Advanced Database Systems

Design and Implementation of a Data-Driven Application

Report of the Coursework

1st April 2022

Aston University

Aston Business School

Advanced Database Systems CS3800

Written by

Lea Mayer

Student Number: 210309311

Modul Leader: Dr. Hai Wang

Lecturer : Dr. Amal Htait

Table of Content

Table of Figures

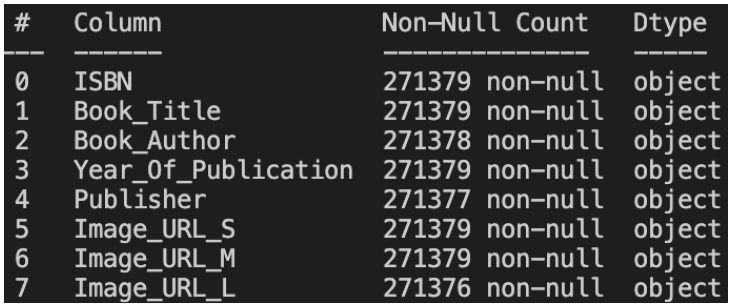
1 Area of Analysis

"BookCrossing is the library for the whole world". It tracks how a book is passed from reader to reader. This creates a lot of data about books and their readers that can be analyzed. The following report describes the analysis of this data and the development of an application with an innovative business value for the BookCrossing company. To satisfy and further retain customers, an important business use case is to provide recommendations for other books that a reader might read. Therefore, a new application is being implemented that could be further developed into a full-fledged novel app for BookCrossing. For now, however, this application will be able to provide 5 recommendations depending on a book title specified by the user of the application.

To get an idea of how the application should be designed and implemented, the data must first be analyzed. The data consists of three CSV files. The review was done using Pandas DataFrames, which provide an easy-to-use data structure for a first direct analysis. To successfully load the data into another database, one needs to know the general structure of the data first.

To load the CSV file into a DataFrame, some adjustments were necessary. The separator, to properly separate rows into columns, is a semicolon. Unfortunately, the values of the dataset also contain semicolons that must be removed before converting to a DataFrame. For example, "&", which refers to an ampersand, has been replaced with “and” to avoid an error. Also, hyphens have been replaced with underscores because they are ambiguous during querying.

The first DataFrame contains information about the books. The primary key for each of the 278858 records is the ISBN, which is unique for each book. There are also many attributes for each book (see Figure 1). It is important to note that there are a total of 271379 records and only the Author, Publisher, and URL columns of the large image contain null values. The null values must be handled in the following data cleaning process. All values are objects.



#	Column	Non-Null Count	Dtype
0	ISBN	271379 non-null	object
1	Book_Title	271379 non-null	object
2	Book_Author	271378 non-null	object
3	Year_Of_Publication	271379 non-null	object
4	Publisher	271377 non-null	object
5	Image_URL_S	271379 non-null	object
6	Image_URL_M	271379 non-null	object
7	Image_URL_L	271376 non-null	object

Figure 1 Book DataFrame

The DataFrame contains 271379 records about a user and information about his age and location. The primary key is the UserID. Over 40% of the age data is missing (see Figure 2).

#	Column	Non-Null Count	Dtype
0	UserID	278858 non-null	int64
1	Location	278858 non-null	object
2	Age	168096 non-null	float64

Figure 2 User DataFrame

The DataFrame book_rating matches a UserID to an ISBN and a book rating between 0 and 10 (see Figure 3). The primary key is the UserID and consists of 1149780 records. It is important for further analysis that the rating table establishes a relationship between the tables about books and users. With similar keys in the other tables, a join operation is possible.

#	Column	Non-Null Count	Dtype
0	UserID	1149780 non-null	int64
1	ISBN	1149780 non-null	object
2	BookRating	1149780 non-null	int64

Figure 3 BookRating DataFrame

The following graph shows the distribution of the BookRatings in the book rating DataFrame (see Figure 4). It depicts that most of the books are rated 0 and the other majority is rated over 5, which leaves the ratings from 1 to 4 low.

