movies.py

```python
from flask import Blueprint, request, make_response, jsonify
from decorators import jwt_required, admin_required
import globals
from bson import ObjectId

movies_bp = Blueprint("movies_bp", __name__)

movies = globals.db.movieInfo


@movies_bp.route("/api/v1.0/movies", methods=["GET"])
def show_all_movies():
    page_num, page_size = 1, 10
    if request.args.get("pn"):
        page_num = int(request.args.get("pn"))
    if request.args.get("ps"):
        page_size = int(request.args.get("ps"))
    page_start = page_size * (page_num - 1)

    data_to_return = []
    for movie in movies.find().skip(page_start).limit(page_size):
        movie["_id"] = str(movie["_id"])
        for review in movie["reviews"]:
            review["id"] = str(review["_id"])
        data_to_return.append(movie)

    return make_response(jsonify(data_to_return), 200)


@movies_bp.route("/api/v1.0/movies/<string:id>", methods=["GET"])
def show_one_movie(id):
    movie = movies.find_one({"_id": ObjectId(id)})
    if movie is not None:
        movie["_id"] = str(movie["_id"])
        for review in movie["reviews"]:
            review["_id"] = str(review["_id"])
        return make_response(jsonify(movie), 200)
    else:
        return make_response(jsonify({"error": "Invalid Movie ID"}), 404)


@movies_bp.route("/api/v1.0/movies", methods=["POST"])
@jwt_required
def add_movie():
    if ("Title" in request.form and "Year" in request.form and "IMDb Rating" in request.form and "Runtime" in request.form and "Genre" in request.form and "Director" in request.form and "Actors" in request.form and "Plot" in request.form):
        new_movie = {
            "Title": request.form["Title"],
            "Year": request.form["Year"],
            "IMDb Rating": request.form["IMDb Rating"],
            "Runtime": request.form["Runtime"],
            "Genre": request.form["Genre"],
            "Director": request.form["Director"],
            "Actors": request.form["Actors"],
            "Plot": request.form["Plot"],
            "reviews": [],
        }
        new_movie_id = movies.insert_one(new_movie)
        new_movie_link = "http://localhost:5000/api/v1.0/movies/" + str(
            new_movie_id.inserted_id
        )
        return make_response(jsonify({"url": new_movie_link}), 201)
    else:
        return make_response(jsonify({"error": "Missing form data"}), 404)


@movies_bp.route("/api/v1.0/movies/<string:id>", methods=["PUT"])
@jwt_required
def edit_movie(id):
    if ("Title" in request.form and "Year" in request.form and "IMDb Rating" in request.form and "Runtime" in request.form and "Genre" in request.form and "Director" in request.form and "Actors" in request.form and "Plot" in request.form):
        result = movies.update_one(
            {"_id": ObjectId(id)},
            {
                "$set": {
                    "Title": request.form["Title"],
                    "Year": request.form["Year"],
                    "IMDb Rating": request.form["IMDb Rating"],
                    "Runtime": request.form["Runtime"],
                    "Genre": request.form["Genre"],
                    "Director": request.form["Director"],
                    "Actors": request.form["Actors"],
                    "Plot": request.form["Plot"],
                    "reviews": [],
                }
            },
        )
        if result.matched_count == 1:
            edited_movie_link = "http://localhost:5000/api/v1.0/movies/" + id
            return make_response(jsonify({"url": edited_movie_link}), 200)
        else:
            return make_response(jsonify({"error": "Invalid movie ID"}), 404)
    else:
        return make_response(jsonify({"error": "Missing form data"}), 404)


@movies_bp.route("/api/v1.0/movies/<string:id>", methods=["DELETE"])
@jwt_required
@admin_required
def delete_movie(id):
    result = movies.delete_one({"_id": ObjectId(id)})
    if result.deleted_count == 1:
        return make_response(jsonify({}), 204)
    else:
        return make_response(jsonify({"error": "Invalid movie ID"}), 404)


@movies_bp.route("/api/v1.0/movies/title", methods=["GET"])
def search_movies():
    query = request.args.get("Title")
    if not query:
        return jsonify({"error": "Missing title parameter"}), 400

    data_to_return = []
    for movie in movies.find():
        title = movie.get("Title", "")
        if query.lower() in title.lower():
            movie["_id"] = str(movie["_id"])
            if "reviews" in movie:
                for review in movie["reviews"]:
                    if "_id" in review:
                        review["_id"] = str(review["_id"])
            data_to_return.append(movie)

    if not data_to_return:
