movies.py

```
1 | from flask import Blueprint, request, make_response, jsonify
2 | from decorators import jwt_required, admin_required
3 | import globals
4 | from bson import ObjectId
5 |
6 | movies_bp = Blueprint("movies_bp", __name__)
7 |
8 | movies = globals.db.movieInfo
9 |
10|
11| @movies_bp.route("/api/v1.0/movies", methods=["GET"])
12| def show_all_movies():
13|     page_num, page_size = 1, 10
14|     if request.args.get("pn"):
15|         page_num = int(request.args.get("pn"))
16|     if request.args.get("ps"):
17|         page_size = int(request.args.get("ps"))
18|     page_start = page_size * (page_num - 1)
19|
20|     data_to_return = []
21|     for movie in movies.find().skip(page_start).limit(page_size):
22|         movie["_id"] = str(movie["_id"])
23|         for review in movie["reviews"]:
24|             review["id"] = str(review["_id"])
25|         data_to_return.append(movie)
26|
27|     return make_response(jsonify(data_to_return), 200)
28|
29|
30| @movies_bp.route("/api/v1.0/movies/<string:id>", methods=["GET"])
31| def show_one_movie(id):
32|     movie = movies.find_one({"_id": ObjectId(id)})
33|     if movie is not None:
34|         movie["_id"] = str(movie["_id"])
35|         for review in movie["reviews"]:
36|             review["_id"] = str(review["_id"])
37|         return make_response(jsonify(movie), 200)
38|     else:
39|         return make_response(jsonify({"error": "Invalid Movie ID"}),
404)
40|
41|
42| @movies_bp.route("/api/v1.0/movies", methods=["POST"])
43| @jwt_required
44| def add_movie():
45|     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):
46|         new_movie = {
47|             "Title": request.form["Title"],
48|             "Year": request.form["Year"],
49|             "IMDb Rating": request.form["IMDb Rating"],
50|             "Runtime": request.form["Runtime"],
51|             "Genre": request.form["Genre"],
52|             "Director": request.form["Director"],
53|             "Actors": request.form["Actors"],
54|             "Plot": request.form["Plot"],
55|             "reviews": [],
56|         }
```

```
57|         new_movie_id = movies.insert_one(new_movie)
58|         new_movie_link = "http://localhost:5000/api/v1.0/movies/" +
str(
59|             new_movie_id.inserted_id
60|         )
61|         return make_response(jsonify({"url": new_movie_link}), 201)
62|     else:
63|         return make_response(jsonify({"error": "Missing form data"}),
404)
64|
65|
66| @movies_bp.route("/api/v1.0/movies/<string:id>", methods=["PUT"])
67| @jwt_required
68| def edit_movie(id):
69|     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):
70|         result = movies.update_one(
71|             {"_id": ObjectId(id)},
72|             {
73|                 "$set": {
74|                     "Title": request.form["Title"],
75|                     "Year": request.form["Year"],
76|                     "IMDb Rating": request.form["IMDb Rating"],
77|                     "Runtime": request.form["Runtime"],
78|                     "Genre": request.form["Genre"],
79|                     "Director": request.form["Director"],
80|                     "Actors": request.form["Actors"],
81|                     "Plot": request.form["Plot"],
82|                     "reviews": [],
83|                 }
84|             },
85|         )
86|         if result.matched_count == 1:
87|             edited_movie_link =
"http://localhost:5000/api/v1.0/movies/" + id
88|             return make_response(jsonify({"url": edited_movie_link}),
200)
89|         else:
90|             return make_response(jsonify({"error": "Invalid movie
ID"}), 404)
91|     else:
92|         return make_response(jsonify({"error": "Missing form data"}),
404)
93|
94|
95| @movies_bp.route("/api/v1.0/movies/<string:id>", methods=["DELETE"])
96| @jwt_required
97| @admin_required
98| def delete_movie(id):
99|     result = movies.delete_one({"_id": ObjectId(id)})
100|     if result.deleted_count == 1:
101|         return make_response(jsonify({}), 204)
102|     else:
103|         return make_response(jsonify({"error": "Invalid movie ID"}),
404)
104|
105|
106| @movies_bp.route("/api/v1.0/movies/title", methods=["GET"])
107| 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)

