R-NAUGHT

A PROJECT REPORT

submitted by

KEVIN SEBASTIAN SANI

MAC17CS028

to

A P J Abdul Kalam Technological University

in partial fulfillment of the requirements for the award of the Degree

of

Bachelor of Technology

in

Computer Science and Engineering

Department of Computer Science and Engineering

Mar Athanasius College of Engineering

Kothamangalam, Kerala, India 686 666

FEBRUARY 2021

R-NAUGHT

A PROJECT REPORT

submitted by

KEVIN SEBASTIAN SANI
MAC17CS028

to

A P J Abdul Kalam Technological University in partial fulfillment
of the requirements for the award of the Degree

of

Bachelor of Technology
in
Computer Science and Engineering

Department of Computer Science and Engineering

Mar Athanasius College of Engineering
Kothamangalam, Kerala, India 686 666

FEBRUARY 2021

01.06.2021 External Examiner(s)

DECLARATION

I undersigned hereby declare that the project report R-NAUGHT, submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by me under supervision of Prof. Ani Sunny. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Kothamangalam

KEVIN SEBASTIAN SANI

01.06.2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MAR ATHANASIUS COLLEGE OF ENGINEERING
KOTHAMANGALAM

CERTIFICATE

This is to certify that the report entitled **R-NAUGHT** submitted by Mr. **KEVIN SEBASTIAN SANI** to the APJ Abdul Kalam Technological University in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering is a bonafide record of the project carried out by them under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Prof. Ani Sunny	Dr. Surekha Mariam Varghese	Prof. Joby George
Project Coordinator	Project Coordinator	Head of the Dept.

Internal Examiner(s)

Dept. Seal

01.06.2021

External Examiner(s)

ACKNOWLEDGEMENT

First and foremost, I sincerely thank the ‘God Almighty’ for his grace for the successful and timely completion of the project.

I express my sincere gratitude and thanks to **Dr. Mathew K**, Principal and **Prof. Joby George**, Head of the Department Computer Science and Engineering for providing the necessary facilities and their encouragement and support.

I owe special thanks to my project guide and staff-in-charge **Dr. Surekha Mariam Varghese** for her guidance, constant supervision, encouragement and support throughout the period of this project work.

I am grateful to the project coordinator and staff-in-charge **Prof. Ani Sunny** for her corrections, suggestions and sincere efforts to co-ordinate the project under a tight schedule.

I express my sincere thanks to staff members in the department of Computer Science and Engineering for helping me with the successful completion of the project.

Finally, I would like to acknowledge the heartfelt efforts, comments, criticisms, co-operation and tremendous support given to me by my dear friends during the preparation of the project and also during the presentation without whose support this work would have been all the more difficult to accomplish.

ABSTRACT

The World Health Organization has declared the outbreak of the novel coronavirus, Covid-19 as pandemic across the world. With its alarming surge of affected cases throughout the world, lockdown, and awareness (social distancing, use of masks etc.) among people are found to be the only means for restricting the community transmission. In a densely populated country like India, it is very difficult to prevent the community transmission even during lockdown without social awareness and precautionary measures taken by the people. This app mainly focuses on development of an Android application which can inform people of their probability of contraction. This Android application sends a unique code to another phone which has installed the same app via bluetooth.The app uploads the details of their contact into the Neo4j database. The application also notifies the users if they have come in contact with a covid positive patient.The app also gives details of the hotspots surrounding a person based on their location. To achieve all these functionalities, many tools, and APIs like beacon,Neo4j are used in this application. Therefore, this application can be used as a tool for creating further social awareness about the arising need of precautionary measures to be taken by the people of India.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVATIONS

GDB	Graph Database
DP3T	Decentralised Privacy-Preserving Proximity Tracing
EphIDs	Ephemeral IDs
RDBMS	Relational Database Management System
BLE	Bluetooth Low Energy

CHAPTER 1
Introduction

Currently there are several research works undergoing in the country to prevent Covid-19 cases from rising. Previously our country was importing medical kits like PPE (Personal Protection Kits), mask from outside, but now it has been successful in developing these kits. Along with taking initiatives to fight this disease, our country has also taken steps to make people aware of the disease. The news and media have a great part in creating this awareness by informing the public about the preventive measures that can keep them away from infection. Awareness among the people to carry out all the preventive measures can immensely help to reduce spread of the virus. The country has created containment zones throughout the cities wherever Covid-19 cases have been reported to prevent further spread of the virus. These containment zones have been kept isolated from the outside public to ensure no contamination occurs outside.

After more than 2 months of the lockdown, the government has relaxed some of the lockdown rules and has permitted reopening of government offices, bus and other road transportation facilities and shopping markets. People can move inside the city for work and other purposes. But people have become careless leading to more infected cases.Once a person comes in contact with another infected person it leads to cycle of new cases.Therefore, informing people about their probability of contraction can enable them to test themselves and thereby reduce the chance of community transmission.

The application further tracks the user’s location and provides notification alert if the user has entered a containment zone. The application also provides daily Covid-19 case statistics to the users to keep them updated. The application is developed on Android SDK and uses Firebase Cloud Firestore to store the location data.

1.1 Existing Apps in Google Playstore Related to Covid-19

We had conducted a brief survey on the existing apps published in Google playstore which are related to Covid-19. Efforts had been made to include most of the apps in the survey. The summary of the survey told us that the current crop of apps couldn't solve our problem what we are looking for is an efficient and easy way to find the contacts of a covid positive patient and alert them of their probability of contarction.

1.2 Project Outline

A unique user id is interchanged between two users with the same app when they come in proximity of the same bluetooth network.The bluetooth signal is broadcasted by a flutter package called beacon.While establishing the contact we retreive the humidity,temperature,contact duration and the location details of the place and send the data to the Neo4j database.The data gets stored in the Neo4j Graph Database where it is used to find out the contacts of a covid positive patient.When a person is tested positive his contacts are retrieved and their data is processed and the corresponding probability of contraction is obtained and is sent back to the users.

CHAPTER 2
Background

To date, no effective vaccine and no licensed medication for protection or cure is yet available against the global pandemic caused by COVID-19, a disease with high transmission potential that spreads through respiratory droplet infection. Contact tracing constitutes a key mechanism for breaking the chain of disease transmission by identifying, locating, and assessing contact people exposed to a COVID-19 positive case. Trained healthcare workers interview the confirmed or suspected patients to identify all their contacts during the likely period of communicability, and quarantining, testing, and isolating them in a continuous loop. Furthermore, effective contact-tracing allows the early detection of hotspots and clusters, and their sanitization and geographical containment.

Nevertheless, it is well-established that contact tracing for infectious diseases during pandemics involves several challenges. Mistrust of contact tracing personnel due to perceived stigmatization in the community may sometimes render it difficult to identify the contact , some of whom may refuse to disclose their identity and addresses. Logistic challenges in identifying the contact-persons in areas like dense-neighbourhoods, busy markets, and travel through mass public transport are evident. Contact tracing personnel can also become demotivated and inefficient if they perceive themselves to be at higher risk of contracting the disease, especially if not provided with appropriate personal protective equipment. Furthermore, during the COVID-19 pandemic, instances of health staff facing violent attacks during community surveillance activities have been occasionally reported. Finally, contact tracing in COVID-19 is complicated by the shedding of the virus even before the development of symptoms, and by some who are chronic asymptomatic carriers.

There is growing recognition that digital technology can complement contact tracing operations for infectious diseases. Moreover, these applications also enable a system of participatory disease surveillance wherein the users are expected to self-report specific symptoms to assist public health personnel in analyzing and responding to the emerging health threats.

The feasibility of implementing such technology is also driven by the exponential worldwide growth of mobile phone and information technology.

Several automated contact tracing applications have also been developed by countries globally in their efforts to combat COVID-19, which can notify the user of any potential exposure to a COVID-19 infected patient. Furthermore, the big data generated on movement patterns and syndromic mapping can catalyze the efforts of public health officials in identifying and predicting the growth of hotspots or clusters of patients through epidemiological modeling and implement effective interventions for preventing or containing the spread of the disease. Digital contact tracing also has the advantage of foolproof recording of the user's movements since it is possible
that some people who test COVID-19 positive may not recall all their known and unknown contacts, especially when traversing through public transport, visiting busy markets or when participating in large gatherings.

2.1 Beneficence and effectiveness

Contact tracing applications are designed to benefit both the user through notifications alerting them of having come in contact with an infected person and the society by enabling early identification of disease clusters and formulation of effective public health measures to prevent further disease transmission. Although the effectiveness of contact tracing applications during pandemics does not yet have a robust body of supportive evidence, this paucity is expected, considering this is the first pandemic in an era of ubiquitous digital technology exemplified by the global penetration of smartphones and mobile internet. Nevertheless, according to the government of India, based on preliminary data, it was found that among the Aarogya Setu app users who were recommended testing, 24% were found COVID-19 positive, nearly five-fold higher compared to the overall COVID19 positivity rate in the country. The app is also helping civic authorities of the country to identify the disease trends and map containment zones within their cities.

Consequently, the large-scale use of the application by a country's user is likely to be highly cost-effective if it can reasonably complement and mitigate the work involved in the labor intensive manual tracing operations. Furthermore, for countries under lockdown, such apps could be utilized as a part of their exit or unlock strategies.The potential health-related and economic cost savings on extensive use of such an app would also meet the ethical principle of justice.

2.2 Graph Database

In computing, a graph database (GDB) is a database that uses graph structures for semantic queries with nodes, edges, and properties to represent and store data. A key concept of the system is the graph (or edge or relationship). The graph relates the data items in the store to a collection of nodes and edges, the edges representing the relationships between the nodes. The relationships allow data in the store to be linked together directly and, in many cases, retrieved with one operation. Graph databases hold the relationships between data as a priority. Querying relationships is fast because they are perpetually stored in the database. Relationships can be intuitively visualized using graph databases, making them useful for heavily inter-connected data.

Graph databases are a type of NoSQL database, created to address the limitations of relational databases. While the graph model explicitly lays out the dependencies between nodes of data, the relational model and other NoSQL database models link the data by implicit connections. Relationships are a first-class citizen in a graph database and can be labelled, directed, and given properties. This is compared to relational approaches where these relationships are implied and must be reified at run-time. Graph databases are similar to 1970s network model databases in that both represent general graphs, but network-model databases operate at a lower level of abstraction and lack easy traversal over a chain of edges.

The underlying storage mechanism of graph databases can vary. Some depend on a relational engine and ”store” the graph data in a table (although a table is a logical element, therefore this approach imposes another level of abstraction between the graph database, the graph database management system and the physical devices where the data is actually stored).

