

Mikołaj Bul, Michał Skwara, Łukasz Stachoń, Adam Niemiec

Zarys projektu

Głównym założeniem projektu jest stworzenie aplikacji opartej na bazie filmów i osób kina. Dodatkowymi możliwościami będą ocenianie, wystawianie recenzji oraz przeglądanie najnowszych wiadomości ze świata kinematografii. Każdy użytkownik aplikacji będzie mógł sprawdzać swoje indywidualne recenzje, komunikować się z innymi odbiorcami poprzez zostawianie komentarzy dotyczących konkretnych filmów oraz przeglądać aktualności dotyczące filmów/aktorów/nagród itp.

W projekcie planujemy wykorzystać technologie MongoDB oraz Node.js.

Instalacja:

Instrukcja instalacji oraz używania są dostępne w readme na githubie:

<https://github.com/mskwara/MovieHubAPI>

Kolekcje w bazie

Nasza baza składa się z 7 kolekcji, a każda z nich zawiera dokumenty z danymi odpowiednimi dla każdego elementu bazy.

MovieHubDB						
DATABASE SIZE: 5.8MB		INDEX SIZE: 340KB	TOTAL COLLECTIONS: 7		CREATE COLLECTION	
Collection Name	Documents	Documents Size	Documents Avg	Indexes	Index Size	Index Avg
awards	200	22.51KB	116B	1	24KB	24KB
comments	3000	1.32MB	461B	1	60KB	60KB
moviepeople	1000	983.78KB	1008B	1	28KB	28KB
movies	1000	1022.34KB	1.02KB	1	56KB	56KB
news	401	1.39MB	3.54KB	1	36KB	36KB
reviews	2000	946.58KB	485B	1	36KB	36KB
users	501	194.94KB	399B	2	100KB	50KB

Opis kolekcji

Award:

Jest kolekcją która przechowuje informacje o nagrodach przyznawanych filmom lub ludziom związanych z branżą filmową. Każda z nagród ma swoją nazwę i datę przyznania.

```
const awardSchema = new mongoose.Schema({
  type: {
    type: String,
    enum: ["Oscar", "Golden Globe", "Golden Raspberry", "Emmy Award"],
  },
  date: {
    type: Date,
    required: [true, "A award must have a date."],
  },
  moviePerson: {
    type: mongoose.Schema.ObjectId,
    ref: "MoviePerson",
  },
  movie: {
    type: mongoose.Schema.ObjectId,
    ref: "Movie",
  },
});
```

Comment:

Jest to kolekcja która przechowuje komentarze pisane przez użytkowników na temat danego filmu.

```
const commentSchema = new mongoose.Schema({
  content: {
    type: String,
    required: [true, "You can't post an empty comment"],
  },
  date: {
    type: Date,
    default: Date.now,
  },
  userID: {
    type: mongoose.Schema.ObjectId,
    ref: "User",
  },
  movieID: {
    type: mongoose.Schema.ObjectId,
    ref: "Movie",
  },
});
```

Movie:

Kolekcja ta przechowuje informacje o filmach z naszej bazy danych. Zawiera takie informacje jak tytuł, opis, gatunek, obsadę oraz średnią ocen użytkowników.

```
const movieSchema = new mongoose.Schema({
  title: {
    type: String,
    required: [true, "A movie must have a title."],
  },
  description: String,
  genre: {
    type: String,
    enum: ["family", "comedy", "religious", "action", "sci-fi"],
  },
  date: {
    type: Date,
    required: [true, "A movie must have a date."],
    ref: "date",
  },
  actors: [
    {
      type: mongoose.Schema.ObjectId,
      ref: "MoviePerson",
    },
  ],
  ratingAverage: {
    type: Number,
    min: 1,
    max: 5,
    default: 4.5,
    set: (val) => Math.round(val * 10) / 10,
  },
  ratingQuantity: {
    type: Number,
    min: 0,
    default: 0,
  },
});
```

