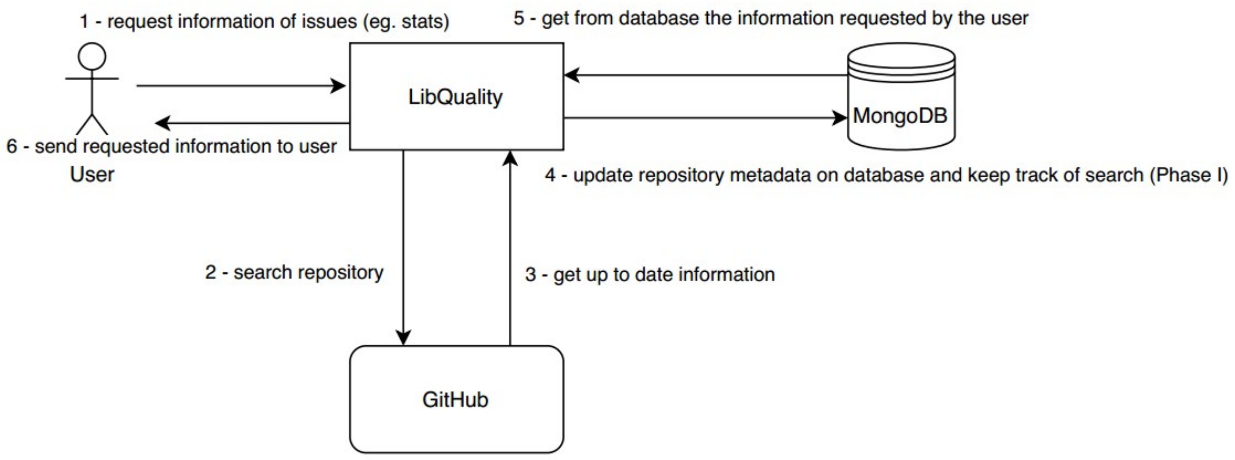
LibQuality

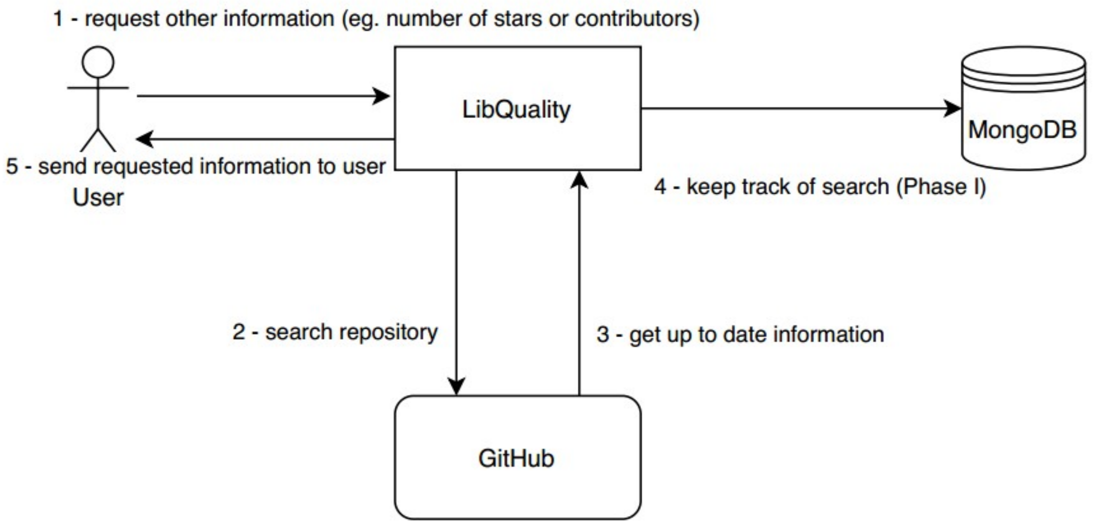
Architecture:

The architecture of LibQuality is composed of an Express service, that communicates with GitHub using its API and with a local MongoDB database. The images below present the architecture.

The first image presents the flow of information for requests about issues. In this situation, LibQuality updates information in MongoDB about the repository problems; and then processes the user's query by returning the requested information.


1 - request information of issues (eg. stats)
5 - get from database the information requested by the user
LibQuality
MongoDB
6 - send requested information to user
User
4 - update repository metadata on database and keep track of search (Phase I)
2 - search repository
3 - get up to date information
GitHub

The second image presents the flow for other information (e. g. number of stars, contributors). For these queries, it is not necessary to save information on MongoDB besides the track of searches because the data retrieved from GitHub already contains the information requested by the user.


1 - request other information (eg. number of stars or contributors)
LibQuality
MongoDB
5 - send requested information to user
User
4 - keep track of search (Phase I)
2 - search repository
3 - get up to date information
GitHub

Database:

MongoDB was chosen because it is a document storage database, having the flexibility to save the information retrieved from GitHub API without creating complex schemas. It also is performant, allowing to create new indexes when necessary. This characteristic is important because as the project evolves it would be required to index new fields inside the documents.

Activities:

The definition of database schema and indexes was the first action performed. Then, I developed the service *repository.js* to retrieve information from GitHub using its API. The functions of *repository.js* are flexible enough to be used in many situations, besides the ones of Phase I, II and III. After it, I developed *issues.js* service, thinking about the performance when saving and updating information on MongoDB (that is why *bulkWrite* was used, saving and updating all information at once) and then I wrote tests to verify if this service is correctly working. Finally, the routes of LibQuality were developed using Express and Docker-compose was configured to deploy this system.