Others use a key–value store or document-oriented database for storage, making them inherently NoSQL structures. As of 2017, no universal graph query language has been adopted in the same way as SQL was for relational databases, and there are a wide variety of systems, most often tightly tied to one product. Some standardization efforts have occurred, leading to multi-vendor query languages like Gremlin, SPARQL, and Cypher. In addition to having query language interfaces, some graph databases are accessed through application programming interfaces (APIs). Graph databases differ from graph compute engines. Graph databases are technologies that are translations of the relational online transaction processing (OLTP) databases. On the other hand, graph compute engines are used in online analytical processing (OLAP) for bulk analysis. Graph databases attracted considerable attention in the 2000s, due to the successes of major technology corporations in using proprietary graph databases,along with the introduction of open-source graph databases.

2.3 Overview of Neo4j as Graph Database

Neo4j is a graph database which follows NoSQL principles such as flexible schema and scaling support. Neo4j provides ACID transactions reliability, which is not a common feature for any NoSQL databases. It is multi-relational and works on the Property Graph Model. Under this model, we have nodes, relationships and properties. It comes with its own declarative query language named as Cypher and is written using the Java language. It provides a browser interface so that querying can be done using Cypher. After running of any Cypher query, the results are returned in the form of graph visualizations in the interface. The results of the query can also be shown in tabular format. The main disadvantage of graph visualization is that it is limited to 300 nodes with their relationships as per the current version of Neo4j. We can also save all the Cypher queries as favourites for future use.

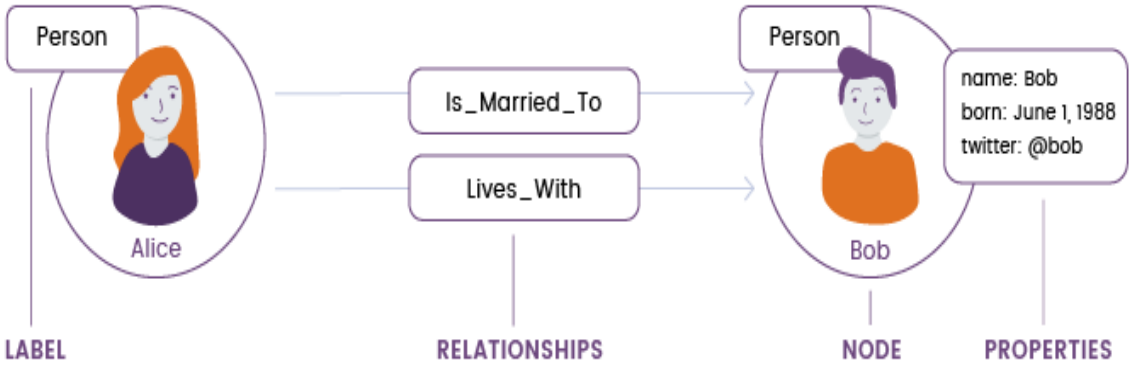

Figure1: A General Neo4j Structure

2.4 Comparison between RDBMS and Graph Databases

Neo4j In Relational Data Base Management System, data is stored using highly structured format in tables with predetermined columns of specific types and many rows of those defined type of information. The tables in the database are linked through Primary Key – Foreign Key Relationships. On the other hand, in the case of Graph Databases, data storage is highly unstructured. It is having various nodes which are connected by meaningful relationships. In relational database, we have the concept of Entity-Relationship Model whereas in graph database, we have a Graph Data Model. In relational database, the storage capacity is limited whereas in graph database, we can store any amount of data. In relational database, the performance of queries degrades with increase of multiple JOIN statements whereas in graph database, the query performance is always very fast due to index free adjacency. In relational database, we have SQL as the query language whereas in graph database, we have Cypher as the query language for Neo4j. In relational database, index scanning is done to look up rows in tables and join them with rows from other tables whereas in graph database, it uses indexes to find the starting points for a query. In relational database, intuitiveness, speed and flexibility is less as compared to graph database. After analyzing the above comparison, I can conclude that Neo4j as a graph database outperforms the relational database on many points, thus making an ideal choice for my implementation in this research.

2.5 The Cypher Language

Cypher is a declarative query language for property graphs. Cypher provides capabilities for both querying and modifying data, as well as specifying schema definitions.

2.6 Linear queries.

A Cypher query takes as input a property graph and outputs a table. These tables can be thought of as providing bindings for parameters that witness some patterns in a graph, with some additional processing done on them. Cypher structures queries linearly. This allows users to think of query processing as starting from the beginning of the query text and then progressing linearly to the end. Each clause in a query is a function that takes a table and outputs a table that can both expand the number of fields and add new tuples. The whole query is then the composition of these functions. This linear order of clauses is understood purely declaratively – implementations are free to re-order the execution of clauses if this does not change the semantics of the query. Thus, rather than declaring the projection at the beginning of the query like SQL does with SELECT, in Cypher the projection goes at the end of the query as RETURN. The linear flow of queries in Cypher extends to query composition. By using WITH, the query continues with the projected table from the query part before WITH as the driving table for the query part after WITH. The WITH clause allows the same projections as RETURN, including aggregations. It also supports filtering based on the projected fields. In addition to this linear way of composing queries, Cypher also supports nested subqueries such as UNION queries.

2.7 Pattern matching.

The central concept in Cypher queries is pattern matching. Patterns in Cypher are expressed in a visual form as “ASCII art”, such as (a)-[r]->(b). The MATCH clause in Cypher uses such a pattern and introduces new rows (synonymous with records) with bindings of the matched instances of the pattern in the queried graph. Cypher’s functionality was influenced by XPath and SPARQL .and its patterns express a restricted form of regular path queries: the concatenation and disjunction of single relationship types, as well as variable length paths (essentially, transitive closure). Cypher also supports matching and returning paths as values.

2.8 Data modification.

Cypher supports a rich update language for modifying the graph. Updating clauses re-use the visual graph pattern language and provide the same simple, top-down semantic model as the rest of Cypher. The basic clauses for updates include CREATE for creating new nodes and relationships, DELETE for removing entities, and SET for updating properties. Additionally, Cypher provides a clause called MERGE which tries to match the given pattern, and creates the pattern if no match was found. An implementation of Cypher can use database synchronization primitives such as locking to ensure that patterns matched by MERGE are unique within the database.

2.9 Neo4j implementation.

Query execution in Neo4j follows a conventional model, outlined by the Volcano Optimizer Generator. The query planning in Neo4j is based on the IDP algorithm [44, 54], using a cost model described in [21]. The final query compilation uses either a simple tuple-at-a-time iterator-based execution model, or compiles the query to Java bytecode with a push-based

execution model based on. An execution plan for a Cypher query in Neo4j contains largely

the same operators as in relational database engines and an additional operator called Expand. Semantically Expand is very similar to a relational join. It finds pairs of nodes that are connected through an edge. In terms of implementation it utilizes the fact that the data representation of Neo4j contains direct references from each node via its edges to the related nodes. that Expand never needs to read any unnecessary data, or proceed via an indirection such as an index in order to find related nodes

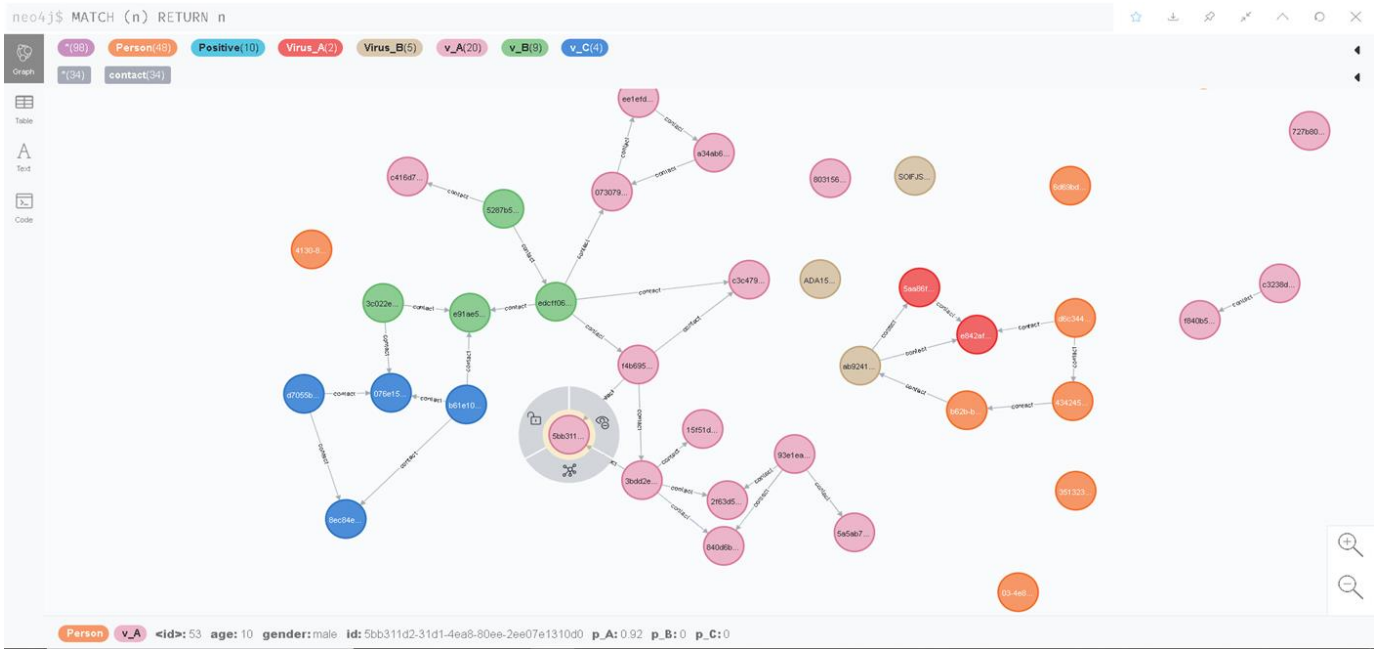

Figure2: A Basic Neo4j Graph

2.10 Python and Flask

Python is a general purpose, high level programming language that focuses on the code readability, for web development lines of code will be fewer than other languages. It is possible for Python because of large standard libraries which make the web development code simple and short. These libraries have pre-coded functions provided by Python community which can

be easily downloaded and can be used as per the development needs. Initially Python was designed for web servers to deal with the incoming traffic on the server.

Flask is a micro framework of Python which provides the basic functionality of web framework and allows more plug-ins to be added so the functionality and feature set can be extended to a new level.

Flask is called as micro framework of Python because it makes the core functionality simple but extensible in terms of development. It can also be used to save time building web applications. Flask uses Jinja Template Engine and the Werkzeug WSGI Toolkit. Flask structure is categories into two parts “Static files & Template files”, template file have all the Jinja templates including Html pages, where as static file have all static codes needed for website such as CSS code, JavaScript code and Image files.

2.11 Flutter

Flutter is a cross-platform framework that aims at developing high-performance mobile applications. Flutter is publicly released at 2016 by Google. Not only can Flutter applications run on Android and iOS, but also Fuschia, Google’s next generation operating system, chooses Flutter as its application-level framework.

