Politecnico di Milano

AA 2018-2019

POLITECNICO
MILANO 1863

Computer Science and Engineering

Software Engineering 2

DD

Design Document

Version 1.0 - 10.12.2018

Sankari Gopalakrishnan

David Brellmann

Louis Lesieur

Table of Contents

1.INTRODUCTION

1.1 Purpose

The purpose of this document is to give an insight into design details for the application system. While the RASD presented the main goals and requirements of the system, this document aims to present the implementation design of the system including components, services and their integration. It also includes the testing strategy to be adopted .

More precisely, the document presents:

High level architecture
Main components and their services
Design Decisions and patterns
Runtime behavior
Implementation plan
Integration plan
Testing plan

1.2 Scope

As already mentioned, the Data4Help service is expected to give anonymous health data to third parties requiring it. The anonymity should be always considered for the privacy of the users. That is to say that the application should not provide data which could be misused by third parties, for instance if the size of specific category of persons whose data is required is too few.

The second service is meant to call help for elderly people if they need it. Thus, the application should monitor the data continuously, and not just retrieve the data of the device once a day. In order to know when one's situation becomes dangerous, AutomatedSOS should allow elderly users to set thresholds in their parameters to determine their limits. Moreover, the system should manage to contact an ambulance facility quickly enough to be helpful.

For the Track4Run service in particular, and also for the others, TrackMe has to ensure the interaction with the GPS integrated in the device, to monitor the location of the users

1.3 Definitions, Acronyms

1.3.1 Definitions

- Device/Wearable: any device owned by the user which is able to collect the data and sending it to the application

- Third-Party: association or business interested in collecting data
- Health data: cardiac rhythm, body temperature, number of steps in a period of time
- User : Person who has registered to the data4help service
- Spectator : Person who wants to view the run situation and participant's position
- Client – User/Third-Party/Spectator who uses the application

1.3.2 Acronyms

- API – Application Programming Interface
- GPS – Global Positioning System
- RO – Run organizers
- RP – Run participants

1.4. Document Structure

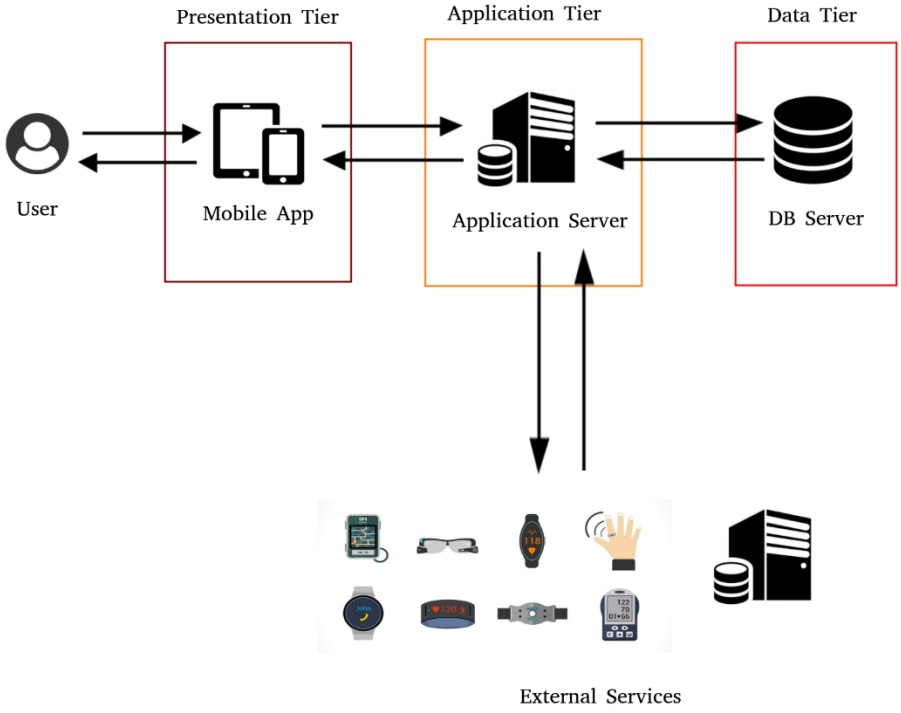
Chapter 1 introduces the design document. It contains the purpose and the scope of the document, as well as definitions in order to provide a better understanding of the document. Chapter 2 deals with the architectural design of the application. It gives an overview of the architecture and it also contains relevant diagrams to show component view, class view, deployment view, runtime view and interaction between the component interfaces. Some of the architectural designs and designs patterns are also presented here, with corresponding explanation of their purpose. Chapter 3 shows a flow between the UI mockups. Chapter 4 explains how the requirements that have been defined in the RASD map to the design elements that are defined in this document. Chapter 5 identifies the order in which it is planned to implement the subcomponents of the system and the order in which it is planned to integrate such subcomponents and test the integration. Chapter 6 shows the effort spent by each group member while working on this project. Chapter 7 includes the reference documents.

