Final Report

Designing Emulated Distributed File System for exploration of Outbreaks: COVID-19, Ebola, and SARS

Team Members: Tushar Sharma, Laxmi Garde, Raajitha Rajkumar

A. Project Overview

The project provides a study and analysis of impacts of COVID-19, SARS and Ebola outbreaks. The study examines the available datasets and provides relationships between the deaths, geography, recovered cases, and total infected people due to different diseases. The project is based on an emulated distributed file system implemented using Firebase and MySQL, and uses the file system operations implemented and provides these features to the end users.

B. Project Design / Emulated Partition Systems Implementations

The implementation of all operations mentioned in Task 1, Task 2 and Task 3 of the Project have been completed successfully. Details on the Project timelines are mentioned in the Project Tracker (Section C). The implementation details include:

B.1 MySQL Implementation

For our MySQL implementation, we are using two tables created with the following structure:

fid	name	parent	parentid	content	file
1	root		0	/	

pid	fid	tableName	partitionName	part
…	…	…	…	…

- ***mkdir()***

Creating a new directory is very simple as we just insert a new row into the directory table after gathering all the column information. After parsing through user input and tracking parent information and determining content information, we just insert it.

```
sql = "INSERT INTO dir (name, parent, parentid, content, file) VALUES (%s, %s, %s, %s, %s)"
val = (name, parent, int(parentid), content, file)
```

- ***ls()***

We are able to grab the children files from a directory by simply tracking the parentid grabbed from the file id where the user put in the directory name.

```
cursor.execute("SELECT fid FROM dir WHERE content LIKE '"+ dirname +"'")
result = cursor.fetchall()
dirid = result[0][0]
sql = "SELECT name FROM dir WHERE parentid = " + str(dirid)
```

- ***rm()***

For rm() we use a simple delete statement to delete the row from the dir table

```
sql = "DELETE FROM dir WHERE content LIKE '" + filename + "'";
```

- ***put()***

1. Take filename and create a tableName for new table
2. Create a new row in dir table with filename and tableName
3. Create a new table for file contents (.csv, .txt) and assign partition number
4. Insert values into table
5. For each partition, create a new table and insert a new partition location in the partition table

```
# create partition tables
sql = "SELECT fid FROM dir WHERE content LIKE '" + content + "'"
cursor.execute(sql)

fid = cursor.fetchall()

for x in range(k):
    partitionName = tableName + "_" + str(x)
    cursor.execute("CREATE TABLE " + partitionName + " SELECT * FROM " + tableName +" PARTITION(p" + str(x) + ")")
    sql = "INSERT INTO partitions (file_id, tableName, partition_name, part) VALUES (" + str(fid[-1][0])
    + ", '" + tableName + "','" + partitionName + "', " + str(x+1) + ")"
    cursor.execute(sql)
    db.commit()
```

Results of put():

```
: put("archive/day_wise.csv", "/user", 3)
```

```
processing table into directory...
done
```

```
: dir()
```

	fid	name	parent	parentid	content	file
0	1	root		0	/	
1	2	user	root	1	/user	
2	4	day_wise.csv	user	2	/user/day_wise.csv	day_wise

```
: partitions()
```

	pid	file_id	tableName	partition_name	part
0	1	4	day_wise	day_wise_0	1
1	2	4	day_wise	day_wise_1	2
2	3	4	day_wise	day_wise_2	3

- ***cat()***

For cat(), we use a simple select statement to output the table

- ***getPartitionLocations()***

To get the location of the partitions in the EDFS, we simple map to their IDs in their table

- ***readPartitions()***

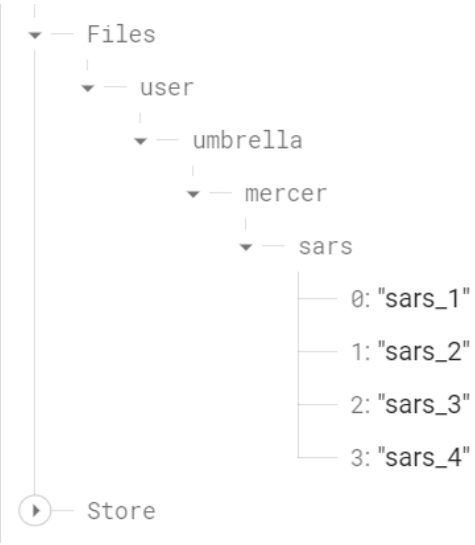
To read a partition, we map to the partition table created in put, and map to the part specified by the user

B.2 Firebase Implementation

For our Firebase implementation, we are using 3 cards:

1. **Dir** -> Directory tree , stores children of a node and its content structure

2. **Files** -> Metadata tree stores the files and their partition names



3. **Store** -> Location where the partitioned file contents are stored

The above 3 cards and their structure is essential for the following commands to run properly.

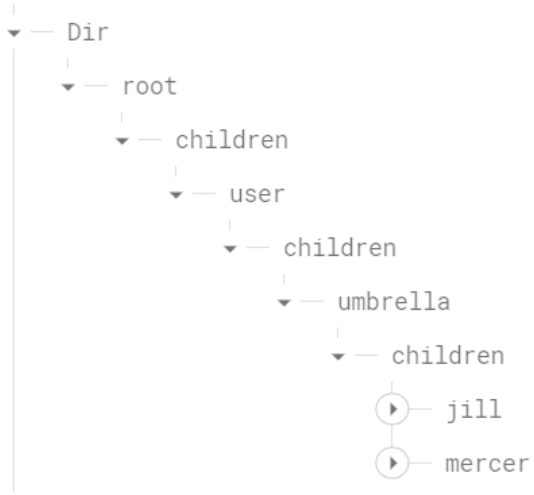
- ***mkdir()***

Creating a new directory involves first querying the Database for a pre-existing directory. If the directory is new and does not exist, the function proceeds to create the whole directory tree, including and new components.

For eg : /user/umbrella/mercer as an input would create all directories including user, umbrella even if they did not exist. (placeholder acts as an identity file allowing firebase to actually create directory structure. This cannot be done if the card does not have any content)

```
if(verify_mkdir(url)):
    req = requests.patch(url,data= json.dumps(placeholder))
    if(req.status_code == 200):
        print(req.json())
```

```
D:\MS\DSCI-551\HW\Project>python tushar_prjkt.py /user/umbrella/jill sars.csv
https://dsci-project-28e72-default-rtdb.firebaseio.com/Dir/root/children/user/children/umbrella/children/jill/.json
{'content': ['.'], 'children': {}}
None
No preexisting repo here
{'content': ['.'], 'children': None}
https://dsci-project-28e72-default-rtdb.firebaseio.com/Dir/root/children/user/children/umbrella/children/jill/.json
```



Result of mkdir

- ***ls()***

Using the directory structure we first need to modify the input to match our directory structure with '/children/' for each directory giving access to its children.

Once done we can list the contents of any directory.

For empty directories we display a message stating the current directory has no children.

```
D:\MS\DSCI-551\HW\Project>python tushar_prjkt.py /user/umbrella/jill sars.csv
https://dsci-project-28e72-default-rtdb.firebaseio.com/Dir/root/children/user/children/umbrella/children/jill/.json
{'content': ['.'], 'children': {}}
None
No preexisting repo here
{'content': ['.'], 'children': None}
https://dsci-project-28e72-default-rtdb.firebaseio.com/Dir/root/children/user/children/umbrella/children/jill/.json
No children found for directory::user/children/umbrella/children/jill
```

```
D:\MS\DSCI-551\HW\Project>python tushar_prjkt.py /user/umbrella
{'directories': ['jill', 'mercer']}
```

```
D:\MS\DSCI-551\HW\Project>python tushar_prjkt.py /user/umbrella/mercer
{'files': ['.', 'sars.csv']}
```

- ***rm()***

For rm we have to do a 3 pronged approach and delete the file from 3 places:

a. Dir structure from content list of the path card
b. Files structure to remove the partitions
c. Stores structure to remove the actual partitions

- ***put()***

Put is one of the more complex implementations, it has the following steps:

1. Split file into given number of partitions:

```
data = read(filepath)
print('TOTAL FILE SIZE ::'+str(len(data)))
part_size = int(len(data)/numpart)
print('PART SIZE::'+str(part_size))
files = []
for idx in range(numpart):
    print(idx)
    start = idx * part_size
    stop = start + part_size
    if idx == numpart-1:
        stop = len(data)
    print('START IDX ::'+str(start))
    print('STOP IDX ::'+str(stop))
    files.append(json.dumps(data[start : stop]))
#print(len(files))
```

