POLITECNICO
MILANO 1863

Trace & Track Contacts

Systems and Methods for Big and Unstructured Data
SMBUD-Fall 2021

Amirhossein Zhalehmenrabi-10832946 Ghodrat Rezaei-10695039 Mahsa Meymari-10833116

Mahshid Shiri-10832947 Alireza Nadirkhanlou -10823835

Problem Specification

Problem Description

The goal is to Trace and Track contacts among people. The problem is about handling connections among people who:

- ✓ Lives in the same place and are always in connection.
- ✓ Meet each other and we get notified by getting signals from devices such as their cellphones.
- ✓ Are in the same place in the same time like a restaurant.

Also, it is needed to save user's personal data such as test result, vaccines and etc; and warn people who were in contact with someone who is infected by CORONA directly or indirectly.

How we managed…

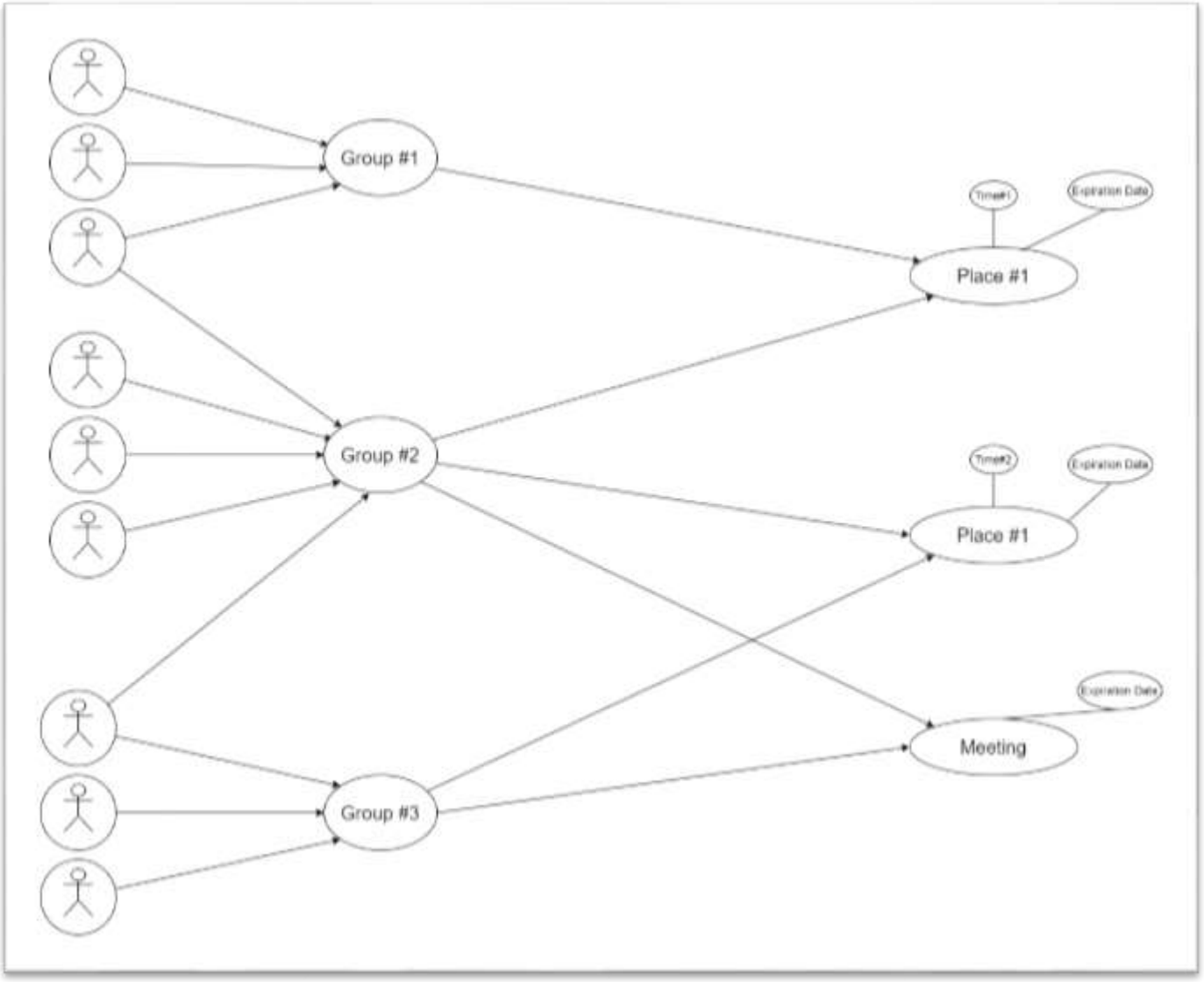
- *A person?* Each person is an instance with attributes corresponding for personal data.
- *Always in connection?* We have group definition for this, a group represent people who are always in touch like families, housemates, colleges and etc. Each person can be a member of different groups.
- *A meeting?* Each meeting is unique so whenever we get notified of a meeting we make a new instance that represent it. Then we draw an arc from User's groups to new instance.
- *Same place, same time?* We have time slots and save different instances of a place for different time slots. When someone goes to a place, an arc is drawn from its groups to the instance corresponding that place in that time slot. So, we can extract implicit connections.
- *Warning about infection and its possibility?* If one of the user's group member is infected, the user is consider as dangerous. If user was in the same place and time with a person, who is infected, he/she is in danger of level 1 and a possibility of getting infected in that place in that time is given to user by the long history that we save. And if user was in connection with a person that is in danger of level 1,then he/she is in danger of level 2 and get the report about possibility of infection in the place that person with danger of level 1 was.

