COMP.SE.210 LARGE SCALE SOFTWARE DESIGN: SPRING-2021
Assignment – Corona Tracker

1. DOCUMENT INFORMATION

The documentation is based on a modified architectural description of 'Koronavilkku' mobile application as a 'Large Scale Software Design' course group project. At first, original Koronavilkku application is observed and analyzed the functionalities of it. After that, as per the project work requirements the objective of the overall project was set. For modification purposes, '4+1' architecture model was chosen to imply as per the instructions. Detailed discussion of this model is presented in this document along with the logical explanation. There are several issues and obstacles which are denoted in the 'Limitations' section. The UML diagram is implemented by following the modification requirements and described the architectural decisions and rationale in the 'Justification' section. Working environment is analyzed and discussed after that according to the deployment diagram. Lastly, all the required UML diagrams are illustrated.

2. DESCRIPTION OF THE ORIGINAL KORONAVILKKU APPLICATION

Koronavilkku application is a mobile tracker app that tracks COVID-19 and it can help in stopping the chains of COVID-19 infection. This app allows people to participate and influence the prevention of infections. Koronavilkku makes it possible to trace users who may have been exposed to the coronavirus. The app alerts the user to their potential exposure to the coronavirus and provides them with instructions e.g., following the COVID-19 guidelines, contacting health care services and, if necessary, having a test.

The operating principle of this app is simple and effective. After downloading the app, user's phone will record a character sequence regarding any encounters in which users may have been a chance to get exposed by corona virus. Security level is maintained as no personal or geospatial data is saved; only anonymous key codes are stored. If any user falls ill or test positive to covid-19, they will be provided with a verification code from healthcare services. It helps to warn other people anonymously that the affected user has met during the incubation period using his/her phone. Users will also receive an exposure warning if any persons they have encountered falls ill. The verification code ensures that, the app warnings are based on the confirmed infections. The person who sends a warning and people who receive them will not provide any information about each other's identity.

3. Objectives

The objective of this documentation is to implement the necessary and required modification in the application architecture. Major goal of this documentation is given below:

- Strong authentication is supported by the system.
- Swift process in health status checking for any location
- Maximum level of load balancing
- Secure and quick authentication
- Low power consumption

Addressing these objectives, the UML diagrams are implemented.

4. Requirements/Limitations

There are some privacy concerns in the architecture document of Koronavilkku++. The current role of Koronavilkku is only marginal in tracking infections. The role of the tracker is to be strengthened. So, to overcome tracking issue we must put privacy aside. In the modified version of Koronavilkku application, reporting the infection is no more voluntary. There are also few requirements to make this app work properly. All the Finnish residents must install the application, carry the phone, and have Bluetooth on to track encounters and Location on to track geospatial data.

5. Description of chosen architecture model

Software architecture deals with the design and implementation of the high-level structure of the software. It deals with abstraction, with decomposition and composition. To describe a software architecture of Koronavilkku application, we use a model composed of multiple views or perspectives i.e., 4+1 view model. This model is used for "describing the architecture of software-intensive systems, based on the use of multiple, concurrent views".[1] Hence, the model contains:

- **Logical view:** The logical view documents the functionality. UML diagrams are used to represent the logical view, and include class diagrams, and state diagrams. In this documentation, we implemented class diagram which defines plethora of classes along with their functionalities.
- **Process view:** The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the run time behavior of the system. UML diagrams to represent process view include the sequence diagram, communication diagram, activity diagram.[2]
  The images 1-4 describes how to the software processes work. As the infection tracking and detection is probably one of the most important parts in this software, it has been described more than the other processes in the image 1. The following images describes the communication between components in the images 2-4 on a general level.
- **Development view:** The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation

view. It uses the UML Component diagram to describe system components. UML Diagrams used to represent the development view include the Package diagram.

- **Physical view:** The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer as well as the physical connections between these components. This view is also known as the deployment view. UML diagrams used to represent the physical view include the deployment diagram.
- **Scenarios:** It represents the user's interactions with the system. The scenarios describe sequences of interactions between objects and between processes. They also serve as a starting point for tests of an architecture prototype. This view is also known as the use case view.

6. JUSTIFICATIONS

The software has been split into a central database/server and microservices to make sure that the program has the capability to handle possibly millions of connections at the same time. As the program is split into multiple different parts, if one server crashes, the other ones should be able to handle any incoming requests. In the case of the central database crashing, the other servers must be able to work on the local databases, which is why the tracking information is also copied to the servers. Although in the case of the central database crashing creating accounts or logging in won't work, it will be able to allow logged in users to log in and retrieve the latest copied data from the server database whilst the central database is being fixed.