MoviePerson

Kolekcja przechowuje informacje o ludziach związanych z branżą filmową. Przechowuje informacje o imieniu, dacie urodzin, zawodzie oraz filmach przy którym dana osoba pracowała.

```
const moviePersonSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, "A person must have a name."],
  },
  birthdate: {
    type: Date,
    required: [true, "A person must have a birthdate."],
    ref: "birthdate",
  },
  role: {
    type: String,
    enum: [
      "actor",
      "director",
      "screenwriter",
      "camera operator",
      "composer",
      "makeup artist",
    ],
    required: [true, "A person must have a role."],
  },
  description: {
    type: String,
    required: [true, "A person must have a description."],
  },
  movies: [
    {
      type: mongoose.Schema.ObjectId,
      ref: "Movie",
    },
  ],
});
```

News

Kolekcja zawiera “newsy” ze świata filmu. Każdy dokument ma swoją zawartość oraz użytkownika, będącego autorem wpisu.

```
const newsSchema = new mongoose.Schema({
  content : {
    type: String,
    required: [true, "News cannot be empty."],
  },
  userID: {
    type: mongoose.Schema.ObjectId,
    ref: "User"
  },
  date: {
    type: Date,
    default: Date.now
  }
});
```

Review

Kolekcja przechowująca informacje o ocenach filmów. Każdy z użytkowników może wystawić danemu filmowi ocenę oraz napisać recenzję.

```
const reviewSchema = new mongoose.Schema({
  content: {
    type: String,
  },
  rating: {
    type: Number,
    min: 1,
    max: 5,
  },
  movieID: {
    type: mongoose.Schema.ObjectId,
    ref: "Movie",
  },
  userID: {
    type: mongoose.Schema.ObjectId,
    ref: "User",
  },
  date: {
    type: Date,
    default: Date.now,
  },
});
```

User

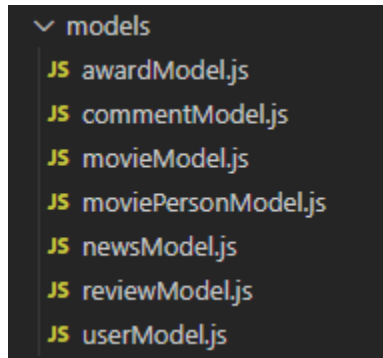
Przechowujemy informacje o użytkownikach portalu oraz o rolach jakie pełnią.

```
const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: [true, "User must have username."],
    unique: [true, "This username already exists."]
  },
  description: {
    type: String,
  },
  password: {
    type: String,
    required: [true, "User must have a password"],
    select: false,
  },
  passwordChangedAt: Date,
  role: {
    type: String,
    enum: ["admin", "user"],
    default: "user",
    select: false,
  },
});
```

Struktura projektu

Dostęp do poszczególnych funkcji systemu jest realizowany poprzez **API** stworzone w języku **Javascript**, używając biblioteki mongoose do zarządzania bazą danych **MongoDB**. Komunikacja z bazą danych odbywa się poprzez requesty do serwera.

Modele



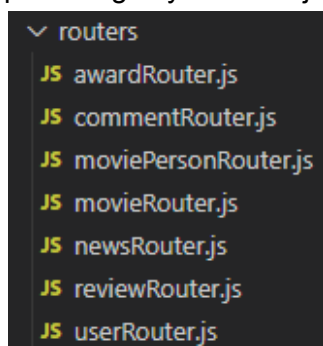
W folderze models znajdują się schematy każdej kolekcji, zawartość poszczególnych plików została opisana w poprzednim punkcie.

Routery

```
15 app.use("/awards", awardRouter);
16 app.use("/movies", movieRouter);
17 app.use("/moviepersons", moviePersonRouter);
18 app.use("/users", userRouter);
19 app.use("/comments", commentRouter);
20 app.use("/reviews", reviewRouter);
21 app.use("/news", newsRouter);
```

W pliku app.js wyszczególniamy, który Router będzie odpowiedzialny za które requesty. Przykładowo requesty na endpoint rozpoczynający się od **/awards** zostaną obsłużone przez **awardRouter**. Analogicznie jest z requestami rozpoczynającymi się na **/movies**, **/moviepersons** itd.

W folderze routers znajdują się pliki zawierające szczegółowe endpointy dotyczące poszczególnych kolekcji.



