# AUTOMATED REVIEW RATING SYSTEM – BACKEND (DJANGO)

## INTRODUCTION

Automated Review Rating System is an Deep learning (BiLSTM) model to predict the rating based on the review body provided by the user. In this documentation provides an overview of the project, including its architecture, setup, and integration details. It explains the Django project structure, React integration, Deep Learning (DL) model integration, and URL routing.

## API DOCUMENTION

### 1. Post a review

**Endpoint**: http://127.0.0.1:8000/api/reviews/

**Method**: POST

**Body** (raw - JSON):

**Input**:


{

"author":"justin",

"title":"comic publisher",

"body": "in a few hundred words you learn less than you would from wikipedia skip it"

}


**Response**:


{

"status": "success",

"message": "Your review posted successfully!",

"data": {

"id": 9,

"author": "justin",

"title": "comic publisher",

"body": "in a few hundred words you learn less than you would from wikipedia skip it",

"rating": 1,

"added_at": "2025-09-16T09:08:39.643548Z"

```
    }
}
```


## 2. Retrieve all reviews

**Endpoint**: http://127.0.0.1:8000/api/reviews/

**Method**: GET

**Response:**


```
{
    "average": 2,
    "totalRatings": 3,
    "breakdown": [
        {
            "stars": 3,
            "count": 1
        },
        {
            "stars": 2,
            "count": 1
        },
        {
            "stars": 1,
            "count": 1
        }
    ],
    "reviews": [
        {
            "id": 12,
            "author": "justin",
            "title": "about books",
```

"body": "im surprised that this novel won the national book award for its not a big or ambitious book as its predecessor the corrections was one of the characters is a photographer whose metier is extreme closeups of familiar sights a lovers hip a babys fist the book is very much like that an extended close up of several domestic arrangements its an extremely wellwritten elegantly structured miniature the first novella in the series collies deals with paul mcleod a scotsman whose wife has recently died this is the gem of the collection its easy to understand why glasss novella won awards it is spare understated elegant as we learn what pauls life is like in the present without his difficult but beloved maureen we also discover what his married life has been like for decades and we witness his attempts to envision a future on his own a couples entire life together a mans past present and future all in only pages miraculous collies stands on its own as a wonderfully complete read well worth the time invested in it i dont know if glasss original conception of the book were a three part novel or if she decided to extend the opening successful novella but the remainder of the book suffers by comparison there are a number of small clever touches to link all three stories together but the links at times seem forced the most interesting characters in three junes are the mcleods senior paul and maureen older people on the verge of being senior citizens a demographic whose stories are often ignored in contemporary fiction or made the subject of satire here glass treats them with respect and restraint and they are worth getting to know unfortunately they do not appear in the rest of the book other reviewers have admired the long central section the story of fenno mcleod the eldest of three sons but his story is a fairly familiar one however well told the final section of the novel brings some of the earlier characters together and moves along rather briskly but feels much less substantial perhaps the weakest aspect of this book is its unbalanced structure the disproportionately lengthy middle section is actually longer than the other two sections combined by itself it could well be a complete novel so one is left wondering why three junes rather than just one the first one glass is an accomplished writer the book is on the whole less than the sum of its parts but it is worth reading particularly its opening section i remain perplexed however as to what recommended this book to the award panel i cant imagine reading it again or urging others to pick it up",

"rating": 3,

"added_at": "2025-09-16T09:27:41.341503Z"

},

{

"id": 11,

"author": "justin",

"title": "about books",

"body": "i was so absolutely in love with maybe somedaythat i thought i would try another colleen hoover book perhaps it was too soon i ended up disappointed the

characters didnt seem to stay consistent throughout the storyand the writer made a few choices that left me feelingconcerned",

"rating": 2,

"added_at": "2025-09-16T09:27:00.380996Z"

},

{

"id": 10,

"author": "justin",

"title": "comic publisher",

"body": "in a few hundred words you learn less than you would from wikipedia skip it",

