emax.digital scheduled crawler mvp challenge

PLEASE IMPLEMENT EVERYTHING IN TYPESCRIPT

Preferred would be an implementation on NestJS

1. Requirements:

   You need to have `docker` and `docker-compose` installed.

2. Start the application

   You can start the application by running `docker-compose up` in the root folder. This will start the two servers `engine`, `crawler` and the message broker `RabbitMQ` and automatically install all necessary dependencies, including npm libaries. You will see all logs in the terminal running the application with `docker-compose up`.

   Hot reload is enabled for `cralwer` and `engine`, meaning when you save a file in the project, the server or script will restart automatically.

   If you want to install additional npm libaries, use docker-compose and execute the following command inside the root folder:

   For engine: `docker-compose exec engine npm i lodash`

   For crawler: `docker-compose exec engine npm i lodash`

3. GET endpoint `get-title` for `engine`

   Create an endpoint in `engine` with the path `get-title` that accepts GET requests (use the pre-installed npm libary express). The endpoint should accept one query param: `asin`. An `asin` is a unique identifier of amazon for a specific product (example `asin`: `B000RL5C0K`).

   You can access `engine` through the following url: `http://localhost:3030`.

4. Send task from `engine` to `cralwer` using RabbitMQ

   When receiving a GET request in `get-title` send a task with the `asin` to a queue in `RabbitMQ` (use the pre-installed npm libary `amqplib`).

   The `crawler` should listen to this queue and receive the task with the `asin`. In the context of `RabbitMQ` this means that `engine` is the `publisher` and `crawler` is the `consumer`.

   From within the application you can connect to `RabbitMQ` using following url: `amqp://queue`

   Without finishing 2.: Send the task to the queue when the script starts without using the endpoint.

5. Scrape title of detail page of given `asin` on amazon

   When receiving a task from RabbitMQ in `crawler`, the `crawler` should visit the detail page according the the `asin` and fetch the title (use the pre-installed npm libary puppeteer for this). There is already a function called `getBrowser` to initialize the browser for scraping. Use console.log() to return the title. The url can be built like this: `https://amazon.de/dp/${asin}`.

   Without finishing 3.: Scrape the title of any asin when the script starts without consuming from RabbitMQ.

6. Scheduled scraping

   Add a new enpoint `get-title-scheduled` that accepts 2 arguments, `asin` and `crontab`. When a request is posted to the endpoint, start a *schedule* that gets the title of the amazon detail page (like above) according to the given crontab.

   For example: This is posted to the `get-title-scheduled` endpoint:

   ```
   {"asin": "B000RL5C0K", "crontab": "0 * * * *"}
   ```

   In this case, the crawler should get the title of the amazon detail page for `B000RL5C0K` every hour.

   Add another endpoint to stop this specific *schedule*.

7. To think about (no code needed)

   This service will run in a stateless kubernetes cluster. What are the implications and possible issues? How will the service behave when scaling it horizontally?