        return make_response(jsonify({"error": "Invalid Title"}), 404)

    return make_response(jsonify(data_to_return), 200)


@movies_bp.route("/api/v1.0/movies/director", methods=["GET"])
def search_movies_by_director():
    director = request.args.get("Director")

    data_to_return = []
    for movie in movies.find():
        title = movie.get("Director", "")
        if director.lower() in title.lower():
            movie["_id"] = str(movie["_id"])
            if "reviews" in movie:
                for review in movie["reviews"]:
                    if "_id" in review:
                        review["_id"] = str(review["_id"])
            data_to_return.append(movie)

    if not data_to_return:
        return make_response(jsonify({"error": "Invalid Director"}), 404)

    return make_response(jsonify(data_to_return), 200)


@movies_bp.route("/api/v1.0/movies/searchyear", methods=["GET"])
def search_by_year():
    year = request.args.get("Year")
    results = movies.find({"Year": year})
    data_to_return = []

    for movie in results:
        movie["_id"] = str(movie["_id"])

        if "reviews" in movie:
            for review in movie["reviews"]:
                if "_id" in review:
                    review["_id"] = str(review["_id"])
        data_to_return.append(movie)

    if not data_to_return:
        return make_response(jsonify({"error": "Invalid Year"}), 404)

    return make_response(jsonify(data_to_return), 200)


@movies_bp.route("/api/v1.0/movies/year-range", methods=["GET"])
def movies_by_year_range():
    start = int(request.args.get("start"))
    end = int(request.args.get("end"))

    results = movies.find({"Year": {"$gte": start, "$lte": end}})

    data_to_return = []
    for movie in results:
        movie["_id"] = str(movie["_id"])

        if "reviews" in movie:
            for review in movie["reviews"]:
                if "_id" in review:
                    review["_id"] = str(review["_id"])
        data_to_return.append(movie)

    if not data_to_return:
        return make_response(jsonify({"error": "Invalid Start and End Dates"}), 404)

    return make_response(jsonify(data_to_return), 200)


@movies_bp.route("/api/v1.0/movies/genre", methods=["GET"])
def movies_by_genre():
    genre = request.args.get("Genre")
    genre = genre.lower()
    results = movies.find()

    data_to_return = []
    for movie in results:
        genre_field = movie.get("Genre", "")
        genre_list = genre_field.lower().split(",")

        if genre in [genre.replace(" ", "") for genre in genre_list]:
            movie["_id"] = str(movie["_id"])

            if "reviews" in movie:
                for review in movie["reviews"]:
                    if "_id" in review:
                        review["_id"] = str(review["_id"])

            data_to_return.append(movie)

    if not data_to_return:
        return make_response(jsonify({"error": "No Genres were found"}), 404)

    return make_response(jsonify(data_to_return), 200)


@movies_bp.route("/api/v1.0/movies/minrating/<float:minrating>", methods=["GET"])
def showMoviesAboveMinRating(minrating):
    page_num, page_size = 1, 10
    if request.args.get("pn"):
        page_num = int(request.args.get("pn"))
    if request.args.get("ps"):
        page_size = int(request.args.get("ps"))
    page_start = page_size * (page_num - 1)

    data_to_return = []
    for movie in (
        movies.find({"IMDb Rating": {"$gte": minrating}}).skip(page_start).limit(page_size)
    ):
        movie["_id"] = str(movie["_id"])
        if "reviews" in movie:
            for review in movie["reviews"]:
                if "_id" in review:
                    review["_id"] = str(review["_id"])
        data_to_return.append(movie)

    if not data_to_return:
        return make_response(
            jsonify(
                {
                    "error": "No movies were found with a rating more than or equal to " + str(minrating)
                }
            ),
            404,
        )

    return make_response(jsonify(data_to_return), 200)


@movies_bp.route("/api/v1.0/movies/by-actor/<actor_name>", methods=["GET"])
def movies_by_actor(actor_name):
    query = {
        "$expr": {
            "$in": [
                actor_name.lower(),
                {
                    "$map": {
                        "input": {"$split": ["$Actors", ", "]},
                        "as": "actor",
                        "in": {"$toLower": "$$actor"},
                    }
                },
            ]
        }
    }
    projection = {"_id": 0, "Title": 1, "Actors": 1}
    results = list(movies.find(query, projection))
    return make_response(jsonify(results), 200)
```