```

```
168|     return make_response(jsonify(data_to_return), 200)
169|
170|
171| @movies_bp.route("/api/v1.0/movies/year-range", methods=["GET"])
172| def movies_by_year_range():
173|     start = int(request.args.get("start"))
174|     end = int(request.args.get("end"))
175|
176|     results = movies.find({"Year": {"$gte": start, "$lte": end}})
177|
178|     data_to_return = []
179|     for movie in results:
180|         movie["_id"] = str(movie["_id"])
181|
182|         if "reviews" in movie:
183|             for review in movie["reviews"]:
184|                 if "_id" in review:
185|                     review["_id"] = str(review["_id"])
186|         data_to_return.append(movie)
187|
188|     if not data_to_return:
189|         return make_response(jsonify({"error": "Invalid Start and End
Dates"}), 404)
190|
191|     return make_response(jsonify(data_to_return), 200)
192|
193|
194| @movies_bp.route("/api/v1.0/movies/genre", methods=["GET"])
195| def movies_by_genre():
196|     genre = request.args.get("Genre")
197|     genre = genre.lower()
198|     results = movies.find()
199|
200|     data_to_return = []
201|     for movie in results:
202|         genre_field = movie.get("Genre", "")
203|         genre_list = genre_field.lower().split(",")
204|
205|         if genre in [genre.replace(" ", "") for genre in genre_list]:
206|             movie["_id"] = str(movie["_id"])
207|
208|             if "reviews" in movie:
209|                 for review in movie["reviews"]:
210|                     if "_id" in review:
211|                         review["_id"] = str(review["_id"])
212|
213|             data_to_return.append(movie)
214|
215|     if not data_to_return:
216|         return make_response(jsonify({"error": "No Genres were
found"}), 404)
217|
218|     return make_response(jsonify(data_to_return), 200)
219|
220|
221| @movies_bp.route("/api/v1.0/movies/minrating/<float:minrating>",
methods=["GET"])
222| def showMoviesAboveMinRating(minrating):
223|     page_num, page_size = 1, 10
224|     if request.args.get("pn"):
225|         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

```
1 | from flask import Blueprint, request, make_response, jsonify
2 | from decorators import jwt_required, admin_required
3 | from bson import ObjectId
4 | import globals
5 |
6 | reviews_bp = Blueprint("reviews_bp", __name__)
7 |
8 | movies = globals.db.movieInfo
9 |
10|
11| @reviews_bp.route("/api/v1.0/movies/<string:id>/reviews",
methods=["POST"])
12| @jwt_required
13| def add_new_review(id):
14|     new_review = {
15|         "_id": ObjectId(),
16|         "username": request.form["username"],
17|         "comment": request.form["comment"],
18|         "stars": request.form["stars"],
19|     }
20|     result = movies.update_one(
21|         {"_id": ObjectId(id)}, {"$push": {"reviews": new_review}}
22|     )
23|     if result.matched_count == 1:
24|         new_review_link = ("http://localhost:5000/api/v1.0/movies/" +
id + "/reviews/" + str(new_review["_id"]))
25|         return make_response(jsonify({"url": new_review_link}), 201)
26|     else:
27|         return make_response(jsonify({"error": "Invalid movie ID"}),
404)
28|
29|
30| @reviews_bp.route("/api/v1.0/movies/<string:id>/reviews",
methods=["GET"])
31| def fetch_all_reviews(id):
32|     data_to_return = []
33|     movie = movies.find_one({"_id": ObjectId(id)}, {"reviews": 1,
"_id": 0})
34|     for review in movie["reviews"]:
35|         review["_id"] = str(review["_id"])
36|         data_to_return.append(review)
37|     return make_response(jsonify(data_to_return), 200)
38|
39|
40| @reviews_bp.route("/api/v1.0/movies/<mid>/reviews/<rid>",
methods=["GET"])
41| def fetch_one_review(mid, rid):
42|     movie = movies.find_one({"reviews._id": ObjectId(rid)}, {"_id": 0,
"reviews.$": 1})
43|     if movie is None:
44|         return make_response(jsonify({"error": "Invalid movie ID or
review ID"}), 404)
45|     movie["reviews"][0]["_id"] = str(movie["reviews"][0]["_id"])
46|     return make_response(jsonify(movie["reviews"][0]), 200)
47|
48|
49| @reviews_bp.route("/api/v1.0/movies/<mid>/reviews/<rid>",
methods=["PUT"])
50| @jwt_required
51| def edit_review(mid, rid):
```

