



Moviefy

İzleme Alışkanlıklarınızdan İlham Alan Film Önerileri

Hazırlayanlar

Kadir Öner

200542012

Tuğçe Özelmaci

200541006

Pelin Altürk

200542014

Projeye Giriş

Moviefy, kullanıcıların izleme alışkanlıklarını analiz eden ve buna göre film önerileri sunan bir Python tabanlı uygulamadır. Kullanıcıların daha çok hangi filmlere tıkladıysa buna uygun film önerileri sunan bir uygulamadır. Bu sayede kullanıcılar, ilgi alanlarına uygun filmleri daha kolay keşfederler.

Problem

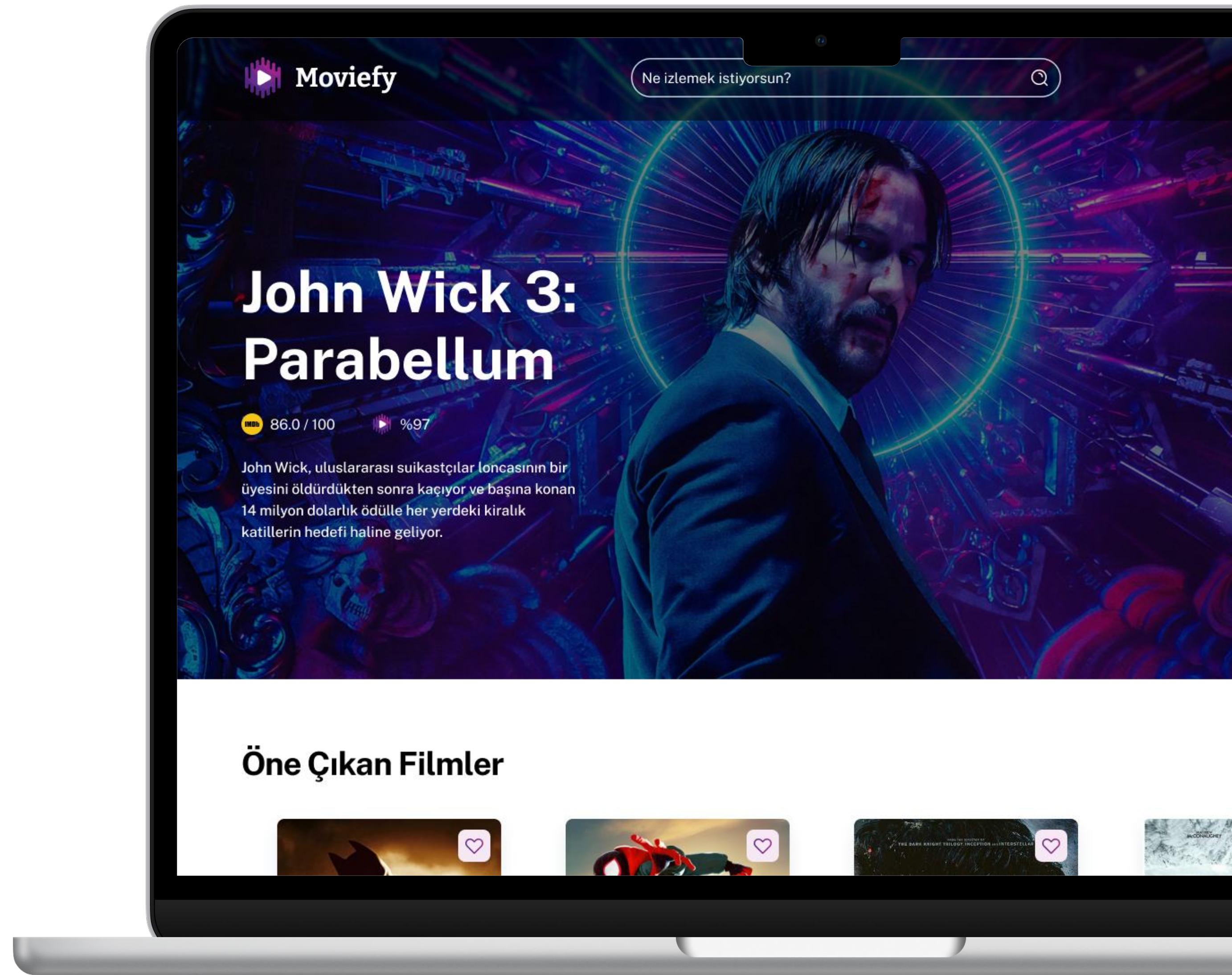
Günümüzde birçok kullanıcı, izlemek istedikleri filmleri bulmakta zorlanmaktadır ve geniş film yelpazesi arasında kaybolmaktadır. Farklı platformlarda binlerce film ve dizi bulunmasına rağmen, kullanıcılar genellikle ilgi alanlarına uygun içerikleri keşfetmekte güçlük çekiyor. Bu durum, zaman kaybına ve izleme deneyimlerinin olumsuz etkilenmesine neden oluyor.