2. Insert file into the Dir Structure:

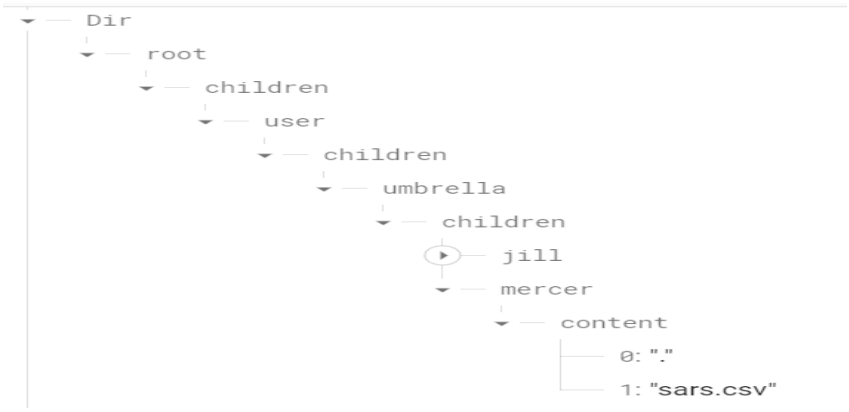
```
def insert_file_name_in_dir(file,path:str):
    path_parts = path_preprocess(path)
    filename  = file
    data,url_update = fetch_files(path_parts,filename)
    if 'content' in data.keys():
        print('Content found in file')
        if filename in data['content']:
            print("FILE ALREADY EXISTS!!")
            return
        data['content'].append(filename)
    else:
        print('No content child found')
        data['content'] = [filename]
    patch(url_update,filename,data)
```

3. Insert file partition names into the Files Structure:

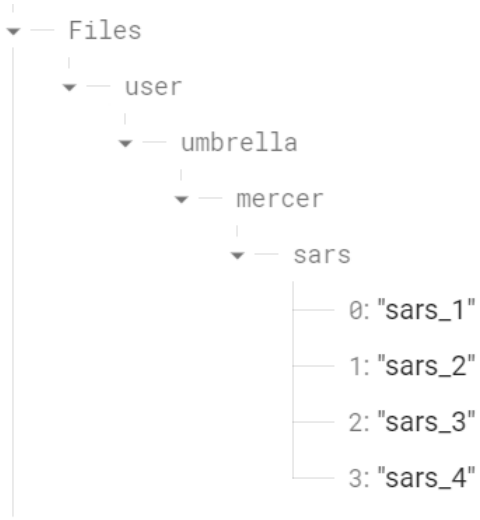
```
def insert_file_partition_names_in_meta(file_parts:list,dirpath:str,filen
    parts = []
    for idx in range(len(file_parts)):
        name = filename.split('.')[0]+'_'+str(idx+1)
        parts.append(name)
    files_parts_url = files_base_url+base_end
    payload = dict()
    payload[dirpath] = parts
    res = patch(files_parts_url,"test",data = payload)
    if res != None:
        return parts
```

4. Insert data into store with partitions:

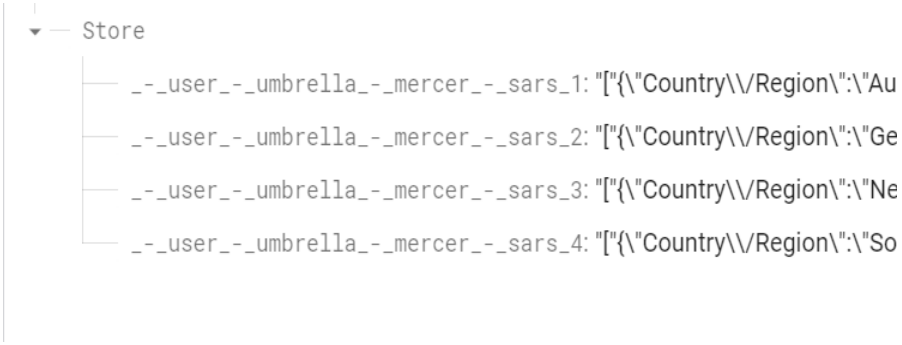
```
def store_parts(file_parts:list,dirpath:str,filename:str):
    parts = []
    dirpath = dirpath.replace('/', '_-_')
    path = store_base_url +'/'+ base_end
    for index,file in enumerate(file_parts):
        name = filename.split('.')[0]+'_'+str(index+1)
        name = dirpath+'_-_'+name
        data = dict()
        data[name] = file_parts[index]
        res = patch(path,dirpath, data=data)
```



Directory Structure after put request



Files Structure after put request



Store Structure after put request

- ***getPartitionLocations()***

We simply access the two Cards Files and Store to get the Partition Locations.

```
def getPartitionLocations(file : str):
    filename = file.split('/')[-1].split('.')[0]
    dirPath = file.removesuffix('/'+filename+'.csv')
    #print(dirPath)
    data = ls(dirPath)
    #print(data)
    locs = None
    if "files" in data and filename+'.csv' in data['files']:
        #now return the partitions
        partsNames = getParts(filename,dirPath)
        locs = generateLocs(dirPath,partsNames)
    return locs
```

```
D:\MS\DSCI-551\HW\Project>python tushar_prjkt.py /user/umbrella/mercer sars.csv
{'files': ['.', 'sars.csv']}
{'sars_1': 'Store/user/umbrella/mercer/sars_1', 'sars_2': 'Store/user/umbrella/mercer/sars_2', 'sars_3': 'Store/user/umb
rella/mercer/sars_3', 'sars_4': 'Store/user/umbrella/mercer/sars_4'}
```

- ***readPartitions()***

For reading a numbered partition of the given file, we first ensure that the file exists. Then we use the generated getPartitionLocations() to fetch data from the location.

```
def readPartition(file,partNumber):
    locs = getPartitionLocations(file)
    filename = file.split('/')[-1].split('.')[0]
     (variable) dirPath: Any      ix(filename+'.csv')
    dirPath = dirPath.replace('/', '_-_')
    path = store_base_url +'/'+dirPath +filename + '_' +str(partNumber)
    get_req = requests.get(path)
    return get_req.json()
```

