Data Brokers

|	
Chirag Shah (chirags)	chirags@seas.upenn.edu
Samarth Shah (ssamarth)	ssamarth@seas.upenn.edu
Prahalad Venkataramanan (prahalad)	prahalad@seas.upenn.edu
Akshay Sriraman (akss)	akss@seas.upenn.edu

https://github.com/chirags27/CIS550_Data_Brokers/upload/master

INTRODUCTION:

Our team Data Brokers cracked the challenge of designing a DLMS - DataLake Management System that deals with accepting files of different formats from open web data sources for linking and storage. The implementation extracts data in the form of nodes and then generates links across the entire datalake. This linking is exploited to return results to queries on the linked/stored data. There are two levels of permissions (root access and base access) which are associated with every search query and the result or the graph which is output to the user is in accordance with the authorization level of the user.

ARCHITECTURE:
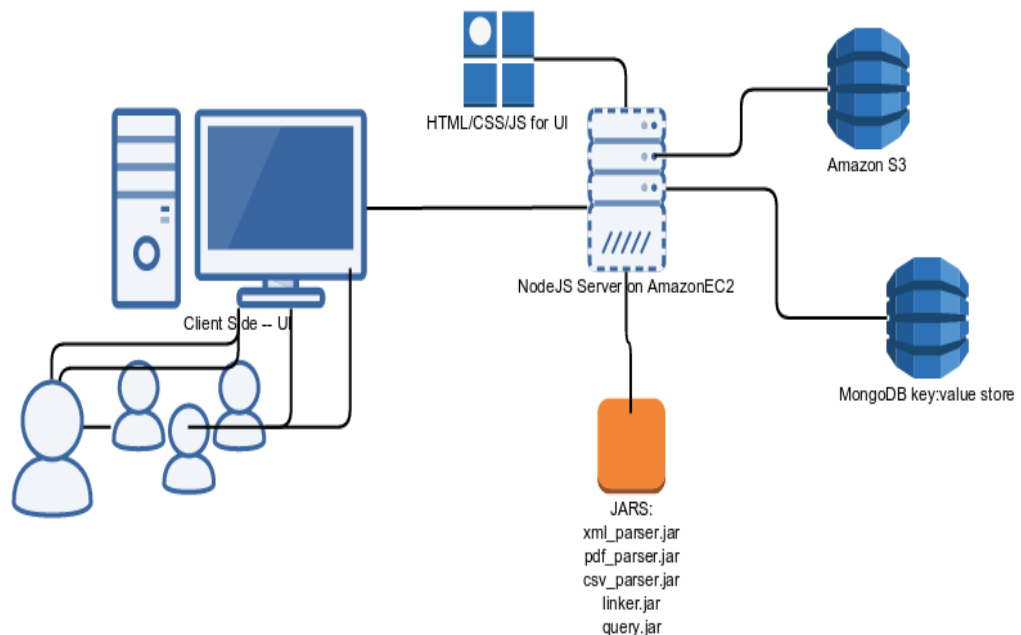
COMPONENTS' IMPLEMENTATION:

1. <u>UI:</u>
   SIGNUP & LOGIN:
   We have designed a visually pleasing UI incorporating flipping of the forms in order to signup and login for users. This gives a nice effect and is quick as it does not load a different page. Once the user signs up and logs in, he is forwarded to a search page where he can enter different single word and two word search queries.
   - ***SEARCH:***
   The search page provides a text field to type different queries. When you type the queries, it first gets processed in order to stem the words. We are using Porter stemmer in the front end in NodeJS and PorterStemmer for storing the stemmed inverted indices in the back end in Java. The search page also provides radio buttons to check incase of root and base permissions. Root permission gives added access of .csv and pdf files as well in case you give the correct root password (admin). This leads to visualizing some added results in case there is a path that goes through root permitted files. Whereas under the base permission, a restricted access won't give you results that involves any links that goes through .pdf and .csv files.
   - ***RESULT:***
   The result page displays the linked directed graph for the two search word queries. It is displayed in a background of the universe with every node representing a star. We are displaying the best possible path linking the two queries. The other possible paths can be observed from the command line. In case of a single word query, we display all the files in which the particular word is present.

