Design-documentation.md

Overview

The application consists of two parts: the backend and the frontend. The frontend is a REACT-based web client that provides the user an easy way to use the application and uses the REST-API from the backend. The backend is a spring boot application where all the business logics are implemented. We use Postgresql to store the data.

Backend

SpringApplication class

This is the main class of this application. To run the server, this main class needs to be run.

Controller class

Refer to: https://github.com/amosproj/amos2022ws04-specitem-database/wiki/Rest-API

Importer classes

In order to handle the information that comes with each document we need a tool that parses it. Therefore, we have implemented the function `processFile()` which reads the whole document and separates the commit information at the beginning and each *Specification Item* (SpecItem). Afterwards, we use the `getSpecItemsFromString()` function to assign the correct values to each of the SpecItem's attributes. This happens by the help of a regular expression which matches the values from the new splitted strings after the document is processed. The same function exists for the commit's data.

File generator

The `filegenerator` package consists of a series of interfaces and the corresponding implementations for the file generation. The main interface, which acts as an entry point for the generation process, is named `FileGenerator` and exposes the methods for creating a file. Each procedure takes the name of the target file and the number of SpecItems to be made. An additional method accepts a parameter that describes the desired number of incomplete SpecItems to be created. This method gives the possibility to create only a specific number of attributes of a SpecItem, which are chosen at random. The generation process follows a set of rules to create files that have a rigid structure. The following block depicts an exemplary file that follows the format:

```
CommitHash: 29ec1a67-b329-
CommitDate: 2022-11-29 16:15:53
CommitMsg: oz8q4u3seczhm1n4tt8j
CommitAuthor: Williams Homenick




Fingerprint: 6lx8knj3353789wy
ShortName: SpecItem_0
Category: Category3
LC-Status: Status4
UseInstead: SpecItem_32
TraceRefs: SpecItem_30,SpecItem_62,SpecItem_58,SpecItem_59,SpecItem_99,SpecIt
LongName: corrupti autem esse sit architecto repudiandae aut
Content: totam]}]magnam][]qui](]ad]ipsam]iure]molestiae]||])]!]{]\t]*]explica
```

On the top of each generated file, there is commit information. All components of a commit but the commit date are randomly generated. The interface `CommitProvider` exposes a single method called `generateCommit` , which creates a `Commit` object populated with the necessary information. After the commit information, a given number of SpecItems is appended. SpecItems are generated via `SpecItemProvider` . The content of each SpecItem is generated via `ContentCreator` . `ContentCreator` generates highly randomized content of various complexity. The created content can span multiple lines and have many different characters. The created structure is written to a .txt file with the help of `WriterService` . The writer service also ensures the correct formatting of a file (number of spaces and new lines).

Service classes

Here, the applications and business logics are implemented.

SpecItemService

- `void savesDocument(String filename)` saves or updates a document in database Every time a specitem in the document is updated, i.e. there is already an existing SpecItem with the same shortName in the database, the system will create a new SpecItem with the same shortName but a different timestamp, which is the timestamp of the current commit. The one with latest timestamp is the most updated one. The reason for this approach is to allow users to view version change histories of SpecItems.
- `List<CompareResult> compare(String id, LocalDateTime t1, LocalDateTime t2)` queries the database for these 2 versions of the specified specitem and compares them. It thows `IllegalArgumentException` if no such specitem exists in database.
- `TagInfo completeTagAdditionProcess(final SpecItem taggedSpecItem, final String newTags)` updates the tags of the taggedSpecItem with the newTags by creating a new version of that SpecItem.
- `List<SpecItem> getListOfSpecItemsByContent(String content, int page)` filters the list of the (current) SpecItems by content
- `List<SpecItem> getListOfSpecItemsByIDAndContent(String shortName, String content)` filters among all versions of the SpecItem (its history) with shortname by content
- `List<SpecItem> getAllSpecItems(int page)` gets the list of all (current) SpecItems on the i-th page. Every page consists of maximal 50 SpecItems.
- `SpecItem getSpecItemById(String shortname)` gets the (current) SpecItem with shortname
- `List<SpecItem> getListOfSpecItemsById(String shortname)` gets all versions (history) of the SpecItem with shortname
- `void deleteSpecItemById(String shortname)` marks the latest SpecItem with shortname as deleted
- `int getPageNumber()` calculates the maximum page number of SpecItem in database.

