CIS 550 Final Project Write-up

Data lake Manage System

TEAM XXXL

Zhi Xu
Yicheng Lin
Jiawei Xue
Xin Zhang

Acknowledgement

We would like to take this opportunity to express our special appreciation and thanks to our Professor Zachary Ives. We would like to thank you for giving us the brilliant lectures. We would also like to thank our great TAs. They spend a lot of time grading and giving feedback on our assignments. All of you have been there to teach and supports us through the whole semester.
Thanks to ourselves, we did a good job.

● Introduction

Follow from the project template, there is a big amount of demand on providing a data lake manage system for big data revolution. The key to this data lake if how each data file would be arranged in the system and how to store as well as to fetch from the database. The basic problem we are facing is to determine a way to extract the data first from the file and try to come up with some ideas on how to build connections between different files as well as the data nodes inside the file, link between node key, value or even files. Differ from the google search engine or other kind of search method, the data lake should perform a data-structured search method by which not only the keywords in a file would be found, other data connected or linked to the key would also come out as a result. In this way, people would get a better way to look at all the data in the data lake and do other high-level structure management base on the result. Our challenges are to build such a system which would convert data file into different structures, created links in between and as well as creating some search methods for a structured result.

● Database Architecture

Our top-level structure is shown as the figure 1.

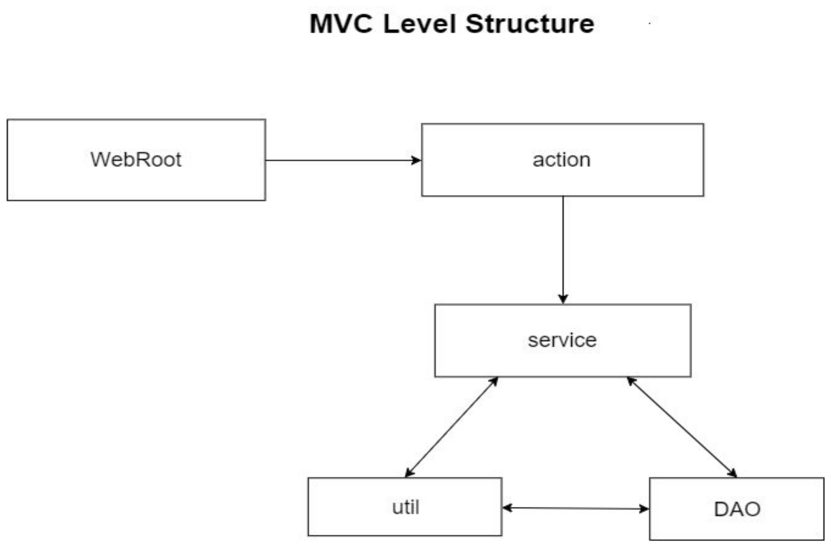

Figure 1. Top level structure

Webroot: define web page structure and design: error page, homepage, index, login, register, registersuccess, success.
Action: define action handler when webpage action sent, takes requests from webroot.
Service: define methods used for file operation including parser extractor, set

up links and updates links, interaction with DAO and util.
Util: define all the methods used by DAO and service, including data search, parser graph extract methods (JSON,XML,CSV and Tika for other files.),

Our data lake architecture is shown as the following figure 2.

