Advanced Web Development
RESTful APIs

Disuqi Hijazi
@00609702 | d.hijazi@edu.salford.ac.uk

Contents

1. Code Report

1.1 Fruit Movies API

1.1.1 Implementation

The API implementation has thorough alignment with Massé's (2012) recommended best practices for designing RESTful APIs and Fredrich's (2024) best practices. Let's delve into a more detailed exploration of each aspect:

1. **Hierarchical URIs:** One prominent feature is the consistent use of forward slash separators (/) in URIs to denote hierarchical relationships. This structural clarity is evident in paths such as GET api/v1/movies/{id}/crewMembers and api/v1/users/{id}/reviews.
2. **URI Formatting:** Across the board, trailing forward slashes (/) are deliberately omitted from all URIs, maintaining a uniform and tidy formatting standard.
3. **Readability and Clarity:** The intentional avoidance of hyphens (-) and underscores (_) in URIs contributes to enhanced readability, ensuring that resource paths are clear and intelligible.
4. **Case Consistency:** Lowercase letters are meticulously employed throughout URI paths, with each URI consistently commencing with a lowercase letter. This uniformity aids in maintaining coherence and predictability in API interactions.
5. **File Extensions:** A noteworthy design choice is the omission of file extensions from URIs, underscoring a commitment to simplicity and adaptability in resource representation.
6. **Subdomain Uniformity:** The utilization of consistent subdomain names, such as movies, users, reviews, and crewMembers, fosters an organizational structure that is both intuitive and coherent. Furthermore, deeper subdomain levels are managed with consistency and clarity.
7. **Resource Representation:** Adhering to the guideline of employing nouns to represent resources and collections, the API ensures a logical and intuitive approach to resource identification. For instance, /api/v1/movies represents a collection of movie resources, while /api/v1/movies/{id} denotes a single movie resource.
8. **Query Parameters:** The incorporation of query parameters for resource filtering enhances the flexibility and usability of the API for clients. For instance, /api/v1/movies?page=2 enables clients to retrieve specific pages of movies, while /api/v1/crewMembers?role=Actor facilitates the retrieval of crew members based on their roles.
9. **HTTP Methods:** The API rigorously adheres to the appropriate use of HTTP methods for manipulating resources. GET is utilized for resource retrieval, POST for resource creation, PUT for resource updating, and DELETE for resource deletion, ensuring adherence to REST principles.
10. **HTTP Response Codes:** Utilizing HTTP response codes to convey the status of requests, the API ensures clear and consistent communication with clients. Successful requests yield a 200 OK response code, while errors are indicated by 4xx response codes.
11. **Media Type Standardization:** The API adheres to the use of standard media types, such as application/json, for representing resources. This standardization promotes compatibility and interoperability across different systems and platforms.

In addition to these foundational principles, the API implementation also encompasses advanced strategies and features:

- **Resource Modeling:** Each path segment in the URIs corresponds to a unique resource, reflecting a robust approach to resource modeling and management. (Fredrich, 2024) (Massé, 2012) (What Is a REST API? | IBM, 2024)
- **Documentation:** The FruitMovies platform boasts detailed documentation utilizing the OpenAPI Specification (OAS) format. This comprehensive resource offers extensive insights into available endpoints, data schemas, and authentication prerequisites. Additionally, it includes an auto-generated documentation page featuring illustrative examples and the functionality to experiment with API calls. Such robust documentation greatly enhances accessibility and usability for developers seeking to integrate with the platform. (Sohan et al., 2017).
- **Authentication and Authorization:** The API incorporates authentication mechanisms, such as JSON Web Tokens (JWT), and role-based access control (RBAC) to secure access to resources based on user roles. (Jones et al., 2015)
- **Error Handling:** Robust error handling mechanisms provide clear and informative error messages to clients, enhancing the overall reliability and usability of the API. (Fredrich, 2024) (Massé, 2012)
- **Pagination and Versioning:** Features such as pagination and versioning contribute to scalability and maintainability, enabling efficient management of API resources and updates over time. (Fredrich, 2024) (Massé, 2012)

In summary, the API implementation not only adheres to best practices for RESTful API design but also incorporates advanced features and strategies to ensure robustness, security, and scalability. It provides a comprehensive and powerful interface for interacting with the Fruit Movies platform, catering to a wide range of use cases and scenarios.

