General Overview:

Using this code base, we are trying to calculate *KPIs* (Key performance indicators) using github as a data source. So, for this we first do data crawling. We do this using mongoDB (a non relational db), it is a *key-value* collection of documents. They use *JSON objects* in a DB, where *collections* are just like *tables* and *documents* are like table entries (*columns*) in SQL. The database name here is *"Github mirror"*. As a database scheme there is one entry for the *repository*.

In this *repository* collection, we store the *owner* name, *repo* and then the links (via objectid) to different data (like *diffs* which contains pullrequest related data, *issues* which basically are tickets raised in a repository, *releases* which contains information related to the same etc.)

The basic idea is that there is one data model where links to actual data needed to calculate the KPIs are stored. The repository can be considered as a big lookup table for the data.

Repository	Collects data with owner and repo name, this data (object id) can be linked to *diffs, issues, releases* etc. These id are a *lookup table* for further information.
Diffs	Has following entries: *RepositoryFiles*: a lookup table (describe the state of the repository at that time when the pull request is being made) *Pullrequestfiles*: files which were actually changed. *PullRequest*: information like title and url of pullrequest
Issuewithevents	Has following entries: *Issue*: objectId for actual tickets (i.e., issue collection) *IssueEventtypes*: Events related to that particular issue
Issue	Has following entries: It has data related to the *issue* (ticket), important ones are the *assignee* information, when issue was *created*, *updated* and *closed*
IssueEventtypes	Has information related to the *issue*, important one is the *event* entry to find about the *event*, i.e., if the event is *assigned*, *labelled* etc
Releases	We need this to get information regarding when pullrequest was created to calculate workinprogress.

In order to understand different sections of *Github API*, here are some of the links to understand the information we can gather from each model:

https://docs.github.com/en/rest/reference/issues

https://docs.github.com/en/rest/reference/repos

https://docs.github.com/en/developers/webhooks-and-events/issue-event-types

https://docs.github.com/en/rest/reference/repos#releases

Some repositories we used:

1. repo: rest.js, owner: octokit
   This is a small repository which is extracted in few minutes, however, to analyses ticketing data, *releases information etc is missing*

2. Preferred - Link : https://github.com/soot-oss/soot
   a huge repository, with about 700+ pull request, with 4000+ files, 1000+ tickets

3. https://github.com/corona-warn-app/cwa-website - no releases data is available in this package.

4. Preferred: https://github.com/corona-warn-app/cwa-app-android

Code:

We chiefly use a nodejs serverr, with database added as a docker file,

In the docker-compose file we can set the environment variables for the DB.

Note: We can to make sure *DB_User & DB_User_password* matches the *.env.local file*.

For further information also refer to the readme.

The server is primarily split into two main parts. There is *github_api* and *database.*

The *github_api.controller.ts* has the end points for the servers. So we have currently, *diffs, issues, releases, statistics* end points. All these are reachable over http://localhost:8081/db/githubMirror/. The *repositoryidentifierDTO* refers to the class with owner and repository as string. We automatically validate the request made to the API endpoint, and see if the body contains the right strings.

github-api.service.ts is where the actual logic is applied where we store and process the data in the repository. For example, we can query pullresquests, issues etc. (*octokit.rest.pulls, octokit.rest.issues*). We use ***octokit*** library to access the github api. We can access the rest api through it. For example, *ocktokit.rest.repos.getcommit* etc. We also have *graphql* access. However, we do not use graphql till now.

After querying the data, we have the *statistic.service.ts* where we do the actual calculation of the KPIs. Here we applying the mongodb aggregation methods/queries. For the calculation of the KPI we use to do some lookup, addfields, groups, match the

data in DB as we want. You can refer to -> https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/ for more methods.

For more information regarding each KPI refer to the comments above each method in the *statistic.service.ts.*

Different database schemas are defined in the *schemas* folder. The classes with different properties are defined, this is what is stored in the DB.

Current State:

So far we have calculated the following:

1. Most files changed. An average of the changes made to the first 100 files.
2. Calculation of the number of files changed together.
   This method gives the count of the filenames that are changed together
   E.g.
   1. pullRequestFiles: [File A, File B, File C, File D]
   2. pullRequestFiles: [File A, File B, File E, File F]
   3. pullRequestFiles: [File A, File B, File X, File Y]
   4. pullRequestFiles: [File C, File D, File Z]

   output: Files A & B changed together 3 times
3. Calculation of the changes in the pullrequest
4. Number of issues with no assignees
5. Calculate the number of open tickets
6. Calculate Avg Number of assignees until the ticket closes
   Calculations involve only tickets which are closed. Find the tickets which are closed, if assignees is null count them, if assignees is not null count number of assignees.
7. Calculate average time until ticket was assigned
8. Calculate the workinProgress. Tickets are assigned to someone and then we count the number of releases that have been made while the ticket was open, i.e., (Average) number of releases until we close the ticket. Only tickets which are 'assigned' and tickets whose 'closed_at' key is not null is taken into account.

Pending:

There is nothing pending as such. I implemented all the KPIs we discussed so far. However, after I created the pull request to the main branch there were some comments and also ToDo remarks related to modification of the KPIs (statistics.service.ts) by Jan after he reviewed the code which have not been addressed yet.