Flutter is unique. Rather than utilizing web views or relying on the device’s OEM widgets, Flutter renders every view components using its own high-performance rendering engine. This nature provides possibility to build applications that are as high-performance as native applications can be. Architecture wise, the engine’s C/C++ code is compiled with Android’s NDK and LLVM on iOS respectively, and any Dart code is AOT-compiled into native code during compilation .

Flutter supports stateful hot-reload while developing, which is considered as a major factor to boost development cycle. Stateful hot-reload is essentially implemented by injecting updated

source code into the running Dart VM without changing the inner structure of the application, thus all transitions and actions of the application will be preserved after hot-reloading.

2.12 Dart

In Flutter, all applications are written with Dart. Dart is a programming language that is developed and maintained by Google. It is widely used inside of Google and it has been proved to have the capability to develop massive web applications, such as AdWords. Dart was originally developed as a replacement and successor of JavaScript. Thus, it implements most of the **import**ant characteristics of JavaScript's next standard (ES7), such as keywords "async" and "await". However, in order to attract developers that are not familiar with JavaScript, Dart has a Java-like syntax. Akin to other systems that utilize reactive views, Flutter application refreshes the view tree on every new frame. This behaviour leads to a drawback that many objects, which may live for only one frame, will be created. Dart, as a modern programming language, is optimized to handle this scenario in memory level with the help of "Generational Garbage Collection"

CHAPTER 3

Related Work

The development of contact tracing approaches is currently considered as one of the main weapons to confront the spread of the COVID-19 worldwide. Indeed, contact tracing is considered by the WHO as a key component of the infection monitoring by including contact identification, listing, and follow-up aspects [1]. So far, contact tracing has been mainly based on manual procedures, in which infected people are interviewed in an attempt to trace their contacts. Then, the health authority reaches each contact to check if they present any symptoms and advises them accordingly, e.g., get tested and/or self-quarantine. This approach is time-consuming, resource demanding, and prone to errors, since people might not remember all their contacts or, even if they do, they might not know them in person or how to contact them. To cope with these issues, digital contact tracing has emerged with different initiatives, which are currently driven by organisations and governments worldwide. The main purpose is to efficiently detect people who have been in close contact with infected individuals, so they can be promptly and properly advised on the next steps to follow. This way, potentially infected individuals can be easily detected and self-isolated even before showing symptoms. Therefore, the infection chain is interrupted as early as possible. During the last few months, numerous contact tracing frameworks and smartphone applications (apps) have emerged. These frameworks comprise the backend infrastructure and the protocols used to communicate among subsystems, whereas the apps are installed on peoples' smartphones and interact with the backend infrastructure.

3.1. Decentralised Privacy-Preserving Proximity Tracing (DP3T)

The Decentralised Privacy-Preserving Proximity Tracing (DP-3T) represents a decentralised contact tracing approach, which is driven by several international experts from academia and research institutions. The DP-3T consortium was formed by several members of the Pan-European Privacy-Preserving Proximity Tracing (PEPP-PT) initiative
(which is described in the next subsection) as a decentralised alternative, which is open source at a GitHub repository .

According to the DP-3T team , the main objectives of the system are to enable a quick notification of contact people at risk and to help epidemiologists to analyse the spread of the virus. Furthermore, the consortium has recently defined additional goals, including the communication with interested stakeholders to improve tracing systems, contributions about the effectiveness of tracing solutions, or collaboration for the development of related apps .

The DP-3T system is based on the broadcast of identifiers (IDs) through Bluetooth Low Energy (BLE) by the user's smartphone. Therefore, nearby users are enabled to receive and store such IDs. In case an infected person is detected, their smartphone is authorised to send their IDs to the backend, which in turn broadcasts the IDs to the users of the system. This way, each receiving user compares the received IDs against the list of stored IDs, and in case of an ID match, the app notifies the user that they have been in contact with an infected person.

From an architectural perspective, the DP-3T system only requires a backend server and the users' smartphones, where the corresponding app is installed. Furthermore, the existence of a health authority is assumed. Then, the following two main processes are defined:

(i) Generation and storage of ephemeral IDs (EphIDs)

(ii) (ii) Proximity tracing

3.2 Generation and Storage of Ephemeral IDs (EphIDs).

As already pointed out, the approach defines a core solution in which each smartphone broadcasts changing ephemeral IDs (EphIDs), which are sent through BLE beacons (advertisements). These IDs are generated from a secret key SKt, where t represents the current day. Furthermore, the same key is refreshed every day by using a hash function H. This is a hash chain scheme, meaning that if a key is compromised, then all the subsequent SKs are revealed, but not the SKs before it. Then, SKt is used to derive a set of EphIDs by using a pseudorandom function (PRF), say, HMAC-SHA-256, and a pseudorandom generator (PRG), say, AES in counter mode. Each EphID has a validity period of several minutes. EphIDs are received by nearby users through BLE advertisements. Then, each EphID is stored by these users together with an exposure measurement, e.g., signal attenuation, and the day when the beacon was received.

3.3 Proximity Tracing.

The process of proximity tracing illustrated in is triggered when a user is diagnosed as infected by the health authority. The latter authority is responsible for notifying test results , authorising users to upload information to the backend server, and calculating the time during a patient is contagious, also known as "contagious window." When a person is diagnosed as contagious and is authorised by the health authority, say, via an authorisation code, they upload the key SKt and the first day t that they were considered to be contagious. This information can be encoded in the authorisation code. Therefore, the backend will receive a pair (SKt, t) of each infected individual. Then, the different ðSKt, tÞ pairs are periodically downloaded by the registered users . It should be noted that the backend is only intended to broadcast this information, instead of processing any data. With this information, users are enabled to compute the list of EphIDs associated to a given ðSKt, tÞ pair. In case such an EphID is included in their stored list, it means the user was in contact with an infected person. Then, for each matching beacon, the data on receive time and exposure measurement is sent to an exposure estimation component, which is intended to estimate the duration of the smartphone owner's exposure to infected users in the past.

The previous description pertains to the DP-3T design "low-cost decentralised proximity tracing." However, the DP-3T consortium has also proposed an alternative approach called "unlinkable decentralised proximity tracing", which is intended to provide better privacy properties, but at the expense of stronger performance requirements on the smartphone side. In this case, when a user is diagnosed as infected, they can decide which IDs are shared to avoid the potential linking of EphIDs. For example, a user may choose not to share the IDs corresponding to a certain period, e.g., Sunday morning. The approach is based on the use of Cuckoo filters , in which the IDs of an infected person are hashed and stored.

Furthermore, a third alternative has been defined in the latest version of the DP-3T white paper, which integrates the advantages of the aforementioned designs. Precisely, it is called "hybrid decentralised proximity tracing" in which seeds are generated and used to create ephemeral IDs according to the first design, but these seeds are only uploaded in case they are relevant to exposure estimation for other users.

This way, protection against linking ephemeral IDs is enhanced compared to the low-cost design, but tracking protection is weaker than for the unlinkable design. It should be noted that, according to DP-3T , this design closely resembles the Google/Apple framework in which time windows are 1-day long, so one seed is used to generate the ephemeral IDs of that day.

Moreover, the DP-3T consortium has proposed an enhancement that can be applied to diverse proximity tracing systems called "EphID Spreading with Secret Sharing." The main goal of this approach is to block an adversary from recording a proximity event, even in case the contact was during a very short period of time, or when the distance is actually long among people. Therefore, such an attacker could acquire a potentially large amount of EphIDs that could be used to infer additional information about a certain user. To mitigate such an issue, the approach is based on splitting each EphID into different shares so that each share is transmitted using a certain BLE advertisement. On the downside, a potential receiver needs to get a minimum number of shares to be able to construct the corresponding EphID.

(1) Google/Apple Exposure Notification.

It should be noted that the solution proposed by Google and Apple, namely, Exposure Notification , follows a decentralised approach, which was "heavily inspired" by DP-3T, according to Google . Indeed, as already mentioned, the last version of DP-3T considers that this approach could be seen as a particular case of the "hybrid decentralised proximity tracing" design. In Exposure Notification, the pseudorandom IDs that are broadcast over BLE, namely, Rolling Proximity Identifiers (RPIs), are generated in a similar way as in DP-3T: a temporary exposure key, which is changed every day, is used to derive the RPIs employing a hash function and the AES algorithm. The RPIs are renewed every time the BLE randomised address is changed,

namely, about every 15 min, to prevent linkability and wireless tracking. Whenever a user is diagnosed as positive to COVID-19, they share the latest temporary exposure keys, e.g., covering the most recent 14 days, with a central server.

The mechanism followed by an infected user to report the collected temporary IDs will be determined by their public health authority, say, by using a one-time password, but this is not specified by the protocol.

The server aggregates all the received temporary exposure keys, and the users of the Exposure Notification system periodically download this list of keys. If a user is never diagnosed as positive, their temporary exposure keys do not leave the smartphone. The deployment of Exposure Notification has already started on May 25, 2020, with a large-scale pilot in Switzerland..

3.4 Pan-European Privacy-Preserving Proximity Tracing (PEPP-PT)

The Pan-European Privacy-Preserving Proximity Tracing (PEPP-PT) is a digital proximity-tracing framework that uses BLE advertisements to discover and locally log to a user's smartphone other users that are in close proximity.

According to its designers [17], it notifies people at risk with a 90% true-positive and 10% false-negative rates. Initially, many different systems following either the centralised Wireless Communications and Mobile Computing 3 or decentralised approach were participating under this initiative, including DP-3T, whose partners eventually resigned from the PEPP-PT consortium. PEPP-PT refers to two very similar centralised systems, the PEPP-PT NTK [18] and ROBERT [19]. These systems employ a centralised reporting server to process contact logs and individually notify clients of potential contact with an infected patient. The PEPP-PT system comprises the following components:

(i) A user mobile app for proximity tracing

(ii) A backend server for generating temporary IDs used with the app and processing the data received by the app

(iii) A push notification service (ROBERT does not include this component because it follows a pure pull approach where the app regularly checks the infection status of its user, in contrast to NTK where the backend requests from a subset of all users to check their status.) to trigger the app to pull notification from the backend.

3.5 Other Frameworks.

While the previous frameworks represent the main contact tracing approaches nowadays, additional solutions have been recently proposed, and they are described below.

BlueTrace .

This framework represents the approach used by the TraceTogether app ,which was initially developed by Singapore's Government Technology Agency and

Ministry of Health. The approach follows a centralised solution in which users register their phone numbers in the backend service, which provides random IDs that are associated with such numbers. These IDs are used during smartphones' encounters.

However, in case a user is infected, they will be authorised to share their encounter history with the health authority, which in turn can obtain the phone number of the infected person and the number of people who were in contact with them.

Therefore, based on the BlueTrace design, the backend service is able to access users' personally identifiable information.

TraceSecure .

