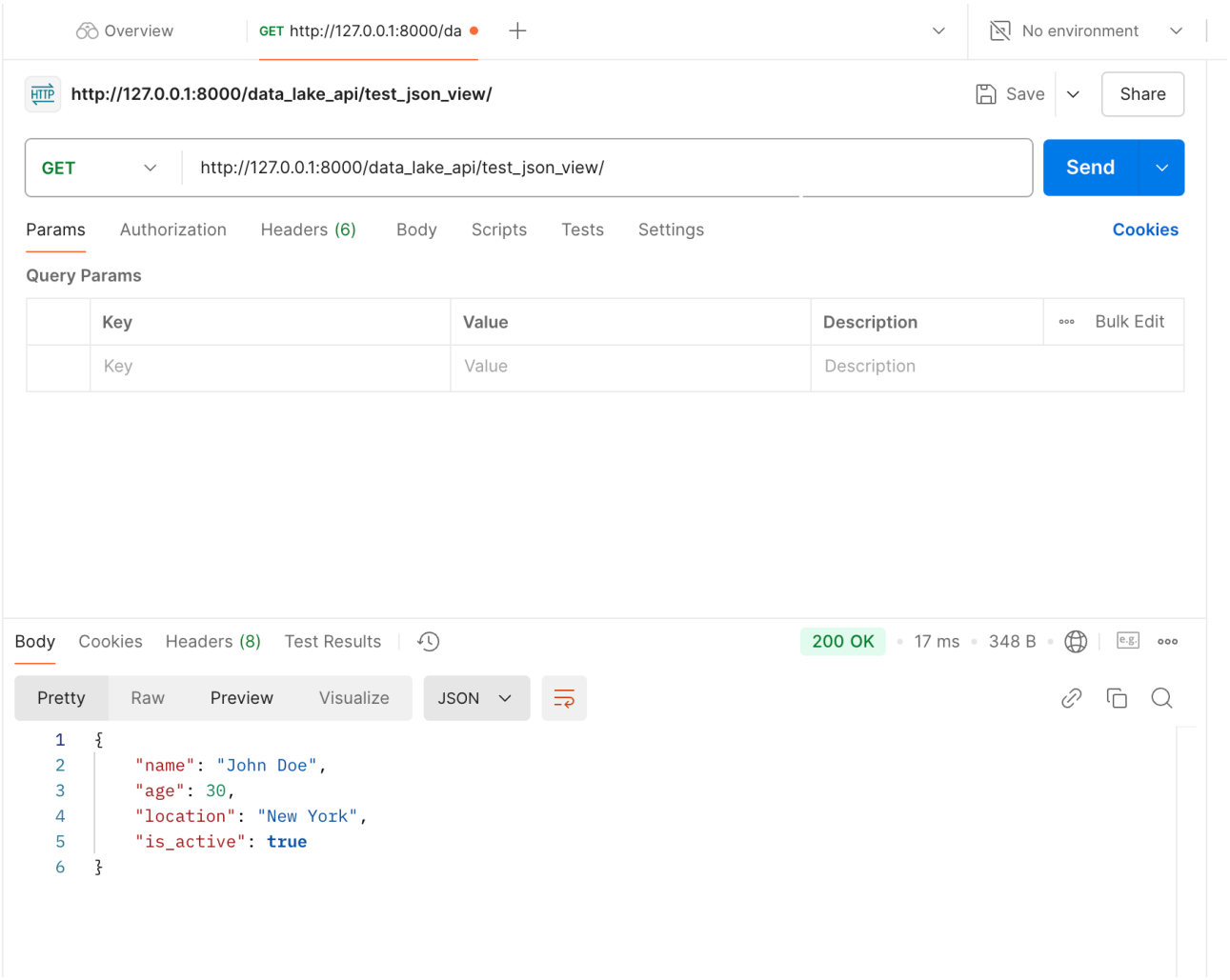



## You have added your first endpoint

**With Postman or with whatever the tool you have**
Here to test your endpoint, you used your browser. Test the same endpoint using postman or curl

