# Exploring the Test Driven Development of an Information Retrieval System

Daniel Staegemann[a], Sujith Sudhakaran, Christian Daase[b] and Klaus Turowski[c]
*Magdeburg Research and Competence Cluster VLBA, Otto-von-Guericke University Magdeburg, Magdeburg, Germany*

Keywords: Software Engineering, Information Retrieval, Test Driven Development, TDD, Testing, Quality Assurance.

Abstract: Today's society is heavily driven by data intensive systems, whose application promises immense benefits. However, this only applies when they are utilized correctly. Yet these types of applications are highly susceptible to errors. Consequently, it is necessary to test them comprehensively and rigorously. One method that has an especially high focus on test coverage is the test driven development (TDD) approach. While it generally has a rather long history, its application in the context of data intensive systems is still somewhat novel. Though, rather recently, a microservice-based test driven development concept has been proposed for the big data domain. The publication at hand explores its feasibility regarding the application in an actual project. For this purpose, a prototypical, microservice based information retrieval system is implemented in a test driven way with particular consideration for scalability.

## 1 INTRODUCTION

Test driven development (TDD) is a popular approach for developing software. However, despite the associated potential advantages and some corresponding works (Daase et al. 2023; Staegemann et al. 2022b; Staegemann et al. 2022a), it is still rather underexplored in the context of big data (BD) and the related applications. Here, in our study we apply TDD to implement an information retrieval system. While the presented prototypical implementation is rather small in a BD context, it still showcases a typical BD use case (Volk et al. 2020) and the presented architectural approach allows for further scaling. For this purpose, we write unit tests starting from tokenization to similarity computation.

However, it has to be noted that the primary focus of the publication at hand is to further explore the use of the TDD approach itself. Therefore, the information retrieval system is just a vehicle to achieve this. Thereby, it is in our interest to further increase the complexity, which we did my adding an explainability component to the posed task.

Oftentimes, machine learning models are black box models which compute the results without explaining how the results were computed. These black box models are difficult to interpret making it very hard to understand the reason behind the outcome. In order to make it more interpretable, the features of the model can be exploited to provide basic explanations to at least somewhat open the black box. This helps to verify the outcome if the model works correctly and helps in gaining more trust from the user.

In our proposed system, we try to improve the trust and fairness of the retrieval system by providing valid explanations on how results are retrieved. We try to transfer the system into a white box system by providing necessary explanation and analyse how the system performance can be improved using such explanations. We follow test driven development during each phase of the development cycle, and we develop the system as microservice architecture so that the components are loosely coupled and largely scalable.

The publication is structured as follows. Succeeding this introduction, the test driven development approach is described. Afterwards, the general design of the prototypical application is outlined. This is followed by a section dedicated to

[a] https://orcid.org/0000-0001-9957-1003
[b] https://orcid.org/0000-0003-4662-7055
[c] https://orcid.org/0000-0002-4388-8914



Staegemann, D., Sudhakaran, S., Daase, C. and Turowski, K.
Exploring the Test Driven Development of an Information Retrieval System.
DOI: 10.5220/0012147700003552
In *Proceedings of the 20th International Conference on Smart Business Technologies (ICSBT 2023)*, pages 104-113
ISBN: 978-989-758-667-5; ISSN: 2184-772X

the actual implementation. Then, the findings are discussed. Finally, a conclusion of the work is given.

## 2 TEST DRIVEN DEVELOPMENT

When consulting the scientific literature (Staegemann et al. 2021), the application of TDD is highlighted as a promising way to improve the quality of an implementation as long as the associated increase in development time and effort is considered an acceptable trade-off.

In general, the approach aims at improving the quality of the product under consideration by mainly influencing two aspects. It aims to increase test coverage, which helps to find and subsequently fix problems that occurred during the implementation of the artifact under consideration. Further, TDD also influences the design process itself by leading to a more manageable and pre-planned structure that helps to avoid errors and incompatibilities (Crispin 2006; Shull et al. 2010). The main application area is software development, but process modelling (Slaats et al. 2018), the special case of BD application implementation (Staegemann et al. 2020), and ontology development (Davies et al. 2019; Keet and Ławrynowicz 2016) can also be found in the literature.

In "traditional" software development, a function or change to be realized is implemented and then tested. In contrast, the test-driven approach reverses the order of implementation and testing. That is, after the desired change is designed, it is broken down into its smallest meaningful parts (Fucci et al. 2017). For these, one or more tests are written to ensure that the required functionality is provided. Then the tests are run, and are expected to fail because the actual functionality is not yet implemented (Beck 2015). Only then is the production code written to provide the new functionality. Factors such as the elegance of the code are not yet considered; instead, the simplest solution is sought. Once the code is created, it must pass the previously written tests (Crispin 2006). If it is successful, the code is revised to improve aspects such as readability or compliance with standards and best practices (Beck 2015). In this process, its functionality is constantly validated against the tests.

