Final Project
Name: Chaudhary Tejaswi
Banner ID: B00858613
Date: 14/12/2020

Overview:

Create program that help us to manage the COVID-19 pandemic by tracing contacts. Create program that work as mobile device. Mobile device functionalities are, it stores contacts with other devices and upload other contacts to the central database, stores user's COVID-19 positive test record to the central database and alert mobile user when central database gives alert to it that mobile user has been in contact with COVID-19 positive person.

Apart from this, it also provides functionality to Government for managing situation. Government can find large gathering for particular day that helps Government to understand the people's agreement with physical distancing counseling. Apart from this, it also allows stores COVID-19 test result tested by Test Agency. It also notifies mobile user if he came in contact with COVID-19 positive person.

Files and external data:

This program consists following files:

1) **mainUI.java:** This file contains main program that guide user to perform different operations. It allows mobile device user to perform operation such as record contact and upload to central database, report they have been tested positive for COVID-19. In addition, it allows Government to store contacts in central database, record test result provided by test agency and find large gathering.

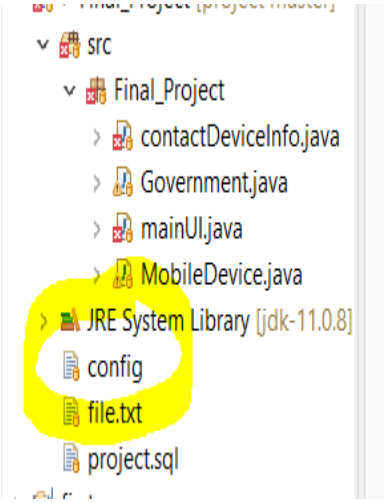
2) **MobileDevice.java:** this file stores users COVID_19 test number(testHash) provided by test agency, mobile user's contact with other mobile device and send alert to user sent by government that he/she has been in contact with COVID-19 positive person. It also allows mobile device user to upload their contact and tests to the central database.

3) **ContactDeviceInfo.java:** This is support class of MobileDevice class. It stores other contacted device hashcode, date of contact and duration of contact.

4) **Government.java:** This file stores contacts to central database, report mobile user if has been in contact with someone who is diagnosed with COVID-19, upload test result tested given by test agency to the central database and find large gathering on given date.

5) **File1.txt:** This file stores mobile device configuration values.
   Example: address=10.0.0fh.02hj
   deviceName=Samsung
   NOTE: user needs to give file name with its path.

6) **config :** This file contains database configuration value.
Example: database=databasedomain/databaseName?timezone
(jdbc:mysql://db.cs.dal.ca:3306/databaseName?serverTimezone=UTC)
User=xyz
Password=sfdg@1334
Note: user needs to create this file inside project where source and bin folders are present.
For reference:



7) **project.sql :** This file contains SQL script to create schema and tables. This file creates 5 tables. One for storing mobile devices that are synchronized with government server, second for storing contact details of two devices, third for storing test results, fourth for storing.

Data Structure:

Following Data Structures are used in this program.

Object of class contactDeviceInfo is used to store contacted other device hash, date of contact and duration of contact.

HashMap1<String, List<String>> is used to store contacts information of mobile device where mobile device is stored as key in Map and contacts as key in values. For value, ArrayList is used so multiple contacts infomation can be stored for mobile device and its object of contactDeviceInfo. Same as, **HashMap2<String, List<String>>** is used to store testHashes(COVID-19 test result identifier) of current mobile device. In HashMap2, mobile device hashcode is stored as key, and value is ArrayList that stores list of all testHashes for that mobile device.

ArrayList is used to store contacted othe device hashcode while storing to the central database by reading xml formatted string for finding, and that is used to find COVID-19 positive contacts.

HashMap3<String, List<String>>It stores device hash code as key in HashMap3 and list of all other contacts in ArrayList as value**. HashMap4<String, List<String>>** is also used in find gathering to store **ArrayList3** that has list of devices that satisfy density condition. **Set1<ArrayList<String>>** is used to store all the set**(ArrayList3**) find for each pair.

Database Schema:

Here , for this project 5 tables are created.

1)mobiledevice- this table stores initiator

Field	Type	Null	Key
device	varchar(50)	NO	PRI

2)covidtest- this table stores COVID-19 test result

Field	Type	Null	Key
testHash	varchar(50)	NO	PRI
dateoftest	(11)	NO	
testResult	tinyint(1)	NO	

3)devicetestinfo- this table stores device and testHash allocated to it.