"rating": 1,

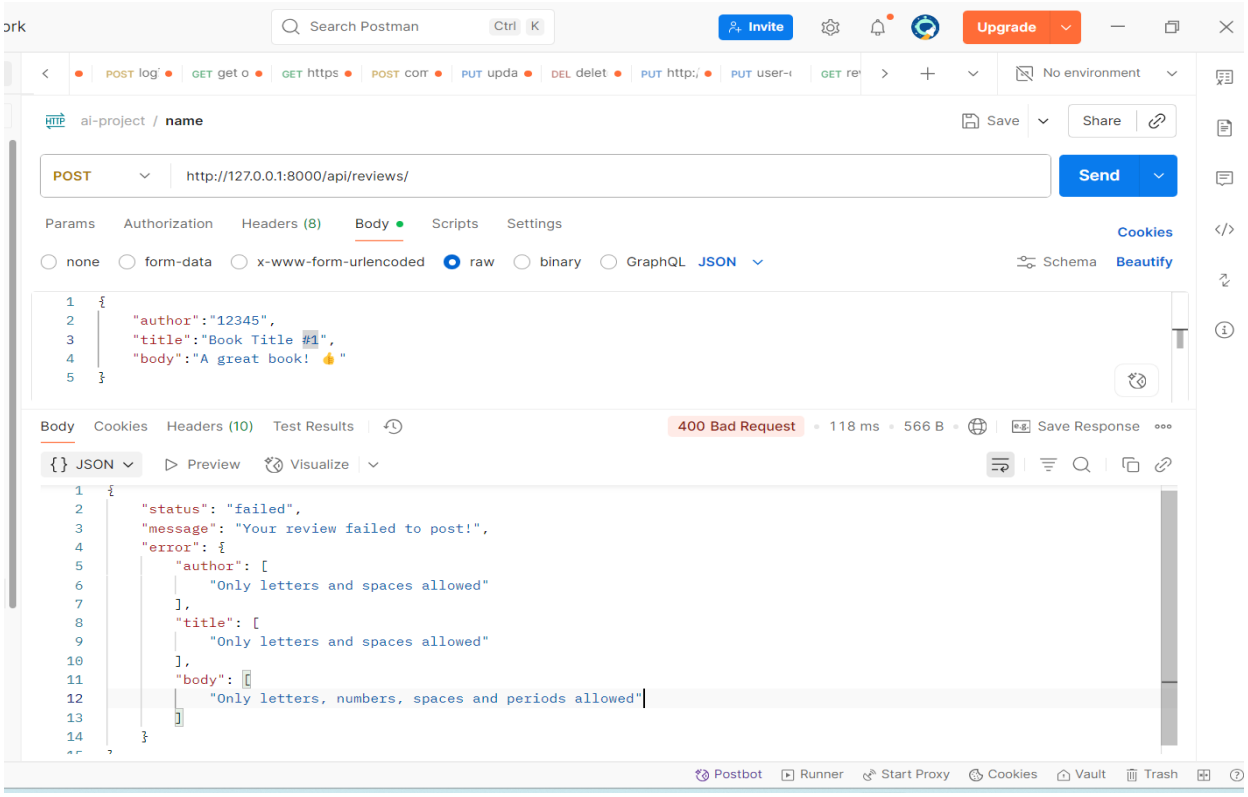
"added_at": "2025-09-16T09:25:59.043394Z"

}

]

}

## 3. Error Validation



**Endpoint**: http://127.0.0.1:8000/api/reviews/

**Method**: POST

**Body** (raw - JSON):

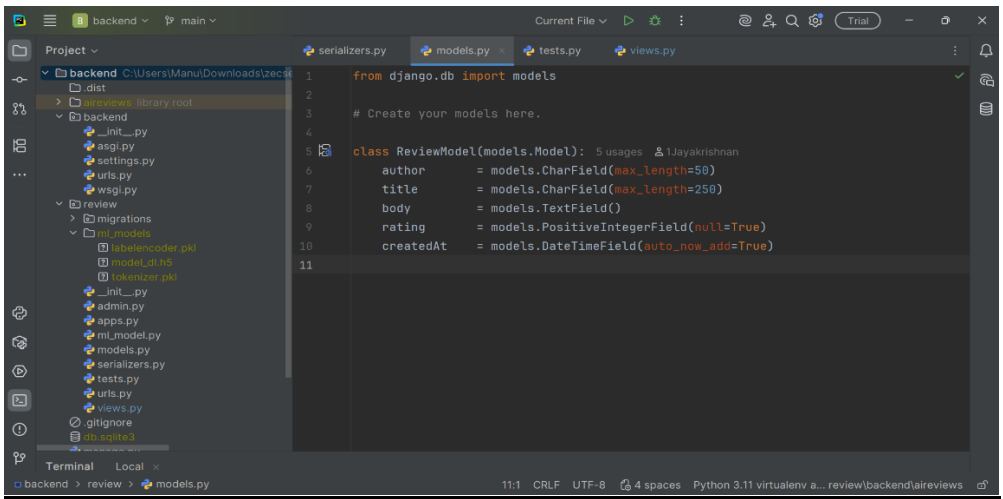
**Input**:


```
{
   "author":"aaaaa",
   "title":"bbbbbba",
   "body":"hhhhhhhhhhhhhah"
}
```


**Response:**


```
{
   "status": "failed",
   "message": "Your review failed to post!",
   "error": {
      "author": [
         "Author looks invalid (repeated characters)."
      ]
   }
}
```


## PROJECT STRUCTURE AND CODE



- **manage.py** – The entry point for running the project. You use it for commands like runserver, migrate, and creating apps.