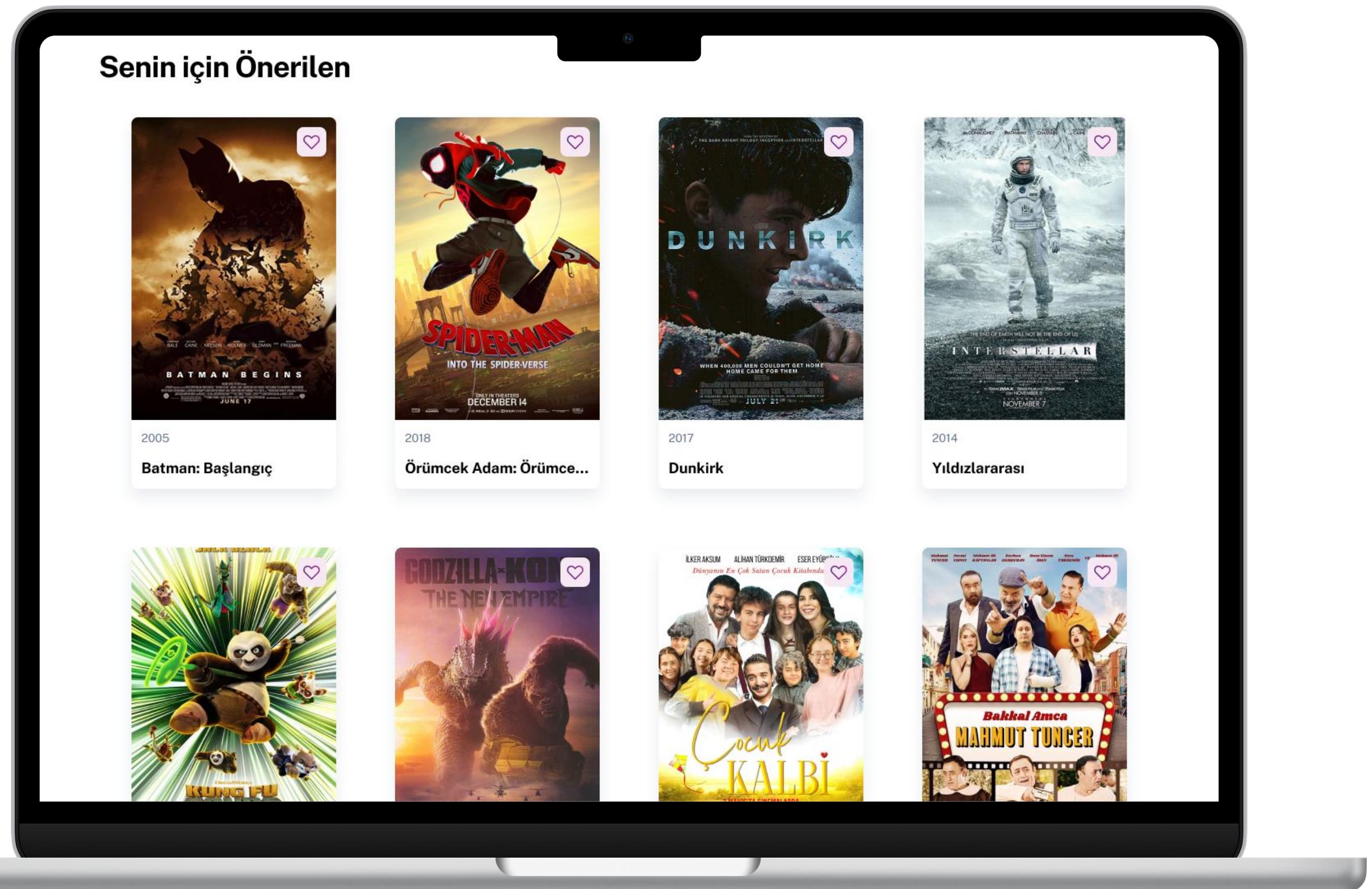
Çözüm

Moviefy, kullanıcıların izleme alışkanlıklarını analiz ederek kişiselleştirilmiş film önerileri sunar. Kullanıcıların geçmişte hangi filmlere tıkladığını ve izlediğini analiz ederek, benzer türde ve temada filmleri önerir. Böylece kullanıcılar, kendilerine hitap eden içeriklere daha hızlı ve kolay bir şekilde ulaşırken, izleme deneyimlerini en üst düzeye çıkarırlar. Moviefy, kullanıcılarla zaman kazandırır ve film seçiminin keyifli hale getirir.

