كلية الحاسبات والذكاء الإصطناعي
Faculty of Computers & Artificial Intelligence
جامعة حلوان
Helwan University

HELWAN UNIVERSITY
Faculty of Computers and Artificial Intelligence
Information System Department

[Authorship identification]

A graduation project dissertation by:

[Sara Alaa Mohammed Ali (201900332)]

[Mahmoud Hossam El-deen Rashad(201900765)]

[Mostafa Ahmed Ashour Mohammed(201900810)]

[Ahmed Mahmoud Ahmed Ali (201900098)]

[Mohammed Emad Maihoub(201900715)]

[Nada Ayman Abdel Majeed Sayed(201900893)]

Submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computers & Artificial Intelligence, at the Information System Department, the Faculty of Computers & Artificial Intelligence, Helwan University

Supervised by:

[Ahmed Al-sayed]

June 2023

Authorship Identification

Abstract:

Authorship identification software is a computational tool that aims to identify the author of a given text. It is based on the idea that everyone has a unique writing style that can be identified through various linguistic features of their writing, such as the use of certain words, sentence structures, and punctuation patterns. The software uses machine learning algorithms to analyze these features and generate a writing style profile for each author. One area where authorship identification software is particularly useful is in forensic linguistics. Forensic linguists use language analysis to help solve crimes, and authorship identification software can assist in determining the author of a threatening letter or a ransom note. By analyzing the linguistic features of the text, the software can provide valuable information to investigators. Another area where authorship identification software is useful is in the detection of plagiarism. By comparing the writing style of a suspected plagiarized text to the writing style of potential sources, the software can help determine whether the text has been copied from another source or not. In addition, authorship identification software can also be used in literary analysis. By analyzing the writing style of various authors, the software can provide insights into their unique writing styles, which can be useful in understanding their works and comparing them to other authors. There are many challenges involved in developing authorship identification software. One challenge is developing algorithms that can accurately identify an author's writing style even when the text is noisy or incomplete. Another challenge is developing software that can handle multiple languages and dialects. Despite these challenges, authorship identification software has shown great potential in a variety of fields, and there is ongoing research to improve its accuracy and expand its capabilities.

Acknowledgment:

To be able to finish this project successfully we were not alone but we would thank every person who participated in making our goal come true even if it was just spiritual support.

Words cannot express our gratitude to our professor (Dr Ahmed El-Sayed) for his feedback, instructions and invaluable patience throughout the year. We could not have undertaken this journey and achieved our goal without him. As he was providing us generously with knowledge.

Thanks also should go to the librarians and study participants from the university, who impacted and inspired us.

Lastly, we would be remiss if we didn't mention our families, especially our parents. For their belief in us has kept our spirits and motivation high during this journey.

Table of Contents

Chapter 1

Introduction

In this chapter we will take a quick overview for the project, we will Discuss things like the project's objective & purpose, what is the project Scope and how we managed to achieve our purpose, what were the Constraints we passed by and how we overcame it.

1.1Gantt Chart:-

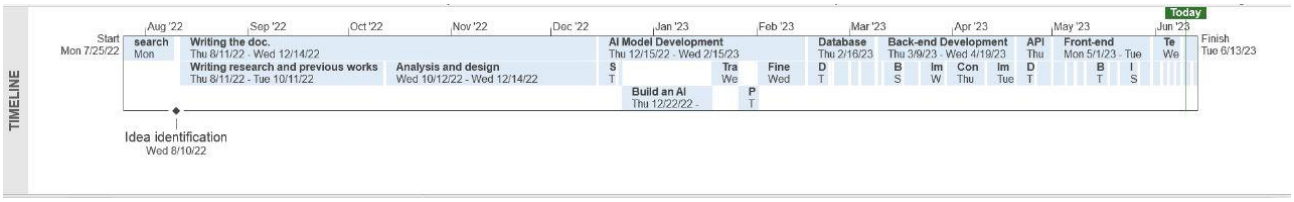
GANTT CHART

	Task Mode	Task Name	Duration	Start	Finish
1		**search for idea**	12 days	Mon 7/25/22	Tue 8/9/22
2		**Idea identification**	0 days	Wed 8/10/22	Wed 8/10/22
3		**Writing the doc.**	**90 days**	**Thu 8/11/22**	**Wed 12/14/22**
4		Writing research and previous works	44 days	Thu 8/11/22	Tue 10/11/22
5		Analysis and design	46 days	Wed 10/12/22	Wed 12/14/22
6		**AI Model Development**	**45 days**	**Thu 12/15/22**	**Wed 2/15/23**
7		Search for data set	4 days	Thu 12/15/22	Tue 12/20/22
8		Divide the data set for training and testing purposes	2 days	Wed 12/21/22	Thu 12/22/22
9		Build an AI model suitable for your application using machine learning python	20 days	Thu 12/22/22	Wed 1/18/23
10		Train the model using the prepared training data set	7 days	Wed 1/18/23	Thu 1/26/23
11		Perform testing and evaluation of the model's performance	5 days	Thu 1/26/23	Wed 2/1/23
12		Fine-tune the model based on the test results and iterate	11 days	Wed 2/1/23	Wed 2/15/23
13		**Database**	**15 days**	**Thu 2/16/23**	**Wed 3/8/23**
14		Design the database schema based on the application requirements	5 days	Thu 2/16/23	Wed 2/22/23
15		Set up the database management system PostgreSQL useing pgAdmin4 Tool	1 day	Thu 2/23/23	Thu 2/23/23
16		Create the necessary tables/collections and establish relationships between them	3 days	Fri 2/24/23	Tue 2/28/23
17		Implement database queries, including CRUD operations	3 days	Wed 3/1/23	Fri 3/3/23
18		implement necessary measures access control and security	4 days	Sat 3/4/23	Wed 3/8/23
19		**Back-end Development**	**30 days**	**Thu 3/9/23**	**Wed 4/19/23**
20		Set up the server-side environment	2 days	Thu 3/9/23	Fri 3/10/23
21		Build the server-side logic using Spring Boot	5 days	Sat 3/11/23	Thu 3/16/23

GANTT CHART

	Task Mode	Task Name	Duration	Start	Finish
22		Develop the necessary endpoints and routes for handling API requests using Spring Boot	3 days	Fri 3/17/23	Tue 3/21/23
23		Implement business logic and data processing on the server-side.	6 days	Wed 3/22/23	Wed 3/29/23
24		Connect to the database and perform CRUD operations (Create, Read, Update, Delete) as required	8 days	Thu 3/30/23	Mon 4/10/23
25		Implement user authentication and authorization	7 days	Tue 4/11/23	Wed 4/19/23
26		**API Development**	**8 days**	**Thu 4/20/23**	**Sun 4/30/23**
27		Design and develop the required APIs to expose functionality and data to the front-end and external clients	4 days	Thu 4/20/23	Tue 4/25/23
28		Implement API endpoints corresponding to different actions and data retrieval	4 days	Wed 4/26/23	Sun 4/30/23
29		**Front-end Development**	**22 days**	**Mon 5/1/23**	**Tue 5/30/23**
30		Searching for Web Layouts for Visual Enhancement	4 days	Mon 5/1/23	Thu 5/4/23
31		Collect images and icons for the website.	2 days	Fri 5/5/23	Sun 5/7/23
32		Determine the text that will be written on all pages	3 days	Mon 5/8/23	Wed 5/10/23
33		Build individual pages using HTML, CSS, and JavaScript	5 days	Thu 5/11/23	Wed 5/17/23
34		Create a framework using Vue.js) for the front-end	3 days	Thu 5/18/23	Sat 5/20/23
35		Integrate the front-end with the back-end through APIs	6 days	Sun 5/21/23	Fri 5/26/23
36		Fill the pages with dynamic data obtained from the API	3 days	Sat 5/27/23	Tue 5/30/23
37		**Testing and Deployment**	**7 days**	**Wed 5/31/23**	**Thu 6/8/23**
38		Perform thorough testing of the application's functionality, including front-end, back-end, and API testing	3 days	Wed 5/31/23	Fri 6/2/23
39		Identify and fix any bugs or issues discovered during testing	2 days	Sat 6/3/23	Sun 6/4/23
40		Deploy the website/app to a production environment or hosting platform	2 days	Mon 6/5/23	Tue 6/6/23

	Task Mode	Task Name	Duration	Start	Finish
37		**Testing and Deployment**	**7 days**	**Wed 5/31/23**	**Thu 6/8/23**
38		Perform thorough testing of the application's functionality, including front-end, back-end, and API testing	3 days	Wed 5/31/23	Fri 6/2/23
39		Identify and fix any bugs or issues discovered during testing	2 days	Sat 6/3/23	Sun 6/4/23
40		Deploy the website/app to a production environment or hosting platform	2 days	Mon 6/5/23	Tue 6/6/23
41		Monitor the application's performance and address any scalability or performance-related concerns.	2 days	Wed 6/7/23	Thu 6/8/23
42		**presentation and Demo.**	**3 days**	**Fri 6/9/23**	**Tue 6/13/23**
43		presentation	2 days	Fri 6/9/23	Sun 6/11/23
44		Demo	2 days	Mon 6/12/23	Tue 6/13/23



1.2Problem Statement:

The primary problem that our project aims to solve is the need to identify the author of a given text, which can be useful in a variety of fields. In literary analysis, it can help identify the author of an anonymous or disputed work. To solve this problem, authorship identification software must be able to identify specific linguistic features that are characteristic of an individual's writing style. These features can include word choice, sentence structure, punctuation patterns, and even the frequency and distribution of specific function words. Once these features have been identified, machine learning algorithms can be used to analyze them and generate a unique writing style profile for each author. However, there are several challenges involved in developing authorship identification software. One major challenge is the need to handle noisy or incomplete data, which is common in forensic and historical texts. Another challenge is the need to handle multiple languages and dialects, as writing styles can vary significantly across different languages and regions. Finally, the software must be user-friendly and accessible, so that individuals and organizations can easily use its services. Despite these challenges, authorship identification software has shown great potential in a variety of fields, and ongoing research is focused on improving its accuracy and expanding its capabilities to handle more complex linguistic features and data sets.

1.3Objectives And Scope:

The objective of the project is to develop a computational tool that can automatically identify the author of a given text, based on their unique writing style, and to provide accurate and reliable results. The project aims to achieve this by developing algorithms that can analyze various linguistic features, such as word choice, sentence structure, and

Punctuation patterns, to generate a writing style profile for each author. In addition to developing accurate algorithms, the project also aim to create a user-friendly interface that allows users to easily upload and analyze text documents.

The specific objectives of project include:

1-Developing algorithms that can accurately identify an author's writing style based on specific linguistic features.

2- Creating a user-friendly interface that allows users to upload and analyze text documents.

3-Testing the software on various datasets.

4-To make it possible that authors works are not copied and attributed to other authors.

1.4Scope:

Readers and Authors

1.5Report Organization:

In Chapter 1 an overview was written about the project as an introduction the objectives and scope then the working methodology we followed.

In Chapter 2 we will show the literature review about the scientific researches we read and what we concluded from each paper.

In Chapter 3 we will show the beginning of our analyzation for the project from the non-functional and functional requirements and then we will show the design phase that shows the diagrams as use case diagram, activity diagram and sequence diagram.

In Chapter 4 we will start showing the implementation of the project and the test cases we applied

In Chapter 5 we will show some test cases for our project.

In Chapter 6 we will show our conclusions after finishing everything and start suggesting for the teams completing after us.

1.6Work Methodology:

We used Agile Methodology

1.6.1Justifications of the used Approach:

Agile methodology is a software development approach that offers several benefits, including flexibility, collaboration, and iterative development. It can lead to faster delivery of working software, better communication and teamwork, and increased customer satisfaction. Agile methodology is particularly useful in projects where requirements are not well-defined or are subject to change, or in projects where speed and flexibility are essential. By emphasizing adaptability, customer input, and continuous improvement, agile methodology can help teams build high-quality software that meets customer needs and exceeds expectations.

Flexibility: Agile methodology is designed to be flexible and adaptable to changing requirements or circumstances. It allows for changes to be made during the development process, which can be particularly useful in projects where requirements are not well-defined or are likely to change.

1.6.2Advantages of the used Approach:

1-Faster time-to-market: Agile methodology allows for faster development and delivery of software by breaking down the project into smaller, more manageable iterations. This allows for quicker feedback and faster delivery of working software.

2-Customer satisfaction: Agile methodology is designed to prioritize customer satisfaction by delivering working software quickly, incorporating customer feedback into the development process, and ensuring that the software meets the customer's needs and expectations. Collaboration: Agile methodology emphasizes collaboration between team members, stakeholders, and end-users. This can lead to better communication, increased team morale, and a better understanding of the needs and goals of the project.

3-Continuous improvement: Agile methodology encourages continuous improvement through regular feedback and reflection. This helps to identify and

Address issues or challenges early on in the development process, leading to better quality software and a more efficient development process.

4-Risk mitigation: Agile methodology can help to mitigate risks by providing regular opportunities for testing and validation, allowing for issues to be identified and addressed early in the development process.

5-Transparency: Agile methodology emphasizes transparency and visibility into the development process, making it easier for stakeholders to track progress, identify issues, and provide feedback.

1.6.3Disadvantages of the used Approach:

1-Lack of predictability: Agile methodology is designed to be flexible and adaptable, which can make it difficult to accurately predict timelines, costs, and deliverables. This can be challenging for project managers who need to plan and manage resources effectively.

2-Scope creep: Agile methodology can make it easy for the scope of the project to expand beyond the original requirements. This can lead to added costs and delays, as well as a loss of focus on the original goals of the project.

3-Higher demands on team members: Agile methodology requires a high level of collaboration and communication between team members, which can be challenging for teams that are distributed across different locations or time zones. Additionally, team members need to be skilled in their areas of expertise and able to work effectively in an Agile environment.

4-Lack of documentation: Agile methodology prioritizes working software over documentation, which can make it difficult to maintain a clear record of the development process. This can be a challenge for teams that need to comply with regulatory or legal requirements.

5-Dependency on customer input: Agile methodology relies heavily on customer feedback and input, which can be challenging if the customer is not available or engaged in the development process.

6-Cultural resistance: Agile methodology requires a shift in mindset and culture, which can be challenging for teams and organizations that are used to working in a more traditional, hierarchical manner.

1.6.4Phases of the used Approach:

1-Planning: In the planning phase, the team identifies the goals of the project, defines the scope of the work, and develops a roadmap for how the project will be executed. This includes identifying user stories, backlog items, and prioritizing them based on business value.

2-Analysis: In the analysis phase, the team works with stakeholders and end-users to identify the requirements of the software. This involves defining the features and functionality of the software, as well as any technical requirements or constraints.

3-Design: In the design phase, the team creates a high-level architecture and design for the software. This includes developing wireframes, user interface designs, and other specifications.

4-Development: In the development phase, the team begins to write the code for the software. This is typically done in short iterations or sprints, with each iteration delivering a working software increment.

5-Testing: In the testing phase, the team tests the software at each iteration to ensure that it meets the requirements and specifications. This includes manual and automated testing, as well as continuous integration and delivery.

6-Deployment: In the deployment phase, the team releases the software to production. This involves deploying the software to the end-users and ensuring that it is running smoothly.

7-Maintenance: In the maintenance phase, the team provides ongoing support and maintenance for the software. This includes fixing bugs, adding new features, and making updates as needed.

Chapter 2

Literature Review

In this chapter includes citation of theoretical background, related work and

General constraints should be included.

2.1Authorship identification using ensemble learning (paper)[A]

Abstract:

We study proposed an authorship identification system combined with two feature extraction techniques that extract the information related to each author's writing style. To proposed approach depends on mainly two steps. The first step is selecting maximum suitable characteristics to explain the multiple authors writing styles from unstructured text documents. On the other hand, the second step is selecting multiple algorithms (RF, XGB, MLP, LR, Ensemble, and Distil BERT) to identify and classify the authors that belong to the actual text.

*Propose a framework to extract authors-related concise information from textual data using Count victories and term frequency-inverse document frequency (TFIDF) that automatically learn features without human interference.

2.1.1Authorship analysis:

*Authors in24 worked on a study related to software forensics. They worked on four essential authorship analysis regions: authorship detection, authorship characterization, similarity identification, and authorship discrimination.

2.1.2Language modelling:

*We use unseen data to identify how likely that author wrote it during the prediction phase. It would generate a high prediction value if that author wrote the textual data. This is the working process of language models for authorship attribution. If you deal with more than one language model, each model will be trained on a specific author's writing. There will be no change in the model's architecture; the model will be the same; change the training dataset to train the model to identify individual authors. These approaches also have some disadvantages, primarily related to stylometry-based approaches that have some problems during the selection of specific features and also some language dependencies.

*The proposed approach identifies the most suitable features using the count vectorizer and bi-gram TF-IDF related to all the earlier work. This approach also attained bench-marked results using an ensemble learning approach and an NLP-based language model to identify authors. We compare the proposed approach with the baseline approach, which shows that we outperform the baseline approach with better performance.

2.1.3Proposed approach:

*The proposed approach depends on multiple phases. The first step is data preprocessing to handle imbalanced class data, overfitting data, handling missing values, and handling a large and noisy dataset; the second step is the selection of useful features using the supreme feature engineering technique to extract the most important features that help the model for classification and identification of actual authors. The last step is the model selection to identify the author of the actual text. The proposed ensemble method and multi-depth Distil BERT model performed well on the dataset with higher accuracy than baseline approaches.

2.1.4Data pre-processing:

*The textual data sometimes may be noisier, and it requires appropriate data preparation for better classification purposes. The authorship analysis and identification are based upon the particular writing style of every author. We need to analyze the data in a way that does not change the actual meaning of the sentence and does not change the author's writing style. The various pre-processing steps were taken (identify missing values, check for duplicated values, and many more).

1-One way to analyze the dataset is by calculating word frequency to know how frequently words appear in an article. It is a key component to understanding the relevancy of a given article and its actual author.

2-To understand the text's context and convert the words to their meaningful base form, we use lemmatization. It is used with the technique that converts words to their base form, for example, "played" to "play."

3- Stop words are generally utilized in NLP to remove words that do not carry much helpful information. In the third step of pre- processing, we remove stop words to overcome the noise, such as (“is,” “a,” “the”). The removal of stop words does not affect the actual meaning of the sentence.

4- Due to word capitalization, sometimes it understands the same word as two different words. The model cannot differentiate between uppercase words and lower case words. To avoid this, we adjusted and converted all capital words to lower case words that do not change the meaning of the actual word.

5- Most authors used shortened forms and abbreviations of words in the text. We apply the contraction mapping technique to shorten the words or phrases by dropping or replacing a letter with an apostrophe. The contraction mapping is the process that drops the vowels from words. Contraction mapping is essential while working with textual data.

6- We use text blob because it provides noun phrase extraction, part- of-speech tagging, and sentiment analysis. The primary step of pre-processing is Part-of-speech (POS) tagging. It builds the parse trees used to construct “most named entities are nouns” (NERS) and extracts the relationship between words. It is also used for applying lemmatizes to return a word to its base form.

2.1.5Feature extraction:

*The goal of feature extraction is to extract the most important features from a dataset for better classification, and authorship identification.

*This study uses a Count Victories and bigram TFIDF techniques for feature extraction.

2.1.6Count victimizer:

*The count victimizer represents a word matrix. In this matrix, the columns are represented by unique words in the text, and the word count of the text represents the rows.

*we use the TF-IDF feature extraction technique to extract the important features from textual data. We used TF-IDF for text analysis; it extracts weighted features for boosting the execution process. The TF-IDF technique's weighted features take a dot product of term-frequency and inverse document frequency.

*The frequency count of features in a particular text document is Term-Frequency (Tf).

$TF = count_{t,d} / totalcount_d$

* The IDF identified the increase in term t, which is more informative during the model training

$idf = i/df_t$

*The represents the total number of documents

$*tf - idf = tf_{t,d} * log(idf)$

2.1.7Classification methods:

-Logistic regression:

*It used the logic function and predicted the probability of the discrete classes. It is a supervised learning algorithm. It uses a logistic sigmoid function to predict the target class. It is a machine learning model used widely for classification purposes.

-Multi-depth DistilBERT:

*In this study, we implemented a pre-trained multi-depth DistilBERT transformation model. Based on the previously existing models like BERT, we carry out various modifications by reducing the number of layers and feeding the last layer token-type embedding for each token.

*The results show that embedding from various layers provide higher representations and boost the model's overall performance.

*We divided our data into train, test, and validation categories.

* Training and testing sessions are not included in the validation data. The amount of the validation data is unimportant because it does not affect the accuracy value.

*The training set is 70%, the testing set is 20%, and the validation set is 10%.

-Ensemble classifiers:

*Ensemble learning is a prevalent technique in the research domain among researchers.

*This study used different machine learning algorithms for ensemble learning purposes.

*Ensemble learning is used to identify authors explicitly related to the article.

* We combine different ML classifiers and achieve better classification performance based on the voting mechanisms.

* We used a majority voting mechanism in which every single classifier in an ensemble learning predicts a class label when we get a new variable or instance.

* The class with high classifer prediction or majority votes is assigned as the target label of that variable or instance.

* The ensemble learning achieves better performance than the conventional single ML model.

*The proposed ensemble learning approach combines multiple classifers' predictions, and the final output depends upon the majority voting mechanism. We fine-tuned every single classifier to get a better result.

-Random forest:

*Random Forest is a classification algorithm used for the ensemble learning method.

* It is an ML ensemble classifier used for multiple tasks classification regression.

* It works by building several decision trees that utilize as an ensemble, and the output target label depends upon the votes taken from those trees.

* Random Forest decreases the over-fitting problem by making several decision trees.

* RF is also used to deal with complex data, unlike conventional ML classifiers.

-Extreme gradient boosting:

*Carries efficiency and memory resources.

*It consists of many weak learners that are parallels working because of this mechanism.

*XGboost is faster and gives more speed boost up.

*The ensemble algorithm uses Extreme Gradient Boosting to improve better classification performance.

*Extreme Gradient Boosting is a unique model that combines weak learning models into a stronger one.

*At each iteration, the residual error is optimized based on the previous predictor and optimized the loss function.

-Multi-layer perceptron (MLP):

*is mainly used for classification and prediction problems.

*To train on dataset it used function $F(X) : Rn \rightarrow Ro$. The o is the total output dimensions, where n is the input dimensions.

*We used a feature set $X = x1, x2, x3....xn$ with the target variable.

*Every node is called a neuron, and it has a nonlinear activation function

used for the classification or regression process.

Reached Accuracy: 91%

- ## 2.2Deep Learning based Authorship Identification (paper) [B]

2.2.1Abstract:-

In this project, we apply basic classification models and explore (GRU, LSTM and Bi-LSTM) at the sentence and article levels to identify the authors of a given piece of text. We deal with the pre-processing and feature vectorization of texts from the Reuters 50 50 C-50 data-set

2.2.2Introduction:-

Firstly, the quality of the dataset is inspected along with running a thorough analysis of the features to be used. Preprocessing of the dataset with TF-IDF implementation is conducted at the article level for every author in the training and testing datasets. Basic classification models are then trained so as to leverage the data-set features and a higher-level implementation (LSTM GRU). A comparative analysis of the results of each of these models is conducted, using different analytical metrics, and future scope for implementation of the project is suggested.

2.2.3Data Pre-Processing:-

1- Given that news articles may sometimes consist of casing errors, lower-casing has been implemented here to improve the efficiency of performance tasks. Instances of stop words in English viz. ”a”, ”the”, ”is”, ”are” and so forth are eliminated since they are low information words. We attempt to diminish the number of features that help keep our models within manageable sizes by their removal. In order to ensure high information gain, we have only retained those words that appear in at least 5 documents and ignored those words that frequent in more than 50% of the documents

2- TF-IDF algorithm was applied on textual data to form a sparse matrix with large feature size (since every word accounts for an individual feature) and the frequency matrix of stylistic features was scaled down to 0-1. The aggregate of both these matrices was the dataset on which the model trained subsequently

2.2.4Training

1- Standard Models: In order to justify the inclusion of stylistic features, we initially trained and evaluated the models without the same accuracy scores. The scores significantly improved on the inclusion of stylistic features. It was expected of SVM and Random Forest Classifiers to be able to generate clusters of different authors for all their written text on a generic dataset

Table 1. Stylistic Features in the dataset

Feature	Feature
Full-stops	Apostrophes
Comma	Line Breaks
Question marks	White spaces
Exclamation marks	Ellipsis
Ampersands	Left curly brackets
Percentage sign	Right curly brackets
Dollar Sign	Right slash
Left Parenthesis	Right Parenthesis
Colons	Semicolons
Asterisks	Accents
Average word length	Average no. of sentences

2- Deep learning implementation: In this section, we list out the deep learning models we used in our project. We implemented sentence level LSTM, GRU and BiLSTM and article level LSTM, GRU and BiLSTM.

For word representations we utilized GloVe word vectors of size 100. We utilized one of the most sophisticated models of glove.6B.100 where most of the words were represented

2.2.5Results

1- Standard Models the results for training using some standard models viz. Linear SVC, Random Forests, and Multinomial Naive Bayes classifiers are tabulated in table 2

Table 2. Results from Standard Models

heightModel	Accuracy	Precision-Recall
Linear SVC	84.75	0.88
Random Forest	92.2	0.95
Multinomial NB	87.4	0.98

2- It is also seen that GRU training accuracy is higher than that of LSTM which is mainly because GRU is much more efficient than LSTM when the data set is small. This model of sentence level GRU, LSTM and Bi-LSTM has few models and many parameters and hence could lead to over-fitting therefore, we implemented article level methods
3- We utilized the same C50 data set to implement article level deep learning models. We plotted training accuracies, losses and F1 Score for the same.
4- For the above models we used a hidden size of 300 with a learning rate of 0.01. It is seen from the above figures that the testing accuracies for article level GRU, LSTM and Bi-LSTM models are much better than the sentence level implementations.

Model	Accuracy	F1 score	Loss
Sentence Level GRU	37.14%	37.26%	0.066
Sentence Level LSTM	37.42%	37.41%	0.0488
Sentence Level BiLSTM	36.7%	36.85%	0.058
Article Level GRU	55%	53.45%	0.026
Article Level LSTM	65.13%	65.09%	0.023
Article Level BiLSTM	71%	71%	0.021

From the above table, it is seen that the testing accuracies of all the three article level deep learning models are much better than the sentence level models. It can also be seen that the Bi-LSTM model is the best model with higher accuracy, F1 score and less loss. Refer to the Appendix for confusion matrices of these models.

2.2.6Conclusion

In this project we implemented Linear SVM, Random Forest and Multinomial Naive Bayes models with Random Forest providing an accuracy of 92.2% but with lesser precision/recall score than Multinomial NB. Depending on the application, one metric can be weighed more than the other and the optimal model can be identified. It is also

seen that, Article level deep learning models perform better than sentence level models. Article level Bi-LSTM models have the highest accuracy of around 71%. Better stylometric features could be implemented on our standard models and hence they have better accuracies.

2.3Deep Learning based Authorship Identification (paper)[C]

2.3.1Abstract:

Authorship identification is an important topic in the field of Natural Language Processing (NLP). It enables us to identify the most likely author of articles, news or messages. Authorship identification can be applied to tasks such as identifying anonymous author, detecting plagiarism or finding ghost writer. In this project, we tackled this problem at different levels, with different deep learning models and on different datasets. Among all models we tested, article-level GRU achieves the best result of 69.1% accuracy on C50 dataset and 89.2% on Gutenberg dataset. We further studied authorship verification, on which task our Siamese network based model outputs 99.8% accuracy on both C50 and Gutenberg.

