Building Your First Django Project: Project Structure, Apps, and Admin Mastery

🛠️ Building Your First Django Project: Project Structure, Apps, and Admin Mastery

Now that you've understood what Django is and how it works at a high level, it's time to roll up your sleeves and build your first Django project. In this module, we'll walk you through the proper structure of a Django project, the concept of apps, how to register models, and how to use the Django admin panel like a pro. Django consulting services Django app development services

🗂️ Understanding the Django Project Structure

When you run the command `django-admin startproject myproject`, Django creates a base project directory. Let's break down what each file/folder does:

manage.py: A command-line utility that lets you interact with your project. You use it to run the server, make migrations, and more.

myproject/: This is the actual Python package for your project. It includes:

- **__init__.py:** Makes this folder a Python package.
- **settings.py:** The configuration file for your Django project. Includes database settings, installed apps, middleware, etc.
- **urls.py:** The URL dispatcher. It maps URLs to views.
- **wsgi.py/asgi.py:** Used for deploying your project to web servers.

It's important to keep your project clean and modular. Django encourages a structure where features are divided into reusable **apps**.

📦 What is a Django App?

In Django, a project is made up of one or more apps. Think of an app as a single unit of functionality. For example, a blog, a user profile section, or a payment system can each be their own app.

To create an app, you run: `python manage.py startapp core`

Inside the new `core` folder, you'll see files like:

models.py: Where you define your database schema using Python classes.

views.py: Contains functions or classes that respond to web requests.

admin.py: Used to customize how your models appear in the Django admin panel.

apps.py: Configuration for your app.

tests.py: Where unit tests are written.

migrations/: Auto-generated files that record changes in your models.

After creating an app, you must add it to your project's settings:

```
# settings.py
INSTALLED_APPS = [
    ...
    'core',
]
```

🗄️ Creating and Managing Models

Models are Python classes that define your database tables. Django comes with a powerful ORM (Object Relational Mapper) that lets you interact with the database using Python, without writing raw SQL.

Example model:

```
# core/models.py
from django.db import models

class Task(models.Model):
    title = models.CharField(max_length=100)
    completed = models.BooleanField(default=False)

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

After defining a model, run the following to apply changes:

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

This will create the corresponding table in the database.

🛡️ The Django Admin Interface

One of the best features of Django is its built-in admin interface. It allows you to manage your models through a user-friendly web interface.

To register a model in the admin:

```
# core/admin.py
from django.contrib import admin
from .models import Task

admin.site.register(Task)
```

Now, when you go to `http://localhost:8000/admin`, log in using your superuser account (created with `python manage.py createsuperuser`), you'll see your Task model listed and can add/edit/delete records.

You can also customize how the model appears by creating a ModelAdmin class:

```
class TaskAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'completed')
    search_fields = ('title',)

admin.site.register(Task, TaskAdmin)
```

🔗 Connecting URLs to Views

Views define what happens when a user accesses a URL. You need to map views to URLs using the `urls.py` file.

```
# core/views.py
from django.http import HttpResponse
```

```
def home(request):
    return HttpResponse("Hello from Django!")

# core/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home),
]

# myproject/urls.py
from django.contrib import admin
from django.urls import path, include

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

Now when you visit your local server, you'll see your custom homepage!

📌 Summary

Django projects are made of apps – each app focuses on one part of the site
Models define your data, and Django automatically handles the database
The admin interface is great for managing data visually
Views handle logic, URLs connect views to browser routes

This first hands-on experience is crucial. By understanding the structure of a Django project and working with models, views, and admin, you're now ready to start building actual features. Take your time, make mistakes, and push your code to GitHub regularly.

Up next, we'll explore how to build full CRUD APIs using Django REST Framework – your first real API!