**backend/**

The main Django project folder that contains global configuration.

- **urls.py -** The root URL router that directs requests to the correct app.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
   path("admin/", admin.site.urls),  # django admin panel
   path('api/', include('review.urls'))  # review app API endpoints
]
```

- **settings.**py - Holds project settings (installed apps, middleware, database, static files, DL paths, etc.).

- **review/**
  - **models.py -** Defines the database schema for storing user reviews and ratings.
  - **serializers.py -** Converts Django models to JSON and validates input data for API requests.
  - **views.py** – Implements API endpoints for submitting reviews, predicting ratings, and fetching stored reviews.
  - **urls.py** – Contains the app-level URL configurations, mapping API endpoints to views.
  - **ml_model.py** – Handles loading of the trained ML model, tokenizer, and label encoder. It is responsible for text preprocessing and generating predictions.
  - **admin.py** – Registers models in the Django Admin dashboard.
  - **tests.py** – Contains unit tests for validating the app's functionality.
- **review/ml_models (Machine Learning Files)**
  - **model_dl.h5** – The trained sentiment analysis model.
  - **tokenizer.pkl** – Tokenizer for text preprocessing.
  - **labelencoder.pkl** – Encoder for mapping predictions to labels.
- **requirements.txt**
  A text file containing the list of required dependencies such as Django, Django REST Framework and TensorFlow.

## MAIN CODES

### settings.py (Database configuration)

```
DATABASES = {
   'default': {
```

```
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

**settings.py (App configuration)**

```
INSTALLED_APPS = [
    ...
    "review",
    “rest_framework”
   …
   ]
```

## models.py

```
from django.db import models

# Create your models here.

class ReviewModel(models.Model):
    author      = models.CharField(max_length=50)
    title       = models.CharField(max_length=250)
    body        = models.TextField()
    rating      = models.PositiveIntegerField(null=True)
    createdAt   = models.DateTimeField(auto_now_add=True)
```

## serializers.py

```
from rest_framework import serializers
from .ml_model import predict_rating
from . models import ReviewModel

class ReviewModelSerialization(serializers.ModelSerializer):
    class Meta:
        model = ReviewModel
        fields = "__all__"
        read_only_fields = ["rating", "createdAt"]

def validate_author(self, value):
    value = value.strip()
    if not value:
```

```
            raise serializers.ValidationError("Author cannot be empty")
        if len(value) < 2:
            raise serializers.ValidationError("Characters should contain at least 2")
        if len(value) > 50:
            raise serializers.ValidationError("Characters should not exceed 50")
        if not re.fullmatch(r"[A-Za-z ]+", value):
            raise serializers.ValidationError("Only letters and spaces allowed")
        if not re.search(r"[A-Za-z]", value):
            raise serializers.ValidationError("Must have at least one letter")
        if not re.search(r"[aeiouAEIOU]", value):
            raise serializers.ValidationError("Must have at least one English vowel")
        if re.fullmatch(r"^(.)\1+$", value):
            raise serializers.ValidationError("Author looks invalid (repeated characters).")
        return value

    def validate_title(self, value):
        value = value.strip()
        if not value:
            raise serializers.ValidationError("Title can’t be empty")
        if len(value) < 5 or len(value) > 100:
            raise serializers.ValidationError("Title must be 5–100 characters")
        if not re.fullmatch(r"[A-Za-z ]+", value):
            raise serializers.ValidationError("Only letters and spaces allowed")
        if not re.search(r"[A-Za-z]", value):
            raise serializers.ValidationError("Must have at least one letter")
        if not re.search(r"[aeiouAEIOU]", value):
            raise serializers.ValidationError("Must have at least one English vowel")
        if re.fullmatch(r"^(.)\1+$", value):
           raise serializers.ValidationError("Title looks invalid (repeated characters).")
        if value.lower() in ["test", "abcde"]:
            raise serializers.ValidationError("Title too generic/placeholder")
        return value

    def validate_body(self, value):
        value = value.strip()
        if len(value) < 10:
            raise serializers.ValidationError("Body must be at least 10 characters")
        if not re.search(r"[aeiouAEIOU]", value):
            raise serializers.ValidationError("Must have at least one English vowel")
        if not re.fullmatch(r"[A-Za-z0-9 .]+", value):
            raise serializers.ValidationError("Only letters, numbers, spaces and periods
allowed")
        if re.fullmatch(r"^(.)\1+$", value):
```

```
            raise serializers.ValidationError("Body looks invalid (repeated characters).")
        if value.lower() in ["abcdef", "12345"]:
            raise serializers.ValidationError("Body is placeholder/sequential text")
        return value

    def create(self, validated_data):
        # Predict rating if not provided
        if not validated_data.get("rating"):
            review_text = validated_data.get("body", "") or ""
            validated_data["rating"] = predict_rating(review_text)
        return super().create(validated_data)
```