reviews.py

```python
from flask import Blueprint, request, make_response, jsonify
from decorators import jwt_required, admin_required
from bson import ObjectId
import globals

reviews_bp = Blueprint("reviews_bp", __name__)

movies = globals.db.movieInfo


@reviews_bp.route("/api/v1.0/movies/<string:id>/reviews", methods=["POST"])
@jwt_required
def add_new_review(id):
    new_review = {
        "_id": ObjectId(),
        "username": request.form["username"],
        "comment": request.form["comment"],
        "stars": request.form["stars"],
    }
    result = movies.update_one(
        {"_id": ObjectId(id)}, {"$push": {"reviews": new_review}}
    )
    if result.matched_count == 1:
        new_review_link = ("http://localhost:5000/api/v1.0/movies/" + id + "/reviews/" + str(new_review["_id"]))
        return make_response(jsonify({"url": new_review_link}), 201)
    else:
        return make_response(jsonify({"error": "Invalid movie ID"}), 404)


@reviews_bp.route("/api/v1.0/movies/<string:id>/reviews", methods=["GET"])
def fetch_all_reviews(id):
    data_to_return = []
    movie = movies.find_one({"_id": ObjectId(id)}, {"reviews": 1, "_id": 0})
    for review in movie["reviews"]:
        review["_id"] = str(review["_id"])
        data_to_return.append(review)
    return make_response(jsonify(data_to_return), 200)


@reviews_bp.route("/api/v1.0/movies/<mid>/reviews/<rid>", methods=["GET"])
def fetch_one_review(mid, rid):
    movie = movies.find_one({"reviews._id": ObjectId(rid)}, {"_id": 0, "reviews.$": 1})
    if movie is None:
        return make_response(jsonify({"error": "Invalid movie ID or review ID"}), 404)
    movie["reviews"][0]["_id"] = str(movie["reviews"][0]["_id"])
    return make_response(jsonify(movie["reviews"][0]), 200)


@reviews_bp.route("/api/v1.0/movies/<mid>/reviews/<rid>", methods=["PUT"])
@jwt_required
def edit_review(mid, rid):
    edited_review = {
        "reviews.$.username": request.form["username"],
        "reviews.$.comment": request.form["comment"],
        "reviews.$.stars": request.form["stars"],
    }
    movies.update_one({"reviews._id": ObjectId(rid)}, {"$set": edited_review})
    edit_review_url = "http://localhost:5000/api/v1.0/movies/" + mid + "/reviews/" + rid
    return make_response(jsonify({"url": edit_review_url}), 200)


@reviews_bp.route("/api/v1.0/movies/<mid>/reviews/<id>", methods=["DELETE"])
@jwt_required
@admin_required
def delete_review(mid, rid):
    movies.update_one(
        {"_id": ObjectId(mid)}, {"$pull": {"reviews": {"_id": ObjectId(rid)}}}
    )
    return make_response(jsonify({}), 204)


@reviews_bp.route("/api/v1.0/movies/reviews/stars", methods=["GET"])
def reviews_by_stars():
    stars = request.args.get("stars")
    stars = str(stars)
    results = movies.find()
    data_to_return = []

    for movie in results:
        if "reviews" in movie:
            for review in movie["reviews"]:
                if "_id" in review:
                    review["_id"] = str(review["_id"])

                if "stars" in review and review["stars"] == stars:
                    movie["_id"] = str(movie["_id"])
                    data_to_return.append(movie)
                    break
    if not data_to_return:
        return make_response(jsonify({"error": "Invalid Stars Parameter"}), 404)

    return make_response(jsonify(data_to_return), 200)


@reviews_bp.route("/api/v1.0/movies/<mid>/reviews/stars", methods=["GET"])
def movie_name_and_stars(mid):
    stars = request.args.get("stars")
    if not stars:
        return jsonify({"error": "Missing stars parameter"}), 400

    movie = movies.find_one({"_id": ObjectId(mid)})

    if not movie:
        return jsonify({"error": "Movie not found"}), 404

    if "_id" in movie:
        movie["_id"] = str(movie["_id"])

    if "reviews" in movie:
        for review in movie["reviews"]:
            if "_id" in review:
                review["_id"] = str(review["_id"])
        movie["reviews"] = [
            review
            for review in movie["reviews"]
            if "stars" in review and str(review["stars"]) == str(stars)
        ]

    return make_response(jsonify(movie), 200)


@reviews_bp.route("/api/v1.0/movies/reviewed_by/<string:username>", methods=["GET"])
def show_movies_reviewed_by_user(username):
    pipeline = [{"$match": {"reviews": {"$elemMatch": {"username": username}}}}]
    data_to_return = []
    for movie in movies.aggregate(pipeline):
        movie["_id"] = str(movie["_id"])
        for review in movie["reviews"]:
            review["_id"] = str(review["_id"])
        data_to_return.append(movie)

    if not data_to_return:
        return make_response(jsonify({"error": "No reviews for user " + username}), 404)

    return make_response(jsonify(data_to_return), 200)