2.3.2Introduction:

A published author usually has a unique writing style in his works. The writing style is to some extent irrelevant with the topics and can be recognized by human readers. The process of identifying the author from a group of candidates according to his writing samples (sentences, paragraphs or short articles) is authorship identification. In this work, two labelled datasets are adopted to train and test our models. Several most advanced deep learning models are utilized at different levels to evaluate the accuracy of authorship identification. The contributions of this paper are summarized as follows.

• The authorship identification is performed on both a news dataset (Reuters 50 50) and a story dataset (Gutenberg) at both sentence level and article level.

• Gated Recurrent Unit (GRU) network and Long Short Term Memory (LSTM) network are implemented, tuned and evaluated on the performance of authorship identification.

• Siamese network is proposed to examine the similarity of two articles. It proves to be powerful on authorship verification.

2.3.3Datasets:

Two different datasets are exploited to train and test the model. The two datasets are Reuters 50 50 news dataset and Gutenberg story dataset.

2.3.4Data Pre-processing:

The pre-processing of the data in work primarily consists of two parts. They are respectively word representations and input batch alignment.

1- Word Representations

The GloVe [10] word vectors of size 50 is used as the pretrained word embeddings. The vocabulary contains 400,000 tokens. The GloVe vectors were used to initialize the word embeddings and the gradient are propagated to the embeddings. There could exist rarely-seen words which cannot be represented. Hence, we eliminated the occurrences of numbers and special characters to match the feature of our word representations. During the process of parsing, we trimmed each word to ensure that it does not contain any number or special character at both of its ends.

2- Input Batch Alignment

Batch input is adopted to make use of the parallel computing advantage of graphical processing unit (GPU), in order to accelerate the training process of our model. Since the batch has a fixed length as input, we truncated the input if it exceeds the fixed length. If the input doesn't reach the fixed length, we append several "magic word" to the end of the input. The "magic word" is a word created by ourselves, which doesn't exist in the world. In addition, words which cannot be found in the GloVe look-up table are also replaced by the "magic word". To eliminate the impact of the "magic words" on the result, we masks these words at the output,

Making the back-propagation of errors ignore these "magic words" and only extract the "real words" from the network.

2.3.5Models:

There are Four Models

2.3.6Sentence-Level GRU:

Sentence-level GRU predicts author on a small number (typically 3) of continuous sentences. In this model, the input unit is a word. For each time stamp, one word, represented as a word vector is sent to the GRU network, updating the hidden state

and producing some output. The equation for hidden state update and output computation is given in Eq. 1.

$$\begin{cases} z_t &= \sigma(W^{(z)}x_t + U^{(z)}h_{t-1} + b^{(z)}) \\ r_t &= \sigma(W^{(r)}x_t + U^{(r)}h_{t-1} + b^{(r)}) \\ \tilde{h}_t &= \tanh(r_t \circ U^{(h)}h_{t-1} + W^{(h)}x_t + b^{(h)}) \\ h_t &= (1 - z_t) \circ \tilde{h}_t + z_t \circ h_{t-1} \end{cases} \quad (1)$$

2.3.7Article-Level GRU:

Different from sentence-level GRU, article-level GRU takes a whole or more paragraphs as the input, and predict its author. In this model, the input unit is a sentence. For each time stamp, a complete sentence represented by the average of all its words' vectors is sent to the model. Same as the sentence-level model, hidden state will be updated and an output will be produced upon each input.

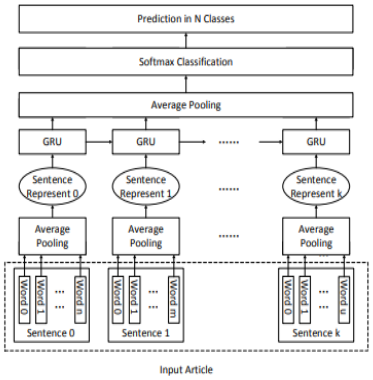


Figure 2: Article Level GRU Diagram

2.3.8Article-Level LSTM:

Article-level LSTM is quite similar to the article-level GRU, sharing the same type of input and data representation, and even the structure of the model, as shown in Figure 3. The only difference lies on the cell: in LSTM model, we use Eq. 2 instead of Eq. 1.

$$
\begin{cases}
i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}) \\
f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}) \\
o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}) \\
\tilde{c}_t &= \tanh(W^{(c)}x_t + U^{(c)}h_{t-1} + b^{(c)}) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \tanh(c_t)
\end{cases} \tag{2}
$$

2.3.9Article-Level Siamese Network:

Siamese network proves to be powerful on verification task that is, determining whether two input are of the same label. Although we mainly focus on authorship attribution, which is a typical identification job, it is meaningful to look into the verification job, where we make assertion whether two given articles are written by the same author.

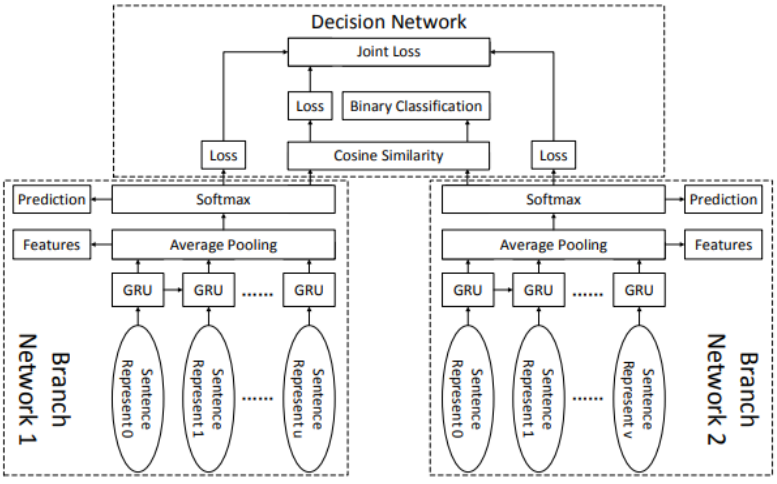


Figure 4: Siamese Network Diagram

2.3.10Results:

1- Sentence-Level GRU:

1.The test accuracy is too low compared to training accuracy.

2. The test accuracy is increasing as training accuracy increases. We concluded that this model has too few input and too many parameters and it could lead to over fitting.

2- Article-Level GRU:

The best accuracy for C50 and Gutenberg datasets are 69.1% and 89.2% respectively.

3- Article-Level LSTM:

The highest test accuracy of LSTM, however, is 62.7%, lower than what the previous model can achieve.

4- Article-Level Siamese Network:

For this model, we varied the parameter λ, which connects the loss from different parameters. The result on C50 dataset is shown in Table

λ	0.0005	0.001	0.005	0.01	0.5
Test accuracy	97.4%	99.8%	98.6%	97.4%	96.3%

2.3.11Conclusion:

Different deep learning models are studied on authorship identification. we designed and implemented 3 models for authorship identification, and also one model for authorship verification on C50 dataset and Gutenberg dataset. Article-level GRU performed best on authorship identification, outputing an accuracy of 69.1% on C50 and 89.2% on Gutenberg dataset. Besides, siamese network achieved an verification accuracy of 99.8% on both C50 and gutenberg dataset.

2.4Authorship Identification with language processing [D]

2.4.1Introduction:-

In the field of natural language processing (NLP), authorship attribution is a well-known task where the goal is to answer the question: who is the author of a given text or document?, based on linguistic patterns and stylistic markers.

Given:-

1- Corpus of sample documents with known authorship

2- Corresponding set of authors the task is to find who wrote a new unseen textual document.

The motives behind this are multiple and are of interest to the fields of humanities, law and intelligence, marketing and computer security.

Authorship attribution can be useful in a forensic investigation for cases of phishing, spam, threats or 10 cyberbullying and, for example, to identify the author of a threatening letter or extremist messages on social networks. The authorship can be evaluated on multiple scales. From a whole document written by only one author, to the sentence level to establish the authors of collaborative work.

The classical approaches to authorship attribution are based on statistical 15 methods. Researchers have applied deep learning methods with promising results [1] but these are offset by high complexity, and require a long training period and large datasets. However, deep architectures faced the lack of large datasets available in authorship attribution as the amount of data available per class is intrinsically limited by the amount of textual data that a single 20 person can produce. As a result, shallower neural models such as vanilla Recurrent Neural Networks (RNN) have achieved interesting results on author identification tasks [2].

Natural languages are composed of series of tokens where the order of appearance is essential. Moreover, textual documents can be seen as temporal signals where RNNs are able to remember previous inputs and take into account 35 token order. Nevertheless, RNNs are known to be very difficult to train and suffer from the

problem of vanishing gradient. RNNs are known to be very difficult to train and suffer from the problem of vanishing gradient.

A first solution to this problem was introduced in [3] which showed that more complex and deeper models such as LSTMs can handle properly this issue.

Another 40 path named Reservoir Computing (RC) was developed in the late 2000s in the fields of machine learning and computational neuroscience respectively under the names Echo State Network (ESN) and Liquid State Machine (LSM).

This RC principle stems from the fact that training only the output layer of a randomly constructed RNN is often enough to achieve excellent performances in practice.

45 The training of an ESN is thus easier, not only because it focuses only on the output layer, but also as it results in the solving of a simple system of linear equations.

Since early work on 19th century, authorship analysis has been viewed as a tool for answering literary questions on works of disputed or unknown authorship. The first computer-assisted approach aimed at solving the famous Federalist Papers case [4] (a collection of essays, a subset of which claimed by both Alexander Hamilton and James Madison). However, in certain cases, the results of authorship attribution studies on literary works were considered controversial [5]. In recent years, researchers have paid increasing attention to authorship analysis in the framework of practical applications, such as verifying the authorship of emails and electronic messages [6, 7], plagiarism detection [8], and forensic cases [9]. Authorship identification is the task of predicting the most likely author of a text given a predefined set of candidate authors and a number of text samples per author of undisputed authorship [10, 11]. From a machine learning point of view, this task can be seen as a single-label multi-class text categorization problem [12] where the candidate authors play the role of the classes. One major subtask of the authorship identification problem is the extraction of the most appropriate features for representing the style of an author, the so-called stylometry. Several measures have

been proposed, including attempts to quantify vocabulary richness, function word frequencies and part-of-speech frequencies. A good review of stylometric techniques is given by Holmes [13].

2.4.2Solution:-

N-Gram Feature Selection:-

The proposed method for variable-length n-gram feature selection is based on an existing approach for extracting multiword terms (i.e., word n-grams of variable length) from texts.

The original approach aimed at information retrieval applications (Silva). In this study, we slightly modified this approach in order to apply it to character n-grams for authorship identification. The main idea is to Compare each ngram with similar n-grams (either longer or shorter) and keep the dominant n-grams.

1- Selecting the Dominant N-Grams

To extract the dominant character n-grams in a corpus we modified the algorithm LocalMaxs introduced in [15]. It is an algorithm that computes local maxima comparing each n-gram with similar n-grams.

2- Representing the Glue

To measure the glue holding the characters of a n-gram together various measures

2.4.3Experimental Settings:-

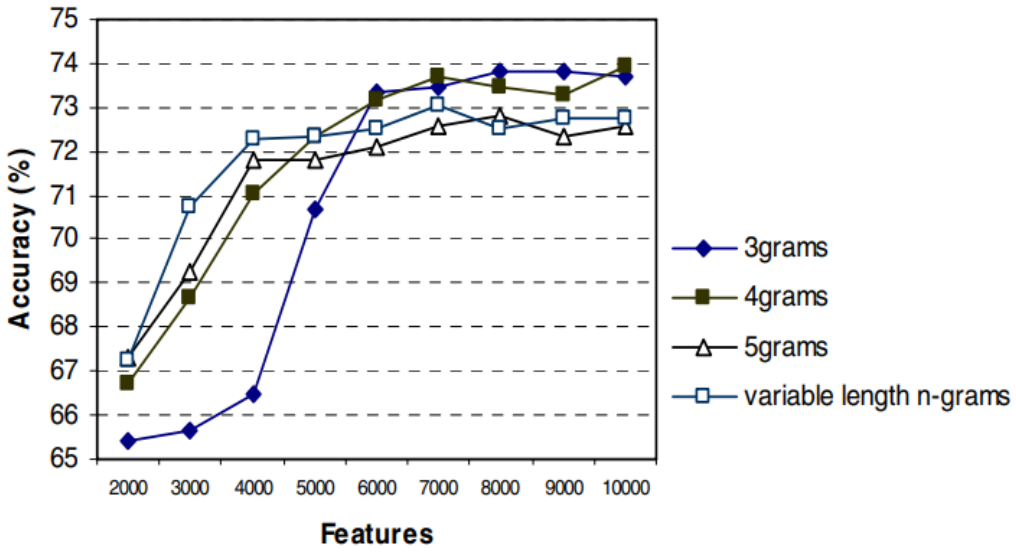
1- Corpus

In 2000, a large corpus for the English language, the Reuters Corpus Volume 1 (RCV1) including over 800,000 newswire stories, become available for research purposes. A natural application of this corpus is to be used as test bed for topic-based text categorization tasks [18] since each document has been manually classified into a series of topic codes (together with industry codes and region codes). There are four main topic classes: CCAT (corporate/industrial), ECAT (economics), GCAT (government/social), and MCAT (markets). Each of these main topics has many subtopics and a document may belong to a subset of these subtopics. Although, not

particularly designed for evaluating author identification approaches, the RCV1 corpus contains 'by-lines' in many documents indicating authorship.

2- Information Gain as Baseline

Most traditional feature selection methods are information-theoretic functions attempting to measure the significance of each feature in distinguishing between the classes.



Author Identification Experiments:-

An initial large feature set consisting of n-grams of variable length is extracted.

2.4.4Results:-

Character n-grams have been proved to be quite effective for author identification problems. Keselj et al. [14] tested this approach in various test collections of English, Greek, and Chinese text, improving previously reported results. Moreover,

A variation of their method achieved the best results in the

Ad- hoc authorship attribution contest.

2.5Authorship identification [E]

2.5.1INTRODUCTION:

What if one could determine who wrote a piece of text? Reveal the writers behind the texts? Was Shakespeare the real author of his plays? If there were a system that allowed us to identify the primary author, such a system would enable us to answer those questions. Author identification works to preserve intellectual property rights, and prevent theft of articles, attributing each article to its primary author. It would enable governments or institutions to give authors credit where credit is due.

2.5.2Problem Statement:

Lately, there has been increased literary theft, loss of literary rights, and concealment of the original author of a particular article or paper. Anybody can take a copy of anybody else's work and put it on a website or in a paper with his or her name on it.

The author identification process is significant for determining who deserves recognition for the text. It is not very easy to see an article in the name of another. It would be perfect if there were a system that could analyze and discover the unstructured article to assign the text to its primary author. As a result, NLP analysis has emerged to analyze articles and extract features to predict author name. This study will focus on NLP analysis of given articles and how the NLP, based on machine learning algorithms, will help to predict the author’s name.

2.5.3 LITERATURE REVIEW

First we have to talk about (Natural Language Processing (NLP)):

Natural Language Processing is the method used to aid machines to understand human natural language. It is a section of artificial intelligence that deals with the interaction between machines and humans using natural language. NLP aims to read, decode, analyze, understand, and make sense of human languages to derive meaning. Authorship identification is an essential topic in the field of NLP. It enables

us to identify the most likely writer of articles, news, text, or messages. Authorship identification can be used to identify anonymous writers or detect plagiarism.

2.5.4Authorship analysis

Authorship analysis is a challenging field that has evolved over the years. It is the procedure of finding the characteristics of a text in order to draw conclusions and analyze its authorship.

Stylometry is the root of authorship analysis, which means the statistical way to analyze the text style in order to characterize the author. The concept of authorship analysis can be defined and divided into three sections as follows:

- Authorship identification (authorship attribution): Finding the real writers of an article or document and the possibility of an author having written some text.

- Author profiling (characterization): getting the writer's profile or characteristics; for example, gender, age, background, and language.

- Similarity detection: Finding the similarity between the texts to determine the possibility of them having been produced by a single writer, without necessarily finding the real author. Commonly used in plagiarism detection

2.5.6Articles:

In study [1] the dataset was manually gathered from several Arabic websites. The dataset consists of 10 authors, with 10 articles for each author, while [2] developed a dataset containing text from different newspapers. The topics of these articles are about current events,

Political and medical issues. There are 20 authors, with 20 texts for each author in the training set, while, for the test set, there are 20 authors with five different texts for each.

In [3], the research consists of approximately 145 student essays of about 1400 words for each essays.

The essays are a real description of the Artificial Life documentary and the students' opinions about it. Thereby the topic, age, and level of education are constant.

In study [4], the dataset contains 20 different authors who write about Economics, Sports, Literature, and miscellaneous subjects. The articles were obtained from two Brazilian publications. Each writer has 30 pieces.

Work [5] is based on thirteen selected Nigerian writers from a Nigerian national daily. They harvested articles published from 2014 to 2016, and collected a total of 20 articles per author, so the total is 260 articles.

Study [6] used the ACL anthology network corpus dataset, which contains 23,766 papers and 18,862 authors. Also in the field of scientific papers, [7] used ACL papers. The dataset includes scientific papers published in several conferences and workshops. The selected papers were from 1965 to 2007; they classified all 2006 papers as development data and all 2007 papers as test data and the remaining papers were used for the training set.

2.5.7 Random Forest:

Pre-processing is a significant step in text mining. It means turning the text into a form that is predictable and analyzable. In [8], the preprocessing was divided into two types of features, depending on the requirement of the model. In the Bag of Words model for extracting content-based features, the author applied stop word removal and stemming, then extract the most frequent specific terms and consider them as a bag of words. The Bag of Words model is used for extracting n-gram features that tend to appear in the author's writing style and can be used to compare the writing style of one author to another, Therefore, as the first step, the author removes punctuation marks and extracts the most frequent character n-grams, word n-grams, and POS n-grams. Once the dataset is prepared and pre-process works done, feature extraction is needed to convert the data to vectors. In this step, the author uses the bag of words to represent the data vector, then uses classification algorithms; specifically, the Naive Bayes Multinomial (NBM) and Random Forest (RF). The author compares predictions with the most frequent

Content-based features with the accuracies of the most frequent character, word, and POS n-grams. The best results of author prediction are achieved when the author uses a combination of content-based features and n-grams, using the Random Forest classifier algorithm, with 91.87% accuracy. [9], on the other hand, evaluates the extracted features -unigram features, Latent semantic features, and similarity - by producing a supervised machine learning algorithm comparing Logistic regression, Random Forest, and SVM. The Random Forest tree produces higher performance than other models. with accuracy of up to 80.12%.

2.5.8Latent semantic analysis:

In [10], the author focused on representing the writing style of the authors; they used lexical-syntactic features. The work is divided into two levels; the first one is Phrase-level features: Word prefixes, Word suffixes, stop words, Punctuation marks, Word n-grams and Skip-grams; the second one is Character-level features: Vowel combination and Vowel permutation. The reason for using lexical-syntactic features is due to the ease of identification. The writer treated them with unsupervised classification, using several metrics to determine the similarity of the feature vectors of the documents of the known author against the documents of the unknown author. To establish the similarity of the documents, they used Latent semantic analysis (LSA), Jaccard similarity, Euclidean distance, Cosine similarity and Chebyshev Distance.

In [11], they create n-grams by sliding a window along the document to use as the features for the Document matrix. Thus, they develop an effective LSA by the use of character n-gram based analysis. In terms of accuracy, the percentage reached 75.68% for Dutch reviews in the PAN dataset.

In study [2], the text was analyzed in several ways: tokenizing, part-of-speech tagging, phrase parsing, and typed dependency parsing. Then they identified pronouns, function words and non-subject stylistic words. Therefore, they used knearest neighbors (KNN), support vector machine (SVM), and latent Dirichlet allocation (LDA) and made comparisons between the performance of different selected feature sets. They used the LIBSVM package for SVM. and a fivefold cross-

Validation way to select it from the candidate dataset. The core approach is a collection of n-gram features and SVM, excluding PCA feature extraction, and n is a positive integer. The LDA achieves higher performance by 98.45%.

2.5.9N-gram:

While classical documents are very well structured and provide various stylometric features, an e-mail [13] consists of a few paragraphs, written by an employee quickly, and frequently with syntactic and grammatical mistakes. All the sample e-mails are divided into groups to build a given author profile into one document that is subsequently divided into small blocks. They applied these processes: replace all numbers with 0; normalize the emails to printable ASCII; converted the emails to lowercase characters; remove white space; remove any punctuation; group all emails by author, to make a document that is divided into blocks.

In the Enron email dataset, the Equal Error Rate (EER) was14.35% for 87 employees for small block sizes. While the acquired results are hopeful, the author decides that more effort must be made to be usable in the real world. They discussed the limitations of their approach.

The accuracy decreased, not only when the number of authors number increased, but also when the number of blocks per employee decreased. They applied 5grams that achieve better results than 3- and 4-grams for a large number of blocks per user. However, in [14], pre-processing was required to produce the haracter n-gram profile. The author removed numerals from the text, eliminated all punctuation marks, partitioned the text into separate tokens, locating all possible n-gram for N = 2, 3, then making sure that each output n-gram in the list has its frequency, sorting the n-gram frequencies in descending order, and for each author, they build a profile size for bi-grams, tri-grams and quad-grams. The authors create the bi- & tri-grams from the author's text called Author's Profile. n-gram was used to calculate the dissimilarity between the frequency of the n-gram in the Author's Profile and the frequency in the test data.

2.6A framework for authorship identification of online messages: Writing-style features and classification techniques

2.6.1Introduction:

Things in authorship attribution studies have changed since the late 1990s, the vast amount of electronic texts available through Internet media has increased the need to deal with this information efficiently. This fact has had a major impactin scientific fields such as information retrieval, machine learning, and natural language processing (NLP). The impact of the development of these fields on the authorship attribution technique as shown:

Information retrieval research has developed effective techniques for representing and classifying large amounts of text.

Powerful machine learning algorithms are becoming available to handle multidimensional and sparse data, allowing for more expressive representations. Furthermore, standard evaluation methodologies for comparing different methods have been developed on the same reference data.

NLP research has developed tools capable of efficiently parsing text and introduced new forms of scales to represent style (eg, syntax-based features).

2.6.2Attribution Methods:

In every authorship-identification problem, there is a set of candidate authors, a set of text samples of known authorship covering all the candidate authors (training corpus), and a set of text samples of unknown authorship (test corpus); each one of them should be attributed to a candidate author. In this survey, we distinguish the authorship attribution approaches according to whether they treateach training text individually or cumulatively (per author). In more detail, some approaches concatenate all the available training texts per author in one big file and extract a cumulative representation of that author's style (usually called the author's profile) from this concatenated text. That is, the differences between texts written by the same author are disregarded. We examine such profile-basedapproaches22 first since early work in authorship attribution has followed this practice (Mosteller &

Wallace, 1964). On the other hand, another family of approaches requires multiple training text samples per author to develop an accurate attribution model. That is,

Each training text is individually represented as a separate instance of authorial style. Such instance-based approaches 33 are described in the next section, followed by hybrid approaches attempting to combine characteristics of profile-based and instance-based methods. We then compare these two basic approaches and discuss their strengths and weaknessesacross several factors.

* Previous studies on authorship attribution have proposed taxonomies of features to quantify the writing style, the so-called style markers, under different labels and criteria (Holmes, 1994; Stamatatos, Fakotakis, & Kokkinakis, 2000; Zheng et al., 2006). The current review of text representation features for stylistic purposes is mainly focused on the computational requirements for measuring them. First, lexical and character features consider a text as a mere sequence of word-tokens or characters, respectively. Note that although lexical features are more complex than character features, we start with them for the sake of tradition. Then, syntactic and semantic features require deeper linguistic analysis while application-specific features can be defined only in certain text domains or languages.

Stylometric Features:

- Lexical
- Character
- Syntactic
- Semantic
- Application-specific

We will explain each feature in detail and how to apply it:-

Lexical Features:

A simple and natural way to view a text is as a sequence of tokens grouped into sentences, with each token corresponding to a word, number, or punctuation mark. The very first attempts to attribute authorship were

based on simple measures such as sentence length counts and word length counts. A significant advantage of such featuresis that they can be applied to any language and any corpus with no additional requirements except the availability of a tokenizer (i.e.,a tool to segment text into tokens). However, for certain natural languages (e.g., Chinese), this is not a trivial task. For use of sentential information,a tool that detects sentence boundaries also should be available. In certain text domains with heavy use of abbreviations or acronyms (e.g., e-mail messages), this procedure may introduce considerable noise in the measures.

The vocabulary richness functions are attempts to quantify the diversity of the vocabulary of a text. Typical examples are the type-token ratio V/N, where V is the size of the vocabulary (unique tokens) and N is the total number of tokens ofthe text, and the number of hapax legomena (i.e., words occurring once).

Unfortunately, the vocabulary size depends heavily on text length (as the text length increases, the vocabulary also increases, quickly at the beginning and thenmore and more slowly).

The most straightforward approach to represent texts is by vectors of word frequencies. The vast majority of authorship attribution studies are (at least partially) based on lexical features to represent the style. This is also the traditional bag-of-words text representation followed by researchers in topic-based text classification . That is, the text is considered as a set of words, eachone having a frequency of occurrence disregarding contextual information.

However, there is a significant difference in style-based text classification: The most common words (articles, prepositions, pronouns, etc.) are found to be among the best features to discriminate between authors (Argamon & Levitan, 2005; Burrows, 1987). Note that such words are usually excluded from the feature set of the topic-based text-classification methods since they do not carryany semantic

information, and they are usually called “function” words.

A simple and very successful method to define a lexical feature set for authorshipattribution is to extract the most frequent words found in the available corpus (comprising all the texts of the candidate authors). Then, a decision has to be made about the amount of the frequent words that will be used as features.

In the earlier studies, sets of at most 100 frequent words were consideredadequate to represent the style of an author.

Another factor that affects the feature-set size is the classification algorithm that will be used since many algorithms overfit the training data when the dimensionality of the problem increases. However, the availability of powerful machine learning algorithms able to deal with thousands of features, such as support vector machines , enabled researchers to increase the feature-set size of this method. Koppel, Schler, and Bonchek-Dokow used the 250 most frequent words while Stamatatos extracted the 1,000 most frequent words. On a larger scale, Madigan et al. used all the words that appear at least twice in the corpus. Note that the first dozens of most frequent words of a corpus are usually dominated by closed-class words (articles, prepositions, etc.) After a few hundredwords, open-class words (nouns, adjectives, verbs) are the majority. Hence, whenthe dimensionality of this representation method increases, some content- specific words also may be included in the feature set.

Despite the availability of a tokenizer, word-based features may require additional tools for their extraction. This would involve simple routines such as conversion to lowercase to more complex tools such as stemmers , lemmatizersor detectors of common homographic forms . Another procedure used by van Halteren is to transform words into an abstract form. For example, the Dutch word “waarmaken” is transformed to “#L#6+/L/ken,” where the first “L” indicateslow frequency, “6+” indicates the length of the token, the second “L” a lowercasetoken, and “ken” are its last three characters.

The bag-of-words approach provides a simple and efficient solution, but

Disregards word-order (i.e., contextual) information. For example, the phrases“take on,” “the second take” and “take a bath” would just provide three occurrences of the word “take.” To take advantage of contextual information, word n-grams (n contiguous words also known as word collocations) have been proposed as textual features.

