

ROTTEN CARROTS

Projekt Bazy Danych 2023

Autorzy:

Adam Górka 406346(adamgorka@student.agh.edu.pl)
Kacper Augustyn 411084 (kacperau@student.agh.edu.pl)
Wiktor Wilkusz 411605 (wiktorwilk@student.agh.edu.pl)

Spis treści

1. Temat projektu
2. Architektura systemu
 - 2.1. Baza danych
 - 2.2. Framework
3. Funkcje systemu
 - 3.1. Endpointy
 - 3.1.1. /auctions/all
 - 3.1.2. /auctions/active
 - 3.1.3. /auctions/{auctionID}
 - 3.1.4. /auctions/game/{gameID}
 - 3.1.5. /auctions/price?minPrice={x}&maxPrice={y}
 - 3.1.6. /auctions/buy?auctionid={x}&userid={y}
 - 3.1.7. /auctions/game/genre/{genre}
 - 3.1.8. /auctions/{auctionID}
 - 3.1.9. /games/all
 - 3.1.10. /games/{gameID}
 - 3.1.11. /games/genre/{genre}
 - [3.1.12. /games/sortByTitle](#)
 - [3.1.13. /games/byTitle/{title}](#)
 - [3.1.14. /games/](#)
 - [3.1.15. /games/](#)
 - 3.1.16. /news/all
 - 3.1.17. /news/{newsID}
 - 3.1.18. /news/newestNews
 - 3.1.19. /news/newestNews/{limit}
 - 3.1.20. /news/newsByAuthor?author={x}
 - 3.1.21. /news/newsByAuthorFirstLetters?letters={x}
 - 3.1.22. /news/{from_}/{to_}
 - 3.1.23. /news/byYear?year={x}
 - 3.1.24. /news/byYearAndMonth?year={x}&?month={y}
 - 3.1.25. /news/{author}/{from_}/{to_}
 - 3.1.26. /news/newsByTitle?title={x}
 - 3.1.27. /news/sortByAuthor
 - 3.1.28. /news/newsByAuthorAndContent?content={x}&?author={y}
 - 3.1.29. /news/newsByContent?content={x}
 - 3.1.30. /reviews/all
 - 3.1.31. /reviews/{reviewID}
 - 3.1.32. /reviews/reviewsByCarrotRate
 - 3.1.33. /reviews/top
 - 3.1.34. /reviews/reviewsByContent

3.1.35. /reviews/reviewsByAuthor

3.1.36. /reviews/newest

3.2. Triggery

3.2.1. UpdateGameAndUserOnReviewInsert

3.2.2. ValidateNickname

3.2.3. UpdateUserOnAuctionInsert

3.2.4. UpdateAuctionStatusInUser

4. Podsumowanie

1. Temat projektu

Projekt dotyczy systemu bazodanowego dla serwisu z newsami, opiniami oraz kalendarzem premier gier komputerowych. Celem systemu jest umożliwienie użytkownikom dostępu do najnowszych informacji związanych z grami komputerowymi, możliwość przeglądania opinii innych użytkowników, sprawdzanie dat premiery gier oraz dostęp do funkcjonalności zakupu i sprzedaży gier na rynku.

