bitweb

Full-Stack arendaja proovitöö

Word cloud

The goal of the given assignment is to create a text processing application. The main usecase of the application is to accept different texts, count the words within the text and persist these counts in the persistent storage. The user should be able to retrieve the list with corresponding word counts for any given text and possibly use that data to generate a word cloud image. User interface should be clean and nice.

Users can submit any text file with up to a 100MB in size. Contents of these files should be sent to the message queue for further processing by the workers. As a result the submitter should receive an identifier which can be later used to retrieve the statistics.

Although the use case is simple, it is important to understand that a given application will be used by many users and each of them is expected to upload many large texts continuously. Therefore, it makes sense to take into consideration the possibility for scaling. In order to make application more flexible in that regard, the architecture should be microservice oriented. The application architecture is described below.

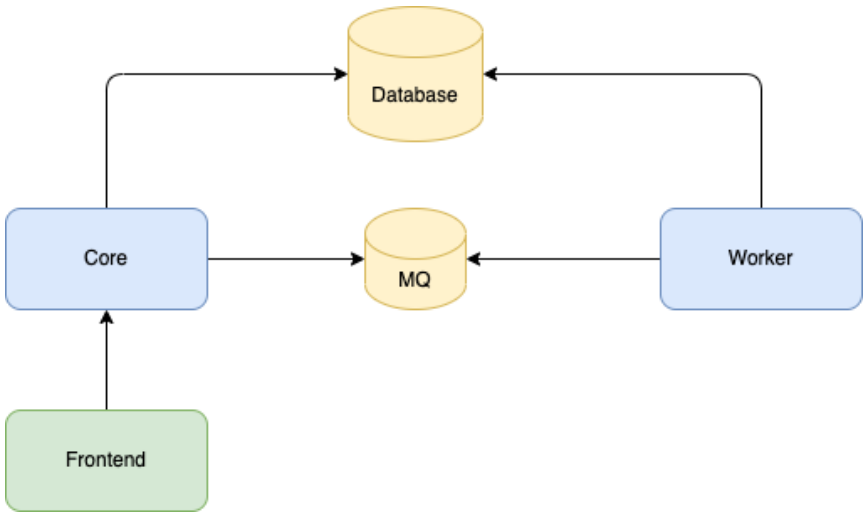

Figure 1. Component diagram of the application

Note: There can exist multiple instances of Workers and API applications depending on the workload, but in given assignment we will limit both to one instance.

bitweb

Use cases

- User can upload a text file up to 100MB in size. Upon upload, user is given an identifier.
- User can retrieve the JSON formatted list of words and corresponding word counts with provided identifier.
- Bonus: user can see a word cloud image generated from that text.
- Bonus: user can identify the status of the processing (pending, in progress, complete).

Technical Requirements

- Frontend application should be implemented in Angular or React framework. Given component is the user interface. Bonus points are provided if frontend can generate a word cloud with some existing library.
- Core application should provide HTTP REST/JSON API interface for frontend to use. Given app should also convert the text to MQ message(s) and send them to MQ for workers to process. How large of a payload to capsulate in message is up to you to decide. Given application should be implemented with Spring Boot Framework.
- Worker application is responsible for processing the text and saving the statistics to database. Text processing is capsulated to a separate application because text processing can be quite resource intensive as user might want to do some filtering and data cleanup. Consider omitting some words that repeat a lot and have no meaning (“and”, “or” “the”, “a”). User might want to filter out outliers, e.g., words that exists within text only once or twice. User might want to detect typos and fixing them. Providing these features is not mandatory but provides a bonus. Given application should be implemented with Spring Boot Framework.
- DB is persistence storage service - database. It is up to you to choose which database to use.
- MQ is Message Queue Broker should be RabbitMQ.
- For dependency management, Gradle should be used. For any dependency, latest stable version should be used.

- For database integration, JPA/Hibernate (or any other ORM) solution should be used.

Infrastructure requirements

- All source code should be stored in GIT version control. Each component should have separate repository.
- For GIT repository GitHub is preferred.
- Every component should run in Docker.
- Reuse existing images from DockerHub.
- Whole application should be deployed with a single docker-compose command.
- Database schema should be created with Flyway by Core component.

Results

- Utilize best practices and design patterns even if there is not much codebase
- For results, please provide GIT repositories.