1.1.2 Future Works and Improvements

There are numerous avenues for refining and enhancing the API architecture of the Fruit Movies platform. Below, we explore several potential directions for future development:

1. **Enhanced Access Control:** While the current API employs role-based access control (RBAC) to govern resource access, there may arise scenarios necessitating more nuanced control. Implementing attribute-based access control (ABAC) or other adaptable mechanisms could offer finer-grained control over resource access. (Rivera et al., 2017)
2. **Integration of Webhooks:** Introducing support for webhooks would enable clients to receive real-time notifications upon specific system events, such as the addition of a new movie or the submission of a review. This addition could foster timely updates and bolster API responsiveness. (Biehl, 2017)
3. **Refined Error Handling:** While the API presently incorporates error handling mechanisms, opportunities exist to enhance the clarity and utility of error messages. Providing comprehensive details regarding error causes and suggesting potential remedies could expedite issue diagnosis and resolution for clients.
4. **Incorporation of Sorting and Filtering:** Enabling sorting and filtering capabilities within the API would empower clients to efficiently retrieve and manipulate specific subsets of data. For instance, clients could filter movies by genre, release date, or

rating, or sort them by various criteria like title or popularity. This augmentation would amplify the API's flexibility and potency, facilitating the creation of more sophisticated integrations with the Fruit Movies platform. (Fredrich, 2024) (Massé, 2012)

5. **Adoption of HATEOAS Principle:** Implementing Hypermedia as the Engine of Application State (HATEOAS), a foundational tenet of RESTful API design, involves embedding links to related resources in API responses. This dynamic approach enables clients to discover and navigate the API seamlessly, potentially reducing reliance on hard-coded URLs or additional requests to access related resources. Such a measure could greatly enhance the API's intuitiveness and user-friendliness. (Varanasi & Sudha Belida, 2015)
6. **Development of Integration Tools:** Crafting a dedicated library or software development kit (SDK) for seamless integration with the API could significantly streamline the integration process. By abstracting away intricacies associated with making HTTP requests and handling responses, such a tool would simplify API interaction for developers and lower the barrier to entry for newcomers, thus fostering broader adoption and facilitating innovative integrations with the Fruit Movies platform.

By addressing these potential areas for improvement and considering other avenues for refinement, the Fruit Movies API stands to become more resilient, scalable, and user-centric. Such enhancements would empower developers to create even more potent and innovative integrations with the platform, ultimately enriching the overall user experience.

1.2 API Consumption

The FruitMovies website currently integrates two APIs, namely the Tmdb REST API (TMDB API, 2024) and the OpenStreetMap API (API - OpenStreetMap Wiki, 2022), facilitated by Guzzle (Overview — Guzzle Documentation, 2021).

The Tmdb API serves multiple purposes within the website's functionality. Firstly, it generates individual pages for each actor or director, a feature previously absent from the site. These pages furnish details such as the individual's role, gender, TMDB popularity, and their notable movies or TV shows, all sourced from TMDB. Additionally, for each movie, the website retrieves relevant data including TMDB popularity, average vote, and videos (such as clips and trailers), presenting this information seamlessly to the user on the same page. Moreover, when a user attempts to add a movie, the form utilizes TMDB API suggestions, beginning with the title. As the user types, requests are dispatched to the TMDB API to search for movies with similar names. Upon selecting a title from the suggestions, the overview of that movie is also provided as a suggestion. Furthermore, the director and actors are suggested, with a convenient option available to add all actors automatically, streamlining the addition process.

Conversely, the OpenStreetMap API fulfills a singular function on the website. Specifically, on the movie page, a section lists the closest cinemas, with each listing comprising the cinema title and address. Upon first accessing a movie page, users are prompted to allow location services. If permission is granted, the website suggests nearby cinemas; however, if declined, no suggestions are provided. Clicking on a cinema opens a new tab to the Google Maps website, facilitating the user in obtaining precise directions to the chosen cinema.

Through the integration of these APIs, the FruitMovies website enriches its user experience by offering comprehensive information and convenient features related to movies, actors, and nearby cinemas.