- ***cat()***

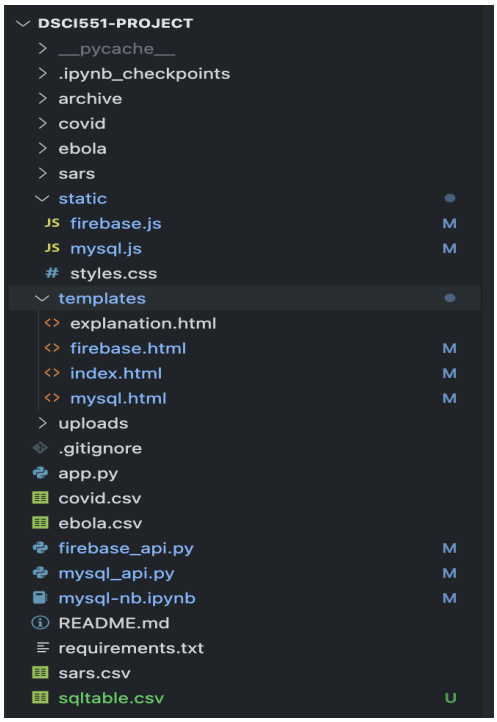
Cat uses the existing function of getPartitionLocations and readPartitions to reconstruct the complete file and return it.

```
D:\MS\DSCI-551\HW\Project>python tushar_prjkt.py /user/umbrella/mercer/sars.csv
["{\"Country\\/Region\":\"Australia\",\"Cumulative male cases\":4,\"Cumulative female cases\":2,\"Cumulative total cases
\":6,\"No. of deaths\":0,\"Case fatalities ratio (%)\":0,\"Date onset first probable case\":\"2003-02-26\",\"Date onset
last probable case\":\"2003-04-01\",\"Median age\":15.0,\"Age range\":\"1-45\",\"Number of Imported cases\":6.0,\"Percen
tage of Imported cases\":100.0,\"Number of HCW affected\":0,\"Percentage of HCW affected\":0}", "{\"Country\\/Region\":\
"Canada\",\"Cumulative male cases\":151,\"Cumulative female cases\":100,\"Cumulative total cases\":251,\"No. of deaths\"
:43,\"Case fatalities ratio (%)\":17,\"Date onset first probable case\":\"2003-02-23\",\"Date onset last probable case\"
:\"2003-06-12\",\"Median age\":49.0,\"Age range\":\"1-98\",\"Number of Imported cases\":5.0,\"Percentage of Imported cas
es\":2.0,\"Number of HCW affected\":109,\"Percentage of HCW affected\":43}", "{\"Country\\/Region\":\"China\",\"Cumulati
ve male cases\":2674,\"Cumulative female cases\":2607,\"Cumulative total cases\":5327,\"No. of deaths\":349,\"Case fatal
ities ratio (%)\":7,\"Date onset first probable case\":\"2002-11-16\",\"Date onset last probable case\":\"2003-06-03\",\
"Median age\":null,\"Age range\":null,\"Number of Imported cases\":null,\"Percentage of Imported cases\":null,\"Number o
f HCW affected\":1002,\"Percentage of HCW affected\":19}", "{\"Country\\/Region\":\"Hong Kong SAR, China\",\"Cumulative
male cases\":977,\"Cumulative female cases\":778,\"Cumulative total cases\":1755,\"No. of deaths\":299,\"Case fatalities
 ratio (%)\":17,\"Date onset first probable case\":\"2003-02-15\",\"Date onset last probable case\":\"2003-05-31\",\"Med
ian age\":40.0,\"Age range\":\"0-100\",\"Number of Imported cases\":null,\"Percentage of Imported cases\":null,\"Number
of HCW affected\":386,\"Percentage of HCW affected\":22}", "{\"Country\\/Region\":\"Macao SAR, China\",\"Cumulative male
 cases\":0,\"Cumulative female cases\":1,\"Cumulative total cases\":1,\"No. of deaths\":0,\"Case fatalities ratio (%)\":
0,\"Date onset first probable case\":\"2003-05-05\",\"Date onset last probable case\":\"2003-05-05\",\"Median age\":28.0
,\"Age range\":null,\"Number of Imported cases\":1.0,\"Percentage of Imported cases\":100.0,\"Number of HCW affected\":0
,\"Percentage of HCW affected\":0}", "{\"Country\\/Region\":\"Taiwan\",\"Cumulative male cases\":218,\"Cumulative female
 cases\":128,\"Cumulative total cases\":346,\"No. of deaths\":37,\"Case fatalities ratio (%)\":11,\"Date onset first pro
bable case\":\"2003-02-25\",\"Date onset last probable case\":\"2003-06-15\",\"Median age\":42.0,\"Age range\":\"0-93\",
\"Number of Imported cases\":21.0,\"Percentage of Imported cases\":6.0,\"Number of HCW affected\":68,\"Percentage of HCW
 affected\":20}", "{\"Country\\/Region\":\"France\",\"Cumulative male cases\":1,\"Cumulative female cases\":6,\"Cumulati
ve total cases\":7,\"No. of deaths\":1,\"Case fatalities ratio (%)\":14,\"Date onset first probable case\":\"2003-03-21\
",\"Date onset last probable case\":\"2003-05-03\",\"Median age\":49.0,\"Age range\":null,\"Number of Imported cases\":7
.0,\"Percentage of Imported cases\":100.0,\"Number of HCW affected\":2,\"Percentage of HCW affected\":29}"]["{\"Country\
```


B.3 Application User Interface Implementation

For the application's user interface, we used Python3, HTML5/CSS3, Javascript, JQuery, Bootstrap libraries, Axios, and the application uses the Flask web framework as a local development server.

The file structure for the project follows the same structure as specified in the Flask documentation. The Flask web application project structure is as follows:



Flask Project Structure

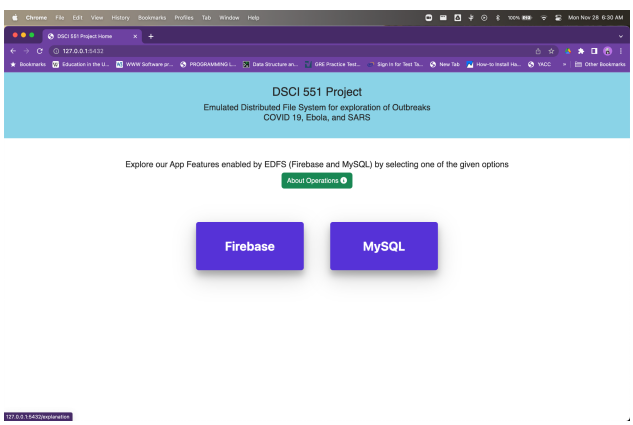
The ***static*** folder contains the static items that are served by the server. The firebase.js and mysql.js files contain the Javascript, JQuery implementations that render and update the HTML based on the API responses. The styles.css provides the CSS styling template for the project. The ***templates*** folder contains index.html, firebase.html, mysql.html, and explanation.html code for front-end. The ***app.py*** file creates the Flask application instance as it imports the Flask objects and registers all the front-end paths. The firebase_api.py and mysql_api.py files are the backend code of the application, and they have the REST APIs for each feature implementation.

To start the Flask application instance the following commands are required to be executed on the command line:

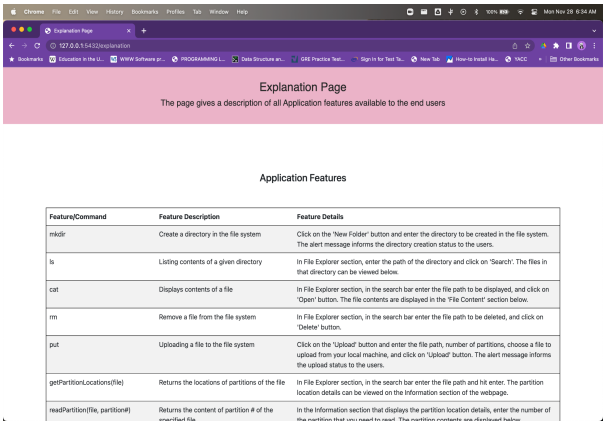
```
dsci551-project git:(integration-frontend) ✗ export FLASK_APP=app.py
dsci551-project git:(integration-frontend) ✗ flask --app app.py --debug run --port 5432
```

```
 * Serving Flask app 'app.py'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5432
Press CTRL+C to quit
 * Restarting with stat
```

The application can be accessed on URL : http://127.0.0.0.1:5432
The URL takes the users to the Project Home page which contains two buttons, which allows users to click on the specific EDFS implementation.
The Home page also contains a 'About Operations' button which redirects the user to the explanation page. The explanation page gives the list of features provided by the application.

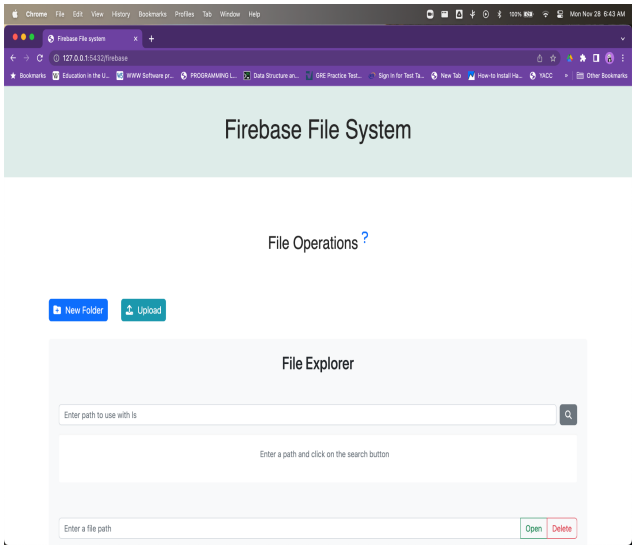


Project Home Page

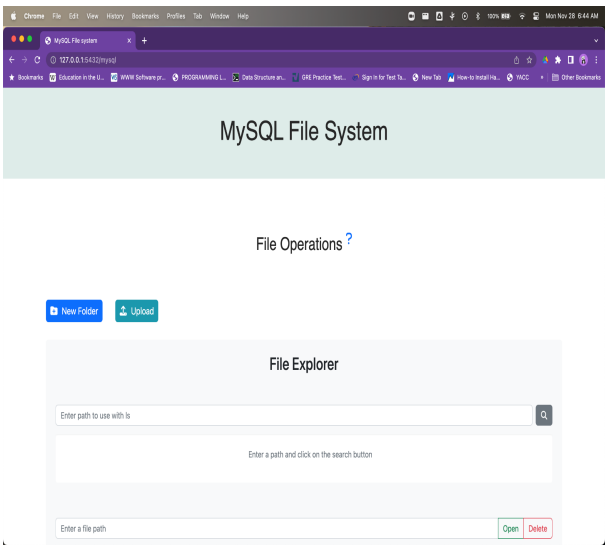


Explanation Page

By clicking on the Firebase or MySQL button on the Project Home page, the user can go to one of the implementations.