2.ARCHITECTURAL DESIGN

2.1 Overview

Application architecture design is a process which has to be executed in a defined flow. The flow basically includes three different layers (Three-tier architecture):

-



Presentation Tier is the topmost level of the application. It is the layer in which clients have a dynamic GUI that communicates with the Application Server, in other words, it is human-computer interaction. It includes UI components and UI process.

Application Tier is pulled out from the presentation tier and, as its own layer, it controls the application's functionality by performing detailed processing. Moreover, this layer interacts with the external devices such as wearables and Google Services for collecting health data or with the ambulance facility to call an ambulance. The various management and control rules of the system are implemented in this layer. To perform these tasks, it uses the data in the lower layer, the Persistence Tier, and send the answers to the Presentation Tier.

Persistence Tier comprises of the database/data storage system and data access layer. This includes data access components, data helpers/utilities, and service agents.

2.2 High Level Architecture

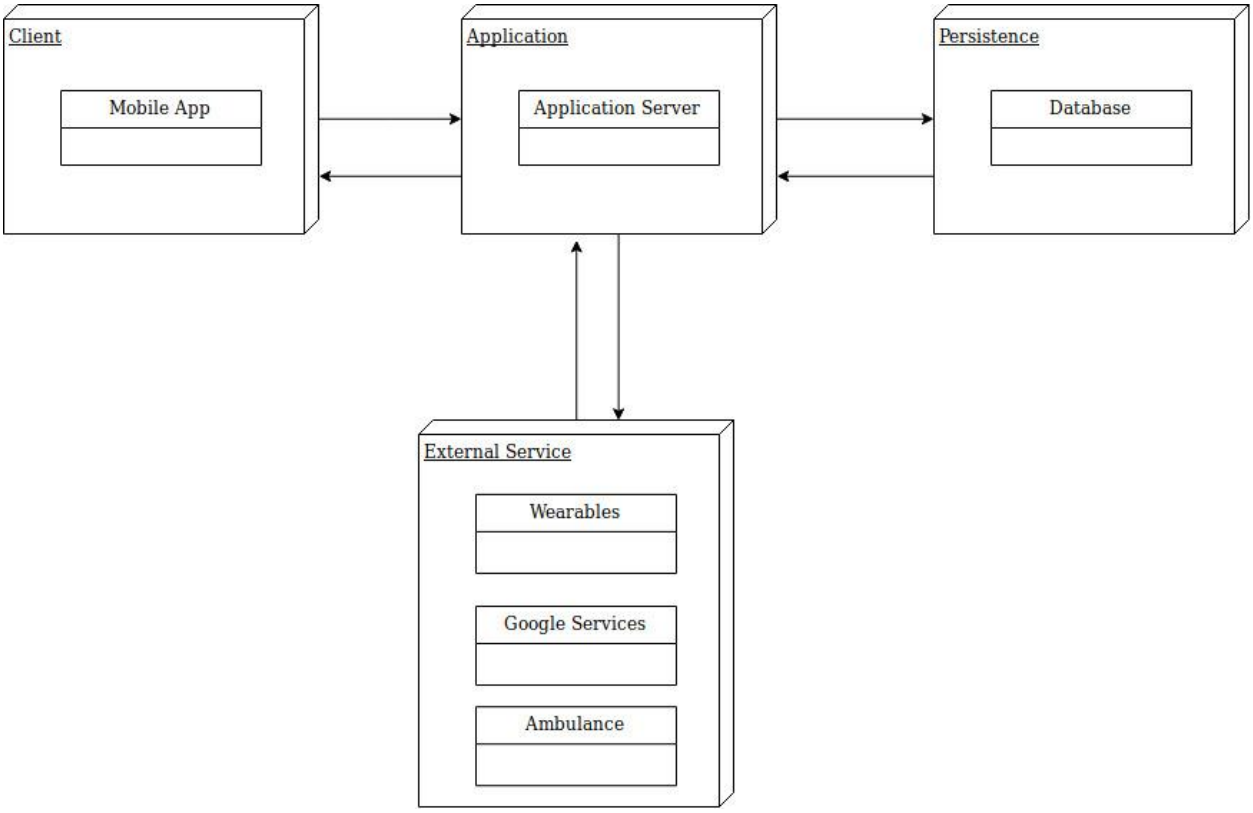


Figure 1: High level component and their interactions.

