**Система управления новостями**

Разработать RESTful web-service, реализующей функционал для работы с системой управления новостями.

Основные сущности:

* **news** (новость)содержит поля: id, time, title, text и comments (list).
* **comment** содержит поля: id, time, text, username и news\_id.

Требования:

1. Использовать Spring Boot 3.x, Java 17, Gradle и PostgreSQL.
2. Разработать API согласно подходам REST (UI не надо):

* CRUD для работы с новостью
* CRUD для работы с комментарием
* просмотр списка новостей (с пагинацией)
* просмотр новости с комментариями относящимися к ней (с пагинацией)
* /news
* /news/{newsId}
* /news/{newsId}/comments
* /news/{newsId}/comments/{commentsId}
* полнотекстовый поиск по различным параметрам (для новостей и комментариев)

Для потенциально объемных запросов реализовать постраничность

1. Разместить проект в любом из публичных git-репозиториев (Bitbucket, **github**, gitlab)
2. Код должен быть легко читаемый и понятный, с использованием паттернов проектирования
3. Реализовать на основе Spring @Profile (e.g. test & prod) подключение к базам данных.
4. Подключить liquibase:

- при запуске сервиса накатываются скрипты на рабочую БД (генерируются необходимые таблицы из одного файла и наполняются таблицы данными из другого файла, 20 новостей и 10 комментариев, связанных с каждой новостью

- при запуске тестов должен подхватываться скрипт по генерации необходимых таблиц + накатить данные по заполнению таблиц (третий файл)

1. Создать реализацию кэша, для хранения сущностей. Реализовать два алгоритма LRU и LFU. Алгоритм и максимальный размер коллекции должны читаться из файла application.yml. Алгоритм работы с кешем:

* GET - ищем в кеше и если там данных нет, то достаем объект из dao, сохраняем в кеш и возвращаем
* POST - сохраняем в dao и потом сохраняем в кеше
* DELETE - удаляем из dao и потом удаляем в кеша
* PUT - обновление/вставка в dao и потом обновление/вставка в кеше.

1. Весь код должен быть покрыт юнит-тестами (80%) (сервисный слой – 100%)
2. Реализовать логирование запрос-ответ в аспектном стиле (для слоя Controlles), а также логирование по уровням в отдельных слоях приложения, используя logback
3. Предусмотреть обработку исключений и интерпретацию их согласно REST (см. <https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc>) осталось в коммент
4. Все настройки должны быть вынесены в \*.yml
5. Код должен быть документирован @JavaDoc, а назначение приложения и его интерфейс и настройки должны быть описаны в README.md файле
6. Использовать Spring REST Docs или другие средства автоматического документирования (например asciidoctor <https://asciidoctor.org/docs/asciidoctor-gradle-plugin/> и т.д) и/или Swagger (OpenAPI 3.0)
7. Использовать testcontainers в тестах на persistence layer (для БД)
8. Написать интеграционные тесты
9. Использовать WireMock в тестах для слоя clients (разбиение на микросервисы)
10. Использовать Docker (написать Dockerfile – для spring boot приложения, docker-compose.yml для поднятия БД и приложения в контейнерах и настроить взаимодействие между ними)
11. \* Подключить кэш провайдер Redis (в docker) (в случае реализации, использовать @Profile для переключения между LRU/LFU и Redis)
12. \* Spring Security:

* API для регистрации пользователей с ролями admin/journalist/subscriber
* Администратор (role admin) может производить CRUD-операции со всеми сущностями
* Журналист (role journalist) может добавлять и изменять/удалять только свои новости
* Подписчик (role subscriber) может добавлять и изменять/удалять только свои комментарии
* Незарегистрированные пользователи могут только просматривать новости и комментарии

Создать отдельный микросервис с реляционной базой (postgreSQL) хранящей  
информацию о пользователях/ролях. Из главного микросервиса (отвечающего за  
новости) запрашивать эту информацию по REST с использованием spring-cloud-  
feign-client.

1. \* Настроить Spring Cloud Config (вынести в отдельный сервис и настроить разрабатываемый сервис на получение их в зависимости от профиля)
2. \* Реализацию логирования п.9 и обработку исключений п.10 вынести в отдельные  
   spring-boot-starter-ы.
3. \*\* Сущности веб интерфейса (DTO) должны генерироваться при сборке проекта из .proto файлов (см. <https://github.com/google/protobuf-gradle-plugin>)

“\*” – необходимо минимум выполнить два задания со звёздочкой, больше заданий будет существенным плюсом. К этим пунктам лучше приступать после качественного решения базовых задач с применением принципов SOLID, декларативных подходов, оптимальных алгоритмов.

Отправлять в [форму](https://forms.gle/7qaNherfLYAgfZmUA)