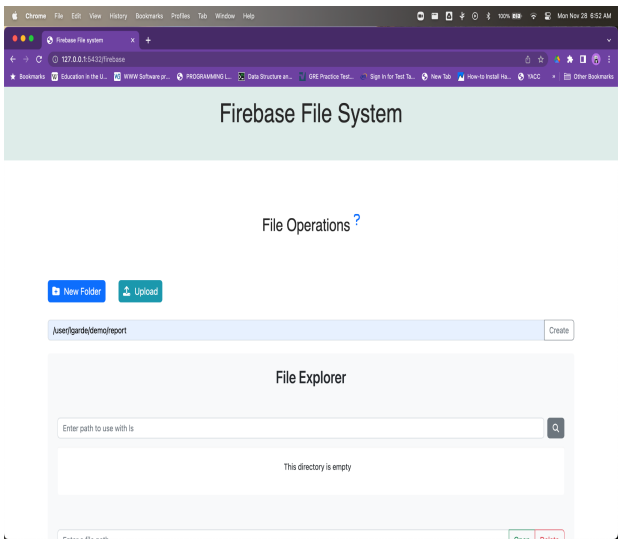
Firebase Operations Page



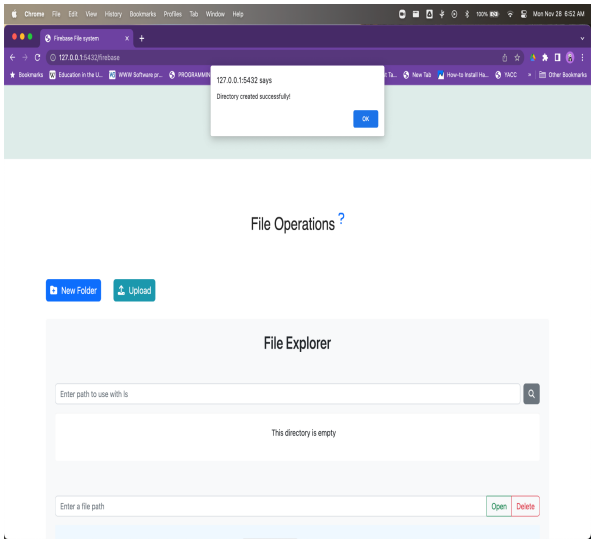
MySQL Operations Page

- ***mkdir()***

mkdir on the front-end is given by the 'New Folder' button and by clicking on it, an input field appears where the path of the new directory to be created is specified. After that by clicking on the 'Create' button, the directory is created. The button click internally calls the Firebase (or MySQL) /mkdir REST API which takes 'path' as a parameter, backend processes the request and the API response in JSON format is parsed and rendered on the interface.



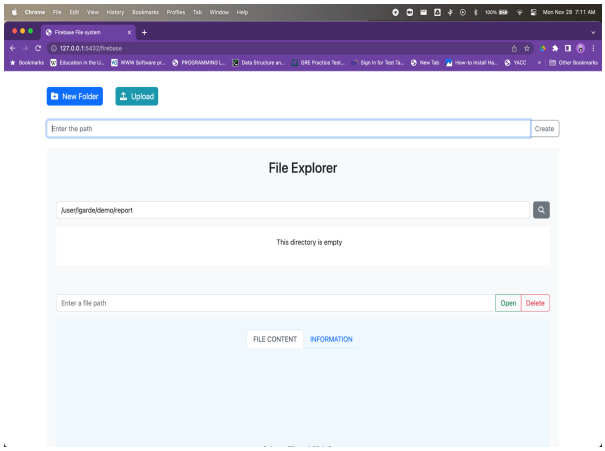
Mkdir directory path entry



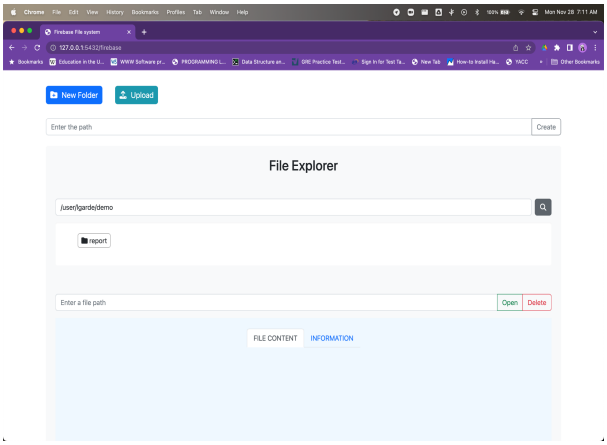
Directory created successfully

- ***ls()***

The ls command can be executed by checking in the File explorer section, by entering the file system path for which the user wants to check the contents for. By clicking on the search icon, the contents of the file are displayed in the section below. If the directory is present, 'This directory is empty' message is displayed.



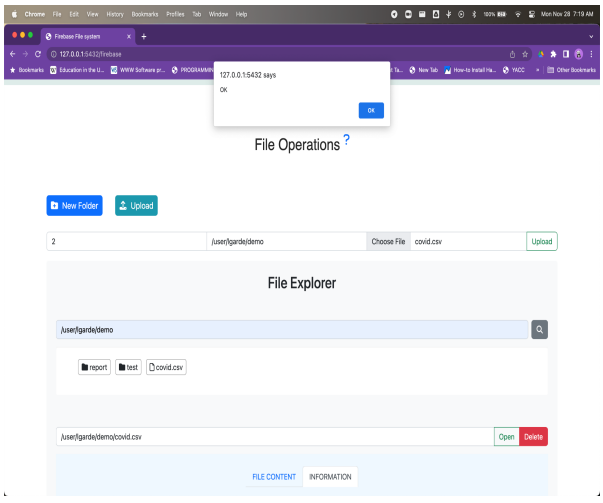
Empty Directory



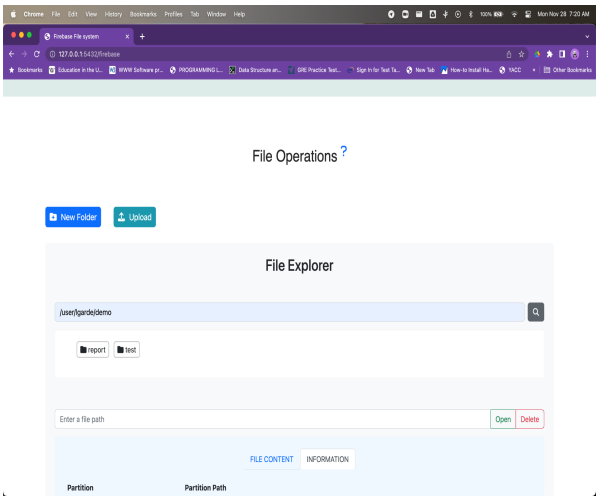
Directory contents displayed

- ***rm()***

The rm command can be executed using the 'Delete' button in the FIle Explorer section of the interface. The parameter the /rm API accepts is the filepath. By clicking on the Delete button, if the delete file operation is successful, the alert box is displayed. After deleting the file, check if the file is deleted by using the ls command.