Hypothesis

- Each person is a member of at least one group and each group has at least one member.
- Contacts are in scale of groups not individual which means:
  - ✓ If the test result of a member of each group is positive, the whole group is considered as positive and dangerous.
  - ✓ A group status would change from positive and dangerous to normal if and only if all positive members represent negative test result.
  - ✓ Only one test result is saved for each person, because the oldest ones would not be valid anymore.
- Contacts are valid for 10 days. Whenever the system wants to extract information, it will remove expired data.
- The exact time of meetings are not saved. The system consider a time slot for each meeting. The time slots are morning, afternoon and night.
- The system works based on places and time slots, so implicit and explicit contacts such as contacts among people in a cinema in the same time are extracted.
- Possibility of infection is reported as two types with possibility of infection in the possible place in that time slot.
- A person is considered infected only if he/she represents positive test result.
- The place of infection always would be found in level 1 or level 2 connections.
- Each person's name is unique and act as identifier.

Diagram

Contact Graph:



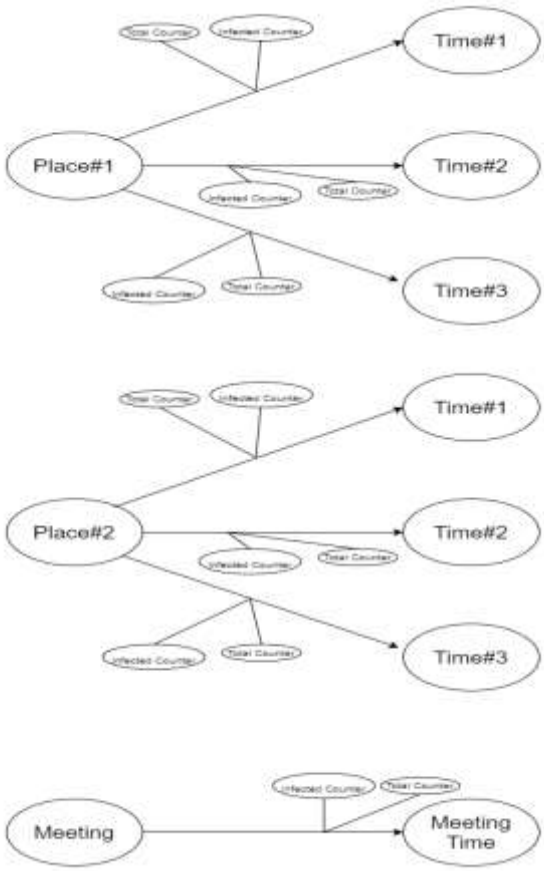
This graph contains information about contacts happening between people. There are some nodes in the graph which need explanation.

1. Person node: the nodes which has dummy image is "Person" node. It contains information about a person, his last test result and if he is vaccinated or not.
2. Group node: people can live together in different ways, for example, being a family member, housemate or colleague. These people are tightly connected so we assumed every kind of groups as "Group" node.
3. Place node: we assumed that every place has just 3 time slots. The "morning", "afternoon" and "night". If we want to generalize this, we can increase it to 24 slots to model whole day. If a group goes to a place, the date that they go and the time is unique for every meeting happening in that place. For example, if two groups went to a restaurant today morning,

there will be a place node with restaurant as main label and time and expiration date as its attributes. Expiration date is the date that this connection will be expired, assumed to be 10 days after the day that meeting happens. Using this expiration date, we will update our database to be as recent as possible.