Field	Type	Null	Key
device	varchar(50)	NO	MUL
testHash	varchar(50)	NO	MUL

4)alertcontact- this table stores information of alert. It stores device that is alerted, other device from which device is alerted and other device testHash and date of test.

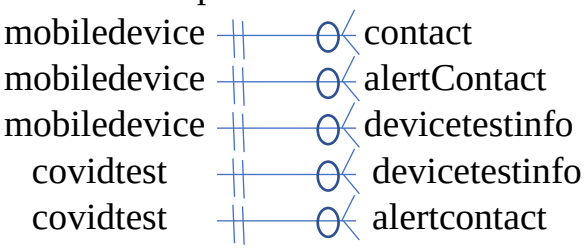
Field	Type	Null	Key
device	varchar(50)	NO	MUL
otherdevice	varchar(50)	NO	
testHash	varchar(50)	NO	MUL
dateofcontact	int(11)	NO	

s

5)contact- this table stores contacts information.

Field	Type	Null	Key
device	varchar(50)	NO	MUL
otherdevice	varchar(50)	NO	
dateofcontact	int(11)	NO	
durationofcontact	int(11)	NO	

Relashionship between tables:



Design elements:

MobileDevice.java:

MobileDevice(String configurationFile, Government contactTracer):

- This constructor store Government class object that will be used to record contacts to the central database.
- It will read configurationFile line by line, split line values by '=' and store device address and device name.
- Contacte address and device name and generate hash code with the help of string.hashCode() .
- Store mobile device hashCode in **hashMap1** and in **hashMap2.** Here, current mobile device is stored as key in both the hashMap and values as ArrayList that contains contacts details and user testHash given by test agency.
- Mark device as current device, as this class stores data for multiple mobile device. It will be used later in program to store contacts and device test hash.
- It will store multiple devices hashcode in Map. If once it is stored in Map it will not store that device hashcode again, and it will mark that device as current device for mobile class.
- It will throw Exception if file name is empty or null, one of the device configuration is not exist in file, one of the or both configuration values are null or empty or file is not exists.

public boolean recordContact(String individual, int date, int duration):

- This method will record current mobile device's contact with other mobile devices.
- Firstly, it will check input comes from keyboard. It will return false if date is less than 1, duration is less than one, individual is empty string or null, or individua hashcode is same as current active mobile hashcode, as mobile device cannot record contact with it self, and return false to user tif contact is not recorded if one of the above case encounters.
- It will call ContactDeviceInfo class to store individual, date and duration with other contact.
- It will add contact information to the ArrayList of current mobile device, and return true if contact is locally recorded successfully.

public boolean positiveTest(String testHash):

- This method will store testHashes(identifier of COVID-19 test report) of current mobile device.
- Firstly, it will validate testHash string. It will return false is testHash is null or empty String. In addition, it will also return false if testHash is already exists (duplicate entry) in **hashMap2** or teshHash string is not in the form of alphanumeric.
- It will store valid testHash string to the ArrayList of current mobiledevice. Here, current mobile device is key and ArrayList that contains tets hash is value in the hashMap.

public boolean synchronizeData( ):

- It will send locally stored data of current mobile device (Contacs and testHashes) to government class method name mobileContact in order to upload data in the central database. It will pass current active mobile device as initiator and xml fomated string as contact info.
- This method is called periodically. When this method is called, it will package all the information of mobile device in a formatted string (XML format) and send it to Government class.
- To format information in XML, it will call xmlFormate method. It will pass current mobile device hashcode as parameter to xmlFormate().
- It will return true to mobile device user if government notify to it that current mobile device has been around someone who is tested positive for covid-19 withing last 14 days, or it will return false to mobile device user if government notify to it that current mobile device has not been around someone who is tested positive for covid-19 within last 14 days.
- It will also return false to user if mobileContact throw any exception to it.
- After getting response from Governemnt, whether its positive, negative or exception, it will clear all the locally stores current mobile device information.

private String xmlFormte (String currentDevice):

- It will store all the current mobile device information in xml format and append it in a string.
- Firstly, it will store contacts details such as other device hashcode, date and duration. Then, it will store list of all the testhashes of current mobile device.
- Return that stored data string to synchronizeData.
  Sample of xml formatted string:

```
< ?xml version=\"1.0\" encoding=\"UTF-8\"? >
<details>
        <otherContacts_list>
                <contactInfo>
                        <Device>123654</Device>
                        <Date>52</Date>
                        <Duration>23</Duration>
                </contactInfo>
        </otherContacts_list>
```

```
        <deviceTest>
            <test>
                <DeviceTestHash>Abc123</DeviceTestHash>
            </test>
        </deviceTest>
    </details>
```

contactDeviceInfo.java
contactDeviceInfo(String individual, int date, int duration):

