Django Internship Assignment Report & Implementation Guide

June 13, 2025

Abstract

This report details the implementation of a Django-based internship assignment, encompassing a production-ready Django project setup, API development with Django REST Framework (DRF), Celery integration for background tasks, Telegram bot integration, and code management practices. The project demonstrates secure configuration, user authentication, asynchronous task processing, and external service integration, adhering to Python and Django best practices.

Contents

1 Introduction

The Django Internship Assignment is designed to showcase proficiency in backend development using Django and related technologies. The project involves creating a Django application with a RESTful API, integrating Celery for background tasks, incorporating a Telegram bot to capture user data, and maintaining clean, documented code in a GitHub repository. This report outlines the implementation steps, code structure, and considerations for each component.

1.1 Objectives

- Configure a production-ready Django project with secure settings.
- Develop public and protected API endpoints using DRF.
- Integrate Celery with Redis for asynchronous task processing.
- Implement a Telegram bot to collect and store user data.
- Manage code with Git, ensuring proper documentation and commit history.

2 Project Components

2.1 Django Project Setup

The project begins with a robust Django setup using Django REST Framework (DRF) and production-ready configurations.

2.1.1 Implementation

1. **Create Project and App**: Initialize the project and app using:

```
django-admin startproject myinternship
cd myinternship
python manage.py startapp core
```

2. **Install Dependencies**: Install DRF and environment variable management:

```
pip install djangorestframework python-decouple
```

   Add `rest_framework` and `core` to `INSTALLED_APPS` in `settings.py`.

3. **Production Settings**: Configure `settings.py` for security:

```
from decouple import config

SECRET_KEY = config('SECRET_KEY')
DEBUG = config('DEBUG', default=False, cast=bool)
ALLOWED_HOSTS = ['127.0.0.1', 'localhost']
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': config('DB_NAME'),
        'USER': config('DB_USER'),
        'PASSWORD': config('DB_PASSWORD'),
        'HOST': config('DB_HOST'),
        'PORT': config('DB_PORT', default='5432', cast=int),
    }
}
```

Create a `.env` file (excluded via `.gitignore`):

```
SECRET_KEY=your_super_secret_django_key
DEBUG=True
DB_NAME=myinternship_db
DB_USER=django_user
DB_PASSWORD=your_db_password
DB_HOST=localhost
DB_PORT=5432
```

2.2 API Development

Two API endpoints are implemented: a public endpoint and a protected endpoint using Token Authentication. Djangos built-in login supports web-based access.

2.2.1 Implementation

1. **Serializers**: Define in `core/serializers.py`:

```
from rest_framework import serializers
from django.contrib.auth.models import User

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'username', 'email', 'password']
        extra_kwargs = {'password': {'write_only': True}}

    def create(self, validated_data):
        user = User.objects.create_user(**validated_data)
        return user

class PublicDataSerializer(serializers.Serializer):
    message = serializers.CharField(max_length=200)
    timestamp = serializers.DateTimeField(read_only=True)
```

2. **Views**: Implement in `core/views.py`:

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from rest_framework.permissions import IsAuthenticated
from rest_framework.authentication import TokenAuthentication
from rest_framework.generics import CreateAPIView
from datetime import datetime
```

```python
from .serializers import PublicDataSerializer, UserSerializer

class PublicView(APIView):
    def get(self, request):
        data = {'message': 'This is a public endpoint, accessible to everyone.', 'timestamp': datetime.now()}
        serializer = PublicDataSerializer(data)
        return Response(serializer.data, status=status.HTTP_200_OK)

class ProtectedView(APIView):
    authentication_classes = [TokenAuthentication]
    permission_classes = [IsAuthenticated]
    def get(self, request):
        return Response({'message': f'Hello {request.user.username}, this is a protected endpoint!'})

class UserRegisterView(CreateAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    permission_classes = []
```

3. **URLs**: Configure in `myinternship/urls.py` and `core/urls.py`:

```python
# myinternship/urls.py
from django.contrib import admin
from django.urls import path, include
from rest_framework.authtoken.views import obtain_auth_token

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('core.urls')),
    path('api-token-auth/', obtain_auth_token),
    path('api-auth/', include('rest_framework.urls')),
]

# core/urls.py
from django.urls import path
from .views import PublicView, ProtectedView, UserRegisterView

urlpatterns = [
    path('public/', PublicView.as_view(), name='public_endpoint'),
    path('protected/', ProtectedView.as_view(), name='protected_endpoint'),
    path('register/', UserRegisterView.as_view(), name='user_register'),
]
```

4. **Web-based Login**: Enabled via `api-auth/` for DRFs browsable API.

2.3 Celery Integration

Celery with Redis handles background tasks, such as sending a welcome email after user registration.

2.3.1 Implementation

1. **Install Dependencies**:

```
pip install celery redis
```

2. **Configure Celery**: Create `myinternship/celery.py`:

```
import os
from celery import Celery
from django.conf import settings

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'myinternship.settings')
app = Celery('myinternship')
app.config_from_object('django.conf:settings', namespace='CELERY')
app.autodiscover_tasks(lambda: settings.INSTALLED_APPS)
```

Update `myinternship/`$_{init.py}$:

`@shared`$_t askdefsend_r egistration_e mail(user_e mail, username) : print(f"Simulatingsendingemailtouser_e ma$

Trigger Task: Update `UserRegisterView` in `core/views.py`:

```
from .tasks import send_registration_email

class UserRegisterView(CreateAPIView):
    def perform_create(self, serializer):
        user = serializer.save()
        send_registration_email.delay(user.email, user.username)
```

2.4 Telegram Bot Integration

A Telegram bot captures user data upon receiving a `/start` command and stores it in the Django database.

2.4.1 Implementation

1. **Create Bot**: Obtain a token from @BotFather on Telegram and add to `.env`:

```
TELEGRAM_BOT_TOKEN=YOUR_TELEGRAM_BOT_API_TOKEN
```

Install `python-telegram-bot`:

```
pip install python-telegram-bot
```

2. **Model**: Define in `core/models.py`:

```python
from django.db import models

class TelegramUser(models.Model):
    telegram_username = models.CharField(max_length=255, unique=
       True, null=True, blank=True)
    chat_id = models.CharField(max_length=255, unique=True)
    first_name = models.CharField(max_length=255, null=True,
       blank=True)
    last_name = models.CharField(max_length=255, null=True,
       blank=True)
    date_joined = models.DateTimeField( auto_now_add=True)

    def __str__(self):
        return self.telegram_username or f"Chat␣ID:␣{self.
           chat_id}"
```

Run migrations:

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

3. **Bot Handler**: Create `telegram`$_b ot_l istener.py$:

`async def start`$_c ommand(update : Update, context : ContextTypes.DEFAULT_T YPE) - >$
$None : user_t g = update.effective_u serchat_i d = update.effective_c hat.idtelegram_u sername =$
$user_t g.usernamefirst_n ame = user_t g.first_n amelast_n ame = user_t g.last_n ametry : telegram_u ser, created =$
$awaitTelegramUser.objects.aget_o r_c reate(chat_i d = str(chat_i d), defaults = 'telegram_u sername' : telegram_u se$
$updated = Falseiftelegram_u ser.telegram_u sername! = telegram_u sername : telegram_u ser.telegram_u sername$
$telegram_u sernameupdated = Trueiftelegram_u ser.first_n ame! = first_n ame : telegram_u ser.first_n ame =$
$first_n ameupdated = Trueiftelegram_u ser.last_n ame! = last_n ame : telegram_u ser.last_n ame =$
$last_n ameupdated = Trueifupdated : awaittelegram_u ser.asave()awaitupdate.message.reply_t ext(f"Hellofirs$
$awaitupdate.message.reply_t ext("Anerroroccurred.Pleasetryagain.")$

`def main():  application = Application.builder().token(config('TELEGRAM`$_B OT_T OKE$
$Update.ALL_T YPES)$

`if` $_n ame_{=="}{}_{main_{"}:main()}$

2.5 Code Management

The project is maintained in a GitHub repository with clean, documented code and a detailed README.

2.5.1 Implementation

1. **Git Setup**:

```
git init
git add .
git commit -m "Initial␣Django␣project␣setup␣with␣DRF"
```

2. **.gitignore**:

```
.env
__pycache__/
*.pyc
*.sqlite3
venv/
```

3. **README**: Create `README.md`:

~~~
# Django Internship Assignment

## Overview
A Django project demonstrating API development, Celery
   integration, Telegram bot functionality, and code management.

## Setup
1. Clone: `git clone https://github.com/your_username/your_repo.
   git`
2. Install: `pip install -r requirements.txt`
3. Configure `.env`:
```
SECRET_KEY=your_key
DEBUG=True
DB_NAME=myinternship_db
DB_USER=django_user
DB_PASSWORD=your_password
DB_HOST=localhost
DB_PORT=5432
TELEGRAM_BOT_TOKEN=your_token
CELERY_BROKER_URL=redis://localhost:6379/0
CELERY_RESULT_BACKEND=redis://localhost:6379/0
```
4. Migrate: `python manage.py migrate`
5. Run server: `python manage.py runserver`
6. Run Celery: `celery -A myinternship worker -l info`
7. Run bot: `python telegram_bot_listener.py`

## API Endpoints
- **Public**: `GET /api/public/`
- **Protected**: `GET /api/protected/` (Token Auth)
- **Register**: `POST /api/register/`
~~~

3 Conclusion

The Django Internship Assignment successfully implements a secure Django project with DRF APIs, Celery for background tasks, a Telegram bot, and proper code management. The project adheres to best practices, including environment variable management, PEP 8 compliance, and comprehensive documentation.