References

1. *API - OpenStreetMap Wiki*. (2022). Openstreetmap.org. https://wiki.openstreetmap.org/wiki/API
2. Biehl, M. (2017). *Webhooks – Events for RESTful APIs*. API-University Press.
3. Fredrich, T. (2024). *RESTful Web Services Resources*. Restapitutorial.com. https://www.restapitutorial.com/resources.html
4. Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT)*. https://doi.org/10.17487/rfc7519
5. Massé, M. (2012). *REST API design rulebook [designing consistent RESTful web service interfaces]*. Beijing [U.A.] O'reilly.
6. *Overview — Guzzle Documentation*. (2021). Guzzlephp.org. https://docs.guzzlephp.org/en/stable/overview.html
7. Rivera, K., Demurjian, S. A., & Baihan, M. S. (2017). *Achieving RBAC on RESTful APIs for Mobile Apps Using FHIR*. https://doi.org/10.1109/mobilecloud.2017.22
8. Sohan, S. M., Maurer, F., Anslow, C., & Robillard, M. P. (2017). *A study of the effectiveness of usage examples in REST API documentation*. https://doi.org/10.1109/vlhcc.2017.8103450
9. *TMDB API*. (2024). The Movie Database (TMDB). https://developer.themoviedb.org/docs/getting-started
10. Varanasi, B., & Sudha Belida. (2015). HATEOAS. *Apress EBooks*, 165–174. https://doi.org/10.1007/978-1-4842-0823-6_10
11. *What is a REST API? | IBM*. (2024). Ibm.com. https://www.ibm.com/topics/rest-apis#:~:text=REST%20APIs%20need%20to%20be,end%20application%20or%20an%20intermediary.&text=REST%20APIs%20usually%20send%20static,(such%20as%20Java%20applets).

Fruit Movies API 1.0.0 OAS 3.0

Authentication

POST /api/login_check

Login and get JWT token

Parameters

No parameters

Request body required

application/json

Example Value Schema

```
{
  "username": "Username",
  "password": "?pA$$w0rd!"
}
```

Responses

Code	Description	Links
200	Successful operation Media type application/json Controls `Accept` header. **Example Value** Schema `{ "token": "string" }`	*No links*

POST `/api/token/refresh`

Refresh JWT token

Parameters

No parameters

Responses

Code	Description	Links
200	Successful operation Media type application/json Controls `Accept` header. **Example Value** Schema `{ "token": "string" }`	*No links*
401	Missing JWT Refresh Token Media type application/json **Example Value** Schema `{ "code": 401, "message": "Missing JWT Refresh Token" }`	*No links*

Documentation

GET `/api/v1/docs` Get API Documentation

Get API Documentation

Parameters

No parameters

Responses

Code	Description	Links
200	Successful operation Media type application/json Controls `Accept` header. **Example Value** Schema	*No links*


```
{
  "openapi": "string",
  "info": {
    "title": "string",
    "version": "string",
    "contact": {
      "email": "string"
    }
  },
  "paths": {}
}
```


Crew Members

GET `/api/v1/crewMembers` Get a list of crew members

Parameters

Name	Description
page `integer` *(query)*	The page number page
role `string` *(query)*	The role of the crew member *Available values* : actor, director actor

Responses

Code	Description	Links

200	List of crew members
	
	Media type
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	
	[
	{
	"id": 1,
	"name": "John Doe",
	"photo": "path/to/image.jpg",
	"role": "Director"
	}
	]
		No links

400	Bad request
	
	Media type
	application/json
	
	Example Value Schema
	
	{
	"message": "Bad request"
	}
		No links

401	Unauthorized
	
	Media type
	application/json
	
	Example Value Schema
	
	{
	"code": 401,
	"message": "JWT Token not found"
	}
		No links

404	Out of range
	
	Media type
	application/json
	
	Example Value Schema
	
	{
	"message": "Out of range"
	}
		No links

POST `/api/v1/crewMembers` Add a new crew member

Parameters

No parameters

Request body required

application/json

Crew member data

Example Value Schema


```
{
  "id": 1,
  "name": "John Doe",
  "photo": "path/to/image.jpg",
  "role": "Director"
}
```


Responses

Code	Description	Links

201	Crew member created successfully
	
	Media type
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	`{ "message": "Successfully created Crew Member" }`
	
