Application Programming Interface (API)

An **API** (Application Programming Interface) is a set of rules and protocols that allows different software applications to communicate with each other. It acts as an intermediary, enabling the exchange of data and functionality between systems without exposing the underlying implementation details.

For example, when you use a third-party payment service like PayPal on an e-commerce site, an API facilitates the connection between the website and PayPal's servers to process the payment.

Introduction to REST

What is REST?

REST, or Representational State Transfer, is an architectural style for designing networked applications. REST relies on a stateless, client-server, cacheable communication protocol—almost always HTTP.

Key Principles of REST:

- **Stateless:** Each request from a client to the server must contain all the information needed to understand and process the request. The server doesn't store any state about the client session.
- **Client-Server:** The client and server operate independently. The client sends requests, and the server processes and responds.
- **Cacheable:** Responses from the server must define themselves as cacheable or non-cacheable to prevent clients from reusing stale or inappropriate data.
- **Uniform Interface:** REST relies on a uniform interface between components to decouple the architecture. This is achieved through standard HTTP methods like GET, POST, PUT, DELETE.

Overview of HTTP Methods

When building RESTful APIs, you'll use various HTTP methods to perform different actions:

- **GET:** Retrieve data from a specified resource.
  - Example: `GET /items/` retrieves a list of items.
- **POST:** Submit data to be processed to a specified resource.
  - Example: `POST /items/` creates a new item.
- **PUT:** Update a specified resource with new data.
  - Example: `PUT /items/1/` updates the item with ID 1.
- **DELETE:** Remove a specified resource.
  - Example: `DELETE /items/1/` deletes the item with ID 1.

3. Django REST Framework Introduction

Now that you have a basic understanding of REST, let's introduce Django REST Framework (DRF).

Installing Django REST Framework

To get started with DRF, you need to install it and add it to your Django project:

1. **Install DRF:**

```
pip install djangorestframework
```

2. **Add DRF to `INSTALLED_APPS`:**

   - Open `myproject/settings.py` and add `'rest_framework'` to the `INSTALLED_APPS` list:

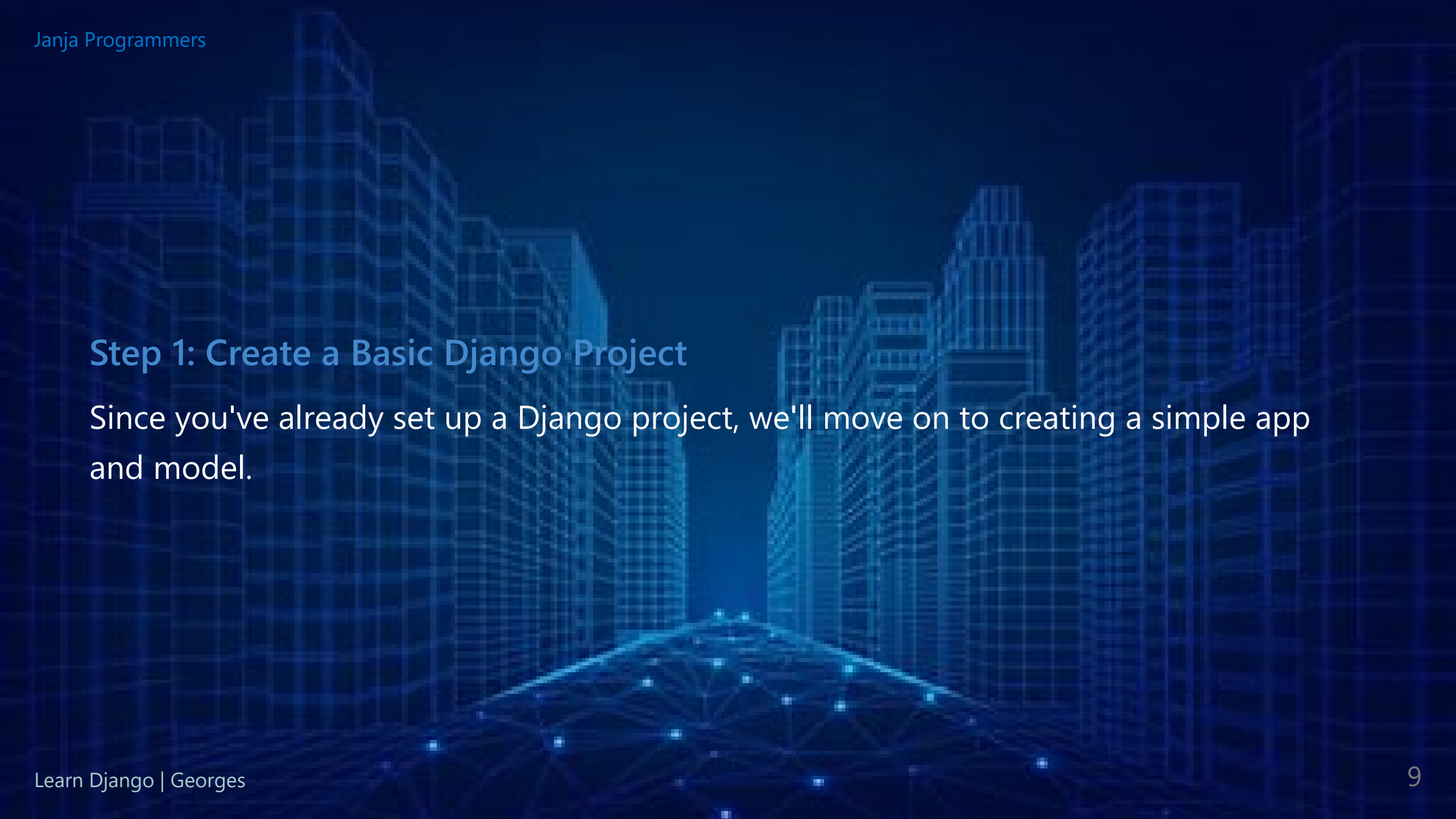
```
INSTALLED_APPS = [
    # other apps
    'rest_framework',
]
```