```

awards.py

```python
from flask import Blueprint, jsonify, make_response, request
import globals
from decorators import jwt_required, admin_required
from bson import ObjectId

awards_bp = Blueprint("awards_bp", __name__)

movies = globals.db.movieInfo
awards = globals.db.awards


@awards_bp.route("/api/v1.0/awards/all", methods=["GET"])
def get_all_awards():
    data_to_return = []
    for award in awards.find():
        award["_id"] = str(award["_id"])
        award["movie_id"] = str(award["movie_id"])
        data_to_return.append(award)
    return make_response(jsonify(data_to_return), 200)


@awards_bp.route("/api/v1.0/movies/<movie_id>/awards", methods=["GET"])
def get_awards_for_movie(movie_id):
    data_to_return = []
    for award in awards.find({"movie_id": ObjectId(movie_id)}):
        award["_id"] = str(award["_id"])
        award["movie_id"] = str(award["movie_id"])
        data_to_return.append(award)
    if not data_to_return:
        return make_response(jsonify({"error": "No awards found for this movie"}), 404)
    return make_response(jsonify(data_to_return), 200)


@awards_bp.route("/api/v1.0/awards/<int:year>/winners", methods=["GET"])
def get_award_winners_by_year(year):
    data_to_return = []
    for award in awards.find({"year": year, "won": True}):
        award["_id"] = str(award["_id"])
        award["movie_id"] = str(award["movie_id"])
        data_to_return.append(award)
    if not data_to_return:
        return make_response(
            jsonify({"error": "No award winners found for year " + str(year)}), 404
        )
    return make_response(jsonify(data_to_return), 200)


@awards_bp.route("/api/v1.0/awards/add", methods=["POST"])
@jwt_required
def add_award():
    data = request.get_json()
    if not data or "title" not in data:
        return make_response(jsonify({"error": "Missing form data"}), 404)

    movie = movies.find_one({"Title": data["title"]})
    if not movie:
        return make_response(jsonify({"error": "Movie not found in database"}), 400)

    new_award = {
        "movie_id": movie["_id"],
        "award_name": data["award_name"],
        "category": data["category"],
        "year": data["year"],
        "won": data["won"],
    }

    result = awards.insert_one(new_award)
    new_award_link = "http://localhost:5000/api/v1.0/awards/" + str(result.inserted_id)
    return make_response(jsonify({"url": new_award_link}), 201)


@awards_bp.route("/api/v1.0/awards/<string:id>", methods=["PUT"])
@jwt_required
def edit_award(id):
    if (
        "award_name" in request.form and "category" in request.form and "year" in request.form and "won" in request.form and "movie_id" in request.form):
        result = awards.update_one(
            {"_id": ObjectId(id)},
            {
                "$set": {
                    "award_name": request.form["award_name"],
                    "category": request.form["category"],
                    "year": int(request.form["year"]),
                    "won": request.form["won"].lower == "true",
                    "movie_id": ObjectId(request.form["movie_id"]),
                }
            },
        )
        if result.matched_count == 1:
            edited_award_link = "http://localhost:5000/api/v1.0/awards/" + id
            return make_response(jsonify({"url": edited_award_link}), 200)
        else:
            return make_response(jsonify({"error": "Invalid award ID"}), 404)
    else:
        return make_response(jsonify({"error": "Missing form data"}), 404)


@awards_bp.route("/api/v1.0/awards/<string:id>", methods=["DELETE"])
@jwt_required
@admin_required
def delete_award(id):
    result = awards.delete_one({"_id": ObjectId(id)})
    if result.deleted_count == 1:
        return make_response(jsonify({}), 204)
    else:
        return make_response(jsonify({"error": "Invalid award ID"}), 404)