The user identity verification is on a third-party server as it cannot be integrated into our system, Koronavilkku++ will have to follow the API (Application Programming Interface) provided by the suomi.fi verification. Because of this the inner structures of the verifications are omitted from the document.

As the webservers must be reliable, AWS servers will be used as the base for the microservices, as the needed functionality it built into them and under a heavy load AWS will dynamically add more system resources to the server. In the class diagram, hence AWS is added as the cloud storage to ensure reliability and scalability (load balancing). Another crucial aspect is AWS storage fulfils the criteria of strong, quick, and safe authentication services.

As the main packets used to check if a user has been exposed mainly contain some thousands of lines of USER_IDs (for example, 1 USER_ID = UUID = 36 bytes, 1000 infected USER_IDs = 36kb) and some login tokens the packets will be rather small. As the packets are small, it allows the packet processing to be fast and download times small even in regions where the internet connection is not good, which means that the total time for the health status check should be rather fast other than in very exceptional circumstances.

The proximity tracking must be implemented with the Bluetooth receiver. In order to determine if a user has been in close proximity with another user, the Bluetooth receiver should broadcast the USER_ID to all nearby devices every 10 seconds, and other devices should listen to any incoming signals at all times. The timing is selected to be 10 seconds as it provides enough accuracy to detect a 2min contact period, it won't drain as much battery as more frequent broadcasts. Although the distance the proximity should be tracked is 10m, it is very difficult or even impossible to get accurate results. If device location would be sent with the Bluetooth broadcast, it wouldn't provide much benefit as the accuracy for any background location *scans is very inaccurate, and could even message the other phone that it is further away than the Bluetooth broadcast range. In addition, as devices have multiple different strengths for Bluetooth broadcasts or they may be obstructed in some way, the signal strength cannot be used to determine the distance. Although these negative sides make the implementation sound difficult, the Bluetooth broadcast range is short, which means that if a broadcast is being detected, it will be close to the sender of the said signal. It is also generally believed that the average working distance for a Bluetooth signal is around 10 meters.*

As the application that runs in the background uses only the minimal possible activities, it allows the battery to last as long as possible for the users phone, where the hourly battery consumption limit is set to be 5%.

Decision on deployment diagram implementation for physical view is based on standard way of package installation process for mobile application. Mobile device should have at least 3 necessary components or internal devices: Bluetooth receiver and transmitter to send data to near devices, touch screen display to interact with device user. Third is either baseband processor or Wi-Fi modem so application load from app-server is possible. Application runs either on iOS or Android. As exception in case for Android, if mobile device is Huawei, "AppGallery" will act as app server instead of "Play Store".

As per the requirements, geo-location should be added in the storage. In class diagram implementation, this modification is applied by applying a separate class named 'Geo-location' which contains all the required attributes including address, latitude, longitude and so on. Proximity checking and tracking is ensured by the 'Mobile App' class. Exposure notification is implemented to ensure that the latest information is available along with the user-id and case detection data. 'THL Service' class is maintained as a persistent store of infections.