```
52|     edited_review = {
53|         "reviews.$.username": request.form["username"],
54|         "reviews.$.comment": request.form["comment"],
55|         "reviews.$.stars": request.form["stars"],
56|     }
57|     movies.update_one({"reviews._id": ObjectId(rid)}, {"$set":
edited_review})
58|     edit_review_url = "http://localhost:5000/api/v1.0/movies/" + mid +
"/reviews/" + rid
59|     return make_response(jsonify({"url": edit_review_url}), 200)
60|
61|
62| @reviews_bp.route("/api/v1.0/movies/<mid>/reviews/<id>",
methods=["DELETE"])
63| @jwt_required
64| @admin_required
65| def delete_review(mid, rid):
66|     movies.update_one(
67|         {"_id": ObjectId(mid)}, {"$pull": {"reviews": {"_id":
ObjectId(rid)}}}
68|     )
69|     return make_response(jsonify({}), 204)
70|
71|
72| @reviews_bp.route("/api/v1.0/movies/reviews/stars", methods=["GET"])
73| def reviews_by_stars():
74|     stars = request.args.get("stars")
75|     stars = str(stars)
76|     results = movies.find()
77|     data_to_return = []
78|
79|     for movie in results:
80|         if "reviews" in movie:
81|             for review in movie["reviews"]:
82|                 if "_id" in review:
83|                     review["_id"] = str(review["_id"])
84|
85|                 if "stars" in review and review["stars"] == stars:
86|                     movie["_id"] = str(movie["_id"])
87|                     data_to_return.append(movie)
88|                     break
89|     if not data_to_return:
90|         return make_response(jsonify({"error": "Invalid Stars
Parameter"}), 404)
91|
92|     return make_response(jsonify(data_to_return), 200)
93|
94|
95| @reviews_bp.route("/api/v1.0/movies/<mid>/reviews/stars",
methods=["GET"])
96| def movie_name_and_stars(mid):
97|     stars = request.args.get("stars")
98|     if not stars:
99|         return jsonify({"error": "Missing stars parameter"}), 400
100|
101|     movie = movies.find_one({"_id": ObjectId(mid)})
102|
103|     if not movie:
104|         return jsonify({"error": "Movie not found"}), 404
105|
106|     if "_id" in movie:
```

```
107|          movie["_id"] = str(movie["_id"])
108|
109|      if "reviews" in movie:
110|          for review in movie["reviews"]:
111|              if "_id" in review:
112|                  review["_id"] = str(review["_id"])
113|          movie["reviews"] = [
114|              review
115|              for review in movie["reviews"]
116|              if "stars" in review and str(review["stars"]) ==
str(stars)
117|          ]
118|
119|      return make_response(jsonify(movie), 200)
120|
121|
122| @reviews_bp.route("/api/v1.0/movies/reviewed_by/<string:username>",
methods=["GET"])
123| def show_movies_reviewed_by_user(username):
124|      pipeline = [{"$match": {"reviews": {"$elemMatch": {"username":
username}}}}]
125|      data_to_return = []
126|      for movie in movies.aggregate(pipeline):
127|          movie["_id"] = str(movie["_id"])
128|          for review in movie["reviews"]:
129|              review["_id"] = str(review["_id"])
130|          data_to_return.append(movie)
131|
132|      if not data_to_return:
133|          return make_response(jsonify({"error": "No reviews for user "
+ username}), 404)
134|
135|      return make_response(jsonify(data_to_return), 200)
136|
```

awards.py