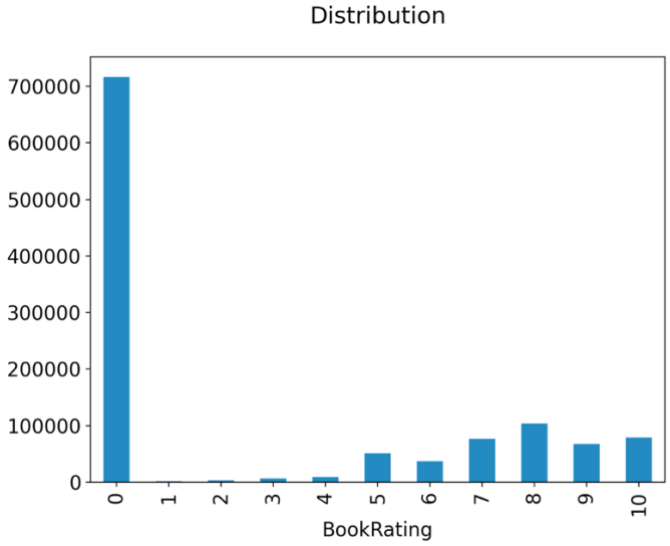


Figure 4 Distribution of the Variable BookRating

The results of this analysis suggest the following. When selecting a suitable database to store the files, the large number of records in each table must be considered. The columns with a large proportion of null values must be handled or excluded in the further design of the application. According to the content of the book rating table, it is possible to link (join) with other tables to create further relationships, since the same keys are used. The distribution of book ratings must be considered when aggregating the data.

In the now-following design part, these limitations and opportunities are taken into consideration.

2 Design

This part considers the findings from the previous analysis part, such as the limitations of the dataset, when selecting a suitable database and creating the data flow concept. The design thus creates the basis for a plan to successfully implement the application to meet the requirements.

2.1 Choosing a Database

First, it needs to be decided where to store the data. “Pandas provide data structures for in-memory analytics, which makes using pandas to analyze datasets that are larger than memory datasets somewhat tricky.” (pandas_development_team, 2022) So another permanent storage for a large amount of data needs to be found.

Therefore, different types of databases need to be compared. To choose a Database Management System, the advantages of the concepts of Relational Database Management Systems (RDMS) and Not Only SQL (NoSQL) must be weighed. The following table compares the major differences between the two database concepts (Figure 5).

	RDBMS	NoSQL
DB Theory	Full	Partial
Scale	Big (GB)	Extremely big (TB ~ PB)
Schema	Fixed	Flexible
Query	Efficient Effective	Efficient for simple queries Not effective/efficient for complex queries
Consistency	Strong	Weak
Integrity	Easy	Hard
Scalability	So so	Good
Availability	Good	Very good
Standardization	Yes	No

Figure 5 RDBMS vs. NoSQL

It needs to be analyzed which is better fitting to the purpose of the application. The CAP Theorem can help set the right priorities for the database. “The CAP theorem is a belief […] about distributed data stores that claim, in the event of a network failure on a distributed database, it is possible to provide either consistency or availability—but not both.” (Johnson, 2022)

The recommendations for the BookCrossing users will be based on querying a large amount of data. This data will get even larger as more users, books, and ratings will join over time. The data needs to be constantly available, to get recommendations all the time for the best user experience. Additionally, the recommendations need to be made as fast as possible to satisfy the user. Therefore, the priorities are partitioning and availability, to still get a result when there

is a network failure. Consistency is secondary, the recommendations do not depend on having the latest data but on analyzing a large amount.

Because of the need for availability, partitioning, and a large amount of data in general, a NoSQL database will be used for the application. This choice brings other advantages like a flexible schema and the possible implementation of nested data (Figure 5). How the compatibility issues are handled will be explained later.

The following table depicts several data models in the NoSQL environment (Figure 6).

