FAVS API DOCUMENTATION

Favs is a new company that aims to provide a better way to organize your favorite things: music, clothes, courses, etc., all in one place.

1.- DESCRIPTION

FAVS API allows user to create a list of favorites, like movies, series, books, etc. The API have been created with a database consisted of two tables, one for the user, and the other for the List of Favorites. Every user can have more than one ListFav, with more than one favorite on each list.

2.- REQUIREMENTS

- Have an IDE where to run the program (Visual Studio Code).
- Have installed Node.js v14+ to run.
- Have installed Postman.
- Have internet connection.
- Check example.env for setting Vars (test.env / dev.env)

3.- HOW TO RUN THE API

Clone the repository https://github.com/Dravernuss/Assesment-Backend-Rodas to your local. Open the repository on you IDE and type the next script on the terminal:

npm install

This script will install all dependencies the project needs. Next, to run the API, type on the terminal the next script:

npm run dev

This script will start the program using the dev.env, which calls the dev database from mongodb where information is storage.

Now, import the Postman Collection from the repository in the Postman App, there you will find all the endpoints to use for the API. The API consist of 11 endpoints, which are explained next. You have to be sure that port 5000 is free for the API.

NOTE: The application is deployed in HEROKU, the API route is:

https://favs-api-esteban.herokuapp.com/

In the repository, open the HEROKU POSTMAN COLLECTION where you will find all the endpoints with the Heroku app.

3.1- METHODS

3.1.1.- USER METHODS:

a) **Register a User**: This endpoint creates a new user to be able to use the functionalities of the webservice. The **POST** endpoint to be called is http://localhost:5000/api/users/create . Is necessary to send a body parameter in JSON with the email and password for the user.

Example Value:

```
{
   "password":"1aAs24",
   "email":"test@test.com"
}
```

FIELD	TYPE	DESCRIPTION
Password	String	The password must have at least 1 uppercase, 1 lowercase letter and 1 number, with minimum length of 6 and cannot be null.
Email	String	The email must have the correct format, can't be repeated nor null.

RESPONSES:

- Status 200: The user has been created successfully.
- Status 403: The program will return a message with the error founded. (see field's descriptions). Also, a message in terminal will appear with the same error.
  - Email cannot be null: email given on body is null.
  - Invalid email: email format given is incorrect.
  - Email already exists!: email to create already exists on database.
  - Password cannot be null: password to be created is null.
  - Password must be at least 6 characters: password length is below 6 characters.
  - Password Must have at least 1 uppercase, 1 lowercase letter and 1 number

b) Login User: This endpoint allows user to log in the webservices, returning the necessary token for other methods. The **POST** endpoint to be called is http://localhost:5000/api/users/login . Is necessary to send a body parameter in JSON with the email and password for the user.

Example Value:

```
{
   "password":"1aAs24",
   "email":"test@test.com"
}
```

FIELD	TYPE	DESCRIPTION
Password	String	The password must match with the one used for the registration

Email	String	The email must have the correct format, can't be repeated nor null. If it doesn't exist, an error will be thrown.

RESPONSES:

- Status 200: Login Successfully.
- Status 403: The program will return a message with the error founded. (see field's descriptions). Also, a message in terminal will appear with the same error.
  - User not found: the user to log don't exist on database.
  - Invalid Password: password for login is incorrect.

c) Get all Users: This endpoint returns all the users that exist in the database. The **GET** endpoint to be called is http://localhost:5000/api/users . Is necessary to send a header key called Authorization with the token generated when the user logs in.

Example Value:

RESPONSES:

- Status 200: Return the list of all users in JSON.

FIELD	TYPE	DESCRIPTION
_id	String	Unique id for all the users
Email	String	Email used for registration
password	String	Encrypted password for the user
_ _v	Int	Generated automatically from database

- Status 403: The program will return a message with the error founded. (see field's descriptions). Also, a message in terminal will appear with the same error
  - Missing Bearer Auth (JWT): Header authorization with Bearer token is missing on call.

d) Get one User by Id: This endpoint returns one user from database by giving his id in URL params. The **GET** endpoint to be called is http://localhost:5000/api/users/:id . Is necessary to send a header key called Authorization with the token generated when the user logs in.

Example Value:

RESPONSES:

- Status 200: Return information of the user in JSON.