History Graph:

This graph contains information about the long-term history of crowdedness and dangerousness of a place. For every place, there are as much as time slots that are defined in last part. The places are connected to these time slots using HistoryArc which has 2 attributes, infected counter and the Total counter. Infected counter is the number of people which represent positive test after they have gone to that place in that time slot. It will be updated once a person represent a positive test. The Total counter is the total number of people that went to that place in that particular time slot. It will get updated once a group goes to a place. Using these two attributes, we can compute how much dangerous a place is in a time slot.



Dataset Description

Database consists of 5 nodes (Person, Group, Meeting, Place and TimeHistory) and accordingly 3 types of relationships (BelongsTo, HasGoneTo and HistoryArc) which connected these nodes to each other. Person node has 5 attributes (name, Gender, Test, Vaccine, and Age) and Group node implicitly means as family or other kinds of entities which Persons belong to. Therefore so-called relationship BelongsTo connect these two nodes to each other. Since one infected Person in group can affect other members of the group, group itself was considered as a mover to Place and Meeting Nodes and relationship HasGoneTo demonstrate this connection. Having time attribute (morning, afternoon and night) in Place, whole information of time of people who went to the place was summarized in attribute of the Node Place rather than in one new node. Place was considered as a place (bar, restaurant, coffee, …) where people get in touch with each other, while Meeting node is just a meeting without necessarily going to place. Last node called TimeHistory has three values of morning, afternoon and night in its Type attribute. To count

number of infected people and how virus can be transferred by these people through places, we added to the relationship HistoryArc two attributes: infection and total. Total attribute is considered number of times that people went to the place and infection is number of times that infected people went to the place. So by connecting two nodes TimeHistoy and Place with relationship HistoryArc, total and infected index for each place is obtained.

Commands & Queries

Commands:

- *Contact_occurs:*
  - ✓ **Input:** place, time, a list of individuals
  - ✓ **Procedure:** The groups of each individuals is found at first. Then, the system checks, if the node corresponding to the place and time does not exist, it creates a node. An arc is drawn between individuals groups and the node. The Update_history_graph commands are called at the end.

  Ps: the contacts which are tracked by devices report the place as meeting. Each meeting is unique, so the system always creates a new node to represent that specific meeting.
- *Find_dangerous_contact:*
  - ✓ **Input:** person who is infected/user who wants report , action/no action
  - ✓ **Procedure:** First, the system finds user's groups. Then places nodes that groups where user or his/her groups has gone to, is found. The groups which are connected to the places' nodes are scanned to see that if a dangerous group was there or not. If the dangerous contact is found, the system reports it. Else, the system search the second level, which means doing the same for all groups that User's groups were in contact.

  Ps: At first the system calls Delete_expired_contacts to have only up-to-date data. The Update_history_graph should be called by admin at the end if the call is for action.
  - ✓ **Output:** place, time slot, level of infection
- *Delete_expired_contacts:*
  - ✓ **Input:** --
  - ✓ **Procedure:** Based on expiration date attribute, it deletes every arc which is connected to the node. At the end, the node is deleted too.
- *Update_test_result:*
  - ✓ **Input:** positive/negative, a person
  - ✓ **Procedure:** the system updates the attribute of the node corresponding the person to Positive/Negative and change the label of each group that the person is a member of, to dangerous/normal. It is checked that if a distinct person in the group has positive test result, the group label is not changed. If the test is positive, Find_dangerous_contact is called at the end.
- *Update_history_graph:*
  - ✓ **Input:** a place, a time slot, positive/normal
  - ✓ **Procedure:** if it is a normal call, then the total counter that represent number of people that were in that place in that time slot is updated. Else, an infection is caused in this place and time, so the infected counter on the arc for positive tests is ++.

Queries:

- *Most dangerous places:*

each place in history graph has two attributes on the arc to a time slot that shows number of infections and total number of people that were in that place in that time. So we can calculate $\sum_{\text{time slots}} \#\text{infections} / \sum_{\text{time slots}} \#\text{visitors}$. The system calculates these possibilities for all places then sort them and shows a sorted list.