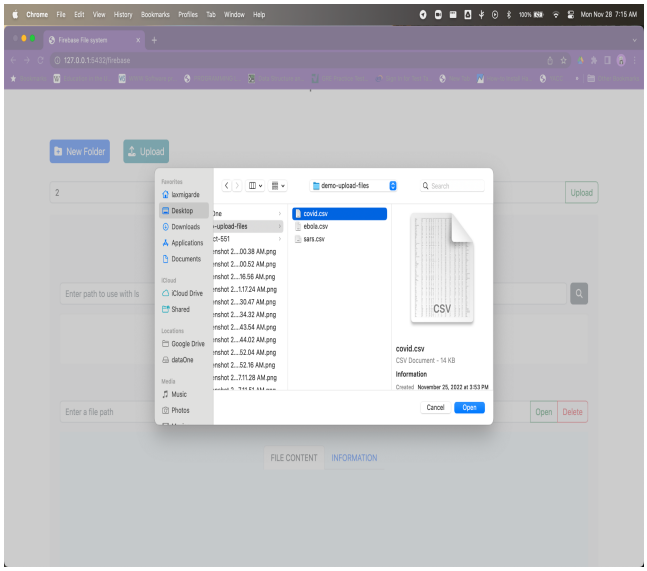
File deleted successfully



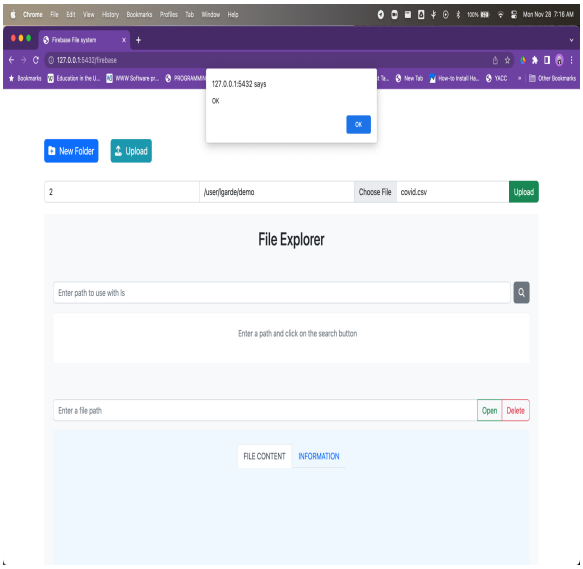
Verifying by using ls command

- ***put()***

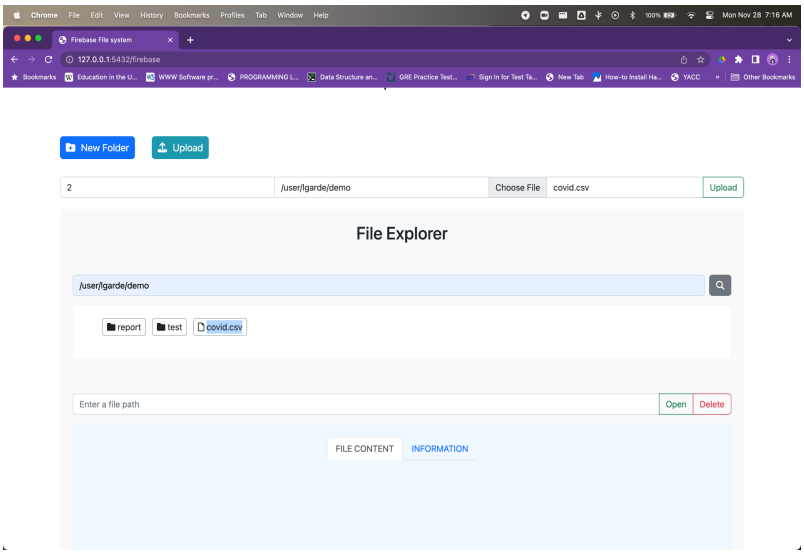
The put command can be used by the user by clicking on the ‘Upload’ button. The number of partitions, file path, and selecting a file from the local system to post the contents is performed by clicking on Upload. Once the upload is successful, alert message is displayed.



Upload file from local



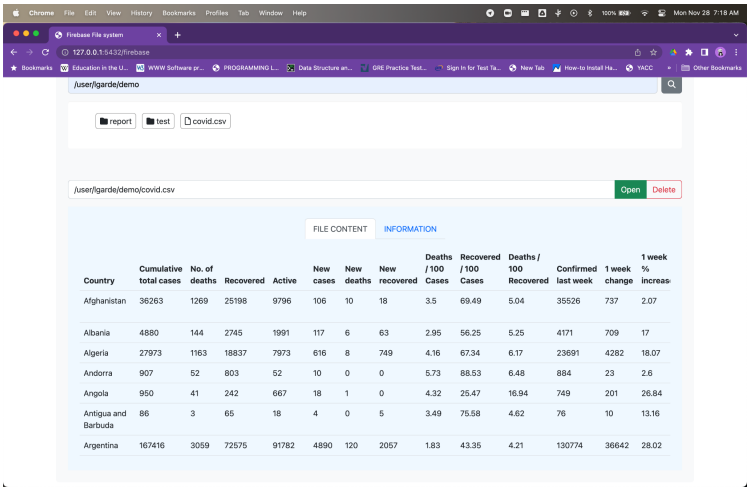
Upload done successfully



Uploaded file checked using ls command

- ***cat()***

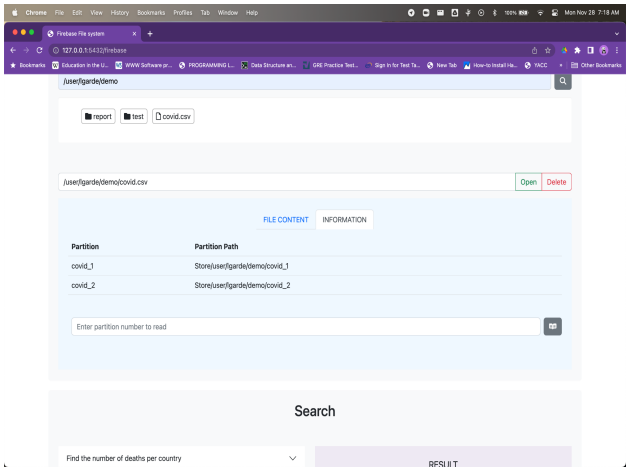
cat command can be executed by the user by clicking on the ‘Open’ button present. The output of the cat command is displayed in the ‘FILE CONTENT’ tab below.



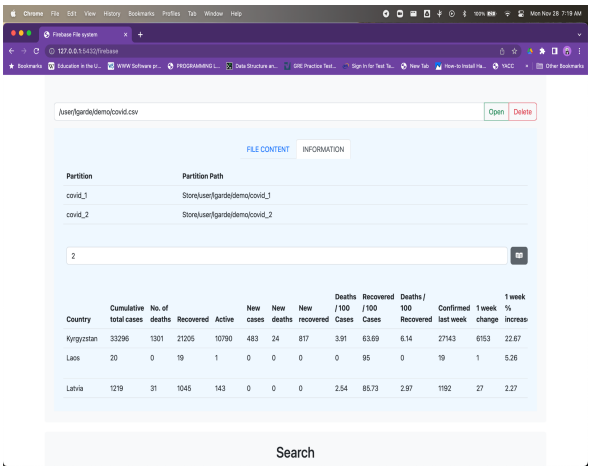
cat command output

- ***getPartitionLocations() and readPartition()***

The output for these two commands can be viewed by the user in the ‘Information’ tab. To view the contents of a particular partition, enter the partition number and click on the search icon. The contents of the partition would be displayed below in tabular format.



getPartitionLocation() output



readPartition() output

The search and analysis interface implementation details are present in section C. Both MySQL and Firebase pages have ‘?’ symbols on top for users to access the Explanation Page.

B.4 Project Integration APIs

BASE_URL= http://127.0.0.1:5432/

Function	Endpoint	Method	Query Parameter	Sample Request URL	Sample JSON Responses
MySQL mkdir	/mysql/mkdir	GET	path	BASE_URL + mysql/mkdir?path=/demo	{ "status": "Directory created successfully!" }
Firebase mkdir	/firebase/mkdir			BASE_URL + firebase/mkdir?path=/demo	
MySQL ls	/mysql/ls	GET	path	BASE_URL + mysql/ls?path=/covid	{ "directories": [], "files":["covid_19_clean_complete.csv"] }
Firebase ls	/firebase/ls			BASE_URL + firebase/ls?path=/covid	
MySQL cat	/mysql/cat	GET	file	BASE_URL + mysql/cat?file=/user/test/sars.csv	["{\"Country\":\"Australia\",\"Cumulative male cases\":4,\"Cumulative female cases\":2,\"Cumulative total cases\":6,\"No. of deaths\":0,\"Case fatalities ratio (%)\":0,\"Date onset first probable case\":\"2\\/26\\/2003\",\",\"Median age\":15.0,\"Age range\":\"Jan-45\",\"Number of Imported cases\":6.0}"]
Firebase cat	/firebase/cat			BASE_URL + firebase/cat?file=/user/test/sars.csv	
MySQL rm	/mysql/rm	GET	file	BASE_URL + mysql/rm?file=/user/test/sars.csv	{"status": "OK"}
Firebase rm	/firebase/rm			BASE_URL + firebase/rm?file=/user/test/sars.csv	
MySQL put	/mysql/put	POST	file, dirPath, numPart	BASE_URL + mysql/put?dirPath=/user/test&numPart=2 Body: file: sars.csv	{"status": "OK"}
Firebase put	/firebase/put			BASE_URL + firebase/put?dirPath=/user/test&numPart=2	