	Headers:
	Name / **Description** / **Type**
	`Location` / `URL of the new crew memeber` / `string`
		No links

208	Crew Member already exists Media type application/json **Example Value** Schema `{ "message": "Crew Member already exists" }`	*No links*
400	Invalid input Media type	*No links*

Code	Description	Links

application/json

Example Value Schema

```
{
  "message": "Invalid input"
}
```

401 Unauthorized — *No links*

Media type

application/json

Example Value Schema

```
{
  "code": 401,
  "message": "JWT Token not found"
}
```

403 Only admins can add crew members — *No links*

Media type

application/json

Example Value Schema

```
{
  "message": "Only admins can add crew members"
}
```

GET `/api/v1/crewMembers/{id}` Get a specific crew member by ID

Parameters

Name	Description
id * required `integer` *(path)*	The ID of the crew member id

Responses

Code	Description	Links

200	Crew member details
	
	Media type
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	
	{
	"id": 1,
	"name": "John Doe",
	"photo": "path/to/image.jpg",
	"role": "Director"
	}
		No links

401	Unauthorized
	
	Media type
	application/json
	
	Example Value Schema
	
	{
	"code": 401,
	"message": "JWT Token not found"
	}
		No links

404	Crew member not found	*No links*

PUT `/api/v1/crewMembers/{id}` Update a specific crew member by ID

Parameters

Name	Description
id * required `integer` *(path)*	The ID of the crew member id

Request body required — application/json

Crew member data

Example Value Schema

{
"id": 1,
"name": "John Doe",
"photo": "path/to/image.jpg",

```
  "role": "Director"
}
```


Responses

Code	Description	Links

202	Crew member updated successfully
	
	Media type
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	
	`{ "id": 1, "name": "John Doe", "photo": "path/to/image.jpg", "role": "Director" }`
	
	Headers:
	
	Name / **Description** / **Type**
	`Location` / `URL of the updated crew member` / `string`
		No links

400	Invalid input Media type **application/json** **Example Value** Schema `{ "message": "Invalid input" }`	*No links*
401	Unauthorized Media type **application/json** **Example Value** Schema `{ "code": 401, "message": "JWT Token not found" }`	*No links*
404	Crew member not found Media type **application/json**	*No links*

Code	Description	Links

Example Value Schema

```
{
  "message": "Crew member not found"
}
```

DELETE `/api/v1/crewMembers/{id}` Delete a specific crew member

Parameters

Name	Description
id * required `integer` *(path)*	The ID of the crew member id

Responses

Code	Description	Links
200	Crew member deleted successfully Media type **application/json** Controls `Accept` header. **Example Value** Schema `{ "message": "Successfully deleted Crew Member" }`	*No links*
401	Unauthorized Media type **application/json** **Example Value** Schema `{ "code": 401, "message": "JWT Token not found" }`	*No links*
403	Only admins can delete crew members	*No links*

Code	Description	Links

Media type

application/json

Example Value Schema

```
{
  "message": "Only admins can delete crew members"
}
```

404	Crew member not found	*No links*

Media type

application/json

Example Value Schema

```
{
  "message": "Crew member not found"
}
```

Movies

GET `/api/v1/movies` Get a page of movies

Parameters

Name	Description
page `integer` *(query)*	The page number page

Responses

Code	Description	Links
200	Page of movies retrieved successfully	*No links*

Media type

application/json

Controls `Accept` header.

Example Value Schema

Code	Description	Links


```
{
  "results": [
    {
      "id": 1,
      "title": "The Godfather",
      "running_time": 175,
      "cover_photo": "path/to/image.jpg",
      "overview": "The aging patriarch of an organized crime dynasty transfers cont
rol of his clandestine empire to his reluctant son.",
      "release_date": "1972-03-14T00:00:00+00:00"
    }
  ],
  "current_page": 0,
  "total_pages": 0
}
```


401 Unauthorized — *No links*

Media type

application/json

Example Value Schema


```
{
  "code": 401,
  "message": "JWT Token not found"
}
```


404 Page not found — *No links*

Media type

application/json

Example Value Schema


```
{
  "message": "Page not found"
}
```


POST `/api/v1/movies` Create a new movie

Parameters

No parameters

Request body required — **application/json**

Movie data

Example Value Schema


```
{
  "title": "Example Movie Title",
  "overview": "This is an example overview",
  "running_time": 120,
```

```
  "release_date": "dd-mm-yyyy"
}
```