Projenin Amacı

Moviefy uygulamasının temel amacı, kullanıcıların izleme alışkanlıklarını analiz ederek kişiselleştirilmiş film önerileri sunmaktır. Günümüzde birçok kullanıcı, izlemek istedikleri filmleri bulmakta zorlanmaktadır ve geniş film yelpazesi arasında kaybolmaktadır. Moviefy, bu sorunu çözmeyi hedefleyerek, kullanıcıların ilgi alanlarına uygun filmleri daha hızlı ve kolay bir şekilde keşfetmelerini sağlar. Uygulama, kullanıcının geçmişte hangi filmlere tıkladığını ve izlediğini analiz ederek, benzer türde ve temada filmleri önerir. Böylece kullanıcılar, kendilerine hitap eden içeriklere ulaşırken zaman kaybetmez ve izleme deneyimlerini en üst düzeye çıkarırlar.





Kullanılan Yöntemler

- Pandas ile Veri Görselleştirme
- NumPy ile Veri Analizi ve Hesaplama
- Torch ile Derin Öğrenme Modelleri
- Flask ile Web Uygulama Geliştirme
- Scikit-learn ile Makine Öğrenimi Modelleme
- Neo4j ile Grafik Veritabanı Kullanımı
- MongoDB ile Büyük Veri Yönetimi
- Raporlama ve Sonuçların Görselleştirmesi

Sistem Mimarisi

Kütüphane ve Konfigrasyon

Bu bölüm, gerekli kütüphaneleri içe aktarıyor ve Flask uygulamasını konfigüre ediyor. MongoDB ve Neo4j bağlantı ayarları yapılmış ve kullanıcılara, filmlere ve derecelendirmelere yönelik Neo4j fonksiyonları tanımlanmıştır.

```
main.py

from flask import Flask, request, redirect, url_for, render_template, session
from flask_pymongo import PyMongo
from werkzeug.security import generate_password_hash, check_password_hash
import pandas as pd
import numpy as np
import torch
import random
from sklearn.metrics.pairwise import cosine_similarity
from bson.objectid import ObjectId
from neo4j import GraphDatabase # Neo4j bağlantısı için ekleme

app = Flask(__name__)
app.secret_key = 'moviefy_secret_key'

app.config["MONGO_URI"] = "mongodb://localhost:27017/film_recommendation"
mongo = PyMongo(app)

neo4j_url = "bolt://localhost:7687"
neo4j_username = "neo4j"
neo4j_password = "password"

driver = GraphDatabase.driver(neo4j_url, auth=(neo4j_username, neo4j_password))

def add_user(tx, user_id, username):
    tx.run("CREATE (:User {user_id: $user_id, username: $username})",
          user_id=user_id, username=username)

def add_movie(tx, movie_id, title):
    tx.run("CREATE (:Movie {movie_id: $movie_id, title: $title})",
          movie_id=movie_id, title=title)

def add_rating(tx, user_id, movie_id, rating):
    tx.run("""
        MATCH (u:User {user_id: $user_id})
        MATCH (m:Movie {movie_id: $movie_id})
        CREATE (u)-[:RATED {rating: $rating}]->(m)
    """, user_id=user_id, movie_id=movie_id, rating=rating)
```

Sistem Mimarisi

Veri Yükleme ve Model Eğitimi

Bu bölümde, veri setleri yükleniyor ve pandas DataFrame'lerine dönüştürülüyor. Film ve kullanıcı verileri işleniyor, ardından bir Matrix Factorization modeli tanımlanıyor ve eğitim verileriyle eğitiliyor. Eğitim tamamlandıktan sonra, film faktörleri arasındaki benzerlik matrisi hesaplanıyor.

```
main.py

df_rating = pd.read_csv("./dataset/Netflix_Dataset_Rating.csv")
df_movies = pd.read_csv("./dataset/Enriched_Movie_data.csv")

movies_data = df_movies.set_index('Movie_ID').to_dict(orient='index')
movie_ids = list(movies_data.keys())
movie_id_to_index = {movie_id: idx for idx, movie_id in enumerate(movie_ids)}

n_users = df_rating['User_ID'].nunique()
n_movies = len(movie_ids)

max_user_id = df_rating['User_ID'].max()
max_movie_id = df_rating['Movie_ID'].max()

n_users = max(n_users, max_user_id + 1)
n_movies = max(n_movies, max_movie_id + 1)

class MatrixFactorization(torch.nn.Module):
    def __init__(self, n_users, n_movies, n_factors=20):
        super().__init__()
        self.user_factors = torch.nn.Embedding(n_users, n_factors)
        self.movie_factors = torch.nn.Embedding(n_movies, n_factors)
        self.user_factors.weight.data.uniform_(0, 0.05)
        self.movie_factors.weight.data.uniform_(0, 0.05)

    def forward(self, user, movie):
        return (self.user_factors(user) * self.movie_factors(movie)).sum(1)

    def predict(self, user, movie):
        return self.forward(user, movie)

model = MatrixFactorization(n_users, n_movies, n_factors=20)

user_ids = torch.tensor(df_rating['User_ID'].values)
movie_indices = torch.tensor([movie_id_to_index[movie_id] for movie_id in df_rating['Movie_ID'].values])
ratings = torch.tensor(df_rating['Rating'].values, dtype=torch.float32)

loss_fn = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

n_epochs = 13
for epoch in range(n_epochs):
    model.train()
    optimizer.zero_grad()
    predictions = model(user_ids, movie_indices)
    loss = loss_fn(predictions, ratings)
    loss.backward()
    optimizer.step()
    print(f"Epoch {epoch + 1}/{n_epochs}, Loss: {loss.item()}")

model.eval()
trained_movie_factors = model.movie_factors.weight.data.cpu().numpy()
similarity_matrix = cosine_similarity(trained_movie_factors)
```