				Body: file: sars.csv	
MySQL getPartitionLocations	/mysql/getPartitionLocations	GET	file	BASE_URL + mysql/getPartitionLocations?file=/user/test/ebola.csv	**{ "ebola_1": "Store/user/test/ebola_1", "ebola_2": "Store/user/test/ebola_2" }**
Firebase getPartitionLocations	/firebase/getPartitionLocations			BASE_URL + firebase/getPartitionLocations?file=/user/test/ebola.csv	
MySQL readPartition	/mysql/readPartition	GET	file, partNumber	BASE_URL + mysql/readPartition?file=/user/test/ebola.csv&partNumber=2	**["{\"Country\":\"Italy\",\"Date\":\"6\\/22\\/2015\",\"Cumulative total cases\":1.0,\"No. of deaths\":0}"]**
Firebase readPartition	/firebase/readPartition			BASE_URL + firebase/readPartition?file=/user/test/ebola.csv&partNumber=2	
MySQL search function 1	/mysql/countrydeathcount	GET	dataset, country	BASE_URL + mysql/countrydeathcount?dataset=covid&country=India	**{"mapper": ["{}", "{\"Country\":{\"0\":\"India\"},\"No. of deaths\":{\"0\":33408},\"Cumulative total cases\":{\"0\":1480073}}", "{}", "{}"], "reducer": "{\"Country\":{\"0\":\"India\"},\"No. of deaths\":{\"0\":33408},\"Cumulative total cases\":{\"0\":1480073}}"}**
Firebase search function 1	/firebase/countrydeathcount			BASE_URL + firebase/countrydeathcount?dataset=covid&country=India	
MySQL search function 2	/mysql/findcountriesbetween	GET	dataset, limit1, limit2	BASE_URL + mysql/findcountriesbetween?dataset=covid&limit1=1000&limit2=5000	**Outputs too large.**
Firebase search function 2	/firebase/findcountriesbetween			BASE_URL + firebase/findcountriesbetween?dataset=covid&limit1=1000&limit2=5000	
MySQL analysis functions	/mysql/analysisdeathpercountry /mysql/analysisrecovery	GET	-	BASE_URL + /mysql/analysisdeathpercountry BASE_URL + mysql/analysisrecovery	**Outputs too large.**
Firebase analysis functions	/firebase/analysisdeathpercountry /firebase/analysisrecovery			BASE_URL + firebase/analysisdeathpercountry BASE_URL + firebase/analysisrecovery	

C. Search and Analytics Functions

Search Functions:

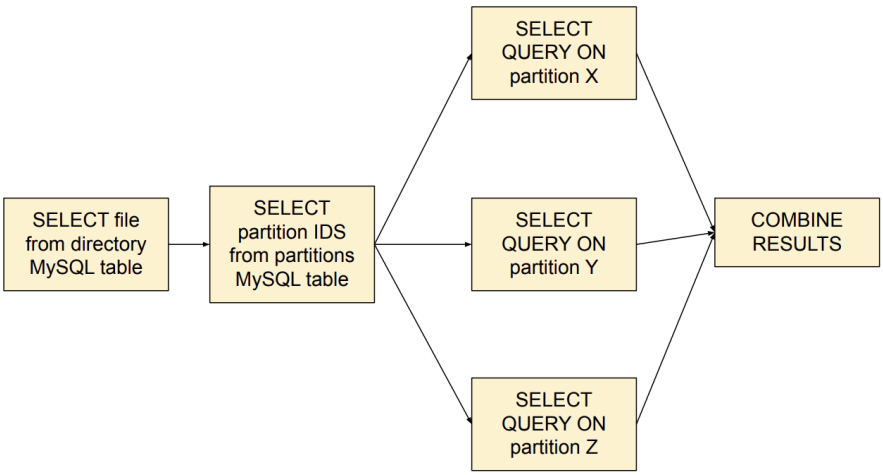
1. **Death count for a specific disease (of 3) for a specific country.**
2. **Find countries with a death count between two numbers for a specific disease.**

Analytics Functions:

1. **Analyze the count of deaths per country for each disease**
2. **Analyze the number of people who recovered after being affected by the disease for each disease.**

C1. MySQL Implementation

For the MySQL implementation for partition-based map and reduce on data stored in our EFDS, we performed our following search and analytic functions on our dataset partitions and combined the results.



Above is my flow of how I coded the search and analytic functions for MySQL. As you can see, I go through all the tables in my database to get to my final answer.

Search Functions:

1. parameters(disease, country)

For example, if a user puts in the country “Canada” and the disease “sars” then we can look through the sars partition ids, query all the death counts from canada and combine them easily in python.

```
search_deathsByCountryAndDataset("Canada", "sars")
```

```
Getting partition ids for 'sars_2003_complete_dataset_clean.csv'...

(6,)
(7,)
(8,)
(9,)
(10,)

mapper:

  country  deaths
0  Canada     346
0  Canada     363
0  Canada     345
0  Canada     456
0  Canada     503

reducer:

  country  deaths
0  Canada    2013


Total Deaths for Canada: 2013
```

You can see in the example above, there are 5 partitions and 5 death counts mapped and reduced to a total death count.

2. parameters(disease, upper/lower thresholds)
   Following the same workflow, you can see that there is only 1 partition for covid, and for all the sums of deaths per country, it gets filtered between the upper and lower limits decided by the user. The example here uses covid with limits of 2000 and 5000.

```
countries_casesBetweenXY("covid", 2000, 5000)
```

```
Getting partition ids for 'covid/covid_19_clean_complete.csv'...

(1,)
                country      cases
0           Afghanistan  1936390.0
1               Algeria  1179755.0
2               Andorra    94404.0
3                Angola    22662.0
4    Antigua and Barbuda     4487.0
..                   ...        ...
182               Yemen    67180.0
183             Comoros    15823.0
184          Tajikistan   383026.0
185             Lesotho     6794.0
186             Albania   196702.0

[187 rows x 2 columns]


 Countries with cases between 2000 and 5000
2000
                              country   cases
5                 Antigua and Barbuda  4487.0
17                             Belize  2636.0
19                             Bhutan  4971.0
49                           Dominica  2059.0
59                               Fiji  2266.0
63                             Gambia  4845.0
69                            Grenada  2466.0
94                               Laos  2229.0
141                       Saint Lucia  2236.0
142  Saint Vincent and the Grenadines  2771.0
148                       Seychelles  3977.0
168                      Timor-Leste  2487.0
```

Analytics Functions:

1. In order to find the total deaths per country across all datasets, we found the total deaths per country for each dataset, and combined all three results of each dataset. Here is an example of a SQL query we used…
   "SELECT CountryRegion, SUM(Deaths)
   FROM partition
   GROUP BY CountryRegion"
2. In order to find the average number of people recovered from all datasets, we query all partitions in such a way…
   "SELECT AVG(Confirmed), AVG(Recovered)
   FROM partition"
   We used the average number of confirmed cases as a way to analyze the data. We take all the averages and combine them to make a total average across all datasets to get an output like this:

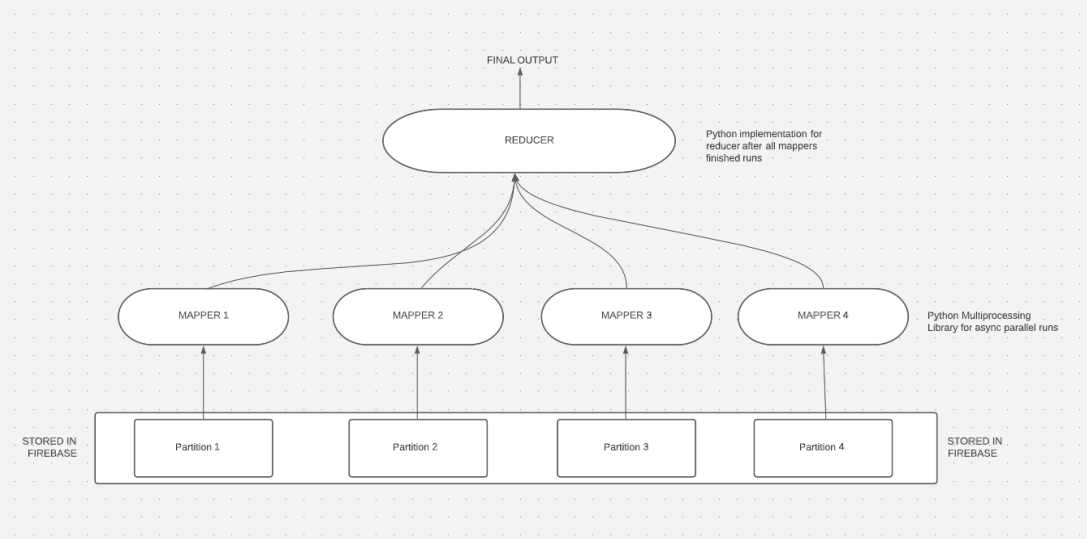
```
averageNumberofRecoveredPerOutbreak()
```

```
{'mapper': {'sars': [[175, 275], [107, 154], [142, 264], [100, 177], [154, 238]], 'ebola': [[299, 2863], [205, 2863],
[184, 2861], [171, 2863]], 'covid': [[7915, 16885]]}, 'reducer': {'sars': [136, 222], 'ebola': [215, 2863], 'covid':
[7915, 16885]}}
```

C2. Firebase Implementation

The general implementation involves creating a mapper and a reducer function for each specific search and analysis function.