In this case, two alternative solutions are proposed. The first is based on the framework provided by the TraceTogether app and employs public key cryptography. Specifically, the authors define two versions requiring two or three separate noncolluding parties who administer the system that can be associated to health authorities or other government services. The second approach is based on the use of homomorphic encryption by leveraging the ability of the parties to exchange secrets for the sake of providing advanced privacy features.

DESIRE.

This system has been recently proposed by INRIA and integrates different aspects of centralised and decentralised models. Specifically, DESIRE follows the ROBERT architecture in which risk scores and notifications are handled by a server. Nevertheless, the approach relies on the concept of Private Encounter Tokens (PETs), which are privately generated by users and thus are unlinkable.

The server is intended to match the PETs provided by infected users with the PETs of requesting people. Furthermore, the information hosted by the server is encrypted with cryptographic keys, which however are locally stored in the users' smartphones.

TCN.

The Temporary Contact Numbers (TCN) is a decentralised contact tracing protocol based on the exchange of 128-bit temporary IDs among nearby smartphones using BLE. These IDs are pseudorandom and generated locally on the smartphone. When a user develops symptoms or is diagnosed as infected, the app can upload a report with the collected IDs to a central portal. Users' apps connect periodically to the portal to check if their ID has been reported by an infected user. One of the characteristics of TCN is that the involvement of a health authority is optional.

The approach of the Privacysensitive protocols And mechanisms for mobile Contact 6 Wireless Communications and Mobile Computing Tracing (PACT) is mainly developed by researchers from the University of Washington and is based on the

definition of a third-party free framework for mobile contact tracing. Particularly, the authors define a set of protocols to strengthen privacy aspects by keeping users' data in their smartphones. Indeed, this approach is related to the DP-3T system, as only infected people will be enabled to share their data on a voluntary basis.

PACT (MIT)

The Private Automated Contact Tracing (PACT) protocol has been mainly developed by researchers of the Massachusetts Institute of Technology (MIT) and bears similarities with other relevant decentralised solutions like TCN, DP-3T, and PACT (UW). The user's app generates locally pseudorandom IDs, called chirps, which change every few minutes. The chirps are broadcast using BLE and stored locally in the phone for up to 3 months.

The receiving apps can store chirps for up to 3 months as well; optionally, the receiving smartphone can also store the location of the encounter. The upload of collected chirps from infected individuals to a central database is authorised by health professionals via one-time passwords. Regularly, the apps download the database locally and check if the chirps contained in the database are present in their local contact list as well.

3.6 OpenCovidTrace

This is an open-source platform following a decentralised approach. Its aim is to integrate the most popular contact tracing protocols based on BLE and provide additional features on top of them. Such protocols include DP-3T, Google/Apple Exposure Notification, and BlueTrace. This integration is envisaged to facilitate interoperability between open-source, say, DP-3T and proprietary platforms, including Google/Apple Exposure Notification and BlueTrace. OpenCovidTrace follows the original DP-3T specifications. When the Google/Apple framework is used, pseudorandom temporary IDs are generated locally on the user's smartphone, following the DP-3T approach. I

If a user reports COVID-19 symptoms, the app sends to a central server the keys used to generate the temporary IDs and the location (i.e., city) of the user in the last 14 days. Periodically, the user's app downloads the keys of users reporting symptoms from the server and information on who has been in the same area with the requesting user. If a match is found, the user is notified as potentially being at risk. The BlueTrace is not yet supported by OpenCovidTrace, but it reportedly will in its next release.

When a user is infected, the app uploads to a central server the seeds that were used to produce the temporary IDs of the last 2 weeks. No health authority is involved in certifying the infection, and no authorisation is foreseen to upload the collected temporary IDs. Each user's app is sporadically synchronised with the central server, and when there are new keys, they are downloaded and processed locally to produce the temporary IDs of infected users. If a match is found, it means that the local user has been in contact with an infected user; an algorithm to estimate the exposure risk is out of the scope of the protocol. To optimise the process of temporary ID downloading, the authors propose to include a location dimension to the uploaded data so that a user only downloads temporary IDs from infected users that have visited the same geographic area.

CHAPTER 4

Design and Implementation

4.1 ARCHITECTURE

A schematic of the general architecture of the R-Naught system is shown in Figure 1. The app automatically records close contact (i.e, defined as within the threshold distance for at least 15 minutes in the app) on Android devices by employing Bluetooth technology. The R-Naught system consists of the following major sections: mobile terminal apps for individuals (i.e, infected and potentially exposed), and a contact information sharing system. It complies with the decentralized framework; this means that the app only locally tracks close contacts and performs matching inference of exposure risk, during which no personal private information is requested or collected.

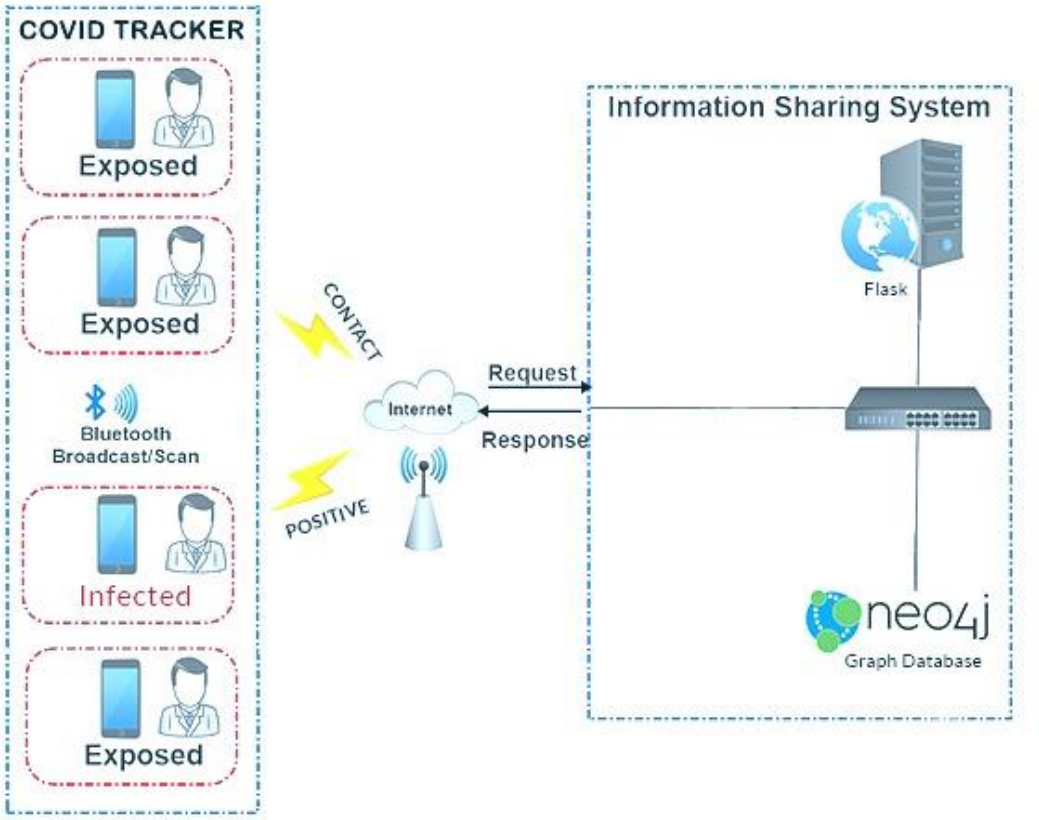

Figure3: Architecture of R-Naught System

4.2 GRAPH DATABASE

A graph database is an online database management system with Create, Read, Update and Delete (CRUD) operations working on a graph data model. The building blocks are vertices and edges. Each node represents an entity (a person, place, thing, category or other piece of data), and each relationship represents how two nodes are associated. This general-purpose structure allows you to model all kinds of scenarios – from a system of roads, to a network of devices, to a population's medical history or anything else defined by relationships. Unlike other databases, relationships take first priority in graph databases. This means your application doesn't have to infer data connections using things like foreign keys or out-of-band processing, such as MapReduce.

Graph databases didn't see a greater advantage over relational databases until recent years, when frequent schema changes, managing explosives volume of data, real-time query response time, and more intelligent data activation requirements paved way to realize the advantages of the graph model.

We chose graph database over other data management systems because of the following advantages it has:

4.2.1Performance

They have superior performance for querying related data, big or small. A native graph has the so-called index-free adjacency property, where each vertex maintains its neighbor vertices information only, no global index about vertex connections exists. This makes native graph exhibit constant performance while data size grows. The performance is constant since that traversing a given vertex's neighbour nodes via edges are independent of the graph size. It never needs to load or touch unrelated data for a given query. They're an excellent solution for

real-time big data analytical queries where data size grows rapidly.

4.2.2 Update Data in Real-Time and Support Queries Simultaneously

Graph databases can perform real-time updates on big data while supporting queries at the same time. This is a major drawback of existing big data management systems such as Hadoop HDFS since it was designed for data lakes, where sequential scans and appending new data (no random seek) are the characteristics of the intended workload, and it is an architecture design choice to ensure fast scan I/O of an entire file. The assumption there was that any query will touch the majority of a file, while graph databases only touch relevant data, so a sequential scan is not an optimization assumption.

4.2.3 Flexible Schema

Graph databases offer a flexible schema evolvement while serving your query. We can constantly add and drop new vertex or edge types or their attributes to extend or shrink your data model. Most of the existing queries will still work. It is so convenient to manage explosive and constantly changing object types. The relational database just cannot easily adapt to this requirement, which is commonplace in the modern data management era.

4.2.4 Object-Oriented Thinking

This means very clear, explicit semantics for each query you write. There are no hidden assumptions, such as relational SQL where you have to know how the tables in the FROM clause will implicitly form cartesian products. Besides ease-of-use, such as regular path pattern matching, accumulator concepts allows fine control to keep mid-way query state in-place of the data.

4.2.5 Make Powerful Recursive Path Query Easily Accessible:

Many useful, real-life queries are finding direct and indirect connections in a graph (or network of data). Answering this class of reachability queries is one of the core powers of the graph database. Further, we can extend the single-pair-vertex reachability queries with multiple reachability queries sharing some common vertices. A set of reachability path queries can be bundled together to constrain each other to form an interesting subgraph pattern. This is called conjunctive graph query (CQ). CQ allows users to come up with a subgraph pattern and asks the database to return all subgraph instances that match this pattern.

Reachability queries are notoriously hard to do in a relational database, as there is no pre-determined number of JOINs. It becomes harder if we rank the connections (paths) based on some measurement(s) of the paths. It is easy to use a regular expression to express this class of recursive path queries in the edge pattern of a graph query language. In contrast, it's very hard to do this in SQL even with the SQL 99 recursive clause.

4.3 NEO4J AS GRAPH DATABASE

Among the graph databases available, we chose neo4j because of its following features:

- Highly Performant Read and Write Scalability, Without Compromise

  Neo4j delivers the lightning-fast read and write performance, while still protecting data integrity. It is the only enterprise-strength graph database that combines native graph storage**,** scalable architecture optimized for speed, and ACID compliance to ensure predictability of relationship-based queries.