CIS 550 Project Final Report

Login Page

SIGNUP PAGE

CIS 550 Project Final Report
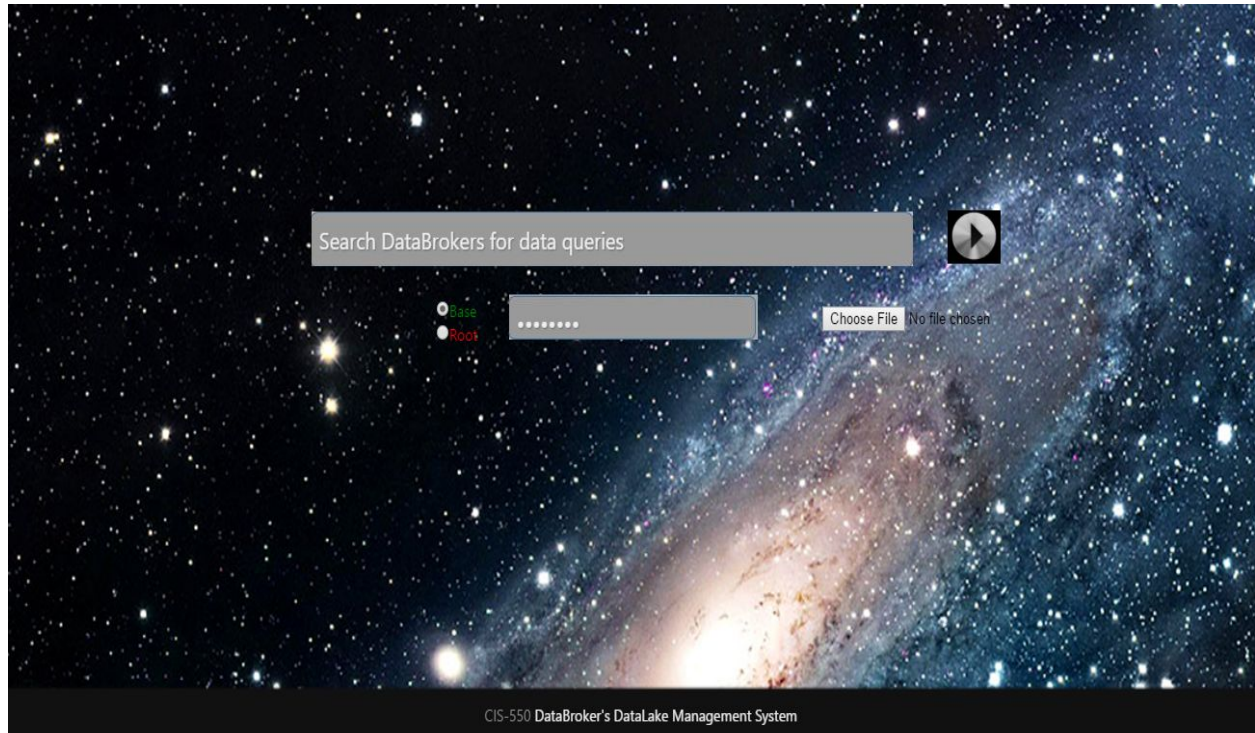
SEARCH PAGE
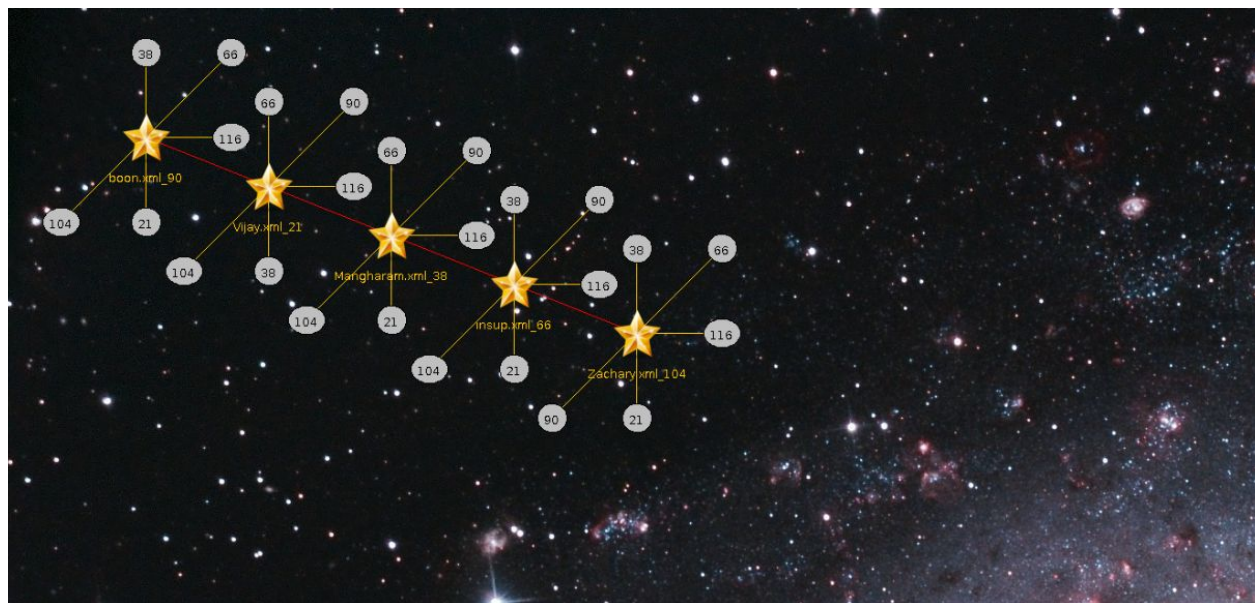
RESULT FOR BOON AND ZACHARY