However, this approach not only affects test coverage, but also the design of the software, since small work packages are generated instead of large tasks. In addition, this focus on incremental changes (Williams et al. 2003), where testing and implementation are interwoven, provides more timely feedback to the developer by creating short test cycles (Janzen and Saiedian 2005). Although most tests are written specifically for these small units, other tests such as integration, system, or acceptance tests can also be used in TDD (Sangwan and Laplante 2006).

In addition, to fully exploit the potential of TDD without tying up the developers' attention by forcing them to manually execute the tests, TDD is often used in conjunction with test automation in a continuous integration (CI) context (Karlesky et al. 2007; Shahin et al. 2017). Here, to ensure the latest code addition or change does not negatively impact existing parts of the implementation, a CI server automatically starts and re-executes all applicable tests when a new code commit is registered by the versioning system.

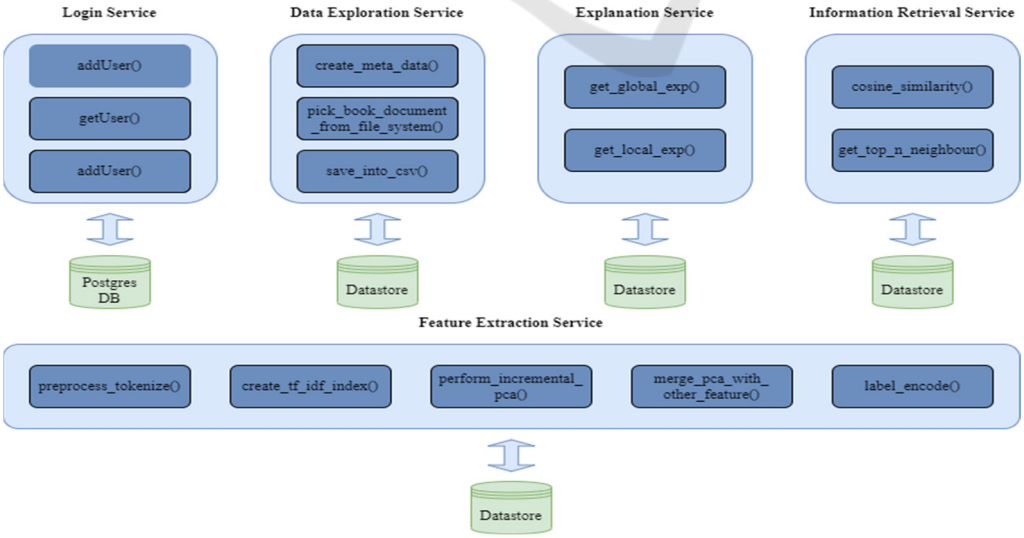


Figure 1: The implemented microservices.

# 3 THE DESIGN

As suggested in (Staegemann et al. 2020), we chose to design the application as a microservice architecture, which is more modular and scalable compared to a monolithic implementation (Ataei and Staegemann 2023). Figure 1 shows all the services that are part of our information retrieval system.

We have developed multiple services which use their own database. For the login service, we had to choose between relational and non-relational databases. Since the login data of the user is structured, we decided to implement it in the postgres relational database. For convenience reasons, we decided to use an on-premise database to run the postgres database locally. We used the java programming language and the Spring Boot framework for development of the login microservice. Besides that, we used the python programming language and the flask framework for the development of other services such as the Data Exploration Service, the Feature Extraction Service, and the Information Retrieval Service. In this prototypical implementation, these services use the computer's internal file system for processing and storage of documents. However, in a real-world scenario, this would of course also be handled through distinct databases. The information retrieval components are developed in python as there are plenty of libraries available as open source to use them for various functionalities.

Benjelloun at al. (2020) did a comparative study of different techniques used for processing data in data heavy applications. These data processing methods adhere to the same cycle such as the data collection, data preparation, data input, data processing, data output/interpretation, and data storage. The different methods of processing data are batch processing, stream processing, and real time processing. The proposed system doesn't require data to be processed frequently in real time. Therefore, we designed a pipeline that can process data in batches and schedule the data to be extracted as chunks. The data that we processed consisted of 13499 books in varying sizes and the size of the dataset was almost seven GB.

Although this is already a somewhat big size, it is still a rather low scale in the context of BD. Yet, for a prototypical scenario it should be sufficient. However, the goal is, naturally, to showcase the suitability in a BD scenario. Therefore, to process it, we employed a pipeline, which we term as *Combi-Pipeline*, for feature extraction. The term Combi-Pipeline refers to the pipeline that we designed which has the combination of data from google drive and the local file system and merged them to feature vectors. As shown in Figure 2, we used this Combi-Pipeline to extract the features such as named entity recognition and term frequency-inverse document frequency (tf-idf) vectors. The features were extracted by dividing the data into 14 chunks having 1000 books in each chunk except the last chunk which has 499 books. This shall show the general possibility to distribute the processing across multiple different (types of) workers.