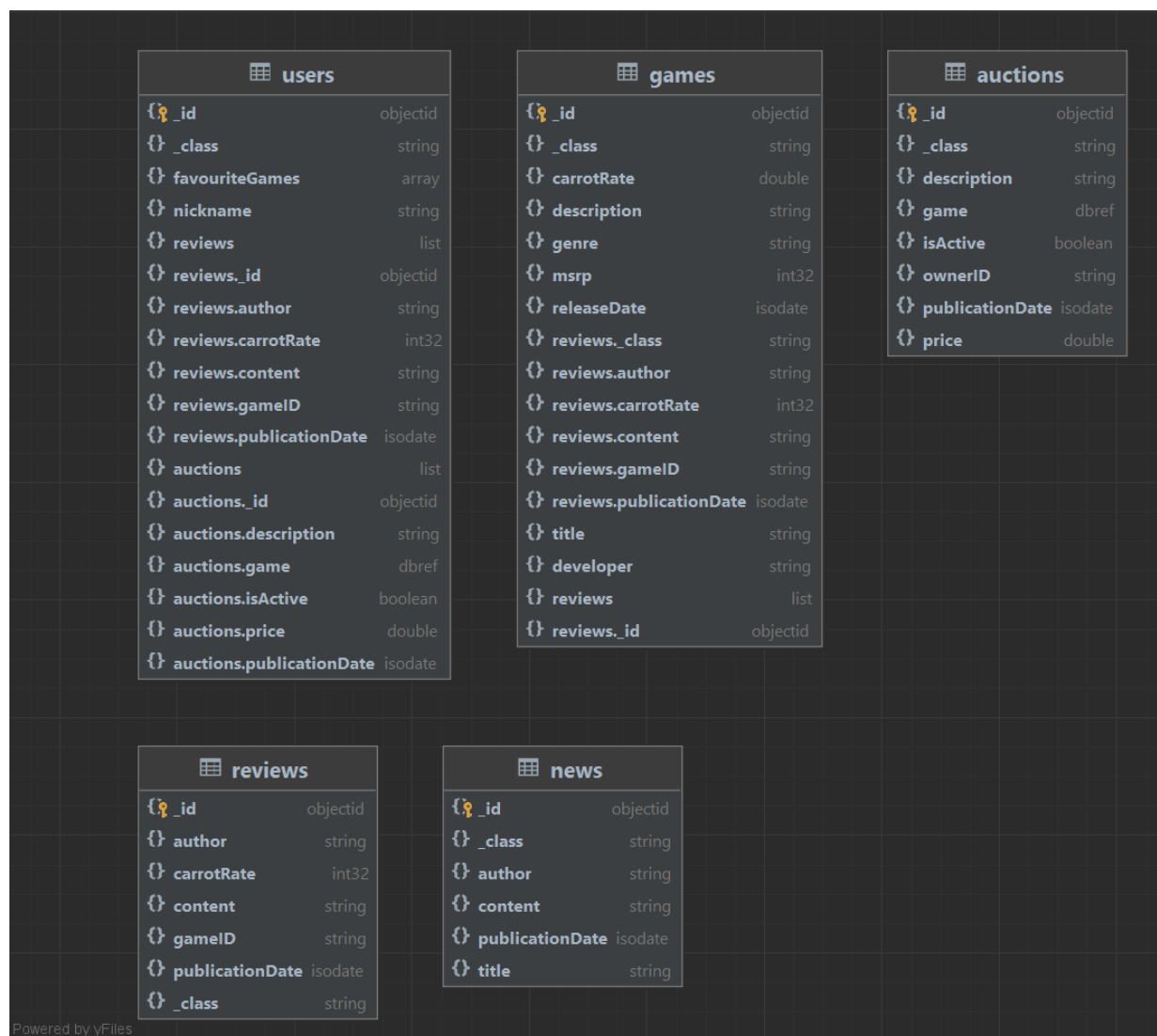
2. Architektura systemu

2.1. Baza danych

W projekcie została wykorzystana baza danych **MongoDB**. MongoDB jest systemem bazodanowym typu NoSQL, który oferuje elastyczną strukturę dokumentową, umożliwiającą przechowywanie i indeksowanie danych w formacie dokumentów BSON (Binary JSON).

Baza danych została zahostowana na chmurowy serwisie **MongoDB Atlas**.