RESULTS FOR CIS_505 and CIS_555

2. Parsing and Extraction:

- **Input:** Our datalake management system can take files of .pdf, .xml, .csv and .json input formats. Each of these files is parsed by a function specifically dealing with the file format.
- **Extraction:** The parser extracts the information from these input files in the form of key-value pairs. The "author" field of each document is the key a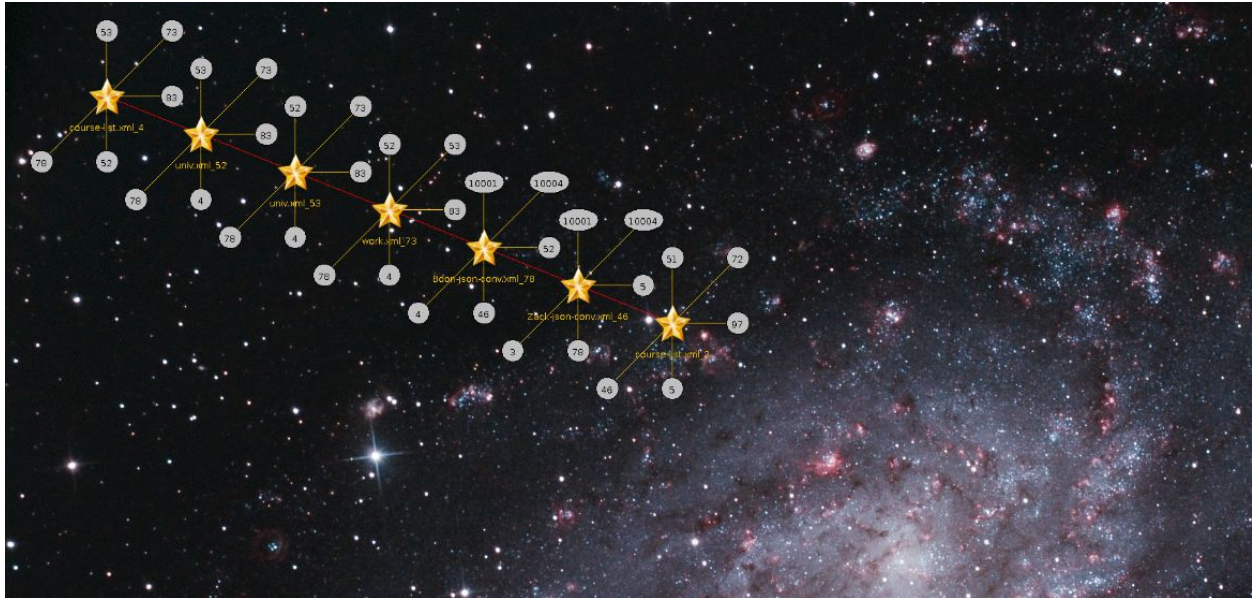nd the name of the author is the value. Space separated author names (values) are considered to be multiple values for the associated key. The parser assign node IDs on the fly as and when documents are added. Node IDs ranging from 1 - 10,000 are allotted for .xml files, 10,000 - 30,000 for .pdf files and values above 30,000 for .csv files. The.json type files are converted to .xml files and then parsed.
- **Storage:** The parsed data is stored as - nodeID, key, value, parentID, document name and document type - format using MongoDB.

