Please create a git repository on GitHub and work with it during the whole development process. Commit history should be clear. You should approach this task keeping in mind all good practices, clean code, and readability. Start with reading all points of the task to have a complete overview of the requirements.

To complete this task please check https://swapi.dev documentation.
This API returns resources from the Star Wars universe. Every hero (/people endpoint) has his own homeworld, films, species, vehicles, and starships which are also available through API.
Example:
https://swapi.dev/api/people/1

Create Node.js API (REST or GraphQL) with features:

1. Provide endpoints to return resources:
    a. films - should return all films
    b. species - same, but species
    c. vehicles - same, but vehicles
    d. starships - same, but starships
    e. planets - same, but planets

    Endpoints should provide an option to paginate (check API results, page size doesn't have to be controlled) and filter returned data.
2. Users can also get resources, through API, with a specific id.
3. Cache mechanism - every resource pulled from the Star Wars API should be cached for 24 hours. Every next request for a specific resource should firstly check if it exists in the cache. If so, the server should return the object from the cache instead of requesting the *Star Wars API*. Utilize the chosen database for this task.
4. Each film entry returned from the **/films/** API has an opening_crawl property which contains a short film plot description. Write an endpoint that will return:
    A. an array of pairs of unique words from all films openings paired with their number of occurrences in the text. Words should not be empty and should be separated by space or any number of consecutive control characters (i.e carriage return, line feed).
    B. a name of a character from the **/people** API that appears the most often across all of the openings of the film (or an array of names if there are more characters with the same number). We are interested only in exact name matches excluding eventual control characters in between when the name is longer than a single word.
5. At least some parts of the application should be appropriately tested. The tests can either use real API or mocked responses. Tests code should be clean and non-redundant.
6. Every endpoint should be well documented. You should choose the appropriate form of documentation yourself.
7. Developers should be able to run API on their local environment with Docker Compose.

Stack:

- API - REST / GraphQL - (preferred GraphQL)
- Database - SQL / noSQL - your choice
- JavaScript/TypeScript (preferred TypeScript)
- NestJS preferred, but you can use any framework/library
- Jest, or other test framework/library
- Docker
- Docker Compose