Data Model	Key/Value	Document	Column-Family	Graph
Products	Redis, Riak, SimpleDB, Chordless, Scalaris, Memcached	MongoDB, CouchDB, Terrastore, ThruDB, RavenDB, …	BigTable, HBase, Cassandra, HadoopDB, GreenPlum, PNUTS	Neo4J, OrientDB, InfoGrid, Infinite Graph, GraphDB, FlockDB
Application Scenarios	Content caching, e.g. sessions, shopping carts, configuration files	Store, index and manage document-like or semistructured data	Distributed large-scale data management	Highly connected less structured data,, e.g. social network
Advantage	Scalable, flexible, high throughputs for writes	Scalable, flexible, not complex,	Fast querying, extremely high scalability, low complexity, easy to scale out	High flexibility, supporting complex graph algorithms
Disadvantage	No structure at all, conditional query inefficient	Lack of unified query language	Not full support of transaction, Limited consistency	High complexity, limited scalability
Industrial Users	Redis: Baidu CloudDB, Twitter(+memcached), StackOverFlow, Instagram Riak: GitHub, BestBuy	MongoDB: Baidu CloudDB, SAP, Codecademy, Foursquare RavenDB: NBC News	HBase: Facebook, Yahoo! Cassandra: Ebay, NASA, Instagram, Twitter (Cassandra+HBase),	Neo4J: Adobe, Cisco, T-Mobile

Figure 6 NoSQL Data Models

For the recommendation application, a document storage will be used because of the semi-structured and object-oriented concept (Figure 6). The data for the application is currently stored in three tables, which have a structure that is important for interpreting the data. This structure must be maintained. Another major advantage is the high scalability of the application to meet the need for fast queries. As a representative of the document storage model, Mongo DB is used because of the easy access with the Pymongo API. Through a large amount of documentation about the MongoDB query language, the lack of a unified language of document storages can be overlooked.

“MongoDB is designed for OLTP workloads, so more transactional, online, real-time workloads.” (diginomica, 2022) However, Mongo DB is one of a few NoSQL storages that can also be used for OLAP on a limited level (galaktika-soft, 2022). Since the queries in the application are performed within MongoDB, the application considers OLTP and OLAP. To combine both concepts in one database, MongoDB is the right choice for the application's database.

2.2 Creating a Recommendation Algorithm

Now to discuss how the recommendation is created.

The first thing to think of when talking about recommendations is probably Machine Learning (ML). ML models can also be saved in Mongo DB (simplified, 2022). ML provides reliable handling of any data and can create new opportunities in customer interaction. (TechVidvan, 2022) On the other side, it takes many resources like time and space (TechVidvan, 2022). Therefore, it needs to be considered carefully whether ML should be used. With ML the books could be sorted into clusters according to their attributes and the recommendation would result in other books within this cluster. However, these attributes just include information about the title, author, year of publication, and publisher (Figure 1). The clusters would not include any information about the content of the books, which is the major point when it comes to recommending a book. Therefore, clustering will not be used.

Rather the relationship between the books and users will be used to create recommendations. Therefore, an algorithm will be created which includes multiple database queries. The idea, to get book recommendations with similar content, is to check the books liked by other users, who also liked your book. This is possible because of the book rating table which links users and books.

How will this idea be realized? Querying a Mongo database is most efficient with aggregation pipelines. “An aggregation pipeline provides better performance and usability than a map-reduce operation.” (MongoDB, MongoDB, 2022) The data does not need to be extracted from the database to perform the queries, which would be too time-consuming. The major advantage of the pipeline is that they connect different aggregation operations together so that the result does not need to be written into the collection each time a single operation is performed.

In Mongo DB it is possible to save nested data structures. Therefore, each book document could contain a list of its readers and each reader could contain a list of the books he read. To achieve this stage of nested data the three tables need to be joined once, which would be time-consuming and costly. Furthermore, is the relationship between books and users a many to many relationships which would make every single document very large. This would make the querying cumbersome. So, the aggregation pipelines will not be performed on nested documents but on the three separated collections that will be created from each CSV file.