Responses

Code	Description	Links

201	Movie created successfully
	
	Media type
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	
	`{ "message": "Successfully added Movie" }`
	
	Headers:
	
	Name / **Description** / **Type**
	`Location` / `URL of the new movie` / `string`
		No links

400	Invalid input Media type **application/json** **Example Value** Schema `{ "message": "Invalid input" }`	*No links*
401	Unauthorized Media type **application/json** **Example Value** Schema `{ "code": 401, "message": "JWT Token not found" }`	*No links*
403	Only admins can create movies Media type **application/json**	*No links*

Code	Description	Links

Example Value Schema

```
{
  "message": "Only admins can create movies"
}
```

409 Movie already exists — *No links*

Media type: **application/json**

Example Value Schema

```
{
  "message": "Movie already exists"
}
```

GET /api/v1/movies/{id} Get a specific movie by ID

Parameters

Name	Description
id * required `integer` *(path)*	The ID of the movie id

Responses

Code	Description	Links

200 Movie retrieved successfully — *No links*

Media type: **application/json**

Controls `Accept` header.

Example Value Schema

```
{
  "id": 1,
  "title": "The Godfather",
  "running_time": 175,
  "cover_photo": "path/to/image.jpg",
  "overview": "The aging patriarch of an organized crime dynasty transfers control
of his clandestine empire to his reluctant son.",
```

Code	Description	Links


```
  "release_date": "1972-03-14T00:00:00+00:00"
}
```


401 Unauthorized — *No links*

Media type: application/json

Example Value Schema


```
{
  "code": 401,
  "message": "JWT Token not found"
}
```


404 Movie not found — *No links*

Media type: application/json

Example Value Schema


```
{
  "message": "Movie not found"
}
```


PUT /api/v1/movies/{id} Update an existing movie

Parameters

Name	Description
id * required integer *(path)*	The ID of the movie to update id

Request body required — application/json

Movie data

Example Value Schema


```
{
  "id": 1,
  "title": "The Godfather",
  "running_time": 175,
  "cover_photo": "path/to/image.jpg",
  "overview": "The aging patriarch of an organized crime dynasty transfers control of his clandestine empire to his reluctant son.",
  "release_date": "1972-03-14T00:00:00+00:00"
}
```

Responses

Code	Description	Links

202	Movie updated successfully
	
	Media type
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	
	`{ "message": "Successfully updated Movie" }`
	
	Headers:
	
	Name / **Description** / **Type**
	`Location` / `URL of the updated movie` / `string`
		No links

400	Invalid input Media type **application/json** **Example Value** Schema `{ "message": "Invalid input" }`	*No links*
401	Unauthorized Media type **application/json** **Example Value** Schema `{ "code": 401, "message": "JWT Token not found" }`	*No links*
403	Only admins can modify movies Media type **application/json** **Example Value** Schema	*No links*

Code	Description	Links

```
{
  "message": "Only admins can modify movies"
}
```

404 Movie not found — *No links*

Media type

application/json

Example Value Schema

```
{
  "message": "Movie not found"
}
```

DELETE /api/v1/movies/{id} Delete a movie

Parameters

Name	Description
id * required `integer` *(path)*	The ID of the movie to delete id

Responses

Code	Description	Links
200	Movie deleted successfully Media type application/json Controls `Accept` header. **Example Value** Schema `{ "message": "Successfully deleted movie" }`	*No links*
401	Unauthorized Media type application/json	*No links*

Code	Description	Links

Example Value Schema

```
{
  "code": 401,
  "message": "JWT Token not found"
}
```

403 Only admins can delete movies — *No links*

Media type

application/json

Example Value Schema

```
{
  "message": "Only admins can delete movies"
}
```

404 Movie not found — *No links*

Media type

application/json

Example Value Schema

```
{
  "message": "Movie not found"
}
```

GET `/api/v1/movies/{id}/crewMembers` Get the crew members of a movie

Parameters

Name	Description
id * required `integer` *(path)*	The ID of the movie id

Responses

Code	Description	Links
200	Crew members retrieved successfully	*No links*

Media type

Code	Description	Links

application/json

Controls `Accept` header.