However, the classification accuracy achieved by word n-grams is not always better than individual word features). The dimensionality of the problem following this approach increases considerably with n to account for all the possible combinations between words. Moreover, the representation produced by this approach is very sparse since most of the word combinations are not encountered in a given (especially short) text, makingit very difficult to be handled effectively by a classification algorithm. Another problem with word n-grams is that it is quite possible to capture content-specificinformation rather than stylistic information. From another point of view, Koppel and Schler proposed various writing error measures to capture the idiosyncrasies of an author's style. To that end, they defined a set of spelling errors (e.g., letter omissions and insertions) and formatting errors (e.g., “all caps” words) and proposed a methodology to extractsuch measures automatically using a spell checker. Interestingly, human experts mainly use similar observations to attribute authorship; however, the availabilityof accurate spell checkers is still problematic for many natural languages.

Character Features:

Text is viewed as a mere sequence of characters. That way, various character-level measures can be defined, including alphabetic characters count, digit characters count, uppercase and lowercase characters count, letter frequencies, punctuation marks count, and so on. This type of information is easily available for any natural language and corpus, and it has been proven to be quite useful to quantify the writing style.

A more elaborate, although still computationally simplistic, approach is to extract frequencies of n-grams on the character level. For instance, the character 4-grams of the beginning of this paragraph would be: 11 |A_mo|, |_mor|, |more|, |ore_|,|re_e|, and so on. This approach is able to capture nuances of style, including lexical information (e.g., |_in_|, |text|), hints of contextual information (e.g.,|in_t|), use of punctuation and capitalization, and so on. Another advantage of this representation is its ability to be tolerant to noise. In cases where the texts in question are noisy, containing grammatical errors or making strange use of punctuation, as it usually happens in e-mail or in online forum messages, the character n-gram representation is not affected dramatically. For example, the

words “simplistic” and “simpilstc” would produce many common character trigrams. On the other hand, these two words would be considered different in a lexically based representation. Note that in style-based text categorization, such errors could be considered personal traits of the author . This information also is captured by character n-grams (e.g., in the uncommon trigrams |stc| and |tc_|). Finally, for oriental languages where the tokenization procedure is quite hard, character n-grams offer a suitable solution. the computational requirements of character n-gram features are minima

$$A : PP \rightarrow P : PREP + PC : NP$$

Syntactic Features: Baayen, van Halteren, and Tweedie (1996) were the first to use syntactic information measures for authorship attribution. Based on a syntactically annotated English corpus, comprising a semiautomatically produced full parsetree of each sentence, they were able to extract rewrite rule frequencies. Each rewrite rule expresses a part of syntactic analysis; for instance, the following rewrite rule: equation image.Means that an adverbial prepositional phrase is constituted by a preposition followed by a noun phrase as a prepositional complement. That detailed information describes both what the syntactic class of each word is and how the words are combined to form phrases or other structures. Experimental results have shown that this type of measure performs better than do vocabulary richness and lexical measures. On the other hand, it required a sophisticated andaccurate fully automated parser able to provide a detailed syntactic analysis of English sentences. Similarly, Gamon (2004) used the output of a syntactic parser to measure rewrite rule frequencies as described earlier.

Although the propose Syntactic features alone performed worse than did lexical features, thecombination of the two improved the results.

A more elaborate text-representation method is to employ syntactic information. The idea is that authors tend to unconsciously use similar syntactic patterns. Therefore, syntactic information is considered more a reliable authorial fingerprint in comparison to lexical information. Moreover, the success of function words in representing style indicates the usefulness of syntactic information since they are usually encountered in certain syntactic structures. Onthe other hand, this type of information requires robust and accurate NLP tools able to perform syntactic analysis of texts. This fact means that the syntactic measure extraction is a language-dependent procedure since it relies on the availability of a parser able to analyze a particular natural language with relativelyhigh accuracy. Moreover, such features will produce noisy datasets due to unavoidable errors made by the parser.

Tools that perform partial parsing can be used to provide syntactic features of varying complexity (Hirst & Feiguina, 2007; Luyckx & Daelemans, 2005; Uzuner & Katz, 2005). Partial parsing is between text chunking and full parsing, and can handle unrestricted text with relatively high accuracy. Hirst and Feiguina (2007) transformed the output of a partial parser into an ordered stream of syntactic Labels. For instance, the analysis of the phrase “a simple example” would produce The following stream of labels:

in words, a noun phrase consisting of a determiner, an adjective, and a noun. Then, they extracted measures of bigram frequencies from that stream to represent contextual syntactic information, and found this information useful todiscriminate the authors of very short texts (i.e., ∼200 words long).

An even simpler approach is to use just a part-of-speech (POS) tagger, a tool that assigns a tag of morpho-syntactic information to each word-token based on contextual information. Usually, POS taggers perform quite accurately in unrestricted text, and several researchers have used POS tag frequencies or POS tag n-gram frequencies to represent style . However, POS tag information provides only a hint of the structural analysis of sentences since it is not clear how the words are combined to form phrases or how the phrases are combined into higher level structures.

Finally, Karlgren and Eriksson (2007) described a preliminary model based on two syntactic features: adverbial expressions and occurrence of clauses within sentences. However, the quantification of these features is not the traditional relative frequency of occurrence within the text. They used sequence patterns aiming to describe the use of these features in consecutive sentences of the text. Essentially, this is an attempt to represent the distributional properties of the features in the text, a promising technique that can capture important stylistic properties of the author.

Sematic Features:

It should be clear by now that the more detailed the text analysis required for extracting stylometric features, the less accurate (and the noisier) the produced measures. NLP tools can be applied successfully to low-level tasks suchas sentence splitting, POS tagging, text chunking, and partial parsing, so relevant features would be measured accurately and so the noise in the corresponding datasets remains low. On the other hand, more complicated tasks such as full syntactic parsing, semantic analysis, or pragmatic analysis cannot yet be handled adequately by current NLP technology for unrestricted text. As a result, very few attempts have been made to exploit high-level features for stylometric purposes.

Perhaps the most important method of exploiting semantic information so far was described by Argamon et al. (2007). Inspired by the theory of Systemic Functional Grammar (SFG; Halliday, 1994), they defined a set of functional features that associate certain words or phrases with semantic information. In more detail, in SFG, the “CONJUNCTION” scheme denotes how a given clause expands on some aspect of its preceding context. Types of expansion could be “ELABORATION” (exemplification or refocusing), “EXTENSION” (adding new Information), or “ENHANCEMENT” (qualification). Certain words or phrases Indicate certain modalities of the “CONJUNCTION” scheme. For example, the Word “specifically” is used to identify a “CLARIFICATION” of an “ELABORATION” ofa “CONJUNCTION” while the phrase “in other words” is used to identify an “APPOSITION” of an “ELABORATION” of a “CONJUNCTION.” To detect such semantic information, they used a lexicon of words and phrases produced semiautomatically based on online thesauruses including WordNet. Each entry in the lexicon associated a word or phrase with a set of syntactic constraints (in the form of allowed POS tags) and a set of semantic properties. The set of functional measures, then, contained measures showing, for instance, how many “CONJUNCTIONs” were expanded to “ELABORATIONs” or how many “ELABORATIONs” were elaborated to “CLARIFICATIONs,” and so on. However, noinformation was provided on the accuracy of those measures. Experiments of authorship identification on a corpus of English novels of the 19th century showed that the functional features

can improve the classification results when combined with traditional function-word features.

Word “specifically” is used to identify a “CLARIFICATION” of an “ELABORATION” ofa “CONJUNCTION” while the phrase “in other words” is used to identify an “APPOSITION” of an “ELABORATION” of a “CONJUNCTION.” To detect such semantic information, they used a lexicon of words and phrases produced semiautomatically based on online thesauruses including WordNet. Each entry in the lexicon associated a word or phrase with a set of syntactic constraints (in the form of allowed POS tags) and a set of semantic properties. The set of functional measures, then, contained measures showing, for instance, how many “CONJUNCTIONs” were expanded to “ELABORATIONs” or how many “ELABORATIONs” were elaborated to “CLARIFICATIONs,” and so on. However, noinformation was provided on the accuracy of those measures. Experiments of authorship identification on a corpus of English novels of the 19th century showed that the functional features can improve the classification results when combined with traditional function-word features.

Application-Specific Features:

The previously described lexical, character, syntactic, or semantic features are application-independent since they can be extracted from any textual data given the availability of the appropriate NLP tools and resources required for their measurement. Beyond that, one can define application-specific measures to better represent the nuances of style in a given text domain. This section reviews the most important of these measures.

The application of the authorship attribution technology in domains such as e-mail messages and online-forum messages revealed the possibility to define structural measures to quantify the authorial style. Structural measures include the use of greetings and farewells in the messages, types of signatures, use of indentation, paragraph length, and so on (de Vel et al., 2001; Li, Zheng, & Chen, 2006; Teng, Lai, Ma, & Li, 2004; Zheng et al., 2006). Moreover, provided the texts in question are in HTML format, measures related to HTML tag distribution (de Vel et al., 2001), font-color counts, and font-size counts (Abbasi & Chen, 2005) also can be defined. Apparently, such features can be defined only in given text genres. Moreover, they are particularly important in very short texts where the stylistic properties of the textual content cannot be adequately represented using application-independent methods; however, accurate tools are required for their extraction. Zheng et al. (2006) reported that they had difficulties in accurately measuring their structural features.

In general, the style factor of a text is considered orthogonal to its topic. As a result, stylometric features attempt to avoid content-specific information to be more reliable in cross-topic texts. However, in cases in which all the available texts for all the candidate authors are on the same thematic area, carefully selected, content-based information may reveal some authorial choices. To bettercapture the properties of an author's style within a particular text domain, content-specific keywords can be used. In more detail, given that the texts in question deal with certain topics and are of the same genre, one can define certain words frequently

used within that topic or that genre. For example, in the framework of the analysis of online messages from the newsgroup misc.forsale. computers, Zheng et al. (2006) defined content-specific keywords such as “deal,” “sale,” or “obo” (i.e., or best offer). The difference of these measures and the function words discussed earlier is that they carry semantic information and are characteristic of particular topics and genres. It remains unclear how to select such features for a given text domain.

2.6.3Suggestions for feature selection in authoring attribution:

The feature sets used in authorship attribution studies often combine many types of features. In addition, some feature types such as lexical and character features can considerably increase the dimensionality of the feature set. In such cases, feature selection algorithms can be applied to reduce the dimensionality of the representation. That way, the classification algorithm is helped to avoid overfitting on the training data.

In general, the features selected by these methods are examined individually on the basis of discriminating the authors of a given corpus; however, certain features that seem irrelevant when examined independently may be useful in combination with other variables. In this case, the performance of certain classification algorithms that can handle high-dimensional feature sets (e.g., SVM)might be diminished by reducing the dimensionality. To avoid this problem, feature subset selection algorithms examine the discriminatory power of feature subsets . For example, Li et al. (2006) described the use of a genetic algorithm to reduce an initial set of 270 features to an optimal subset for the specific training corpus comprising 134 features. As a result, the classification performance improved from 97.85% (when the full set was used) to 99.01% (when the optimal set was used).

However, the best features may strongly correlate with one of the authors due to content-specific rather than to stylistic choices (e.g., imagine we have two authors for whom there are articles about politics for the one and articles about sports for the other). In other words, the features identified by a feature selection algorithm may be too corpus-dependent with questionable general use. On the other hand, in the seminal work of Mosteller and Wallace (1964), the features were carefully selected based on their universal properties to avoid dependency on a specific training corpus.

The most important criterion for selecting features in authorship attribution tasks is their frequency. In general, the more frequent a feature, the more stylistic variation it captures. Forsyth and Holmes (1996) were the first to compare (character n-gram) feature sets selected by frequency with feature sets selected by distinctiveness; they found the latter more accurate. However, they restricted the size of the extracted feature sets to a relatively very low level (96 features). Houvardas and Stamatatos (2006) proposed an approach for extracting character n-grams of variable length using frequency information only. The comparison of this method with information gain, a well-known feature selection algorithm individually examining the discriminatory power of features (Forman, 2003), showed that the frequency-based feature set was more accurate for feature sets comprising up to 4,000 features. Similarly, Koppel, Akiva, and Dagan (2006) presented experiments comparing frequency-based feature selection with odds ratio, another typical feature selection algorithm using discrimination information (Forman, 2003). More importantly, the frequency information they used was not extracted from the training corpus. Again, the frequency-based feature subsets performed better than did those produced by odds ratio. When the frequency information was combined with odds ratio, the results were further improved.

Koppel, Akiva, and Dagan (2006) also proposed an additional important criterion for feature selection in authorship attribution, the instability of features. Given a number of variations of the same text, all with the same meaning, the features that remain practically unchanged in all texts are considered stable. In other words, stability may be viewed as the availability of “synonyms” for certain language characteristics. For example, words such as “and” and “the” are very stable since there are no alternatives for them. On the other hand, words such as “benefit” or “over” are relatively unstable since they can be replaced by “gain” and “above,” respectively, in certain situations. Therefore, unstable features are more likely to indicate stylistic choices of the author. To produce the required variations of the same text, Koppel, Akiva, and Dagan (2006) used several machine translation programs to generate translations from English to another language and then back to English. Although the quality of the produced texts wasobviously low, this procedure was fully automated. Let {d1, d2, … ,dn} be a set of texts and

Equation image a set of variations of the ith text, all with roughly the same

Meaning. For a stylometric feature c, let equation image be the value of feature c in the jth variation of the ith text and equation image. Then, the instability of c is defined by:

$$IN_c = 1 - \frac{\sum_i \left[k_i \log k_i - \sum_j c_i^j \log c_i^j \right]}{\sum_i k_i * \log m}$$

Experiments showed that features selected by the instability criterion alone were not as effective as features selected by frequency; however, when the frequency and the instability criteria were combined, the results were much better.

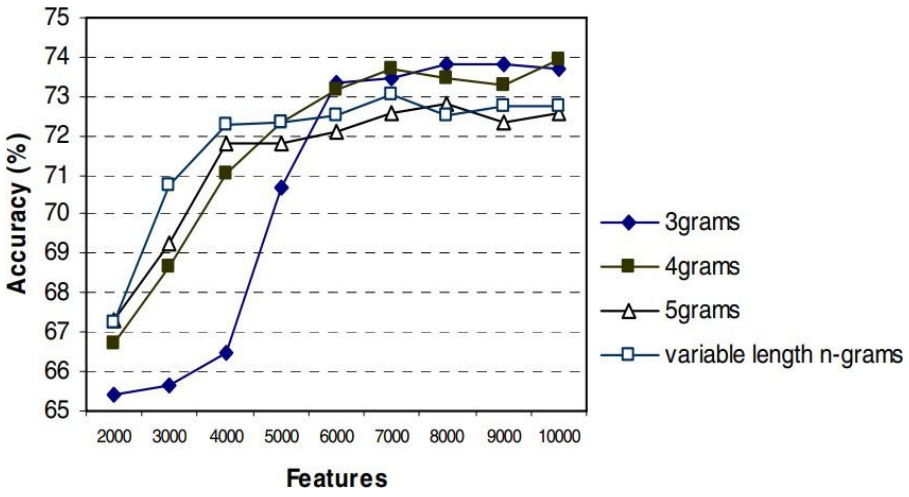
Another approach to reduce dimensionality is via feature extraction (Sebastiani, 2002). Here, a new set of “synthetic” features is produced by combining the initial set of features. The most traditional feature-extraction technique in authorship attribution studies is the principal components analysis, which provides linear combinations of the initial features. The two most important principal components can, then, be used to represent the texts in a two-dimensional space . However, the reduction of the dimensionality to a single feature (or a couple of features) has the consequence of losing too much variation information. Therefore, such simple features are generally unreliable to be used alone. Another, more elaborate feature-extraction method was described by Zhang andLee (2006). They first built a suffix tree representing all the possible character n- grams of the texts and then extracted groups of character n-grams according to frequency and redundancy criteria. The resulting key-substring-groups, each one accumulating many character n-grams, were the new features. The application ofthis method to authorship

2.7N-Gram Feature Selection for Authorship Identification

2.7.1Introduction:

In this paper, we propose a feature selection method for variable-length n-grams based on a different view. The original idea is based on previous work for extracting multiword terms (word n-grams of variable length) from texts in the framework of information retrieval applications.

2.7.2Accuracy:



2.7.3Methodology:

Since early work on 19th century, authorship analysis has been viewed as a tool for answering literary questions on works of disputed or unknown authorship.

In recent years, researchers have paid increasing attention to authorship analysis in the framework of practical applications, such as verifying the authorship of emails and electronic messages, plagiarism detection, and forensic cases.

One major subtask of the authorship identification problem is the extraction of the most appropriate features for representing the style of an author, the so-called stylometry. Several measures have been proposed, including attempts to quantify vocabulary richness, function word frequencies and part-of-speech frequencies. A good

review of stylometric techniques is given by Holmes.

A promising alternative text representation technique for stylistic purposes makes use of character n-grams (contiguous characters of fixed length.

Character n-grams are able to capture complicated stylistic information on the lexical, syntactic, or structural level.

Character n-grams have been proved to be quite effective for author identification problems. Keselj et al. Tested this approach in various test collections of English, Greek, and Chinese text, improving previously reported results. Moreover, a variation of their method achieved the best results in the ad- hoc authorship attribution contest

To keep dimensionality on low level, we used words longer than 5 characters as an alternative for longer n-grams.
However, the results when using the additional words were not encouraging. It would be interesting for one to explore the full use of long n-grams as well as the distribution of selected n- grams into different n-gram lengths especially when texts from different natural languages are tested.

Chapter 3:

System Analysis and Design

In this chapter we are going to discuss and go deeper in Author Identification website

Design and present its diagrams and database

6/14/2023

3.1System Analysis:

3.1.1User Requirements:

- can create new account
- can login to his personal account using the email address and the password
- can search for the author of the text
- can view his own profile
- can update his own profile
- can view the history of his search
- can delete any of his search

3.2 System Requirements:

3.2.1User:

-can create new account:

User input: email, password, age, username Output: open home page

Situation: show error message if the account is already existing

-can login to his personal account using the email address and the password User input: email, password

Output: open home page

Situation: show error message in case of wrong email or password

-can search for the author of the text User input: text

Output: the name of the author and the accuracy

-can view his own profile System input: user ID Output: view profile

-can update his own profile

User input: email, password, age, username

Output: show message which prompts the data updated successfully

Situation: show error message if the new data's pattern does not match the current pattern. so, data will not be changed.

-can view the history of his search System input: user ID

Output: view the history of his search page

-can delete any of his search System input: search ID

Output: show message which prompts the search deleted successfully

3.2.2Non-Functional Requirements:

-Easy to search the information about the author.

-The web page load time should not exceed 1 second.

-Any operation that will be done by any system user will not exceed 1 second.

-Each modification operation requires user password.

-To use the system, it is a must to have an account and be logged in / registered.

-Password encryption to enhance privacy and security.

-Show fake password in password field while login process instead of the one inputted by user.

-There is no way to use fake accounts. As it is required to check email to get the personal login ID.

-Each password must have a pattern to follow (At least 8 characters, a mixture of both uppercase and lowercase letters, a mixture of letters and numbers, inclusion of at least one special character and don't have <, >, ‘,”).

-Each ID must be unique.

-Each class must have only one instance.

3.3Functional Requirements:

- can create new account

- can login to his personal account using the email address and the password
- can search for the author of the text
- can view his own profile
- can update his own profile
- can view the history of his search
- can delete any of his search

3.4Non-Functional Requirements:

-Easy to search the information about the author.

-The web page load time should not exceed 1 second.

-Any operation that will be done by any system user will not exceed 1 second.

-Each modification operation requires user password.

-To use the system, it is a must to have an account and be logged in / registered.

-Password encryption to enhance privacy and security.

-Show fake password in password field while login process instead of the one inputted by user.

-There is no way to use fake accounts. As it is required to check email to get the personal login ID.

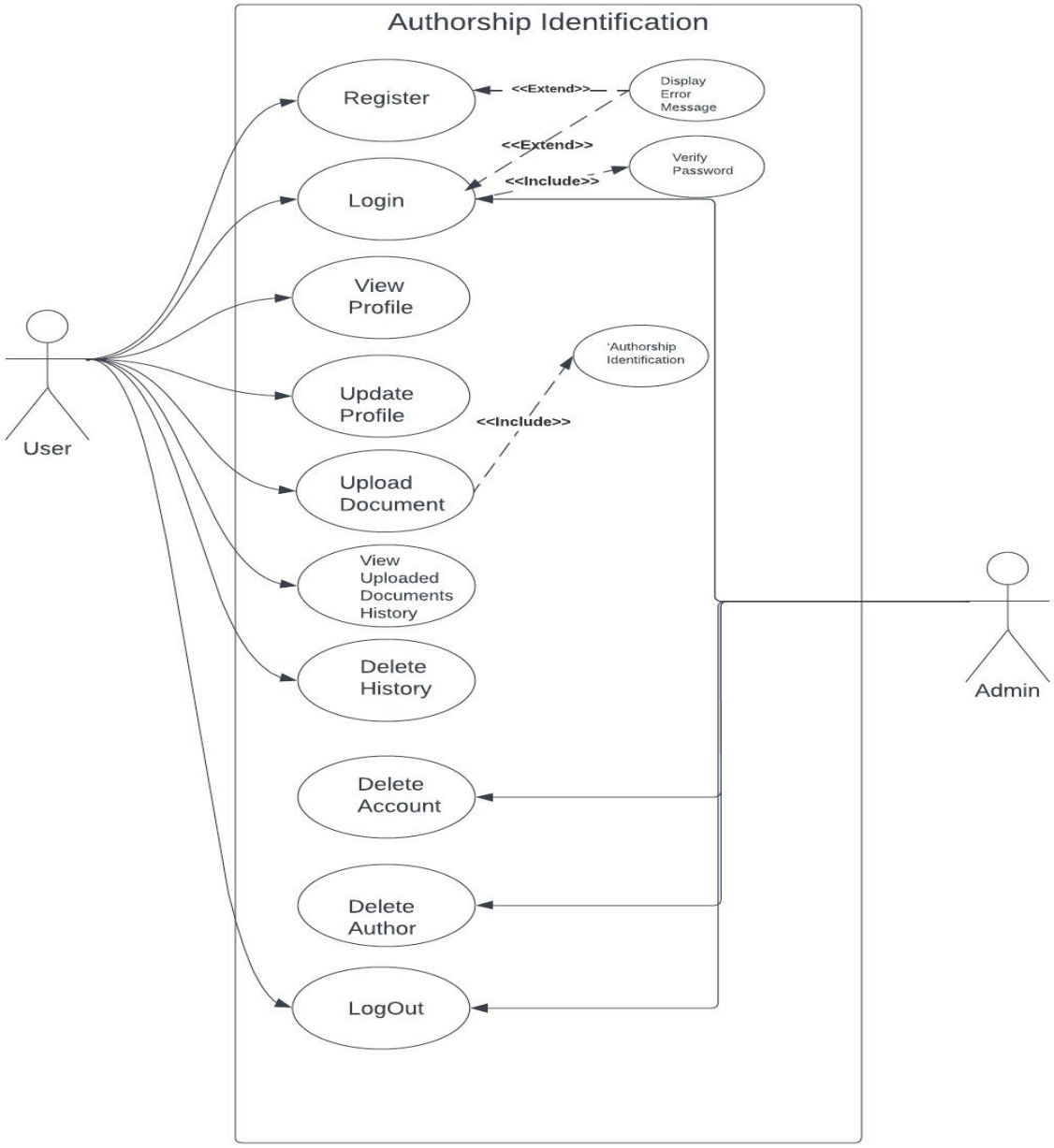
-Each password must have a pattern to follow (At least 8 characters, a mixture of both uppercase and lowercase letters, a mixture of letters and numbers, inclusion of at least one special character and don't have <, >, ‘,”).

-Each ID must be unique.

-Each class must have only one instance.

3.5System Design:

3.5.1Use Case Diagram:



Use case name		Register
Actors		User
Pre-conditions		User is not logged in to the system and goes to the login page.
	Description	1. The System prompts user for registration information, Username, password, etc. 2. The user enters in their information.
	Post-conditions	System verifies information and creates account and is returned to the home page as a logged in user.
Alternative flow and exceptions		1. User clicks submit after entering information system asked for. 2. System displays information with appropriate message to correct invalid information. 3. User re-enters information.
Non-functional requirements		Registration process must be secure and fast.

Use case name		Login
Actors		Admin, User.
Pre-conditions		Login page is opened.
	Description	The actor will open home page, and login if the password is verified.
	Post-conditions	The actor can do all his operations.
Alternative flow and exceptions		Fail to login and stay in the Login page.
Non-functional requirements		Login process must be secure and fast.

Use case name		View profile
Actors		User.
Pre-conditions		User must be logged in.
	Description	The actor will click the profile button, the profile data page will be opened and can edit or update his/her data and password.
	Post-conditions	Data must be saved in the database if changed.
Alternative flow and exceptions		Fail to logout as an operation is working, still in current page.
Non-functional requirements		Data must be secure.

Use case name		Update profile
Actors		User.
Pre-conditions		User must be logged in.
	Description	The actor will click the profile button, the profile data page will be opened and can edit or update his/her data and password.
	Post-conditions	Data must be saved in the database if changed.
Alternative flow and exceptions		Fail to logout as an operation is working, still in current page.
Non-functional requirements		Data must be secure.

Use case name		Upload Document
Actors		User.
Pre-conditions		User must be logged in.
	Description	The actor will click the upload pdf button, the pdfs on his own device will be opened so he chooses and it will be uploaded.
	Post-conditions	Data must be saved in the database if changed.
Alternative flow and exceptions		Fail to logout as an operation is working, still in current page.
Non-functional requirements		Data must be secure.

Use case name		View Uploaded Documents History
Actors		User.
Pre-conditions		User must be logged in.
	Description	The actor will click the View History button, the page including all the uploaded pdfs will be opened.
	Post-conditions	Data must be saved in the database if changed.
Alternative flow and exceptions		Fail to logout as an operation is working, still in current page.
Non-functional requirements		Data must be secure.

Use case name		Delete Documents History
Actors		User.
Pre-conditions		User must be logged in.
	Description	The actor will click the Delete History button, All the history will be deleted ,the page including all the uploaded pdfs will be opened and shown empty.
	Post-conditions	Data must be saved in the database if changed.
Alternative flow and exceptions		Fail to logout as an operation is working, still in current page.
Non-functional requirements		Data must be secure.

Use case name		View Uploaded Documents History
Actors		User.
Pre-conditions		The Uploaded Documents History page must be opened.
	Description	The actor will click the Star button, the page including all the added to favorites will be opened.
	Post-conditions	Data must be saved in the database if changed.
Alternative flow and exceptions		Fail to logout as an operation is working, still in current page.
Non-functional requirements		Data must be secure.