7. WORKING/DEVELOPMENT ENVIRONMENT

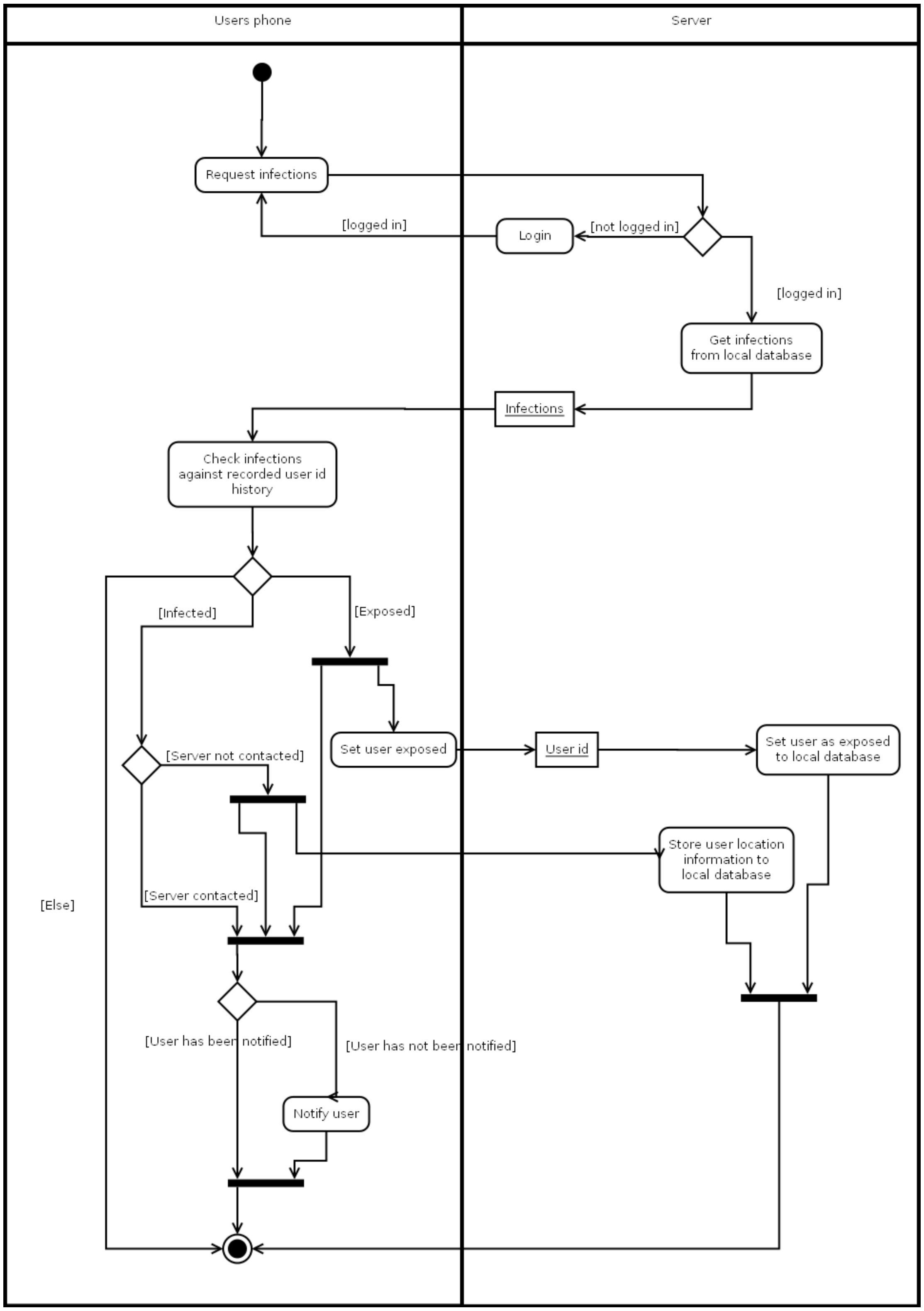
The main programming language to be used is C++14 using the Qt framework, which provides the capabilities to produce Android, iOS, and possibly Windows Phone applications for future releases. The code is stored into GitHub or other git-based version control. Unit will be implemented in the Qt style, and every push that goes to git must be tested with SonarCloud. As the application must work on multiple different devices, unit testing has to be done on different versions of Android and iOS.

To make sure the server is capable of handling large amounts of connections C++ and CMake shall be used