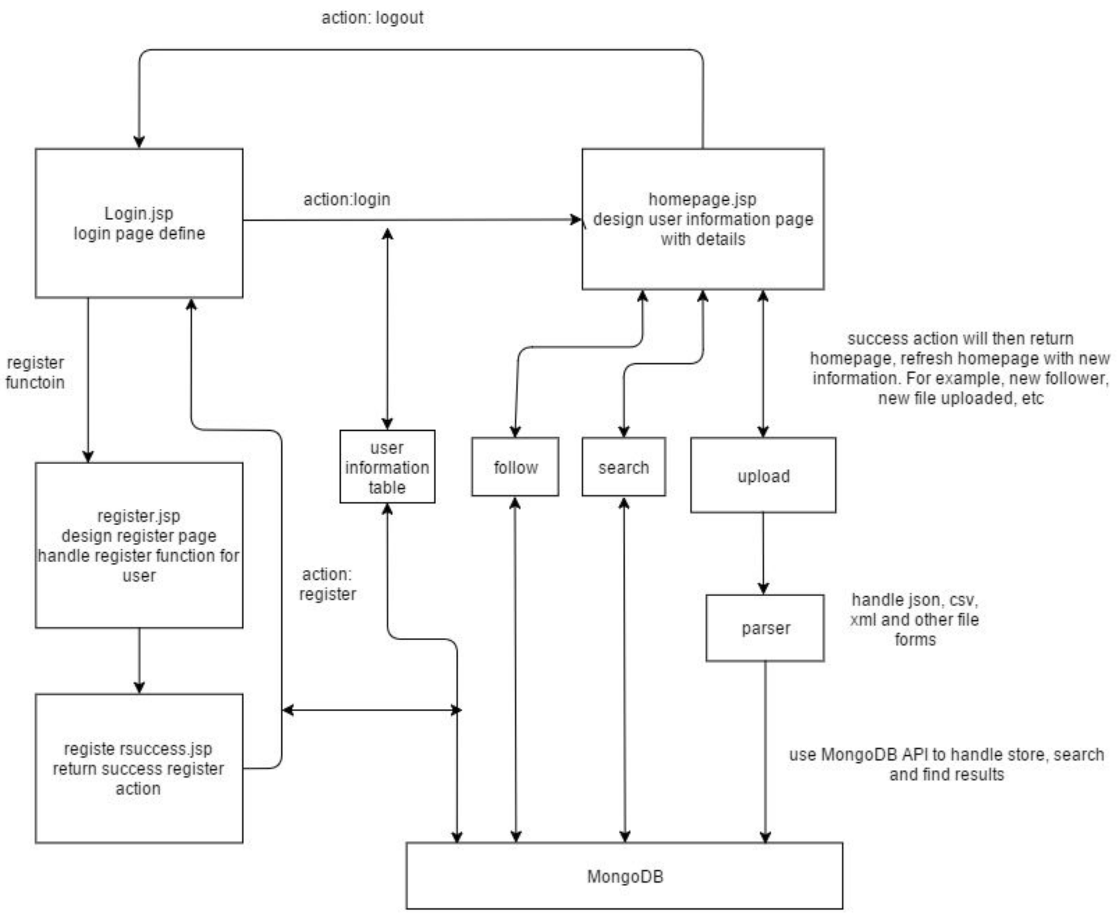

Figure 2. Data lake architecture

Login: The login is the initial page for the user login which handles the user's username and password. Upon the login action, it will check with the user information table in the MongoDB to see if a user is already registered. Then with success login action, it will jump to the homepage block.

Register: The register is used for handling user register action. The register jsp has functions to take user new name and password in, while connect with a user table in the MongoDB to store user information.

Homepage: The homepage.jsp will define the user profile information including the user name, user's followers and the file under the current user and his follower. The homepage.jsp also has input block for search keyword: first keyword and second keyword, file upload input block and follow input block. In the homepage, there is also a window which can display the file structure visualization. Using a Sigma.js open source API library, each node in the file and links are displayed as a graph.

Follow: follow in the homepage handles the follow action. It takes the following input from the homepage.

Search: search in the homepage handles the search action, take the first and second keyword into searchHelper method to search in the database.

Upload/Parse: upload block take the file input and detect which type it is and pass into correspond parser to parse the file and send to database. It will use the MongoDB API to handle the file object.

● Implementation

Login:
In this web page, the basic components include:
Form, which is used to record the information about user name and password.
Login button, which is used to submit the form and then call the method in file XXXL/src/com/xxxl/action/UserAction.java. Once such method is called, the connection with corresponding MongoDB collection will be established, and query will be conducted. It will return success and redirect into Homepage if both username and password are matched, otherwise, it will return error.
Register link, which is a link to Register page. Once it is clicked, then the Register page will be reached.

Register:
Basic components include:
Form, which is used to record the information about user name and password.
Register button, which is used to submit the form and then call the method in file XXXL/src/com/xxxl/action/Register.java. Once such method is called, the connection with MongoDB will be established, and query will be conducted to determine such user whether exists or not. It will return success and redirect into Loginpage if both username and password are matched, otherwise, it will return error.

Upload:
FileUploadAction, defined in the xxxl.action handles the upload and file extract. This class has 5 main private element: **file1**, type file, takes the upload file object in; **file1FileName**, type String, take the extract file name from upload file; **file1ContentType**, type String, defined from the suffix, in this project, the file extract will handle json, xml, csv and other type (including pdf, word, text using external API). The content type element will be used to define which parser will be called to extract the file data; **request**, type HttpServerRequest, which will take the input from the homepage server request when the file upload action is called; **session**, type Map, which will be used to

display the content on the user homepage.

The main method in this class is the **execute** method. The API we used for the upload is apache, so the execute is override here. When the homepage send the upload request, the API will extract the file information and set up the file, filename and type, the execute method will use a simple if-else loop to check the **file1ContentType** element and return the corresponding parser to extract the file.

Parser:
There are four parsers defined in the util: parser for JSON, CSV,XML and other files.