Sistem Mimarisi

Veritabanı ve Flask Uygulaması

Bu bölüm, Neo4j grafının oluşturulması ve kullanıcıların ana sayfada gösterilmesi ile ilgili fonksiyonları içerir. `create_graph()` fonksiyonu veritabanına kullanıcıları ve filmleri ekler. `home()` fonksiyonu, kullanıcının kaydedilmiş filmlerine dayalı olarak önerilen filmleri getirir ve ana sayfada görüntüler.

```
main.py

def create_graph():
    with driver.session() as session:
        for user in df_rating['User_ID'].unique():
            session.execute_write(add_user, user, f"user_{user}")

        for movie_id in df_movies['Movie_ID']:
            title = df_movies.loc[df_movies['Movie_ID'] == movie_id, 'Name'].values[0]
            session.execute_write(add_movie, movie_id, title)

        for _, row in df_rating.iterrows():
            session.execute_write(add_rating, row['User_ID'], row['Movie_ID'],
row['Rating'])

@app.route('/')
def home():
    user_id = session.get('user_id')
    username = None
    if user_id:
        user_data = mongo.db.users.find_one({'_id': ObjectId(user_id)})
        username = user_data['username'] if user_data else None

    if user_id:
        user_track_data = mongo.db.user_track.find_one({'user_id': user_id})
        if user_track_data and 'movie_ids' in user_track_data:
            saved_movie_ids = user_track_data['movie_ids']
            recommended_movies = get_similar_movies(saved_movie_ids)
        else:
            recommended_movies = random.sample(movie_ids, min(len(movie_ids), 15))
    else:
        recommended_movies = random.sample(movie_ids, min(len(movie_ids), 15))

    movies = [ {'Movie_ID': movie_id, **movies_data[movie_id]} for movie_id in recommended_movies if movies_data[movie_id]['Poster_Path']]
    return render_template('home.html', movies=movies, username=username)

def get_similar_movies(saved_movie_ids, top_n=15):
    similar_movies = set()
    for movie_id in saved_movie_ids:
        movie_idx = movie_id_to_index[movie_id]
        similar_indices = np.argsort(similarity_matrix[movie_idx])[:-1][1:top_n+1]
        similar_movies.update([movie_ids[idx] for idx in similar_indices])
    similar_movies.difference_update(saved_movie_ids)
    return list(similar_movies)[:top_n]
```

Sistem Mimarisi

Kullanıcı İşlemleri ve Uygulama Çalıştırma

Bu bölüm, kullanıcı kayıt, giriş, çıkış ve film detaylarının gösterilmesi ile ilgili rotaları içerir. Ayrıca, filmleri kaydetme ve kaydedilmiş filmleri gösterme işlemlerini yapar. Flask uygulaması başlatıldığında, veritabanı bağlantıları oluşturulur ve uygulama çalıştırılır.

