CS Dept. student job hunting tracking system

1. Team Member

   Haorui Chen (4454226030), Zihao Zhang (4798698599), Zehao Li (3717211170)

2. Responsibilities

   Haorui Chen: system design, front-end implementation, backend backbone implementation, progress monitoring, presentation, screencast
   Zihao Zhang: DB schema design and part 1+2 implementation with, presentation MySQL, screencast
   Zehao Li: DB schema design and part 1+2 implementation with Firebase, presentation, screencast

3. Timeline
   Midterm: complete implementation for Firebase version, part 1 for MySQL version, front-end, sample database, midterm report
   Final: part 2 for MySQL version, screencast, final report

4. Background
   a. Project intuition:
      Many students in CS department are being busy job hunting again. Having a system tracking their job-hunting status can help staffs in the department to know how many students eventually landed on a job, what is the distribution of these students across the specifications, and who are major employers of USC CS students. This may help the department adjust courses/teaching plan for each specification, develop further relationship with certain employers, and aid in ConnectSC permission granting (e.g., students who have been hired cannot apply for positions).
   b. DB conceptual design

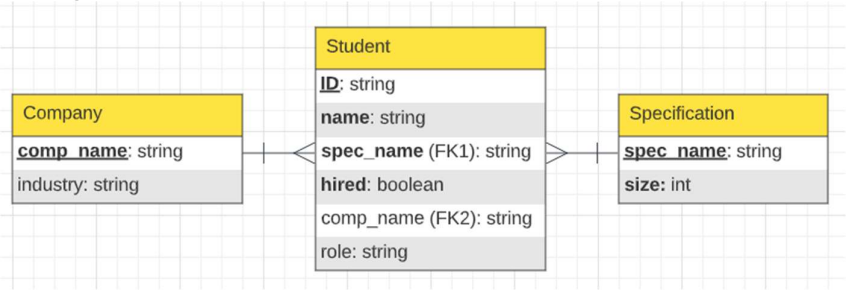
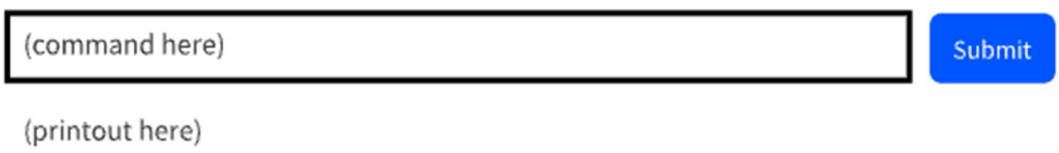


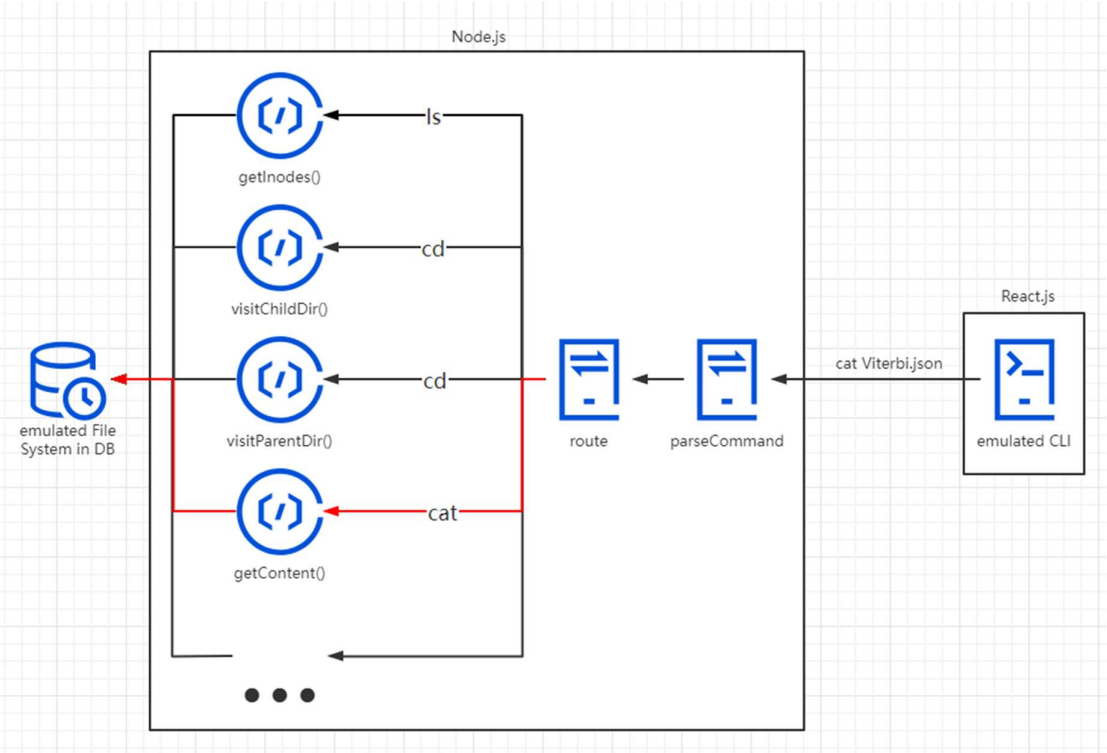
Fig 1. Database E-R diagram (with crow's foot notation) initial design