This is the snapshot of the extraction table as it is stored in the database


```
> db.ext_table.find()
{ "_id" : 30000, "key" : "Name", "value" : "Boon", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30001, "key" : "PHD", "value" : "Berkeley", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30002, "key" : "MS", "value" : "Stanford", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30003, "key" : "DEPARTMENT", "value" : "CIS", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30004, "key" : "GRAD_YEAR", "value" : "2006", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30005, "key" : "WORKED", "value" : "Microsoft", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30006, "key" : "COUNTRY", "value" : "USA", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30007, "key" : "UNIVERSITY", "value" : "UPENN", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30008, "key" : "Name", "value" : "Kostas", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30009, "key" : "PHD", "value" : "Karlsruhe", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30010, "key" : "MS", "value" : "Karlsruhe", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30011, "key" : "DEPARTMENT", "value" : "CIS", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30012, "key" : "GRAD_YEAR", "value" : "1992", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30013, "key" : "WORKED", "value" : "GermanArmy", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30014, "key" : "COUNTRY", "value" : "USA", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30015, "key" : "UNIVERSITY", "value" : "UPENN", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30016, "key" : "Name", "value" : "Rahul", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30017, "key" : "PHD", "value" : "CMU", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30018, "key" : "MS", "value" : "CMU", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
{ "_id" : 30019, "key" : "DEPARTMENT", "value" : "ESE", "parent_id" : 0, "doc_name" : "data-professor", "type" : "csv" }
```

CIS 550 Project Final Report

Along with the extraction table, an inverted index table is also stored in the database which maps a given key/value to a given list of nodeIDs which is separated by delimeter "_" and the size which stores the information about how many nodeIDs are associated with a given word.

```
> db.inv_table.find()
{ "_id" : "2006", "value_list" : "30004", "size" : 1 }
{ "_id" : "Microsoft", "value_list" : "30005", "size" : 1 }
{ "_id" : "Karlsruhe", "value_list" : "30009_30010", "size" : 2 }
{ "_id" : "1992", "value_list" : "30012", "size" : 1 }
{ "_id" : "ESE", "value_list" : "30019", "size" : 1 }
{ "_id" : "2008", "value_list" : "30020", "size" : 1 }
{ "_id" : "1983", "value_list" : "30028", "size" : 1 }
{ "_id" : "Name", "value_list" : "30000_30008_30016_30024_30032", "size" : 5 }
{ "_id" : "PHD", "value_list" : "30001_30009_30017_30025_30033", "size" : 5 }
{ "_id" : "MS", "value_list" : "30002_30010_30018_30026_30034", "size" : 5 }
{ "_id" : "DEPARTMENT", "value_list" : "30003_30011_30019_30027_30035", "size" : 5 }
{ "_id" : "CIS", "value_list" : "30003_30011_30027_30035", "size" : 4 }
{ "_id" : "GRAD_YEAR", "value_list" : "30004_30012_30020_30028_30036", "size" : 5 }
{ "_id" : "1984", "value_list" : "30036", "size" : 1 }
{ "_id" : "WORKED", "value_list" : "30005_30013_30021_30029_30037", "size" : 5 }
{ "_id" : "GOOGLE", "value_list" : "30037", "size" : 1 }
{ "_id" : "COUNTRY", "value_list" : "30006_30014_30022_30030_30038", "size" : 5 }
{ "_id" : "USA", "value_list" : "30006_30014_30022_30030_30038", "size" : 5 }
{ "_id" : "UNIVERSITY", "value_list" : "30007_30015_30023_30031_30039", "size" : 5 }
{ "_id" : "UPENN", "value_list" : "30007_30015_30023_30029_30031_30039", "size" : 6 }
```

3. **Indexer:**
   Indexer takes the key-value pairs which are outputted from the extractor and indexes them by key and stores the data in a separate MongoDB table. Every key gets processed first. We basically stem the words and remove special characters before storing the output in the table. We do the indexing at the same time of extraction reducing the processing time of separately performing an indexing job as we have access to every node in the extraction part as well and we go on feeding and updating the indexer table.