- This constructor stores current mobile device contact with other device information such as other device hashcode, date of contact and duration of contact**.**

Government.java:
Government(String configurationFile):

- This constructor stores database configuration vales, such as domain name, user and password for later use in program to connect with database to store data in the centralized database.
- Firstly, it will find path of configurationFile. And then read file data's line by line.
- It will split each configuration value by '=' and store values (database domain name, user and password) for later use.
- It will throw an exception if file name is empty or null, file not exists, one of the three configuration value is not exists in file or null or empty string.

public boolean mobileContact(String initiator, String contactInfo):

- This method has multiple responsibility. It will read xml formatted string and store each value in the central database. It will also look up in the database to check whether any other devices contacted initiator tested COVID positive and notify to mobile device.
- Firstly, it will store current day from date January 1, 2020 by subtracting January 1, 2020 from systems current date. That will be used to in later to find out that any other device tested COVID positive and has been in contact with initiator within last 14 days.
- Then it will establish connection with database.
- After that, it will use DOM to read xml formatted string.
- Firstly, it will read all the contacts information one by one that are inside "contactInfo" node or element, and store otherdevice hashcode, date and duration in local variables. It will pass these parameters to SQLcontact method to store contact information permanent in the central government database. If SQLcontact return false to it, it will print message that contacts are not recorded in the database so user get the idea that their data is not uploaded to the database successfully. It will also store each other device testhash from contacts to the ArrayList for later use in finding covid-19 positive contacts within last 14 days.
- After storing contacts, it will read testHashes one by one that are inside "test" node, and it will call SQLCovidTest method to store testhashes to the central database permanently. If SQLCovidTest return false to it, it will print message that testHashes

are not recorded in the database so user get the idea that their data is not uploaded to the database successfully.

- At the last, this will call findPositiveContact method in order to check whether initiator has been around someone who is tested COVID positive within last 14 days. If findPositiveContact method return true, it will notify true to mobile device, or if findPositiveContact method return false, it will return false to mobile device.
- It will throw exception to synchronizeData() , if database connection fails. Before retuning to synchronizeData() it will close database connection.

private boolean SQLcontact(String initiator, String otherdevice, int date, int duration):

- This method record contacts to the central database.
- Firstly, it will check that initiator is already exists in mobiledevice table.
- If initiator not exists in the mobiledevice table, it will record initiator in mobiledevice table, and then it will record contact details (otherdevice hashcode, date and duration) in contact table. It will also store initiator along with contact information in the contact table. Here, initiator(device) is foreing key that refers mobiledevice initiator(device) column. initiator(device) in mobiledevice table is primary key.
- After checking for initiator(device)in mobiledevice table, it will store contact information and initiator in contact table.
- If initiator already exists in the mobiledevice table, it will check in contact table whether any duplicate entry exists in contact table(this will happen when same device comes multiple times in contacts with initiator at the same day), or not. If current contact information already exists in contact table, it will just add new duration with stored previous duration and update duration column for that entry. If current contact information is not present in contact table, it will add as new entry in the contact table.
- It will return false to mobileContact if any error occurs while storing contacts information. If all the record stores in database successfully, it will return true.

private boolean SQLCovidTest(String initiator, String testHash):

- This method will store mobile device testhashes provided by test agency to the central database.
- Before storing to database, it will check whether the given test hash is already stored in devicetestinfo table with anothera mobile device or with initiator. If any entry with given testHash exists in devicetestinfo table, it will not store it again, as each testhash has only one user and user will not have same testHash in devicetestinfo.
- It will also check that initiator is stored in mobiledevice table and testHash is also already exists in covidtest table. If both exists then only it will record current data in devicetestinfo table because device(initiator) in devicetestinfo table is foreign key tat reference the device(initiator) in mobiledevice table, as well as testHash in devicetestinfo table is also foreign key references to testHash of covidtest table.
- It will return false, if current testHash is not exists in covidtest table, any entry with testHash is already exist in devicetestinfo or any error occurs while storing tuple in the table.

private findPositiveContact(String initiator, ArrayList<string> contact_list):
following steps will be performed to find out COVID positive contact.

