# 📆 Semana 2 – Introducción al Proyecto + GitHub

**Clase 1 (4h) – Presentación + Organización de Grupos**

**Objetivos:** Comprender el proyecto, definir grupos, aprender a usar GitHub.

1. **Introducción al Proyecto Marketplace (45 min)**
   * Explicación del problema real: compraventa local.
   * Funcionalidades esperadas (usuarios, productos, pagos, chat, etc.).
   * Mostrar ejemplos (Wallapop, OLX, Marketplace de Facebook).
   * Debate: ¿Qué innovaciones podemos agregar?
2. **Organización de equipos (30 min)**
   * Formación de grupos de 2–3.
   * Elección de roles:
     + **Líder técnico** (coordina el repo).
     + **Dev backend**.
     + **Dev frontend**.
     + (Roles pueden rotar durante el proyecto).
3. **Introducción a GitHub (1h 15 min)**
   * Explicar repositorios, issues, ramas, PRs.
   * Crear cuenta GitHub (si no tienen).
   * Crear repositorio grupal: marketplace-grupoX.
   * Configurar README.md inicial con:
     + Descripción del proyecto.
     + Integrantes del grupo.
4. **Práctica guiada (1h 15 min)**
   * Primer git clone + git add + git commit + git push.
   * Cada alumno crea un archivo hola.txt con su nombre.
   * Subir cambios a rama propia y hacer **Pull Request**.

**Cierre (15 min):**

* Revisión en clase de los PRs.
* Entrega: repositorio con README.md + al menos 1 PR aprobado por grupo.

**Clase 2 (4h) – Flujo de Trabajo en GitHub**

**Objetivos:** Aprender buenas prácticas en Git/GitHub.

1. **Repaso rápido (20 min)**
   * Qué es un commit, una rama y un PR.
   * Que 2 grupos expliquen con sus palabras.
2. **Flujo de trabajo en equipo (1h)**
   * **Branching model** (main → feature branches).
   * Asignación de issues.
   * Revisión de PRs entre compañeros.
3. **Práctica de colaboración (1h 30 min)**
   * Crear issue “Agregar CONTRIBUTING.md”.
   * Cada alumno edita el archivo en su rama y crea PR.
   * Resolver conflictos intencionalmente (dos editan la misma línea).
   * Merge con aprobación de otro miembro.
4. **Documentación inicial (1h)**
   * Añadir a README.md:
     + Descripción detallada del proyecto.
     + Tecnologías que se usarán (Django, PostgreSQL, Render, Kivy, etc.).
     + Reglas de commits (ej: feat:, fix:, docs:).

**Entrega:** Repositorio grupal con README.md y CONTRIBUTING.md completo.

### 🐍 Código base – Clase 3 (Django + DRF + Users API)

# Crear proyecto Django

django-admin startproject marketplace

cd marketplace

# Crear app de usuarios

python manage.py startapp users

# Instalar dependencias

pip install djangorestframework psycopg2-binary python-decouple

**marketplace/settings.py**

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'rest\_framework',

'users',

]

# Configuración inicial REST Framework

REST\_FRAMEWORK = {

'DEFAULT\_PERMISSION\_CLASSES': [

'rest\_framework.permissions.AllowAny',

]

}

**users/models.py**

from django.db import models

class User(models.Model):

username = models.CharField(max\_length=50, unique=True)

email = models.EmailField(unique=True)

password = models.CharField(max\_length=128)

def \_\_str\_\_(self):

return self.username

**users/serializers.py**

from rest\_framework import serializers

from .models import User

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = '\_\_all\_\_'

**users/views.py**

from rest\_framework import viewsets

from .models import User

from .serializers import UserSerializer

class UserViewSet(viewsets.ModelViewSet):

queryset = User.objects.all()

serializer\_class = UserSerializer

**users/urls.py**

from rest\_framework.routers import DefaultRouter

from .views import UserViewSet

router = DefaultRouter()

router.register(r'users', UserViewSet)

urlpatterns = router.urls

**marketplace/urls.py**

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('api/', include('users.urls')),

]

Ahora ya tenés el endpoint:

* GET /api/users/ → lista todos los usuarios.
* POST /api/users/ → crea un usuario nuevo.

# Configuración para Render

### 1. Archivos necesarios en el proyecto

📂 **requirements.txt**

Django>=5.0

djangorestframework

psycopg2-binary

gunicorn

python-decouple

whitenoise

📂 **Procfile**

web: gunicorn marketplace.wsgi

📂 **runtime.txt**

python-3.11.5

📂 **.gitignore** (importante que lo agreguen en GitHub)

\*.pyc

\_\_pycache\_\_/

db.sqlite3

.env

### 2. Configuración en Django para PostgreSQL + Render

📂 **marketplace/settings.py** (agregar estas líneas)

import os

from decouple import config

import dj\_database\_url

SECRET\_KEY = config("SECRET\_KEY", default="unsafe-secret-key")