This DB model is succinct in tracking recording student job hunting status while avoiding gathering redundant information/violating students' privacy.

5. App design
   a. Tech selection
      Framework: React.js+Node.js
      DB: Firebase, MySQL
   b. UI design



   Idea: All commands in part 1 can be typed directly in input box, commands for search and explanation functionalities need to be defined, but can also be typed here.
   c. Command routing



d. Command implementation

i. MySQL

1. mkdir path
   Check the correctness of each path node
   Find the direct parent of the directory to build
   Add a new line in table “Meta data table”, initiate an id and directory name
   Add a new line in table “Directory structure table”, bound the new directory with its direct parent
2. ls path
   Check the correctness of each path node
   Traverse table “Directory structure table”, and output all file names with their direct parent directories same as the given one.
3. rm path
   Decide whether it’s a file or a directory
   If file, find its id
   ->drop all its data partition table with info from “Partition info table”
   ->delete its info in table “Partition info table”,” Directory structure table” and “Meta data table”
   If directory, find its id
   ->recursively delete all its child data node
   ->delete its info in table “Directory structure table” and “Meta data table”
4. put path
   Check the correctness of each path node
   Create a record in table “Meta data table”, initiate a new file id
   Create a new line in Directory structure table, to bound it with its parent directory
   Based on its data feature, apply appropriate range to partition the file, and record the partition info in Partition info table
   Fulfill each partition node with data
5. getPartitionLocations(file)
   Search in Partition info table to obtain all partition paths for the file
6. readPartition(file, partition#)

Search in Partition info table to get exact partition path, then traverse the according data table to obtain actual data.

ii. Firebase

1. def mkdir(self, path):
    a. first check if parent directory exists
    b. then check if the new directory exists
    c. if both do not exist, create a new json file with a default key and value in the firebase.
2. def ls(self, path):
    a. first check if the path exists
    b. if exists, list out all the keys of the current json file
3. def cat(self, path):
    a. first check if the path exists
    b. if exists, list out all the keys of the current json file
4. def rm(self, path):
    a. first check if the path of the file exists
    b. if exists, replace the contents of the json file with en empty dictionary and then the file will be deleted automatically.
5. def put(self, file, path, k):
    a. first check the parent path exists
    b. create k partitions
    c. read csv file into k partitions
    d. put csv file
    e. save file path in a dictionary for future use
6. def getPartitionLocations(self, file):
    a. first check if the file in the file dictionary
    b. if exists, return the file path
    c. else, return "file not found"
7. def readPartition(self, file, partitionNum):
    a. first check if the file in the file dictionary
    b. if exists, find the path of the file with partitionNum
    c. then list out all the keys of the partition json file

e. MapReduce Implementation

i. MySQL

1. Since our web application is about new graduate students' job hunting process tracking, I believe it is a fair idea to hash all records with the name of company involved. And for user table, it's user's name.
2. For search or analytics functions, our EDFS would first call getPartitionLocations(file) to obtain info for all partition belongs to this file. Then, it will call readPartition() to read each partition and filter the lines that meet the searching requirements.
3. With all the records collected from all partitions of the target file, the EDFS does combine function to build the result for output.

ii. Firebase

def mapPartition(self, p):

1. first use getPartitionLocations() to find the path of the file
2. then use readPartition() to find contents of each partitions
3. read the while file with contents and filer out the data we need

f. Database schema

i. MySQL

We are going to emulate the DFS with MySQL tables. There would be tables as follows:

1. Meta data table, recording id, file name, build time, etc., for each file, with id as primary key.

2. Directory structure table, recording parent documentary of each file, including current file id, parent documentary id, with current file id as primary key.
3. Partition info table, recording file id, partition id, location of actual data for each partition,
4. Data table, recording actual data in lines

ii. FireBase

```
"root": {
    "students": {
        "John": {
            "student_id.json": {
                "p1": < location1 > ,
                "p2": < location2 >
            }
            "applied_company.json": {
                "p1": < location1 > ,
                "p2": < location2 >
            }
            "result.json": {
                "p1": < location1 > ,
                "p2": < location2 >
            }
        }
    }
}
```