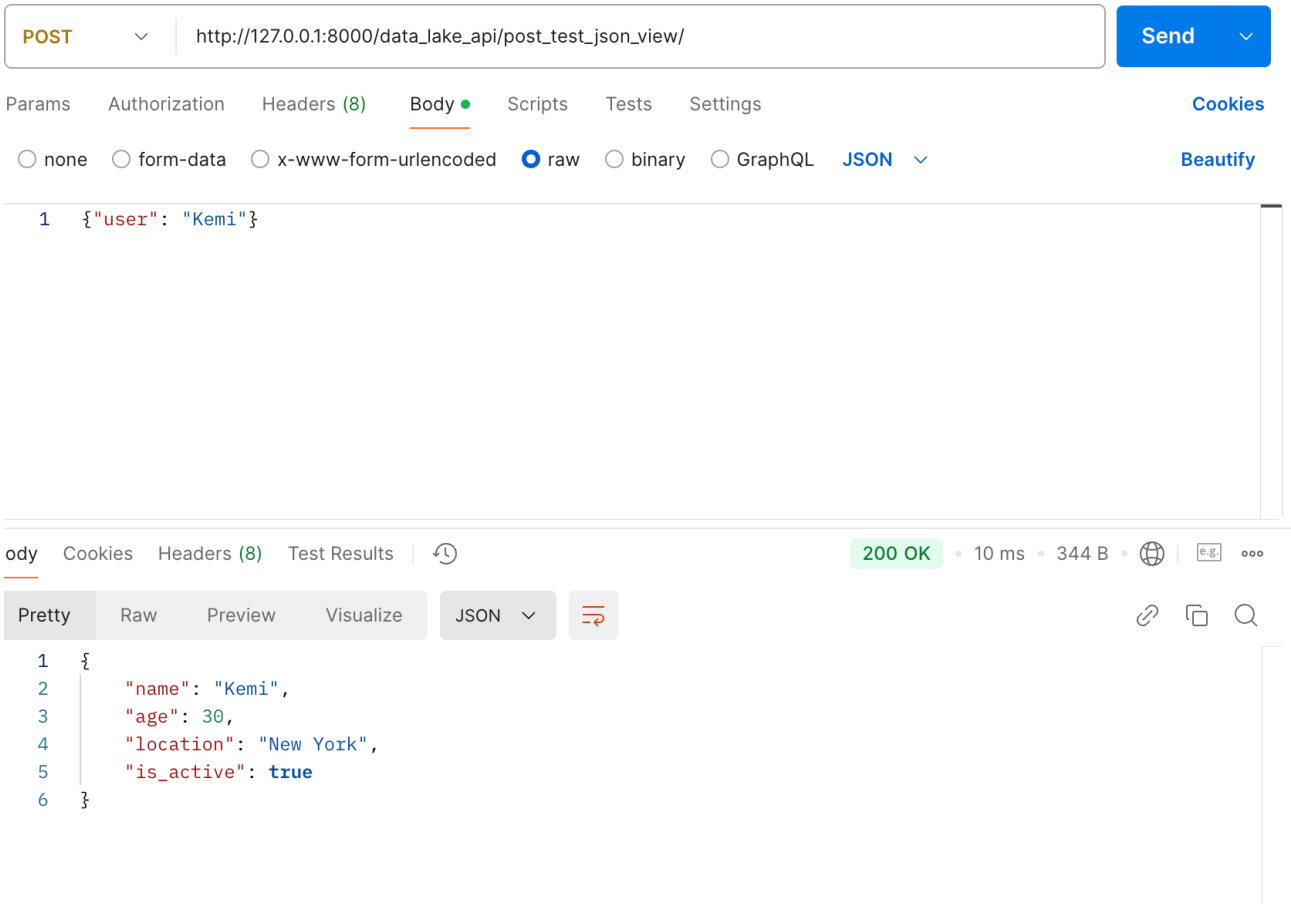
Install postman from here: https://www.postman.com/downloads/

Use it as on the screen


Overview
GET http://127.0.0.1:8000/da
No environment
http://127.0.0.1:8000/data_lake_api/test_json_view/
Save
Share
GET
http://127.0.0.1:8000/data_lake_api/test_json_view/
Send
Params
Authorization
Headers (6)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
200 OK
17 ms
348 B
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "name": "John Doe",
3 "age": 30,
4 "location": "New York",
5 "is_active": true
6 }