FIELD	TYPE	DESCRIPTION
_id	String	Unique id for all the users
Email	String	Email used for registration
password	String	Encrypted password for the user
_ _v	Int	Generated automatically from database

- Status 403: The program will return a message with the error founded. (see field's descriptions). Also, a message in terminal will appear with the same error
  - Missing Bearer Auth (JWT): Header authorization with Bearer token is missing on call.

e) Delete a User: This endpoint deletes one user from database by giving his id in URL params. The **DELETE** endpoint to be called is http://localhost:5000/api/users/delete/:id . Is necessary to send a header key called Authorization with the token generated when the user logs in.

Example Value:

RESPONSES:

- Status 200: User deleted Successfully

- Status 403: The program will return a message with the error founded. (see field's descriptions). Also, a message in terminal will appear with the same error
  - Missing Bearer Auth (JWT): Header authorization with Bearer token is missing on call.
  - No user to delete: no user with the id given was founded.

3.1.2.- LISTFAV METHODS:

a) Get all ListsFavs for all Users: This endpoint lists all list of favorites for all the users in the database. The **GET** endpoint to be called is http://localhost:5000/api/favs . Is necessary to send a header key called Authorization with the token generated when the user logs in.

Example Value:

RESPONSES:

- Status 200: Returns a JSON with all the ListFavs.

FIELD	TYPE	DESCRIPTION
_id	String	Unique id for the ListFav
user_id	String	Id of the user to whom the list belongs
name	String	The name of the list of favorites
favs	Array of Objects	An array of objects with the title, description and link from the fav
_ _v	Int	Generated automatically from database

- Status 403: The program will return a message with the error founded. (see field's descriptions). Also, a message in terminal will appear with the same error.
    - Missing Bearer Auth (JWT): Header authorization with Bearer token is missing on call.

b) Get all ListsFavs by User: This endpoint lists all list of favorites for one user by giving his id in URL params. The **GET** endpoint to be called is http://localhost:5000/api/favs/:id . Is necessary to send a header key called Authorization with the token generated when the user logs in.

Example Value:

RESPONSES:

- Status 200: Returns a JSON with all list for one user.

FIELD	TYPE	DESCRIPTION
_id	String	Unique id for the ListFav
user_id	String	Id of the user to whom the list belongs
name	String	The name of the list of favorites
favs	Array of Objects	An array of objects with the title, description and link from the fav
_ _v	Int	Generated automatically from database

- Status 403: The program will return a message with the error founded. (see field's descriptions). Also, a message in terminal will appear with the same error.
    - Missing Bearer Auth (JWT): Header authorization with Bearer token is missing on call

c) Get One ListFav by list_id: This endpoint returns one list by giving his id in URL params. The **GET** endpoint to be called is

http://localhost:5000/api/favs/singlelist/:id . Is necessary to send a header key called Authorization with the token generated when the user logs in.

Example Value:

RESPONSES:

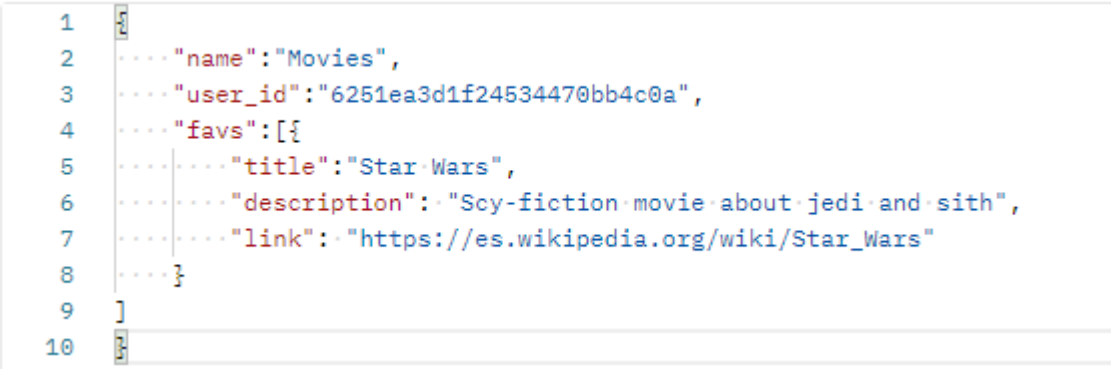
- Status 200: Returns a JSON with the list searched.