Bu açıklamalarla birlikte kodu dört parça halinde slaytlarına ekleyebilirsin. Her bir parçada kodun ne yaptığını açıklayan notlar eklemek, dinleyicilerin anlamasını kolaylaşdıracaktır.

```
main.py

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        existing_user = mongo.db.users.find_one({'username': username})
        if existing_user is None:
            hashpass = generate_password_hash(password, method='pbkdf2:sha256')
            result = mongo.db.users.insert_one({'username': username, 'password': hashpass})
            user_id = result.inserted_id

            with driver.session() as session:
                session.execute_write(add_user, str(user_id), username)

            return redirect(url_for('login'))
            return 'User already exists!'
        return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        login_user = mongo.db.users.find_one({'username': username})
        if login_user and check_password_hash(login_user['password'], password):
            session['user_id'] = str(login_user['_id'])
            return redirect(url_for('home'))
        return 'Invalid username/password combination'
    return render_template('login.html')

@app.route('/logout')
def logout():
    session.pop('user_id', None)
    return redirect(url_for('home'))

@app.route('/movie/<int:movie_id>')
def movie_detail(movie_id):
    user_id = session.get('user_id')
    username = None
    if user_id:
        user_data = mongo.db.users.find_one({'_id': ObjectId(user_id)})
        username = user_data['username'] if user_data else None

    if movie_id not in movies_data:
        return "Movie ID not found", 404

    movie_data = movies_data[movie_id]
    movie_idx = movie_id_to_index[movie_id]
    similar_indices = np.argsort(similarity_matrix[movie_idx])[::5]
    similar_movies = [{"Movie_ID": movie_ids[idx], **movies_data[movie_idx]} for idx in similar_indices if movies_data[movie_ids[idx]]['Poster_Path']]
    return render_template('movie_detail.html', movie_data=movie_data, similar_movies=similar_movies, username=username)

@app.route('/save-movie', methods=['POST'])
def save_movie():
    movie_id = request.form.get('movie_id')
    user_id = session.get('user_id')
    if user_id and movie_id:
        movie_id = int(movie_id)
        mongo.db.user_track.update_one({'user_id': user_id}, {'$addToSet': {'movie_id': movie_id}}, upsert=True)

        # Neo4j'de kullanıcı ve film arasındaki ilişki
        with driver.session() as session:
            session.execute_write(add_rating, user_id, movie_id, 5)

    return redirect(url_for('saved_movies_for_user'))
    return 'Unauthorized', 401
```

```
main.py

@app.route('/saved-movies')
def saved_movies_for_user():
    user_id = session.get('user_id')
    username = None
    if user_id:
        user_data = mongo.db.users.find_one({'_id': ObjectId(user_id)})
        username = user_data['username'] if user_data else None

    if not user_id:
        return redirect(url_for('login'))

    user_track_data = mongo.db.user_track.find_one({'user_id': user_id})
    if not user_track_data or 'movie_ids' not in user_track_data:
        return 'No saved movies found.'

    saved_movie_ids = user_track_data['movie_ids']
    saved_movies = [{"Movie_ID": movie_id, **movies_data[movie_id]} for movie_id in saved_movie_ids if movies_data[movie_id]['Poster_Path']]
    return render_template('saved_movies.html', saved_movies=saved_movies, username=username)

from threading import Thread

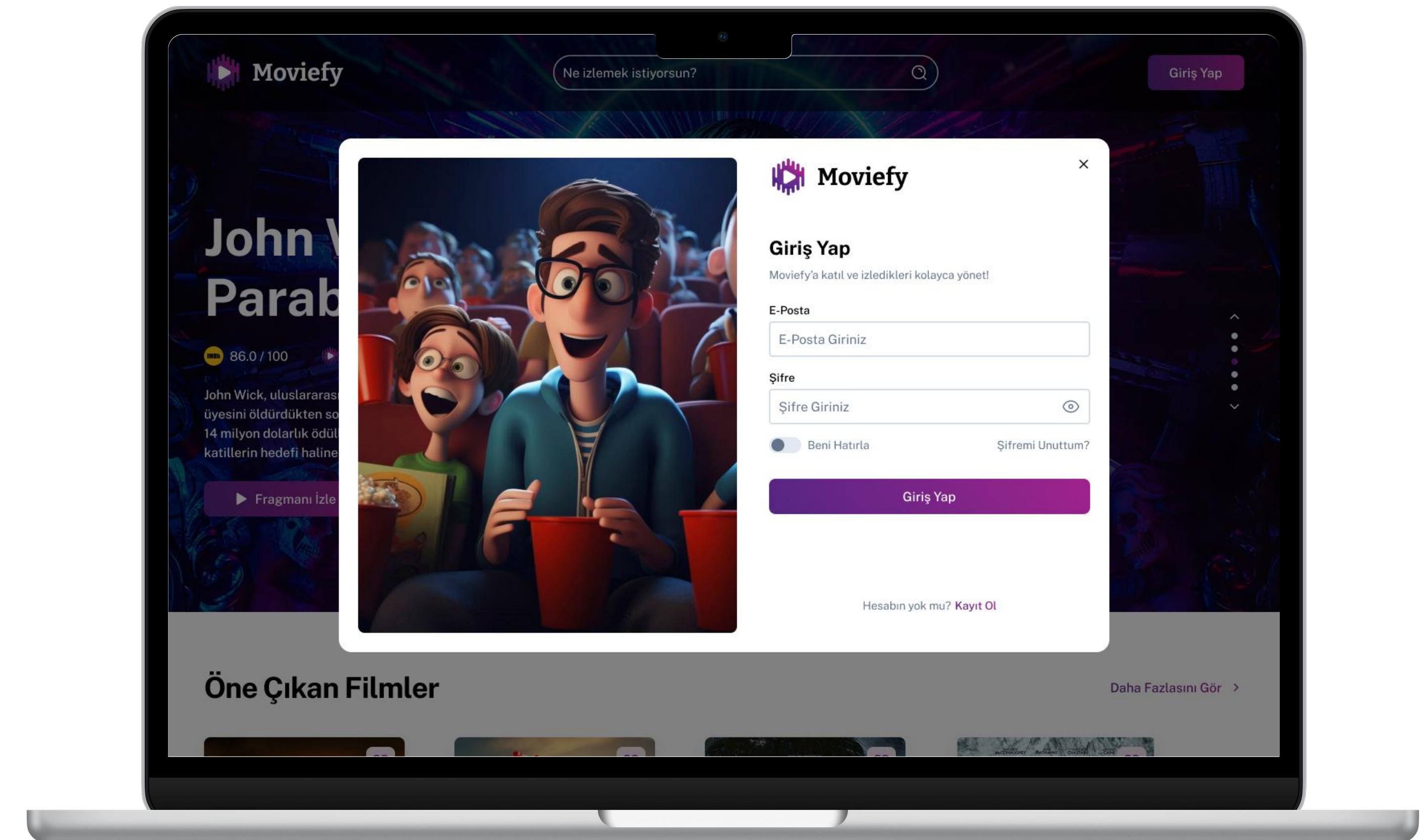
if __name__ == '__main__':
    graph_thread = Thread(target=create_graph)
    graph_thread.start()

    app.run(debug=True)

    graph_thread.join()
    app.run(debug=True)
```