Kolekcje zawarte w bazie jak i budowa poszczególnych dokumentów są widoczne poniżej.



users	
_id	objectId
_class	string
favouriteGames	array
nickname	string
reviews	list
reviews_id	objectId
reviews.author	string
reviews.carrotRate	int32
reviews.content	string
reviews.gameID	string
reviews.publicationDate	isodate
auctions	list
auctions_id	objectId
auctions.description	string
auctions.game	dbref
auctions.isActive	boolean
auctions.price	double
auctions.publicationDate	isodate

games	
_id	objectId
_class	string
carrotRate	double
description	string
genre	string
msrp	int32
releaseDate	isodate
reviews_class	string
reviews.author	string
reviews.carrotRate	int32
reviews.content	string
reviews.gameID	string
reviews.publicationDate	isodate
title	string
developer	string
reviews	list
reviews_id	objectId

auctions	
_id	objectId
_class	string
description	string
game	dbref
isActive	boolean
ownerID	string
publicationDate	isodate
price	double

reviews	
_id	objectId
author	string
carrotRate	int32
content	string
gameID	string
publicationDate	isodate
_class	string

news	
_id	objectId
_class	string
author	string
content	string
publicationDate	isodate
title	string

Dane do bazy danych zostały wygenerowane za pomocą <https://app.json-generator.com> . Wykorzystany w tym celu kod znajduje się w pliku **data_generator.js** .

Przykłady poszczególnych dokumentów:

Auctions:

```
_id: ObjectId('648da202b75f480affc80349')
game: DBRef('games', '648d71a3020f97c10309f2c3')
description: "commodo quis ad veniam voluptate ea ipsum proident fugiat laboris"
price: 19.51
publicationDate: 2022-08-21T16:12:10.994+00:00
isActive: true
_class: "com.rotten.carrots.Auctions.Auction"
ownerID: "648d724e020f97c10309f31c"
```

Games:

```
_id: ObjectId('648d71a3020f97c10309f2bd')
▼ reviews: Array
  ▼ 0: Object
    _id: ObjectId('648d7d0a020f97c10309f95c')
    _class: "com.rotten.carrots.Review.Review"
    author: "Pacheco6"
    gameID: "648d71a3020f97c10309f2bd"
    content: "tempor pariatur esse exercitation laboris aliquip laboris aliquip fugi..."
    carrotRate: 47
    publicationDate: 2018-12-26T14:00:25.670+00:00
  ▶ 1: Object
  ▶ 2: Object
  ▶ 3: Object
  ▶ 4: Object
title: "id enim irure"
genre: "Board Game"
description: "id culpa cupidatat consectetur irure dolore sit proident ut cillum nis..."
developer: "Microsoft"
releaseDate: 2021-12-30T22:05:08.509+00:00
msrp: 49
carrotRate: 32.4
_class: "com.rotten.carrots.Game.Game"
```

News:

```
_id: ObjectId('648cd663020f97c10309f082')
author: "Luz9"
title: "qui dui ea nisi"
content: "incididunt sint consequat enim eu sint ex ex mollit laboris dolore non..."
publicationDate: 2020-04-01T17:25:10.347+00:00
_class: "com.rotten.carrots.News.News"
```

Reviews:

```
_id: ObjectId('648d7d0a020f97c10309f951')
_class: "com.rotten.carrots.Review.Review"
author: "Concepcion6"
gameID: "648d71a3020f97c10309f319"
content: "culpa tempor ad ex ipsum velit laborum sit consequat minim"
carrotRate: 31
publicationDate: 2019-09-21T00:42:51.107+00:00
```