**Question**
We just added an endpoint to be used with GET.
Add a new url that can be used with a POST. The endpoint should send in the post a json body: {“user”: “a name”}
And the endpoint should return the same message as before, but the field name should be the one passed in input
Test your endpoint with postman or curl

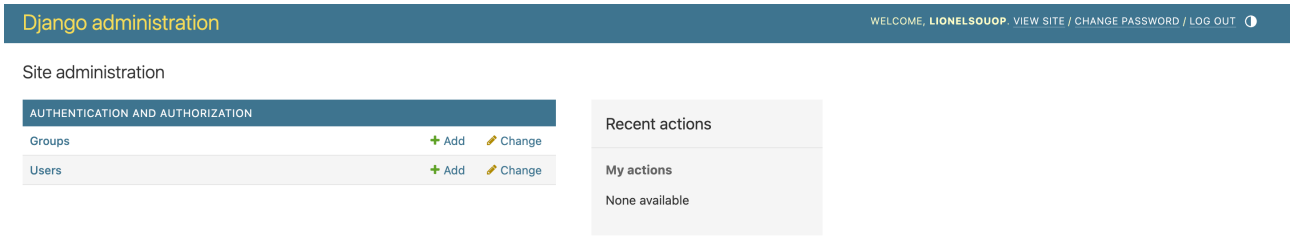
To be used as the following



**Database with sqlite3**
In this part, we will create and manipulate sql databases
For this, create an admin user running the command. You can shutdown your server first
> python manage.py createsuperuser

Then add user name, email and password and run again your server

On your browser go to http://127.0.0.1:8000/admin/ and connect
You will see a UI like the following




**Create your first model**
In the same project, open the models.py file and add the following content

```
class Product(models.Model):
    name = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    description = models.TextField(blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.name
```

This is the way to create relational table using Django models

**Fields**: Fields like CharField, DecimalField, and DateTimeField represent column types in the database.
**__str__ method**: This helps in representing the model instances as strings, which is useful in Django admin and Django shell.

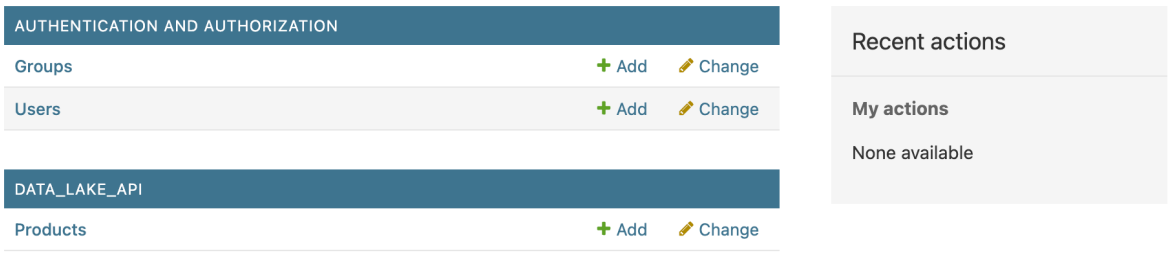
Open the settings.py file and in INSTALLED_APPS, add your application name in the list

**Make migrations**
Migrations are Django's way of propagating changes to your models (e.g., creating new tables, adding fields) to the database schema.
Run the following commands:
>python manage.py makemigrations
>python manage.py migrate

- makemigrations: Creates a migration file based on the changes in your models.
- migrate: Applies the migration, creating the actual database table for your model.

**Register the Model in admin.py**
Open the admin.py file

Now your new table is accessible via the UI




Now add some random products in the table products

**Question:**
Add an endpoint that will return all the products available in the Products table

**Question**:
Add an endpoint to retrieve the most expensive product

**Question:**
Add an endpoint to add a new product I the table

**Question**:
Add an endpoint to update a new product I the table

**Question:**
Create a new model to handle authorization. You want to control who is accessing your data in your table, and the rights will be stored in a table. Add rights to call the different endpoints you've created

**Question:**
Add at least 10 products in your DB. Your endpoints should return maximum 3 products at once. Implement a mechanism to progressively retrieve your data: a paging mechanism