For the design and development of front-end web pages, there are multiple frameworks such as AngularJS, React, Marko js etc. available. The JavaScript framework AngularJS was developed to enhance HTML's syntax. By using this framework, developers may create rich internet applications more

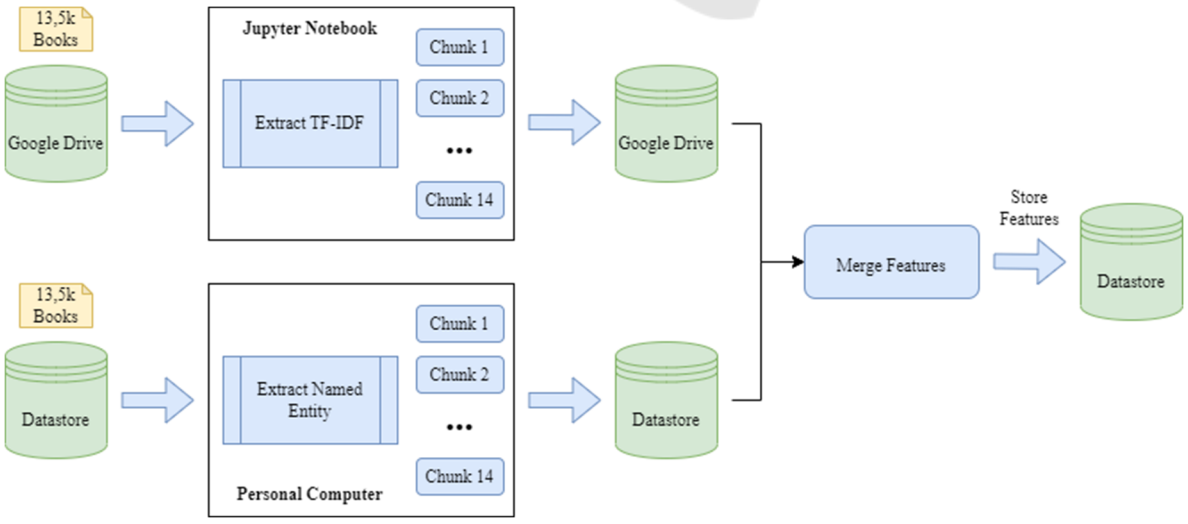


Figure 2: The feature extraction pipeline.

quickly and easily. Further, AngularJS allows the development of application as single page applications and provides techniques for boosting HTML, whereas many JavaScript frameworks concentrate on enhancing the capabilities of JavaScript itself. AngularJS enables developers to avoid the complicated workarounds that are typically required when attempting to construct responsive web apps with HTML front ends by introducing features like data-binding. Therefore, we chose Angular JS for developing the front-end of our system.

# 4 THE IMPLEMENTATION

The entire retrieval system architecture is shown in Figure 3. There are two back-end blocks and a front-end block in our retrieval system. As mentioned earlier, the front-end web application is developed in Angular JS, a framework based on typescript and the back end units are developed using python and java. We have used Springboot (a java-based framework) for the implementation of the login service. The user data is persisted in the postgres database as shown.

We have used flask (a python-based framework) for the implementation of the data exploration service, the feature extraction service, and the information retrieval service. As described in the previous section, in this prototypical implementation, these services use the local file system as the database storage. The user communicates with the front-end web application, which, in turn, queries from the login service for authentication. Once the authentication is successful, the python service for retrieval is invoked.

The services for feature exploration and extraction were already invoked and loaded into the file system so as to reduce the response time for each query. The architecture is designed based on diverse languages such as java script, java, and python for the purpose of highlighting the possibility of mixing different programming languages when using a microservice architecture (Shakir et al. 2021). Further, it also allowed to explore the test driven methodology with different frameworks that are used by the languages.

## 4.1 Login Service

We developed the service starting with writing the tests for the functionalities involved with the service.

The login service has three major methods used for login purposes. The *getUser()* method is used for logging an existing user into the application. The user triggers a post request with username and password in the request payload. The *addUser()* method is used for registering a new user to the application. This is a post http request, which gets parameters such as username, email, password, and designation. The designation is an *enum* variable and it can be "student", "researcher" or "others". This is tracked so that we can perform analytic on who is searching for what books in the future. The *updataUser()* method is designed for updating the existing user information. The user shall update their username, email, designation and password by triggering a PUT http request. Further, there is also a *getUsers()* method to fetch all the users from the database. This is not used currently but it can be used in the future to report certain metrics based on the users in the system. As we store information about the designation of users, we shall use this to derive certain analytics such as users tagged as "Researchers" search for a particular book genre and users tagged as "Students" search for particular books.