The algorithm will query and shape the collections with pipelines. The first pipeline retrieves the matching ISBNs for the title specified by the user. The result is stored and used as a parameter for the next query pipeline. This fetches the matching UserIDs from the user collection according to the ISBN input and stores them in a new collection called readers. Iterating over the results, a new pipeline retrieves all ISBNs from the book_rating collection that match the UserID input and stores them in a new collection called recommendations. The final pipeline reshapes the recommendations collection by grouping, counting, sorting, and

determining the most frequent and highest rated ISBNs. These ISBNs are presented to the application user with additional attributes in a GUI. To overcome the compatibility issues of Mongo DB, the results are stored in a Pandas DataFrame so that the information is easily accessible. This is possible because the result consists of only a few records selected by the algorithm. As none of the variables with null values needs to be considered for this approach, the cleaning of this data can be neglected.

2.3 Performance and Dataflow

To further improve the availability of the data and the performance of the application, MongoDB will be run in a virtual cluster. In the context of MongoDB, this means having a replica set or a sharded cluster. “A replica set is the replication of a group of MongoDB servers that hold copies of the same data.” (MongoDB, MongoDB, 2022) These copies ensure high availability and redundancy. They also provide extra read operations capacity and subsequently reduce the overall load of the cluster. (MongoDB, MongoDB, 2022) A sharded cluster means horizontal scaling, data is distributed across many servers. In the following a replica set will be used as sharded clusters should not be used if the application is still evolving, which is the case (Pratap, 2022).

To run MongoDB with replica sets within a cluster, the Database-as-a-Service MongoDB Atlas will be used (Figure 7). The managed MongoDB service is offered within Amazon Web Services, a comprehensive, evolving cloud computing platform (techtarget, 2022). The following figure depicts the interaction and dataflow of between the user, GUI, application, database, and the cloud (Figure 7).

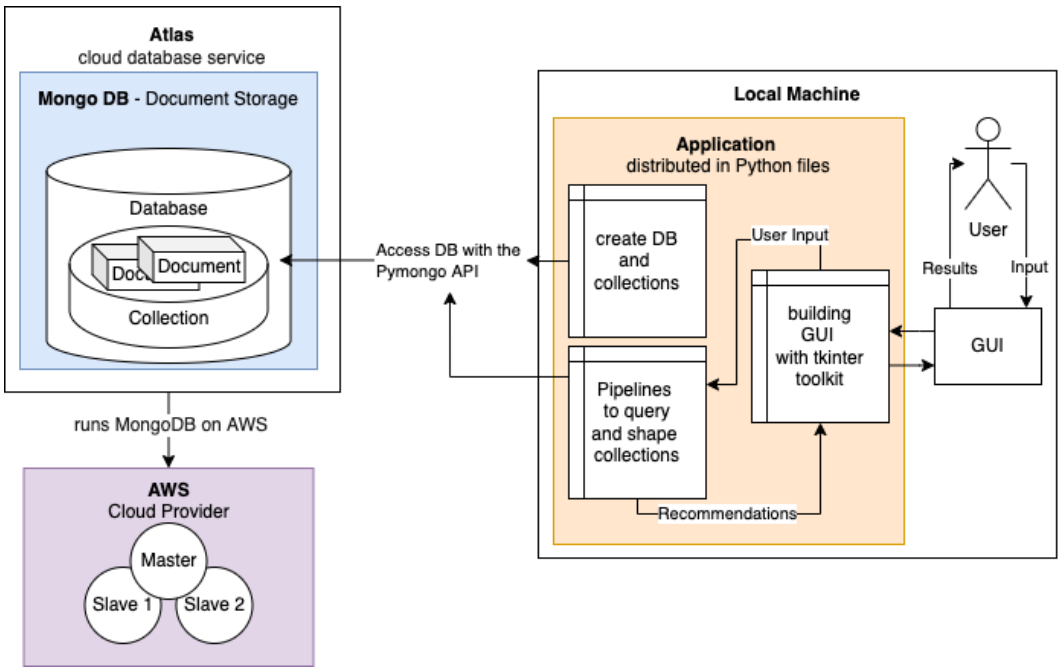


Figure 7 Architecture and Dataflow of the Application

3 Implementation

The application is written in distributed python files.