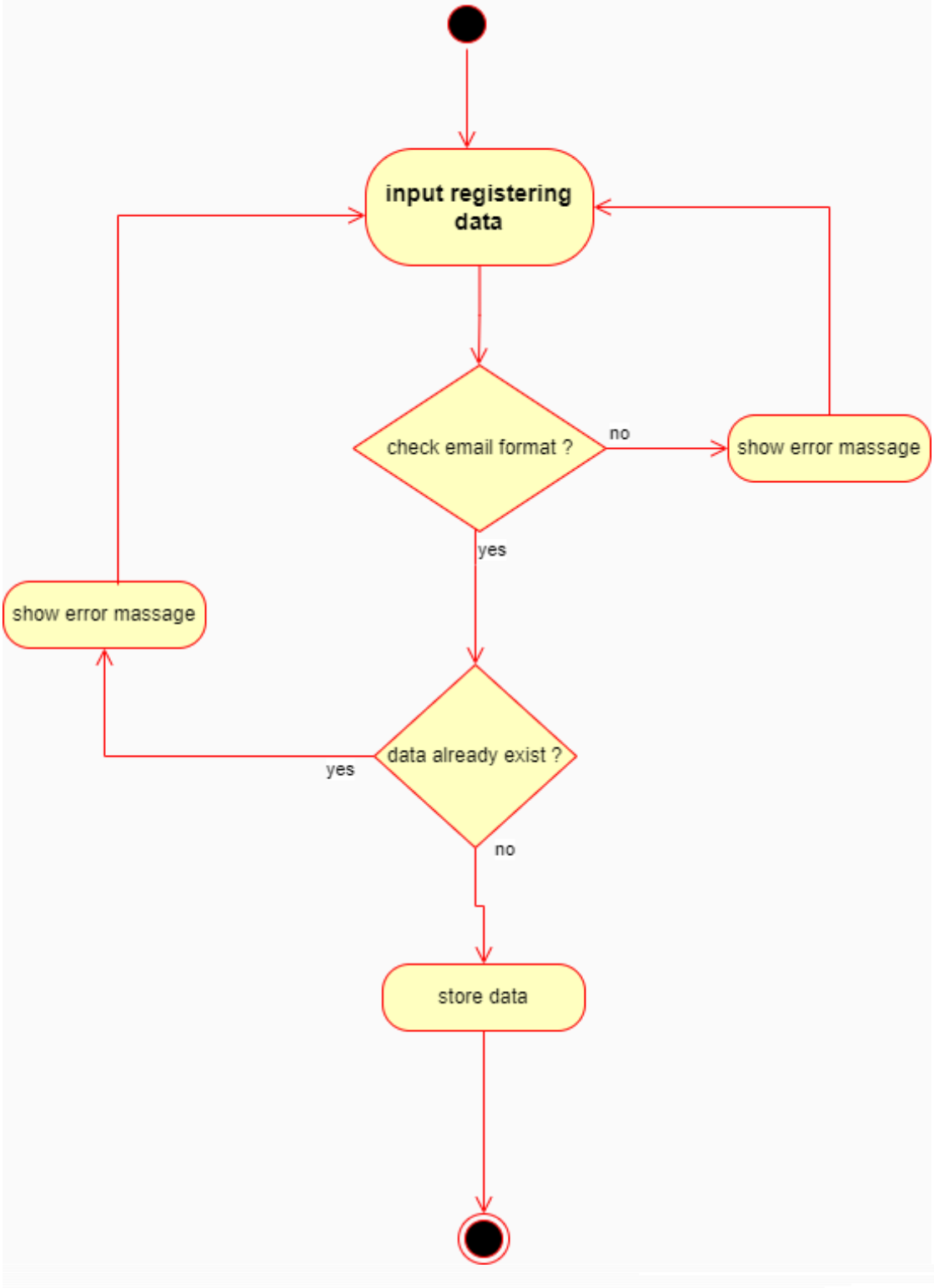
Use case name		Delete Author And His Books
Actors		Admin
Pre-conditions		Admin logged in successfully.
	Description	After logging in, the admin dashboard will be shown and (s)he can select the operation (s)he wants from the buttons which are already exist in the dashboard and do the required operation.
	Post-conditions	The changes must be saved in the database and the admin will return to the dashboard to be able to select another operation to do or logout.
Alternative flow and exceptions		Operation failed, show error message.
Non-functional requirements		All operations' processes must be secure and fast.

Use case name		Delete User Account
Actors		Admin
Pre-conditions		Admin logged in successfully.
	Description	After logging in, the admin dashboard will be shown and (s)he can select the operation (s)he wants from the buttons which are already exist in the dashboard and do the required operation.
	Post-conditions	The changes must be saved in the database and the admin will return to the dashboard to be able to select another operation to do or logout.
Alternative flow and exceptions		Operation failed, show error message.
Non-functional requirements		All operations' processes must be secure and fast.

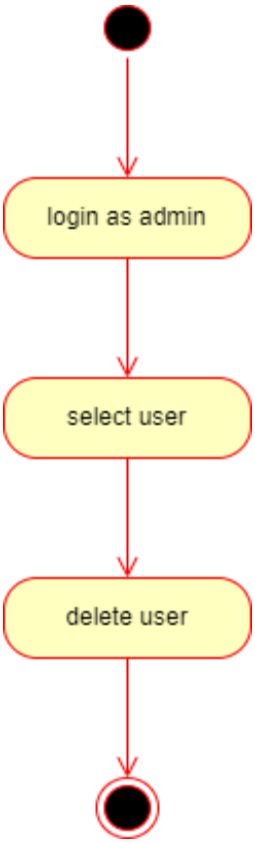
Use case name		Logout
Actors		Admin, User.
Pre-conditions		Actor's dashboard is opened.
	Description	The actor will click the logout button, and the home page will be opened instead of current one.
	Post-conditions	The actor cannot do any operation until (s)he login again.
Alternative flow and exceptions		Fail to logout as an operation is working, still in current page.
Non-functional requirements		Logout process must be secure and data are safe.

3.5.2Activity Diagram:

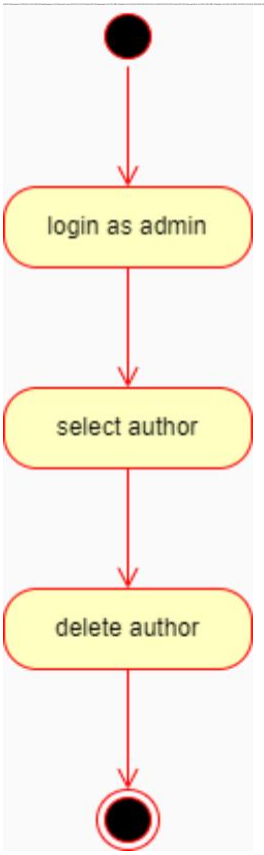
Register:



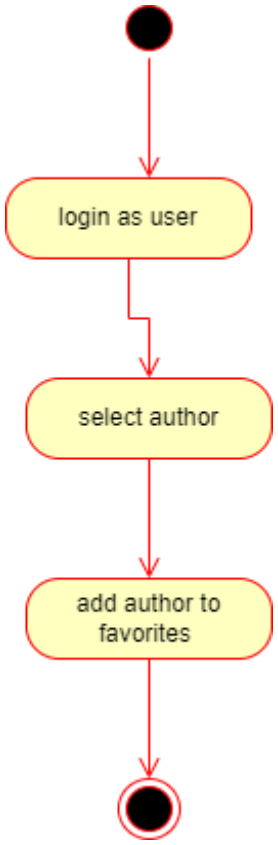
Delete User:



Delete Author:

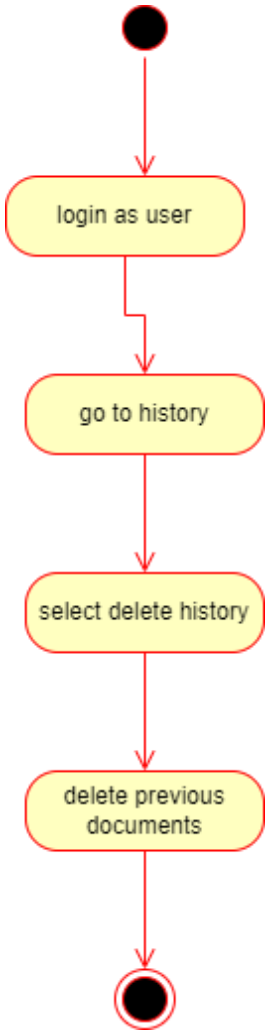


Add To Favorites:

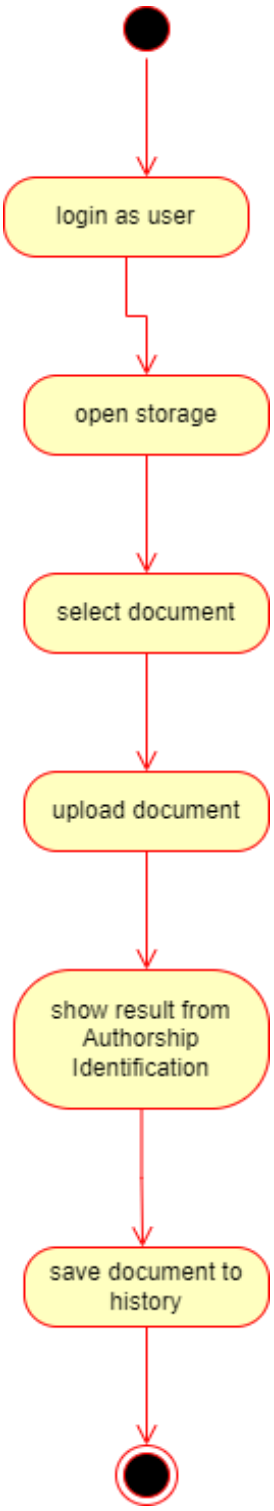


View Uploaded Document History:

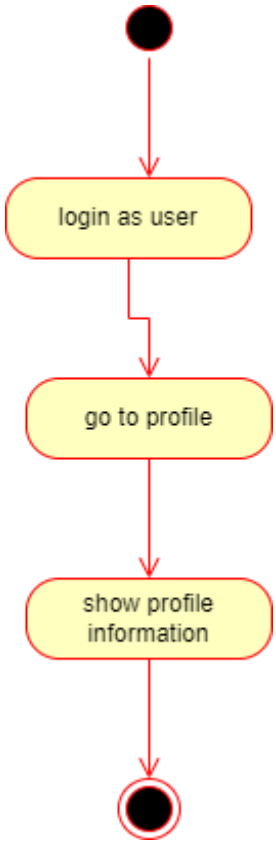
Delete History:



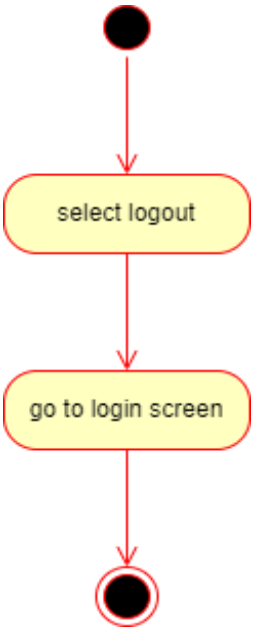
Upload Document:



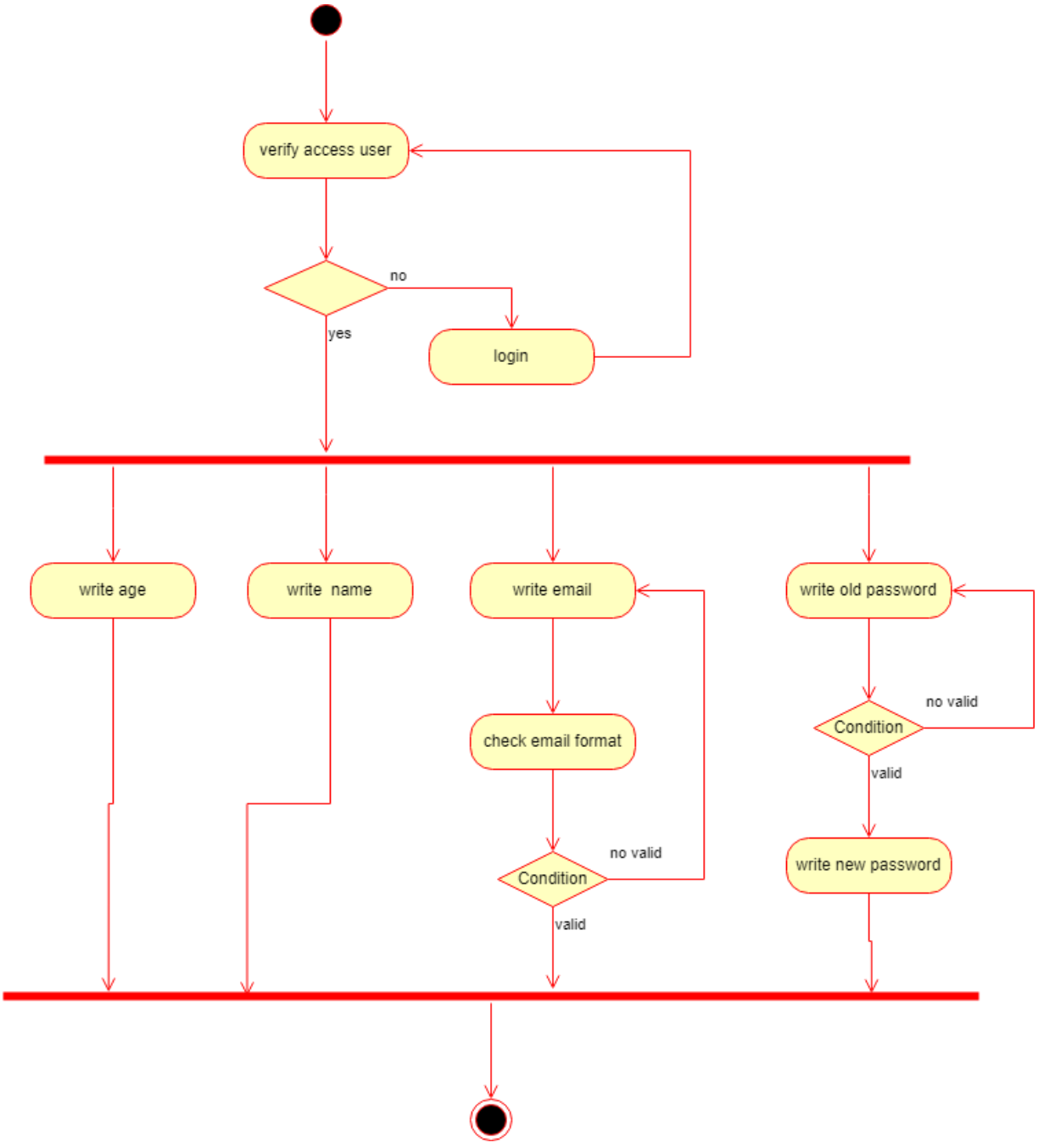
View Profile:



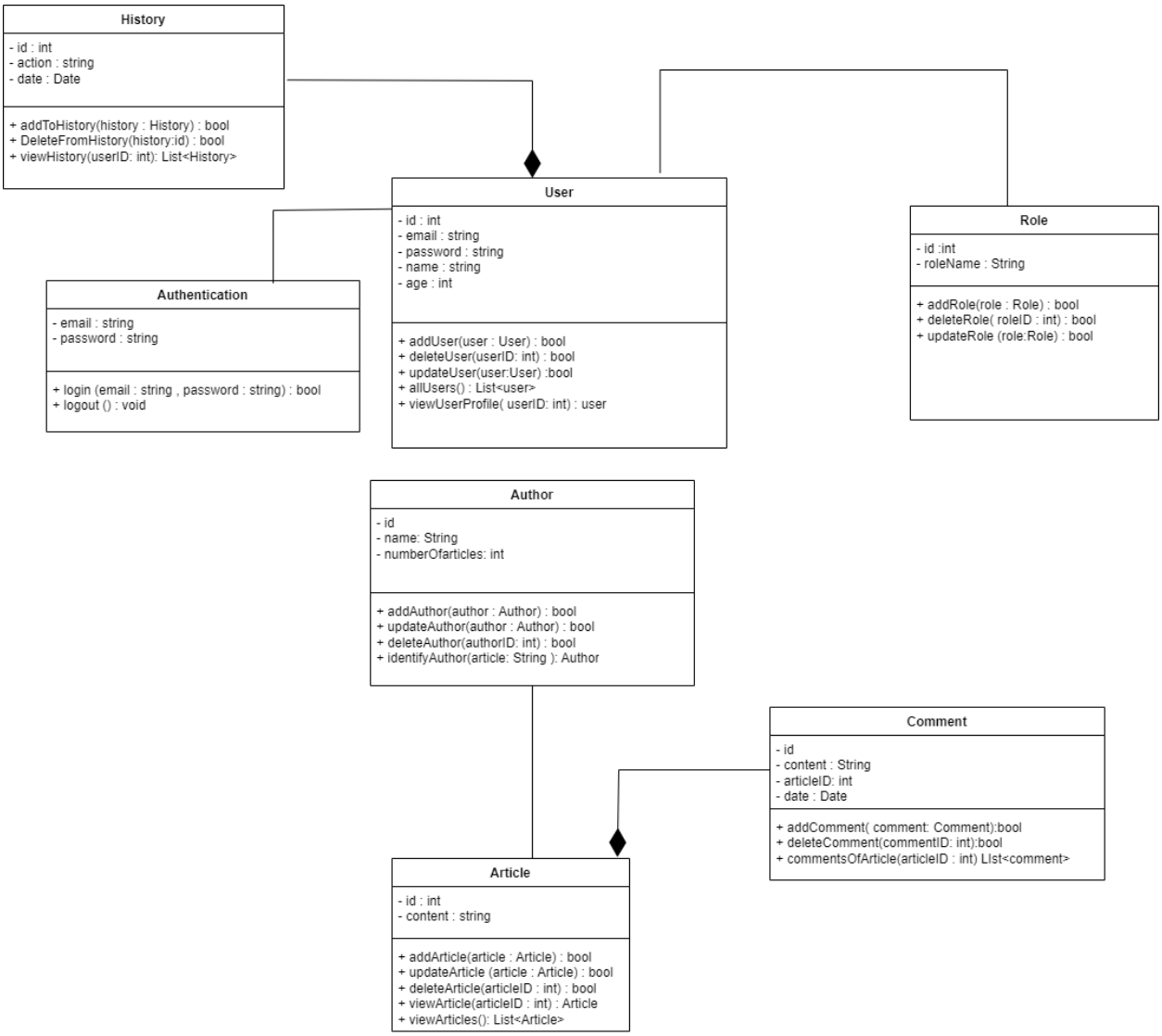
Log out:



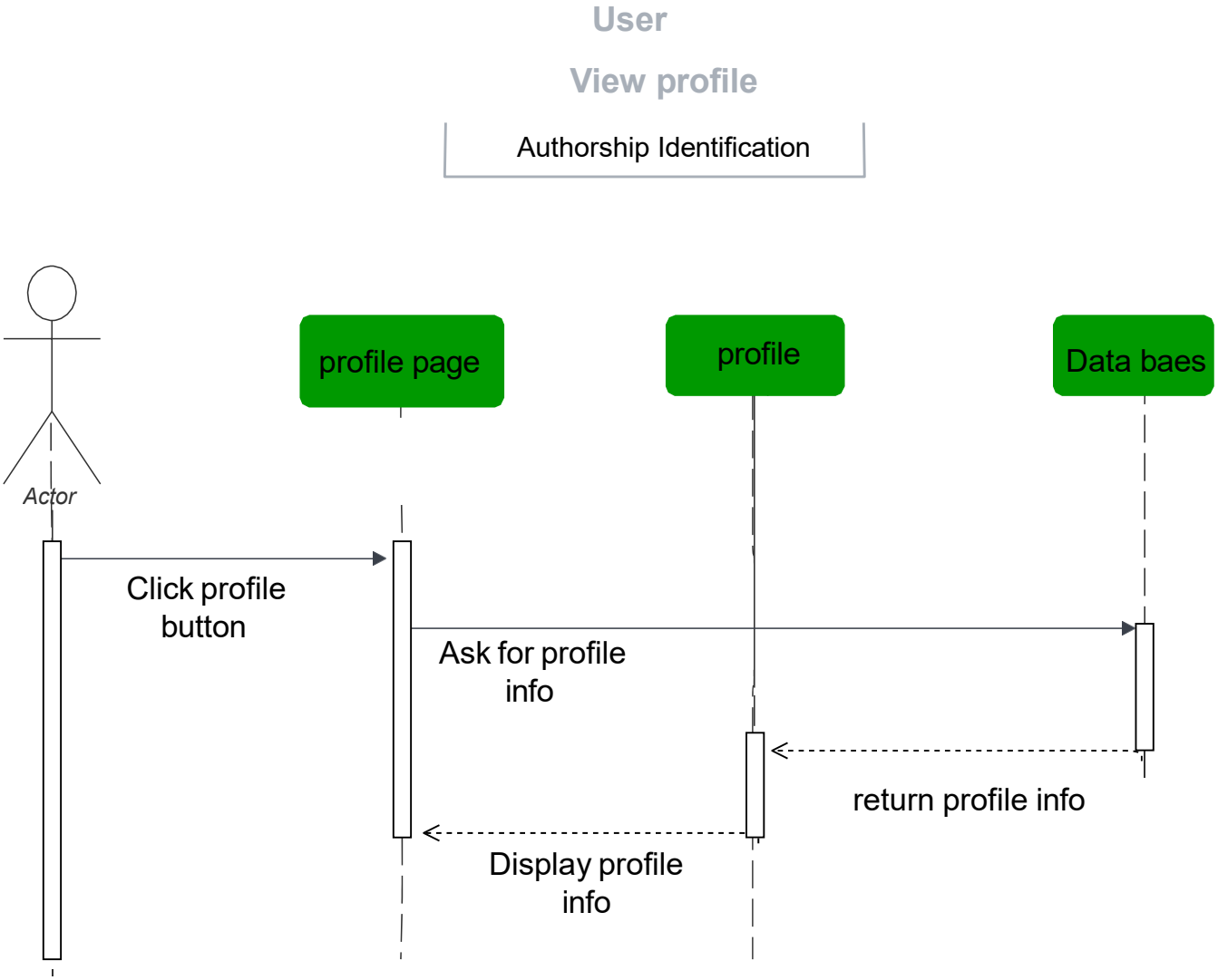
Update Profile:

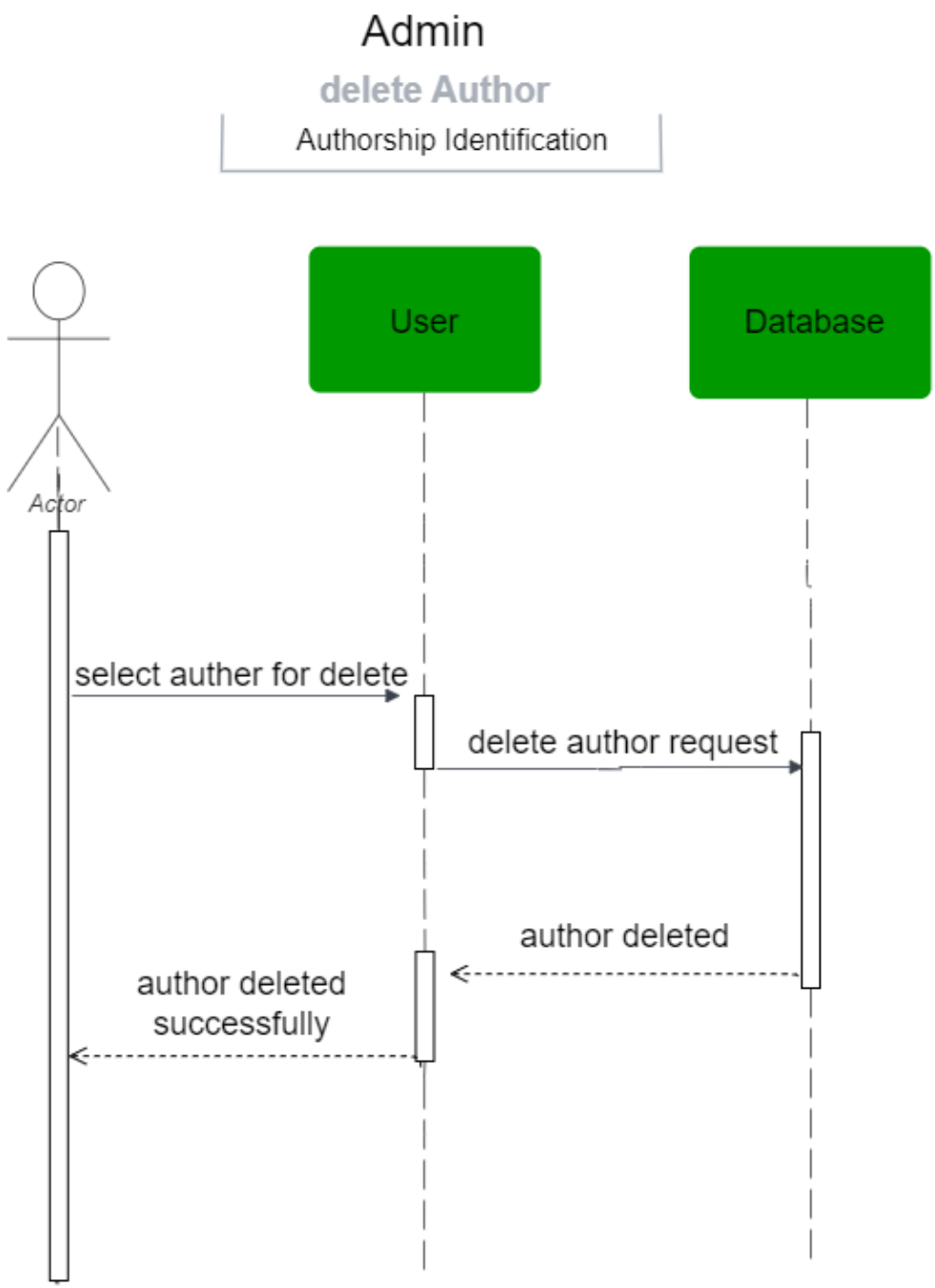


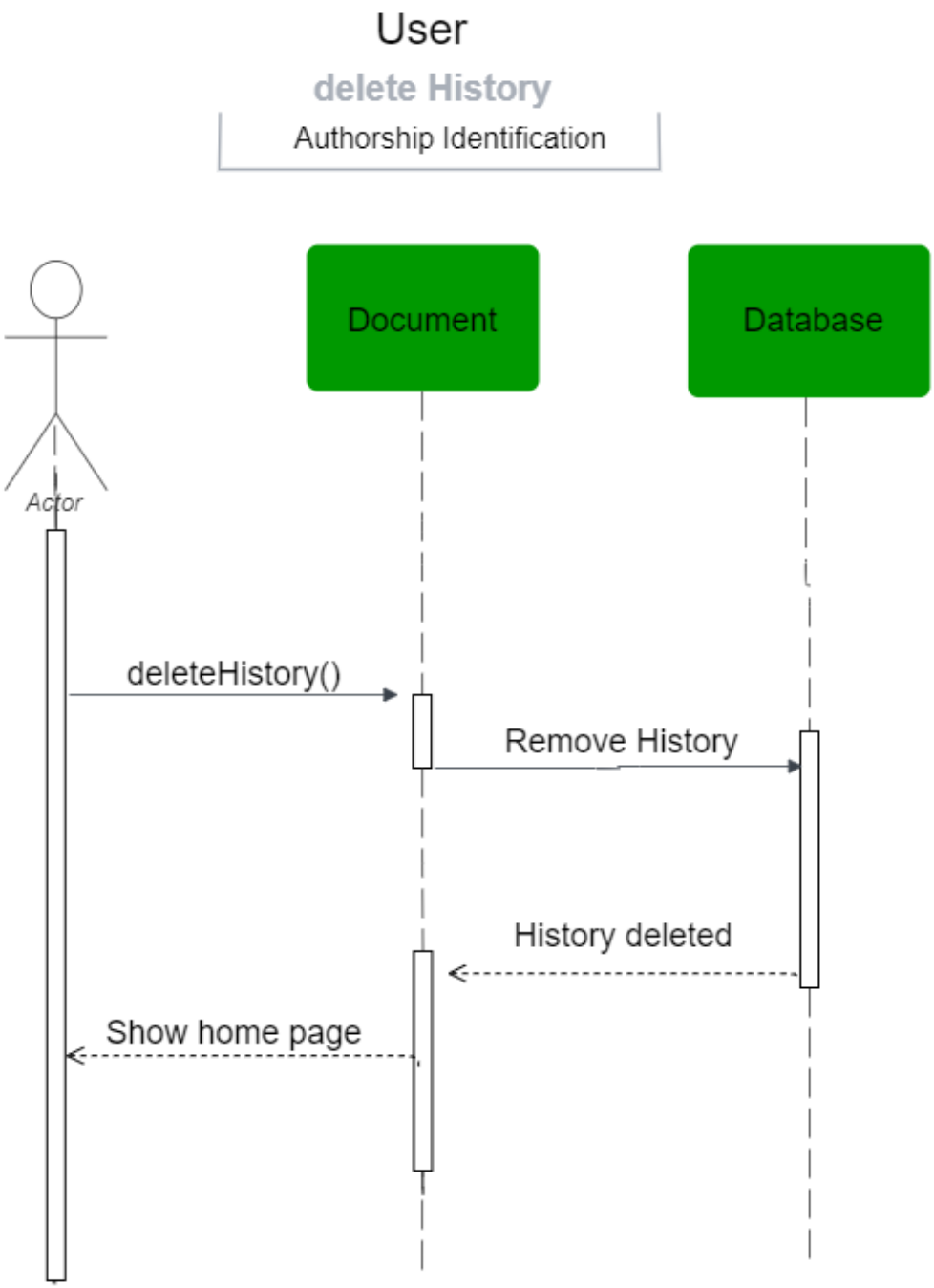
3.5.3Class Diagram

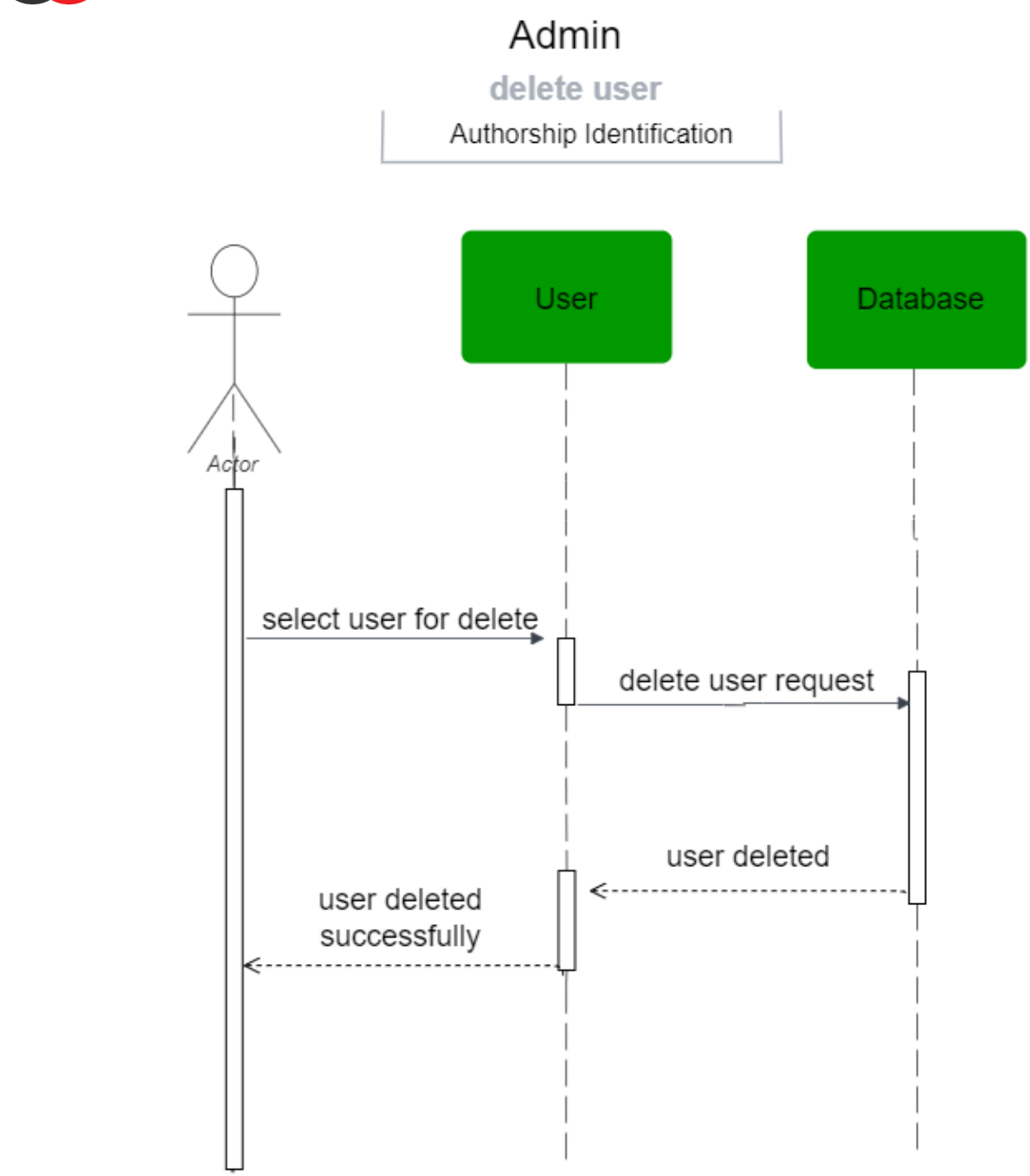


3.5.4Sequence Diagrams

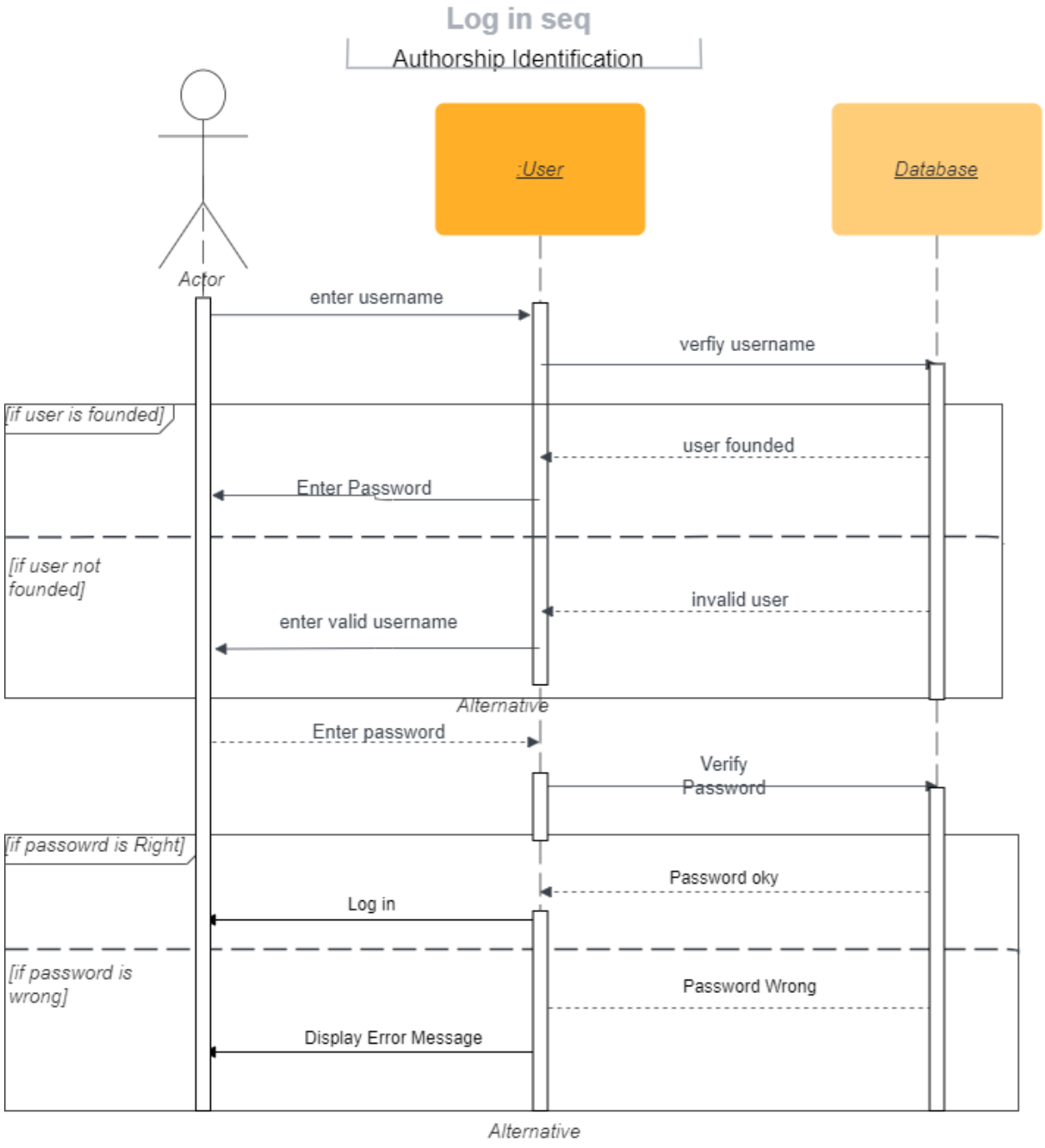


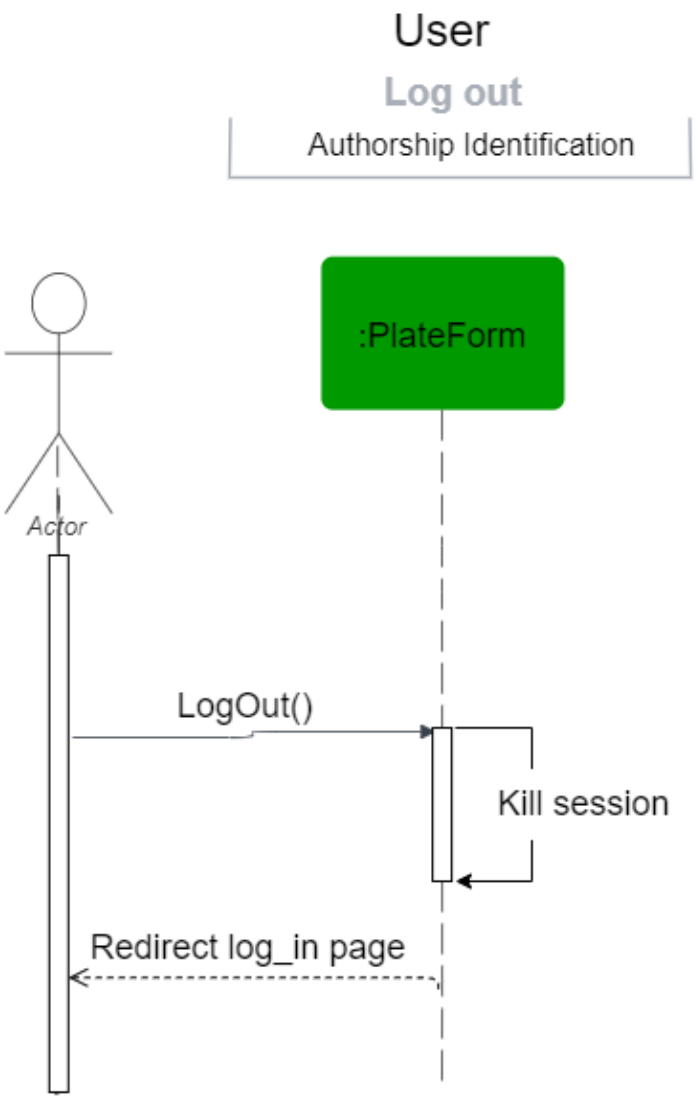

Admin
delete Author
Authorship Identification
User
Database
Actor
select auther for delete
delete author request
author deleted
author deleted
successfully

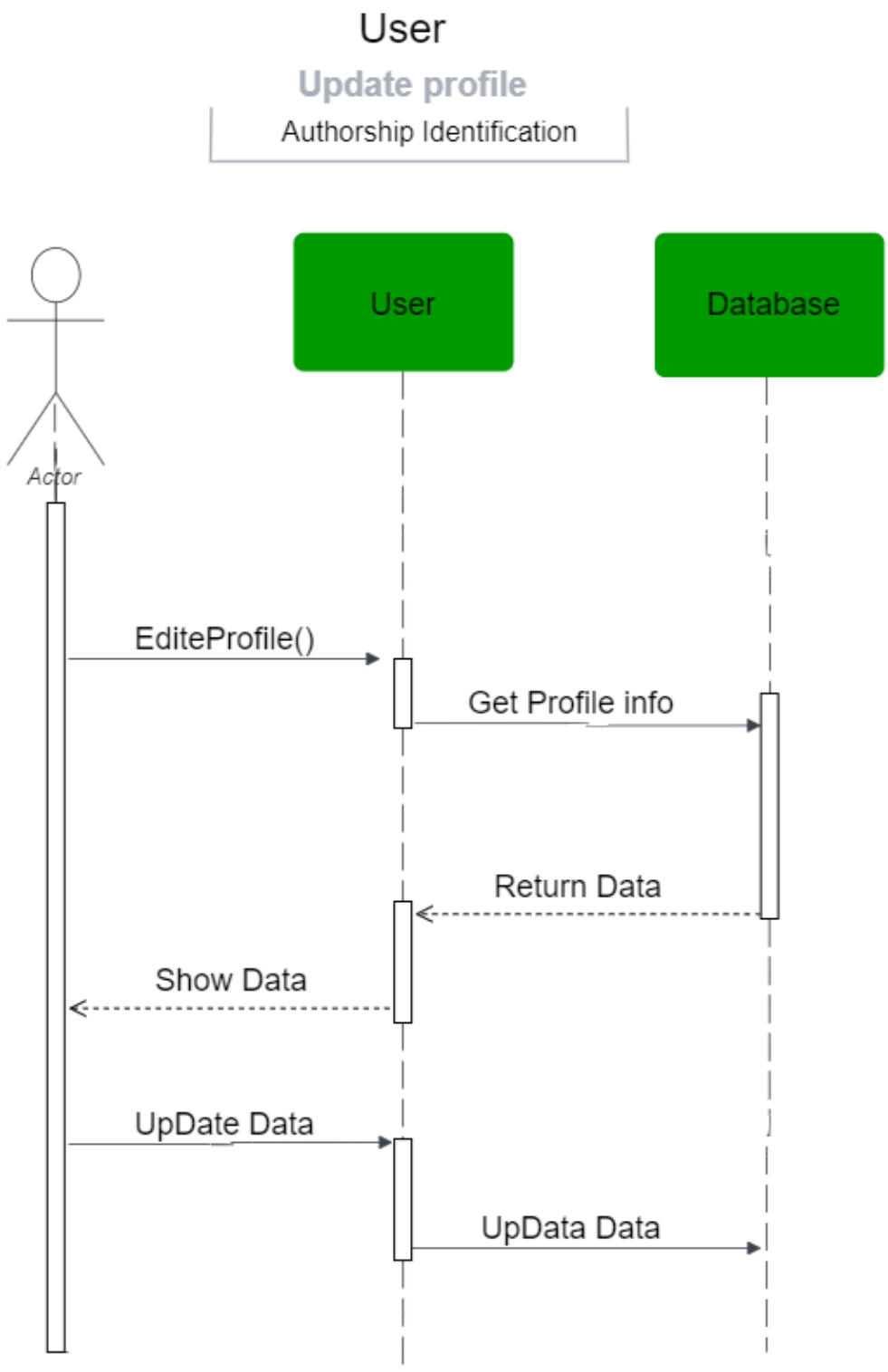

User
delete History
Authorship Identification
Document
Database
Actor
deleteHistory()
Remove History
History deleted
Show home page

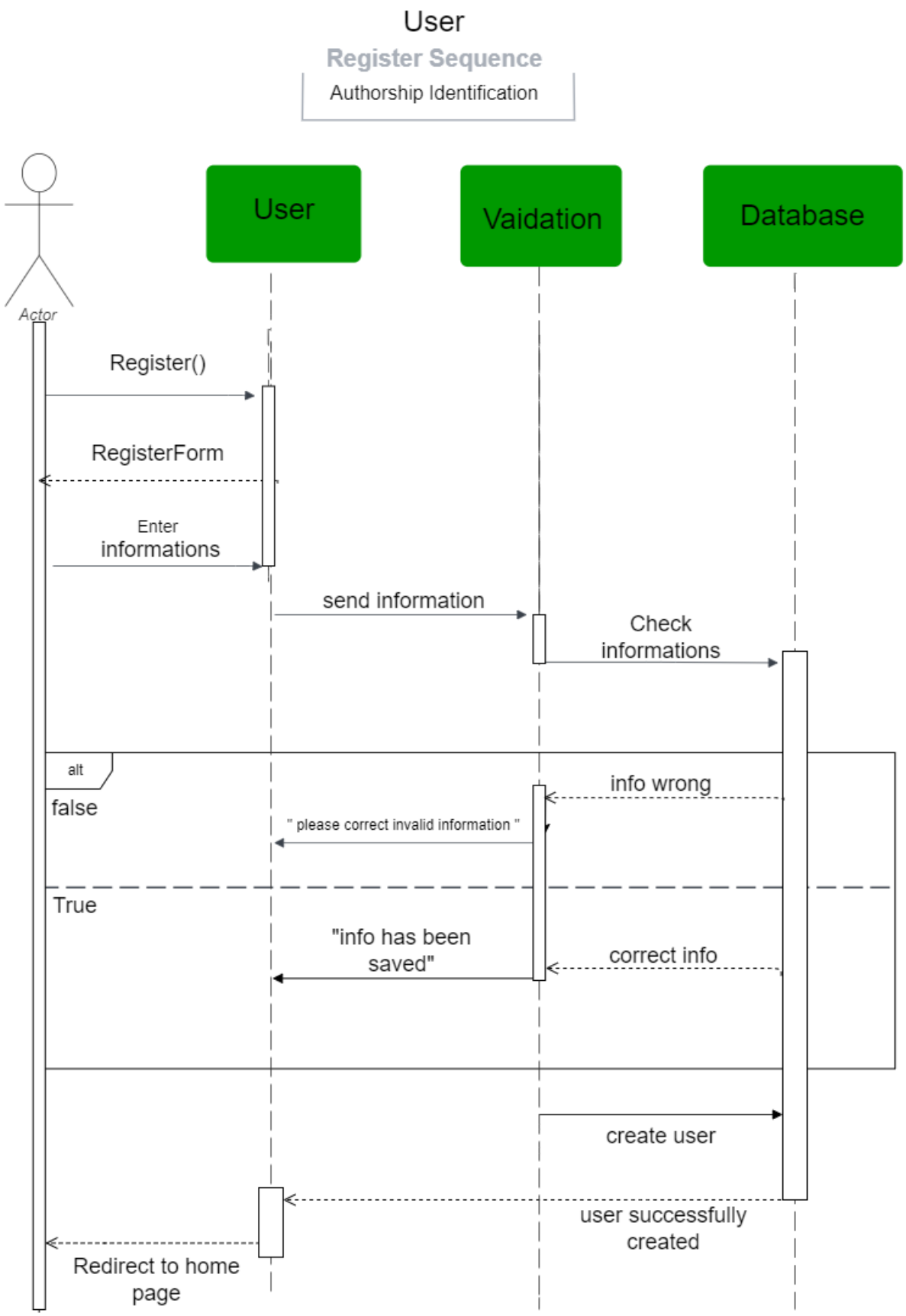

Admin
delete user
Authorship Identification
User
Database
Actor
select user for delete
delete user request
user deleted
user deleted
successfully

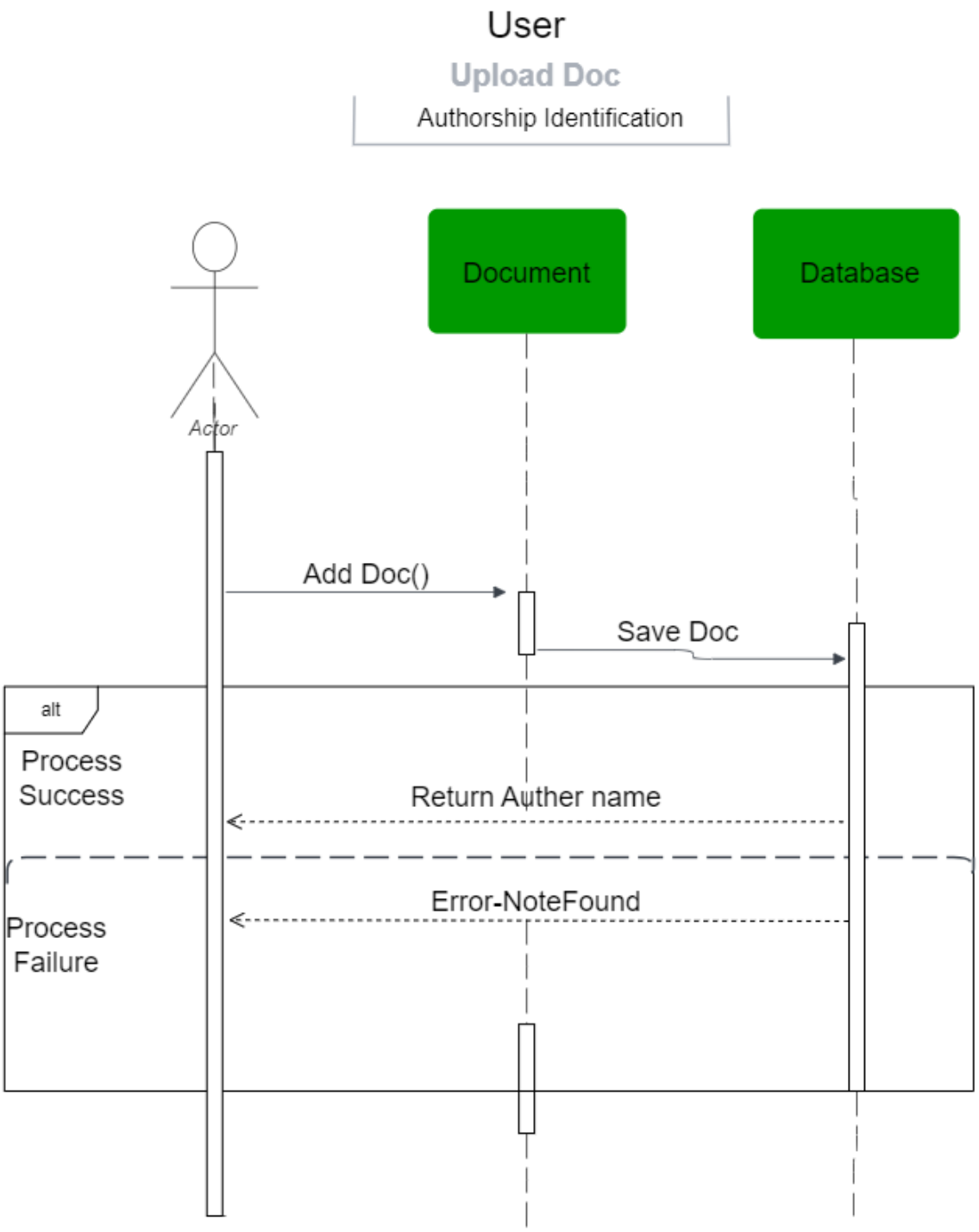
User

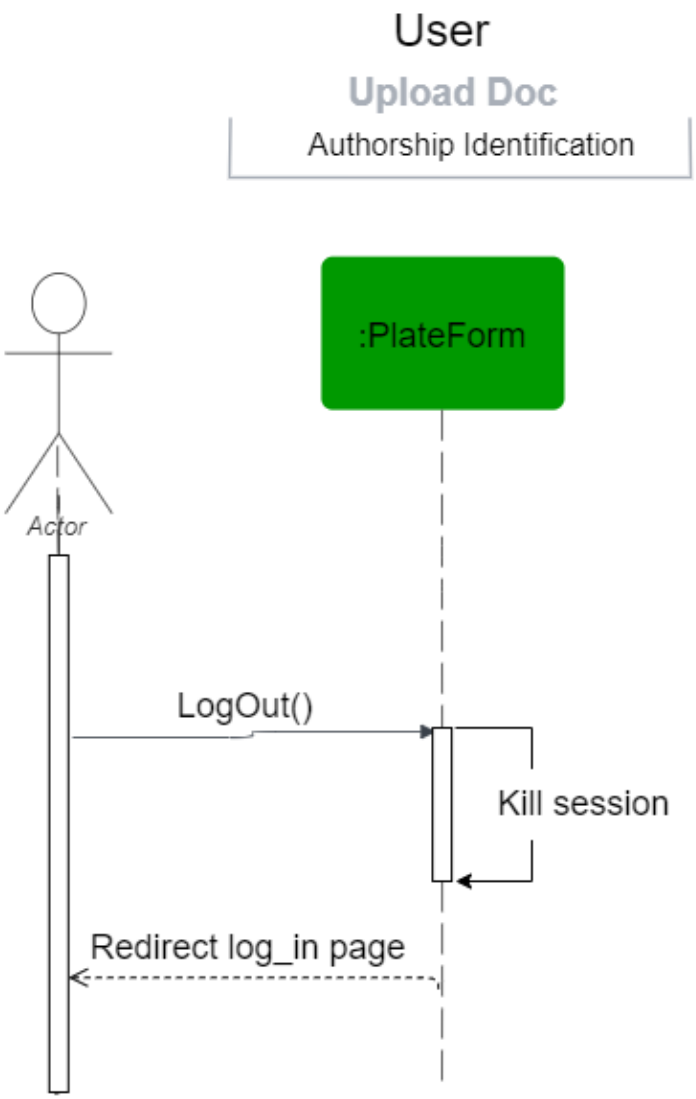

Log in seq
Authorship Identification
Actor
:User
Database
enter username
verfiy username
[if user is founded]
user founded
Enter Password
[if user not founded]
invalid user
enter valid username
Alternative
Enter password
Verify Password
[if passowrd is Right]
Password oky
Log in
[if password is wrong]
Password Wrong
Display Error Message
Alternative


User
Log out
Authorship Identification
:PlateForm
Actor
LogOut()
Kill session
Redirect log_in page


User
Update profile
Authorship Identification
User
Database
Actor
EditeProfile()
Get Profile info
Return Data
Show Data
UpDate Data
UpData Data


User
Register Sequence
Authorship Identification
Actor
User
Vaidation
Database
Register()
RegisterForm
Enter
informations
send information
Check
informations
alt
false
info wrong
" please correct invalid information "
True
"info has been
saved"
correct info
create user
user successfully
created
Redirect to home
page


User
Upload Doc
Authorship Identification
Document
Database
Actor
Add Doc()
Save Doc
alt
Process
Success
Return Auther name
Error-NoteFound
Process
Failure


User
Upload Doc
Authorship Identification
:PlateForm
Actor
LogOut()
Kill session
Redirect log_in page

Chapter 4

Implementation

In this chapter we are going to discuss and go deeper in Author Identification website Snapshots and present its code and machine learning model used to build it.