Example Value Schema

```
{
  "director": "string",
  "actors": [
    "string"
  ]
}
```

401 Unauthorized — *No links*

Media type

application/json

Example Value Schema

```
{
  "code": 401,
  "message": "JWT Token not found"
}
```

404 Movie not found — *No links*

Media type

application/json

Example Value Schema

```
{
  "message": "Movie not found"
}
```

PUT /api/v1/movies/{id}/crewMembers Update the crew members of a specific movie

Parameters

Name	Description
id * required `integer` *(path)*	The ID of the movie id

Request body required — application/json

Crew member data

Example Value Schema

```
{
  "id": 1,
  "name": "John Doe",
  "photo": "path/to/image.jpg",
  "role": "Director"
}
```


Responses

Code	Description	Links

202	Crew members updated successfully
	
	Media type
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	
	`{ "id": 1, "name": "John Doe", "photo": "path/to/image.jpg", "role": "Director" }`
	
	Headers:
	
	Name / **Description** / **Type**
	`Location` — `URL of the updated movie` — `string`
		No links

400	Invalid input Media type application/json **Example Value** Schema `{ "message": "Invalid input" }`	*No links*
401	Unauthorized Media type application/json **Example Value** Schema `{ "code": 401, "message": "JWT Token not found" }`	*No links*

Code	Description	Links

403	Only admins can edit movies
	Media type
	application/json
	Example Value Schema
	{
	"message": "Only admins can edit movies"
	}
		No links

404	Movie or crew member not found
	Media type
	application/json
	Example Value Schema
	{
	"message": "Movie or crew member not found"
	}
		No links

Reviews

GET `/api/v1/reviews` Get a page of reviews

Parameters

Name	Description
Authorization * required `string` *(header)*	Bearer token needed for this operation Authorization

Request body application/json

Page data

Example Value Schema

```
{
  "page": 1
}
```

Responses

Code	Description	Links
200	Reviews retrieved successfully Media type application/json Controls `Accept` header. **Example Value** Schema	*No links*


```
{
  "results": [
    {
      "id": 1,
      "movie": {
        "id": 1,
        "title": "The Godfather",
        "running_time": 175,
        "cover_photo": "path/to/image.jpg",
        "overview": "The aging patriarch of an organized crime dynasty transfers co
ntrol of his clandestine empire to his reluctant son.",
        "release_date": "1972-03-14T00:00:00+00:00"
      },
      "user": {
        "id": 1,
        "username": "john_doe",
        "restricted": false,
        "date_joined": "2022-01-01T00:00:00+00:00",
        "profile_image": "path/to/image.jpg"
      },
      "score": 5,
      "comment": "This movie was amazing!",
      "date_reviewed": "2022-01-01T00:00:00+00:00"
    }
  ],
  "current_page": 0,
  "total_pages": 0
}
```


404	Out of range Media type application/json **Example Value** Schema	*No links*


```
{
  "message": "Out of range"
}
```


POST `/api/v1/reviews` Create a new review

Parameters

No parameters

Request body required

application/json

Review data

Example Value Schema


```
{
  "score": 5,
  "comment": "Great Movie",
  "movie_id": 20
}
```


Responses

Code	Description	Links

201	Successfully added review
	
	Media type
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	
	`{ "message": "Successfully added review" }`
	
	Headers:
	
	Name / **Description** / **Type**
	`Location` / `URL of the new review` / `string`
		No links

400	Bad request Media type application/json **Example Value** Schema `{ "error message": "Something is wrong in your request" }`	*No links*
401	Unauthorized Media type application/json **Example Value** Schema	*No links*

Code	Description	Links


```
{
  "code": 401,
  "message": "JWT Token not found"
}
```


404	Movie not found Media type application/json **Example Value** Schema	*No links*


```
{
  "message": "Movie not found"
}
```


409	Review already exists Media type application/json **Example Value** Schema	*No links*


```
{
  "message": "Review already exists"
}
```


GET `/api/v1/reviews/{id}` Get a review by its ID

Parameters

Name	Description
Authorization * required `string` *(header)*	Bearer token needed for this operation Authorization
id * required `integer` *(path)*	The ID of the review id

Responses

Code	Description	Links
200	Review retrieved successfully	*No links*

Code	Description	Links

Media type

application/json

Controls `Accept` header.