Users:

```
_id: ObjectId('648d724e020f97c10309f31c')
nickname: "Goldie4"
▼ reviews: Array
  ▼ 0: Object
    _id: ObjectId('648d7d0a020f97c10309f975')
    gameId: "648d71a3020f97c10309f2dc"
    author: "Goldie4"
    content: "tempor irure proident ea occaecat cupidatat sint veniam nisi in"
    carrotRate: 7
    publicationDate: 2022-02-04T16:56:54.933+00:00
    ▸ 1: Object
    ▸ 2: Object
    ▸ 3: Object
    ▸ 4: Object
    ▸ 5: Object
    ▸ 6: Object
  ▼ auctions: Array
    ▼ 0: Object
      _id: ObjectId('648d7435020f97c10309f345')
      game: DBRef('games', '648d71a3020f97c10309f2c3')
      description: "commodo quis ad veniam voluptate ea ipsum proident fugiat laboris"
      price: 19.51
      publicationDate: 2022-08-21T14:12:10.994+00:00
      isActive: true
      ▸ 1: Object
      ▸ 2: Object
      ▸ 3: Object
      ▸ 4: Object
      ▸ 5: Object
      ▸ 6: Object
      ▸ 7: Object
    ▼ favouriteGames: Array
      0: DBRef('games', '648d71a3020f97c10309f311')
      1: DBRef('games', '648d71a3020f97c10309f317')
      2: DBRef('games', '648d71a3020f97c10309f2b9')
      3: DBRef('games', '648d71a3020f97c10309f2d2')
      4: DBRef('games', '648d71a3020f97c10309f2de')
      5: DBRef('games', '648d71a3020f97c10309f2ed')
      6: DBRef('games', '648d71a3020f97c10309f2c9')
    _class: "com.rotten.carrots.User.User"
```


2.2. Framework

W projekcie wykorzystano **Java Spring** jako framework do budowy aplikacji backendowej. Zastosowana wersja Javy to **Java 17**. Java Spring oferuje wiele funkcji i narzędzi, które przyspieszają proces tworzenia aplikacji webowych oraz ułatwiają zarządzanie komponentami.

Projektu został zorganizowany zgodnie z konwencją Spring, gdzie główne komponenty aplikacji będą podzielone na pakiety, takie jak controllers, services, models i repositories. Kontrolery będą obsługiwać żądania HTTP i definiować endpointy API. Usługi będą zawierać logikę biznesową, taką jak dodawanie i usuwanie newsów, obsługę opinii, zarządzanie kalendarzem premier oraz funkcjonalności zakupu i sprzedaży gier. Modele będą reprezentować dane w aplikacji, takie jak newsy, opinie, gry itp. Repozytoria będą odpowiedzialne za dostęp do bazy danych i wykonywanie operacji CRUD na danych.

3. Funkcje systemu

System oferuje następujące funkcje:

- Zarządzanie newsami: Umożliwia dodawanie, edytowanie i usuwanie newsów związanych z grami komputerowymi. Każdy news będzie zawierał tytuł, treść, datę publikacji oraz informacje o autorze.
- Opinie użytkowników: Użytkownicy będą mogli dodawać opinie na temat konkretnych gier komputerowych. Opinie będą zawierać ocenę, treść, datę dodania oraz informacje o autorze.

- Kalendarz premier: System umożliwi przeglądanie dat premiery gier komputerowych. Każda gra będzie powiązana z datą premiery oraz posiadać informacje o gatunku, producencie i innych szczegółach.
- Zakup i sprzedaż gier: Użytkownicy będą mieć możliwość zakupu gier wystawionych przez innych użytkowników oraz wystawiania swoich gier na sprzedaż. Informacje o dostępnych grach będą zawierać tytuł, opis, cenę oraz stan dostępności.
- Wyszukiwanie i filtrowanie: Użytkownicy będą mogli przeszukiwać i filtrować gry, newsy i opinie na podstawie różnych kryteriów, takich jak tytuł, gatunek, ocena itp.

3.1. Endpointy

3.1.1. /auctions/all

wszystkie aukcje

```
@GetMapping("/all")
public List<Auction> getAll() {
    return auctionService.getAll();
}
```

3.1.2. /auctions/active

aktywne aukcje

```
@GetMapping("/active")
public List<Auction> getActive() {
    return auctionService.getActive();
}
```

```
@Query("{ 'isActive': true }")
List<Auction> findActive();
```

3.1.3. /auctions/{auctionID}

aukcja o danym ID

```
@GetMapping("/{auctionID}")
public Optional<Auction> getAuctionByID(@PathVariable("auctionID") String id) {
    return auctionService.getAuctionByID(id);
}
```

