Movie Review API

by Hashim Aziz Muhammad

Development Phases

1. Setup Django project and configure Django REST Framework
2. Design and implement database models
3. Implement user authentication system using `rest_framework_simplejwt`
4. Integrate external movie API (OMDb)
5. Develop core API functionality (CRUD operations for movies and reviews)
6. Implement additional features (rating system, user profiles)
7. Thorough testing and debugging
8. Final Feature Fixing.
9. Deployment to PythonAnywhere
10. Documentation and final testing

Technologies Used:

- **Backend Framework with Views:** Django with Django REST Framework
- **Database:** SQLite
- **External API:** OMDb API for movie data
- **Authentication:** Django REST Framework Simple JWT (JSON Web Tokens)
- **Deployment:** PythonAnywhere

Key Features

1. **User Authentication and Registration:** Implemented a secure registration and login system using Django REST Framework built-in authentication and Simple JWT for token-based authentication.

2. **Movie Search:** Integrated the OMDb API to allow users to search for movies and retrieve detailed information.

3. **Review System:** Developed CRUD operations for users to create, read, update, and delete movie reviews.

4. **Rating System:** Implemented a star-based rating system for movies.

5. **API Endpoints:** Designed and implemented RESTful API endpoints for all major functionalities.

API Endpoints

I have implemented four different types of services within the API endpoints.

Movies: Invokes OMDB external API

Reviews: API endpoints for Reviews

Users: API endpoints for users

Authentication: API endpoints for Authentication

Recommendations: Invokes OMDB external API

- Movies
  1. search_movie view which queries the OMDB external API.
  2. /search/ (GET): list movies retrieved from OMDB
- Reviews
  1. /reviews/ (GET, POST): List reviews, create a new review.
  2. /reviews/<int:id>/ (GET, PUT, DELETE): Retrieve, update, or delete a specific review.
  3. /reviews/<int:id>/comment/ (POST): Add a comment to a review.
  4. /reviews/<int:id>/like/ (POST): Like a review.
- Users
  1. /api/user-profile/ (GET, POST): Register a new user.
  2. /api/user-profile/edit(GET, PUT): Retrieve or update a user profile.
- Authentication
  1. /api/token/ (POST): Obtain JWT token for authentication.
  2. /api/token/refresh/ (POST): Refresh JWT token.
  3. /login/ : User login endpoint using Django built-in.
  4. /logout/ : User logout endpoint using Django built in.
  5. /register/: User registration endpoint.
- Recommendations
  1. /recommendations/ (GET): Get movie recommendations. Using external OMDB API

Future Improvements

1. Implement more advanced filtering and search capabilities for movies and reviews.
2. Add social features, such as following other users and sharing reviews.
3. Enhance the user interface with a React-based frontend to create a full-stack application.
4. Implement more comprehensive test coverage, including integration and end-to-end tests.
5. Optimize database queries for improved performance with large datasets

Challenges Faced and Solutions

- **External API Integration**: Handled API rate limits & ensured data consistency
  - *Solution*: Error handling and reliable API calls
- **JWT Authentication**: Implemented secure token-based authentication
  - *Solution*: Simple JWT for token generation and validation
- **Deployment**: Overcame initial issues with deploying to PythonAnywhere
  - *Solution*: Followed best practices for deployment

Reflection and Key Learnings

API Design: Gained experience in designing RESTful APIs

Authentication: Deepened understanding of JWT and Django REST Framework security

External API Management: Learned effective ways to handle third-party API dependencies

Deployment: Developed skills in deploying Django applications in a production environment

Personal Growth: Built a project that aligns with my passion for cinema and technology