- Firstly, this method will find out last 14^{th} day from current day.
- After, it will retrieve all the contacts (device hashcode) and date of contact they meet with initiator from the contact table where date of contact is grater than the last 14^{th} day.
- After, retrieving all the contact of within last 14 days, it will fetch testhashes of that other devices, and it will use its result to fetch date of COVID test and testhash result from covidtest table for current other mobile device where testHash of covidtest table is in result of inner query, result of COVID test is true and difference of date of contacts meet and date of test result is less than 14 and or greater than -14 (as if user meet today in other device test result published today then difference will be 0).
- Before returing true to mobileContact(), it will make sure that the alert for current other device is not send to initiator. For that, it will look in alertContact table that stores inititor, other device for which alert is already sent, other device testHash and date of test. If such entry exists in alertContact table, it will check in ArrayList whether the current other device comes in contact with initiator again, if comes again it will set flag variable to 1 that means initiator comes in contact with someone who is tested COVID-19 positive withing last 14 days, otherwise it will ignore that device contact.
- If such entry not exists in alertContact table, it will set flag variable to 1 and add data of current alert in the alertContact table.
- At the end, it will check flag variable. If it was set to 1 it will return true to mobileContact() else return false.
- It will also return false and print message for user if any error occurs while finding positive contacts.

public boolean recordTestResult(String testHash, int date, boolean result):

- It will return false if testHash is not alphanumeric,null or empty string , date is less than 1 or any error occure during uploading test result to the central database. Also , it will print valid message so user get to know the why test result is successfully not recorded.
- If input is valid, it will connect to database.
- Before recording to testHash in covidtest table , it will check where testHash is already exists in table or not. If testHash is already exists, it will return false because testHash is primary key of covidtest table which cause error.
- If testHash not exists in covidtest table, it will record test result in the table and return true to user.

public int findGatherings(int date, int minSize, int minTime, float density):

- This will find large gathering on the given date.
- It will return 0 if any invalid input stch as date is less than 1, minSize is less than 2 , minTime is less than 1, density is less than or equals to 0 or any error occurs while

fetching data from database to find large gathering without performing furthure operation.

- To find large gathering following steps will be performed:
    - fetch all the contacts from contact table who meet on given date and for given minTime.
    - Create adjacency list for all the contacts (eg. A ={B,C}, B={A}, C={A}). To create adjacency list, used HashMap3 that stores key as device and values as list of other contacted devices.
    - Also, add each device in ArrayList2 so later it will be used to create all possible pairs.
    - It will use ArrayList2 and make pair of two devices.
    - After that, it will take pair of two devices, find all possible people who are connected to both devices. And add two paired devices and other devices connected to both paired device in ArrayList3 called it neighbors. After that it will sort ArrayList3 so it comes in descending order in ArrayList. In next step, it will store ArrayList3 in Set.
    - After finding neighbout for current pair, it will check whether ArrayList3 has atleast minSize number of devices. If ArrayList3 contains at least minSize number of devices, it will process with this list further, else it will neglect that ArrayList and took another pair of devices.
    - If minSize condition satisfy, it will check in Set1 whether set1 contains ArrayList3 elements to avoid duplicate neighbor set. If Set1 contains current ArrayList3, it will also neglect this Set and will consider density for given set is already counted and checked.
    - If Set1 does not contain ArrayList3 it will find all connection exists between all the devices present in ArrayList3 by calling findConnection method called it c.
    - After finding all possible connection, it will find density by divide it with maximum number of connection possible by using n(n-1)/2 where n is number of devices in ArrayList3.
    - It will consider this Set(ArrayList) for density if c/(n(n-1)/2) is greater than density.
    - Apart from this, it will also remove subset if any ArrayList present in Set1 has superset(for example: it will remove {A,B,C} if {A,B,C,D} encounters).
    - To remove subset, it add all valid ArrayList3 which satisfy density condition in HashMap4 as key and density as value.
    - If superset comes it will remove subset from HashMap4, and store its superset. If superset exists in Set1 but not stores in HashMap4 at that time it will add subset in HashMap4. If superset comes in HashMap4, it will remove previously stores subset. If subset satisfy density condition, it will check in HashMap4 whether its superset is exists, if exists it will neglect that subset.
    - The above steps, will be performed for all possible pairs.
    - It will return size of HashMap4 as large gathering as HashMap4 contains all that sets that will be considered as large gathering.

private int findConnection(ArrayList<String> people, Map<String, List<String>> connectedPairs):

- It wil find all connection between devices stored in ArrayList with by looking in Map(adjacency list).
- Return number of existed connection to findGathering().