Przykładowa struktura tych plików wygląda następująco:

```
1  const express = require("express");
2  const awardController = require("../controllers/awardController");
3  const authController = require("../controllers/authController");
4
5  const router = express.Router();
6
7  router
8    .route("/")
9    .get(awardController.getAllAwards)
10   .post(
11     authController.protect,
12     authController.restrictTo("admin"),
13     awardController.createAward
14   );
15
16  router
17    .route("/:awardID")
18    .get(awardController.getAward)
19   .patch(
20     authController.protect,
21     authController.restrictTo("admin"),
22     awardController.updateAward
23   )
24   .delete(
25     authController.protect,
26     authController.restrictTo("admin"),
27     awardController.deleteAward
28   );
29
30  router.route("/awardName/:name").get(awardController.getAwardByName);
31  router.route("/between/:begin/:end").get(awardController.getAwardsInPeriod);
32
33  module.exports = router;
```

Przykładowo request “localhost:8000/awards/606f1efc8b0556360c8bbf37” metodą **GET** wywoła funkcję **getAward** dostępną w **awardController**, zaś metodą **PATCH** wywoła po kolei 3 funkcje **protect**, **restrictTo** oraz na końcu **updateAward**, jeżeli poprzednie nie zgłoszą błędu, np. z powodu tego że użytkownik wykonujący tego requesta nie ma przypisanej roli “admin”.

Kontrolery

```
▼ controllers
JS  authController.js
JS  awardController.js
JS  commentController.js
JS  movieController.js
JS  moviePersonController.js
JS  newsController.js
JS  reviewController.js
JS  userController.js
```

Funkcje obsługi requestów zostały podzielone na osobne pliki w zależności, z którą kolekcją są związane. Dodatkowo występuje **authController** zawierający funkcje dotyczące autentykacji użytkowników i funkcje związane z bezpieczeństwem.

app.js

W tym pliku definiujemy routery, oraz określamy który z nich będzie odpowiedzialny za które requesty.

```
const movieRouter = require("../routers/movieRouter");
const userRouter = require("../routers/userRouter");
const awardRouter = require("../routers/awardRouter");
const reviewRouter = require("../routers/reviewRouter");
const newsRouter = require("../routers/newsRouter");
const moviePersonRouter = require("../routers/moviePersonRouter");
const commentRouter = require("../routers/commentRouter");
```

```
app.use("/awards", awardRouter);
app.use("/movies", movieRouter);
app.use("/moviepersons", moviePersonRouter);

app.use("/users", userRouter);
app.use("/comments", commentRouter);

app.use("/reviews", reviewRouter);
app.use("/news", newsRouter);
```

server.js

Za pomocą tego pliku stawiamy lokalny serwer którym łączymy się z bazą danych, co umożliwia wykonywanie requestów.

```
const app = require("../app");
mongoose
  .connect(process.env.DATABASE, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useCreateIndex: true,
    useFindAndModify: false,
  })
  .then(() => {
    console.log("DB connection established!");
  });

const port = 8000;
const server = app.listen(port, () => {
  console.log(`App running on port ${port}...`);
});
```

generator.js

Plik zawierający skrypt generujący dokumenty, które następnie umieszczane są w odpowiedniej kolekcji.