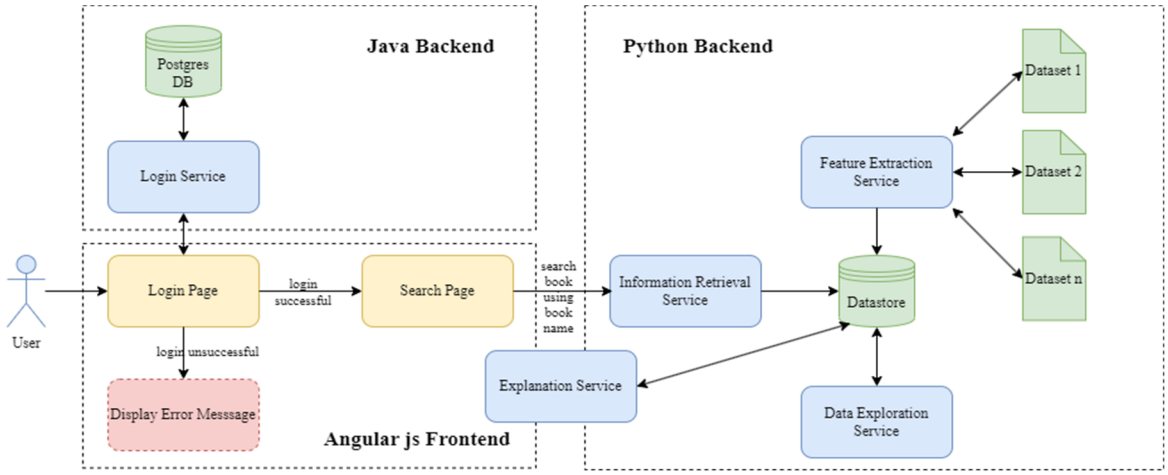


Figure 3: The architecture of the retrieval system prototype.

The following unit test cases were derived from our use cases of adding user, updating user, getting user and deleting user:

- Successful addition of user into database on correct values
- Error message on passing incorrect username and password
- Encryption of password while inserting into database
- Decryption of password while authenticating users from database
- Test to check if the user is updated correctly for the PUT http request

## 4.2 Data Exploration Service

The data is extracted from dataset "The book Corpus" (Zhu et al. 2015) since there are many researches being performed in the field of NLP using the book corpus dataset. Bandy et al. (2021) discuss certain limitations involved with the book corpus dataset but also highlight how it is used in popular systems such as Open-AIs GPT models, further emphasizing its significance.

We have also extracted metadata of the dataset from a GitHub account that also had the actual dataset (soskek 2019). Note that the actual book dataset and metadata json listed in the GitHub are fetched from different scraped sources and so they have different book counts and orders. While the book dataset comprised 17869 titles, the metadata set had only 15000 entries.

Hence, we had to find the intersecting books in the two datasets to merge the two datsets to construct the master dataset. This is done in the *create_meta_data()* method. The *pick _book_document_from_file_system* method is created because we require to choose only the books that are available in the metadata dataset to perform all subsequent tasks such as pre-processing and feature extraction. To do so, we decided to go with a simple approach. We created a new column called "exists" in the meta data. This column determines if the book title in the metadata exists in the books in file system. The "exists" value is true if the book exists in our file system or false if it does not exist in the file system. All the book names in the file system are scanned and the column "exists" is populated if the book is present in the file system.

Further, as the name suggests, the *save_into_csv* method saves the updated *meta_data.csv* into the file system. In the service's other methods, we modify the existing meta data file and save the updated version into the file system. Thereby, this service acts as a helper method to accomplish the task of saving the metadata into the local disk.

The data exploration service is developed in the python programming language using the flask framework to facilitate the microservice architecture. We have written the tests using the *unittest* library in python and we used the mock class from the same library for mocking method calls. Before starting with the functionalities on the data extraction and exploration, we began with the test cases that are listed below:

- Test if the specific directory in the local disk is being scanned for the books and meta data is created
- Tests asserting the size of metadata records, number of columns, books retrieved are verified
- Since we are excluding books which have a number of words greater than 200000 from the dataset (to have a more homogeneous set), there is a test that explicitly checks if the dataset includes any book over 200000 words
- As we have two different datasets extracted from different sources one containing meta data in json format and the other containing the books as compressed files, we had to write a test to match the metadata with the book dataset within the directory. So, only the books that are present in the local directory are used for processing while the books that only have meta data and don't have an actual book associated in the local directory are ignored

All these tests were written keeping in mind that the core functionality of the data extraction should not be hindered, and any possibility of functional errors must be eliminated.

## 4.3 Feature Extraction Service

The data that is extracted from the previous step is passed to the feature extraction step where we derive the features from the books and the meta data available. The feature extraction service is developed with the python programming language using the flask framework. In this step, we extract the features by converting books to tf-idf vectors and combined it with a set of handcrafted features to yield better results for the retrieval system. The feature extraction pipeline can be divided into three parts with their own functionalities. These are further described in the following.

#### 4.3.1 Extraction of Hand-Crafted Features Such as Named Entities and NRC Emotions

We decided on a few handcrafted features such as Named Entities and NRC emotions for our retrieval system.

Named Entity Recognition (NER) is an important feature (Mansouri et al. 2008) in the text retrieval domain as it can detect the person, place, quantity, time etc., which is useful for retrieval systems. We used a popular library called spaCy (spaCy 2023) for detecting the named entities in the books. SpaCy provides pre-trained model with set of named entities and it recognises the tokens and predicts the named entity for the token.