- *Most dangerous times:*

The system calculates $\sum_{\text{arcs}} \#\text{infections} / \sum_{\text{arcs}} \#\text{visitors}$ for each time slot in all places. The sorted list is shown.

- *Most dangerous time of a place:*

For finding the most dangerous time of a place, the system calculates $\sum \#\text{infections} / \sum \#\text{visitors}$ for every time slot and then, based on the result, the system will report the time slot that is more dangerous for that place.

- *Possibility of user infection:*

The system uses Find_dangerous_contact with no action input. Then it has the dangerous place, time and possibility level of infection.

- *History:*

Using this query, a person can figure out the places that all his groups have gone in last 10 days.

UI Description & guide

Ui:

User Guide:

UI has two parts: for end-user a webpage is provided and for the admin of the system an API is provided. Figure 1 is the webpage.

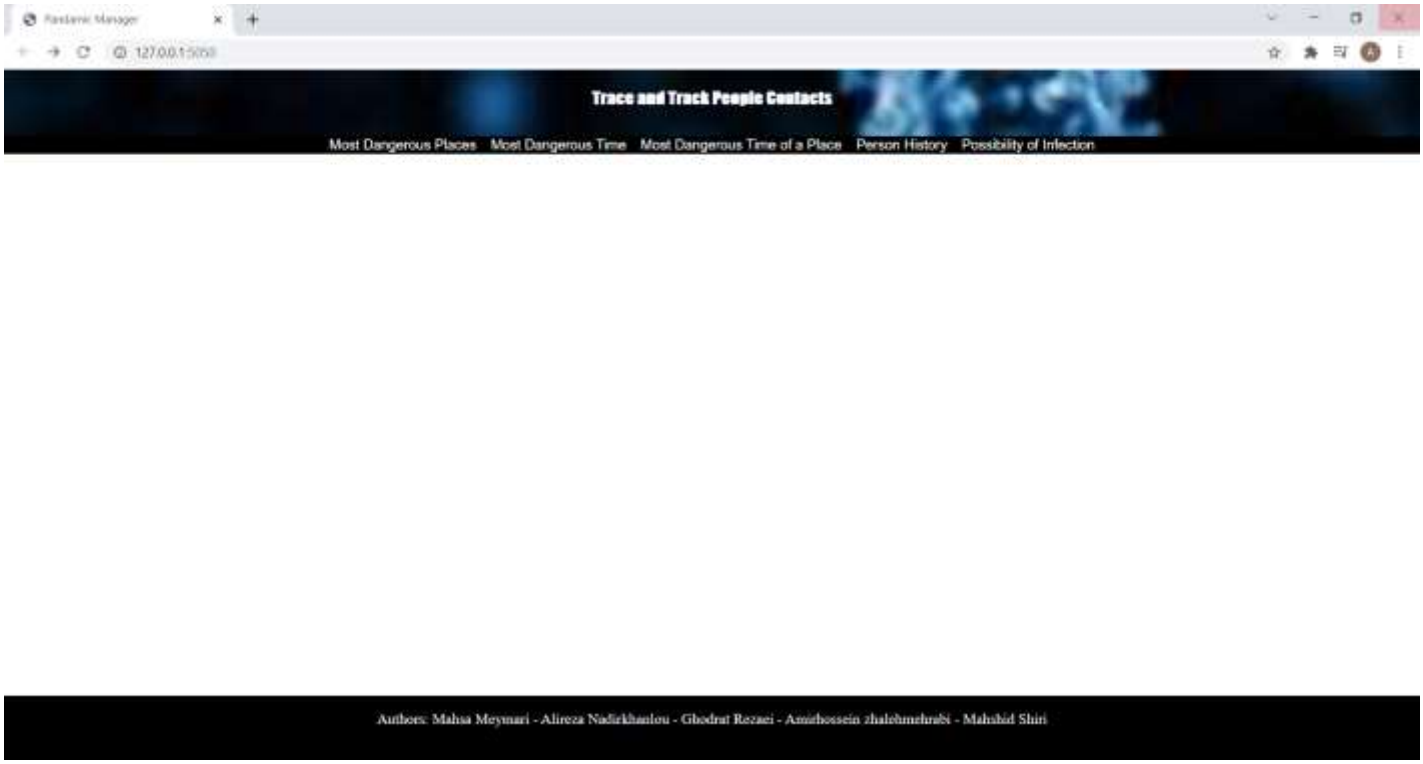


Figure 1

The 5 queries can be seen in header. The user can click on each query and enter the needed input and receive the result. In figure 2 the result of "Most dangerous places" is shown.

