Django Training Manual

Please go through the linked articles, videos, and documentation. Explore the tools. Ask if anything is not clear.

Note: Each week includes tasks that are aligned with what you've already learned or can learn through the resources provided. For example, in Week 2, you'll build a Python To-Do List app with skills learned in Week 1 and start working on a Django project after completing Week 2 resources.

Prerequisites & Tools

1. **Operating System**: Windows, macOS, or Linux.
2. **Python**: Version 3.8+ (Install from python.org).
3. **Package Manager**: `pip` (comes bundled with Python 3).
4. **Virtual Environment** (recommended):
    - Use `venv` (bundled with Python) or a tool like `virtualenv`.
    - Example: `python -m venv env` then `source env/bin/activate` (macOS/Linux) or `env\Scripts\activate` (Windows).
5. **Git** (version control):
    - Install from git-scm.com.
6. **GitHub** account:
    - Create one at github.com.
7. **Text Editor/IDE**:
    - Recommended: VS Code, PyCharm, or any editor you're comfortable with.

Week 1: Python Fundamentals and Git/GitHub

Goal

Establish a strong foundation in **Python** programming and **Git/GitHub** version control.

Requirements & Details

1. **Python Basics**: Variables, data types, loops, conditionals, functions, file handling.
2. **Git Basics**: Cloning, branching, committing, pushing, pulling.

Learning Resources

- **Python**:
    - Udemy Course: Complete Python Bootcamp.
    - Official Python Docs: docs.python.org. (Not mandatory)
- **Git/GitHub**:
    - Tutorial: Git and GitHub for Beginners (HubSpot).
    - Official Git Docs: git-scm.com/docs. (Not mandatory)

Tasks

1. **Set Up Python**:
    - Install Python 3.8+.
    - Create and activate a virtual environment (`venv`).
2. **Practice Python**:
    - Write small Python scripts to demonstrate:
        - **Data types** (int, float, str, list, dict).
        - **Functions** (parameters, return values).
        - **File handling** (read/write to a `.txt` file).
3. **Learn Git Workflow**:
    - Initialize a local Git repository (`git init`).
    - Create a new repository on GitHub.
    - Practice commit, push, pull, branch management.

Deliverables

- **Python Scripts**: Show proficiency in variables, loops, lists, functions, file handling (e.g., a script that reads a `.txt` file and prints each line).
- **GitHub Repository**: A sample Python project pushed to GitHub with a clear commit history (at least 3 meaningful commits).

Week 2: Building a Python To-Do List App & Setting Up Django Project

Goal

Reinforce Python skills with a CLI-based **To-Do List** app and establish the **foundation** of the E-learning site in Django.

Requirements & Details

1. **CLI To-Do List**:
    - Python-based, modular code structure.
    - Use a JSON file (`tasks.json`) or a simple database for persistence.
2. **E-learning Site (Django)**:
    - **Install Django**: `pip install django`.
    - Create a Django project: `django-admin startproject elearning`.
    - Create an app for user authentication (e.g., `accounts`).
    - Implement **user signup, login, logout**.
    - Configure a basic **model**, **view**, **template**, and **admin panel**.

Learning Resources

- **Django Fundamentals**:
    - Udemy Course: Django for Beginners.
    - Udemy Course: Django For Advanced.
    - Official Django Docs: docs.djangoproject.com. (Not mandatory)

Tasks

1. **Python To-Do List Application**:
    - Features:
        - Add tasks.
        - View tasks (mark as done or not done).
        - Update tasks (mark as done).
        - Delete tasks.
    - Acceptance Criteria:
        - **Persistence**: Tasks remain after the script terminates (e.g., via `tasks.json`).
        - **Modular Code**: Separate logic into functions or classes.
2. **E-learning Site (Initial Setup)**:
    - Initialize Django project: `django-admin startproject elearning`.
    - Create a Django app (e.g., `python manage.py startapp accounts`).
    - Configure **urls.py**, **views.py**, **templates** to enable:
        - User **signup** (create new account).
        - User **login** (authenticate).
        - User **logout**.
    - Verify the admin panel is accessible (`/admin`) and functional.

Deliverables

- **Python To-Do List**: A CLI app pushed on GitHub with a clear README.
- **E-learning Site** (Version 1):
    - Basic project structure committed to GitHub.
    - Working authentication system (signup, login, logout).

Week 3: Advanced Features & REST APIs with Django REST Framework

Goal

Enhance the E-learning site with **course management**, **advanced features**, and **REST API integration** using Django REST Framework (DRF).

Requirements & Details

1. **DRF Installation**: `pip install djangorestframework`.
2. **Models & CRUD**:
    - Create Django models for **Course**, **Lecture**.
    - Implement CRUD (Create, Read, Update, Delete) views and templates.
3. **REST API**:
    - Integrate DRF into your Django project.
    - Create **serializers** for models.
    - Implement API views (function-based or class-based) for listing and managing courses.
    - Add **Token Authentication** or **Session Authentication**.
4. **Advanced Django Features**:
    - Use **signals** (e.g., send an email when a student enrolls in a course).
    - Implement **database optimizations** (`select_related`, `prefetch_related`).
5. **Testing**:
    - Write **unit tests** for models, views, and APIs (use `pytest` or Django's built-in `unittest`).

Learning Resources

- **Django REST Framework**:
    - Official Docs: www.django-rest-framework.org.
- **Database Optimizations**:
    - Django Docs on Querysets.