The workflow needs to emulate a hadoop implementation and as such needs to run mappers asynchronously. For this I opted for the multiprocessing library of python and its associated thread pools to run multiple jobs asynchronously for different partitions.



The general solution is represented in the flowchart above. The sample workflow for the specific queries is described below.

1. Death counts for specific diseases for a specified country:
    a. First we ask the user to select a Dataset for which they want to search the death count.
    b. Next we ask the user to enter the name of a country to look for deaths for that specific country.
    c. The following steps are taken to search for the result:
        i. We run parallel processes for mapping (1 for each partition the file is divided into) which does a select filter over the partitioned data to find the country mentioned in the search input.

```
def mapperQ1(file,country):
    # Steps
    # 1. read partition info
    df = read_data_for_querying(file)
    #print(df.head())
    #   2. filter dataframe based on country
    res = df[df['Country'].str.lower() == country.lower()][['Country','No. of deaths'
    #print(res)

    #   4. return value back
    #
    return res
```

ii. After this the results of all the mappers are collected in a list and sent as input to the reducer, which then further searches for the results in the mapper results.

2. Find countries with a death count between two numbers for a specific disease.
    a. For this search query the user enters lower and upper limits for death count for a specific dataset.
    b. The mapper then runs through all the partitions searching for countries that match the condition. The result for the mappers is then collected by the reducer and returned to the API call.

3. Analyze the count of deaths per country for each disease
    a. The analysis questions are posed to repeat the action for all 3 disease datasets - Sars, Covid, and Ebola.
    b. First the mapper runs through the partitions and collects the number of deaths in the current partition for each country.
    c. Then the reducer sums over the mapper results to produce the total number of deaths per country for the dataset.

4. Analyze the number of people who recovered after being affected by the disease for each disease.
    a. The analysis questions are posed to repeat the action for all 3 disease datasets - Sars, Covid, and Ebola.
    b. First the mapper runs through the partitions and finds the number of people who survived the disease.
    c. Then we collect the total number of countries present in our partition, we return these three values to our mapper to combine and return the average number of people who recovered from the specific disease.
    d. Then the reducer sums over the mapper results to produce the total number of people recovered for the diseases, it then sums over the total country count to calculate and return the average number of people who

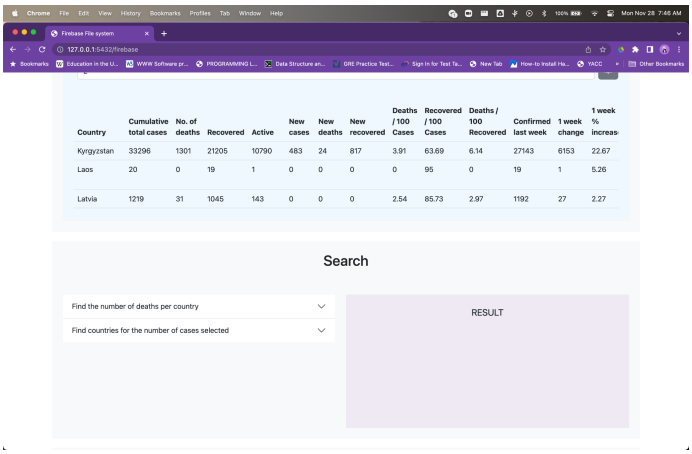
{"mapper": {"sars": [[7693.0, 6964.0], [33.0, 31.0], [259.0, 224.0], [111.0, 103.0]], "ebola": [[1387413.0, 831470.0], [1737814.0, 1021368.0], [1603719.0, 954201.0], [1596516.0, 963003.0]], "covid": [[4167167.0, 4022094.0], [3364889.0, 3209857.0], [1874660.0, 1784999.0], [7073769.0, 6809499.0]]}, "reducer": {"sars": [252.48275862068965, 8096.0], "ebola": [1522.0193782801775, 6325462.0], "covid": [84633.41711229946, 16480485.0]}}

recovered from the disease.

C3. Application User Interface Implementation

1. Search Functions:

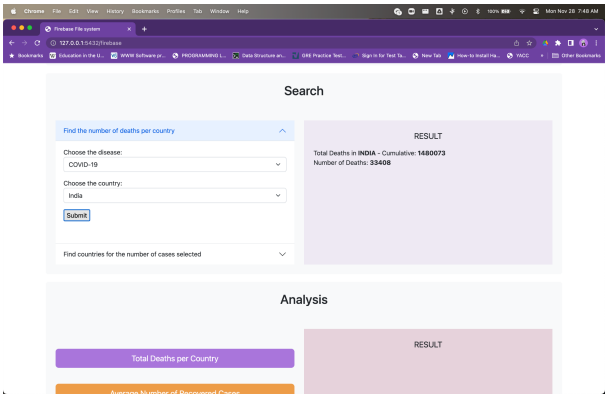
The search functions are present in the Search section of the user interface. The two functions described above in Section C1 and Section C2 are displayed using the accordion HTML component. The output of the search function is displayed in the 'RESULT' section.



Search Section

- ***Finding Death counts for specific disease for a user specified country***

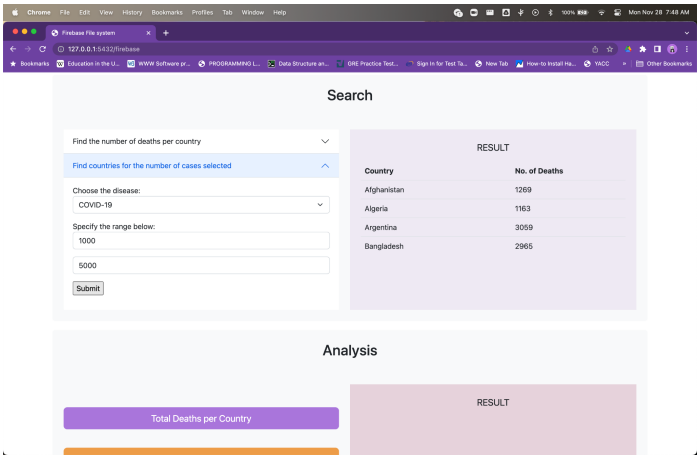
The user can select the country and disease for checking the death counts of that country. By clicking on the 'Submit' button the user can see the number of deaths for that country.



Search death count of the country

- ***Finding countries with a death count between two numbers for a specific disease.***

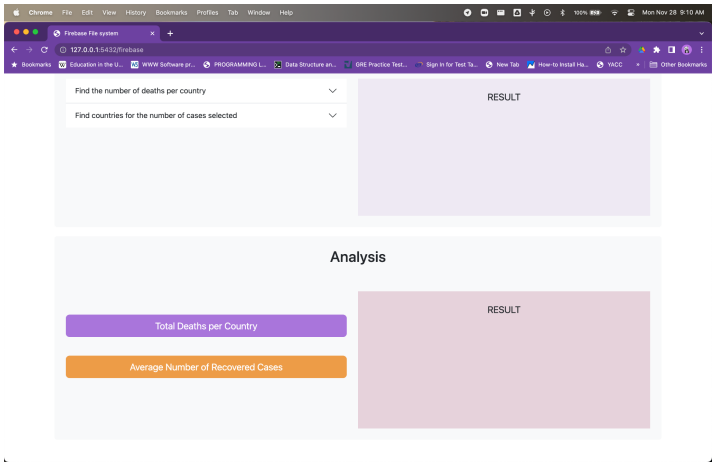
  This function takes in three parameters, disease, limit1 and limit2. By clicking on the 'Submit' button, the countries that have the number of disease cases between limit1 and limit2 are displayed.