Example Value Schema


```
{
  "id": 1,
  "movie": {
    "id": 1,
    "title": "The Godfather",
    "running_time": 175,
    "cover_photo": "path/to/image.jpg",
    "overview": "The aging patriarch of an organized crime dynasty transfers contro
l of his clandestine empire to his reluctant son.",
    "release_date": "1972-03-14T00:00:00+00:00"
  },
  "user": {
    "id": 1,
    "username": "john_doe",
    "restricted": false,
    "date_joined": "2022-01-01T00:00:00+00:00",
    "profile_image": "path/to/image.jpg"
  },
  "score": 5,
  "comment": "This movie was amazing!",
  "date_reviewed": "2022-01-01T00:00:00+00:00"
}
```


404 Review not found — *No links*

Media type

application/json

Example Value Schema


```
{
  "message": "Review not found"
}
```


PUT /api/v1/reviews/{id} Update a review

Parameters

Name	Description
id * required `integer` *(path)*	The ID of the review id

Request body required — application/json

Review data

Example Value Schema


```
{
  "score": 5,
  "comment": "Great Movie",
  "movie_id": 20
}
```


Responses

Code	Description	Links

202	Successfully updated review
	
	Media type
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	`{ "message": "Successfully updated review" }`
	
	Headers:
	Name / **Description** / **Type**
	`Location` / `URL of the updated review` / `string`
		No links

400	Bad request Media type **application/json** **Example Value** Schema `{ "error message": "Something is wrong in your request" }`	*No links*
401	Unauthorized Media type **application/json** **Example Value** Schema `{ "code": 401, "message": "JWT Token not found" }`	*No links*
403	Cannot change another user's review	*No links*

Code	Description	Links

Media type

application/json

Example Value Schema

```
{
  "message": "Cannot change another user's review"
}
```

404	Review or movie not found	*No links*

Media type

application/json

Example Value Schema

```
{
  "message": "Review or movie not found"
}
```

DELETE /api/v1/reviews/{id} Delete a review

Parameters

Name	Description
id * required `integer` *(path)*	The ID of the review id

Responses

Code	Description	Links
200	Successfully deleted review	*No links*

Media type

application/json

Controls `Accept` header.

Example Value Schema

```
{
  "message": "Successfully deleted review"
}
```

Code	Description	Links

401	Unauthorized
	Media type
	application/json
	Example Value Schema
	{
	"code": 401,
	"message": "JWT Token not found"
	}
		No links

403	Cannot delete another user's review
	Media type
	application/json
	Example Value Schema
	{
	"message": "Cannot delete another user's review"
	}
		No links

404	Review not found
	Media type
	application/json
	Example Value Schema
	{
	"message": "Review not found"
	}
		No links

GET `/api/v1/reviews/{id}/votes` Get the votes of a review

Parameters

Name	Description
Authorization * required `string` *(header)*	Bearer token needed for this operation Authorization
id * required `integer` *(path)*	The ID of the review id

Responses

Code	Description	Links

200	Votes retrieved successfully
	
	Media type
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	
	[
	{
	"review_id": 20,
	"user_id": 1,
	"liked": true
	}
	]
		No links

404	Review not found
	
	Media type
	application/json
	
	Example Value Schema
	
	{
	"message": "Review not found"
	}
		No links

Users

GET `/api/v1/users` Get users

Parameters

Name	Description
page `integer` *(query)*	The page number page

Responses

Code	Description	Links

200	Successfully retrieved users
	
	Media type
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	
	{
	"results": [
	{
	"id": 1,
	"username": "john_doe",
	"restricted": false,
	"date_joined": "2022-01-01T00:00:00+00:00",
	"profile_image": "path/to/image.jpg"
	}
	],
	"current_page": 0,
	"total_pages": 0
	}
		No links