Overview of DRF Components

- **Serializers:** Convert complex data types, such as querysets and model instances, into native Python datatypes that can be easily rendered into JSON, XML, or other content types. Serializers also help in deserializing data back into complex types after validating the incoming data.
- **Views & ViewSets:** Define the logic for handling requests. Views handle individual endpoints, while ViewSets provide a way to automatically create CRUD views for your models.
- **Routers:** Automatically generate URL confs for your ViewSets, making it easier to create a RESTful API.

4. Building Your First API

Let’s put everything into practice by building a simple API.



Step 1: Create a Basic Django Project

Since you've already set up a Django project, we'll move on to creating a simple app and model.

Step 2: Set Up a Simple Django Model

Create a New App

In your project directory, run the following command to create a new app called `api`:

```
python manage.py startapp api
```

Define a Simple Model

Open `api/models.py` and define a simple model, `Item`:

```
from django.db import models

class Item(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()

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

Apply the Migrations

Run the following commands to create and apply the migrations for the `Item` model:

```
python manage.py makemigrations
python manage.py migrate
```

Create a Serializer for the Model

Create a `serializers.py` file

Inside the `api` app, create a new file called `serializers.py` and define a serializer for the `Item` model:

```
from rest_framework import serializers
from .models import Item

class ItemSerializer(serializers.ModelSerializer):
    class Meta:
        model = Item
        fields = '__all__'
```

Create a Basic API Endpoint Using DRF Views

Create a View

Open `api/views.py` and create a view using DRF's generic views:

```
from rest_framework import generics
from .models import Item
from .serializers import ItemSerializer

class ItemListCreate(generics.ListCreateAPIView):
    queryset = Item.objects.all()
    serializer_class = ItemSerializer
```

Set Up the URL Route

Open `api/urls.py` (create this file if it doesn't exist) and set up the route for the view:

```
    from django.urls import path
    from .views import ItemListCreate

    urlpatterns = [
        path('items/', ItemListCreate.as_view(), name='item-list-create'),
    ]
```

Then, include the `api/urls.py` in your main `myproject/urls.py`:

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

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

Test Your API

- Start the Django development server:

```
python manage.py runserver
```

- Visit `http://127.0.0.1:8000/api/items/` to see your API in action. You can use tools like Postman or cURL to send POST requests to this endpoint to create new items.

SUMMARY AND Q&A