3.1.4. /auctions/game/{gameID}

aukcje dotyczące danej gry

```
@GetMapping("/game/{gameID}")
public List<Auction> getAuctionsByGame(@PathVariable("gameID") String gameId) {
    Optional<Game> game = gameService.getGameByID(gameId);
    return game.map(auctionService::getAuctionsByGame).orElse(null);
}
```

3.1.5. /auctions/price?minPrice={x}&maxPrice={y}

aukcje z danego przedziału cenowego

```
@GetMapping("/price")
public List<Auction> getAuctionsBetweenPrices(
    @RequestParam("minPrice") double minPrice,
    @RequestParam("maxPrice") double maxPrice) {
    return auctionService.getByPriceBetween(minPrice, maxPrice);
}
```

```
@Query("{ 'price': { $gte: ?0, $lte: ?1 } }")
List<Auction> findByPrices(double minPrice, double maxPrice);
```

3.1.6. /auctions/buy?auctionid={x}&userid={y}

kupowanie aukcji o danym ID x, przez użytkownika o ID y

```
@PostMapping("/buy")
public String buyAuction(
    @RequestParam("auctionid") String auctionID,
    @RequestParam("userid") String userID) {
    boolean success = auctionService.purchaseById(auctionID, userID);
    if (success) return "Purchased";
    return "Failed to Purchase";
}
```

3.1.7. /auctions/game/genre/{genre}

wszystkie aukcje gier z danego gatunku

```
@GetMapping("/game/genre/{genre}")
public List<Auction> getAuctionByGameGenre(@PathVariable("genre") String genre){
    List<Game> games = gameService.getGameByGenre(genre);
    List<Auction> res = new ArrayList<>();
    for(Game game : games){
        res.addAll(auctionService.getAuctionsByGame(game));
    }
    return res;
}
```

```
@Aggregation(pipeline = {
    "{ $lookup: { from: 'games', localField: 'game', foreignField: '_id', as: 'game' } }",
    "{ $match: { 'game.$genre': ?0 } }"
})
List<Auction> findByGameGenre(String genre);
```

3.1.8. /auctions/{auctionID}

usuwanie aukcji o danym ID

```
@DeleteMapping("/{auctionID}")
public ResponseEntity<?> deleteAuction(@PathVariable("auctionID") String id) {
    return auctionService.deleteAuctionByID(id);
}
```

3.1.9. /games/all

wszystkie gry

```
@GetMapping("/all")
public List<Game> getAll() {
    return this.gameService.getAllGames();
}
```

3.1.10. /games/{gameID}

gra o danym ID

```
@GetMapping("/{gameID}")
public Optional<Game> getGameByID(@PathVariable("gameID") String gameID) {
    return this.gameService.getGameByID(gameID);
}
```

3.1.11. /games/genre/{genre}

wszystkie gry danego gatunku

```
@GetMapping("/genre/{genre}")
public List<Game> getGameByGenre(@PathVariable("genre") String genre) {
    return this.gameService.getGameByGenre(genre);
}
```

```
@Query("{ 'genre': { $eq: ?0 } }")
List<Game> findByGenre(String genre);
```

3.1.12. /games/sortByTitle

```
@GetMapping("/sortByTitle")
public List<Game> getGamesByTitle() {
    return this.gameService.getGamesSortedByTitle();
}
```

3.1.13. /games/byTitle/{title}

3.1.14. /games/

3.1.15. /games/

3.1.16. /news/all

wszystkie newsy

```
@GetMapping("/all")
public List<News> getAll(){ return this.newsService.getAllNews(); }
```

3.1.17. /news/{newsID}

news o podanym ID

```
@GetMapping("/{newsID}")
public News getNewsByID(@PathVariable("newsID") String newsID){
    return newsService.getNewsByID(newsID);
}
```

3.1.18. /news/newestNews

10 najnowszych newsów

```
@GetMapping("/newestNews")
public List<News> getNewestNews(){
    return newsService.getNewestNews();
}
```

```
List<News> findTop10ByOrderByPublicationDateDesc();
```