401	Unauthorized
	
	Media type
	application/json
	
	Example Value Schema
	
	{
	"code": 401,
	"message": "JWT Token not found"
	}
		No links

404	Out of range
	
	Media type
	application/json
	
	Example Value Schema
	
	{
	"message": "Out of range"
	}
		No links

GET `/api/v1/users/{id}` Find user by ID

Parameters

Name	Description
id * required `integer`	The ID of the user

Name	Description
(path)	id

Responses

Code	Description	Links
200	Successfully retrieved user Media type **application/json** Controls `Accept` header. **Example Value** Schema `{ "id": 1, "username": "john_doe", "restricted": false, "date_joined": "2022-01-01T00:00:00+00:00", "profile_image": "path/to/image.jpg" }`	*No links*
401	Unauthorized Media type **application/json** **Example Value** Schema `{ "code": 401, "message": "JWT Token not found" }`	*No links*
404	User not found Media type **application/json** **Example Value** Schema `{ "message": "User not found" }`	*No links*

GET /api/v1/users/{id}/reviews Get user's reviews

Parameters

Name	Description
id * required integer *(path)*	The ID of the user id

Responses

Code	Description	Links
200	Successfully retrieved user's reviews Media type **application/json** Controls `Accept` header. **Example Value** Schema
```
[
  {
    "id": 1,
    "movie": {
      "id": 1,
      "title": "The Godfather",
      "running_time": 175,
      "cover_photo": "path/to/image.jpg",
      "overview": "The aging patriarch of an organized crime dynasty transfers cont
rol of his clandestine empire to his reluctant son.",
      "release_date": "1972-03-14T00:00:00+00:00"
    },
    "user": {
      "id": 1,
      "username": "john_doe",
      "restricted": false,
      "date_joined": "2022-01-01T00:00:00+00:00",
      "profile_image": "path/to/image.jpg"
    },
    "score": 5,
    "comment": "This movie was amazing!",
    "date_reviewed": "2022-01-01T00:00:00+00:00"
  }
]
```
 | *No links* |
| 401 | Unauthorized

Media type
application/json

Example Value Schema


```
{
  "code": 401,
  "message": "JWT Token not found"
}
```
 | *No links* |
| 404 | User not found | *No links* |

Code	Description	Links

Media type

application/json

Example Value Schema


```
{
  "message": "User not found"
}
```


Schemas

CrewMember model

```
{
  description:  CrewMember model

  id            integer($int64)
                example: 1
                nullable: true
                ID of the crew member

  name          string
                example: John Doe
                Name of the crew member

  photo         string
                example: path/to/image.jpg
                Photo of the crew member

  role          string
                example: Director
                Role of the crew member

}
```

Movie model {

description: Movie model

id
integer($int64)
example: 1
nullable: true
ID of the movie

title
string
example: The Godfather
Title of the movie

running_time
integer
example: 175
Running time of the movie

cover_photo
string
example: path/to/image.jpg
Cover photo of the movie

overview
string
example: The aging patriarch of an organized crime dynasty transfers control of his clandestine empire to his reluctant son.
Overview of the movie

release_date
string
example: 1972-03-14T00:00:00+00:00
Release date of the movie

}

Review model {

description: Review model

id
integer($int64)
example: 1
nullable: true
ID of the review

movie
Movie model {...}

user
User model {...}

score
integer
example: 5
Score of the review

comment
string
example: This movie was amazing!
Comment of the review

date_reviewed
string
example: 2022-01-01T00:00:00+00:00
Date the review was written

}

ReviewVote {

review_id
integer
example: 20

user_id
integer
example: 1

liked*
boolean
example: true

}

User model {

description:	User model
id	integer($int64) *example: 1* ID of the user
username	string *example: john_doe* Username of the user
restricted	boolean *example: false* Whether the user is restricted
date_joined	string *example: 2022-01-01T00:00:00+00:00* Date the user joined
profile_image	string *example: path/to/image.jpg* Profile image of the user

}

NewMovie {

title*	string *example: Example Movie Title*
overview*	string *example: This is an example overview*
running_time	integer *example: 120*
release_date	string *example: dd-mm-yyyy*

}

NewReview {

score*	integer *example: 5*
comment*	string *example: Great Movie*
movie_id*	integer *example: 20*

}

CrewMemberRole string

title: CrewMemberRole

properties: OrderedMap { "Actor": OrderedMap { "description": "Represents an actor", "type": "object", "enum": List ["actor"] }, "Director": OrderedMap { "description": "Represents a director", "type": "object", "enum": List ["director"] } }

example: actor

CrewMemberRole enum

Enum:

[actor, director]