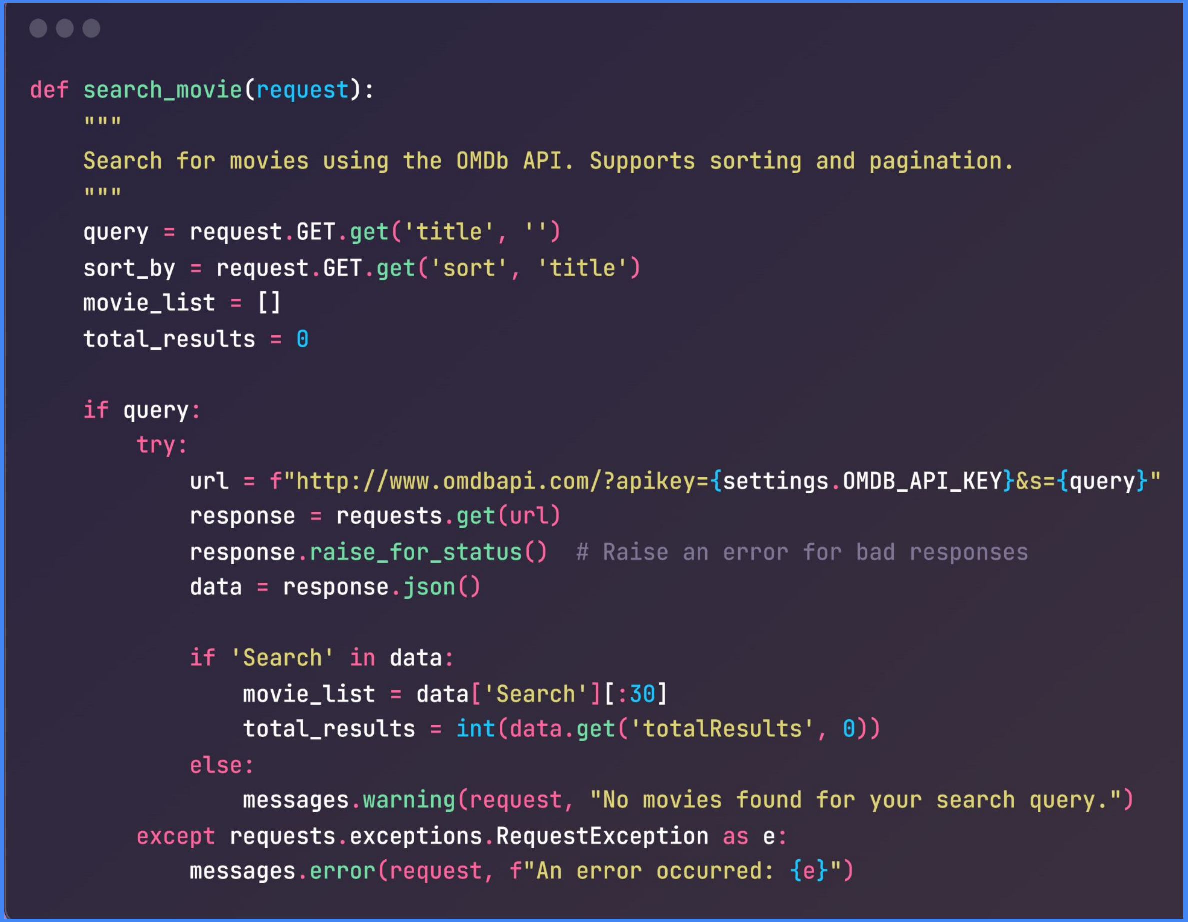
Key Code Fragments

There as 6 key code fragments from my project which i would like to present:

1. External API Call for movie search with Error Handling
2. Ensuring Proper Validation When Creating a New Review on Both the Frontend and Backend