- Easier to Load Data into Neo4j

  It has a staggering loading speed of huge data sizes, with very low memory footprint and we can choose how much and which data to **import**, without

worrying about volume.

- Biggest and Most Active Graph Community
- High Performance due to Native Graph Storage & Processing

  Index-free adjacency shortens read time and gets better as data complexity grows. It provides reliably fast transactions with ultra-high parallelized throughput even as data grows.

4.4 CYPHER

Cypher is Neo4j's graph query language that allows users to store and retrieve data from the graph database. Neo4j wanted to make querying graph data easy to learn, understand, and use for everyone, but also incorporate the power and functionality of other standard data access languages. This is what Cypher aims to accomplish.

Cypher's syntax provides a visual and logical way to match patterns of nodes and relationships in the graph. It is a declarative, SQL-inspired language for describing visual patterns in graphs. It allows us to state what we want to select, insert, update, or delete from our graph data without a description of exactly how to do it. Through Cypher, users can construct expressive and efficient queries to handle needed create, read, update, and delete functionality.

4.5 FLUTTER

Flutter is Google's modern app development kit. It is a cross-platform tool intended for creating Android and iOS apps from a single code base by using a modern, reactive framework. Flutter apps are built using **Dart**, a simple object-oriented programming language. The central idea of Flutter revolves around widgets. The entire UI is made of combining different widgets, each of which defines a structural element (like a button or menu), a stylistic element (like a

font or colour scheme), an aspect of layout (like padding), and so on. Flutter does not use OEM widgets, but provides its own ready-made widgets which look native either to Android (Material Design) or iOS apps (Cupertino). It's also possible to create custom widgets.
We decided to use flutter for front-end development because of the following features and qualities that Flutter has:

- High productivity: Since Flutter is cross-platform, we can use the same code base for our iOS and Android app. This saves both time and resources.
- Great performance: Dart compiles into native code and there is no need to access OEM widgets as Flutter has its own. This means less mediated communication between the app and the platform. This contributes to fast app startup times and less performance issues.
- Fast and simple development: One of the most lauded features of Flutter is *hot reload* which allows us to instantly view the changes made in the code on emulators, simulators and hardware. In less than a second, the changed code is reloaded while the app is running with no need for a restart. This is great not just for building UIs or adding features but also for bug fixing.
- Compatibility: Since widgets are part of the app and not the platform, there are only a few or no compatibility issues on different OS versions. This in turn means less time spent on testing.

Open-source: Both Flutter and Dart are open-source and free to use, and provide extensive documentation and community support to help out with any issues we may encounter.

4.6 BLUETOOTH CONNECTION WITH FLUTTER BEACON

Flutter_beacon is a flutter plug-in to work with iBeacons. It is a hybrid iBeacon scanner and transmitter SDK for Flutter plugin. By using this plug-in, the application acts as an iBeacon and allows data sharing via Bluetooth. Along with data sharing, it also gives strength of the Bluetooth connection. With this info, we can filter out people who are above threshold distance

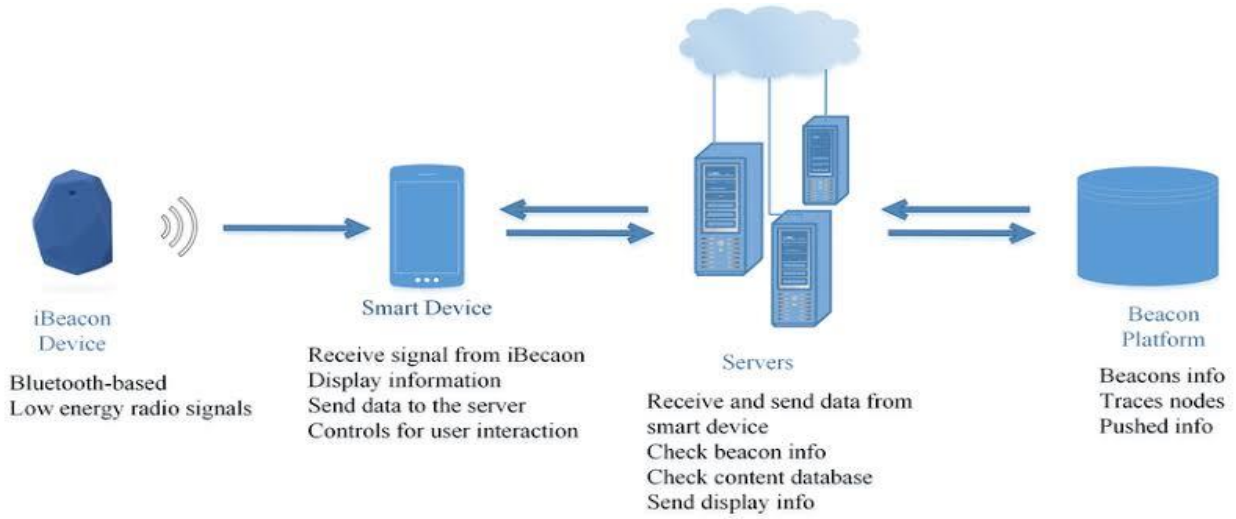

Figure5: Beacon Architecture

The main features of flutter_beacon plug-in are:

- Automatic permission management
- Ranging connected devices
- Monitoring for nearby devices
- Transmit as iBeacon

CHAPTER 5

EVALUATION

5.1 Probability Response Time

Given Below is a table predicting the response time of the app based on the number of contacts for a positive node.

Contact Number	Response Time(ms)
1st Contact	1
2nd Contact	1
3rd Contact	2
4th Contact	3
5th Contact	5

Table 1: Response Time in normal graphs

The above table suggest that as the number of contacts for a node increases the response time to obtain the probability of the node also increases. The contact tracing path increases as the number of nodes in contact increases.

Figure6: Contact Tracing

5.2 Probability Response Time With Respect To Cycles

In the Neo4j Graph database when there is the presence of cycles it is different to calculate the probability and the assign the paths of contraction so there will also be a difference in the response time.

The table below shows the response time for a cyclic graph present in Neo4j. For a cycle to exist in neo4j only two nodes are required.

Contact Number	Response Time(ms)
1st Contact	1
2nd Contact	3
3rd Contact	5
4th Contact	8
5th Contact	10

Table 2: Response Time in fully connected Graphs

Due to the cyclic nature of the graph the retrieval of contacts and probability code will become more complex, due to the complex nature of the code the response time is a little bit higher compared to previous cases..

5.3 Time to Query

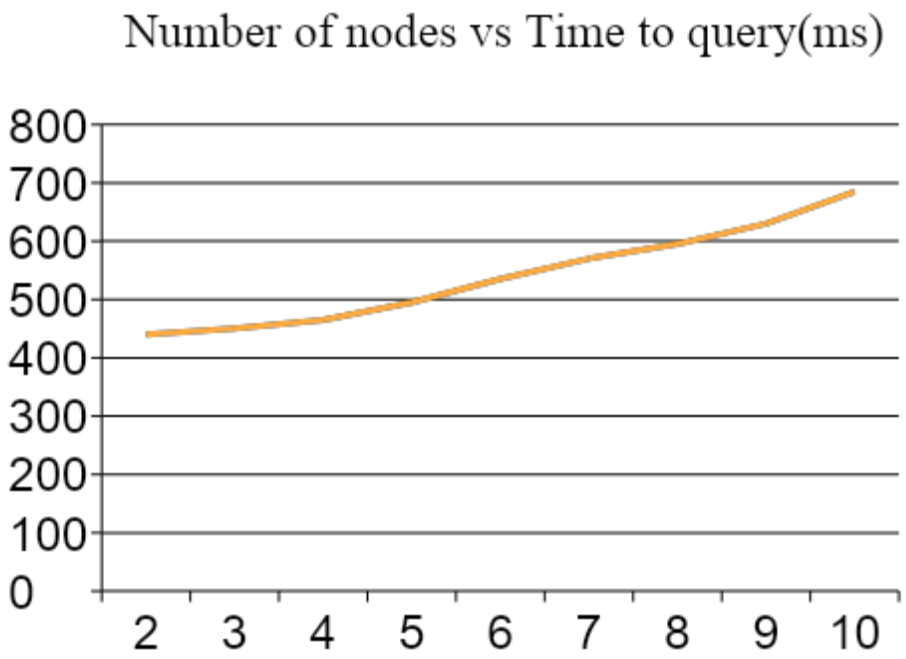

Figure7: Time to query

The above graph was obtained by teaking the case of a fully connected graph where all the nodes are connected to each other. This is the worst case scenario for a graph where the number of nodes and relationships are of maximum value. Due to this the cypher code would be complex leading to more time needed to query the data .From the given data the above graph was plotted.

From the graph we can see that as the number of nodes and relationships increase the time to query will also increase.

5.4 Efficiency Compared to other Alternatives

This app is more efficient compare to other apps due to its higher response time and efficient method to map data and pave path of the contacts of a covid positive patient. The privacy of the app also makes an essential feature which makes it unique compared to other alternatives.

Other apps like Arogya Setu, NHS-France, Covid Safe, etc all have a massive drawback. They don't map data which is a clear disadvantage when you are trying to trace the contacts of a covid positive patient. Privacy is also another concern which is not provide by the above apps. Our app does not require any data of the person. The response time is also significantly lower compared to other apps

Features	**R-Naught**	**Other Apps**
Contact Tracing	Up to 5 levels	Only one level
Source Identification	Yes	No
Tracing Different Variants of Virus	Yes	No
Storage	Graph Database	Local Storage
Data Collection	No privacy concern	Privacy concern

Table3: Comparison with other apps

The above table signifies that R-naught has taken a unique and modern approach to contact tracing which is completely different compared to other alternatives which uses inefficient methods.

CHAPTER 6
CONCLUSION

The conclusion that can be drwn here is the app R-Naught can completely change the pandemic once it’s deployed.The app is proposed to reduce the spread of the virus and to create awarness among the people.This app is the step forward in the right direction to control and reduce the pandemic.Tracking of patient clusters is done easily and the app is highly secure and private compared to other alternatives.It’s simple and effective userface enables to use it much easily compared to other apps.

The app can run on any smartphone due to its front end flutter acts as a cross platform which enables it to run on any environment. The consumes less battery life comapred to other apps. Due to the use of graph databse as the storage database it makes it easy to store and retrieve data. The alert provided by the app creates awarness among the public so the people can quarantine themselves or go get tested if they are covid positive.

This can lead for a complete control over the spread of the virus. The aim of this app is to pave path to a corona free world.