as the programming language for the server, and PostgreSQL for databases. The servers will also implement unit testing and CI/CD testing on SonarCloud, but in addition also smoke testing on AWS servers that are disconnected from the live software. Any server versions must be tested to be working with all released functional mobile applications that are currently in use. As the backend is implemented with pure C++, the developers may choose any IDE capable of editing the code.

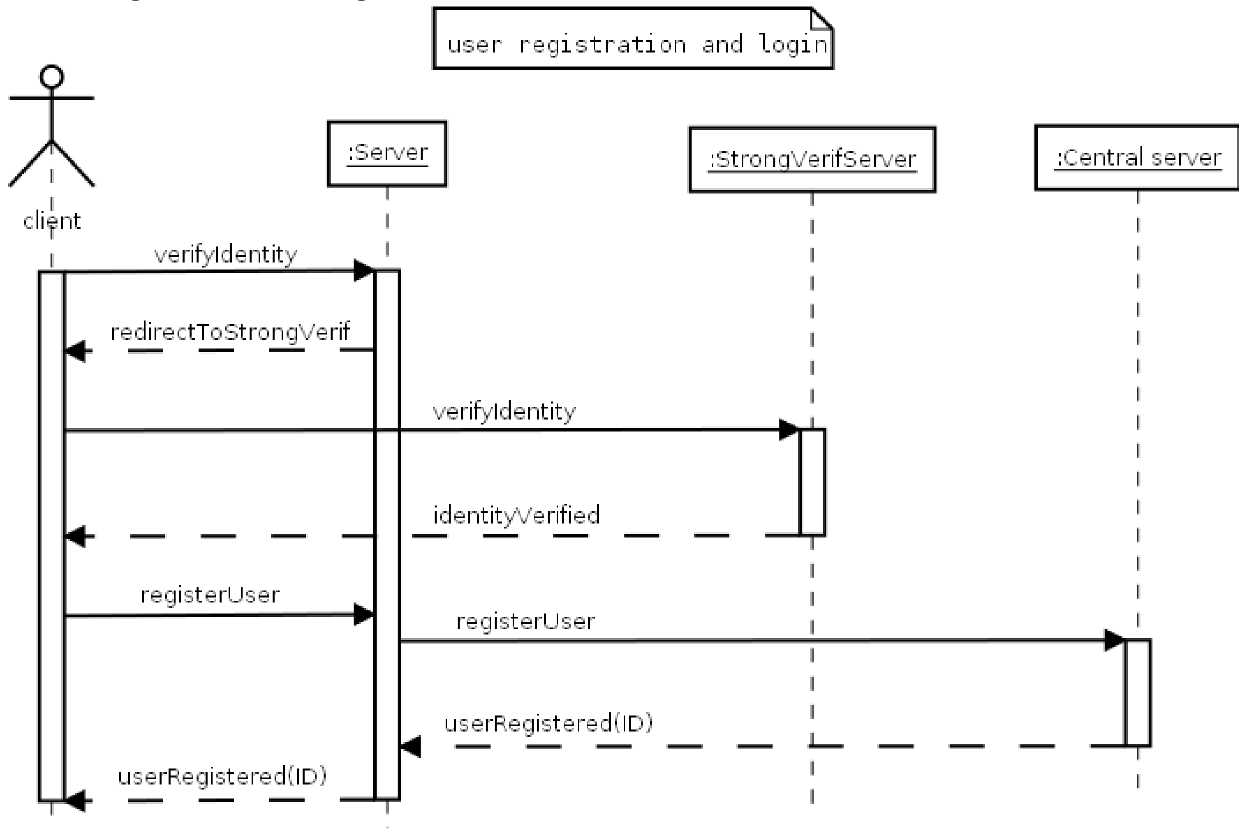
Once the software has been verified to work with all existing live versions of the application the software may be deployed to the production servers.

