WriterPad

Things to keep in mind

1. Time to finish this assignment is max. 3 days. It should not take more than 10 hours to finish the application.
2. Please don't post the solution on public Github, Bitbucket, Gitlab repositories.
3. In case you have any doubts feel free to reach out or email for clarification.
4. Please proceed in story-wise manner so if you find yourself running out of time you can at least submit the already completed stories.

Evaluation criteria

Please keep the following in mind:

1. We are expecting solution built using Java 8 and Spring Boot 2.x.
2. This application will need a backing database - you can use either H2 (or another in-memory DB) or feel free to use any other SQL/NoSQL DB. Please provide details in README.md on your choice of database and how to connect to it. For relational DBs, kindly provide schema creation script and mention the data model.

Your code will be evaluated on following parameters:

- Correctness
- Code quality and cleanliness
- Good use of OOPS concepts
    - Extensible – easily adaptable to future enhancements
    - Highly cohesive and loosely coupled
    - Make use of design pattern wherever possible
- Automation testing
    - Unit / integration test should be present
- REST API Design
- Basic DevOps knowhow. We expect that you will submit Dockerfile with your project.
- A README.md that clearly mentions steps needed to build and run the application.

Requirements

You have to create a REST API service which implements the following stories.

Story 1: REST API to create an article

`POST /api/articles`

Example request body:

```
{
    "title": "How to learn Spring Booot",
    "description": "Ever wonder how?",
    "body": "You have to believe",
    "tags": ["java", "Spring Boot", "tutorial"]
}
```

Required fields: `title`, `description`, `body`

Optional fields: `tags` as an array of Strings, and `featuredImage`

Tags should be saved as lowercase.

Response

1. It will return response code 201 if article is created successfully
2. It will return 400 if validation failed
3. It will return 500 if anything else happened
4. It will return created article as shown below.

```
{
    "id": "a98fd991e69a",
    "slug": "how-to-learn-spring-boot",
    "title": "How to learn Spring Booot",
    "description": "Ever wonder how?",
    "body": "You have to believe",
    "tags": ["java", "spring-boot","tutorial"],
    "createdAt": "2019-11-24T03:22:56.637Z",
    "updatedAt": "2019-11-24T03:48:35.824Z",
    "favorited": false,
    "favoritesCount": 0,
}
```

Story 2: Update an article

```
PATCH /api/articles/:slug-uuid
```

The body of the patch is shown below:

```
{
    "title": "How to learn Spring Boot by building an app",
}
```

All the fields are optional.

The `slug` also gets updated when article is changed.

Returns the updated article.

```
{
    "id": "a98fd991e69a",
    "slug": "how-to-learn-spring-boot-by-building-an-app",
    "title": "How to learn Spring Booot",
    "description": "Ever wonder how?",
    "body": "You have to believe",
    "tags": ["java", "spring-boot","tutorial"],
    "createdAt": "2019-11-24T03:22:56.637Z",
    "updatedAt": "2019-11-25T03:48:35.824Z",
    "favorited": false,
    "favoritesCount": 0,
}
```

Story 3: Get an article

`GET /api/articles/:slug-uuid`

This returns single article

```
{
    "id": "a98fd991e69a",
    "slug": "how-to-learn-spring-boot-by-building-an-app",
    "title": "How to learn Spring Booot",
    "description": "Ever wonder how?",
    "body": "You have to believe",
    "tags": ["java", "spring-boot","tutorial"],
    "createdAt": "2019-11-24T03:22:56.637Z",
    "updatedAt": "2019-11-25T03:48:35.824Z",
    "favorited": false,
    "favoritesCount": 0,
}
```

Story 4: List articles

`GET /api/articles`

This should support pagination.

Story 5: Delete an article

```
DELETE /api/articles/:slug-uuid
```

Story 6: Find time to read for an article

You have to define a REST endpoint that will find time to read for an article given its id

```
GET /api/articles/:slug-uuid/timetoread
```

This should return following response

```
{
"articleId": "slug-uuid",
"timeToRead": {
 "mins" : 3,
 "seconds" : 50
   }
}
```

The logic to calculate time to read is `total number of words / speed of average human`

The speed of average human should be configurable.

Story 7: Generate Tag based metrics

You have to define a REST endpoint that will provide all tags, provided in all articles, with their occurance.

```
GET /api/articles/tags
```

This should return following response

```
[{ "tag" : "java", "occurence" : "10"},
{ "tag" : "spring", "occurance" : "5"},
{ "tag" : "tutorial", "occurance" : "2"}
]
```

You need to look into the tags marked in an article. Tags metrics should not differentiate based on casing. 'Java','java', 'JAVA', 'JaVA' etc, should be considered one for determining count.

Story 8: A user should not be able to submit article with same body twice

In this story, we are implementing a simple copy paste detector. If a same or different user tries to submit article with the body that already exist then you should return bad request to the user.

For example if user `u1` creates an article with body `hello, world` and then `u1` or `u2` tries to create another article article with body `hello, world` then HTTP 400 should be returned.

You have to do this check in both create and update API

You can use https://github.com/rrice/java-string-similarity library to find the text similarity. If two strings are similar i.e. their similarity score is greater than 70% then you should return 400.