CHAPTER 7

REFERNCES

1. WHO, Contact Tracing, 2017, https://www.who.int/newsroom/q-a-detail/contact-tracing.

2. DP-3T, "DP-3T, decentralized privacy-preserving proximity tracing," https://github.com/DP-3T.

3. PEPP-PT,"Pan-European privacy-preserving proximity tracing," https://www.pepp-pt.org.
C. Troncoso, M. Payer, J. P. Hubaux et al., "Decentralize privacy-preserving proximity tracing,"

4. B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: practically better than bloom," in CoNEXT '14: Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, pp. 75–88, Sydney, Australia, December 2014.

5. Apple, ENExposureConfiguration, 2020, May 2020, https://developer.apple.com/documentation/exposurenotification/enexposureconfiguration.

6. Google/Apple, Privacy-Preserving Contact Tracing - Apple and Google, 2020, https://www.apple.com/covid19/contacttracing.

7. E. Barraud, "First pilot for the Google and Apple-based decentralised tracing app," 2020, May 2020, https://actu.epfl.ch/news/first-pilot-for-the-google-and-apple-baseddecentr

8. Y. Gvili, "Security analysis of the COVID-19 contact tracing specifications by Apple Inc. and Google Inc.," Tech. Rep., 2020, May 2020, http://eprint.iacr.org/2020/428.

CHAPTER 8

ANNEXURES

Server

rnaught.py

==========

```
import uuid
from flask import Flask,request,jsonify
from neo4j import GraphDatabase, basic_auth
from datetime import datetime
import json
from collections import defaultdict
def current_time():
   now = datetime.now()


   # dd/mm/YY H:M:S
   dt_string = now.strftime("%d %m %Y %H %M %S")
   dt_string=list(map(int,dt_string.split(" ")))

   ltime=dt_string[-2]+dt_string[-3]*60 +
dt_string[0]*3600+dt_string[1]*3600*30+dt_string[2]*365*3600
   return ltime
def find_probability(a,virus_type):
   return .92/int(a[1])



   #to do:consider other factors to find probability

app=Flask(__name__)
people={}
id_new=0
driver = GraphDatabase.driver(
  "bolt://3.238.138.75:7687",
  auth=basic_auth("neo4j", "gloves-blueprint-badge"),max_connection_lifetime=200)
def runquery(q):
  session=driver.session()
  fr=session.run(q)
  return fr.data()
def findsource(user_id,virus_type):
```

```
    source=None
    query="CALL apoc.export.json.query("
    q="MATCH(a:Person { "+f"id: '{user_id}' "+"}),(b:"+f"{virus_type}),p =
shortestPath((a)-[r:contact*]-(b)) return r"
    query="\"%s\""%q
    query="CALL apoc.export.json.query("+query+",null,{"+"stream:true})YIELD data
"
    print(query)

    fr=runquery(query)[0]['data']
    if len(fr)==0:
      return None
    fr=fr.split("\n")
    fr=min(fr, key = len)
    fr=json.loads(fr)
    fr=fr['r']
    q="match(n{id:"+f"'{user_id}'"+"}) return id(n)"
    val=runquery(q)[0]['id(n)']
    visited=set()

    visited.add(str(val))
    prev={}
    i=fr[0]
    contact_time=(i['properties']['contact_times']).split("\n")[-1]
    if i['start']['id'] in visited:

      prev[i['end']['id']]=int(contact_time.split(":")[0])
      visited.add(i['end']['id'])
    else:
      prev[i['start']['id']]=int(contact_time.split(":")[0])
      visited.add(i['start']['id'])
    for i in fr[1:]:
      contact_time=int(((i['properties']['contact_times']).split("\n")[0]).split(":")[0])
      if i['start']['id'] in visited:
        end_id=i['end']['id']
        visited.add(i['end']['id'])
        start_id=i['start']['id']
      else:
        end_id=i['start']['id']
        visited.add(i['start']['id'])
        start_id=i['end']['id']
      if contact_time<=prev[start_id]:
        prev[end_id]=int(((i['properties']['contact_times']).split("\n")[-1]).split(":")[0])
      else:
        source=end_id
        return source
```

```
    return source
def five_level(user_id,virus_type,is_source):
    val=1
    s_prob="u.p_"+virus_type[-1]
    v_type="u."+virus_type

    q=f"match(u) where u.id='{user_id}' return labels(u)"
    fr=runquery(q)[0]['labels(u)']
    temp=":".join(fr)
    temp="u:"+temp
    if is_source:
        val=.5
    q=f"match(u) where u.id='{user_id}' remove {temp} set
u:Positive:{virus_type}:Person set {s_prob}={val}"
    runquery(q)



    query="CALL apoc.export.json.query("
    q="MATCH p=({id:"+f"'{user_id}'"+"})-[r:contact*..5]-(fr) return
relationships(p),length(p),nodes(p)"
    query="\"%s\""%q
    query="CALL apoc.export.json.query("+query+",null,{"+"stream:true})YIELD data
"
    print(q)
    fr=runquery(query)
    fr=fr[0]['data']
    if fr:
        fr=fr.split("\n")
        fr=list(map(json.loads,fr))
        dat=[]
        visited=set()
        prev_times={}
        for i in fr:
            dat.append((i['relationships(p)'][-1],i['length(p)']))
        dat.sort(key=lambda x:x[1])
        for i in dat:
            if i[1]>1:
                if i[0]['start']['id'] in visited and i[0]['end']['id'] in visited:#checks if already
in previous lvl
                    continue
                if i[0]['start']['id'] in prev_times :
                    id_start=i[0]['start']['id']
                    id_end=i[0]['end']['id']                    #finds the id of start node and end
node
                elif i[0]['end']['id'] in prev_times:
                    id_start=i[0]['end']['id']
```

```
            id_end=i[0]['start']['id']
        else:
            continue
        visited.add(id_end)
        first_contact_times=i[0]['properties']['contact_times'].split("\n")
        temp=[]
        for p in first_contact_times:
            temp.append(int(p.split(":")[0]))
        flag=0
        first_contact_times=temp

        for j in range(len(first_contact_times)):
            if first_contact_times[j]>=prev_times[id_start]:
                contact_time=first_contact_times[j]
                flag=1
                break
        if flag:
            prob=find_probability(i,virus_type)
            q=f"match(u) where id(u)={id_end} return labels(u),{s_prob}"
            q_res=runquery(q)[0]

            prob_end=q_res[s_prob]
            if prob>prob_end:
                prev_times[id_end]=contact_time
                q=f"MATCH (u:Person) WHERE id(u)={id_end} SET {s_prob}={prob} set
u:{virus_type}"
                runquery(q)

    else:
        first_contact_times=i[0]['properties']['contact_times'].split("\n")
        id_start=i[0]['start']['id']
        id_end=i[0]['end']['id']
        visited.add(id_end)
        visited.add(id_start)
        q=f"match(u) where id(u)={id_start} return labels(u),{s_prob}"
        q_res=runquery(q)[0]
        labels_start=q_res['labels(u)']
        print(q_res,s_prob)
        prob_start=q_res[s_prob]
        q=f"match(u) where id(u)={id_end} return labels(u),{s_prob}"
        q_res=runquery(q)[0]
        labels_end=q_res['labels(u)']
        prob_end=q_res[s_prob]

        if current_time()-int(first_contact_times[-1].split(":")[0])<14*3600 and
prob_start!=prob_end:
```

```
            prob=find_probability(i,virus_type)

            if prob_start<prob_end and prob_start<prob:
                prev_times[id_start]=int(first_contact_times[-1].split(":")[0])
                q=f"MATCH (u) WHERE id(u)={id_start} SET {s_prob}={prob} SET
u:{virus_type}"
                runquery(q)
            elif prob_end<prob:
                prev_times[id_end]=int(first_contact_times[-1].split(":")[0])

                q=f"MATCH (u) WHERE id(u)={id_end} SET {s_prob}={prob} SET
u:{virus_type}"
                runquery(q)
    return jsonify(201)
@app.route("/register",methods=['POST'])
def register_user():
    p_A=0
    p_B=0
    p_C=0
    data_recieved =request.data

    data_recieved=json.loads(data_recieved.decode("utf-8"))
    age,gender=data_recieved['age'],data_recieved['gender']
    user_id = uuid.uuid4()

    query=f" id:'{user_id}', age: {age},gender: '{gender}',p_A: {p_A},p_C: {p_C},p_B:
{p_B}"
    query="CREATE (n:Person {"+query+"})"
    session=driver.session()
    session.run(query)
    return jsonify({"user_id":user_id})
@app.route("/new_contact",methods=['POST'])
def new_contact():
    data_recieved =request.data
    data_recieved=data_recieved.decode("utf-8")

    data_recieved=json.loads(data_recieved)


user_id,connected_ids,temperature,humidity=data_recieved['user_id'],data_recieved[
'connections'], data_recieved['temperature'],data_recieved['humidity']



    for m in connected_ids:
        duration=connected_ids[m]
```

```
        m = m.lower()
        query="MATCH (x{id:"+f"'{user_id}'"+"})-[r:contact]-(z{id:"+f"'{m}'"+"})
RETURN properties(r)"
        session=driver.session()
        fr=session.run(query)
        fr= fr.data()
        if fr:

            fr=fr[0]['properties(r)']['contact_times']
            fr=fr.split("\n")
            fr.append(f"{current_time()}:{duration}")
            contact_time_list="\n".join(fr)
            q="MATCH  (:Person {id:"+f"'{user_id}'"+"})-[r:contact]-(:Person
{id:"+f"'{m}'"+"}) Set r.contact_times="+f"'{contact_time_list}'"
            session=driver.session()
            session.run(q)
        else:
            contact_time_list=f"{current_time()}:{duration}"
            query=f"MATCH (a:Person), (b:Person) WHERE a.id ='{user_id}'AND b.id =
'{m}' CREATE (a)-[r:contact " +"{"+f"dur:0,
humidity:'{humidity}',temperature:'{temperature}',contact_times:'{contact_time_list
}'"+"}]->(b)"
            session=driver.session()
            session.run(query)

    return jsonify(200)
@app.route("/probability",methods=['POST'])
def check_probability():
    val=0
    virus_type=None
    data_recieved =request.data
    data_recieved=json.loads(data_recieved.decode("utf-8"))
    user_id=data_recieved['user_id']

    q="match(n{id:"+f"'{user_id}'"+"}) return n.p_B,n.p_A,n.p_C,labels(n)"
    fr=runquery(q)
    virus_type={}
    if fr:
        fr=fr[0]

        p_A,p_B,p_C=int(fr['n.p_A']*100),int(fr['n.p_B']*100),int(fr['n.p_C']*100)
        print(p_A,p_B)
        for i in fr['labels(n)']:
            print(i)
            if i=="Positive" or i=="Person":
                continue
```

```
            if i=="v_A":
                virus_type["Virus A"]=p_A
            elif i=="v_B":
                virus_type["Virus B"]=p_B
            else:
                virus_type["Virus C"]=p_C

    if len(virus_type)==0:
        virus_type={"No Infection": 0}
    return jsonify(virus_type)

@app.route("/positive",methods=['POST'])
def is_positive():
    data_recieved =request.data
    data_recieved=json.loads(data_recieved.decode("utf-8"))
    user_id,virus_type=data_recieved['user_id'],data_recieved['virus_type']# SEnd
virus type as v_A, v_B or v_C
    if virus_type=="v_B":
        ret="a.p_B"
    elif virus_type=="v_A":
        ret="a.p_A"
    else:
        ret="a.p_C"
    query=f"MATCH (a:Person) WHERE a.id='{user_id}' Return {ret}"
    val=runquery(query)
    source=None
    if val:
        val=val[0]
        val=val[ret]
        if val<.05:
            source=findsource(user_id,virus_type)
        if source!=None:
            q=f"match (n) where id(n)={source} return n.id"
            val=runquery(q)[0]['n.id']
            five_level(val,virus_type,1)
    return five_level(user_id,virus_type,0)
@app.route("/police",methods=['POST'])
def police():
    d={}
    data_recieved =request.data
    data_recieved=json.loads(data_recieved.decode("utf-8"))
    connections=data_recieved['connections']

    for i,j in connections.items():
        i = i.lower()
        if int(j)>-100:
```

```
            query="match(n{id:"+f"'{i}'"+"}) return n.p_A,n.p_B,n.p_C"
            fr=runquery(query)
            if len(fr)>0:
               fr=fr[0]
               p_A,p_B,p_C=int(fr['n.p_A']*100),int(fr['n.p_B']*100),int(fr['n.p_C']*100)
               d[i]={"Virus_A":p_A,"Virus_B":p_B,"Virus_C":p_C}# returned probability of
    each virus as dictionary
      if len(d)==0:
         d={"No Person Nearby": 0}
      print(d)
      return jsonify(d)
    app.run(debug=True,host='0.0.0.0',port=5000)
```