Przykładowe generowanie dokumentów z kolekcji Movie:

```
const generateAwards = async () => {
  console.log("Generating awards...");
  const data = [];
  const awardNames = [
    "Oscar",
    "Golden Globe",
    "Golden Raspberry",
    "Emmy Award",
  ];

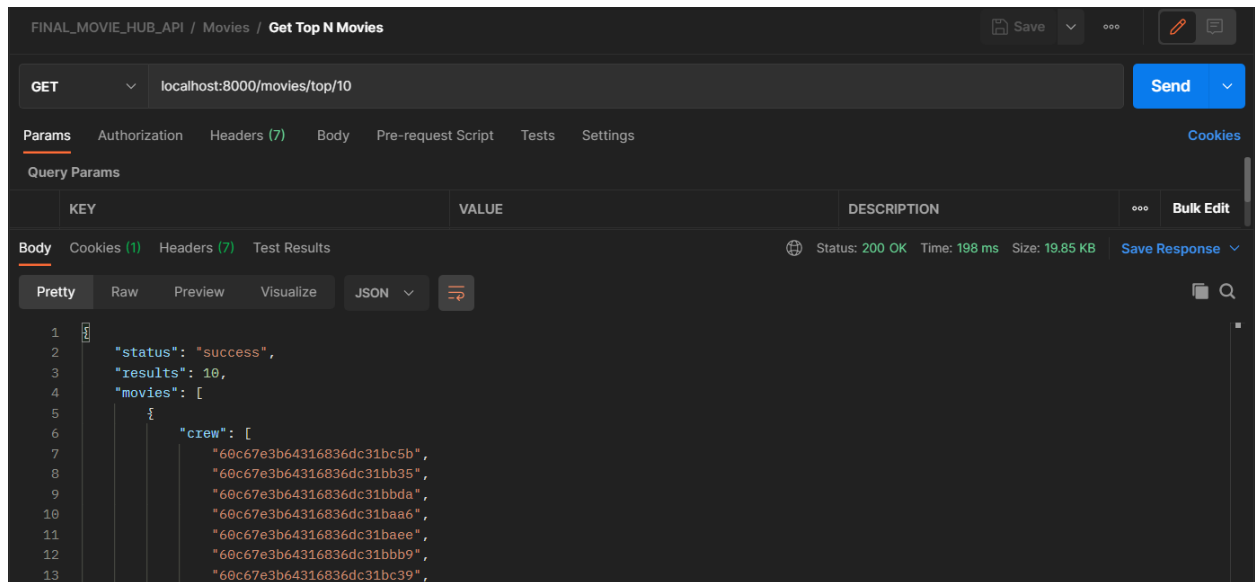
  const moviePersons = await MoviePerson.find();
  const movies = await Movie.find();
  for (let i = 0; i < AWARDS_COUNT; i++) {
    data.push({
      awardName: awardNames[random(0, awardNames.length)],
      moviePerson: moviePersons[random(0, moviePersons.length)]._id,
      movie: movies[random(0, movies.length)]._id,
      date: randomDate(new Date(1979, 0, 1), new Date()),
    });
  }
  await Award.create(data);
  console.log("Awards generating finished!\n");
};
```

Wykonywanie requestów za pomocą Postmana

Po zaimportowaniu pliku requests.json i uruchomieniu serwera jesteśmy w stanie wykonać wszystkie requesty istniejące w bazie. Każda z kolekcji ma zapewnione operacje CRUD-owe, ale stworzyliśmy także inne możliwe wywołania. Omówimy teraz wybrane z nich:

Get Top N Movies

Otrzymujemy wybraną liczbę filmów o najwyższych ocenach w rankingu.