Moreover, sentiment analysis is an important aspect of information retrieval. In our work, we have used the NRCLex library (Bailey 2019) to detect the emotions from the book. This library uses the lexicons from the NRC Word-Emotion Association Lexicon (Mohammad and Turney 2011). This library scans the dataset for specific words and classifies the words based on the emotions. The output of this is the word count of emotions such as anticipation, joy, fear, or sadness.

#### 4.3.2 Extraction of the tf-idf Vectors from the Books

In all the NLP tasks, the natural language should be converted to tokens for identifying the features. In our tokenization process, we scan through the dataset and convert all the books as corresponding tokens. The nltk library (NLTK Project 2023) in python is used for creating tokens out of the books. Thus, we tokenize the entire document corpus using the library.

In our approach we used tf-idf to convert the books to vectors using the python based library sklearn (scikit-learn 2023). In order to perform tf-idf vectorization using sklearn, we passed the optional parameter max_features to the method *TfidfVectorizer* in the library.

Further, we have filtered the stop words occurring in the documents. We have set a maximum features limit of 300000 so as to include only the first 300000 features with the highest weightage. Therefore, after the tokenization step, the tokens are moved to the stop word removal where we remove stop words from the tokens. Then we perform stemming where words with the same stem are grouped into one word and finally, we perform the tf-idf vectorization on the remaining words. After the vectorization we collect the features in a pandas dataframe data structure and store it into a csv file in the file system.

#### 4.3.3 Principal Component Analysis and Feature Combination

After the steps described above, it is necessary to group all features that we constructed into a single file so that it becomes easy for processing. It is important to perform principal component analysis (PCA) on the tf-idf features so to reduce the error caused by a high dimensionality. We also merged all the meta data with the PCA features and performed label encoding to our feature set.

In (Korenius et al. 2007), the authors explain how the search space can be reduced for text retrieval systems using PCA. They also establish the relation between cosine measure and the euclidean distance in association with PCA. Performing PCA on the entire dataset was challenging as we could not fit the entire feature space in the main memory for processing.

Additionally, when considering that this is only a prototypical implementation geared towards BD applications, this issue would be even bigger on larger scale. In response, we tried out multiple approaches to perform PCA. The first approach is to split the data into chunks and perform PCA. The chunk size was set to 1000 books and after computing PCA for all the books, we combined the PCA vectors. As we are performing TDD, we started by writing the test for this approach. However, when running it, we identified that the results are not identical and so we cannot proceed chunking books for PCA as it yields incorrect results.

Another approach is discussed in (Bai et al. 2014). Here, the authors propose an incremental learning for a robust visual tracking systems which performs the PCA in an incremental way without the results being impacted. We used the method *IncrementalPCA* in the sklearn library (scikit-learn 2023) to perform PCA for the books dataset so that we arrive at the correct values for PCA.

The reason why we highlight the first (failed) attempt is to emphasize that the test driven approach helped in arriving at the correct method and yielding correct results when performing PCA. This was one such instance where TDD helped to detect the flaws in the methods chosen for the development of our retrieval system. After performing PCA, we have reduced the multi-dimensional feature space into a three-dimensional space and thus reduced 300000 terms (features) to three features.

After effectively performing the PCA on the tf-idf features and reducing the dimensions, our PCA features were ready to be integrated with the other features. To do so, we merged the PCA features with NER features, NRC emotions, and the meta data to construct the final feature set. This feature set consists

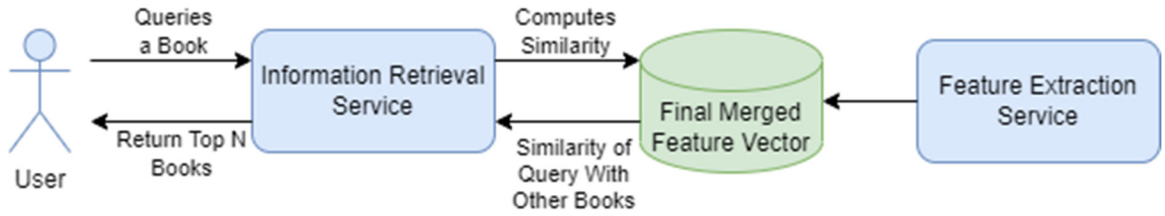


Figure 4: Operational sequence of the information retrieval service.

of 37 features. These features had to be further taken for encoding as the text features should be converted to numbers.

The retrieval system that we developed works on similarity metric (cosine similarity) to find the similarity between the data points (books). In order to compute similarity, it was necessary to convert all the text to numbers. We have features such as authors, category1 (genre 1), category2 (genre 2) and category3 (genre 3), which are text-based features. The number of unique authors in the dataset found is 8283. The category1 feature consist of two unique categories which are fiction or non-fiction. The category2 feature consist of 63 unique categories whereas the category3 feature consist of 889 unique categories. Thus, these features are converted to numbers using label encoders in the sklearn library (scikit-learn 2023).