App

main.dart

==========

```
import 'package:flutter/material.dart';
import 'package:covid_19/router.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 // This widget is the root of your application.
 @override
 Widget build(BuildContext context) {
  // ClusterManager.createMarkers();
  return MaterialApp(
   debugShowCheckedModeBanner: false,
   theme: ThemeData(
    fontFamily: 'OpenSans',
    primaryColor: Color(0xffF9F9F9),
    hintColor: Color(0xffA7A7A7),
   ),
   title: 'Flutter Demo',
   onGenerateRoute: generateRoute,
  );
 }
}
```

pages/Registrationpage.dart

==========

```
import 'package:flutter/material.dart';
import'package:shared_preferences/shared_preferences.dart';
```

```
import 'package:covid_19/Globals.dart' as Globals;
import 'package:http/http.dart' as http;
import 'dart:convert';
class RegistrationPage extends StatefulWidget {
  @override
  _RegistrationPageState createState() => _RegistrationPageState();
}
class _RegistrationPageState extends State<RegistrationPage> {
  // Constants
  final TextEditingController _ageController = new TextEditingController();
  final TextEditingController _genderController = new TextEditingController();
  // States
  bool isLoaded = false;
  String userid;
  // Methods
  @override
  void initState() {
    super.initState();
    // Check if user is already registered,
    SharedPreferences.getInstance().then((prefs) {
      userid = prefs.getString('uid');
      if (userid != null) {
        print(userid);
        Navigator.of(context).pushReplacementNamed('/home');
      }
      setState(() {
        isLoaded = true;
      });
    });
  }
  void registerButton() {
    String gender = _genderController.text;
    int age = int.tryParse(_ageController.text);
    Map body = {
      "gender": gender,
      "age": age,
    };
    http
        .post(Globals.ip_address + 'register',
            headers: {"Content-Type": "application/json"},
            body: json.encode(body))
        .then((response) async {
      final decoded = json.decode(response.body) as Map;
      if (decoded.containsKey('user_id')) {
        String id = decoded['user_id'];
        SharedPreferences prefs = await SharedPreferences.getInstance();
```

```
    prefs.setString('uid', id);
    ScaffoldMessenger.of(context)
        .showSnackBar(SnackBar(content: Text("User Created")));
    setState(() {
     userid = id;
     Navigator.of(context).pushReplacementNamed('/home');
    });
   } else {
    ScaffoldMessenger.of(context).showSnackBar(
       SnackBar(content: Text("Registration failed. no data returned")));
   }
 });
}
@override
Widget build(BuildContext context) {
 return SafeArea(
  child: Scaffold(
     appBar: AppBar(
      title: const Text(
       "R-Naught",
       style: TextStyle(color: Colors.white),
      ),
      backgroundColor: Colors.blue,
     ),
     body: Builder(
      builder: (context) {
       if (isLoaded == true)
        return _getRegistrationForm();
       else
        return Container(); // Show nothing till data is loaded from localstorage
      },
     )),
 );
}
Widget _getRegistrationForm() {
 return Form(
  child: Padding(
   padding: EdgeInsets.all(30),
   child: Column(
    mainAxisSize: MainAxisSize.max,
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
     TextFormField(
      controller: _ageController,
      keyboardType: TextInputType.number,
      decoration: InputDecoration(hintText: "Age"),
```

```
            ),
            TextFormField(
              controller: _genderController,
              keyboardType: TextInputType.text,
              decoration: InputDecoration(hintText: "Gender"),
            ),
            Container(height: 30),
            ElevatedButton(onPressed: registerButton, child: Text("REGISTER"))
          ],
        ),
      ),
    );
  }
}
```