```

```python
@awards_bp.route("/api/v1.0/awards/awardcompany/<string:award_name>", methods=["GET"])
def get_awards_by_award_name(award_name):
    data_to_return = []
    for award in awards.find({"award_name": award_name}):
        award["_id"] = str(award["_id"])
        award["movie_id"] = str(award["movie_id"])
        data_to_return.append(award)
    if not data_to_return:
        return make_response(
            jsonify({"error": f"No awards found in award_name '{award_name}'"}), 404
        )
    return make_response(jsonify(data_to_return), 200)

```

aggregate.py

```python
from flask import Blueprint, jsonify, make_response
import globals

aggregate_bp = Blueprint("aggregate_bp", __name__)

movies = globals.db.movieInfo


@aggregate_bp.route("/api/v1.0/movies/genre-count", methods=["GET"])
def genre_count():
    pipeline = [
        {
            "$addFields": {
                "genreArray": {
                    "$split": [
                        "$Genre",
                        ", ",
                    ]  # Split the Array into each individual Genre
                }
            }
        },
        {"$unwind": "$genreArray"},
        {"$group": {"_id": "$genreArray", "count": {"$sum": 1}}},
        {"$project": {"_id": 0, "genre": "$_id", "count": 1}},
        {"$sort": {"count": -1}},
    ]
    results = list(movies.aggregate(pipeline))
    return make_response(jsonify(results), 200)


@aggregate_bp.route("/api/v1.0/movies/director-count", methods=["GET"])
def director_count():
    pipeline = [
        {
            "$addFields": {
                "directorArray": {
                    "$split": [
                        "$Director",
                        ", ",
                    ]  # Split the Array into each Individual Director
                }
            }
        },
        {"$unwind": "$directorArray"},
        {"$group": {"_id": "$directorArray", "count": {"$sum": 1}}},
        {"$project": {"_id": 0, "director": "$_id", "count": 1}},
        {"$sort": {"count": -1}},
    ]
    results = list(movies.aggregate(pipeline))
    return make_response(jsonify(results), 200)


@aggregate_bp.route("/api/v1.0/movies/actor-count", methods=["GET"])
def actor_count():
    pipeline = [
        {"$addFields": {"actorArray": {"$split": ["$Actors", ", "]}}},
        {"$unwind": "$actorArray"},
        {"$group": {"_id": "$actorArray", "count": {"$sum": 1}}},
        {"$project": {"_id": 0, "actor": "$_id", "count": 1}},
        {"$sort": {"count": -1}},
    ]
    results = list(movies.aggregate(pipeline))
    return make_response(jsonify(results), 200)


@aggregate_bp.route("/api/v1.0/movies/average-review-rating", methods=["GET"])
def average_review_rating():
    pipeline = [
        {"$unwind": "$reviews"},
        {"$addFields": {"numericStars": {"$toDouble": "$reviews.stars"}}},
        {"$group": {"_id": "$Title", "averageReviewRating": {"$avg": "$numericStars"}}},
        {"$project": {"_id": 0, "Title": "$_id", "averageReviewRating": 1}},
    ]
    results = list(movies.aggregate(pipeline))
    return make_response(jsonify(results), 200)