Code Fragment 1

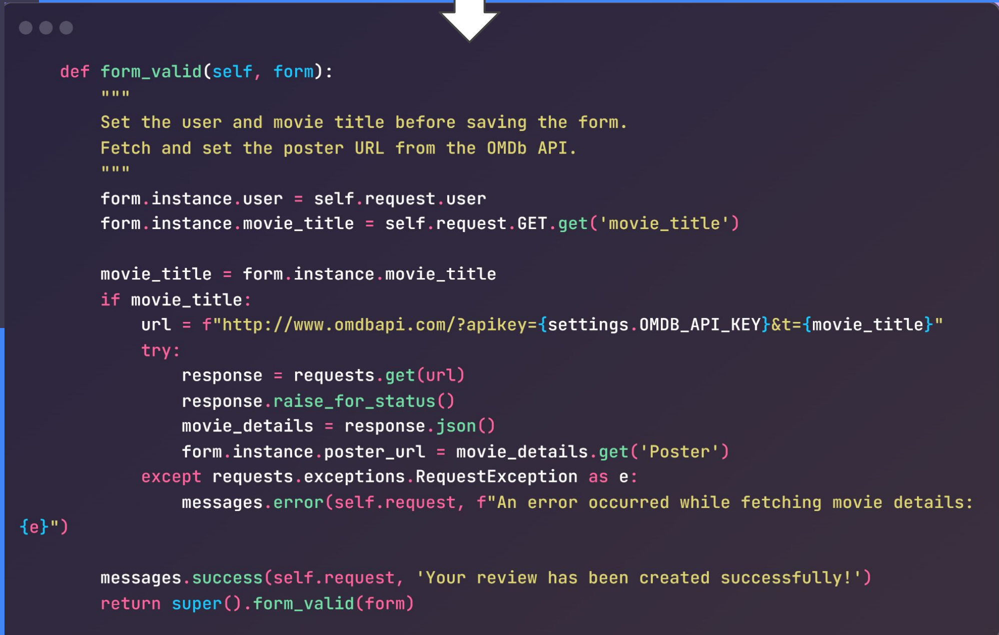
External API Call to OMDB for movie search with Error Handling.



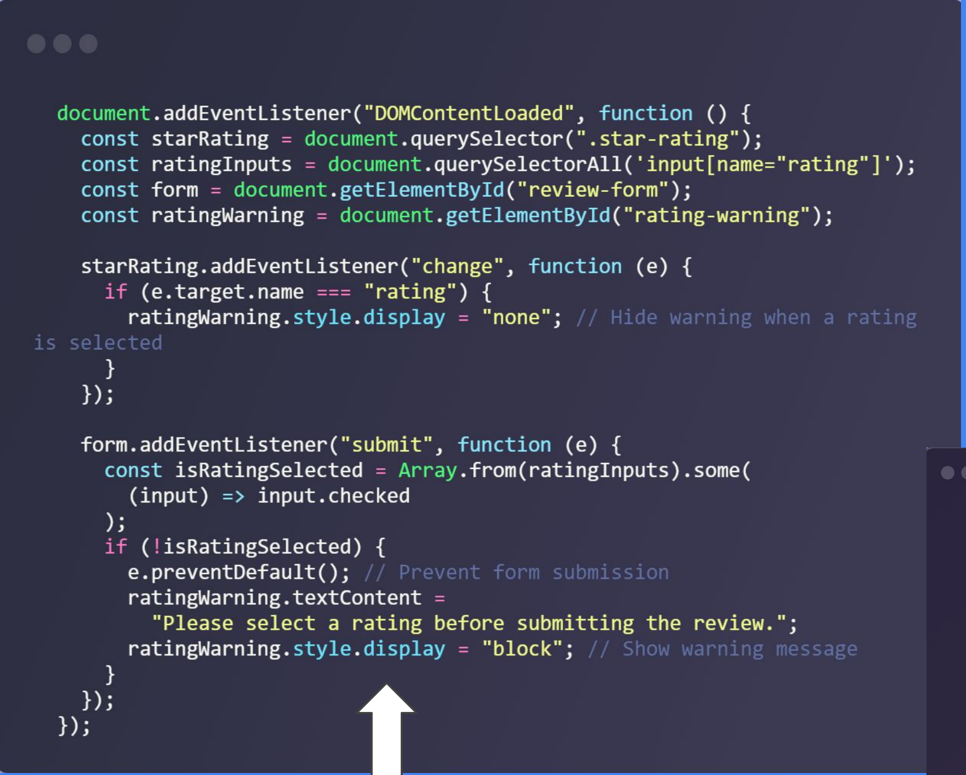
```python
def search_movie(request):
    """
    Search for movies using the OMDb API. Supports sorting and pagination.
    """
    query = request.GET.get('title', '')
    sort_by = request.GET.get('sort', 'title')
    movie_list = []
    total_results = 0

    if query:
        try:
            url = f"http://www.omdbapi.com/?apikey={settings.OMDB_API_KEY}&s={query}"
            response = requests.get(url)
            response.raise_for_status()  # Raise an error for bad responses
            data = response.json()

            if 'Search' in data:
                movie_list = data['Search'][:30]
                total_results = int(data.get('totalResults', 0))
            else:
                messages.warning(request, "No movies found for your search query.")
        except requests.exceptions.RequestException as e:
            messages.error(request, f"An error occurred: {e}")
```