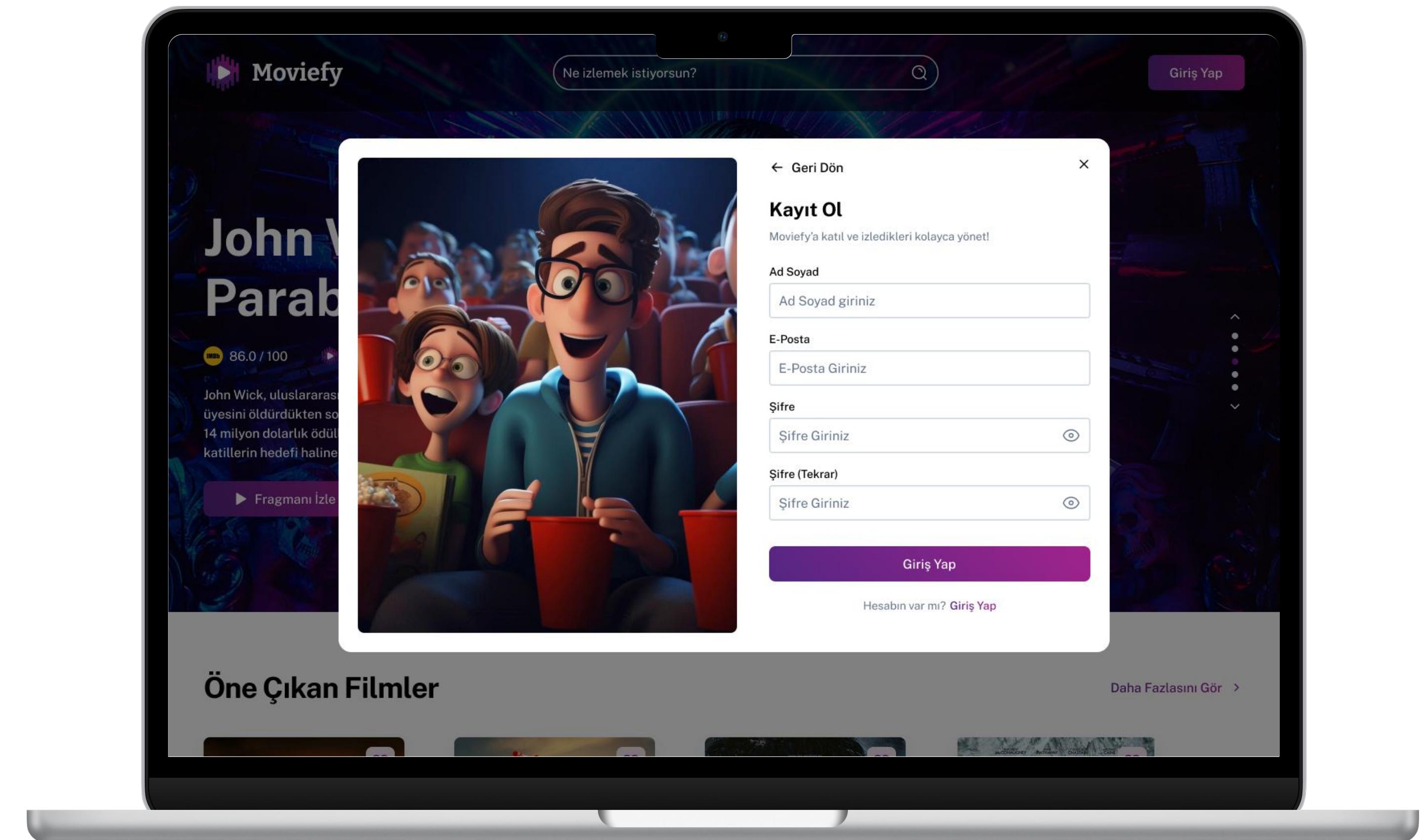
Kullanıcı Arayüzü

Giriş Yap



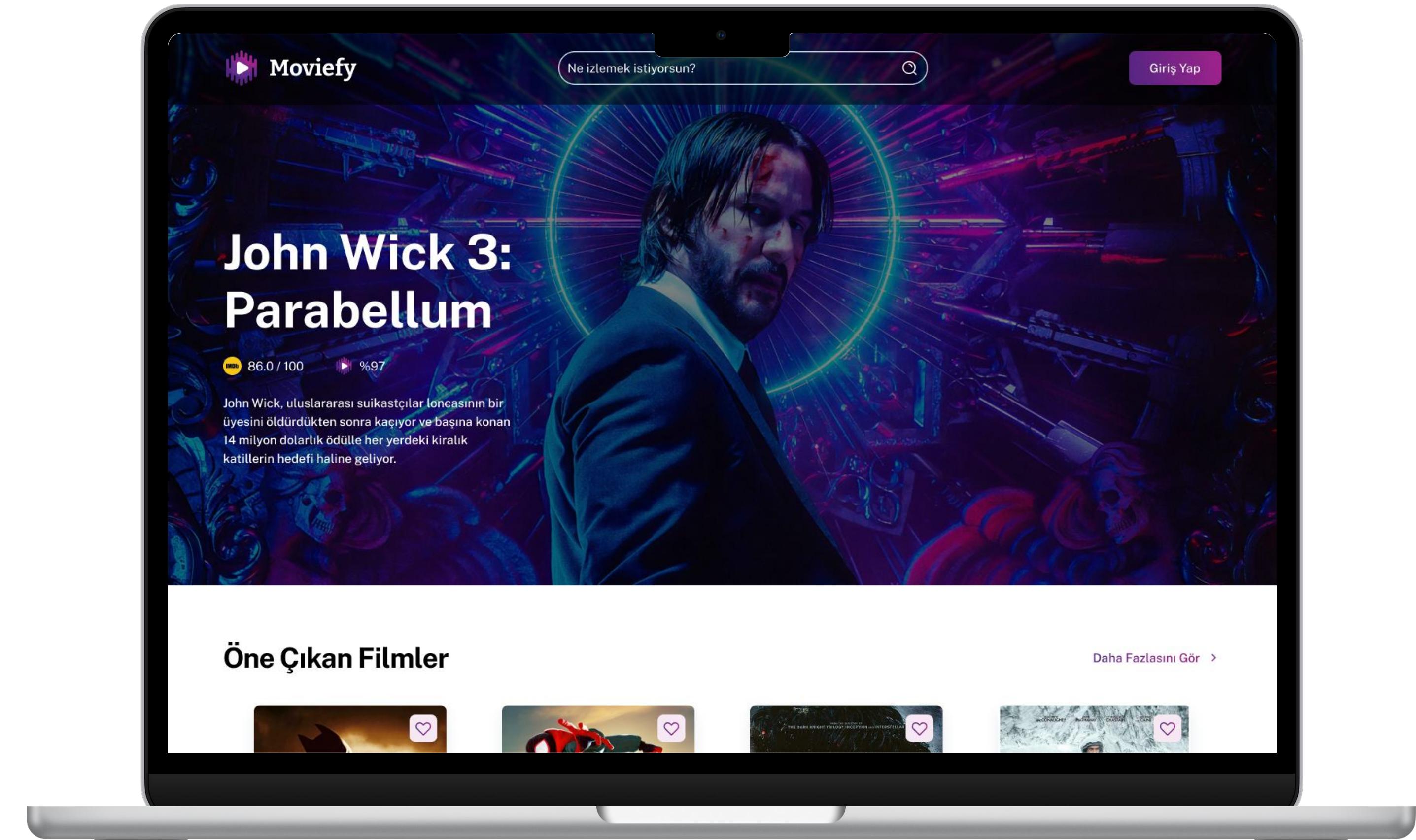
Kullanıcı Arayüzü

Kayıt Ol



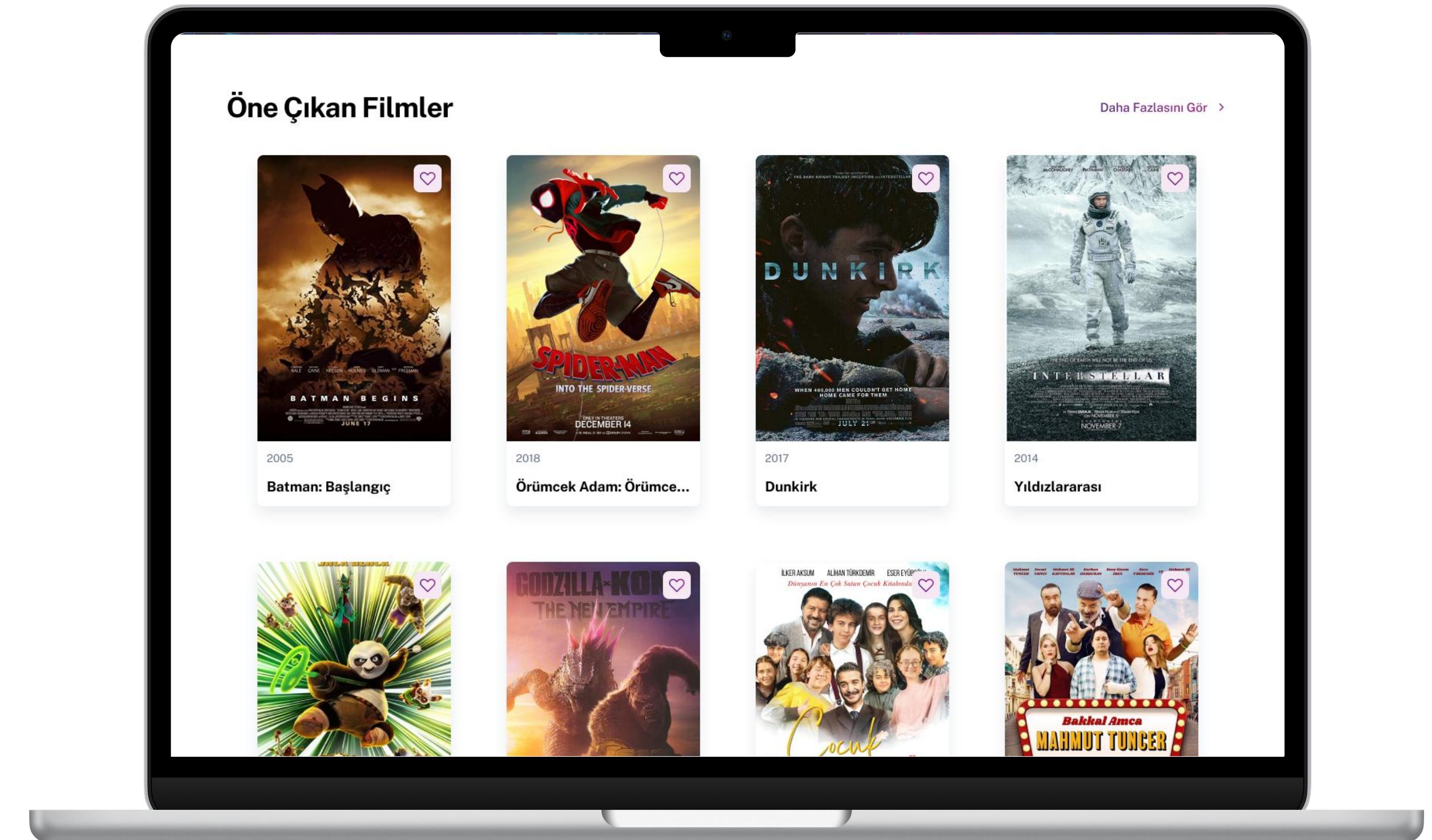
Kullanıcı Arayüzü

Anasayfa



Kullanıcı Arayüzü

Anasayfa → Öne Çıkan Filmler



Kullanıcı Arayüzü

Film Detayı

The image shows a tablet displaying the Moviefy app interface. The top navigation bar includes the Moviefy logo, a search bar with the placeholder "Ne izlemek istiyorsun?", and a user profile dropdown for "Pelin Altürk". The main content area features a movie poster for "INTERSTELLAR" with a male astronaut standing on a desolate, rocky planet. Below the poster, the title "Yıldızlararası" is displayed in large bold letters. A descriptive paragraph in Turkish follows: "Lorem ipsum dolor sit amet consectetur. Dictumst eu bibendum non vel viverra nibh egestas non vitae. Duis proin neque vitae mauris nisl massa convallis aenean quam. Enim scelerisque egestas mi sit. Pharetra ac sem morbi scelerisque id nunc eget. Ac posuere velit tortor lectus nullam. Duis in volutpat nibh rhoncus dictum aenean. Vel pretium dui congue dolor. Facilisis amet odio ac ultricies. Aliquet elementum a sed morbi imperdiet turpis etiam tortor lectus. Duis tincidunt at eget pulvinar sed hendrerit cursus fusce tempor." Below the text is the production year "Yapım Yılı: 2005" and two buttons: "Favoriye Ekle" (white background) and "Favoriden Kaldır" (purple background). The bottom section is titled "Senin için Önerilen" and shows four movie thumbnails: "The Dark Knight", "Spider-Man: Homecoming", "DUNKIRK", and "INTERSTELLAR" again, each with a small heart icon in the top right corner.



Bizi Dinlediğiniz İçin
Teşekkürler.