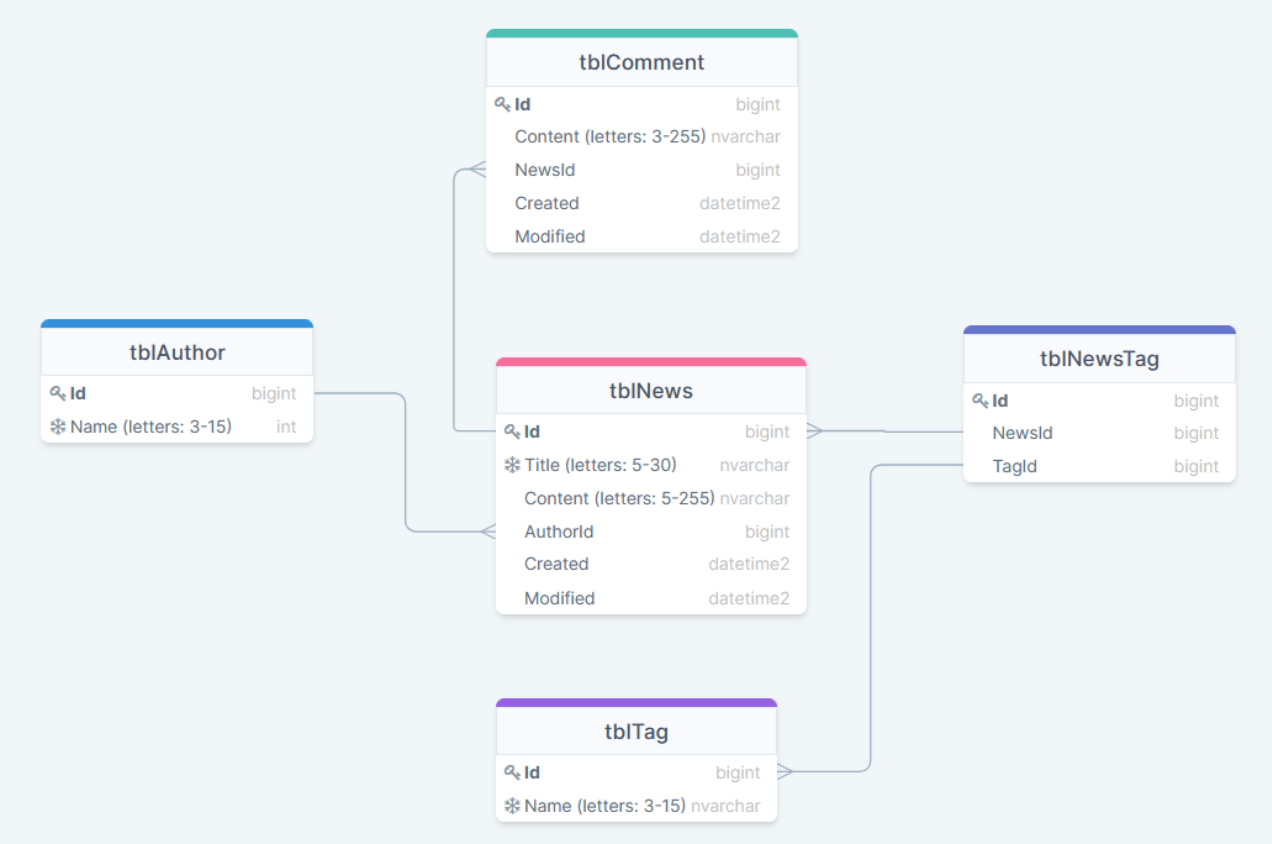
News Management

Business requirements:

1. Develop web service for News Management system with the following entities:



- ❄ – unique value
- All *Name*, *Title* and *Content* fields are required
- *Created*, *Modified* – have ISO 8601 format (wiki: ISO 8601). Example: 2018-08-29T06:12:15.156. More discussion here: stackoverflow: how to get iso 8601.

2. The system should expose REST APIs to perform the following operations:

   – CRUD operations (what is CRUD) for News and Comment. If new tags and authors are passed during creation/modification – they should be created in the DB. For update operation – update only fields, that pass in request, others should not be updated. Batch insert is out of scope.
   – Get News:
      - get all
      - by Id
      - search (all params are optional and can be used in conjunction):
         - by tag names and tag ids (many tags)
         - by author name (one author)
         - by part of Title

        - by part of Content
    - sort by Created, Modified Asc/Desc. Default: Created Desc
- Get Comments:
    - by News Id (URL example: /news/{newsId}/comments)
    - by Id
    - sort by Created, Modified Asc/Desc. Default: Created Desc
- CRUD operations for Tag, and Author
- Get Tags and Authors:
    - get all
    - by Id
    - by part Name
    - by News Id (URL example:

      /news/{newsId}/tags - *should return tags collection*

      /news/{newsId}/authors - *should return 1 author*)
- Get authors with amount of written news. Sort by news amount Desc.
- Optional: load Content of News through separate operation

2. Pagination should be implemented for all GET endpoints. Please, create a flexible and non-erroneous solution. Handle all exceptional cases.

3. Request URL, params and body should be validated.

Return error response if user:

- try to call nonexistent URL
- passes null or empty value into required field or doesn't specify it at all
- passes value with invalid format
- Optional: passes invalid param names or invalid constant values. For example:
    - instead of GET */news?title=breaking* user accidently wrote */news?tile=breaking*
    - instead of GET */news?sortBy=Created|ASC* user accidently wrote */news?sortBy=Cread|ASC* or */news?sortBy=Created|AC*

Register should be ignored for all constant values(e.g. asC, deSc) and all existed param names(e.g. */news?tITle=breaking*).

If there are empty characters at the beginning or end of the value, these characters should be ignored during execution.

4. Provide versioning support.

5. Support HATEOAS on REST endpoints.

6. Use Swagger to document RESTful API.

General requirements:

1. Code should be clean and should not contain any "developer-purpose" constructions.

2. App should be designed and written with respect to OOD and SOLID principles.

3. Clear layered structure should be used with responsibilities of each application layer defined.

4. All business logic should be written in the service layer: mapping model to DTO and vice versa, transactions, validation, etc.

5. JSON should be used as a format for client-server communication messages. Optional: support XML.

6. Convenient error/exception handling mechanism should be implemented: all errors should be meaningful and **localized** (En/Ru) on backend side. Example: handle 404 error:

```
HTTP Status: 404
response body
{
“errorMessage”: “Requested resource not found (id = 55)”,
“errorCode”: 40401
}
```

where *errorCode* is your custom code (it can be based on http status and requested resource - news or comment, etc.).

7. Abstraction should be used everywhere to avoid code duplication.

8. Application should be covered with unit tests.

9. Use REST-Assured for testing REST API.

Application requirements:

1. Gradle, latest version. Multi-module project.

2. Application packages root: com.mjc.school.

3. Hibernate should be used as a JPA implementation for data access.

4. Spring Framework & Spring Boot, the latest version.

5. Spring Transaction should be used in all necessary areas of the application.

6. Java Code Convention is mandatory.

Demo:

1. Application should be deployed before demo.

2. Demonstrate API using Postman tool. Prepare for demo Postman collection with APIs.

3. Candidate should be able to answer theoretical and practical questions during demo session.