## 4.4 Information Retrieval Service

The Information retrieval service is the user facing service where the user interacts with the system and searches for the required books. As the name suggests, this service retrieves the books matching to the query book provided by the user. Its operational sequence is depicted in Figure 4.

In (Ristanti et al. 2019), the authors describe how the cosine similarity measure was used to detect similarity in documents and evaluate the performance after applying K-fold validations. As explained in the paper, we have also used the cosine similarity measure, mainly because it is not affected by length of the document and thus yields a better result for identifying the document similarity.

This functionality was written in python and we used the sklearn library (scikit-learn 2023) to detect the cosine similarity. When the user queries for a book, the information retrieval service computes the cosine similarity between the query vector and all the feature vectors in the database and computes the result, which is the cosine similarity of the query with all the other books in the document corpus.

After finding the similarity score for all the documents with the query vector, it is important to retrieve the top similar books and return them to the user. To retrieve only the top n books, where n is set to 10 from the list of all the books in the corpus, we created the *get_top_n_neighbour method*. It identifies only the top 10 retrieved books, which are similar to the query books and acts as a utility method, which can be used by other services.

Before writing the code, we started with the test scenarios, which ensured that the functionality of the service was complete. In the test, we are passing a test dataframe whose similarity values are already known, and assertions are performed to test if the retrieved similarity rankings are correct. This test was useful because we figured out from the test that the *cosine_similarity()* method from the sklearn library (scikit-learn 2023) returns similarity scores in ascending order. Consequently, we modified our code so that it returns the similarity in descending order, which means that the books with highest similarity score should be fetched first.

## 4.5 Explanation Service

The explanation service is a user facing service, meaning it consists of a front-end user interface, which would be exposed to the user on click of a button that explains why the query book matches with the selected book. Explainability in NLP systems has been explored in various researches to open the black box layer and provide the interpretability to the users.

In (Zini and Awad 2023), a survey on the explainability of deep models is presented and the authors decompose the explainability methods into three categories such as word embeddings, inner working of NLP models, and model's decision. Out of these aspects, we provide explanation that focusses only on the inner working of the model. Further, in (Danilevsky et al. 2020), the authors present a survey on explanations in NLP systems, discussing the types of explanations. The paper also talks about the self-explanations generated by the models and post-hoc explanations provided by generating explanations as post processing techniques after prediction. In our prototype, we focussed on two major kinds of explanation which are generated as post-hoc

```
The selected book is similar to the query book as it falls in _________
    (primary genre) primary genre and _________ (secondary genre)secondary
    genre. The book is written by __________(author) and has the
    __________(Named Entity) has the primary named entity feature.
```

Figure 5: Local explanation template.

explanations. While they might not exhaustively explain the exact position of a retrieved item in the ranking, they at least help the user to get somewhat of an idea on the reasoning behind the assessment, allowing them to evaluate the reasonability and suitability for their purpose.

### 4.5.1 Global Explanation

The global explanation explains the retrieval system on the whole by giving an explanation to the user to understand the factors behind retrieving the results by the system.

It helps the user in understanding the assessment on a high level. This explanation is written in the frontend side of the application where the features on which the retrieval system works are displayed to the user. We used the AngularJS framework, which is based on typescript to design and develop the front-end of the application. We explain that the major features considered while retrieving are content of books, meta data information such as author, genre, named entities, and emotions of the contents.

### 4.5.2 Local Explanation

The local explanation is a backend service written in the python flask framework. When the user clicks on the corresponding button in the web page, it invokes the local explanation functionality. It checks the similarity and constructs the explanation in a specific syntax and returns the explanation as a json response.

As shown in Figure 5, we constructed the explanation telling the user about the primary and secondary genre of the book. We also provide explanation about the author of the book and the most occurring named entities in the book.

## 5 DISCUSSION

Even though the created prototype in its current scale is not necessarily a BD application, it still serves as a demonstration of the utility of TDD for the implementation of data intensive systems. This is because it is implemented test driven and shows the processing of heterogeneous data from different sources in a distributed manner, which are then fused for further utilization. Hence, for a real-world application, it could be scaled up, still using the same principles that were applied here.

While the prototype delivers satisfying results as determined during manual system testing, showing that the followed approach was generally suitable, the use of TDD also explicitly benefitted the development in several ways. For once, it helped to detect and avoid errors during the code writing and helped to install confidence in the created code. Further, it helped to shape the design not only by breaking the functionality down, but also by showing that a considered problem solving approach did not deliver the desired results and an alternative had to be found (cf. section 4.3.3). On top of that, it also allowed to save some time in one instance by helping to figure out the specificities of a utilized function without the need to consult the documentation (cf. section 4.4).

Thereby, the application of TDD based on microservices showed its worth for the implementation of data intensive systems by benefiting the development process. Hence, it generally seems reasonable to apply it also on large scale projects, at least if they are heavily relying on a high quality, and also to further explore the approach in the future.