Figure 2

In the figure 3 we can see the result of "most dangerous time".

Figure 3

For checking worst times of day for going to a specific place, the user should use the third button and enter the name of the place (Figure 4).

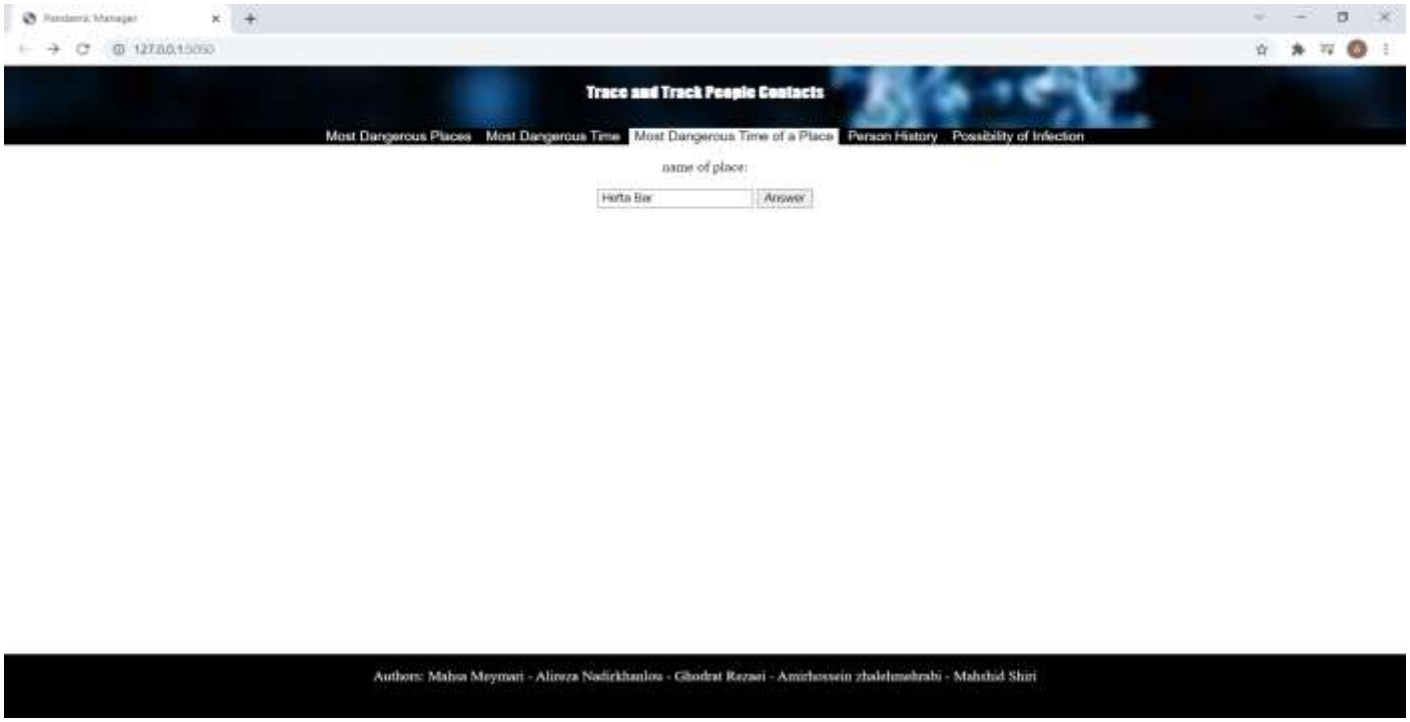


Figure 4

Then the user should click the Answer button to see the result (Figure 5).