Pages/HomePage.dart

==========

```
import 'dart:async';
import 'package:covid_19/pages/PolicePage.dart';
import 'package:http/http.dart' as http;
import 'package:covid_19/Globals.dart' as Globals;
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:flutter_blue/flutter_blue.dart';
import 'package:location_permissions/location_permissions.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:beacon_broadcast/beacon_broadcast.dart';
import 'package:flutter_beacon/flutter_beacon.dart' as flutter_beacon;
//import 'package:geolocation/geolocation.dart';
//import 'package:weather/weather.dart';
class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}
class _HomePageState extends State<HomePage> {
  // Constants
  final FlutterBlue flutterBlue = FlutterBlue.instance;
  final BeaconBroadcast beaconBroadcast = BeaconBroadcast();
  //String myKey = '5576f874bd97fe2728f4da4e17989804';
  // Variables
  Timer _sendToServerTimer;
  StreamSubscription bluetoothStateChangeStream;
  StreamSubscription<flutter_beacon.RangingResult> _streamBeaconRanging;
  double lat;
```

```
double long;
// States
bool isBluetoothOn = false;
bool isLocationOn = false;
bool isAdvertising = false;
String userid;
Map<String, int> recentlyRecievedUUIDS = {};
Set<String> recentlySentUUIDs = new Set<String>();
Map<String, String> selectedUUIDS = {};
Map<String, int> closerUUIDS = {};
Map<String, int> temp1UUIDS = {};
Map<String, int> temp2UUIDS = {};
// Methods
@override
void initState() {
 super.initState();
 initApp();
}
void initApp() async {
 PermissionStatus permission =
    await LocationPermissions().checkPermissionStatus();
 if (permission == PermissionStatus.granted ||
    permission == PermissionStatus.restricted) {
  setState(() {
   isLocationOn = true;
  });
 } else {
  await LocationPermissions().requestPermissions();
  initApp();
 }
 SharedPreferences prefs = await SharedPreferences.getInstance();
 String uuid = prefs.getString('uid');
 setState(() {
  userid = uuid;
 });
 // Wait for bluetooth to turn on
 bluetoothStateChangeStream =
    FlutterBlue.instance.state.listen((bluetoothState) {
  if (bluetoothState == BluetoothState.on) {
   setState(() {
    isBluetoothOn = true;
   });
   startBeaconServices(uuid);
  } else {
   setState(() {
    isBluetoothOn = false;
```

```
    });
    stopBeaconServices();
   }
  });
  // Send UUIDS to server on repeated intervals
  _sendToServerTimer = new Timer.periodic(Duration(seconds: 30), (timer) {
   durationApproximation();
  });
}
void startBeaconServices(uuid) {
 startBeaconAdvertising(uuid);
 listenForBeacons();
}
void stopBeaconServices() {
 beaconBroadcast.stop();
}
void getProbability() {
 Map body = {
  "user_id": userid,
 };
 print("Sending PROBABILITY data to server");
 print(json.encode(body));
 http
    .post(Globals.ip_address + 'probability',
      headers: {"Content-Type": "application/json"},
      body: json.encode(body))
    .then((response) async {
  final decoded = json.decode(response.body) as Map<String, dynamic>;
  print("probability");
  print(decoded);
  final Map<String, int> probabilities = Map.from(decoded);
  print(probabilities);
  if (probabilities.containsKey("No Infection")) {
   noDisease();
  } else {
   showProbability(probabilities);
  }
 }).catchError((e) {
  ScaffoldMessenger.of(context)
     .showSnackBar(SnackBar(content: Text("Can't connect to server")));
 });
}
Future<void> noDisease() async {
 return showDialog<void>(
  context: context,
  barrierDismissible: false, // user must tap button!
```

```
    builder: (BuildContext context) {
     return AlertDialog(
      title: Text('Covid Infection Probability'),
      content: Text('You are not at risk of any infection'),
      actions: <Widget>[
       TextButton(
        child: Text('Close'),
        onPressed: () {
         Navigator.of(context).pop();
        },
       ),
      ],
     );
    },
   );
}
Future<void> showProbability(Map<String, int> values) async {
  return showDialog<void>(
    context: context,
    barrierDismissible: false, // user must tap button!
    builder: (BuildContext context) {
     return AlertDialog(
      title: Text('Infection Probability'),
      content: Text(
         'You have following chance of having each disease: \n $values'),
      actions: <Widget>[
       TextButton(
        child: Text('Close'),
        onPressed: () {
         Navigator.of(context).pop();
        },
       ),
      ],
     );
    },
  );
}
void iamCovidPositive() {
  verification(context);
}
Future<void> verification(BuildContext context) async {
  TextEditingController verifyFieldController1 = TextEditingController();
  TextEditingController verifyFieldController2 = TextEditingController();
  String valueCode;
  String valueType;
  String docCode;
```

```
String typeVirus;
return showDialog(
   context: context,
   barrierDismissible: false,
   builder: (context) {
    return AlertDialog(
     title: Text('Infection Confirmation'),
     content: Column(
      children: [
       TextField(
        onChanged: (value1) {
         setState(() {
          valueCode = value1;
         });
        },
        controller: verifyFieldController1,
        decoration:
          InputDecoration(hintText: "Enter your secret number"),
       ),
       TextField(
        onChanged: (value2) {
         setState(() {
          valueType = value2;
         });
        },
        controller: verifyFieldController2,
        decoration: InputDecoration(hintText: "Enter the virus type"),
       ),
      ],
     ),
     actions: <Widget>[
      TextButton(
       child: Text('CANCEL'),
       onPressed: () {
        setState(() {
         Navigator.pop(context);
        });
       },
      ),
      TextButton(
       child: Text('OK'),
       onPressed: () {
        setState(() {
         docCode = valueCode;
         typeVirus = valueType;
         Navigator.pop(context);
```

```
              });
              sendVerificationToServer(docCode, typeVirus);
            },
          ),
        ],
      );
    });
}
void sendVerificationToServer(String docCode, String typeVirus) {
  Map body = {"user_id": userid, "code": docCode, "virus_type": typeVirus};
  print("Sending VERIFICATION data to server");
  print(json.encode(body));
  http
      .post(Globals.ip_address + 'positive',
          headers: {"Content-Type": "application/json"},
          body: json.encode(body))
      .then((response) async {
    final decoded = json.decode(response.body);
    print("positive");
    print(decoded); //test
    ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Verified as $typeVirus Positive")));
  }).catchError((e) {
    ScaffoldMessenger.of(context)
        .showSnackBar(SnackBar(content: Text("Can't connect to server")));
  });
}
void othersProbability() {
  print("Trying to send POLICE data");
  Map<String, int> newConnections = Map.fromEntries(recentlyRecievedUUIDS
      .entries
      .where((element) => !recentlySentUUIDs.contains(element.key))
      .map((e) => MapEntry(e.key, e.value)));
  if (newConnections.keys.isEmpty) {
    print("No data to send");
    ScaffoldMessenger.of(context)
        .showSnackBar(SnackBar(content: Text("No Person Nearby")));
    return;
  }
  Map body = {
    "connections": newConnections,
  };
  print("Sending POLICE data to server");
  print(json.encode(body));
  http
      .post(Globals.ip_address + 'police',
```

```
        headers: {"Content-Type": "application/json"},
        body: json.encode(body))
    .then((response) async {
   final decoded = json.decode(response.body) as Map<String, dynamic>;
   print("police");
   print(decoded);
   final Map<String, dynamic> probabilityList = Map.from(decoded);
   setState(() {
    recentlySentUUIDs = recentlyRecievedUUIDS.keys.toSet();
   });
   print(probabilityList);
   print(recentlySentUUIDs);
   Navigator.push(
      context,
      MaterialPageRoute(
         builder: (context) =>
            PolicePage(probabilityList: probabilityList)));
  });//.catchError((e) {
    //ScaffoldMessenger.of(context)
     //  .showSnackBar(SnackBar(content: Text("Can't connect to server")));
  //});
}
void durationApproximation() {
  int count = 1;
  int res;
  if (recentlyRecievedUUIDS.keys.isEmpty) {
    print("No connected devices");
  } else {
    for (MapEntry e in recentlyRecievedUUIDS.entries) {
      if (e.value > -100) {
        closerUUIDS[e.key] = count;
      }
    }
    print("closerUUIDS");
    print(closerUUIDS);
    if (temp1UUIDS.keys.isEmpty) {
      temp1UUIDS = Map.from(closerUUIDS);
    } else if (temp2UUIDS.keys.isEmpty) {
      for (MapEntry e in closerUUIDS.entries) {
        if (temp1UUIDS.containsKey(e.key)) {
          temp2UUIDS.putIfAbsent(e.key, () => (e.value) + count);
        }
      }
      temp1UUIDS = Map.from(closerUUIDS);
    } else {
      for (MapEntry e in closerUUIDS.entries) {
```

```
      if (temp2UUIDS.containsKey(e.key)) {
        temp2UUIDS.update(e.key, (value) => value + count);
      } else if (temp1UUIDS.containsKey(e.key)) {
        temp2UUIDS.putIfAbsent(e.key, () => (e.value) + count);
      }
    }
    temp1UUIDS = Map.from(closerUUIDS);
    for (MapEntry e in temp2UUIDS.entries) {
      if (!(closerUUIDS.containsKey(e.key))) {
        setState(() {
          selectedUUIDS.putIfAbsent(e.key, () => e.value.toString());
        });
        res = temp2UUIDS.remove(e.key);
      }
    }
    print("duration");
    print(res);
  }
}
if (closerUUIDS.keys.isEmpty) {
  if (temp2UUIDS.keys.isEmpty) {
    return;
  } else {
    for (MapEntry e in temp2UUIDS.entries) {
      selectedUUIDS.putIfAbsent(e.key, () => e.value.toString());
    }
    temp2UUIDS = {};
  }
} else {
  for (MapEntry e in temp2UUIDS.entries) {
    if (!(closerUUIDS.containsKey(e.key))) {
      setState(() {
        selectedUUIDS.putIfAbsent(e.key, () => e.value.toString());
      });
      res = temp2UUIDS.remove(e.key);
    }
  }
  print("duration");
  print(res.toString());
}
closerUUIDS = {};
print("temp1UUIDS");
print(temp1UUIDS);
print("temp2UUIDS");
print(temp2UUIDS);
print("selectedUUIDS");
```

```
  print(selectedUUIDS);
  print("Trying to send CONTACT data");
  if (selectedUUIDS.keys.isEmpty) {
   print("No data to send");
   return;
  }
  sendUUIDSToServer();
 }
 void sendUUIDSToServer() {
  //Map location  = getMyCoordinates();
  String temperature = "10";
  String humidity = "Low";
  Map body = {
   "user_id": userid,
   "temperature": temperature,
   "humidity": humidity,
   "connections": selectedUUIDS,
  };
  print("Sending CONTACT data to server");
  print(json.encode(body));
  http
     .post(Globals.ip_address + 'new_contact',
        headers: {"Content-Type": "application/json"},
        body: json.encode(body))
     .then((response) async {
    setState(() {
     selectedUUIDS = {};
    });
  }).catchError((e) {
    ScaffoldMessenger.of(context)
       .showSnackBar(SnackBar(content: Text("Can't connect to server")));
  });
 }
//Location
//Now Create Method named _getCurrentLocation with async Like Below.
 //_getCurrentLocation() async {
 //Geolocation.currentLocation(accuracy: LocationAccuracy.best)
 //  .listen((result) {
 // if (result.isSuccessful) {
 //  setState(() {
 //   lat = result.location.latitude;
 //  long = result.location.longitude;
 //});
 // }
 //});
 // }
```

```
 //Map getMyCoordinates(){
 // _getCurrentLocation();
 //String latitude = lat.toString();
 //String longitude = long.toString();
 //return {"latitude":latitude,"longitude":longitude};
 //}
//Temperature
 // Future<String> getTemp() async {
 // WeatherFactory wf = new WeatherFactory(myKey);
 // Weather w = await wf.currentWeatherByLocation(lat, long);
 // String temp = w.temperature.celsius.toString();
 // return temp;
 //}
 void startBeaconAdvertising(uuid) {
  beaconBroadcast
     .setUUID(uuid)
     .setMajorId(1)
     .setMinorId(100)
     .setLayout('m:2-3=0215,i:4-19,i:20-21,i:22-23,p:24-24')
     .setManufacturerId(0x004c)
     .start()
     .then((value) {
    print("Advertising Beacon with uuid= $uuid");
  });
  beaconBroadcast.getAdvertisingStateChange().listen((advertisingStatus) {
   setState(() {
    isAdvertising = advertisingStatus;
   });
  });
 }
 void listenForBeacons() async {
  await flutter_beacon.flutterBeacon.initializeScanning;
  final regions = <flutter_beacon.Region>[
   flutter_beacon.Region(identifier: 'ibeacon')
  ];
  _streamBeaconRanging = flutter_beacon.flutterBeacon
     .ranging(regions)
     .listen((flutter_beacon.RangingResult result) {
   setState(() {
    recentlyRecievedUUIDS = Map.fromEntries(
       result.beacons.map((e) => MapEntry(e.proximityUUID, e.rssi)));
   });
  });
 }
 @override
 Widget build(BuildContext context) {
```

```
if (userid == null)
  return Container();
else
  return SafeArea(
    child: Scaffold(
      appBar: AppBar(
        title: const Text(
          "R-Naught",
          style: TextStyle(color: Colors.white),
        ),
        backgroundColor: Colors.blue,
      ),
      body: Center(
        child: Column(
          mainAxisSize: MainAxisSize.max,
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            if (isBluetoothOn != true)
              Column(
                children: [
                  Icon(Icons.bluetooth, color: Colors.red, size: 30),
                  Text("Please turn your bluetooth on"),
                  SizedBox(height: 100)
                ],
              ),
            if (isAdvertising == true)
              Column(
                children: [
                  Icon(Icons.bluetooth, color: Colors.green, size: 30),
                  Text("Your uid is visible to nearby phones"),
                  SizedBox(height: 100)
                ],
              ),
            Column(
              children: [
                Text("Your ID: "+userid.toUpperCase()),
                Align(alignment: Alignment.topLeft),
              ],
            ),
            ElevatedButton(
              onPressed: getProbability,
              child: Text("Get My Infection Probability")),
            ElevatedButton(
              onPressed: iamCovidPositive,
              child: Text("Infection Verification (Doctors only)"),
```

```
                  style: ElevatedButton.styleFrom(
                    primary: Colors.red, // background
                    onPrimary: Colors.white, // foreground
                  ),
                ),
                ElevatedButton(
                  onPressed: othersProbability,
                  child: Text("Probaility Verification (Officials only)"),
                  style: ElevatedButton.styleFrom(
                    primary: Colors.blueGrey, // background
                    onPrimary: Colors.white, // foreground
                  ),
                ),
                SizedBox(height: 30),
                Text(recentlyRecievedUUIDS.entries
                    .map((e) => e.key + "\t:\t" + e.value.toString())
                    .join("\n"))
              ]),
          )),
      );
  }
  @override
  void dispose() {
    super.dispose();
    _streamBeaconRanging.cancel().onError((error, stackTrace) => null);
    bluetoothStateChangeStream.cancel();
    _sendToServerTimer.cancel();
  }
}
```

Pages/PolicePage.dart

==========

```
import 'package:flutter/material.dart';
class PolicePage extends StatefulWidget {
  final Map<String, dynamic> probabilityList;
  PolicePage({@required this.probabilityList});
  @override
  _PolicePageState createState() => _PolicePageState();
}
class _PolicePageState extends State<PolicePage> {
  @override
  Widget build(BuildContext context) {
    return SafeArea(
      child: Scaffold(
```

```
        appBar: AppBar(
          title: const Text(
            "Probabilities of nearby\n devices",
            style: TextStyle(color: Colors.white),
          ),
          backgroundColor: Colors.blue,
        ),
        body: Center(
          child: Column(
              children: [
                Text(
                  widget.probabilityList.entries
                      .map((e) =>
                          e.key.toUpperCase() +
                          "\t:-\n" +
                          e.value.entries
                              .map((e) => e.key + ":" + e.value.toString())
                              .join(", "))
                      .join("\n"),
                  style: TextStyle(fontSize: 16),
                )
              ]),
        )),
    );
  }
  @override
  void dispose() {
    super.dispose();
  }
}
```