Code Fragment 2

Validation When creating a new review to see if the review is valid and the movie details are correct from the external API call.



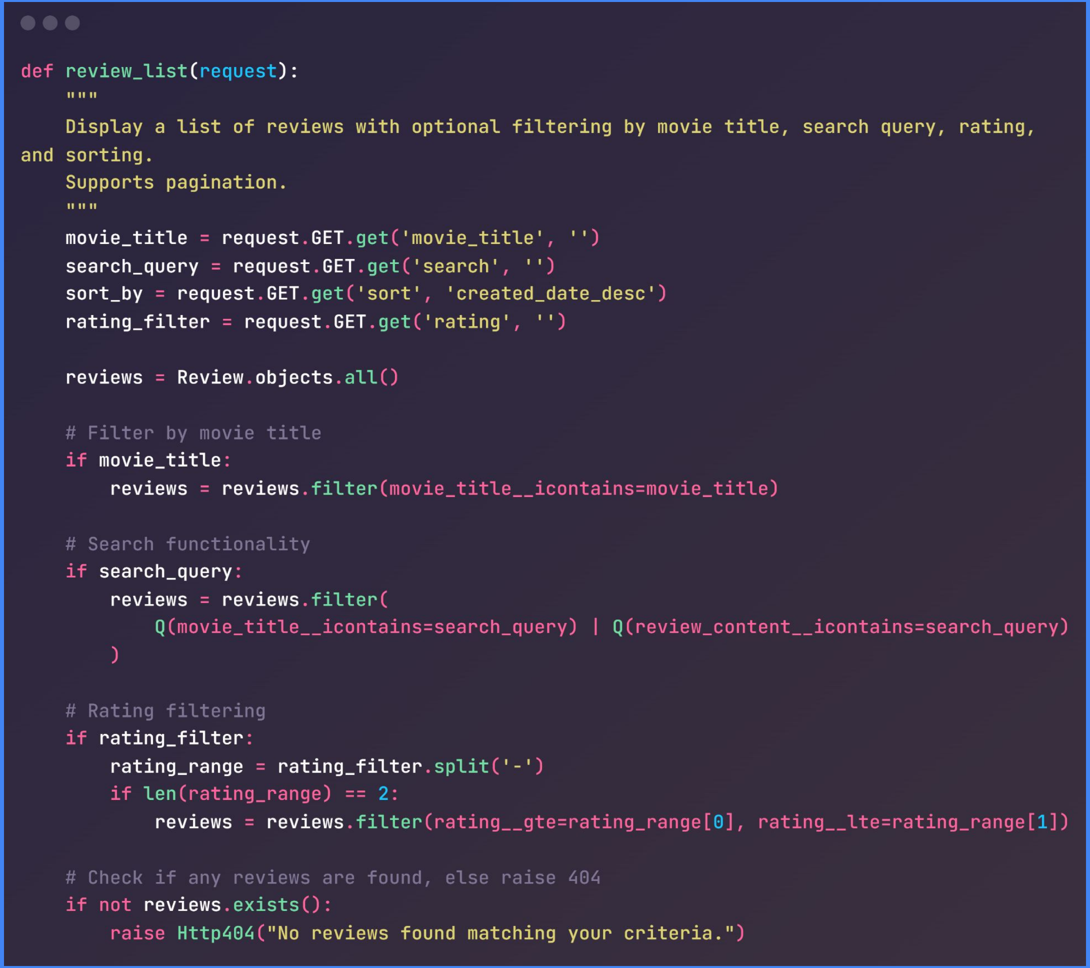
```python
def form_valid(self, form):
    """
    Set the user and movie title before saving the form.
    Fetch and set the poster URL from the OMDb API.
    """
    form.instance.user = self.request.user
    form.instance.movie_title = self.request.GET.get('movie_title')

    movie_title = form.instance.movie_title
    if movie_title:
        url = f"http://www.omdbapi.com/?apikey={settings.OMDB_API_KEY}&t={movie_title}"
        try:
            response = requests.get(url)
            response.raise_for_status()
            movie_details = response.json()
            form.instance.poster_url = movie_details.get('Poster')
        except requests.exceptions.RequestException as e:
            messages.error(self.request, f"An error occurred while fetching movie details:
{e}")

    messages.success(self.request, 'Your review has been created successfully!')
    return super().form_valid(form)
```



```javascript
  document.addEventListener("DOMContentLoaded", function () {
    const starRating = document.querySelector(".star-rating");
    const ratingInputs = document.querySelectorAll('input[name="rating"]');
    const form = document.getElementById("review-form");
    const ratingWarning = document.getElementById("rating-warning");

    starRating.addEventListener("change", function (e) {
      if (e.target.name === "rating") {
        ratingWarning.style.display = "none"; // Hide warning when a rating
is selected
      }
    });

    form.addEventListener("submit", function (e) {
      const isRatingSelected = Array.from(ratingInputs).some(
        (input) => input.checked
      );
      if (!isRatingSelected) {
        e.preventDefault(); // Prevent form submission
        ratingWarning.textContent =
          "Please select a rating before submitting the review.";
        ratingWarning.style.display = "block"; // Show warning message
      }
    });
  });
```