FIELD	TYPE	DESCRIPTION
_id	String	Unique id for the ListFav
user_id	String	Id of the user to whom the list belongs
name	String	The name of the list of favorites
favs	Array of Objects	An array of objects with the title, description and link from the fav
_ _v	Int	Generated automatically from database

- Status 403: The program will return a message with the error founded. (see field's descriptions). Also, a message in terminal will appear with the same error.
  - Missing Bearer Auth (JWT): Header authorization with Bearer token is missing on call

d) Create a new List for a User: This endpoint creates one listfav for a user by giving his id in URL params. The **POST** endpoint to be called is http://localhost:5000/api/favs/create/:id . Is necessary to send a header key called Authorization with the token generated when the user logs in.

Example Value:



```
{
    "name":"Movies",
    "user_id":"6251ea3d1f24534470bb4c0a",
    "favs":[{
        "title":"Star Wars",
        "description": "Scy-fiction movie about jedi and sith",
        "link": "https://es.wikipedia.org/wiki/Star_Wars"
    }
]
}
```

FIELD	TYPE	DESCRIPTION
name	String	The name of the list to be created, cannot be repeated from another list who already exists.
user_id	String	The user id which the new ListFav belongs
favs	Array of Objects	An array of objects with the title, description and link from the fav.

RESPONSES:

- Status 201: Returns a JSON with the list created.

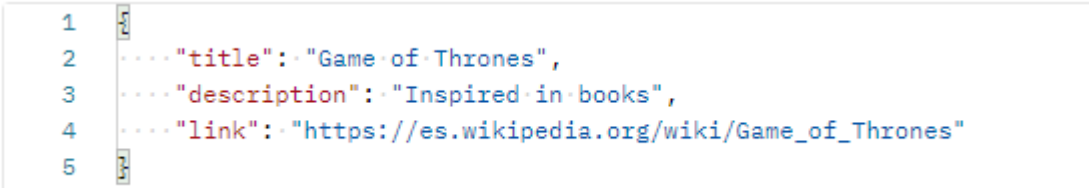
FIELD	TYPE	DESCRIPTION
_id	String	Unique id for the ListFav
user_id	String	Id of the user to whom the list belongs
name	String	The name of the list of favorites
favs	Array of Objects	An array of objects with the title, description and link from the fav.
_ _v	Int	Generated automatically from database

- Status 403: The program will return a message with the error founded. (see field's descriptions). Also, a message in terminal will appear with the same error.
    - Missing Bearer Auth (JWT): Header authorization with Bearer token is missing on call

e) Add items to a List: This endpoint allows user to edit or add favorites to an existing listfav for a user by giving his the list id in URL params. The **PUT** endpoint to be called is http://localhost:5000/api/favs/update/:id . Is necessary to

send a header key called Authorization with the token generated when the user logs in.

Example Value:

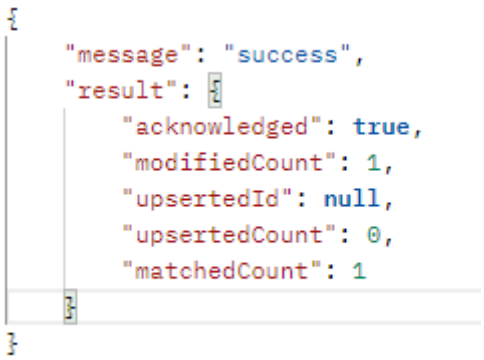


FIELD	TYPE	DESCRIPTION
title	Array	The array of titles of the favorite things the user wants to add to the list.
description	Array	The array of descriptions of the favorite things the user wants to add to the list.
link	Array	The array of links of the favorite things the user wants to add to the list.

RESPONSES:

- Status 200: Returns a JSON with modifiedCount and matchedCount

Ex:



- Status 403: The program will return a message with the error founded. (see field's descriptions). Also, a message in terminal will appear with the same error.
  - Missing Bearer Auth (JWT): Header authorization with Bearer token is missing on call

f) **Delete a ListFavs**: This endpoint deletes one ListFav from database by giving his id in URL params. The **DELETE** endpoint to be called is http://localhost:5000/api/favs/delete/:id . Is necessary to send a header key called Authorization with the token generated when the user logs in.

Example Value:

RESPONSES:

- Status 200: ListFav deleted Successfully

- Status 403: The program will return a message with the error founded. (see field's descriptions). Also, a message in terminal will appear with the same error
  - Missing Bearer Auth (JWT): Header authorization with Bearer token is missing on call.
  - No list to delete: no list with the id given was founded.

4.- TESTING

For testing, call in terminal the script:

npm run server-test

Test database will be called instead of the dev database. Open another terminal in parallel and run the script:

npm run test

for running all tests. All endpoints have been tested with Jest.