Search countries in the user specified range

2. Analysis Functions:

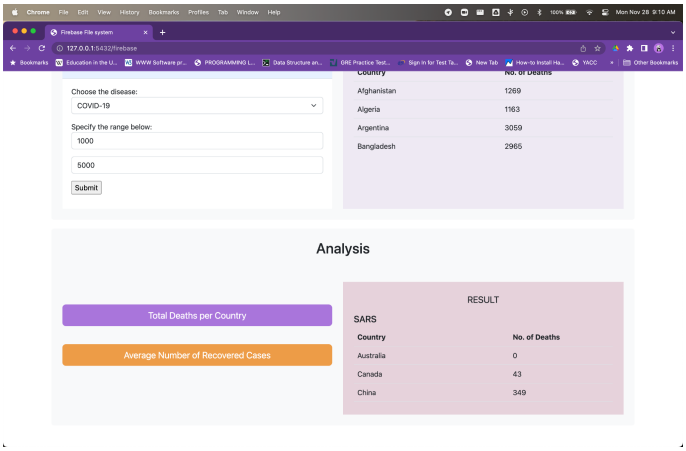
The analysis functions are present in the Analysis section of the user interface. The two functions described for analysis in Sections C1 and C2 are displayed using block buttons. The output of the analysis functions is displayed in the 'RESULT' section.



Analysis Section

- ***Analysis of the death count per country for each disease***

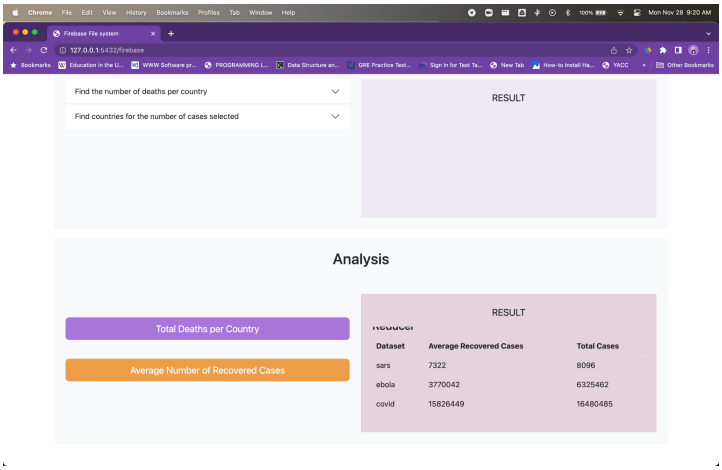
To view the analysis of death count per country, the user can click on the 'Total Deaths per Country' button. The result of the analysis (as explained in Sections C1 and C2) is displayed in the 'RESULT' section.



Number of Deaths per Country output

- ***Analysis on the number of people who recovered after being affected by the disease***

The user can click on the 'Average Number of Recovered Cases' button to view the analysis of the number of people who recovered after being affected by the disease. The result of the analysis is displayed in the 'RESULT' section of the interface.



Number of recovered cases

D. Project Tracker:

Date	Task Description	Responsibilities	Status
9/19	**Proposal**	Group Meeting on zoom to write out proposal	**DONE**
9/19 - 10/3	Research	**Tushar**: Individual research on Firebase **Laxmi**: Individual research on web apps **Raajitha**: Individual research on MYSQL ** Research will include: storing datasets, how we want to use commands, functions to use, user-flow, interface languages, database formatting, contents display, and storing**	**DONE**
10/3 - 10/17	Task-1 Implementation	**Tushar**: implement commands from task one on firebase **Laxmi:** set up web browser app and create structures/interface for potential user **Raajitha:** implement commands from task one on MYSQL **Actually start coding and setting up the commands with their executions. We can use this time to communicate with each other, ask questions and make changes**	**DONE**
10/17-10/31	Task 1 Completion, Task 2 Implementati	**Tushar**: Complete firebase implementation **Laxmi**: Start connecting interface with databases **Raajitha**: Complete MYSQL implementation ** will start to add updates to report **	**DONE**

	on, Midterm Report		
10/31	**Midterm Report**	Group meeting to finalize and submit midterm report	**DONE**
10/31 - 11/14	Task 2 Completion, Task 3 Implementation	Work on PMR operations together	**DONE**
11/14 - 11/28	Complete Task 3 & Integration. Final Report	**Tushar:** Test and finalize task 2 **Laxmi:** Test and finalize task 3 & Integration **Raajitha:** Test and finalize task 2	**DONE**
11/28	**Final Report**	Group meeting to finalize report and presentation video	**DONE**
11/28 - 12/1	Work on Peer Evaluations	** individually work on peer evaluations **	-
12/2	**Peer Evaluations**	Submit evaluations	-

E. Changes to previous project work

1. **MySQL:**
   we changed the put() function to allow lots of data to be loaded fast. I was manually inserting the lines before, but now we just use the “LOAD LOCAL INFILE” to load the csv file extremely quickly because before that, the long datasets were not loading at all.

2. **Firebase:**

   - One of the major changes needed was to introduce children and content in the directory system. This was needed to handle multiple subdirectories and files easily in the EDFS. We further split the location of partitions and

the actual partitions themselves into separate cards. This prevents errors due to path matches and gives a clean and extendable implementation.

- As expected implementation of map reduce emulation required the help of the multiprocessing library in python which essentially allowed the mappers to run in parallel and emulate the working of a distributed file system.

3. **Application User Interface**

The Tree view presented and statically generated during the Midterm is changed on the user interface due to challenges faced during dynamic updates and integration to update and render the Treeview file system successfully. The new approach provides users with intuitive

F. Challenges, Learnings & Future Work:

1. **MySQL:**

- **AWS RDS Instance connection**: This took a lot of time for me to figure out in terms of connecting it to python and forming a proper database using python and MySQL because I do not have windows. I learned a lot with how AWS works and how I can build projects in the future.
- **Loading CSV File**: I had to dabble a lot with the MySQL permissions in order to allow a whole csv file to be loaded into a table. I learned a lot about how you can bulk load csv files easily into MySQL tables instead of inserting each line.
- **Search & Analytics**: These functions were hard to do because some datasets had different data like cumulative columns versus regular columns. I had to do different functions for different datasets. This helped me learn a lot about aggregate functions and how to query on python and MySQL as well as concat multiple tables.

2. **Firebase:**
    - One of the biggest challenges was to get the directory substructure right.
    - Next big challenge was to get all the urls of the firebase system right to ensure the 3 big cards (Dir,Files and Store) are handled correctly

- Thinking about the edge cases for put() function was especially challenging, and it resulted in a big multi-step function.
- The analytics were challenging to process due to the nature of distributed data and cumulative functions, some stats like averages also required me to collect the count of inputs distributed over the dataset partitions. Creating a standardized structure for these distributed requests was a great learning experience.

3. **Application User Interface:**

- The initial design of the user interface of considering the EDFS commands and putting it into an intuitive user interface for the users was a bit challenging.
- As we were new to the Flask web framework and related development, it was a new learning curve for us as a team. We all are relatively new to front-end technologies, and building user interfaces, the initial development phase was a bit challenging for us.
- Javascript, JQuery implementations, handling Axios and rendering the correct response on the front-end, integration and testing were challenging.
- Future work is to host this application on a web application server i.e. on platforms like Heroku, PythonAnywhere, GCP. Improving the performance of the interface, new errors/exception handling, and making the application more intuitive and user friendly.

The project work had a good learning curve, and gave us an opportunity to work together as a team.

G. Useful Links and Resources

Demo Video Link:
https://www.youtube.com/watch?v=bPm5M0QuKhg&feature=youtu.be

Project Code Google Drive Link:
https://drive.google.com/drive/u/0/folders/1M1kANNc5BqycbigY_GKEqcThQ6RSp-z5

GitHub Code Link:
https://github.com/laxmigarde/dsci551-project
https://github.com/laxmigarde/dsci551-project/tree/integration-frontend

Resources and materials referred:

1. Flask Documentation: https://flask.palletsprojects.com/en/2.2.x/
2. Bootstrap Documentation:https://getbootstrap.com/docs/5.0/getting-started/introduction/
3. JQuery: https://www.w3schools.com/jquery/default.asp
4. Axios: https://axios-http.com/docs/intro
5. Javascript MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript
6. Font Awesome: https://fontawesome.com/
7. MySQL APIs: https://webdamn.com/create-restful-api-using-python-mysql/
8. Python web application using Flask: https://www.digitalocean.com/community/tutorials/how-to-create-your-first-web-application-using-flask-and-python-3