8. 4+1 VIEWS

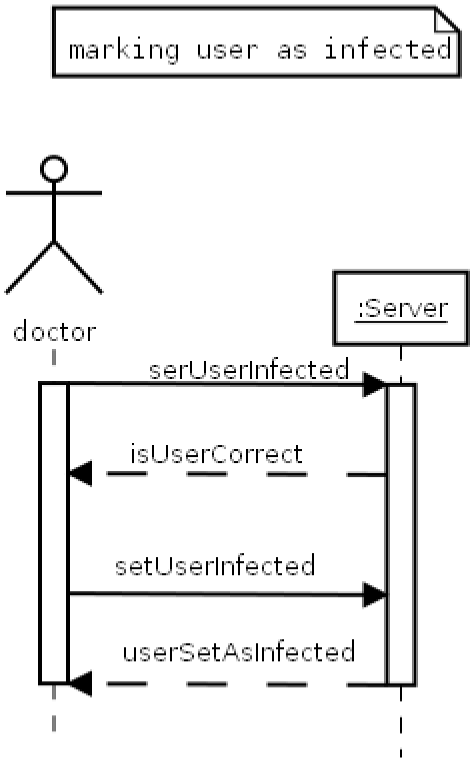
1. Activity diagram



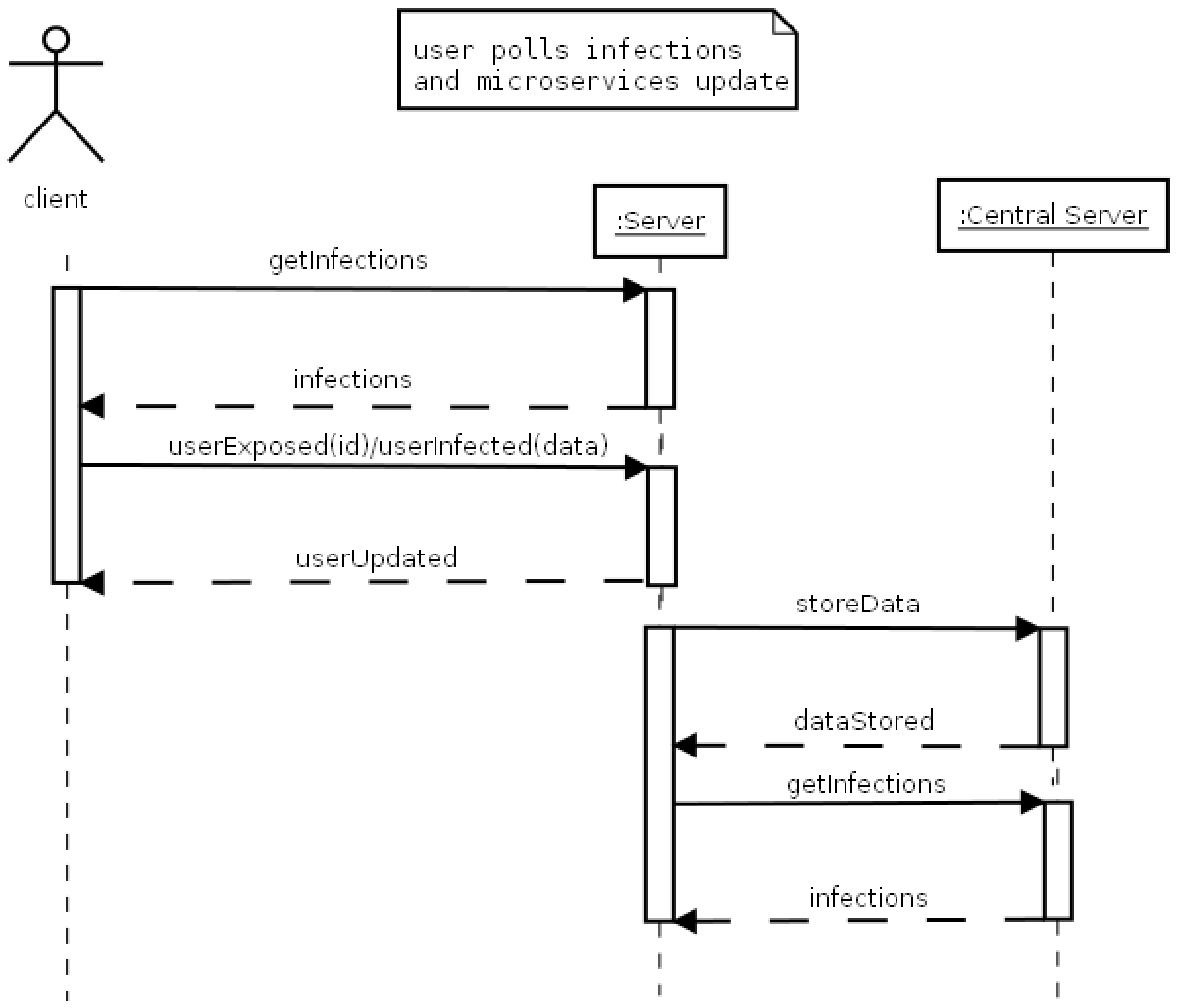
2. User registration and login