Tasks

1. **Course Management**:
    - Models: `Course` (title, description), `Lecture` (title, content, course FK).
    - CRUD views: `ListView`, `DetailView`, `CreateView`, `UpdateView`, `DeleteView`.
    - Templates: Simple HTML forms for create/update, listing lectures, etc.
2. **REST APIs**:
    - Create **serializers** (e.g., `CourseSerializer`, `LectureSerializer`).
    - Define **API views** (or ViewSets).
    - Implement **Token Authentication**: `pip install djangorestframework-simplejwt` (or use DRF's built-in tokens).
3. **Advanced Features**:
    - **Signals** for sending post-enrollment email notifications.
    - **Optimized queries**: Demonstrate usage of `select_related()`, `prefetch_related()`.
4. **Testing & Deployment**:
    - **Unit Tests**: Achieve at least **80% coverage**.

Deliverables

- **REST API-Integrated E-learning Site**:
    - Course/lecture management (CRUD).
    - REST endpoints for courses and lectures.
    - Token-based authentication and permissions (instructor vs student).
- **Basic Testing Suite**: 80% test coverage minimum.

Week 4: Custom Models, Middleware, and Advanced Queries

Goal

Deepen understanding of **custom Django models**, **middleware**, and **query optimizations**.

Requirements & Details

1. **Custom User Model**:
    - Extend Django's default `AbstractUser` or `AbstractBaseUser`.

    - Additional fields (e.g., `role`: student/instructor).
    - Overriding `save()` and using signals.
2. **Custom Managers**:
    - Write model managers for specialized queries (e.g., `CourseManager`).
3. **Middleware**:
    - Create a custom middleware for logging or custom authentication.
4. **Query Optimization**:
    - Advanced queries: `get_or_create()`, `bulk_create()`, further use of `select_related()`.

Tasks

1. **Implement Custom User Model**:
    - Migrate existing user data if needed (or start fresh).
    - Add extra fields and logic in `save()` method.
2. **Custom Managers**:
    - For example, `Course.objects.popular()` that returns courses sorted by enrollment.
3. **Middleware**:
    - Example: Logging request data or checking user roles.
4. **Document Query Optimizations**:
    - Provide examples in your README or wiki showing performance improvements.

Deliverables

- **Enhanced E-learning Site**:
    - Custom user model with additional fields.
    - Middleware integrated into `settings.py`.
    - Demonstrable performance improvements (screenshots or logs).
- **Documentation** of advanced queries and any performance metrics.

Week 5: Management Commands & Admin Customization

Goal

Automate tasks with **Django management commands** and enhance the **admin panel** experience.

Requirements & Details

1. **Management Commands**:
    - Insert dummy data (users, courses, enrollments).
    - Automate scheduled tasks using `cron`.
2. **Admin Panel Customization**:
    - Add search, filters, and custom list displays.
    - Customize admin forms for courses, users, and enrollments.
3. **REST API Refinement**:
    - Explore advanced class-based views (`APIView`, `GenericAPIView`).
    - Refine endpoints (e.g., add pagination, limit fields).

Tasks

1. **Write Management Commands**:
    - `python manage.py create_dummy_data` to create test users/courses.
    - Automate via cron job or scheduler once a day/week.
2. **Admin Panel**:
    - Improve Course and User listings with search on `title`, `username`, etc.
    - Add inline model management for Lectures under Courses.
3. **REST API**:
    - Use `GenericAPIView` and mixins (ListModelMixin, CreateModelMixin, etc.).
    - Document endpoints in your repository wiki or README.

Deliverables

- **Management Commands**:
    - Example usage in README: `python manage.py create_dummy_data`.
    - Cron or scheduled tasks in production environment.
- **Enhanced Admin Panel**:
    - Custom actions, filters, and improved UI for admin users.
- **Refined REST APIs**:
    - Pagination, field filtering, or sorting for courses.

Week 6: Finalizing the E-Learning Site

Goal

Develop robust **REST APIs** for user/course management, implement **comprehensive testing**, and **deploy** the final version of the E-learning site.

Requirements & Details

1. **Complete REST APIs**:
    - **User Authentication**: JWT-based or Token-based.
    - **Course Management**: Listing, creating, updating, deleting.
    - **Enrollment & Progress Tracking**.
    - Filters and sorting with `django-filters`.
2. **Social Login** (optional advanced feature).
3. **Testing**:
    - Comprehensive test coverage (models, views, APIs).
    - Coverage report generation.
4. **API Documentation**:
    - Use **Swagger** or **Redoc** (via DRF's built-in schemas or `drf-yasg`).
5. **Final Deployment**:
    - Production-level deployment on AWS, or any other platform.
    - Proper static/media file handling and environment variable management.

Tasks

1. **API Development**:
    - Implement **JWT authentication** (using `djangorestframework-simplejwt`) or a similar package.
    - Add endpoints for **enrollment** and **progress** (e.g., `/api/enroll/`, `/api/courses/progress/`).
    - Use `GenericAPIView` and mixins for streamlined design.
2. **Filters & Sorting**:
    - Integrate `django-filters` for advanced query parameters (e.g., filter courses by level, date created).
3. **Social Login** (optional):
    - Use `django-allauth` or a similar library.
    - Configure Google or Facebook OAuth.
4. **Testing & Documentation**:
    - Achieve **80%+** coverage using `coverage.py`.
    - Generate and host API docs via **Swagger** or **Redoc**.
5. **Deploy**:
    - Ensure environment variables are secure (`.env` file or platform-specific config).
    - Confirm static files served correctly.
    - Final push to production.

Deliverables

- **Production-Ready E-Learning Site**:
    - Authentication, course management, enrollment, progress tracking.
    - Social login (if implemented).
- **Code Quality**:
    - Minimum **80%** test coverage.
    - Modular, maintainable code (comments, docstrings).
- **Documentation**:
    - **API Docs** accessible via Swagger or Redoc.
    - Detailed README and/or Wiki in GitHub repository.
- **Deployed Application**:
    - Live URL where the site can be tested.
    - Working static/media handling.