DEBUG = config("DEBUG", default=False, cast=bool)

ALLOWED\_HOSTS = ["\*"]

# BASE DE DATOS POSTGRES (Render)

DATABASES = {

"default": dj\_database\_url.config(

default=config("DATABASE\_URL")

)

}

# Archivos estáticos (para servir con whitenoise)

STATIC\_URL = '/static/'

STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')

MIDDLEWARE = [

"django.middleware.security.SecurityMiddleware",

"whitenoise.middleware.WhiteNoiseMiddleware", # servir estáticos en Render

"django.contrib.sessions.middleware.SessionMiddleware",

"django.middleware.common.CommonMiddleware",

"django.middleware.csrf.CsrfViewMiddleware",

"django.contrib.auth.middleware.AuthenticationMiddleware",

"django.contrib.messages.middleware.MessageMiddleware",

"django.middleware.clickjacking.XFrameOptionsMiddleware",

]

### 3. Variables de entorno en Render

Cuando crees el servicio en Render, en **Environment → Environment Variables**, cargá:

SECRET\_KEY=tu\_clave\_super\_secreta

DEBUG=False

DATABASE\_URL=postgres://usuario:contraseña@host:puerto/dbname

👉 El valor de DATABASE\_URL lo copiás directamente de tu base de datos PostgreSQL creada en Render.

### 4. Pasos de Deploy en Render

1. Subir el código a GitHub.
2. En Render: **New → Web Service**.
   * Elegir tu repo.
   * Build Command:
   * pip install -r requirements.txt && python manage.py collectstatic --noinput && python manage.py migrate
   * Start Command:
   * gunicorn marketplace.wsgi
3. Agregar variables de entorno (SECRET\_KEY, DATABASE\_URL, DEBUG).
4. Render va a levantar tu API en una URL tipo:
5. https://marketplace-grupo1.onrender.com
6. Probar endpoint público:
7. https://marketplace-grupo1.onrender.com/api/users/

### 5. 🚀 Próxima práctica para los alumnos

* Probar POST /api/users/ con Postman en la URL pública.
* Hacer un GET y ver que persiste en la DB de Render (PostgreSQL).
* Documentar la URL pública en su README.

### 🔹 Endpoints disponibles (Users API)

| **Método** | **Endpoint** | **Descripción** |
| --- | --- | --- |
| GET | /api/users/ | Lista todos los usuarios |
| POST | /api/users/ | Crea un usuario nuevo |
| GET | /api/users/:id/ | Obtiene un usuario por ID |
| PUT | /api/users/:id/ | Actualiza un usuario |
| DELETE | /api/users/:id/ | Elimina un usuario |

### 🔹 Probando con ****Postman****

Ejemplo de **crear usuario** (POST → https://tuservicio.onrender.com/api/users/):

{

"username": "juanperez",

"email": "juan@example.com",

"password": "123456"

}

Respuesta esperada (201 Created):

{

"id": 1,

"username": "juanperez",

"email": "juan@example.com",

"password": "123456"

}

👉 Hasta ahora **solamente por API** (GET/POST con Postman o navegador).

### 🔹 Agregamos un ****formulario simple en Django (frontend básico)****

Esto sirve para que los chicos vean cómo se conecta **frontend con backend**. Más adelante lo reemplazamos por React o mobile con Kivy, pero ya tienen una **vista rápida para testear**.

📂 **users/forms.py**

from django import forms

from .models import User

class UserForm(forms.ModelForm):

class Meta:

model = User

fields = ["username", "email", "password"]

📂 **users/views.py** (agregar función simple además del ViewSet)

from django.shortcuts import render, redirect

from .forms import UserForm

def user\_form(request):

if request.method == "POST":

form = UserForm(request.POST)

if form.is\_valid():

form.save()

return redirect("/api/users/") # redirige a la lista JSON

else:

form = UserForm()

return render(request, "user\_form.html", {"form": form})

📂 **users/templates/user\_form.html**

<!DOCTYPE html>

<html>

<head>

<title>Crear Usuario</title>

</head>

<body>

<h1>Registrar Usuario</h1>

<form method="POST">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Guardar</button>

</form>

</body>

</html>

📂 **users/urls.py** (agregar ruta del form)

from django.urls import path

from .views import user\_form

from rest\_framework.routers import DefaultRouter

from .views import UserViewSet

router = DefaultRouter()

router.register(r'users', UserViewSet)

urlpatterns = router.urls + [

path("form/", user\_form, name="user\_form"),

]

### 🔹 Ahora el flujo queda así:

* Los alumnos pueden:
  1. Usar **Postman** para probar GET/POST /api/users/.
  2. Entrar a /api/form/ y registrar un usuario vía formulario web.
  3. Ver los usuarios creados en /api/users/ (JSON).

Esto ya conecta los tres mundos:  
✔️ **Backend** (API REST)  
✔️ **Frontend básico (Django Templates)**  
✔️ **Cliente externo (Postman o navegador)**