Figure 5

For checking a person's groups' history, user should click on person history button and enter the person name (figure 6)

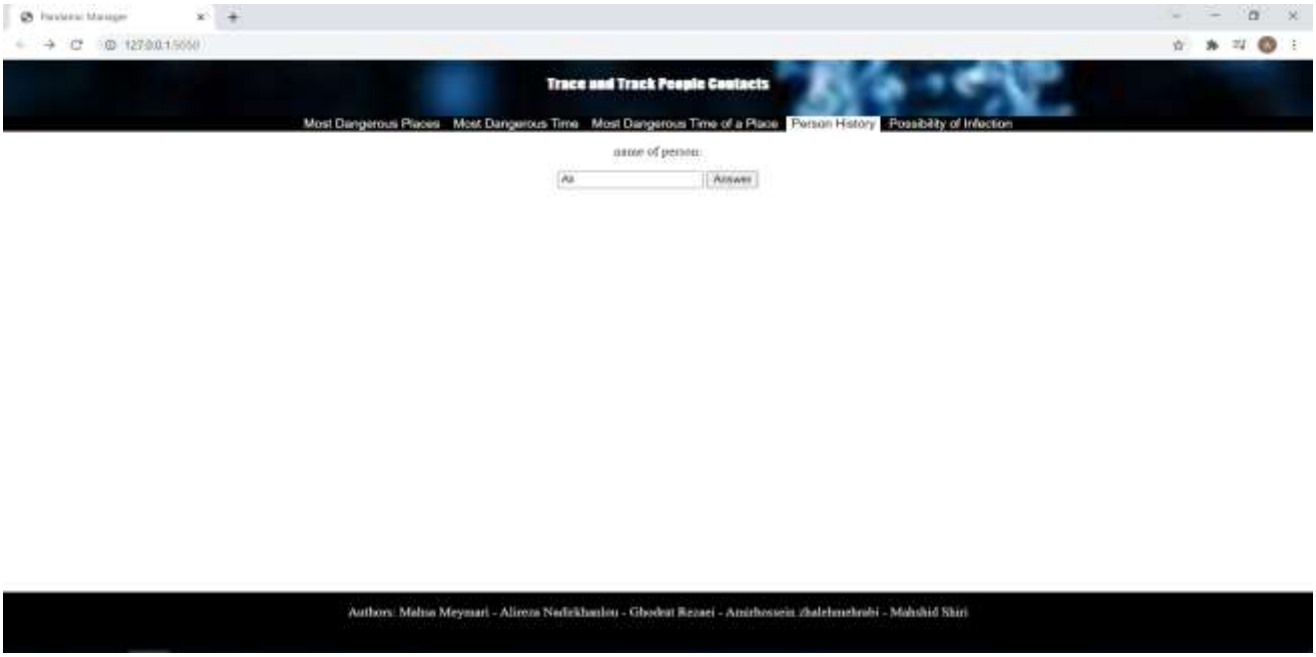


Figure 6

So the result will be shown by clicking on answer (Figure 7).

Figure 7

The fifth button checks possibility of infection. The person's name should be entered in the box and then Answer button should be clicked (Figure 8)