Code Fragment 3

Validation when creating a new review in the javascript front-end if the user hasn’t selected a rating



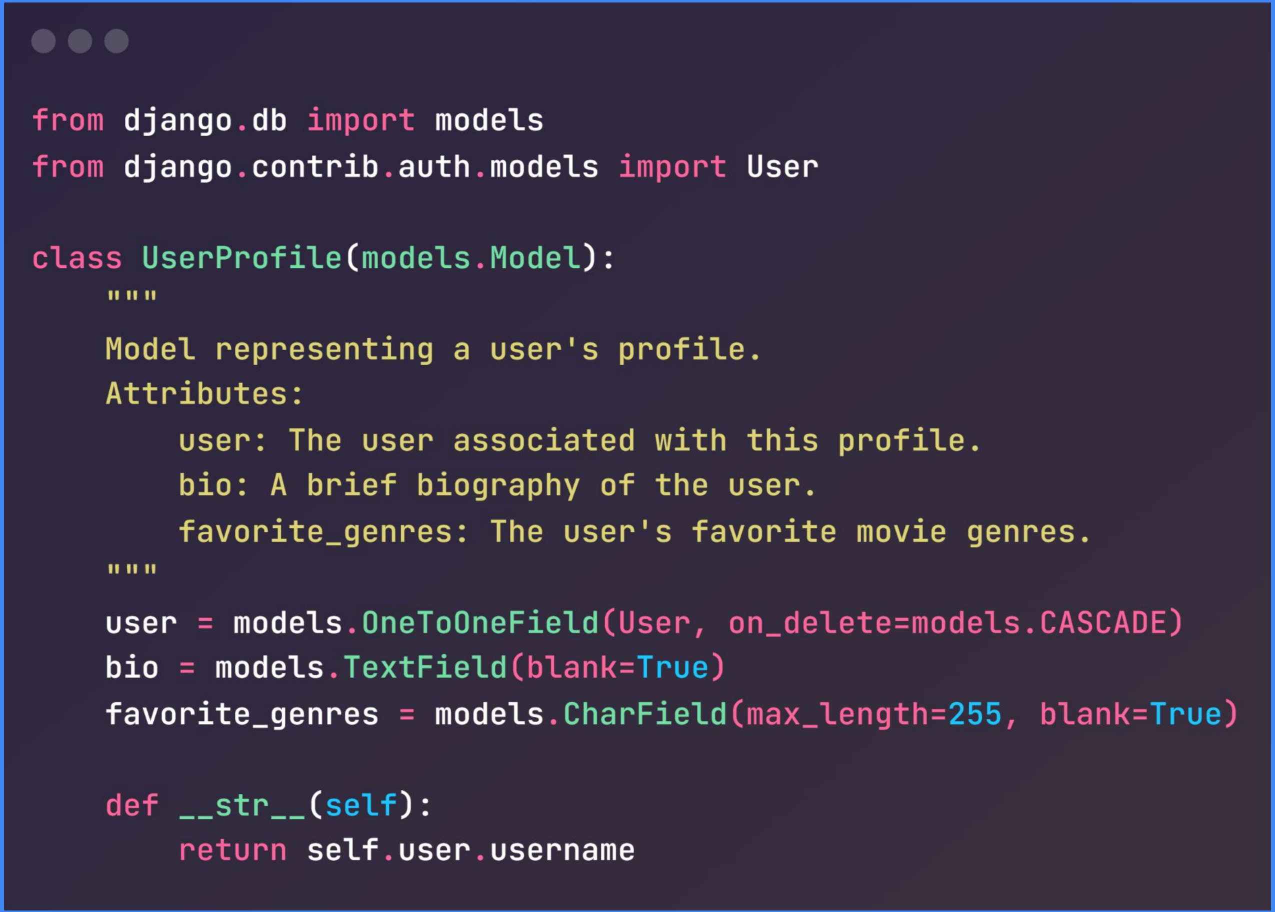
```python
def review_list(request):
    """
    Display a list of reviews with optional filtering by movie title, search query, rating,
and sorting.
    Supports pagination.
    """
    movie_title = request.GET.get('movie_title', '')
    search_query = request.GET.get('search', '')
    sort_by = request.GET.get('sort', 'created_date_desc')
    rating_filter = request.GET.get('rating', '')

    reviews = Review.objects.all()

    # Filter by movie title
    if movie_title:
        reviews = reviews.filter(movie_title__icontains=movie_title)

    # Search functionality
    if search_query:
        reviews = reviews.filter(
            Q(movie_title__icontains=search_query) | Q(review_content__icontains=search_query)
        )

    # Rating filtering
    if rating_filter:
        rating_range = rating_filter.split('-')
        if len(rating_range) == 2:
            reviews = reviews.filter(rating__gte=rating_range[0], rating__lte=rating_range[1])

    # Check if any reviews are found, else raise 404
    if not reviews.exists():
        raise Http404("No reviews found matching your criteria.")
```