```
1 | from flask import Blueprint, jsonify, make_response, request
2 | import globals
3 | from decorators import jwt_required, admin_required
4 | from bson import ObjectId
5 |
6 | awards_bp = Blueprint("awards_bp", __name__)
7 |
8 | movies = globals.db.movieInfo
9 | awards = globals.db.awards
10|
11|
12| @awards_bp.route("/api/v1.0/awards/all", methods=["GET"])
13| def get_all_awards():
14|     data_to_return = []
15|     for award in awards.find():
16|         award["_id"] = str(award["_id"])
17|         award["movie_id"] = str(award["movie_id"])
18|         data_to_return.append(award)
19|     return make_response(jsonify(data_to_return), 200)
20|
21|
22| @awards_bp.route("/api/v1.0/movies/<movie_id>/awards", methods=["GET"])
23| def get_awards_for_movie(movie_id):
24|     data_to_return = []
25|     for award in awards.find({"movie_id": ObjectId(movie_id)}):
26|         award["_id"] = str(award["_id"])
27|         award["movie_id"] = str(award["movie_id"])
28|         data_to_return.append(award)
29|     if not data_to_return:
30|         return make_response(jsonify({"error": "No awards found for
this movie"}), 404)
31|     return make_response(jsonify(data_to_return), 200)
32|
33|
34| @awards_bp.route("/api/v1.0/awards/<int:year>/winners",
methods=["GET"])
35| def get_award_winners_by_year(year):
36|     data_to_return = []
37|     for award in awards.find({"year": year, "won": True}):
38|         award["_id"] = str(award["_id"])
39|         award["movie_id"] = str(award["movie_id"])
40|         data_to_return.append(award)
41|     if not data_to_return:
42|         return make_response(
43|             jsonify({"error": "No award winners found for year " +
str(year)}), 404
44|         )
45|     return make_response(jsonify(data_to_return), 200)
46|
47|
48| @awards_bp.route("/api/v1.0/awards/add", methods=["POST"])
49| @jwt_required
50| def add_award():
51|     data = request.get_json()
52|     if not data or "title" not in data:
53|         return make_response(jsonify({"error": "Missing form data"}),
404)
54|
55|     movie = movies.find_one({"Title": data["title"]})
56|     if not movie:
```

```
57|          return make_response(jsonify({"error": "Movie not found in
database"}), 400)
58|
59|      new_award = {
60|          "movie_id": movie["_id"],
61|          "award_name": data["award_name"],
62|          "category": data["category"],
63|          "year": data["year"],
64|          "won": data["won"],
65|      }
66|
67|      result = awards.insert_one(new_award)
68|      new_award_link = "http://localhost:5000/api/v1.0/awards/" +
str(result.inserted_id)
69|      return make_response(jsonify({"url": new_award_link}), 201)
70|
71|
72| @awards_bp.route("/api/v1.0/awards/<string:id>", methods=["PUT"])
73| @jwt_required
74| def edit_award(id):
75|      if (
76|          "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):
77|          result = awards.update_one(
78|              {"_id": ObjectId(id)},
79|              {
80|                  "$set": {
81|                      "award_name": request.form["award_name"],
82|                      "category": request.form["category"],
83|                      "year": int(request.form["year"]),
84|                      "won": request.form["won"].lower == "true",
85|                      "movie_id": ObjectId(request.form["movie_id"]),
86|                  }
87|              },
88|          )
89|          if result.matched_count == 1:
90|              edited_award_link =
"http://localhost:5000/api/v1.0/awards/" + id
91|              return make_response(jsonify({"url": edited_award_link}),
200)
92|          else:
93|              return make_response(jsonify({"error": "Invalid award
ID"}), 404)
94|      else:
95|          return make_response(jsonify({"error": "Missing form data"}),
404)
96|
97|
98| @awards_bp.route("/api/v1.0/awards/<string:id>", methods=["DELETE"])
99| @jwt_required
100| @admin_required
101| def delete_award(id):
102|      result = awards.delete_one({"_id": ObjectId(id)})
103|      if result.deleted_count == 1:
104|          return make_response(jsonify({}), 204)
105|      else:
106|          return make_response(jsonify({"error": "Invalid award ID"}),
404)
107|
108|
```

```
@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}},
```

```
61|     ]
62|     results = list(movies.aggregate(pipeline))
63|     return make_response(jsonify(results), 200)
64|
65|
66| @aggregate_bp.route("/api/v1.0/movies/average-review-rating",
methods=["GET"])
67| def average_review_rating():
68|     pipeline = [
69|         {"$unwind": "$reviews"},
70|         {"$addFields": {"numericStars": {"$toDouble":
"$reviews.stars"}}},
71|         {"$group": {"_id": "$Title", "averageReviewRating": {"$avg":
"$numericStars"}}},
72|         {"$project": {"_id": 0, "Title": "$_id", "averageReviewRating":
1}},
73|     ]
74|     results = list(movies.aggregate(pipeline))
75|     return make_response(jsonify(results), 200)
76|
```

auth.py