4. **Linker:**
- *Data:* All the extracted documents are stored into the local MongoDB on Amazon EC2. This is stored in a way that there are unique node ID's assigned to all the nodes in the extraction phase and these "key:values" are then used by the linker.
- *Logic:* The linking part is done by comparing every key that is stored in the "ext_table" to every other key which has the same key or the same value. Same procedure is repeated for linking the values of every node with the keys and values of every other node. The complexity of the linking algorithm used will be O(n^2). For a particular match of a key with some other key/value, the linker will store a corresponding entry in the linker_table in mongoDB.
- *Algorithm:* The format used to store the links is "n1" → node1, "n2" → node2, weight → 1. The weight attribute here corresponds to the weight of the link and this is used to compute the shortest path in the cases where the search is done by the user. The number of hops taken by the search algorithm (which is basically DFS or depth first search) will decide the ranking of the path and the shortest path will then be displayed as a graph with nodes and edges on the UI.

This is a snapshot of the linker table as it is stored in the MongoDB:

```
> db.linker_table.find()
{ "_id" : ObjectId("57313b19e4b0640270b93014"), "n1" : 30000, "n2" : 30008, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93015"), "n1" : 30000, "n2" : 30016, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93016"), "n1" : 30000, "n2" : 30024, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93017"), "n1" : 30000, "n2" : 30032, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93018"), "n1" : 30001, "n2" : 30009, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93019"), "n1" : 30001, "n2" : 30017, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b9301a"), "n1" : 30001, "n2" : 30025, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b9301b"), "n1" : 30001, "n2" : 30033, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b9301c"), "n1" : 30002, "n2" : 30010, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b9301d"), "n1" : 30002, "n2" : 30018, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b9301e"), "n1" : 30002, "n2" : 30026, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b9301f"), "n1" : 30002, "n2" : 30034, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93020"), "n1" : 30003, "n2" : 30011, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93021"), "n1" : 30003, "n2" : 30019, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93022"), "n1" : 30003, "n2" : 30027, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93023"), "n1" : 30003, "n2" : 30035, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93024"), "n1" : 30004, "n2" : 30012, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93025"), "n1" : 30004, "n2" : 30020, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93026"), "n1" : 30004, "n2" : 30028, "wt" : 1 }
{ "_id" : ObjectId("57313b19e4b0640270b93027"), "n1" : 30004, "n2" : 30036, "wt" : 1 }
```

5. **Search algorithm and querying:**
   The search algorithm used is Depth First Search on the linker table. The words to be queried are obtained from the node js server running on EC2 which are then used as arguments for running a .jar file in background and getting its output. The way it works is as follows:
   - Initially the linker table is cached into the memory. This is done by reading it from mongoDB and storing it in a hashmap<node_id, List<neighbours>> .
   - When the user enters a search query, all the nodes corresponding to the words are fetched from the inverted index table or the inv_table in the mongoDB. One of the nodes is considered to be the source and other one is considered to be the destination.
   - For all such source → destination pairs a path is computed using DFS. Several feasible paths are computed and then stored into the tree map so that they are sorted according to their weight (which is based on the number of hops). The shortest path is then given to the graph functions which actually plot the graph and store it as an image file.
   - This image file is then rendered on the client side by the node server after all the computation. A typical query evaluation takes almost 10-15 seconds as all the feasible paths are computed for every node.

ROADMAP:

Date	Component
04/17	Understanding of problem statement and design challenges
04/19	Design and Discussion of the initial architecture of our DLMS system and forming our own datasets of various file formats

CIS 550 Project Final Report

04/23	Choice of implementations and softwares/file formats, Extraction of pdf and csv formats
04/28	Extraction of .xml and JSON file formats
04/29	Indexing the extracted files
05/02	Designed the algorithm for Linking
05/04	Implementation of linking algorithm
05/07	Implementation of the search algorithm and UI for our DLMS, Integration of the UI and results already computed and retrieving from MongoDB
05/09	Final testing and validation of our implementation with relevant pointed test cases and final report for submission.

Extra Credit Claimed:
1. Natural Language Processing (Stemming, Stop words remover (for pdfs) and special character remover
2. Upload button for a file
3. Optimized UI (SignUp and LogIn pages-flip UI form on click rather than loading the entire page again)
4. Novel algorithmic contributions - Optimized algorithm for the Linker-Gave us pretty quick results for the UI
5. Creative use of novel datasets - Formed our own dataset comprising of different existing file formats that had relevant links that resulted in a neat visible output.

Team member contributions:

1. *Akshay Sriraman:* Extraction of .csv, .pdf and collated open web data for various file formats
2. *Chirag Shah:* Designed the linking algorithm and the storage api involving mongoDB
3. *Prahalad Venkataramanan:* Handled web application for query-search output graph of nodes
4. *Samarth Shah:* Extraction of json and xml file formats and keyword search tasks + Natural Language Processor