## 6 CONCLUSION

While TDD is generally not a new approach, it is still somewhat underexplored in the context of data intensive systems. Yet, with their growing influence on today's society, assuring their quality becomes increasingly important. To bridge this gap, in the publication at hand, the test driven implementation of an information retrieval system was demonstrated and discussed. This is done on the basis of

microservices, making use of the architecture's property that varying programming languages and frameworks can be mixed to use the most favourable combination. Further, while the implemented application is only a prototype, it already comprises elements of distributed processing, which would allow for further upscaling. However, already on this size, indications for its feasibility and usefulness could be determined.

To gain further insights on the applicability of the approach as well as its strengths, weaknesses, challenges, and opportunities, in the future, additional data intensive applications of varying scales should be implemented in a test driven manner. Moreover, the knowledge and opinions of experienced practitioners should be gathered through expert interviews, to be able to also incorporate the industry perspective on the proposed approach and its application in a real-world scenario.

# REFERENCES


Ataei, P., and Staegemann, D. (2023). "Application of microservices patterns to big data systems," Journal of Big Data (10:1) (doi: 10.1186/s40537-023-00733-4).

Bai, S., Liu, R., Su, Z., Zhang, C., and Jin, W. (2014). "Incremental robust local dictionary learning for visual tracking," in 2014 IEEE International Conference on Multimedia and Expo (ICME), Chengdu, China. 14.07.2014 - 18.07.2014, IEEE, pp. 1-6 (doi: 10.1109/ICME.2014.6890262).

Bailey, M. M. (2019). "NRCLex 4.0," available at https://pypi.org/project/NRCLex/, accessed on May 3 2023.

Bandy, J., and Vincent, N. (2021). "Addressing "Documentation Debt" in Machine Learning Research: A Retrospective Datasheet for BookCorpus,"

Beck, K. (2015). Test-Driven Development: By Example, Boston: Addison-Wesley.

Benjelloun, S., Aissi, M. E. M. E., Loukili, Y., Lakhrissi, Y., Ali, S. E. B., Chougrad, H., and Boushaki, A. E. (2020). "Big Data Processing: Batch-based processing and stream-based processing," in 2020 Fourth International Conference On Intelligent Computing in Data Sciences (ICDS), Fez, Morocco. 21.10.2020 - 23.10.2020, IEEE, pp. 1-6 (doi: 10.1109/ICDS50568.2020.9268684).

Crispin, L. (2006). "Driving Software Quality: How Test-Driven Development Impacts Software Quality," IEEE Software (23:6), pp. 70-71 (doi: 10.1109/MS.2006.157).

Daase, C., Staegemann, D., Volk, M., and Turowski, K. (2023). "Creation of a Framework and a Corresponding Tool Enabling the Test-Driven Development of Microservices," Journal of Software, pp. 55-69 (doi: 10.17706/jsw.18.2.55-69).

Danilevsky, M., Qian, K., Aharonov, R., Katsis, Y., Kawas, B., and Sen, P. (2020). "A Survey of the State of Explainable AI for Natural Language Processing," (doi: 10.48550/arXiv.2010.00711).

Davies, K., Keet, C. M., and Lawrynowicz, A. (2019). "More Effective Ontology Authoring with Test-Driven Development and the TDDonto2 Tool," International Journal on Artificial Intelligence Tools (28:7) (doi: 10.1142/S0218213019500234).

Fucci, D., Erdogmus, H., Turhan, B., Oivo, M., and Juristo, N. (2017). "A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last?" IEEE Transactions on Software Engineering (43:7), pp. 597-614 (doi: 10.1109/tse.2016.2616877).

Janzen, D., and Saiedian, H. (2005). "Test-driven development concepts, taxonomy, and future direction," Computer (38:9), pp. 43-50 (doi: 10.1109/MC.2005.314).

Karlesky, M., Williams, G., Bereza, W., and Fletcher, M. (2007). "Mocking the Embedded World: Test-Driven Development, Continuous Integration, and Design Patterns," in Embedded Systems Conference, San Jose, California, USA. 01.04.2007 - 05.04.2007, UBM Electronics.

Keet, C. M., and Ławrynowicz, A. (2016). "Test-Driven Development of Ontologies," in The Semantic Web. Latest Advances and New Domains, H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S. P. Ponzetto and C. Lange (eds.), Cham: Springer International Publishing, pp. 642-657 (doi: 10.1007/978-3-319-34129-3_39).

Korenius, T., Laurikkala, J., and Juhola, M. (2007). "On principal component analysis, cosine and Euclidean measures in information retrieval," Information Sciences (177:22), pp. 4893-4905 (doi: 10.1016/j.ins.2007.05.027).

Mansouri, A., Affendey, L. S., and Mamat, A. (2008). "Named entity recognition approaches," International Journal of Computer Science and Network Security (8:2), pp. 339-344.