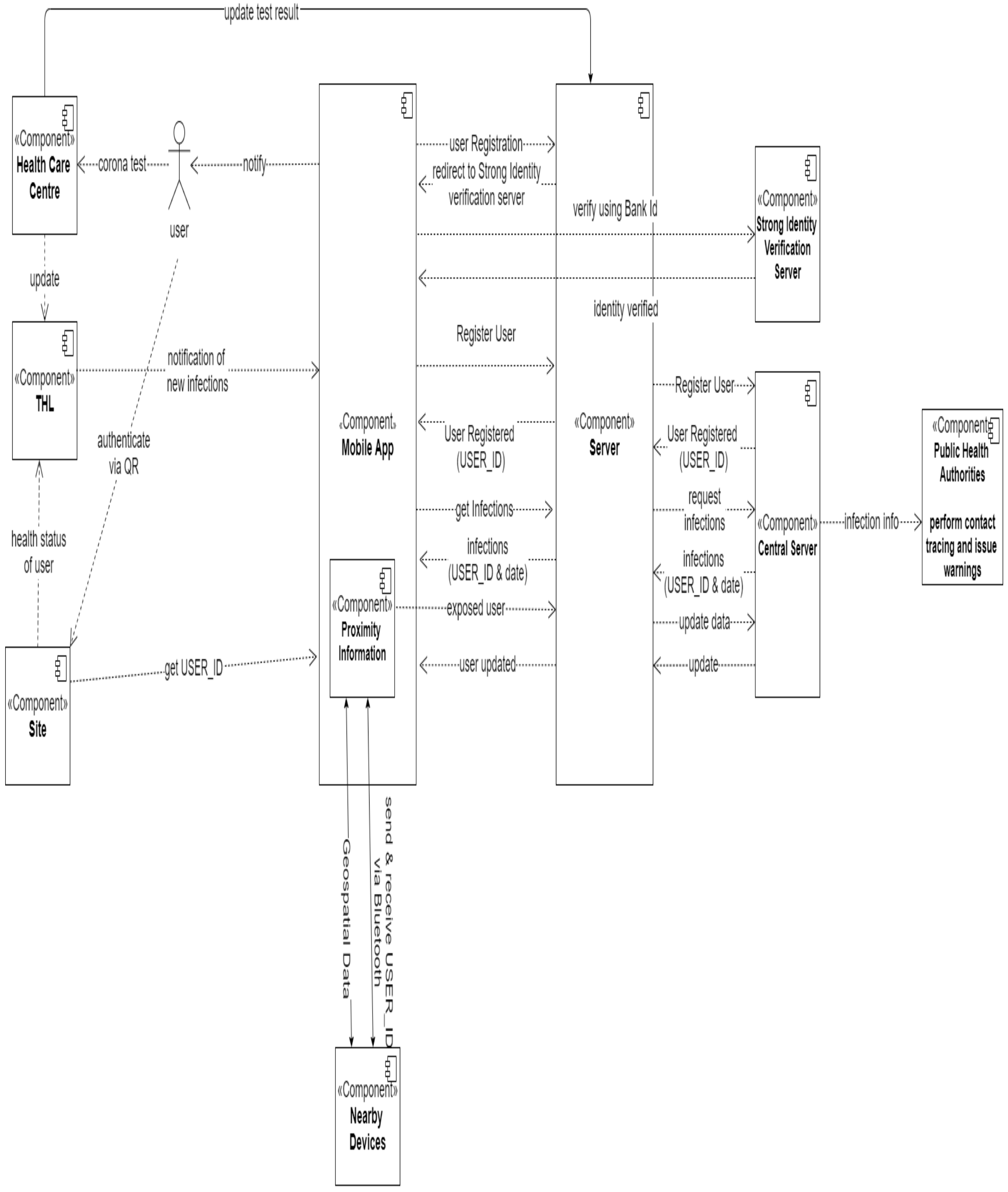
3. Marking user as infected



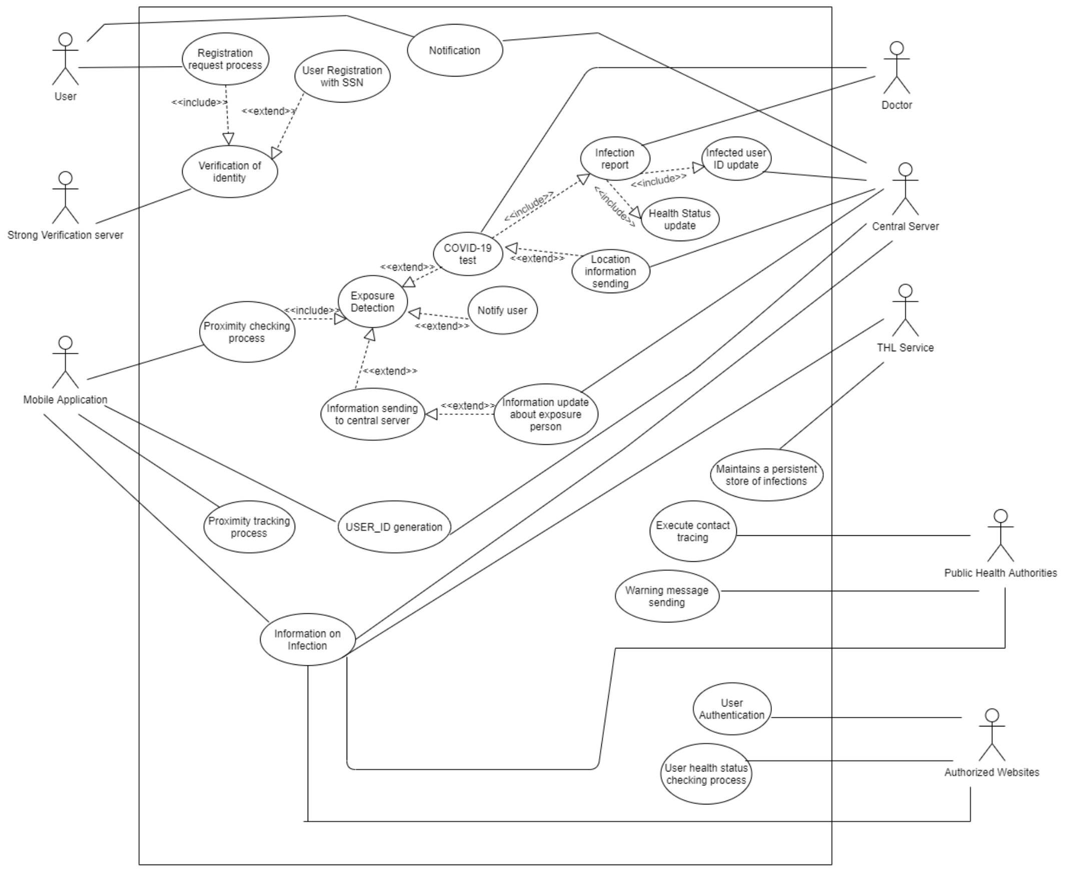
4. Infection poll and microservice update