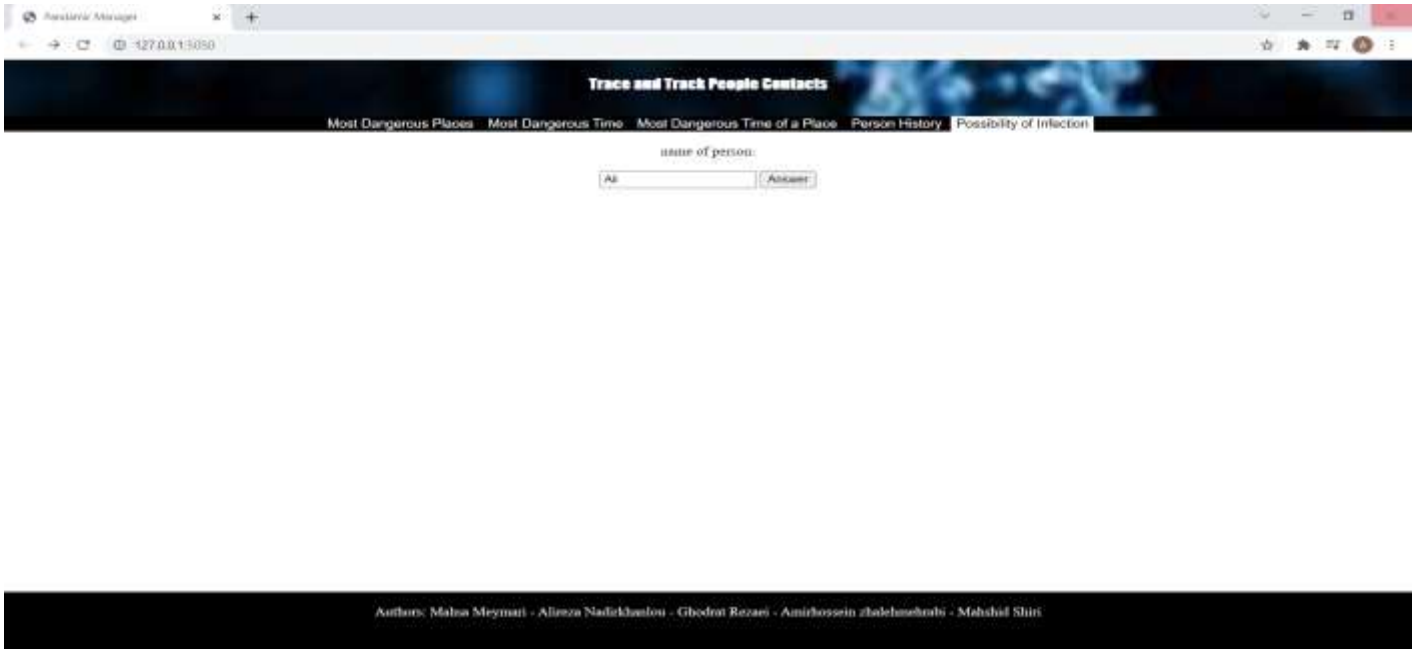


Figure 8

After giving the person name. The result of this query will be shown (Figure 9)

Figure 9

Commands:
Commands are for the admin of the system and can be accessed using API's. the list of API's and their inputs are like below:

1. "/update_test_result/<string:test_result>&<string:person_name>"
2. "/update_history_graph/<string:place_name>&<string:time>&<string:type_of_inc>"
3. "/contact_occurs/<string:place_name>&<string:time>&<string:type>&<string:list_of_individuals>"

There are two more commands but they are not accessed directly. They are used in these three commands.

Web Application

The web application uses the Flask framework in the backend and the Neo4j Python library in order to connect to the database and run the queries. The service connects to the database using the address and credentials previously provided in a text file. At the moment, it is advised that you run the application only on your localhost.

Appendix

COMMANDS:

Contact_occurs:

```
Match(p:Person)-[b:BelongsTo]->(g_p:Group) where p.name ="PERSON1"
Merge(pl: Place{name:"PLACE",time:"TIME"})on create set pl.name="PLACE",
pl.time="TIME", pl.type="TYPE"
Merge (g_p)-[x:HasGoneTo]->(pl)
match(p:PlaceHistory)-[h:HistoryArc]->(t:TimeHistory) where p.name = "PLACE" and
t.type = "TIME" set h.total = h.total + 1 return p,h,t
```

Find_dangerous_contact:

```
match (p:Place) where date() > p.ExDate detach delete p
Match(p:Person)-[b:BelongsTo]->(g_p:Group) where p.name = "NAME"
Match (g_p)-[x:HasGoneTo]->(pl:Place)
optional Match (group_first_type:Group)
where (group_first_type)-[:HasGoneTo]->(pl) and group_first_type.test = "positive"
optional Match(placeOfInfection1:Place)
```

```
where (group_first_type)-[:HasGoneTo]->(placeOfInfection1) and (g_p)-[:HasGoneTo]-
>(placeOfInfection1)
optional Match(group_in_same_place: Group)
where (group_in_same_place)-[:HasGoneTo]->(pl)
optional Match (group_in_same_place)-[:HasGoneTo]->(pl_second_type:Place)
optional Match(group_second_type:Group)
where group_first_type IS NULL and (group_second_type)-[:HasGoneTo]-
>(pl_second_type) and group_second_type.test = "positive"
optional Match(placeOfInfection2:Place)
where (group_second_type)-[:HasGoneTo]->(placeOfInfection2) and
(group_in_same_place)-[:HasGoneTo]->(placeOfInfection2)
return placeOfInfection1, placeOfInfection2
```

delete_expired_contacts:

```
match (p:Place) where date() > p.ExDate detach delete p
```