```
1 | from flask import Blueprint, request, make_response, jsonify
2 | from decorators import jwt_required, admin_required
3 | import globals
4 | import bcrypt
5 | import jwt
6 | import datetime
7 |
8 | auth_bp = Blueprint("auth_bp", __name__)
9 |
10| users = globals.db.Users
11| blacklist = globals.db.blacklist
12|
13| @auth_bp.route('/api/v1.0/logout', methods=["GET"])
14| @jwt_required
15| def logout():
16|     token = request.headers['x-access-token']
17|     blacklist.insert_one({"token":token})
18|     return make_response(jsonify( {'message' : 'Logout successful' } ),
200 )
19|
20| @auth_bp.route('/api/v1.0/login', methods=['GET'])
21| def login():
22|     auth = request.authorization
23|     if auth:
24|         user = users.find_one({'username':auth.username})
25|         if user is not None:
26|             if bcrypt.checkpw(bytes(auth.password, 'UTF-8'),
user["password"]):
27|                 token = jwt.encode({
28|                     'user' : auth.username,
29|                     'admin' : user['admin'],
30|                     'exp' : datetime.datetime.now(datetime.UTC) +
datetime.timedelta(minutes=30)
31|                 }, globals.secret_key, algorithm="HS256")
32|                 return make_response(jsonify({'token': token}), 200)
33|
34|             else:
35|                 return make_response(jsonify({'message' : 'Bad
password'}), 401)
36|         else:
37|             return make_response(jsonify({'message' : 'Bad username'}),
401)
38|     return make_response(jsonify({'message' : 'Authentication
required'}), 401)
39|
```

app.py

```
1 | from flask import Flask
2 | from blueprints.movies.movies import movies_bp
3 | from blueprints.reviews.reviews import reviews_bp
4 | from blueprints.auth.auth import auth_bp
5 | from blueprints.aggregate.aggregate import aggregate_bp
6 | from blueprints.awards.awards import awards_bp
7 |
8 | app = Flask(__name__)
9 | app.register_blueprint(movies_bp)
10| app.register_blueprint(reviews_bp)
11| app.register_blueprint(auth_bp)
12| app.register_blueprint(aggregate_bp)
13| app.register_blueprint(awards_bp)
14|
15| if __name__ == "__main__":
16|     app.run(debug=True)
```

decorators.py

```
1 | from flask import request, jsonify, make_response
2 | from pymongo import MongoClient
3 | import jwt
4 | from functools import wraps
5 | import globals
6 |
7 | client =
MongoClient("mongodb://127.0.0.1:27017/?directConnection=true&serverSelecti
onTimeoutMS=2000&appName=mongosh+2.5.8")
8 | db = client.Movies
9 | movies = db.movieInfo
10| users = db.users
11| blacklist = globals.db.blacklist
12|
13|
14| def jwt_required(func):
15|     @wraps(func)
16|     def jwt_required_wrapper(*args, **kwargs):
17|         token = None
18|         if "x-access-token" in request.headers:
19|             token = request.headers["x-access-token"]
20|         try:
21|             data = jwt.decode(token, globals.secret_key,
algorithms="HS256")
22|         except:
23|             return make_response(jsonify({"message": "Token is
invalid"}), 401)
24|         bl_token = blacklist.find_one({"token": token})
25|         if bl_token is not None:
26|             return make_response(jsonify({"message": "Token has been
cancelled"}), 401)
27|         return func(*args, **kwargs)
28|
29|     return jwt_required_wrapper
30|
31|
32| def admin_required(func):
33|     @wraps(func)
34|     def admin_required_wrapper(*args, **kwargs):
35|         token = request.headers["x-access-token"]
36|         data = jwt.decode(token, globals.secret_key,
algorithms="HS256")
37|         if data["admin"]:
38|             return func(*args, **kwargs)
39|         else:
40|             return make_response(jsonify({"message": "Admin access
required"}), 401)
41|
42|     return admin_required_wrapper
43|
```

globals.py

```
1 | from pymongo import MongoClient
2 |
3 | secret_key = 'movieinfodatabasesecretkey'
4 |
5 | client =
MongoClient("mongodb://127.0.0.1:27017/?directConnection=true&serverSelecti
onTimeoutMS=2000&appName=mongosh+2.5.8")
6 | db = client.Movies
```