3.1.19. /news/newestNews/{limit}

wypisanie danej ilości najnowszych newsów

```
@GetMapping("newestNews/{limit}")
public List<News> getNewestByParam(@PathVariable("limit") int howMuch){
    return newsService.getLatestNews(howMuch);
}
```

```
@Aggregation(value = {
    "{ '$sort' : { 'publicationDate' : -1 } }",
    "{ '$limit' : ?0 }"
})
List<News> findLatestNews(int limit);
```

3.1.20. /news/newsByAuthor?author={x}

newsy od danego autora

```
@GetMapping("/newsByAuthor")
public List<News> getByAuthor(@RequestParam("author") String author) {
    return newsService.getByAuthor(author);
}
```

```
List<News> findByAuthor(String author);
```

3.1.21. /news/newsByAuthorFirstLetters?letters={x}

newsy od autora zaczynającego się na daną literę

```
@GetMapping("/newsByAuthorFirstLetters")
public List<News> getByAuthorFirstLetters(@RequestParam("letters") String letters){
    return newsService.getByAuthorFirstLetters(letters);
}
```

```
@Query("{ 'author': { $regex: '^?0', $options: 'i' } }")
List<News> findByAuthorStartingWith(String letter);
```

3.1.22. /news/{from_}/{to_}

newsy z zadanego przedziału czasowego

```
@GetMapping("/news/{from_}/{to_}")
public List<News> getNewsBetweenDates(@PathVariable("from_") String from_, @PathVariable("to_") String to_) {
    SimpleDateFormat format = new SimpleDateFormat( pattern: "yyyy-MM-dd");
    try {
        Date from = format.parse(from_);
        Date to = format.parse(to_);
        return newsService.getNewsBetweenDates(from, to);
    } catch (ParseException e) {
        return new ArrayList<>();
    }
}
```

```
@Query("{ 'publicationDate': { $gte: ?0, $lte: ?1 } }")
List<News> findBetweenDates(Date from, Date to);
```

3.1.23. /news/byYear?year={x}

newsy z podanego roku

```
@GetMapping("/news/byYear")
public List<News> getNewsFromOneYear(@RequestParam("year") String year) {
    SimpleDateFormat format = new SimpleDateFormat( pattern: "yyyy-MM-dd");
    String from_ = year+ "-01-01";
    String to_ = year + "-12-31";
    try {
        Date from = format.parse(from_);
        Date to = format.parse(to_);
        return newsService.getNewsBetweenDates(from, to);
    } catch (ParseException e) {
        return new ArrayList<>();
    }
}
```

3.1.24. /news/byYearAndMonth?year={x}&?month={y}

newsy z podanego roku i miesiąca

```
@GetMapping("/byYearAndMonth")
public List<News> getNewsFromOneYearAndMonth(@RequestParam("year") String year,
                                              @RequestParam("month") String month) {
    SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
    String from_ = year + "-" + month + "-01";
    String to_ = switch (month) {
        case "1", "3", "5", "7", "8", "10", "12" -> year + "-" + month + "-31";
        case "4", "6", "9", "11" -> year + "-" + month + "-30";
        case "2" -> year + "-" + month + "-28";
        default -> "";
    };
    try {
        Date from = format.parse(from_);
        Date to = format.parse(to_);
        return newsService.getNewsBetweenDates(from, to);
    } catch (ParseException e) {
        return new ArrayList<>();
    }
}
```

3.1.25. /news/{author}/{from_}/{to_}

newsy od podanego autora w zadanym przedziale czasowym

```
@GetMapping("/{author}/{from_}/{to_}")
public List<News> getByAuthorAndBetweenDates(@PathVariable("author") String author,
                                              @PathVariable("from_") String from_,
                                              @PathVariable("to_") String to_){
    SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
    try {
        Date from = format.parse(from_);
        Date to = format.parse(to_);
        return newsService.getByAuthorAndBetweenDates(author, from, to);
    } catch (ParseException e) {
        return new ArrayList<>();
    }
}
```

```
@Aggregation(pipeline = {
    "{$match: {$and: [{author: ?0}, {publicationDate: {$gte: ?1}}, " +
    "{publicationDate: {$lt: ?2}}]}}",
    "{$sort: {publicationDate: -1}}")
})
List<News> findNewsByAuthorAndDate(String author, Date startDate, Date endDate);
```