3.1 The Setup

The first step is to create the database and collections. To do this, Python must connect to Mongo DB. This is possible thanks to Pymongo, which provides an intuitive API for accessing databases, collections, and documents. (DB, Mongo DB, 2022) It is necessary to create an instance of the Mongo Client class and pass the information about the server on which Mongo DB should run. Running the application in the virtual cluster means connecting to Atlas.

After connecting, the database and the collections can be created. To insert the CSV files into the collection, the collection must be modified because the collections can only store JSON files. To provide the correct format, the CSV files are first read into Pandas DataFrames and then converted to JSON files. A new collection is created for each VSC file. To be efficient, the add_collection_to_db() function is created to perform these steps for each collection. All functions related to setup are stored in a file named setup_functions. The setup is performed once to insert the data into Atlas MongoDB.

3.2 The Algorithm

The algorithm to retrieve recommendations according to the user input is distributed in two files, one to link all functions together and hand over the results to the next one and one to provide the aggregation pipelines. (To follow the explanations you can use the pipeline file.)

The algorithm starts by calling the get_isbn() function on the user input. The function aggregates the isbn_pipeline on the book collection. The match processing stage finds the ISBN that matches the title and saves it in a new collection.

The next function get_userids() is called on the result of the previous function. With a match operation, it finds all UserIDs in the book_rating collection that have the corresponding ISBN (**Error! Reference source not found.**). After that, the results are grouped by UserID to get rid of any cases where a user read the same book several times. Additionally, the average book rating for each document is calculated. This is necessary for the next step to filter the documents whose rating is below four. With this only the users who liked the books are kept in the pipeline and the data gets significantly smaller. Finally, the result is returned in a new collection by calling the out operation. It is important to notice that this overwrites everything that has been in this collection so far. This is needed to overwrite the results of the previous search.

Again, an iteration calls another function on each of the results that have been saved in the reader collection. The objective of the get_books_w_merge() is to collect all well-rated books matching the UserIDs retrieved by the previous function. To save the result this time, it is not

possible to work with an out operation. This operation would overwrite all results from the previous iteration. To save all results of matching UserIDs in one collection the merge operation is needed. The operation checks if the document, identified by the unique object ID, is already in the collection and inserts it if not. The recommendations collection needs to be emptied every time a new search is performed.

The last function is called on the recommendations collection created by the merge operation in the previous function. In this pipeline, the result is again saved with an out operation to overwrite the previous results. Therefore, it can be seen as a reshaping of the final number of documents in the recommendations collection. The pipeline groups the ISBNs together counts them and calculates the average rating. This means a single document is now displaying the total amount of readers and the average rating of each ISBN. To keep only the positive ratings, all ratings below 4 are rejected with a match operation.

The next step is to prepare the recommendations for their display in the graphical user interface, which requires more attributes than just the ISBN. To display the result properly the corresponding title, author, and cover URL are added with a lookup operation. This operation performs just as a left side join in SQL and adds the data from the book collection with the ISBN as a key (pipelines.py, line 143). Such joining operations are very costly, this is the reason why a limit operation is performed beforehand. The additional information is just required for the final results which need to be displayed in the GUI. To get these results, the collection is sorted descending first by the number of readers and then the average rating (pipelines.py, line 136). With the following limit operation just the first 10 documents, the most highly rated, are selected.

The lookup operation results in a nested document where the additional information about the books is inside an object-like structure. To be able to access the results later this document needs to be normalized again to be able to properly create a Pandas DataFrame. For this purpose, a projection operation creates a new column for each attribute of the book information (pipelines.py, line 152). The now normalized collection is saved with an out operation to overwrite the previous stage of the recommendations collection.

The last step of the algorithm is to return the result in a Pandas DataFrame for better accessibility. This also means the GUI file does not need to access MongoDB itself.