4.1 Model documentation

We developed an advanced author identification system that utilizes cutting-edge techniques in natural language processing and machine learning to accurately identify and attribute authorship of written texts. By leveraging state-of-the-art algorithms and data analysis, my project aims to revolutionize the field of author identification, enabling applications in forensic linguistics, plagiarism detection, and content verification.

In our groundbreaking graduation project focused on author identification, I developed and evaluated three distinct models to tackle this challenging task. Each model leveraged cutting-edge techniques and algorithms, showcasing their effectiveness in distinguishing authors based on textual patterns.

The first model employed a Long Short-Term Memory (LSTM) architecture, known for its ability to capture and process sequential information. By harnessing the power of LSTM, this model achieved a commendable accuracy of 63%. The LSTM model's capability to understand and learn from the sequential nature of texts proved valuable in identifying unique authorial traits and patterns, contributing to its success.

In addition to LSTM, the second model utilized the Support Vector Machine (SVM) algorithm, a well-established and powerful classification technique. The SVM algorithm demonstrated remarkable performance in author identification, achieving an impressive accuracy of 87%. By effectively defining decision boundaries and maximizing the margin between different author clusters, SVM showcased its ability to discern subtle nuances in writing styles and establish authorship with high precision.

Furthermore, the third model integrated the Bidirectional Encoder Representations from Transformers (BERT) model, a state-of-the-art pre-trained language model. By leveraging BERT's deep contextual understanding of text, this model achieved a noteworthy accuracy of 71%. BERT's ability to capture fine-grained linguistic features and contextual information contributed to its success in accurately attributing authorship.

The deployment of these three distinct models demonstrates the versatility and robustness of the approaches used in this project. By combining the strengths of LSTM, SVM, and BERT, I harnessed their unique capabilities to tackle the author identification challenge from different angles. This comprehensive approach allowed for a more holistic analysis of textual data, resulting in reliable and accurate authorship attribution.

The success of these models holds significant implications for various domains, including forensic linguistics, plagiarism detection, and content verification. The advancements made in author identification through this project pave the way for improved text analysis, enabling the identification and verification of authors with higher precision and efficiency.

Overall, this project showcases the potential of state-of-the-art models in author identification, pushing the boundaries of what is achievable in the field. The combination of LSTM, SVM, and BERT models opens up new avenues for research and practical applications, underscoring their importance in advancing the field of authorship attribution and its broader implications.

4.2Long Short-Term Memory(LSTM):

Let’s start with LSTM model:

```
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
trainAuthors = []
trainArticles = []

with open(main_folder_path, newline='', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        trainAuthors.append(row[0])
        trainArticles.append(row[1])

testAuthors = []
testArticles = []

with open(main_folder_path2, newline='', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile)

    for row in reader:
        testAuthors.append(row[0])
        testArticles.append(row[1])
vocab_size = 5000
embedding_dim = 64
max_length = 500
oov_tok = '<OOV>'
def cleanTheData(articles):
  cleanData=[]
  for article in articles:
    article = re.sub(r'http\S+', '', article)
    article = re.sub(r'[^a-zA-Z0-9\s]', '', article)
    tokens=word_tokenize(article.lower())
    cleaned_tokens = [token for token in tokens if token not in stop_words and token not in punctuation and token !="'"]
    stemmed_tokens = [stemmer.stem(token) for token in cleaned_tokens]

    cleanData.append(stemmed_tokens)
  return cleanData

cleanTrainArticles=cleanTheData(trainArticles)
cleanTestArticles=cleanTheData(testArticles)
cleanTrainArticles = [' '.join(tokens) for tokens in cleanTrainArticles]
cleanTestArticles = [' '.join(tokens) for tokens in cleanTestArticles]
tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(cleanTrainArticles+cleanTestArticles)
```

```
tokenizer.fit_on_texts(cleanTrainArticles+cleanTestArticles)
word_index=tokenizer.word_index
trainSeq=tokenizer.texts_to_sequences(cleanTrainArticles)
testSeq=tokenizer.texts_to_sequences(cleanTestArticles)
print(len(trainSeq[0]))
tfidf_vectorizer = TfidfVectorizer(max_features=vocab_size,ngram_range=(1,4))
# Fit the TF-IDF vectorizer on the training data
tfidf_vectorizer.fit(cleanTrainArticles)
train_tfidf =  tfidf_vectorizer.transform(cleanTrainArticles)
# Convert the test data to TF-IDF vectors
test_tfidf = tfidf_vectorizer.transform(cleanTestArticles)
authorsTokenization=Tokenizer()
allAuthors=[]
allAuthors.extend(trainAuthors)
allAuthors.extend(testAuthors)
authorsTokenization.fit_on_texts(allAuthors)
trainAuthorsSeq=np.array(authorsTokenization.texts_to_sequences(trainAuthors))
testAuthorsSeq=np.array(authorsTokenization.texts_to_sequences(testAuthors))
num_classes = len(authorsTokenization.word_index) + 1
trainAuthorsSeq = to_categorical(trainAuthorsSeq, num_classes=num_classes)
testAuthorsSeq = to_categorical(testAuthorsSeq, num_classes=num_classes)
trainPaded=pad_sequences(trainSeq,maxlen=max_length, truncating='post', padding='post')
testPaded=pad_sequences(testSeq,maxlen=max_length, truncating='post', padding='post')
# Combine the TF-IDF features with the padded sequences
trainTfidf = train_tfidf.toarray()
testTfidf = test_tfidf.toarray()
trainInput = np.concatenate([trainPaded, trainTfidf], axis=1)
testInput = np.concatenate([testPaded, testTfidf], axis=1)

model = Sequential()
model.add(Embedding(vocab_size, int(embedding_dim*2)))
model.add(Dropout(0.2)) # add dropout regularization to the embedding layer
model.add(Bidirectional(LSTM(int(embedding_dim*2), return_sequences=True )))
model.add(GlobalMaxPooling1D())
model.add(Dropout(0.5))

model.add(Dense(num_classes,activation='softmax'))
model.summary()
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)

# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=8, mode='min', restore_best_weights=True)

num_epochs = 40
history = model.fit(trainInput, trainAuthorsSeq, epochs=num_epochs, validation_data=(testInput, testAuthorsSeq),callbacks
    =[early_stopping,model_checkpoint] ,verbose=2)
```

This code is designed to train a deep learning model for author identification based on text articles. The model uses a combination of LSTM (Long Short-Term Memory) layers, TF-IDF (Term Frequency-Inverse Document Frequency) features, and tokenization techniques.

4.2.1Required Libraries:

The code requires the following libraries to be installed:

-os

-pandas

-numpy

-csv

-re

-google.colab (specifically for Google Colab environment)

-tensorflow

-nltk

-string

-sklearn

-matplotlib.pyplot

-You may need to install some of these libraries using pip if they are not already installed.

4.2.2Data Preparation:

- The code assumes that you have two CSV files containing training and test articles respectively. The paths to these files should be set in the variables main_folder_path and main_folder_path2 respectively.

- The code reads the CSV files and extracts the authors and articles into separate lists (trainAuthors, trainArticles, testAuthors, testArticles).

- The function cleanTheData() is defined to preprocess the articles. It performs the following steps:

  -Removes URLs from the articles.

  -Removes non-alphanumeric characters from the articles.

  -Tokenizes the articles into words.

  -Removes stopwords and punctuation from the tokens.

  -Stems the tokens using the Porter stemmer.

  -Adds the preprocessed articles to the cleanTrainArticles and cleanTestArticles lists.

- The cleanTrainArticles and cleanTestArticles lists are converted into sequences using the Tokenizer class from TensorFlow. The tokenizer is fit on the combined training and test articles, and the word index is obtained.

- The articles are converted into sequences using the fitted tokenizer (trainSeq, testSeq).

- The TF-IDF vectorizer is initialized and fit on the cleaned training articles. The training and test articles are then transformed into TF-IDF vectors (train_tfidf, test_tfidf).

- The authors are tokenized using the authorsTokenization tokenizer.

- The tokenizer is fit on all the authors and the training and test authors are converted into sequences (trainAuthorsSeq, testAuthorsSeq).

- The number of classes (authors) is determined and the sequences of training and test authors are one-hot encoded (trainAuthorsSeq, testAuthorsSeq).

- The training and test sequences are padded to a maximum length (max_length) using the pad_sequences() function from TensorFlow. The padded sequences are stored in trainPaded and testPaded.

- The TF-IDF features are converted to arrays (trainTfidf, testTfidf).

- The padded sequences and TF-IDF features are concatenated to create the final input for the model (trainInput, testInput).

4.2.3Model Architecture:

The model is defined using the Sequential class from TensorFlow. It consists of the following layers:

1-Embedding Layer:

Input: Vocabulary size (vocab_size) and embedding dimension (embedding_dim).

Dropout regularization: 0.2

2-Bidirectional LSTM Layer:

Number of units: embedding_dim*2

Return sequences: True

3-Global Max Pooling 1D Layer:

Dropout regularization: 0.5

4-Dense Layer:

Number of units: Number of classes (authors)

Activation function: Softmax

The model is compiled with categorical cross-entropy loss and the Adam optimizer.

The accuracy metric is also calculated.

4.2.4Model Training

The model checkpoints are defined to save the best model based on the validation loss during training. The checkpoints are saved in the file "best_model.h5".

Early stopping is implemented with a patience of 8 epochs and the best weights are restored when training is stopped.

The model is trained for a specified number of epochs (num_epochs) using the training input and one-hot encoded training authors. The validation data is provided as the test input and one-hot encoded test authors.

Output: -

The model training history is stored in the history variable, which can be used to analyze the training and validation performance.

4.3Bidirectional Encoder Representations from Transformers (BERT):

Second one is BERT:

```
main_folder_path = "/content/drive/MyDrive/gpProject/trainArticles.csv"
main_folder_path2 = "/content/drive/MyDrive/gpProject/testArticles.csv"
#!pip3 install ktrain
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
trainAuthors = []
trainArticles = []
with open(main_folder_path, newline='', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        trainAuthors.append(row[0])
        trainArticles.append(row[1])

testAuthors = []
testArticles = []
with open(main_folder_path2, newline='', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile)

    for row in reader:
        testAuthors.append(row[0])
        testArticles.append(row[1])

import ktrain
from ktrain import text
(x_train, y_train), (x_test, y_test), preproc = text.texts_from_array(x_train=trainArticles, y_train=trainAuthors, x_test
    =testArticles, y_test=testAuthors, maxlen=512, preprocess_mode='bert')


model = text.text_classifier(name='bert',
                             train_data=(x_train, y_train),
                             preproc=preproc)


learner = ktrain.get_learner(model=model,
                             train_data=(x_train, y_train),
                             val_data=(x_test, y_test),
                             batch_size=8)

learner.fit_onecycle(lr=2e-5,
                     epochs=7)
```

This code is designed to train a deep learning model for author identification based on text articles using the BERT (Bidirectional Encoder Representations from Transformers) model. The code utilizes the ktrain library, which provides a high-level interface for TensorFlow and Keras.

4.3.1Required Libraries:

The code requires the following libraries to be installed:

-os

-csv

-nltk.stem (PorterStemmer)

-nltk.stem (WordNetLemmatizer)

-ktrain

-ktrain.text

-You may need to install some of these libraries using pip if they are not already installed.

4.3.2Data Preparation:

- The code assumes that you have two CSV files containing training and test articles, respectively. The paths to these files should be set in the variables main_folder_path and main_folder_path2, respectively.

- The training and test articles are read from the CSV files and stored in the trainArticles and testArticles lists, respectively.

- The corresponding authors are also read and stored in the trainAuthors and testAuthors lists, respectively.

- Stemming and lemmatization objects (stemmer and lemmatizer) are created using the NLTK library.

- These objects can be used to perform stemming and lemmatization on the text data if needed.

4.3.3Model Training:

The texts_from_array function from ktrain.text is used to preprocess the training and test data. It takes the training and test articles as input along with their corresponding authors. The maximum length of the input sequences is set to 512. The function preprocesses the text data and returns the preprocessed data along with a preprocessor object (preproc).

The text_classifier function from ktrain.text is used to define the BERT model for text classification. It takes the name of the model ('bert'), the preprocessed training data, and the preprocessor object as input. The function initializes the BERT model for text classification.

The get_learner function from ktrain is used to create a learner object for training the model. It takes the initialized BERT model, the preprocessed training data, and the preprocessed validation data as input. The batch size for training is set to 8.

The fit_onecycle method of the learner object is called to train the model using the one-cycle learning rate policy. The learning rate is set to 2e-5, and the model is trained for 7 epochs.

Output:

The model is trained using the provided training and test data, and the training progress is displayed. The final trained model can be used for author identification on new text articles.

4.4Support Vector Machine (SVM):

Third one is SVM:

```
main_folder_path = "/content/drive/MyDrive/gpProject/trainArticles.csv"
main_folder_path2 = "/content/drive/MyDrive/gpProject/testArticles.csv"
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
train_authors = []
train_articles = []
with open(main_folder_path, newline='', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        train_authors.append(row[0])
        train_articles.append(row[1])

test_authors = []
test_articles = []
with open(main_folder_path2, newline='', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile)

    for row in reader:
        test_authors.append(row[0])
        test_articles.append(row[1])
def cleanTheData(articles):
  cleanData=[]
  for article in articles:
    article = re.sub(r'http\S+', '', article)
    article = re.sub(r'[^a-zA-Z0-9\s]', '', article)
    tokens=word_tokenize(article.lower())
    cleaned_tokens = [token for token in tokens if token not in stop_words and token not in punctuation and token !="'"]
    stemmed_tokens = [stemmer.stem(token) for token in cleaned_tokens]

    cleanData.append(stemmed_tokens)
  return cleanData

cleanTrainArticles=cleanTheData(train_articles)
cleanTestArticles=cleanTheData(test_articles)
cleanTrainArticles = [' '.join(tokens) for tokens in cleanTrainArticles]
cleanTestArticles = [' '.join(tokens) for tokens in cleanTestArticles]
cleanTrainArticles.extend(cleanTestArticles)
train_authors.extend(test_authors)

X = cleanTrainArticles
y = train_authors
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
print(" new class ")
print("step 4")
text_clf = make_pipeline(
        TfidfVectorizer(),
        #TruncatedSVD(n_components=6000,random_state =None),
        LinearSVC()

    )
print("step 5")
print("####################")
# print(data)
print("step 6")
text_clf.fit(X_train, y_train)
print("step 7")
predictions = text_clf.predict(X_test)
print("Accuracy = {} %".format(metrics.accuracy_score(y_test, predictions) * 100))
```

This code performs author identification based on text articles using a Linear Support Vector Classifier (LinearSVC) and TF-IDF (Term Frequency-Inverse Document Frequency) vectorization. The code follows the following steps:

4.4.1Data Preparation:

- The code assumes that you have two CSV files containing training and test articles, respectively. The paths to these files should be set in the variables 'main_folder_path' and main_folder_path2, respectively.

- The training and test articles are read from the CSV files and stored in the train_articles and test_articles lists, respectively. The corresponding authors are also read and stored in the train_authors and test_authors lists, respectively.

- Text Cleaning

The code defines a function named cleanTheData that cleans the text data by removing URLs, non-alphanumeric characters, stopwords, and punctuation. It uses the NLTK library for tokenization, stopword removal, stemming, and lowercase conversion. The function takes a list of articles as input and returns a list of cleaned tokens.

- The cleanTheData function is applied to the training and test articles to

obtain the cleaned versions. The cleaned articles are then concatenated, and the corresponding authors are also extended.

4.4.2Model Training and Evaluation:

The make_pipeline function from the sklearn.pipeline module is used to create a pipeline for text classification. The pipeline consists of a TF-IDF vectorizer and a LinearSVC classifier.

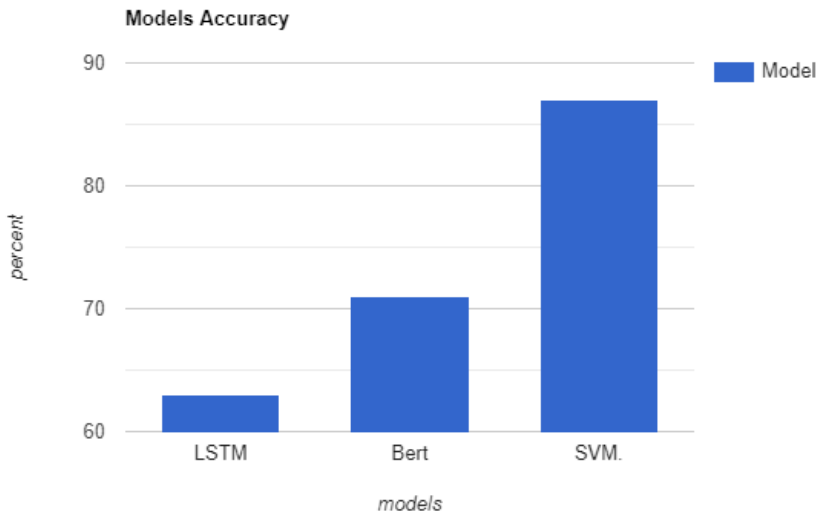
The pipeline is fit to the training data using the fit method, where the articles are represented as TF-IDF vectors, and the authors are the target labels.

The pipeline predicts the authors for the test data using the predict method.

The accuracy of the predictions is calculated using the accuracy_score function from the sklearn.metrics module.

Output:

The code outputs the accuracy of the author identification model, which is calculated based on the predictions made on the test data.



4.5 Software documentation:

4.5.1-Authentication :

-login() method

Description

The `login()` method is used to authenticate the user based on the provided email and password. It checks if the email and password match with the values stored in the database and returns the user object if the authentication is successful. Additionally, it sets the `userName` and `userId` attributes in the `HttpSession` object for later use.

Parameters

- `loginModel` - an object of the `LoginModel` class containing the email and password of the user.

- `httpSession` - an object of the `HttpSession` class to store the user's session data.

Return Value

- Returns a `User` object representing the authenticated user if the authentication is successful.

- Returns `null` if the email and password do not match any user in the database.

```
1 usage
@Override
public User login(LoginModel loginModel, HttpSession httpSession) {
    Optional<List<User>> user =userRepository.login(loginModel.getEmail(),loginModel.getPassword());
    if(!user.isPresent()||user.get().size()==0)return null;
    httpSession.setAttribute( s: "userName",user.get().get(0).getName());
    httpSession.setAttribute( s: "userId",user.get().get(0).getId());
    return  user.get().get(0);
}
```

-logout() method

Description

The `logout()` method is used to invalidate the current user's session by invalidating the `HttpSession` object.

Parameters

- `httpSession` - an object of the `HttpSession` class representing the user's current session.

Return Value

- The method does not return any value.

4.5.2-User:

-addUser() method

Description

The `addUser()` method is used to add a new user to the system. It checks if the user already exists in the system by checking the email field. If the user does not exist, it adds the user to the database.

Parameters

- `user` - an object of the `User` class representing the user to be added.

Return Value

- Returns `true` if the user is added successfully.

- Returns `false` if the user already exists in the database.

```
@Override
public boolean addUser(User user) {
    if(findUserByEmail(user.getEmail())!=null)return false;
    userRepository.save(user);
    return  true;
}
```

-deleteUser() method

Description

The `deleteUser()` method is used to delete a user from the system. It finds the user by ID and deletes the user from the database.

Parameters

- `userId` - an integer representing the ID of the user to be deleted.

Return Value

- Returns `true` if the user is deleted successfully.

- Returns `false` if the user does not exist in the database.

```java
@Override
public boolean deleteUser(int userId) {
    User user= findUserById(userId);
    if(user==null)return  false;
    userRepository.delete(user);
    return  true;
}
```

-updateUser() method

Description

The `updateUser()` method is used to update the information of an existing user in the system. It finds the user by ID and updates the user's information.

Parameters

- `user` - an object of the `User` class representing the updated user information.

Return Value

- Returns `true` if the user is updated successfully.

- Returns `false` if the user does not exist in the database or the updated email already exists in the system.

```java
@Override
public boolean updateUser(User user) {
    User currentUser=findUserById(user.getId());
    if(currentUser==null)return false;
    if(!currentUser.getEmail().equals(user.getEmail())){
        if(findUserByEmail(user.getEmail())!=null)return false;
    }
    currentUser.setAge(user.getAge());
    currentUser.setName(user.getName());
    currentUser.setEmail(user.getEmail());
    currentUser.setPassword(user.getPassword());
    userRepository.save(currentUser);
    return  true;
}
```

-findUserById() method

Description

The `findUserById()` method is used to find a user in the system based on their ID.

Parameters

- `id` - an integer representing the ID of the user to be found.

Return Value

- Returns a `User` object representing the found user if the user exists in the database.

- Returns `null` if the user does not exist in the database.

```
@Override
public User findUserById(int id) {
    Optional<User>user=userRepository.findById(id);
    if(user.isPresent())
        return user.get() ;
    return  null;
}
```

-getAllUsers() method

Description

The `getAllUsers()` method is used to retrieve a list of all users in the system.

Parameters

- The method does not take any parameters.

Return Value

- Returns a `List` of `User` objects representing all users in the system.

```
}

1 usage   Mostafa983469
@Override
public List<User> getAllUsers() { return  userRepository.findAll(); }
```

-findUserByEmail() method

Description

The `findUserByEmail()` method is used to find a user in the system based on their email.

Parameters

- `email` - a string representing the email of the user to be found.

Return Value

- Returns a `User` object representing the found user if the user exists in the database.

- Returns `null` if the user does not exist in the database.

```
@Override
public User findUserByEmail(String email) {

    Optional<List<User>> user=userRepository.findByEmail(email);
    if(user.isPresent()&&user.get().size()>0){
       return user.get().get(0);
    }
    return null;
}
```

-addRoleToUser() method

Description

The `addRoleToUser()` method is used to add a role to a user. It finds the user and role by their IDs and adds the role to the user's list of roles.

Parameters

- `userId` - an integer representing the ID of the user to whom the role will be added.

- `roleId` - an integer representing the ID of the role to be added.

Return Value

- Returns `true` if the role is added to the user successfully.

- Returns `false` if the user or role does not exist in the database.

```
1 usage  ± Mostafa983469
@Override
public boolean addRoleToUser(int userId, int roleId) {
    User user =findUserById(userId);
    Role role=roleService.findRoleById(roleId);
    if(user==null||role==null)return  false;
    user.getRoles().add(role);
    userRepository.save(user);
    return  true;
}
```

4.5.3role:

-addRole() method

Description

The `addRole()` method is used to add a new role to the system. It checks if the role already exists in the system by checking the name field. If the role does not exist, it adds the role to the database.

Parameters

- `role` - an object of the `Role` class representing the role to be added.

Return Value

- Returns `true` if the role is added successfully.

- Returns `false` if the role already exists in the database.

Example Usage: java

Role newRole = new Role("admin");

roleService.addRole(newRole);

```java
@Override
public boolean addRole(Role role) {

    if(findRoleByName(role.getName())!=null)return false;
    roleRepository.save(role);
    return true;
}
```

-updateRole() method

Description

The `updateRole()` method is used to update the information of an existing role in the system. It finds the role by ID and updates the role's information.

Parameters

- `role` - an object of the `Role` class representing the updated role information.

Return Value

- Returns `true` if the role is updated successfully.

- Returns `false` if the role does not exist in the database or the updated name already exists in the system.

Example Usage: java

Role updatedRole = new Role(1, "administrator");

roleService.updateRole(updatedRole);

```java
@Override
public boolean updateRole(Role role) {
    Role currentRole=findRoleById(role.getId());
    if(currentRole==null)return false;
    if(!currentRole.getName().equals(role.getName())){
        if(findRoleByName(role.getName())!=null)return false;
    }
    currentRole.setName(role.getName());
    roleRepository.save(currentRole);
    return  true;
```

-deleteRole() method

Description

The `deleteRole()` method is used to delete a role from the system. It finds the role by ID and deletes the role from the database.

Parameters

- `roleId` - an integer representing the ID of the role to be deleted.

Return Value

- Returns `true` if the role is deleted successfully.

- Returns `false` if the role does not exist in the database.

Example Usage: java

int roleId = 1;

roleService.deleteRole(roleId);

```java
@Override
public boolean deleteRole(int roleId) {
    Role role=findRoleById(roleId);
    if(role==null)return  false;
    roleRepository.delete(role);
    return  true;
}
```

-findRoleById() method

Description

The `findRoleById()` method is used to find a role in the system based on its ID.

Parameters

- `id` - an integer representing the ID of the role to be found.

Return Value

- Returns a `Role` object representing the found role if the role exists in the database.

- Returns `null` if the role does not exist in the database.

Example Usage: java

int roleId = 1;

Role foundRole = roleService.findRoleById(roleId);

```java
@Override
public Role findRoleById(int id) {
    Optional<Role> role=roleRepository.findById(id);
    if(role.isPresent())return  role.get();
    return  null;
}
```

-findRoleByName() method

Description

The `findRoleByName()` method is used to find a role in the system based on its name.

Parameters

- `name` - a string representing the name of the role to be found.

Return Value

- Returns a `Role` object representing the found role if the role exists in the database.

- Returns `null` if the role does not exist in the database.

Example Usage: java

String roleName = "admin";

Role foundRole = roleService.findRoleByName(roleName);

```java
@Override
public Role findRoleByName(String name) {
    Optional<List<Role>>role =roleRepository.findByName(name);
    if(role.isPresent()&&role.get().size()>0){
        return role.get().get(0);
    }
    return  null;
}
```

-getAllRoles() method

Description

The `getAllRoles()` method is used to retrieve a list of all roles in the system.

Parameters

- The method does not take any parameters.

Return Value

- Returns a `List` of `Role` objects representing all roles in the system.

Example Usage: java

List<Role> allRoles = roleService.getAllRoles();

4.5.4- history:

-addToHistory() method

Description

The `addToHistory()` method is used to add an entry to the history table in the database. It creates a new `History` object with the current date, user ID, and action, and saves it to the database.

Parameters

- `action` - a string representing the action to be added to the history.

Return Value

- The method does not return anything.

Example Usage:java

String action = "User logged in";

historyService.addToHistory(action);

-deleteFromHistory() method

Description

The `deleteFromHistory()` method is used to delete an entry from the history table in the database. It finds the history entry by ID and deletes it from the database.

Parameters

- `historyId` - an integer representing the ID of the history entry to be deleted.

Return Value

- Returns `true` if the history entry is deleted successfully.

- Returns `false` if the history entry does not exist in the database.

Example Usage: java

int historyId = 1;

historyService.deleteFromHistory(historyId);

```java
@Override
public boolean deleteFromHistory(int historyId) {
    History history=findHistory(historyId);
    if(history==null)return false;
    historyRepository.delete(history);
    return true;
}
```

-viewHistory() method

Description

The `viewHistory()` method is used to retrieve a list of all history entries for a particular user.

Parameters

- `userId` - an integer representing the ID of the user whose history is to be viewed.

Return Value

- Returns a `List` of `History` objects representing all history entries for the specified user.

Example Usage: java

int userId = 1;

List<History> userHistory = historyService.viewHistory(userId);

-findHistory() method

Description

The `findHistory()` method is used to find a history entry in the database based on its ID.

Parameters

- `historyId` - an integer representing the ID of the history entry to be found.

Return Value

- Returns a `History` object representing the found history entry if it exists in the database.

- Returns `null` if the history entry does not exist in the database.