Assumption:

- configurationFile will contains valid data (Device's network address (String) and device name (String)) and will be in the following form.
  Address=10:00225:556fh
  DeviceName=Samsung
- Each line in text file will contains single configuration value.
- If our mobile device comes into contact with another device twice or more then keep all the new and existing contacts details with that another device.
- User will not enter date greater than the current system date (future date).
- User will not get multiple testHashes as a same day.

Limitation:

- Find gathering may not give accurate result.
- Unused (from very long time which is not useful anymore) data will not be deleted from database tables.

Test Cases:

Following test cases are tested :

Input Validation:

1. **MobileDevice Class:**
   (1) MobileDevice (String configurationFile, Government contactTracer):
   - Null value passed as file name - **not acceptable**.
   - Empty string passed as file name - **not acceptable**.
   - Valid string passed as file name - **acceptable**.
   - Null value is passed to Government class object(contactTracer) - **not acceptable**.
   - Instantiate Government class object(contactTracer) - **acceptable**.

   (2) recordContact (String individual, int date, int duration)
   - Null value passed for individual - **not acceptable**.

- Empty string passed for individual - **not acceptable**.
- Only one configuration value (either network address, or device's name) passed in a string for individual - **not acceptable**.
- Valid input string passed for individual - **acceptable**.
- 0 value passed for a date (day 1 = January 1,2020) - **not acceptable**.
- Negative integer passed for date - **not acceptable**.
- Any valid positive integer for date - **acceptable**.
- 0 value passed for duration - **not acceptable**.
- Negative integer passed for duration - **not acceptable**.
- Any valid integer greater than 0 passed for duration - **acceptable**.
- Provide duration greater than 1140 – **not acceptable**

(3) positiveTest (String testHash)

- Null value passed for testHash - **not acceptable**.
- Empty string passed for testHash - **not acceptable**.
- String passed for testHash is not in the form of alphanumeric string (for ex: contains only characters, integers or alphanumeric with special characters) - **not acceptable**.
- Valid alphanumeric string passed for testHash - **acceptable**.

2. Government Class:

(1) Government (String configurationFile):

- Null value passed as file name - **not acceptable**.
- Empty string passed as file name - **not acceptable**.
- Valid string passed as file name - **acceptable**.

(2) boolean mobileContact (String initiator, String contactInfo)

- Valid string passed for initiator and contactInfo - **acceptable**.

(3) recordTestResult (String testHash, int date, boolean result)

- Null value passed for testHash - **not acceptable**.
- Empty string passed for testHash - **not acceptable**.
- String passed for testHash will not be in the form of alphanumeric string (for example: contains only characters, integers or alphanumeric with special characters) - **not acceptable**.
- Valid alphanumeric string passed for testHash - **acceptable**.
- 0 value passed for date - **not acceptable.**
- Negative integer passed for date - **not acceptable.**
- Any valid positive integer passed for date - **acceptable.**

(4) int findGatherings (int date, int minSize, int minTime, float density)

- 0 value passed for date - **not acceptable**.

- Negative integer passed for date - **not acceptable.**
- Any valid positive integer passed for date - **acceptable**.
- 0 value passed for minSize - **not acceptable**.
- Negative integer passed for minSize - **not acceptable**.
- Valid positive integer greater 1 is passed for minSize -**acceptable**.
- 0 value passed for minTime - **not acceptable**.
- Negative integer passed for minTime - **not acceptable**.
- Valid positive integer passed for minTime - **acceptable**.
- 0 value passed for density - **acceptable**.
- Negative float value passed for density - **not acceptable**.
- Valid positive float value greater than 0 passed for density -**acceptable**.
- Provide minTime greater than 1140- **not acceptable.**

Boundary cases:

1. **MobileDevice:**

(1) MobileDevice (String configurationFile, Government contactTracer)

- File does not exist.
- Empty file.
- Create mobile device with valid configuration values given in the file.
- Load first device(no other device exists in class).

(2) recordContact (String individual, int date, int duration)

- record first contact of current mobile device with individual.
- Record contact with date 1.
- Record contact with date greater than 1.
- Record contact with 1minute duration.
- Record contact with todays date.
- Record contact with duration 1140.

(3) positiveTest (String testHash)

- Provides alphanumeric string of length two to testHash (Ex: A0).
- Provides alphanumeric string of any length greater than two to testHash.