**views.py**

```
from .serializers import ReviewModelSerialization
from .models import ReviewModel
from rest_framework import status
from rest_framework.response import Response
from rest_framework.generics import GenericAPIView
from django.db.models import Avg, Count

class ReviewAPIView(GenericAPIView):

    def get(self, request):
        # sorting the reviews
        sort_option = request.query_params.get("sort", "newest")
        if sort_option == "oldest":
            reviews = ReviewModel.objects.all().order_by('createdAt')
        else:
            reviews = ReviewModel.objects.all().order_by('-createdAt')

        serialized_reviews = ReviewModelSerialization(reviews, many=True).data

        # Calculate average rating
        average = reviews.aggregate(avg=Avg('rating'))['avg'] or 0
        # average = float(f"{average:.1f}")
        average = int(average)

        # Total number of ratings
        total_ratings = reviews.count()

        # Breakdown by rating
        breakdown_queryset = (
```

```
            reviews.values('rating')
            .annotate(count=Count('id'))
            .order_by('-rating')
        )

        rating_counts = {item['rating']: item['count'] for item in breakdown_queryset}
        breakdown = [
        {"stars":stars, "count":rating_counts.get(stars, 0)}
        for stars in range(5, 0, -1)
        ]

        return Response({
            "average": average,
            "totalRatings": total_ratings,
            "breakdown": breakdown,
            "reviews": serialized_reviews
        }, status=status.HTTP_200_OK)

    def post(self, request):
        serializer = ReviewModelSerialization(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response({
                "status": "success",
                "message": "Your review posted successfully!",
                "data": serializer.data
            }, status=status.HTTP_201_CREATED)

        return Response({
            "status": "failed",
            "message": "Your review failed to post!",
            "error": serializer.errors
        }, status=status.HTTP_400_BAD_REQUEST)
```

**urls.py**

```
from django.urls import path
from . import views

urlpatterns = [
    # path for fetch all reviews and post review
    path('reviews/', views.ReviewAPIView.as_view(), name='review')
]
```

## DJANGO-ML MODEL INTEGRATION

Storing the ML Models into a directory called **ml_models** inside a Django app (review)

- **model.h5**
- **tokenizer.pkl**
- **labelencoder.pkl**

Created a python file called **ml_model.py** inside a Django app (review)

## ml_model.py

```
import os, pickle
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load model and preprocessing files
BASE_DIR = os.path.dirname(__file__)
MODEL_PATH = os.path.join(BASE_DIR, "../ml_models/model.h5")
TOKENIZER_PATH = os.path.join(BASE_DIR, "../ml_models/tokenizer.pkl")
ENCODER_PATH = os.path.join(BASE_DIR, "../ml_models/label_encoder.pkl")

model = load_model(MODEL_PATH)
tokenizer = pickle.load(open(TOKENIZER_PATH, "rb"))
label_encoder = pickle.load(open(ENCODER_PATH, "rb"))

MAX_LEN = 150

def predict_review(text):
    seq = tokenizer.texts_to_sequences([text])
    padded = pad_sequences(seq, maxlen=MAX_LEN)
    pred = model.predict(padded)
    label = label_encoder.inverse_transform([np.argmax(pred)])
    return label[0]
```

## DJANGO-REACT INTEGRATION

To enable communication between Django (backend) and React (frontend), configure CORS (Cross-Origin Resource Sharing) in Django.

**Django configuration**

pip install **django-cors-headers**

## settings.py

```
INSTALLED_APPS = [
   ...
   "corsheaders",
  …
  ]

MIDDLEWARE = [
    …
   "corsheaders.middleware.CorsMiddleware",
   ...
  ]

CORS_ALLOWED_ORIGINS = [
   "http://localhost:5173",  # Vite default
]
```

**React configuration**

npm install **axios**

## axios.ts

```
import axios from "axios";

const API_BASE: string =
 "http://127.0.0.1:8000/api/";

const client = axios.create({
  baseURL: API_BASE,
  withCredentials: true,
});

export default client;
```

## GITHUB REPOSITORY

The complete backend source code for **Automated Review Rating System** is hosted on GitHub. It contains the Django project files, the reviews app with API endpoints, and the trained Deep Learning model for predicting ratings.

**Repository Link:** https://github.com/1Jayakrishnan/automated-review-system