Example Usage: java

int historyId = 1;

History foundHistory = historyService.findHistory(historyId);

```
1 usage  Mostafa983469
@Override
public List<History> viewHistory(int userId) { return historyRepository.viewHistory(userId); }
1 usage  Mostafa983469 *
@Override
public History findHistory(int historyId) {
    Optional<History> history=historyRepository.findById(historyId);
    if(history.isPresent())return history.get();
    return  null;
}
```


4.5.5- author:

-addAuthor() method

Description

The `addAuthor()` method is used to add a new author to the database. It saves the `Author` object to the database.

Parameters

- `author` - an object of the `Author` class representing the author to be added.

Return Value

- Returns `true` if the author is added successfully.

Example Usage: java

Author newAuthor = new Author("John Doe", 10);

authorService.addAuthor(newAuthor);

```
@Override
public boolean addAuthor(Author author) {
    authorRepository.save(author);
    return true;
}
```

-updateAuthor() method

Description

The `updateAuthor()` method is used to update an existing author's information in the database. It finds the author by ID and updates the author's name and number of articles.

Parameters

- `author` - an object of the `Author` class representing the updated author information.

Return Value

- Returns `true` if the author is updated successfully.

- Returns `false` if the author does not exist in the database.

Example Usage: java

Author updatedAuthor = new Author(1, "Jane Doe", 15);

authorService.updateAuthor(updatedAuthor);

```
2 usages
@Override
public boolean updateAuthor(Author author) {
    Author currentAuthor=findAuthor(author.getId());
    if(currentAuthor==null)return  false;
    currentAuthor.setName(author.getName());
    currentAuthor.setNumberOfArticles(author.getNumberOfArticles());
    authorRepository.save(currentAuthor);
    return true;
}
```

-deleteAuthor() method

Description

The `deleteAuthor()` method is used to delete an author from the database. It finds the author by ID and deletes the author from the database.

Parameters

- `authorId` - an integer representing the ID of the author to be deleted.

Return Value

- Returns `true` if the author is deleted successfully.

- Returns `false` if the author does not exist in the database.

Example Usage: java

int authorId = 1;

authorService.deleteAuthor(authorId);

```
}
1 usage
@Override
public boolean deleteAuthor(int authorId) {
    Author currentAuthor=findAuthor(authorId);
    if(currentAuthor==null)return  false;
    authorRepository.delete(currentAuthor);
    return  true;
}
```

-findAuthor() method

Description

The `findAuthor()` method is used to find an author in the database based on their ID.

Parameters

- `authorId` - an integer representing the ID of the author to be found.

Return Value

- Returns an `Author` object representing the found author if the author exists in the database.

- Returns `null` if the author does not exist in the database.

Example Usage: java

int authorId = 1;

Author foundAuthor = authorService.findAuthor(authorId);

```java
@Override
public Author findAuthor(int authorId) {
    Optional<Author> author=authorRepository.findById(authorId);
    if(author.isPresent())return  author.get();
    return  null;
}
```

-viewAllAuthors() method

Description

The `viewAllAuthors()` method is used to retrieve a list of all authors in the database.

Parameters

- The method does not take any parameters.

Return Value

- Returns a `List` of `Author` objects representing all authors in the database.

Example Usage: java

List<Author> allAuthors = authorService.viewAllAuthors();

```java
}

1 usage
@Override
public List<Author> viewAllAuthors() { return authorRepository.findAll(); }

1 usage
```

-identifyAuthor() method

Description

The `identifyAuthor()` method is used to identify the author of an article using an external API. It sends a GET request to the API with the article text and receives a `ResponseEntity` object containing the predicted author.

Parameters

- `article` - a string representing the text of the article to be analyzed.

Return Value

- Returns a `ResponseEntity` object representing the response from the external API.

Example Usage: java

String article = "Lorem ipsum dolor sit amet...";

ResponseEntity<String> identifiedAuthor = authorService.identifyAuthor(article);

-findAuthorByName() method

Description

The `findAuthorByName()` method is used to find an author in the database based on their name.

Parameters

- `name` - a string representing the name of the author to be found.

Return Value

- Returns an `Author` object representing the found author if the author exists in the database.

- Returns `null` if the author does not exist in the database.

Example Usage: java

String authorName = "John Doe";

Author foundAuthor = authorService.findAuthorByName(authorName);

-Read PDF Method

Description:

The `readPdf()` method reads the contents of a PDF file located at the specified file path and returns the text contents of the PDF as a string.

Parameters:

- `filePath` (String): The file path of the PDF file to be read.

Returns:

- `text` (String): The text contents of the PDF file.

Exceptions:

- `IOException`: If there is an error reading the PDF file.

Dependencies:

This method requires the Apache PDFBox library to be installed and imported.

Usage example: java

String filePath = "/path/to/my/pdf/file.pdf";

String text = readPdf(filePath);

System.out.println(text);

```java
@Override
public String readPdf(String filePath) throws IOException {
    File file = null;
    if (filePath.startsWith("file:///")) {
        file = new File(new URL(filePath).getFile());
    } else {
        file = new File(filePath);
    }
    PDDocument document = PDDocument.load(file);
    PDFTextStripper stripper = new PDFTextStripper();
    String text = stripper.getText(document);
    text = text.replace( target: "\n",  replacement: "").replace( target: "\r",  replacement: "");
    document.close();
    return text;
```

4.5.6- article:

-addArticle() method

Description

The `addArticle()` method is used to add a new article to the database. It saves the `Article` object to the database.

Parameters

- `article` - an object of the `Article` class representing the article to be added.

Return Value

- Returns `true` if the article is added successfully.

Example Usage:java

Author author = authorService.findAuthor(1);

Article newArticle = new Article(author, "Title", "Content");

articleService.addArticle(newArticle);

-updateArticle() method

Description

The `updateArticle()` method is used to update an existing article's information in the database. It finds the article by ID and updates the article's author and content.

Parameters

- `article` - an object of the `Article` class representing the updated article information.

Return Value

- Returns `true` if the article is updated successfully.

- Returns `false` if the article does not exist in the database.

Example Usage: java

Article updatedArticle = new Article(1, author, "New Title", "New Content");

articleService.updateArticle(updatedArticle);

```java
@Override
public boolean updateArticle(Article article) {
    Article currentArticle=viewArticle(article.getId());

    if(currentArticle==null)return false;

    currentArticle.setAuthor(article.getAuthor());
    currentArticle.setContent(article.getContent());
    articleRepository.save(currentArticle);
    return  true;
}
```

-deleteArticle() method

Description

The `deleteArticle()` method is used to delete an article from the database. It finds the article by ID and deletes the article from the database.

Parameters

- `articleId` - an integer representing the ID of the article to be deleted.

Return Value

- Returns `true` if the article is deleted successfully.

- Returns `false` if the article does not exist in the database.

Example Usage: java

int articleId = 1;

articleService.deleteArticle(articleId);

```java
@Override
public boolean deleteArticle(int articleId) {
    Article currentArticle=viewArticle(articleId);
    if(currentArticle==null)return false;
    articleRepository.delete(currentArticle);
    return true;
}
```

-viewArticlesForAuthor() method

Description

The `viewArticlesForAuthor()` method is used to retrieve a list of all articles written by a specific author.

Parameters

- `authorId` - an integer representing the ID of the author whose articles are to be retrieved.

Return Value

- Returns a `List` of `Article` objects representing all articles written by the specified author.

Example Usage: java

int authorId = 1;

List<Article>authorArticles=articleService.viewArticlesForAuthor(authorId);

```java
@Override
public List<Article> viewArticlesForAuthor(int authorId) {

    Optional<List<Article>> articles=articleRepository.findArticlesForAuthor(authorId);
    if(articles.isPresent()&&articles.get().size()>0)
      return   articles.get();
    return null;
}
```

-viewArticle() method

Description

The `viewArticle()` method is used to find an article in the database based on its ID.

Parameters

- `articleId` - an integer representing the ID of the article to be found.

Return Value

- Returns an `Article` object representing the found article if the article exists in the database.

- Returns `null` if the article does not exist in the database.

Example Usage: java

int articleId = 1;

Article foundArticle = articleService.viewArticle(articleId);

```
3 usages
@Override
public Article viewArticle(int articleId) {
    Optional<Article>article=articleRepository.findById(articleId);
    if(article.isPresent())return article.get();
    return  null;
}
```

-loadSheet() method

Description

The `loadSheet()` method is used to load data from a CSV file into the database. It reads the CSV file and prints the data to the console.

Parameters

- The method does not take any parameters.

Return Value

- The method does not return anything.

```java
no usages
@Override
public void loadSheet() throws IOException, CsvValidationException {
    String filePath = "E:\\mahmoud\\projects\\gp\\trainArticles.csv";
    CSVReader reader = new CSVReader(new FileReader(filePath));
    String[] nextLine;
    while ((nextLine = reader.readNext()) != null) {
        String column1 = nextLine[0];
        String column2 = nextLine[1];
        System.out.println(column1 + ", " + column2);
    }
```

4.5.7- comment:

-addComment() method

Description

The `addComment()` method is used to add a new comment to the database. It sets the current date and time as the comment date and saves the `Comment` object to the database.

Parameters

- `comment` - an object of the `Comment` class representing the comment to be added.

Return Value

- Returns `true` if the comment is added successfully.

Example Usage: java

Article article = articleService.viewArticle(1);

Comment newComment = new Comment(article, "John Doe", "Great article!");

commentService.addComment(newComment);

```
1 usage
@Override
public boolean addComment(Comment comment) {
    comment.setDate(new Date());
   commentRepository.save(comment);
   return true;
}
```

-deleteComment() method

Description

The `deleteComment()` method is used to delete a comment from the database. It finds the comment by ID and deletes the comment from the database.

Parameters

- `commentId` - an integer representing the ID of the comment to be deleted.

Return Value

- Returns `true` if the comment is deleted successfully.

- Returns `false` if the comment does not exist in the database.

Example Usage: java

int commentId = 1;

commentService.deleteComment(commentId);

```
@Override
public boolean deleteComment(int commentId) {
    Comment comment=findComment(commentId);
    if(comment==null) return false;
    commentRepository.delete(comment);
    return true;
}
```

-findComment() method

Description

The `findComment()` method is used to find a comment in the database based on its ID.

Parameters

- `commentId` - an integer representing the ID of the comment to be found.

Return Value

- Returns a `Comment` object representing the found comment if the comment exists in the database.

- Returns `null` if the comment does not exist in the database.

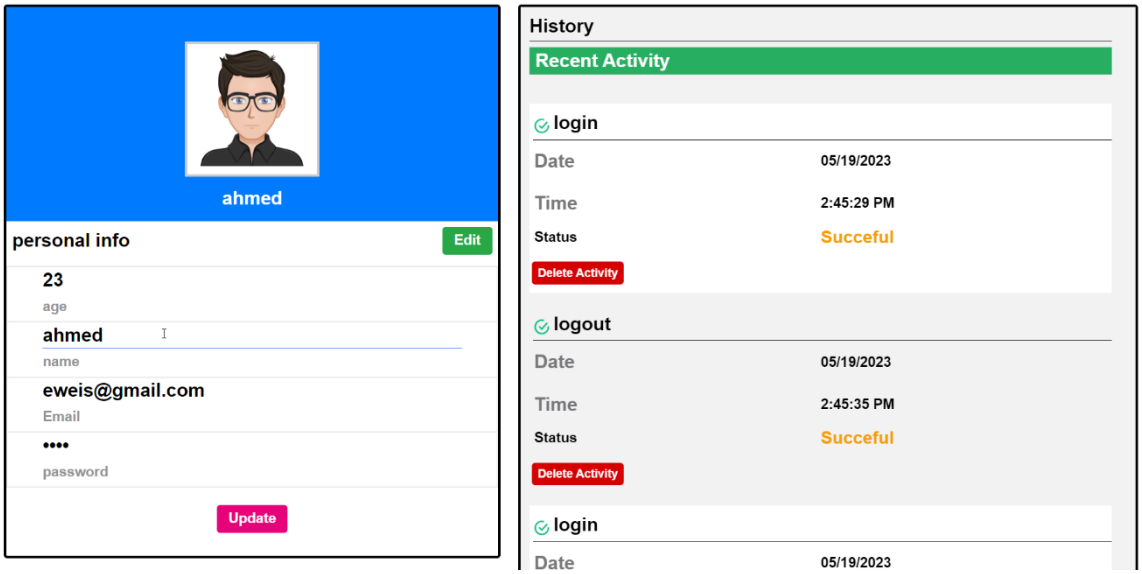
Example Usage: java

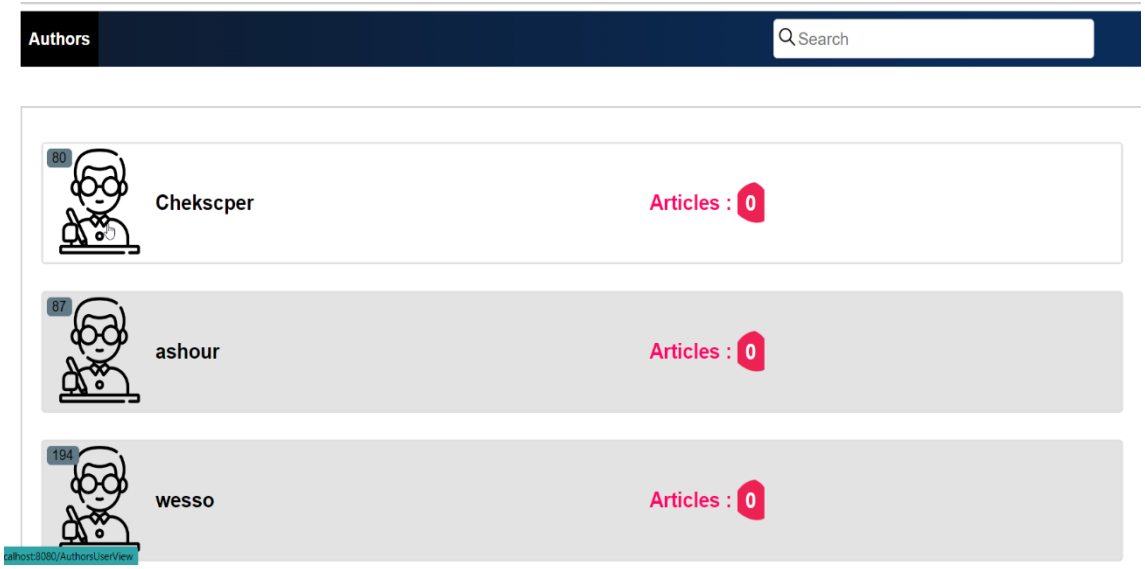
int commentId = 1;

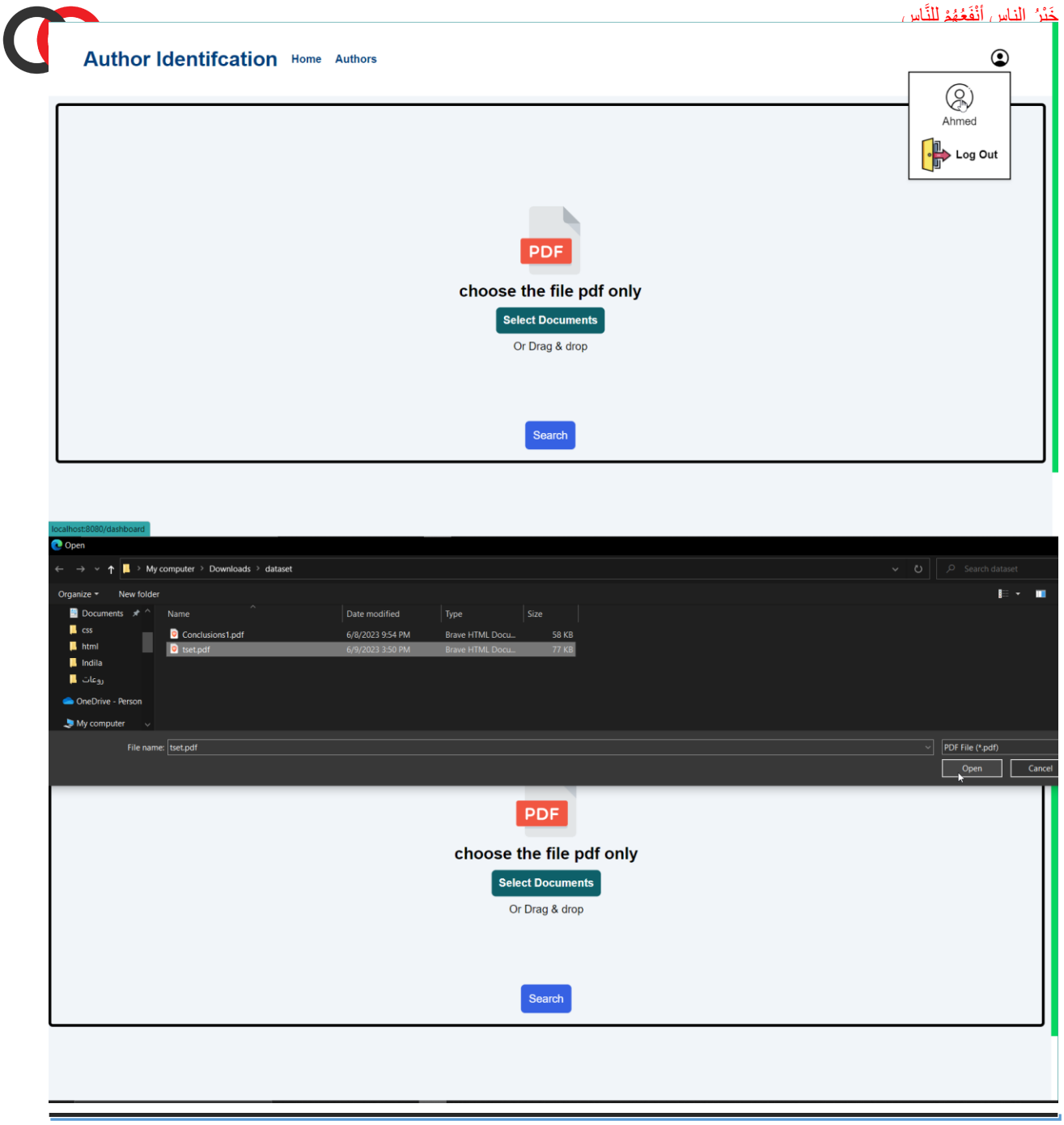
CommentfoundComment=commentService.findComment(commentId);

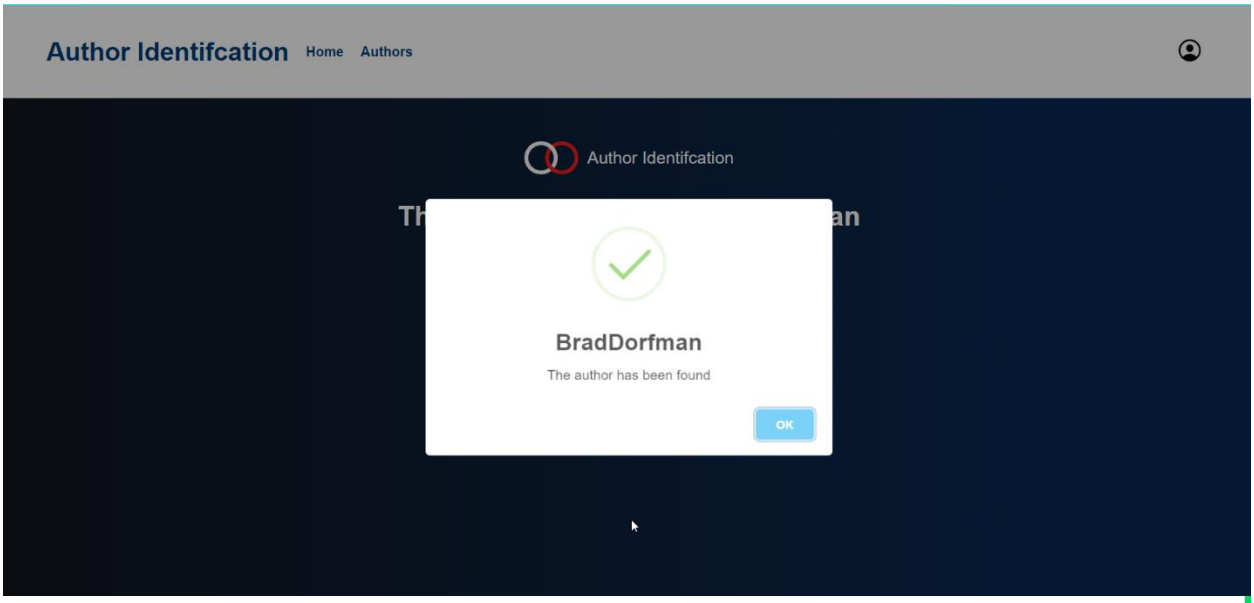
4.6User Interface Screens:

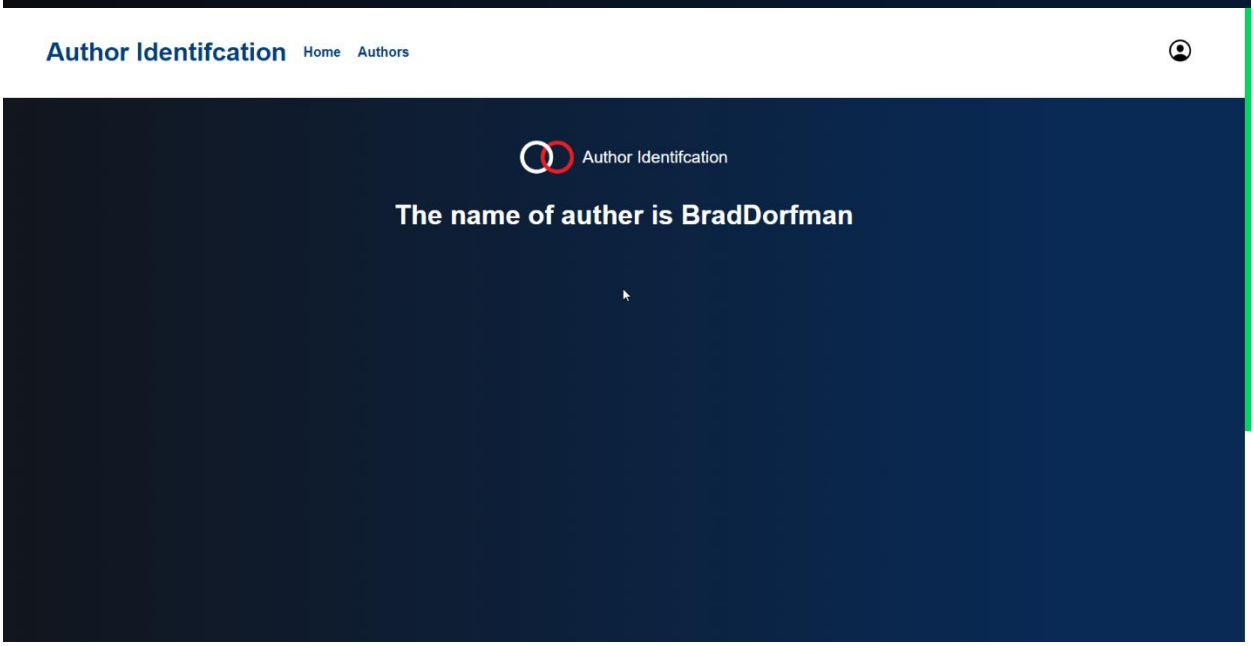
4.6.1User:



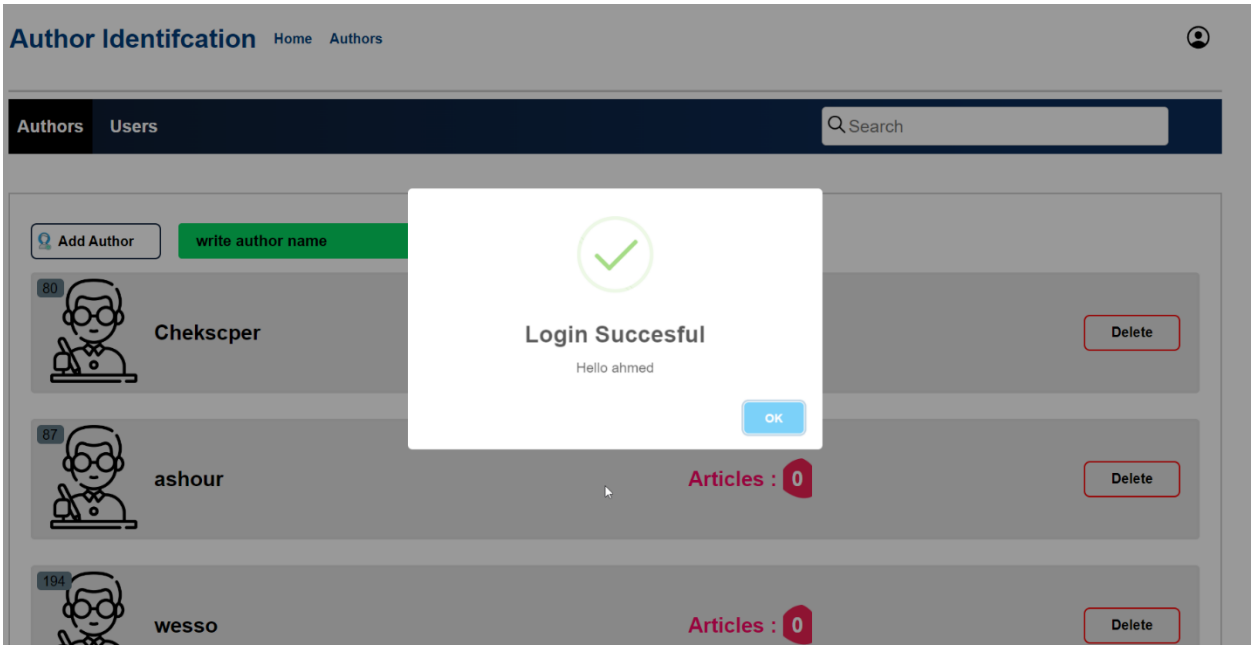


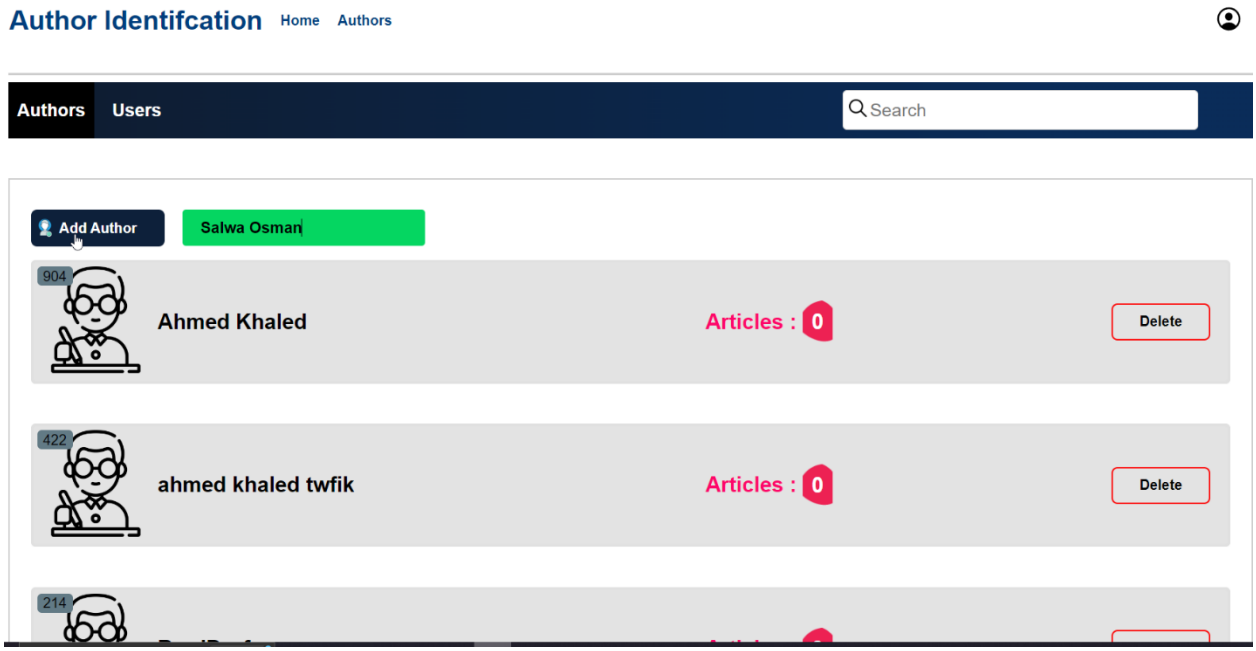

Author Identifcation
Home
Authors
Ahmed
Log Out
PDF
choose the file pdf only
Select Documents
Or Drag & drop
Search
localhost:8080/dashboard
Open
My computer
Downloads
dataset
Search dataset
Organize
New folder
Documents
css
html
Indila
روعات
OneDrive - Person
My computer
Name
Date modified
Type
Size
Conclusions1.pdf
6/8/2023 9:54 PM
Brave HTML Docu...
58 KB
tset.pdf
6/9/2023 3:50 PM
Brave HTML Docu...
77 KB
File name:
tset.pdf
PDF File (*.pdf)
Open
Cancel
PDF
choose the file pdf only
Select Documents
Or Drag & drop
Search