Mohammad, S. M., and Turney, P. (2011). "NRC Word-Emotion Association Lexicon," available at http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm, accessed on May 3 2023.

NLTK Project. (2023). "Natural Language Toolkit," available at https://www.nltk.org/, accessed on May 3 2023.

Ristanti, P. Y., Wibawa, A. P., and Pujianto, U. (2019). "Cosine Similarity for Title and Abstract of Economic Journal Classification," in 2019 5th International Conference on Science in Information Technology (ICSITech), Yogyakarta, Indonesia. 23.10.2019 - 24.10.2019, IEEE, pp. 123-127 (doi: 10.1109/ICSITech46713.2019.8987547).

Sangwan, R. S., and Laplante, P. A. (2006). "Test-Driven Development in Large Projects," IT Professional (8:5), pp. 25-29 (doi: 10.1109/MITP.2006.122).

scikit-learn. (2023). “scikit-learn Machine Learning in Python,” available at https://scikit-learn.org/stable/, accessed on May 3 2023.

Shahin, M., Ali Babar, M., and Zhu, L. (2017). “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” IEEE Access (5), pp. 3909-3943 (doi: 10.1109/ACCESS.2017.2685629).

Shakir, A., Staegemann, D., Volk, M., Jamous, N., and Turowski, K. (2021). “Towards a Concept for Building a Big Data Architecture with Microservices,” in Proceedings of the 24th International Conference on Business Information Systems, Hannover, Germany/virtual. 14.06.2021 - 17.06.2021, pp. 83-94 (doi: 10.52825/bis.v1i.67).

Shull, F., Melnik, G., Turhan, B., Layman, L., Diep, M., and Erdogmus, H. (2010). “What Do We Know about Test-Driven Development?” IEEE Software (27:6), pp. 16-19 (doi: 10.1109/MS.2010.152).

Slaats, T., Debois, S., and Hildebrandt, T. (2018). “Open to Change: A Theory for Iterative Test-Driven Modelling,” in Business Process Management, M. Weske, M. Montali, I. Weber and J. Vom Brocke (eds.), Cham: Springer International Publishing, pp. 31-47 (doi: 10.1007/978-3-319-98648-7_3).

soskek. (2019). “bookcorpus,” available at https://github.com/soskek/bookcorpus, accessed on May 3 2023.

spaCy. (2023). “Industrial-Strength Natural Language Processing,” available at https://spacy.io/, accessed on May 3 2023.

Staegemann, D., Volk, M., Byahatti, P., Italiya, N., Shantharam, S., Chandrashekar, A., and Turowski, K. (2022a). “Implementing Test Driven Development in the Big Data Domain: A Movie Recommendation System as an Exemplary Case,” in Proceedings of the 7th International Conference on Internet of Things, Big Data and Security, Online Streaming, --- Select a Country ---. 22.04.2022 - 24.04.2022, SCITEPRESS - Science and Technology Publications, pp. 239-248 (doi: 10.5220/0011085600003194).

Staegemann, D., Volk, M., Jamous, N., and Turowski, K. (2020). “Exploring the Applicability of Test Driven Development in the Big Data Domain,” in Proceedings of the 31st Australasian Conference on Information Systems (ACIS), Wellington, New Zealand. 01.12.2020 - 04.12.2020.

Staegemann, D., Volk, M., Lautenschläger, E., Pohl, M., Abdallah, M., and Turowski, K. (2021). “Applying Test Driven Development in the Big Data Domain – Lessons From the Literature,” in Proceedings of the 2021 International Conference on Information Technology (ICIT), Amman, Jordan. 14.07.2021 - 15.07.2021, IEEE, pp. 511-516 (doi: 10.1109/ICIT52682.2021.9491728).

Staegemann, D., Volk, M., Perera, M., and Turowski, K. (2022b). “Exploring the Test Driven Development of a Fraud Detection Application using the Google Cloud Platform,” in Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, Valletta, Malta. 24.10.2022 - 26.10.2022, SCITEPRESS - Science and Technology Publications, pp. 83-94 (doi: 10.5220/0011559000003335).

Volk, M., Staegemann, D., Trifonova, I., Bosse, S., and Turowski, K. (2020). “Identifying Similarities of Big Data Projects–A Use Case Driven Approach,” IEEE Access (8), pp. 186599-186619 (doi: 10.1109/ACCESS.2020.3028127).

Williams, L., Maximilien, E. M., and Vouk, M. (2003). “Test-driven development as a defect-reduction practice,” in Proceedings of the 14th ISSRE, Denver, Colorado, USA. 17.11.2003 - 20.11.2003, IEEE, pp. 34-45 (doi: 10.1109/ISSRE.2003.1251029).

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). “Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books,” in 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile. 07.12.2015 - 13.12.2015, IEEE, pp. 19-27 (doi: 10.1109/ICCV.2015.11).

Zini, J. E., and Awad, M. (2023). “On the Explainability of Natural Language Processing Deep Models,” ACM Computing Surveys (55:5), pp. 1-31 (doi: 10.1145/3529755).