```

auth.py

```python
from flask import Blueprint, request, make_response, jsonify
from decorators import jwt_required, admin_required
import globals
import bcrypt
import jwt
import datetime

auth_bp = Blueprint("auth_bp", __name__)

users = globals.db.Users
blacklist = globals.db.blacklist

@auth_bp.route('/api/v1.0/logout', methods=["GET"])
@jwt_required
def logout():
    token = request.headers['x-access-token']
    blacklist.insert_one({"token":token})
    return make_response(jsonify( {'message' : 'Logout successful' } ), 200 )

@auth_bp.route('/api/v1.0/login', methods=['GET'])
def login():
    auth = request.authorization
    if auth:
        user = users.find_one({'username':auth.username})
        if user is not None:
            if bcrypt.checkpw(bytes(auth.password, 'UTF-8'), user["password"]):
                token = jwt.encode({
                    'user' : auth.username,
                    'admin' : user['admin'],
                    'exp' : datetime.datetime.now(datetime.UTC) + datetime.timedelta(minutes=30)
                }, globals.secret_key, algorithm="HS256")
                return make_response(jsonify({'token': token}), 200)

            else:
                return make_response(jsonify({'message' : 'Bad password'}), 401)
        else:
            return make_response(jsonify({'message' : 'Bad username'}), 401)
    return make_response(jsonify({'message' : 'Authentication required'}), 401)

```

app.py

```python
from flask import Flask
from blueprints.movies.movies import movies_bp
from blueprints.reviews.reviews import reviews_bp
from blueprints.auth.auth import auth_bp
from blueprints.aggregate.aggregate import aggregate_bp
from blueprints.awards.awards import awards_bp

app = Flask(__name__)
app.register_blueprint(movies_bp)
app.register_blueprint(reviews_bp)
app.register_blueprint(auth_bp)
app.register_blueprint(aggregate_bp)
app.register_blueprint(awards_bp)

if __name__ == "__main__":
    app.run(debug=True)
```

decorators.py

```python
from flask import request, jsonify, make_response
from pymongo import MongoClient
import jwt
from functools import wraps
import globals

client = MongoClient("mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.8")
db = client.Movies
movies = db.movieInfo
users = db.users
blacklist = globals.db.blacklist


def jwt_required(func):
    @wraps(func)
    def jwt_required_wrapper(*args, **kwargs):
        token = None
        if "x-access-token" in request.headers:
            token = request.headers["x-access-token"]
        try:
            data = jwt.decode(token, globals.secret_key, algorithms="HS256")
        except:
            return make_response(jsonify({"message": "Token is invalid"}), 401)
        bl_token = blacklist.find_one({"token": token})
        if bl_token is not None:
            return make_response(jsonify({"message": "Token has been cancelled"}), 401)
        return func(*args, **kwargs)

    return jwt_required_wrapper


def admin_required(func):
    @wraps(func)
    def admin_required_wrapper(*args, **kwargs):
        token = request.headers["x-access-token"]
        data = jwt.decode(token, globals.secret_key, algorithms="HS256")
        if data["admin"]:
            return func(*args, **kwargs)
        else:
            return make_response(jsonify({"message": "Admin access required"}), 401)

    return admin_required_wrapper

```

globals.py

```python
from pymongo import MongoClient

secret_key = 'movieinfodatabasesecretkey'

client = MongoClient("mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.8")
db = client.Movies
```