FileStorageService

`void storeFile(MultipartFile file, String filename)` save a multipart file in /src/main/resources/tmp

SpecitemComparator

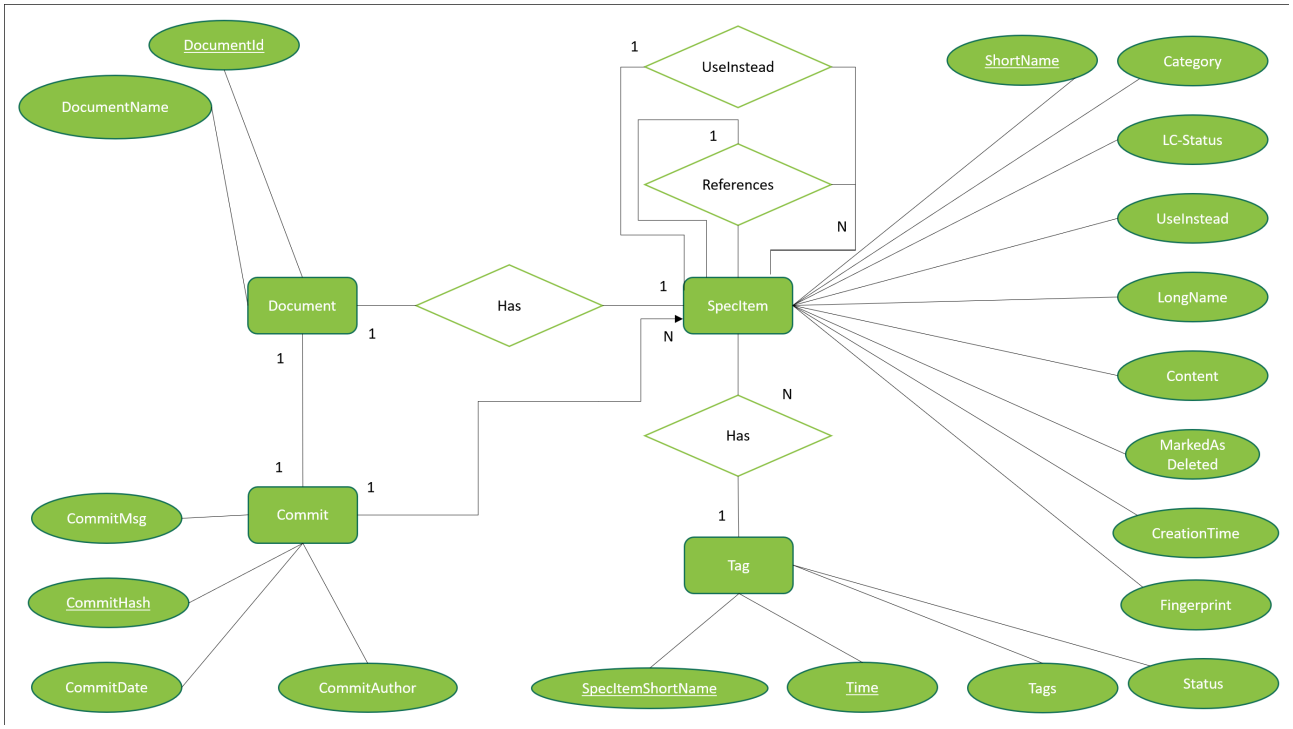
`List<CompareResult> compare(SpecItem s1, SpecItem s2)` compares two versions of a specitem and returns list of columns that are different. `List<CompareResultMarkup> compareMarkup(SpecItem old, SpecItem updated)` compares two versions of a SpecItem and returns the difference as a String with html tag

Models

The models are objects that are to be stored in a database.

- SpecItem: This is the main object that we want to store. The unique key of a SpecItem is the combination of shortName and timeStamp.
  - Tags: The tags are stored in a string, separated by commas. The format of a tag is Key:Value. So the list looks e.g. like "Key1:Value1, Key2:Value2".
- Document: A document contains many SpecItems and user updates or add new SpecItems through it
- Commit: contains commit information and allows versioning

This is the ER-Diagram modeling the objects being stored in the database.



Repositories

Repositories serve as a connector to the database and provide developers with ways to query and save data. Developers can use the already-defined functions from Spring Boot or create their own custom SQL queries. There is one repo for each model.

Configuration

There are web, file and database configurations.

Frontend

We use React Router to navigate between pages and components.

Pages

Please refer to the REST-API documentation for more details about the Http-methods used in this section.

- **main_page**: This component represents the home page (/).
- **specitems_page**: This component represents the page where all SpecItems are shown and filtered (/specitems). It calls a GET(/get/all or /get/cont) method to the server to get a (filtered) list of SpecItems. This component receives exportContext from its parent main_page, which relates to the list of to-be-exported SpecItems in the next component.
- **specitem_page**: This component represents the page where a specific SpecItem is shown. It calls a GET(/get/{id}) method to the server to get a SpecItem. It is also possible to add tags to a SpecItem on this page using a POST request.
- **specitem_history**: This component represents the page where the history of a specific SpecItem is shown. It calls a GET(/get/history/{id}) method to the server to get all versions of a SpecItem. The comparision of two versions can be done and the result is can be fetched by sending GET(/compare) request.
- **export_page**: This component represents the page where users can export saved SpecItems to a downloadable text file (/export). The SpecItems that are saved in specitems_page component are forwarded by its parent main_page to here.

Components

- **CollapseContent**: this component contains full informations about a SpecItem, which can be expanded/hidden by the 'Show' button in the list of specitems on

specitems_page.

- **PageBar**: for the user to navigate between different pages and set the page state accordingly. It calls GET method /pageNumber to get the maximum page number.
- **TagInput**: contains functions to add and delete tag and a view of existing tags belonging to a SpecItem.