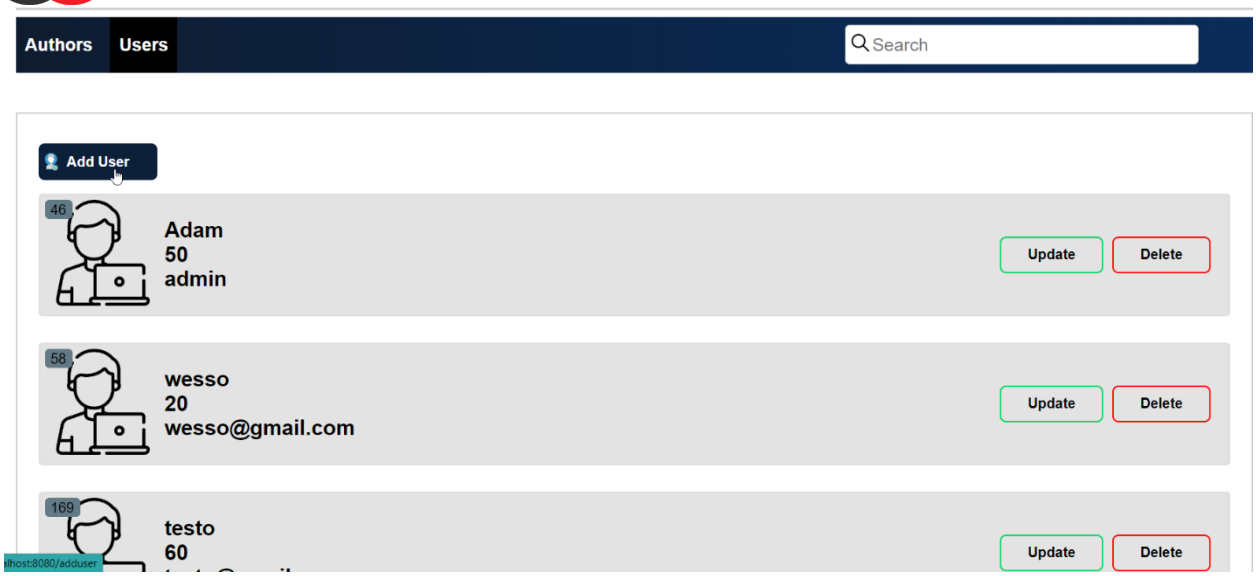

Author Identifcation
Home
Authors
Author Identifcation
BradDorfman
The author has been found
OK

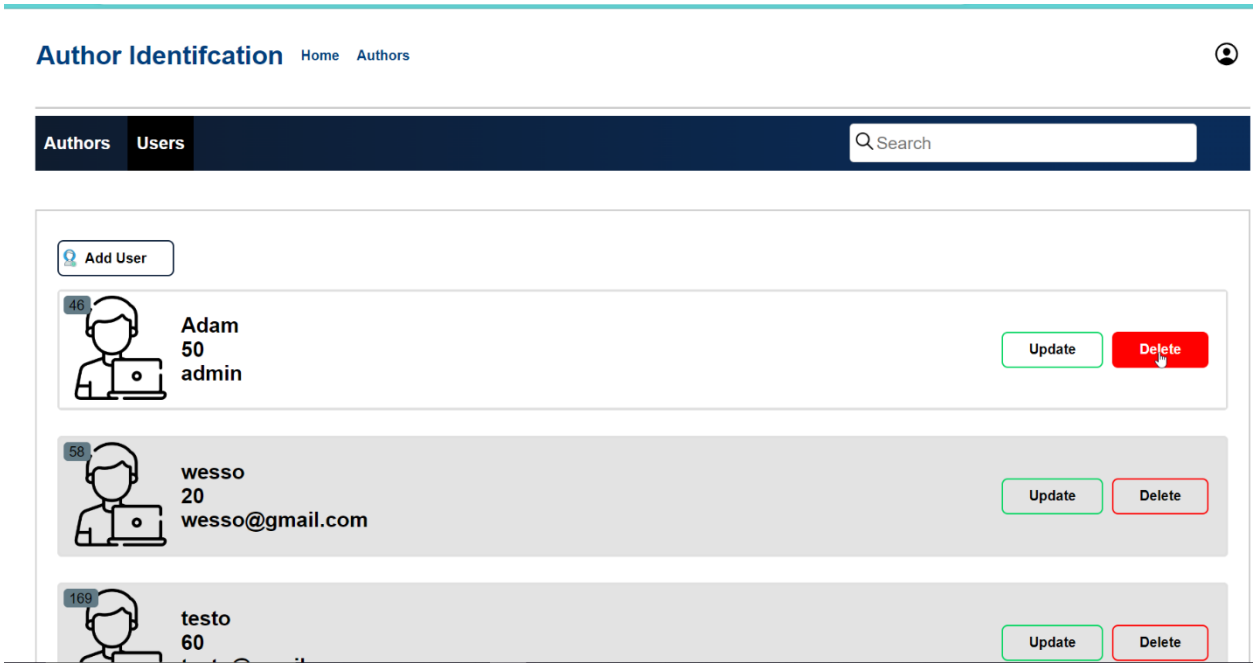

Author Identifcation
Home
Authors
Author Identifcation
The name of auther is BradDorfman

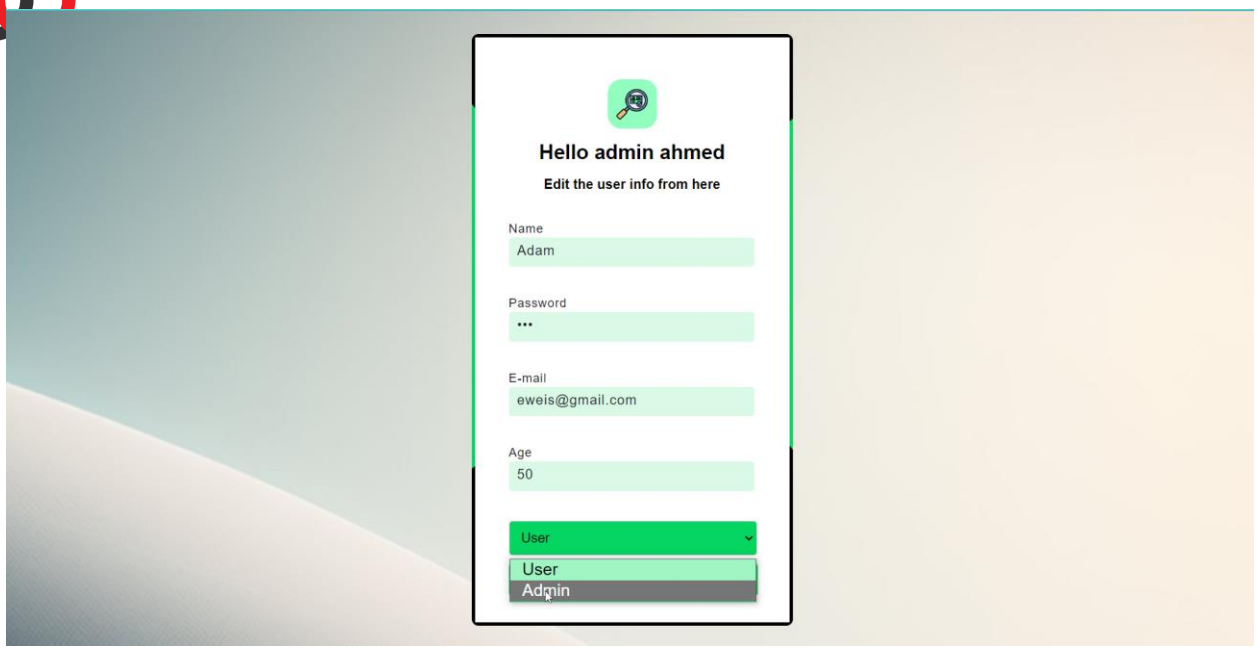
4.6.2Admin:

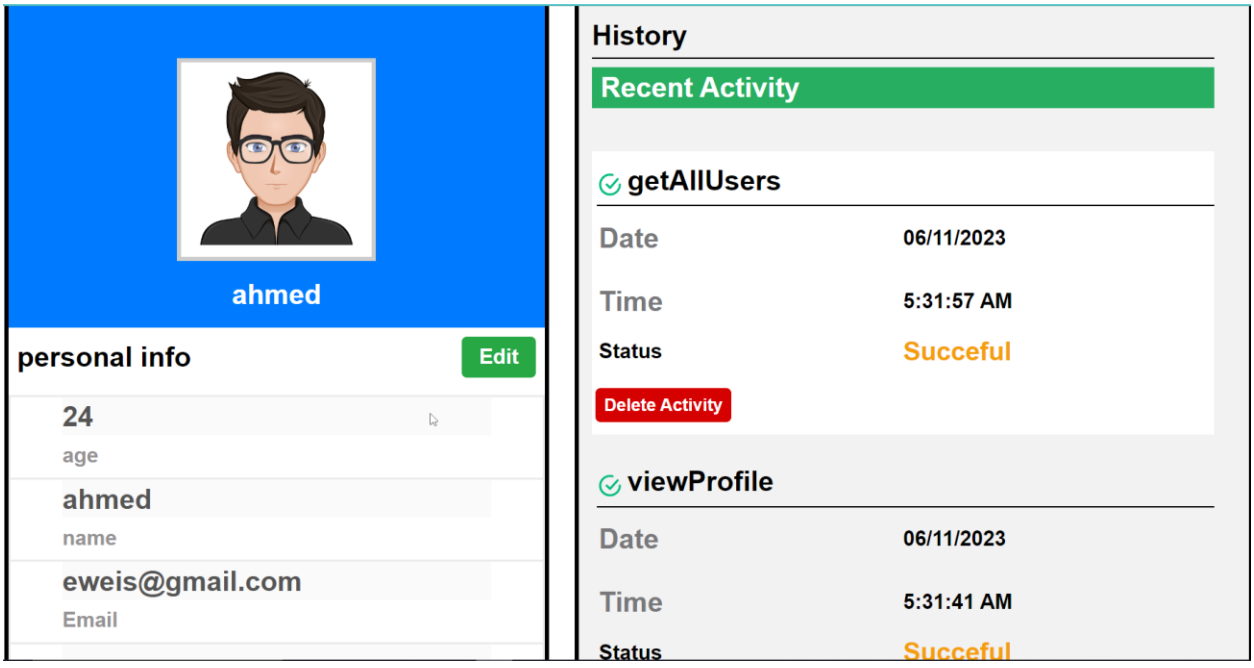

Author Identifcation
Home
Authors
Authors
Users
Search
Add Author
write author name
Login Succesful
Hello ahmed
OK
80
Chekscper
Delete
87
ashour
Articles : 0
Delete
194
wesso
Articles : 0
Delete

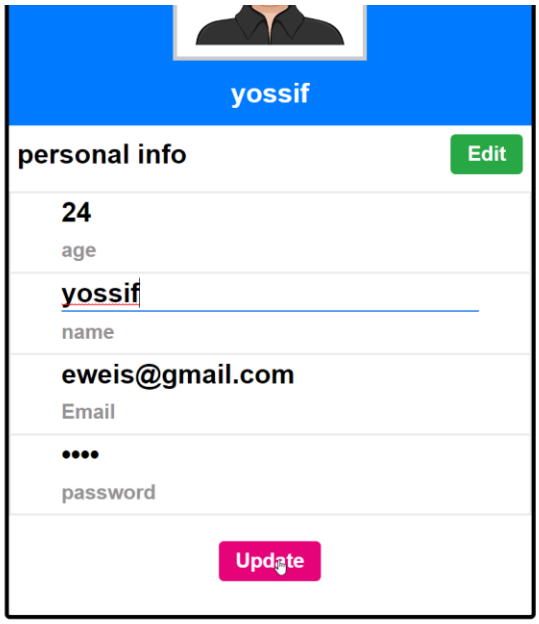

Author Identifcation
Home
Authors
Authors
Users
Search
Add Author
Salwa Osman
904
Ahmed Khaled
Articles : 0
Delete
422
ahmed khaled twfik
Articles : 0
Delete
214

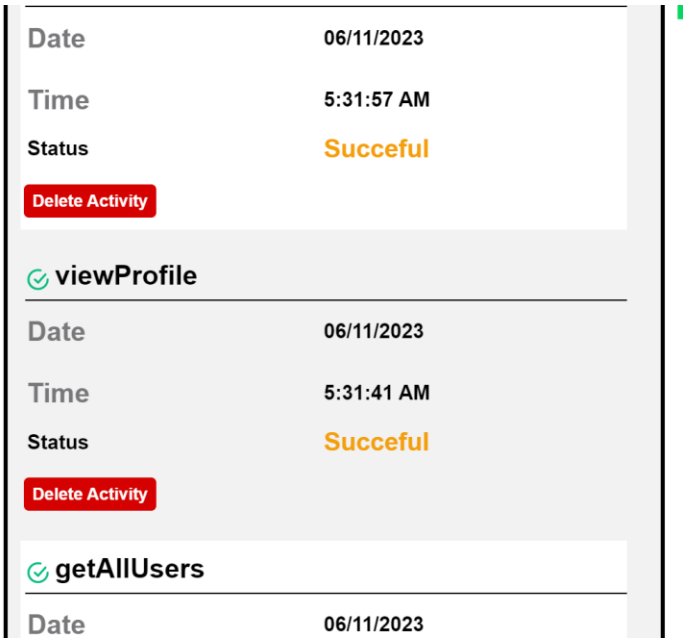

Authors
Users
Search
Add User
46
Adam
50
admin
Update
Delete
58
wesso
20
wesso@gmail.com
Update
Delete
169
testo
60
Update
Delete

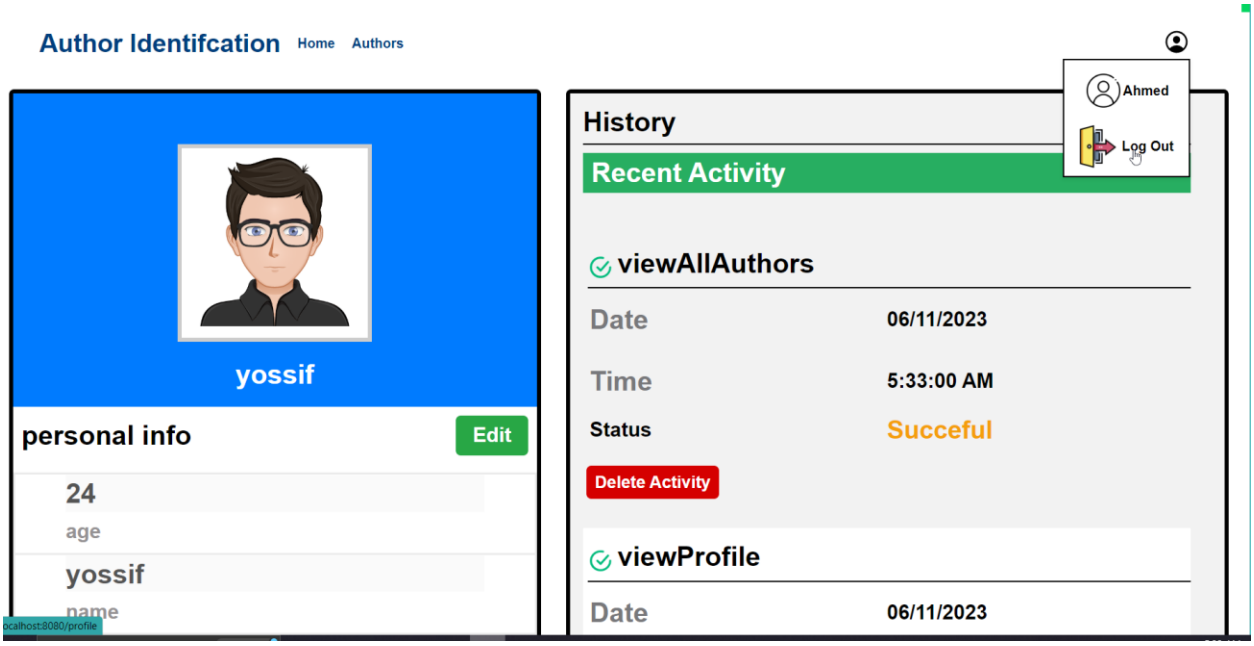

Author Identifcation
Home
Authors
Authors
Users
Search
Add User
46
Adam
50
admin
Update
Delete
58
wesso
20
wesso@gmail.com
Update
Delete
169
testo
60
Update
Delete


Hello admin ahmed
Edit the user info from here
Name
Adam
Password
•••
E-mail
eweis@gmail.com
Age
50
User
User
Admin


ahmed
personal info
Edit
24
age
ahmed
name
eweis@gmail.com
Email
History
Recent Activity
getAllUsers
Date
06/11/2023
Time
5:31:57 AM
Status
Succeful
Delete Activity
viewProfile
Date
06/11/2023
Time
5:31:41 AM
Status
Succeful


yossif
personal info
Edit
24
age
yossif
name
eweis@gmail.com
Email
••••
password
Update


Date
06/11/2023
Time
5:31:57 AM
Status
Succeful
Delete Activity
viewProfile
Date
06/11/2023
Time
5:31:41 AM
Status
Succeful
Delete Activity
getAllUsers
Date
06/11/2023


Author Identifcation
Home
Authors
yossif
personal info
Edit
24
age
yossif
name
Ahmed
Log Out
History
Recent Activity
viewAllAuthors
Date
06/11/2023
Time
5:33:00 AM
Status
Succeful
Delete Activity
viewProfile
Date
06/11/2023

4.7Results:

	Expected Result	Actual Result
LSTM Model	To build a model that achieved accuracy of 70%	The model only achieved 63%
Bert Model	To build a model that achieved accuracy of 75%	The model only achieved 71%
SVM Model	To build a model that achieved accuracy of 90%	The model only achieved 86%

4.8New Features:-

We developed a website that allows users to find the name of the author.

Users can identify the name of the author using either a PDF file or regular text.

The website enables users to interact with articles by adding comments and viewing previous comments.

4.9Discussion:

A new author identification system using advanced NLP and machine learning techniques has been developed, featuring LSTM, SVM, and BERT models. The LSTM model captures sequential information, the SVM model detects writing style nuances, and the BERT model accurately attributes authorship. A BERT-based deep learning model and a LinearSVC classifier with TF-IDF vectorization achieved high accuracy. The tool identifies authors, prevents plagiarism, and is user-friendly. It represents a significant contribution to computational linguistics and text analysis.

Chapter 5

Testing

In this chapter we tested our application and tried different test cases

To ensure that the application have reached the level of quality we are

Targeting.

Introduction:

Software testing is an indispensable part of the software development process, and it plays a critical role in ensuring that software is of high quality, reliable, and performs as expected. By understanding the importance of testing and the different testing techniques available, developers can ensure that their software meets the necessary quality standards and provides a positive user experience. With the increasing complexity of software systems, testing is becoming even more critical, and developers must make testing an integral part of their software development process to ensure that their software meets the highest quality standards.

There are several types of testing that are commonly used in software development. Each type of testing serves a specific purpose and should be used at different stages of the development process. In this section, we'll describe some of the most common types of testing used in software development.

Unit Testing:

This type of testing involves testing individual components or modules of the software in isolation. The purpose of unit testing is to ensure that each component of the software performs as expected and meets the required specifications. Unit testing is typically done by developers using automated testing tools, and it is usually the first type of testing performed in the software development process.

Integration Testing:

Integration testing involves testing the interactions between different components of the software. The purpose of integration testing is to ensure that the components of the software work together as expected and that there are no compatibility issues. Integration testing is typically performed after unit testing is completed.

System Testing:

System testing involves testing the entire system as a whole. The purpose of system testing is to ensure that the software meets all the functional and non-functional requirements and that it performs as expected in a real-world environment. System testing is typically performed by a dedicated testing team.

Acceptance Testing:

Acceptance testing involves testing the software to ensure that it meets the user's requirements and that it is ready for release. The purpose of acceptance testing is to ensure that the software is of high quality and that it meets the user's expectations. Acceptance testing is typically performed by the end-users or a dedicated testing team.

Test-driven development (TDD) is a software development approach that emphasizes writing automated tests before writing any production code. This approach helps to catch errors early in the development cycle, which reduces the time and cost involved in fixing them later. By writing tests before writing code, developers can ensure that the code is thoroughly tested and that any errors or bugs are caught early in the development cycle. TDD also helps developers to write more modular and maintainable code, as the code is written in small increments and is continuously tested. By using TDD, developers can improve the quality of their code, reduce the cost and time required for development, and ensure that their software meets the user's requirements.

Testing Spring Boot applications is an essential part of the software development process. By writing tests for controllers, services, repositories, and other components, we can ensure that the application functions as expected and meets the user's requirements. In this article, we explained how to write tests for various scenarios, such as testing RESTful APIs, database interactions, and error handling. We also provided examples of how to use the appropriate testing frameworks and

utilities to test different components of the application. By writing comprehensive tests, we can improve the quality of our Spring Boot application, reduce the risk of errors and bugs, and ensure that our application meets the highest quality standards.

We can use the appropriate testing frameworks and utilities such as Spring MVC Test, JUnit, and Spring Data JPA Test to write tests for different components of the application. By writing comprehensive tests, we can ensure that our Spring Boot application meets the highest quality standards and is of high quality, reliable, and efficient.

5.1Test cases:

Test ID	Test case	Test scenario	prerequisites	Test procedure	Test data	Expected result	Actual result	Status
1	Valid register	Create new user	New user	1- Enter not used email. 2- Enter valid password 3- enter name 4- enter age	Name: Mohamed email: c@gmail.com password :12456 age : 22	Register done and navigated to log in screen	Register successfully	pass
1	Invalid register	Create new user with used email	Internet connection	1- Enter used email. 2- Enter valid password. 3- enter name 4- enter age	Name: Mohamed email: c@gmail.com password :12456 age: 22	Cannot register and error message appear.	Register failed. Error message: the email has already been taken	pass
2	Valid login	Enter valid email and password	Should be registered	1- Enter valid email. 2- Enter valid password	email: c@gmail.com password :12456	Logged in	Login successfully	pass
2	Invalid login	Enter invalid email valid password		1- Enter invalid email. 2- Enter valid password	email: co@gmail.com password :12456	Cannot login	Login failed and error message appear : email or password not correct	pass
2	Invalid login	Enter valid email invalid password		1- Enter valid email. 2- Enter invalid password	email: c@gmail.com password :124546	Cannot login	Login failed and error message appear : email or password not correct	pass
3	Valid update User	Enter valid email is do not use	Should be login	1- Enter not used email. 2- Enter valid password 3- enter name 4- enter age	Name: Mohamed22 email: c123@gmail.com password :124567 age: 25	Can update	Update successfully	pass

3	Invalid update user	Enter invalid email is use	Should be login	1- Enter not used email. 2- Enter valid password 3- enter name 4- enter age	Name: ahmed email: c123@gmail.com password :35438263 age: 30	Cannot update	Update failed. Error message: the email has already been taken	pass
4	Valid add role	Enter name do not used	Should be login as admin	1- enter not used name	Name: admin	Can add	add successfully	pass
4	Invalid add role	Enter name is used	Should be login as admin	1- enter used name	Name: admin	Cannot add	Add unsuccessfully	pass

Chapter 6

Conclusion and Future Work

In this chapter we will propose what are the future plans we have

Concerning our application as it will not end by the end of the discussion.

and sum up the whole things and summarize the things mentioned before.

6.1Conclusion:

-The developed author identification system utilizing cutting-edge techniques in natural language processing and machine learning has the potential to revolutionize the field of author identification, enabling applications in forensic linguistics, plagiarism detection, and content verification.

-The combination of LSTM, SVM, and BERT models demonstrates the versatility and robustness of the approaches used in the project, allowing for a more holistic analysis of textual data, resulting in reliable and accurate authorship attribution.

-The success of the LSTM model in identifying unique authorial traits and patterns by harnessing the power of LSTM's ability to capture and process sequential information, the remarkable performance of the SVM algorithm in discerning subtle nuances in writing styles and establishing authorship with high precision, and the noteworthy accuracy achieved by the BERT model in accurately attributing authorship, demonstrate the potential of state-of-the-art models in author identification.

-The developed deep learning model for author identification based on text articles using the BERT model provides a high-level interface for TensorFlow and Keras, enabling accurate author identification on new text articles.

-The LinearSVC classifier and TF-IDF vectorization used in the SVM-based author identification approach demonstrated high accuracy in identifying authors, indicating the potential of traditional machine learning techniques in author identification.

-this project has successfully achieved its objectives of developing a computational tool that can automatically identify the author of a given text based on their unique writing style, developing algorithms that can analyze various linguistic features to generate a writing style profile for each author, creating a user-friendly interface that allows users to easily upload and analyze text documents, and testing the software on various datasets.

-The development of this tool will not only aid in identifying the authorship of texts, but also help prevent plagiarism by ensuring that authors receive proper

attribution for their work. The accurate algorithms and user-friendly interface will make the tool accessible to a wide range of users, from researchers to authors to publishers.

-Overall, this project represents a significant contribution to the field of computational linguistics and text analysis, and has the potential to improve the accuracy and reliability of authorship attribution in various domains.

6.2Future Work:

One idea that was considered for our project was the use of social media to gather data and implement a system that can predict the author of posts based on language, tone, and content. This would allow us to identify individuals who start rumors and prevent the spread of false information. Unfortunately, due to time constraints, this idea was not fully developed.

Another idea that could be implemented is plagiarism detection. By comparing the language and writing style of a given text to other known texts, instances of plagiarism can be identified. This would be a valuable tool for educators, publishers, and other professionals who require original content.

Both of these ideas have the potential to enhance our project and improve its functionality. However, they require significant resources and time to implement effectively. As such, it is important to carefully consider the feasibility and benefits of each idea before deciding to pursue them.