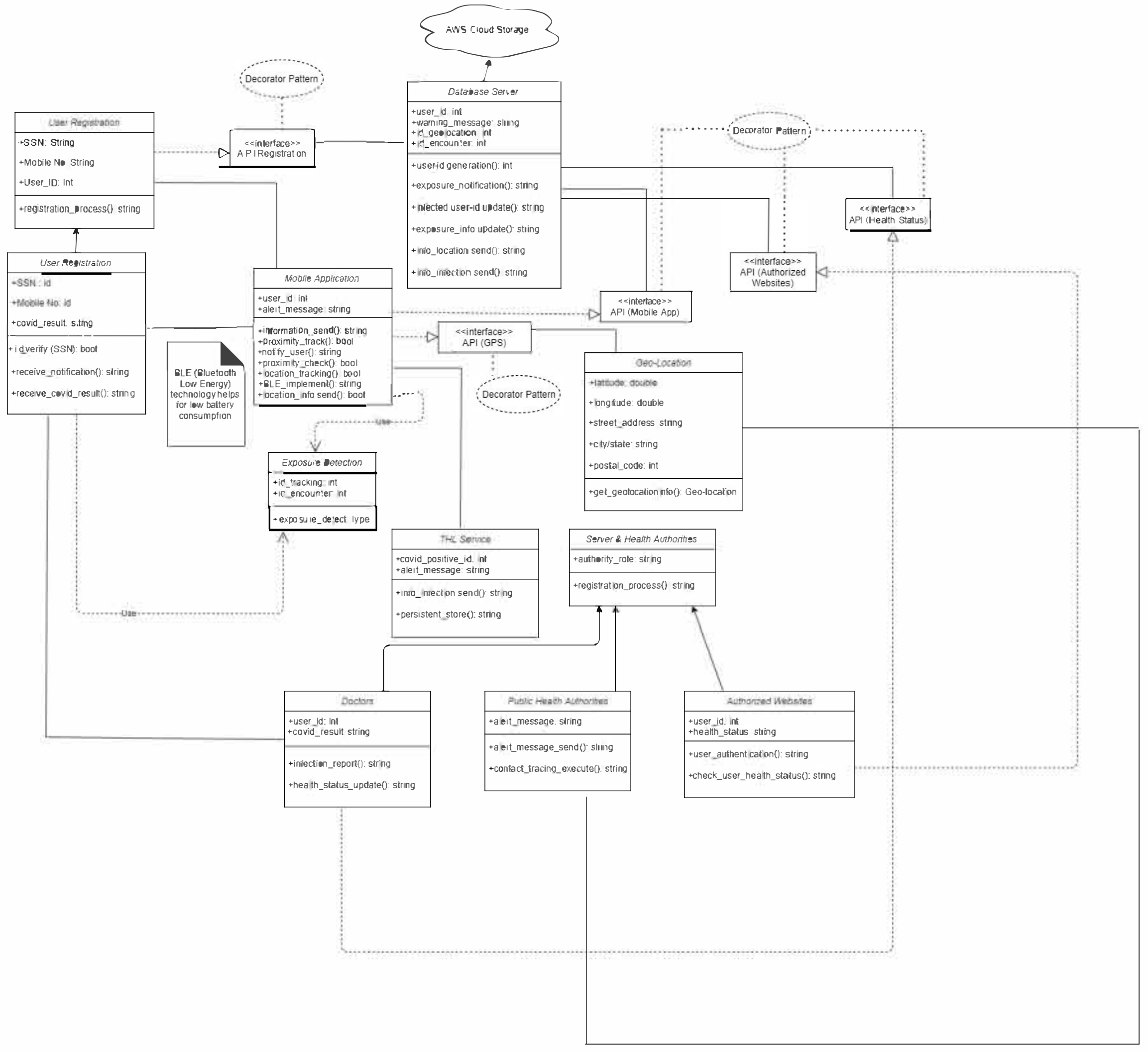
5. Component Diagram


update test result
«Component» Health Care Centre
corona test
user
notify
update
«Component» THL
notification of new infections
authenticate via QR
health status of user
«Component» Site
get USER_ID
«Component» Mobile App
«Component» Proximity Information
user Registration
redirect to Strong Identity verification server
verify using Bank Id
identity verified
Register User
User Registered (USER_ID)
get Infections
infections (USER_ID & date)
exposed user
user updated
«Component» Server
«Component» Strong Identity Verification Server
Register User
User Registered (USER_ID)
request infections
infections (USER_ID & date)
update data
update
«Component» Central Server
infection info
«Component» Public Health Authorities
perform contact tracing and issue warnings
Geospatial Data
send & receive USER_ID via Bluetooth
«Component» Nearby Devices