FINAL_MOVIE_HUB_API / Movies / **Get Top N Movies**

GET localhost:8000/movies/top/10 **Send**

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

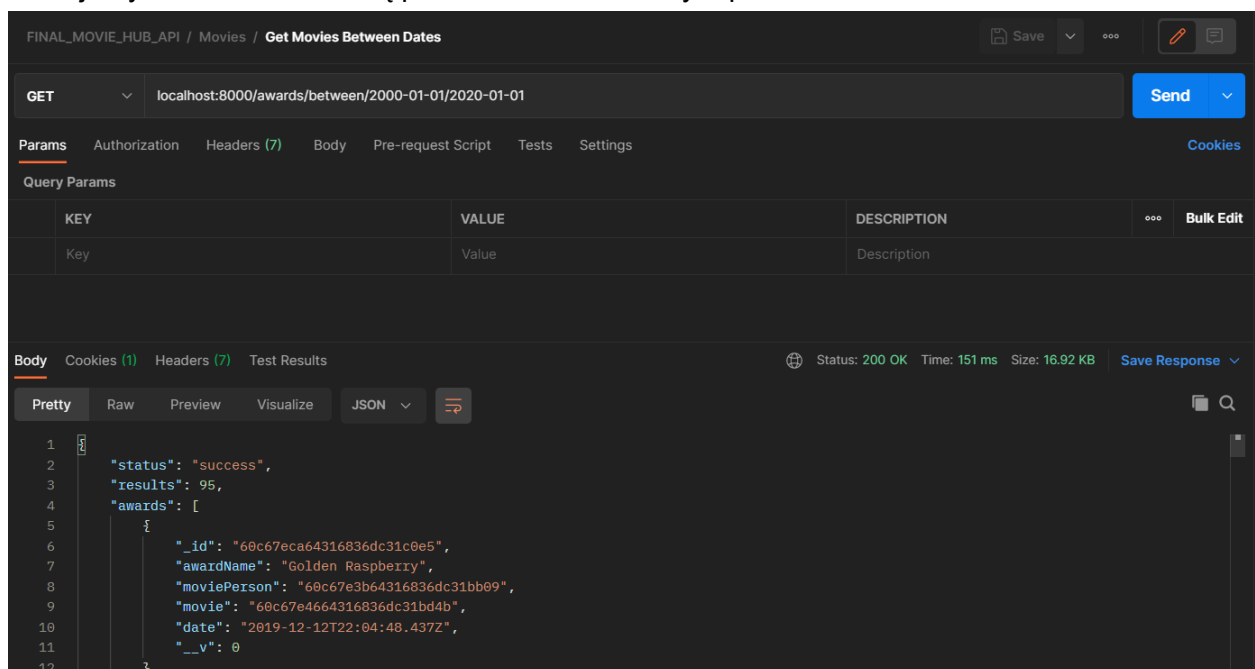
Body Cookies (1) Headers (7) Test Results **Status: 200 OK** Time: 198 ms Size: 19.85 KB **Save Response**

Pretty Raw Preview Visualize JSON **Send**

```
1 {
2   "status": "success",
3   "results": 10,
4   "movies": [
5     {
6       "crew": [
7         "60c67e3b64316836dc31bc5b",
8         "60c67e3b64316836dc31bb35",
9         "60c67e3b64316836dc31bbda",
10        "60c67e3b64316836dc31baa6",
11        "60c67e3b64316836dc31baee",
12        "60c67e3b64316836dc31bbb9",
13        "60c67e3b64316836dc31bc39",
```

Get Movie Between Dates

Dostajemy zbiór filmów z datą powstania w określonym przedziale czasu.



FINAL_MOVIE_HUB_API / Movies / **Get Movies Between Dates**

GET localhost:8000/awards/between/2000-01-01/2020-01-01 **Send**

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies (1) Headers (7) Test Results **Status: 200 OK** Time: 151 ms Size: 16.92 KB **Save Response**

Pretty Raw Preview Visualize JSON **Send**

```
1 {
2   "status": "success",
3   "results": 95,
4   "awards": [
5     {
6       "_id": "60c67eca64316836dc31c0e5",
7       "awardName": "Golden Raspberry",
8       "moviePerson": "60c67e3b64316836dc31bb09",
9       "movie": "60c67e4664316836dc31bd4b",
10      "date": "2019-12-12T22:04:48.437Z",
11      "__v": 0
12    },
```