The function get_recommendations(), which calls all of these functions are stored in a separate file together with the collection_setup(). This is the only file that is accessed by the GUI file.

3.3 The Graphical User Interface

To create the Graphical User Interface (GUI) the Tkinter Toolkit is used. The interface is created in the only executable python file “main” and relates to a separate file ui_functions, where the displaying functions are stored. The two functions are needed to execute and depict

the result of the algorithm. As already mentioned, the algorithm returns a Pandas DataFrame which contains the recommendations. Therefore, the functions do not need to access the Mongo database. For better efficiency, the method result_box () creates the widgets for each result, including title, author, rating, the cover of the book, and the number of readers who liked your search and the recommendation. These boxes are arranged side by side with more information about the input of the user (Figure 8).

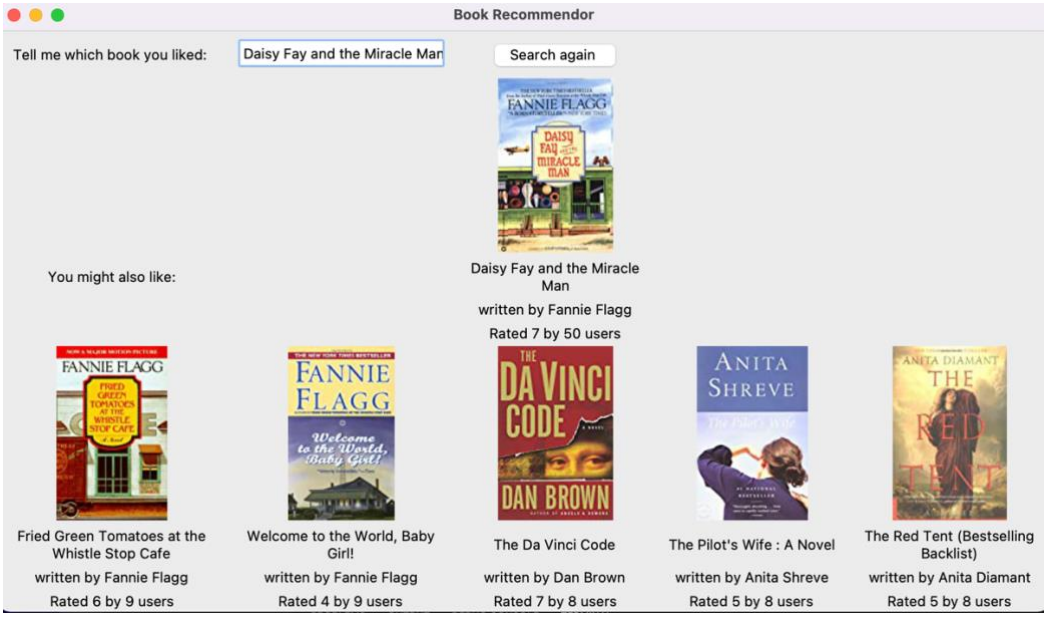


Figure 8 GUI

There are two different functions to execute the algorithm, to allow the user to initiate several searches without closing and opening the application. The get_first_result () function gets the recommendations and exchanges the search button for another button. This new button enables the second function. The only difference in the get_more_results() functions, is the deletion of the previous results widgets. This way multiple searches can be started in a single window.

3.4 Adjustability and Performance of the Application

The way the algorithm is implemented brings a certain amount of adjustability with it. Several parameters can be adjusted to make the search faster or more precise. For instance, the limitations of results that are fed into the next pipeline (core_functions.py, line 56), ensure a faster result by skipping a defined number of documents. The recommendations are presented faster but not all the data was taken into consideration. This parameter can be adjusted according to the resources of the executing machine and the preferences of the person starting the application.

The implementation also ensures the expandability of the application, it can easily be adjusted to get more recommendations for a book or to search not only by title but rather for every other preference like a specific author.