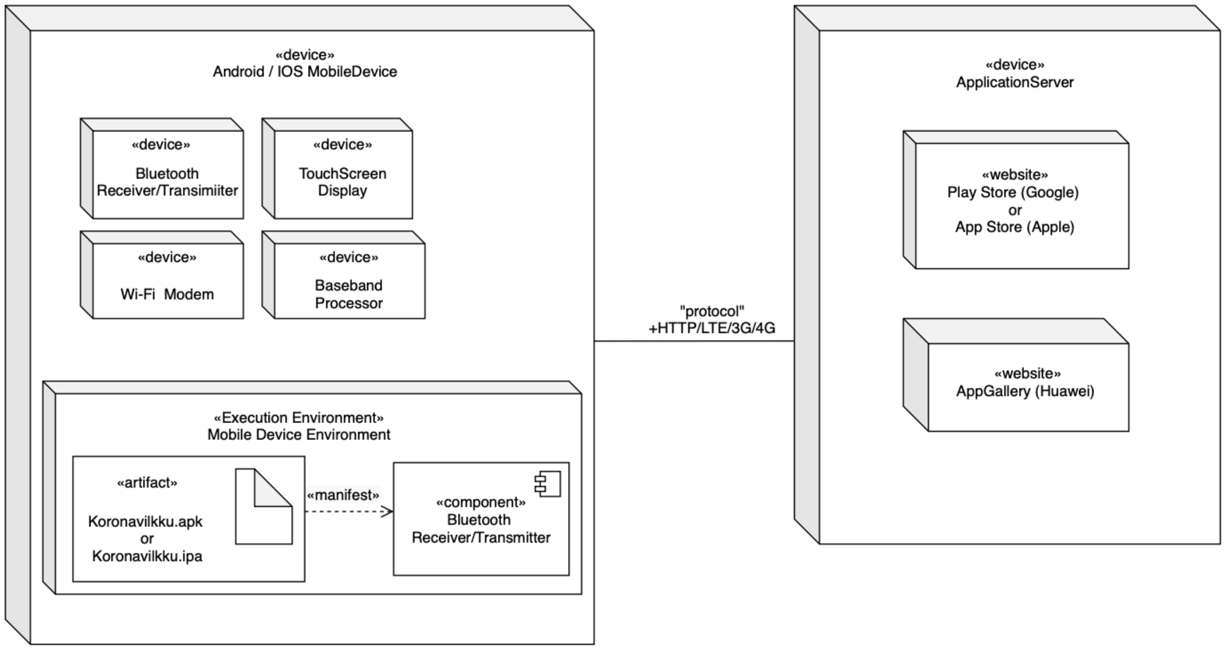
6. Use Case Diagram



7. Class Diagram


AWS Cloud Storage
Decorator Pattern
Database Server
+user_id: int
+warning_message: string
+id_geolocation: int
+id_encounter: int
+userid generation(): int
+exposure_notification(): string
+infected user-id update(): string
+exposure_info update(): string
+info_location send(): string
+info_infection send(): string
User Registration
+SSN: String
+Mobile No: String
+User_ID: Int
+registration_process(): string
<<interface>>
API Registration
Decorator Pattern
<<interface>>
API (Health Status)
<<interface>>
API (Authorized Websites)
<<interface>>
API (Mobile App)
User Registration
+SSN : id
+Mobile No: id
+covid_result: string
+ id verify (SSN): bool
+receive_notification(): string
+receive_covid_result(): string
BLE (Bluetooth Low Energy) technology helps for low battery consumption
Mobile Application
+user_id: int
+alert_message: string
+information_send(): string
+proximity_track(): bool
+notify_user(): string
+proximity_check(): bool
+location_tracking(): bool
+BLE_implement(): string
+location_info send(): bool
<<interface>>
API (GPS)
Decorator Pattern
Use
Exposure Detection
+id_tracking: int
+id_encounter: int
+exposure_detect: type
Geo-Location
+latitude: double
+longitude: double
+street_address: string
+city/state: string
+postal_code: int
+get_geolocation info(): Geo-location
THL Service
+covid_positive_id: int
+alert_message: string
+info_infection send(): string
+persistent_store(): string
Server & Health Authorities
+authority_role: string
+registration_process(): string
Use
Doctors
+user_id: Int
+covid_result string
+infection_report(): string
+health_status_update(): string
Public Health Authorities
+alert_message: string
+alert_message_send(): string
+contact_tracing_execute(): string
Authorized Websites
+user_id: int
+health_status: string
+user_authentication(): string
+check_user_health_status(): string

8. Deployment Diagram



10. References

[1] Kruchten, Philippe (1995, November). Architectural Blueprints — The "4+1" View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50.

[2] Hui, LM; Leung, CW; Fan, CK; Wong, TN (2004). "Modelling agent-based systems with UML". Proceedings of the Fifth Asia Pacific Industrial Engineering and Management Systems Conference.