Get User Reviews

Otrzymujemy wszystkie recenzja wystawione przez danego użytkownika

```
1 {
2   "status": "success",
3   "results": 6,
4   "reviews": [
5     {
6       "_id": "60c62f880769ba7f27e27dcf",
7       "content": "sanki komputer morze wakacje dywan wakacje tygrys sanki ukulele biurko kwiat komputer kwiat dywan komputer tygrys
8         maipa stajnia ukulele studia ogród ogród maipa ukulele ogród król sanki bałwan maipa kwiat studia tygrys król balkon morze
9         komputer stajnia król morze kwiat dywan dywan sanki balkon maipa",
10      "rating": 4,
11      "movieID": "60c62f820769ba7f27e2757b",
12      "userID": "60c62f870769ba7f27e27b87",
13      "date": "2021-06-13T16:17:12.747Z",
14      "__v": 0
15    },
16    {
17      "_id": "60c62f880769ba7f27e27fa5",
18      "content": "dywan studia studia biurko stajnia",
19      "rating": 5,
20      "movieID": "60c62f820769ba7f27e2766e",
21      "userID": "60c62f870769ba7f27e27b87",
22      "date": "2021-06-13T16:17:12.809Z",
23      "__v": 0
24    },
25    {
26      "_id": "60c62f880769ba7f27e27ff5",
27      "content": "komputer kamera komputer dywan ukulele biurko ukulele król biurko balkon morze balkon studia studia kamera sanki
28      ukulele król komputer biurko bałwan komputer komputer maipa kamera wakacje morze dywan tygrys dywan ogród balkon ogród maipa
29      bałwan biurko ukulele król ogród maipa bałwan kamera wakacje balkon komputer balkon balkon tygrys król stajnia morze ukulele
30      dywan komputer dywan wakacje balkon maipa ukulele kamera tygrys wakacje kwiat komputer bałwan ogród stajnia maipa król ukulele
31      król komputer morze komputer dywan balkon ogród wakacje kwiat biurko maipa kamera król studia kamera kwiat",
32      "rating": 1,
33      "movieID": "60c62f820769ba7f27e27486",
34      "userID": "60c62f870769ba7f27e27b87",
35      "date": "2021-06-13T16:17:12.816Z",
36      "__v": 0
37    }
38  ]
39 }
```

Get Person Movies

Otrzymujemy wszystkie filmy nad którymi pracowała dana osoba.

```
1 {
2   "status": "success",
3   "results": 1,
4   "movies": [
5     "5b6f626a656374205365745d"
6   ]
7 }
```

Get Persons With Todays Birthday

Otrzymujemy listę osób, które mają w dzisiejszym dniu urodziny

FINAL_MOVIE_HUB_API / MoviePersons / Get Persons With Todays Birthday

GET localhost:8000/moviepersons/todaysBirthday

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 121 ms Size: 4.43 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": "success",
3   "results": 7,
4   "persons": [
5     {
6       "_id": "60c4d9b42c17c559f8235d76",
7       "name": "morze",
8       "birthdate": "2016-06-13T21:14:40.669Z",
9       "role": "screenwriter",
10      "description": "balkon morze małpa wakacje małpa studia ukulele sanki biurko kwiat ukulele ogród sanki sanki dywan ukulele kwiat
balkon bałwan balkon sanki balkon wakacje tygrys studia studia biurko małpa bałwan bałwan wakacje kwiat sanki ogród komputer
małpa małpa ukulele komputer sanki stajnia biurko ukulele małpa król ukulele balkon biurko tygrys ukulele wakacje balkon
studia ogród wakacje ukulele tygrys bałwan ogród małpa stajnia komputer wakacje bałwan kamera dywan wakacje król stajnia
tygrys kwiat ukulele małpa bałwan sanki ukulele król kwiat tygrys małpa tygrys bałwan ukulele sanki komputer studia wakacje
ogród król balkon małpa sanki bałwan tygrys tygrys ukulele ogród biurko król sanki sanki",
11      "movies": [
12        "5b6f626a656374205365745d"
```

Get Users With At Least N Comments

Zwraca użytkowników z minimalną liczbą N komentarzy.

FINAL_MOVIE_HUB_API / Users / Get Users With At Least N Comments

GET http://localhost:8000/users/minReviews/6

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Body Cookies (1) Headers (7) Test Results

Status: 200 OK Time: 40 ms Size: 6.23 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": "success",
3   "results": 106,
4   "users": [
5     {
6       "_id": "60c62f870769ba7f27e27b87",
7       "username": "Y0lojCYxJ"
8     },
9     {
10      "_id": "60c62f870769ba7f27e27b93",
11      "username": "0BTK8h"
12    },
13    {
14      "_id": "60c62f870769ba7f27e27b96",
15      "username": "V7eBxfu"
16    },
17    {
18      "_id": "60c62f870769ba7f27e27b9b",
19      "username": "o50LdtkucN9"
20    },
21    {
22      "_id": "60c62f870769ba7f27e27ba1",
23      "username": "tdlALLBA"
24    },
25    {
```