

Taller final — Django

Proyecto integrador para evaluar modelos, ORM, vistas, plantillas, formularios, autenticación y pruebas.

Fecha de emisión: 15/02/2026

Modalidad: individual o en parejas (según definas en clase).

Cobertura: debe evidenciar los contenidos de las 4 sesiones del temario.

Este material continúa las bases de POO en Python (clases, encapsulación, herencia y polimorfismo) y las lleva a un contexto real con Django: modelos, vistas, plantillas, formularios, autenticación y pruebas.

Contenido

1. Contexto del proyecto	3
2. Reglas del taller	3
3. Requisitos funcionales (obligatorios)	4
3.1 Catálogo	4
3.2 Usuarios y roles	4
3.3 Préstamos y devoluciones	4
4. Requisitos técnicos (obligatorios)	5
5. Modelo de datos sugerido	6
6. Pruebas automatizadas (mínimo 12)	7
7. Checklist de completitud	8
8. Entregables	8
9. Rúbrica (100 puntos)	8

1. Contexto del proyecto

Vas a construir un sistema web para una biblioteca. El sistema debe permitir administrar un catálogo de libros, usuarios miembros, préstamos y devoluciones. El foco del taller es evidenciar tus decisiones técnicas (modelo, validación, permisos) y tu capacidad de entregar software reproducible (README, migraciones, tests).

2. Reglas del taller

- Debes usar Django y el patrón MVT (vistas + templates).
- Debes tener al menos 2 apps separadas (sugeridas: catalog y loans).
- Debes tener control de acceso por rol (visitante / autenticado / staff).
- Debes incluir pruebas automatizadas (mínimo 12).
- Debe ser ejecutable desde cero siguiendo tu README (sin pasos mágicos).

3. Requisitos funcionales (obligatorios)

3.1 Catálogo

- Listado público de libros con búsqueda por título y por autor (case-insensitive).
- Detalle de libro con: título, autor, ISBN, estado (disponible/prestado) y fecha del último préstamo si aplica.
- Paginación en listado (mínimo: 10 por página).

3.2 Usuarios y roles

- Login y logout (puedes usar django.contrib.auth).
- Página 'Mis préstamos' para usuarios autenticados.
- Solo staff puede crear/editar/eliminar libros y gestionar préstamos.

3.3 Préstamos y devoluciones

- Un préstamo vincula un usuario y un libro.
- Regla crítica: un libro no se puede prestar si ya está prestado.
- Devolución: marca el préstamo como finalizado y el libro como disponible.
- Retraso: si se devuelve tarde, registra una multa simple: monto = días * 1000.

4. Requisitos técnicos (obligatorios)

- Modelos con relaciones (ForeignKey) y migraciones aplicadas.
- Al menos 1 CRUD completo (crear, listar, ver detalle, editar, eliminar) — recomendado: Book.
- Formularios con validación: mínimo 1 validación propia (clean_*) y mensajes al usuario.
- Plantillas con herencia: base.html + 3 templates derivados.
- Archivos estáticos: CSS mínimo y estructura static/ correcta.
- Admin configurado para operar datos (list_display, search_fields, list_filter).
- Pruebas automatizadas: mínimo 12 (sección 6).

Recomendación: mantén el dominio simple y consistente. Un taller incompleto con muchas 'ideas' suele puntuar menos que uno pequeño bien cerrado.

5. Modelo de datos sugerido

Puedes ajustar el modelo, pero debe cubrir el flujo. Sugerencia mínima:

Entidad	Campos mínimos
Author	name
Book	title, author(FK), isbn(unique), is_available(bool) o lógica equivalente
Loan	book(FK), user(FK), start_date, due_date, end_date(null), is_active
Fine	loan(OneToOne), late_days, fine_amount

Regla crítica: define una sola fuente de verdad para la disponibilidad (campo booleano o derivación desde préstamos activos). Si usas campo booleano, garantiza consistencia al prestar/devolver (idealmente en una transacción).

6. Pruebas automatizadas (mínimo 12)

Las pruebas deben ejecutarse con `python manage.py test`. Se evalúa que cubran reglas y permisos (no solo que existan).

- Modelos (4): creación válida, ISBN inválido, regla de disponibilidad, cálculo multa.
- Vistas públicas (2): catálogo responde 200; búsqueda filtra.
- Acceso (3): visitante bloqueado para CRUD; autenticado no staff bloqueado; staff permitido.
- Flujo (3): prestar cambia estado; no se presta dos veces; devolver reactiva y crea multa si aplica.

```
# Ejecutar pruebas
python manage.py test

# Tip docente: si tus tests son frágiles, agrega datos claros y aserciones específicas.
```

7. Checklist de completitud

- El proyecto corre local: `python manage.py runserver` sin errores.
- Migraciones limpias y aplicables desde cero.
- Admin operativo (crear/editar/libros/autores).
- CRUD de libros completo (con permisos).
- Préstamo y devolución implementados + regla de no doble préstamo.
- Página 'Mis préstamos' disponible para autenticados.
- ≥ 12 tests, todos pasando.
- README con pasos reproducibles.

8. Entregables

- Repositorio (o zip) con el proyecto Django completo.
- README.md con: instalar, migrar, crear superusuario, correr servidor, correr tests.
- requirements.txt con dependencias.
- Evidencia de pruebas: salida de `python manage.py test`.
- Evidencia visual: 3 capturas (catálogo, detalle, panel staff) o video corto (opcional).

9. Rúbrica (100 puntos)

Criterio	Puntos	Qué se observa
Modelo de datos + migraciones + ORM	25	Relaciones correctas, consistencia, consultas coherentes
Vistas + templates + navegación	18	MVT bien aplicado, herencia, navegación clara
CRUD + formularios + validación	17	ModelForms, validación propia, mensajes y errores
Auth + permisos	15	Rutas protegidas por rol; staff vs usuario
Préstamos (regla crítica)	10	No doble préstamo, devolución correcta, estado consistente
Pruebas automatizadas	12	≥ 12 tests relevantes, cubren reglas críticas
Calidad general	3	Estructura, nombres, README, ejecución reproducible

Nota: el examen práctico puede ser un subconjunto del taller (por ejemplo: implementar préstamo/devolución + permisos + 2-3 tests).