Update_test_result:

```
match(p:Person) ,(p)-[b:BelongsTo]->(g:Group) where p.name ="NAME" set
p.test="positive" ,g.test="positive"return p,b,g
match (p:Place) where date() > p.ExDate detach delete p
Match(p:Person)-[b:BelongsTo]->(g_p:Group) where p.name = "NAME"
Match (g_p)-[x:HasGoneTo]->(pl:Place)
optional Match (group_first_type:Group)
where (group_first_type)-[:HasGoneTo]->(pl) and group_first_type.test = "positive"
optional Match(placeOfInfection1:Place)
where (group_first_type)-[:HasGoneTo]->(placeOfInfection1) and (g_p)-
[:HasGoneTo]->(placeOfInfection1)
optional Match(group_in_same_place: Group)
where (group_in_same_place)-[:HasGoneTo]->(pl)
optional Match (group_in_same_place)-[:HasGoneTo]->(pl_second_type:Place)
optional Match(group_second_type:Group)
where group_first_type IS NULL and (group_second_type)-[:HasGoneTo]-
>(pl_second_type) and group_second_type.test = "positive"
optional Match(placeOfInfection2:Place)
where (group_second_type)-[:HasGoneTo]->(placeOfInfection2) and
(group_in_same_place)-[:HasGoneTo]->(placeOfInfection2)
return placeOfInfection1, placeOfInfection2
```

Update_history_graph:

```
match(p:PlaceHistory)-[h:HistoryArc]->(t:TimeHistory) where p.name = "PLACE" and t.type
= "TIME" set h.total = h.total + 1 return p,h,t
```

QUERIES:

Most dangerous places:

```
match (p:PlaceHistory)-[h:HistoryArc]->(t:TimeHistory) return
p,sum(h.infection)*1.0/sum(h.total) as prob order by prob DESC
```

Most dangerous times:

```
match (p:PlaceHistory)-[h:HistoryArc]->(t:TimeHistory) where t.type ="morning" return sum(h.infection)*1.0/sum(h.total) as prob
match (p:PlaceHistory)-[h:HistoryArc]->(t:TimeHistory) where t.type ="afternoon" return sum(h.infection)*1.0/sum(h.total) as prob
match (p:PlaceHistory)-[h:HistoryArc]->(t:TimeHistory) where t.type ="night" return sum(h.infection)*1.0/sum(h.total) as prob
```

Most dangerous time of a place:

```
match (p:PlaceHistory)-[h:HistoryArc]->(t:TimeHistory) where p.name ="PLACE" and t.type ="morning"return sum(h.infection)*1.0/sum(h.total) as prob
match (p:PlaceHistory)-[h:HistoryArc]->(t:TimeHistory) where p.name ="PLACE" and t.type ="afternoon"return sum(h.infection)*1.0/sum(h.total) as prob
match (p:PlaceHistory)-[h:HistoryArc]->(t:TimeHistory) where p.name ="PLACE" and t.type ="night"return sum(h.infection)*1.0/sum(h.total) as prob
```

Possibility of user's infection:

```
Match(p:Person)-[b:BelongsTo]->(g_p:Group) where p.name = "NAME"
Match (g_p)-[x:HasGoneTo]->(pl:Place)
optional Match (group_first_type:Group)
where (group_first_type)-[:HasGoneTo]->(pl) and group_first_type.test = "positive"
optional Match(placeOfInfection1:Place)
where (group_first_type)-[:HasGoneTo]->(placeOfInfection1) and (g_p)-[:HasGoneTo]->(placeOfInfection1)
optional Match(group_in_same_place: Group)
where (group_in_same_place)-[:HasGoneTo]->(pl)
optional Match (group_in_same_place)-[:HasGoneTo]->(pl_second_type:Place)
optional Match(group_second_type:Group)
where group_first_type IS NULL and (group_second_type)-[:HasGoneTo]->(pl_second_type) and group_second_type.test = "positive"
optional Match(placeOfInfection2:Place)
where (group_second_type)-[:HasGoneTo]->(placeOfInfection2) and (group_in_same_place)-[:HasGoneTo]->(placeOfInfection2)
optional match(phist:PlaceHistory)-[hist:HistoryArc]->(thist:TimeHistory) where phist.name = placeOfInfection1.name and thist.type = placeOfInfection1.time
optional match(phist2:PlaceHistory)-[hist2:HistoryArc]->(thist2:TimeHistory) where phist2.name = placeOfInfection2.name and thist2.type = placeOfInfection2.time
return placeOfInfection1,sum(hist.infection)*1.0,sum(hist.total), placeOfInfection2, sum(hist2.infection)*1.0,sum(hist2.total)
```

History:

```
match(p:Person)-[b:BelongsTo]->(g:Group),(g)-[h:HasGoneTo]->(pl:Place) where p.name = "NAME" return pl
```