To ensure fast results to satisfy the user with quick recommendations and guarantee the availability of the data the application, Mongo DB runs within a virtual cluster. This means replicas of the data are stored in different nodes. Within Atlas three virtual machines are created. Subsequently, the setup function is used to create the database and move the data from the CSV files into the workable Mongo database. To access the database from multiple machines, the network-access IP is changed to 0.0.0.0/0 which means any IP can perform read and write operations on the data. Once the application is mature enough a sharded cluster can be implemented as the foundation has been created within MongoDB.

Another performance improvement can be realized once the application is mature enough. With the implementation of indexes, faster results can be realized. Indexing can be easily realized for several operations in Mongo DB (DB, Mongo DB, 2022). This means that in a further developed version of the application, indexes can be used to make the queries within the aggregation pipelines even faster.

4 Results and Reflections

The objective of the application is to generate recommendations according to user input. Therefore, a Graphical User Interface should be created to get the best possible user experience. The application should deliver the results as fast as possible. With the use of MongoDB and the implementation of aggregation pipelines to query the database, fast results can be achieved. Additionally, Atlas is used to manage the cluster within AWS. The algorithm links pipelines together to find well-rated books that were read by readers who also liked the book which initiated the search. The procedure indirectly considers the content of the books.

The Application can be adjusted in several ways to better fit the resources of the executing machine and the preferences of the user. The foundation was laid for several performance improvements. Additionally, MongoDB provides a variety of queries and ways to improve pipeline performance. Atlas offers easy deployment of a cluster and helps with monitoring. This leads to a general recommendation for working with the Pymongo API, Mongo DB, and Atlas for further improvements for this application and other big data-related topics.

Nevertheless, there are some limitations to the essential user-friendliness of the application. The toolkit Tkinter, which was used for the implementation of the GUI, quickly reached its limitations. The performance of the interface damages the overall speed of the application. Additionally, the realization of creating new widgets and deleting many previous widgets, without deleting them all, is designed inefficiently. This leads to the recommendation to use another framework for the GUI implementation. In general, one must balance the advantages of using Python or Java for the implementation, as Java has a greater variety of GUI Frameworks. On the opposite side, Python has a straight foreword concept.

In summary, the objectives of the application have been achieved, combined with bases for further adjustments or improvements in performance.

List of References

DB, M. (25. 03 2022). *Mongo DB*. Von https://www.mongodb.com/languages/python abgerufen

DB, M. (26. 03 2022). *Mongo DB*. Von https://www.mongodb.com/docs/manual/indexes/ abgerufen

diginomica. (31. 03 2022). *diginomica*. Von https://diginomica.com/mongodb-cto-mongo-works-doesnt abgerufen

galaktika-soft. (31. 03 2022). *galaktika-soft*. Von https://galaktika-soft.com/blog/sql-vs-nosql.html abgerufen

Johnson, J. (25. 03 2022). *bmc*. Von bmc: https://www.bmc.com/blogs/cap-theorem/ abgerufen

MongoDB. (25. 03 2022). *MongoDB*. Von https://www.mongodb.com/docs/manual/reference/map-reduce-to-aggregation-pipeline/ abgerufen

MongoDB. (30. 03 2022). *MongoDB*. Von https://www.mongodb.com/basics/clusters abgerufen

pandas_development_team. (25. 03 2022). *pandas.pydata.org*. Von https://pandas.pydata.org/pandas-docs/stable/user_guide/scale.html abgerufen

Pratap, A. (30. 03 2022). *Medium*. Von https://medium.com/geekculture/mongodb-why-avoid-sharding-it-should-be-kept-as-the-last-option-cb8fdc693b66 abgerufen

simplified, p. (25. 03 2022). *python simplified*. Von https://pythonsimplified.com/how-to-use-mongodb-to-store-and-retrieve-ml-models/ abgerufen

techtarget. (31. 03 2022). *techtarget*. Von https://www.techtarget.com/searchaws/definition/Amazon-Web-Services abgerufen

TechVidvan. (25. 03 2022). *TechVidvan*. Von https://techvidvan.com/tutorials/advantages-and-disadvantages-of-machine-learning/ abgerufen