(4) boolean synchroizeData()

- Call method when MobileDevice has single mobile device and single contact information recorded to pass to the Government class.
- Call method when MobileDevice has single mobile device and multiple contacts details recorded to pass to the Government class.
- Call method when MobileDevice has multiple mobile devices and multiple contacts are recorded to pass to the Government class.
- Call method when MobileDevice has positive test hashes or not for mobile devices including above three cases.
- Call method when MobileDevice has single positive test hashes.

2. **Government Class:**

(1) Government (String configurationFile)

- File does not exist.
- Empty file.
- Successfully connect to the database with the given configuration values in the file.

(2) boolean mobileContact (String initiator, String contactInfo)

- Method has single mobile device and single contact details to store in the database.
- Method has single mobile device and multiple contacts detail to store in the database.
- Mobile device has not been in contact with someone who is Covid-19 positive.
- Mobile device has been in contact with someone who is Covid-19 positive.

(3) recordTestResult (String testHash, int date, boolean result)

- Provides alphanumeric string of length two to testHash.
- Provides alphanumeric string of any length greater than two to testHash.
- Record Covid-19 test result tested on date 1.
- Record Covid-19 test result tested on date any other than 1.
- Record positive Covid-19 test.
- Record negative Covid-19 test.

(4) int findGatherings (int date, int minSize, int minTime, float density)

- Find large gathering on date is 1.
- Find large gathering on date any other than 1.
- Find large gathering with minSize exactly 2.
- Find large gathering with minSize is greater than 2.
- Find large gathering with minTime 1 minute.
- Find large gathering with minTime 1140.
- Find large gathering with density is more than 0.
- Method finds only one large gathering with given data, minSize, minTime and density.
- Method finds more than one large gatherings with given data, minSize, minTime and density.

Control Flow:

1. **MobileDevice class:**

(1) MobileDevice (String configurationFile, Government contactTracer)

- Read same mobile device configuration values more than once.
- Read new mobile device configuration values.
- File contains single configuration value (either network address, or device’s address).

- One of the configuration values (network address and device's name) is either null, or empty string.

(2) recordContact (String individual, int date, int duration)

- Record contact between current mobile device and individual which is already recorded but comes in contact with different date.
- Record contact between two mobile devices which is not recorded.
- Record contact between two mobile devices meet on same day(they meet twice in a single day).

(3) positiveTest (String testHash)

- Report a new testHash value to mobile device which has already testHash.
- Report same testHash value which is already stored in class.
- Report a testHash value which is failed to store in the database due to database connection failed.

(4) boolean synchroizeData()

- Call method when mobile device has no any contacts detail or positive test hash.
- Call method when mobile device has single contact detail or positive test hash.
- Call method when mobile device has multiple contacts detail or positive test hash.
- Call method when government has alert to notify mobile device which has been recorded some days ago (came in contact with contact and after some days tested positive).
- Call method when mobile device again come in a contact with device for which alert mobile device has received.

2. Government Class:

(1) Government (String configurationFile)

- One or two of the three configuration values is not given in the file (either network address, or device's address).
- One, two or all three configuration values is either null, or empty string.
- One of the three configuration values is not correct (For ex: wrong password or user name or domain name).

(2) boolean mobileContact (String initiator, String contactInfo)

- initiator is not stored in the database.
- Any contacts are not stored in the database.
- initiator or any contacts available in database.

- Store data which is already available in the database (duplicate entry).
- Record testResult which is not stored in government database.
- Record testResult which is already assigned and stored to another device.
- Record testResult which is already stored.
- Initiator has been in contact with COVID-19 positive device.
- Initiator has been in contact with COVID-19 positive device before some day ago.
- Initiator has been not in contact with COVID-19 positive device.

(3) recordTestResult (String testHash, int date, boolean result)

- record Covid-19 test result which is already stored in the database (duplicate entry).
- record Covid-19 test result which is not stored in the database.
- record Covid-19 test result that has same testHash which is available in the database, but have at least one different value of either date, or result (Ex: update in the database for given testHash).

(4) int findGatherings (int date, int minSize, int minTime, float density)

- Find large gathering on given date that has no contacts recorded on that date.
- Find large gathering that is not satisfied either minSize, minTime, or density.

Data Flow:

1. **MobileDevice:**
   - Call synchronizeData before storing current mobile device's contacts with other mobile devices or device.
   - Call synchronizeData before storing current mobile device's testHash with other mobile devices or device.

2. **Government:**
   - Call mobile connect before calling storing Covid-19 test result.
   - Find large gathering before calling mobileConnect.