3.1.26. /news/newsByTitle?title={x}

newsy o podanym tytule


```
@GetMapping("/newsByTitle")
public List<News>getByTitle(@RequestParam("title") String title){
    return newsService.getByTitle(title);
}
```

```
@Aggregation(pipeline = {
    "{$match: {title: ?0}}",
    "{$sort: {publicationDate: -1}}"
})
List<News> findNewsByTitle( String title);
```

3.1.27. /news/sortByAuthor

newsy posortowane według autora

```
@GetMapping("/sortByAuthor")
public List<News>getSortedByAuthor() {
    return newsService.getSortedByAuthor();
}
```

3.1.28. /news/newsByAuthorAndContent?content={x}&?author={y}

newsy od danego autora zawierające podany fragment

```
@GetMapping("/newsByAuthorAndContent")
public List<News>getByAuthorAndContent(@RequestParam("content") String content,
                                       @RequestParam("author") String author){
    return newsService.getByAuthorAndContent(content, author);
}
```

```
@Query("{ 'author': {$regex: '^?1', $options: 'i'}, " +
        "'content': {$regex: '?0', $options: 'i'}}")
List<News> findByAuthorAndContent(String content, String author);
```

3.1.29. /news/newsByContent?content={x}

newsy zawierające podany fragment

```
@GetMapping("/newsByContent")
public List<News>getByContentFragment(@RequestParam("content") String content){
    return newsService.getByContentFragment(content);
}
```

```
@Query("{ 'content': {$regex: '?0', $options: 'i'}}")
List<News> getByContentFragment(String fragment);
```

3.1.30. /reviews/all

wszystkie recenzje

```
@GetMapping("/all")
public List<Review> getAll() {
    return this.reviewService.getAllReviews();
}
```

3.1.31. /reviews/{reviewID}

recenzja o podanym ID

```
@GetMapping("/{reviewID}")
public Optional<Review> getReviewByID(@PathVariable("reviewID") String reviewID) {
    return this.reviewService.getReviewByID(reviewID);
}
```

3.1.32. /reviews/reviewsByCarrotRate

recenzje posortowane według rankingu

```
@GetMapping("/reviewsByCarrotRate")
public List<Review> getByCarrotRate(){
    return reviewService.getByCarrotRate();
}
```

3.1.33. /reviews/top

wyświetlenie najlepszych recenzji według rankingu

```
@GetMapping("/top")
public List<Review> getTopByCarrotRate(){
    return reviewService.getTopByCarrotRate();
}
```

3.1.34. /reviews/reviewsByContent

wyświetlanie recenzji zawierających podany fragment

```
@GetMapping("/reviewsByContent")
public List<Review> getByContent(@RequestParam("content") String content){
    return reviewService.getReviewsByContent(content);
}
```

```
@Query("{ 'content': { $regex: '?0', $options: 'i' } }")
List<Review> getReviewsByContent( String title);
```

3.1.35. /reviews/reviewsByAuthor

wyświetlanie recenzji od podanego autora

```
@GetMapping("/reviewsByAuthor")
public List<Review> getByAuthor(@RequestParam("author") String author){
    return reviewService.getReviewsByAuthor(author);
}
```

3.1.36. /reviews/newest

wyświetlanie najnowszych recenzji

```
@GetMapping("/newest")
public List<Review> getNewest(){
    return reviewService.findByOrderPublicationDateDesc();
}

List<Review> findTop15ByOrderByCarrotRateDesc();
```

3.2. Triggery

MongoDB Atlas zezwala na dodanie jedynie 5 triggerów na klaster (używając darmowego planu).

maximum database trigger count for cluster size='M0' is 5

Add Trigger

Kod triggerów został zapisany w pliku *triggers.js* .