- **JSON parser**: This JSON parser is defined in the userService with a method called extracJSON, which takes 3 input elements: a file type jsonFile, a String type fileName, and a String type username which are generate when the file in load. Jsonfile will first be passed into buffer for reading and passed into the output string, which will then be sent to a method called saveJSON defined in JSONDAO. SaveJSON method takes the json string, userName and fileName to generate a json list. This json list is a linked list which generated by a method called json2Graph. Each element in the list is a json node is defined with 5 variables: **node id**, the current node id is set with the username and file name followed by a index, the root node will have no index. From the root, each node will have a index from 0. **Key,** used with value as key value pair to store the data structure information. **Value,** the corresponding information with the key element. For example, **Desc,** it is a variable to save the node type, the root default is set to be map0, other map will be set with json map, array will set as json array, and other type for example integer is set as the json node. The desc element will be used later when linking the two node, since the node type need to be check to see if two node can be linked together. **LinkedId**, the linkedId is a linked List which store all the linked node id for this node.

  The node structures are same for all the file parser. For example, if we have the following information:
  User1 with t1.json [{"1":{"2":789,"ggg":"a","8":"a"},2:"B"}].
  The root node is defined as:
  id: “User1:ti_json”, key: “t1.json”, value:”content”, dece: file root, linkedId[xxx].
  Initial node is : id: User1:t1_json:0, key: “content”, value: map0, desc: json map0, linkedId[xxx]

  To save the json structure into the MongoDB, the linked List with document type elements is created by using the json graph, then a method called insertMany from MongoDB API is used to send the document to the database system.

- **CSV parser**: A method called **extracCSV** defined in the userService is used to parse the CSV file. Within this method, a **saveCSV** method is called to extract and save the file into database. Also here, the node structure is similar to the json file nodes, a list with JsonNode element is created using a CSV2Graph method to have the same node structure described in the json parser. We uses the apache API for the CSV parser.

- **XML parser**:A method called extracXML in the userService is used to extract XML file. The output node structure is same with json and CSV. The API we used is called dom4j.

- **Other file parser**: For the rest format of file including txt, pdf and etc, we use Tika to parse those data in the file. Each data node is a string or word in the data file. The structure of the output node from here would only have 2 level: a root node and the second level is for each word in the file.

- **Follow:**
  The follower user is what we used to assign the permission for the search result. If two users are following each other, then the search will return the search path based on the data file in those two users account. For example, user1 follow user2 is to give a permission from user1 to user 2.
  To implement the follower, we take a string input from the user for the follower id and save into the database. A **FollowAction** from the webroot will first set up the input string as the follower name and also return the current user name, then invoke a **followUser** in the **UsserService** which insert those two string name into the database. Two string will be passed into a method **insertFollows** in the **UserLoginDAO** which transfer those string to MondoDB document type object and insert into the database.

Search:
The **SearchAction** will be generated from the webroot when the user invoke the search function. It takes the two input: keyword1 and keyword2 and pass the current username and his followers into a method **search** defined in the UserService. The basic algorithm for the search with permission is that first we create a inverse table for all the data node in the data lake. Each data value will have a corresponding id which can be used to find out the user information. For keyword search, when user input 2 keywords, it takes the keyword1 and find all node contains the keyword1 from the inverse table, and for each results, check if the user who owns this data node is the follower from the current user.

If yes, then we defined a method called **findDocs** which will fetch all the data files under the current user and his followers who owns this data node and then do a BFS search with the keyword2, the result will only return the shortest path among those search.

Link
The links for nodes updates when file is uploaded to the database. Each node value will be treat as string and search among all the database data node. Once found same node, the node id will be added into the linkedId list in each other. Here not permission assigned to each node since we use follower as the permission for search node results.

Data Visualization
By implementing open source library Sigma.js, we can display the whole file owned by user or corresponding followers and the connection between these parsed nodes. Besides, we also highlight the path to display our search results.

Validation of Effectiveness
As search for the following two json files:
t1.json:{"1":{"2":789,"ggg":"a","8":"a"},2:"B"}
t2.json:{"a":{"1":"2","3":"t1.json"},"b":[1,789,"abc"]}
To search for the query: “a” and “abc”, the result set would be presented in **milliseconds**:

zhixu->t1.json::map1->8::a||zhixu->t1.json::map0->1::map1||zhixu->t2.json::list0->__content__0::1||zhixu->t2.json::map0->b::list0||zhixu->t2.json::list0->__content__2::abc

zhixu->t1.json::map1->ggg::a||zhixu->t1.json::map0->1::map1||zhixu->t2.json::list0->__content__0::1||zhixu->t2.json::map0->b::list0||zhixu->t2.json::list0->__content__2::abc

which represent the two paths from two nodes containing a to the final destination containing “abc”. “||” separates different nodes. In each nodes, the x->y::a->b::value format has x meaning the user’s name, y meaning file name, a->b meaning key name and value meaning value name.

● Team number contribution

Zhi Xu: Web application, code embedded.
Jiawei Xue:data key word search
Yicheng Lin: data interface with MongoDB
Xin Zhang: data file extraction, parser test and links.

- ## Extra Credit Option

For the first option in the extra credit, we use **lucene library** embedded to increase the search for stemming. It will increase the correctness search result for text or string data. For example, “did”, “done” will be treated as “do” in turn they will be connected in the database.