Code Fragment 4

Database Query with 404 Handling for No Results

Code Fragment 5

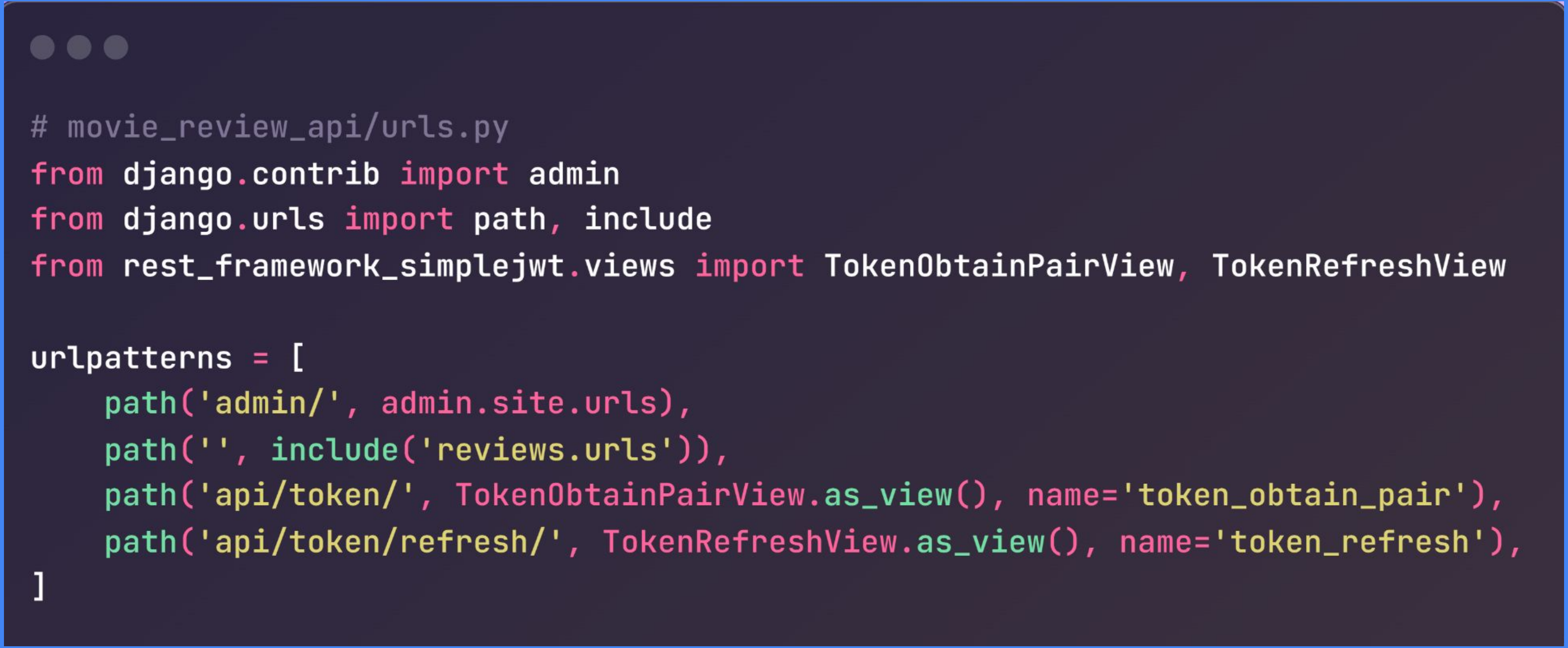
UserProfile Model extending Django User Model displaying OOP principles.



```python
from django.db import models
from django.contrib.auth.models import User

class UserProfile(models.Model):
    """
    Model representing a user's profile.
    Attributes:
        user: The user associated with this profile.
        bio: A brief biography of the user.
        favorite_genres: The user's favorite movie genres.
    """
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    bio = models.TextField(blank=True)
    favorite_genres = models.CharField(max_length=255, blank=True)

    def __str__(self):
        return self.user.username
```

Code Fragment 6

Django REST Framework: Simple JWT Implementation for Authentication



```python
# movie_review_api/urls.py
from django.contrib import admin
from django.urls import path, include
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('reviews.urls')),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
]
```

Thanks!

hashimazizm@gmail.com

View the Deployed site:
Movie Review API