3.2.1. UpdateGameAndUserOnReviewInsert

```
exports = async function(changeEvent) {
  const insertedReview = changeEvent.fullDocument;
  const gameIdObj = BSON.ObjectId(insertedReview.gameID);
  const author = insertedReview.author;

  const gamesCollection = context.services.get("rotten-carrots-database").db("rotten-carrots-database").collection("games");
  const game = await gamesCollection.findOne({_id: gameIdObj});

  const reviewsCollection = context.services.get("rotten-carrots-database").db("rotten-carrots-database").collection("reviews");
  const reviews = await reviewsCollection.find({gameID: insertedReview.gameID}).toArray();

  const usersCollection = context.services.get("rotten-carrots-database").db("rotten-carrots-database").collection("users");
  const user = await usersCollection.findOne({nickname: author});

  // if user has already reviewed the game - delete the duplicate
  if (user.reviews.length > 0){
    if (user.reviews.some(review => String(review.gameID) === String(gameIdObj))) {
      console.log("User has already reviewed the game.");
      await reviewsCollection.deleteOne({_id: insertedReview._id})
      return;
    }
  }

  const userReviews = user.reviews;
  userReviews.push(insertedReview)

  if (reviews.length > 0) {

    let totalRating = 0;
    for (const review of reviews) {
      totalRating += review.carrotRate;
    }
    const averageRating = totalRating / reviews.length;

    await gamesCollection.updateOne(
      { _id: gameIdObj },
      {
        $set: {
          carrotRate: averageRating,
          reviews: reviews
        }
      }
    );

    await usersCollection.updateOne(
      {nickname: author},
      {
        $set: {
          reviews: userReviews
        }
      }
    );

    console.log("Game and User documents updated successfully.");
    console.log(averageRating);
  }
};
```

3.2.2. ValidateNickname

Sprawdzanie, czy użytkownik o podanym nickname'ie już istnieje w bazie.

```
exports = async function(changeEvent) {
  const insertedUser = changeEvent.fullDocument;

  const usersCollection = context.services.get("rotten-carrots-database").db("rotten-carrots-database").collection("users");
  const usersWithNickname = await usersCollection.find({nickname: insertedUser.nickname}).toArray();

  if(usersWithNickname.length > 1){
    console.log("User with this nickname already exists");
    await usersCollection.deleteOne(insertedUser);
  }
};
```

3.2.3. UpdateUserOnAuctionInsert

Służy do dodawania aukcji do listy auctions w użytkowniku, który jest jej właścicielem.

```
exports = async function(changeEvent) {
  const insertedAuction = changeEvent.fullDocument;
  const ownerIdObj = BSON.ObjectId(insertedAuction.ownerID);

  const usersCollection = context.services.get("rotten-carrots-database").db("rotten-carrots-database").collection("users");
  const user = await usersCollection.findOne({_id: ownerIdObj});
  const userAuctions = user.auctions;

  userAuctions.push(insertedAuction);

  await usersCollection.updateOne(
    {_id: ownerIdObj},
    {
      $set: {
        auctions: userAuctions
      }
    }
  );
};
```

3.2.4. UpdateAuctionStatusInUser

Służy do aktualizacji statusu aukcji (aktywna/nieaktywna) w liście auctions użytkownika.

4. Podsumowanie

Efektym wykonania projektu jest system backendowy, który łatwo można zintegrować z frontendem, tworząc użyteczną aplikację. Wykorzystanie frameworka Spring pozwoliło na efektywne zarządzanie żadaniami HTTP, konfigurację aplikacji oraz tworzenie endpointów API. Dzięki temu, system jest elastyczny, skalowalny i oferuje spójne API, które może być używane przez frontendowców do tworzenia bogatych interfejsów użytkownika. Zastosowanie MongoDB w projekcie przyniosło elastyczność przechowywania danych, skalowalność, szybkie operacje odczytu i zapisu oraz ułatwienia w procesie rozwoju aplikacji.