The application server is the main component that links all the other components. It holds the business logic of the system. It interacts with the clients who can be the Third-Party, the User and the Spectator using the mobile application on their devices. It also has an access with the database server which holds all the health and personal data collected and stored by the application. Finally, the application server can interact with external services such as wearable to collect the targeted health data or to get a map from the Google Service or otherwise to call an ambulance.

2.3 Component View

In the following diagram, the mentioned components are more closely examined, with the main focus on the application server. While describing the parts, the notation will be the interface they are providing in order to be more general, because there may be one or more different implementations of the same service within the server.

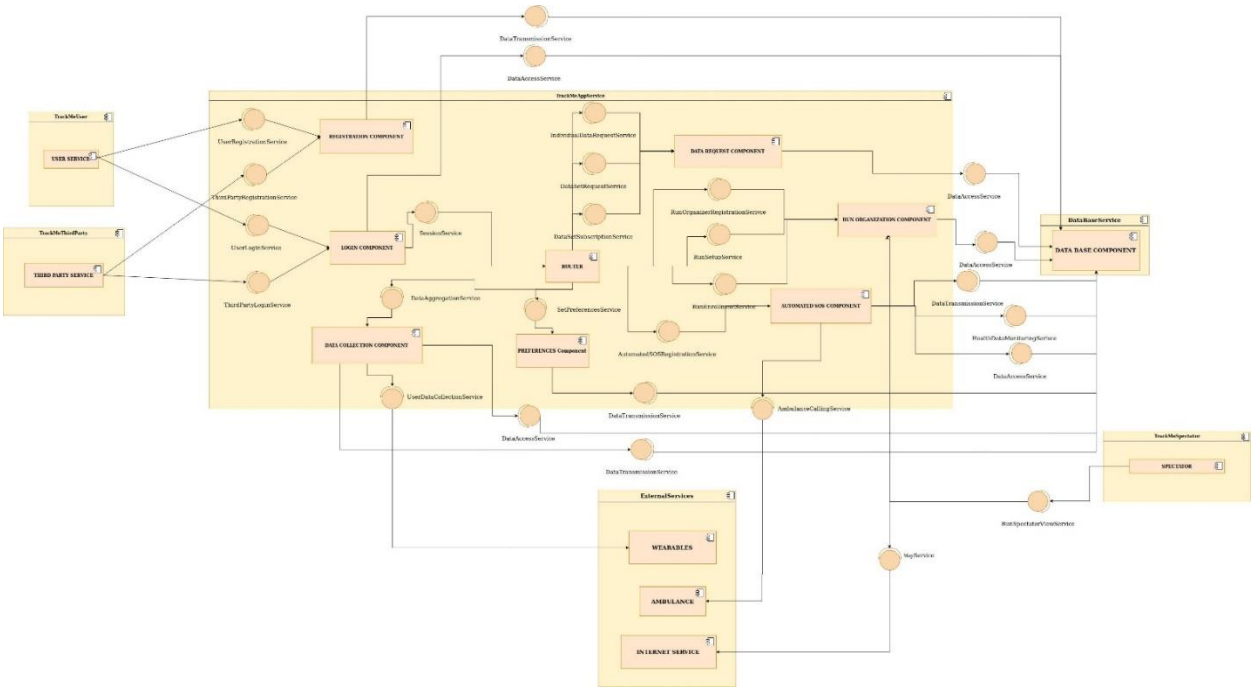


Figure 2: Component Diagram

The high-level components depicted before are defined by the following services :

- **UserRegistrationService / ThirdPartyRegistrationService** : responsible for the registration of the User / Third Party
- **UserLoginService / ThirdPartyLoginService** : responsible for the authentication of the User / Third Party
- **SessionService** : provides the access of all the functionalities proposed by the application
- **UserDataCollectionService** : responsible for collecting the data (health data, position) measured by the wearables
- **DataAggregationService** : provides the collection of data of the user according to user preferences
- **DataAccessService** : provides the access to the data stocked in the database
- **DataTransmissionService** : responsible for sending data to the database
- **SetPreferencesService** : defines the preferences for a given wearable
- **DataSetSubscriptionService** : enables the subscription of the Third Party to collect the data of the users
- **DataSetRequestService** : responsible for sending request to a targeted population of users by a Third-Party
- **IndividualDataRequestService** : responsible for sending individual request to a specific user by a Third-Party
- **RunOrganizerRegistrationService** : responsible for the registration of the Third-Party to organize a run
- **RunSetupService** : provides the definition of the parameters of a run
- **RunEnrollmentService** : responsible for the registration of the users to participate to a run
- **AutomatedSOSRegistrationService** : responsible for the registration of the user to the service AutomatedSOS

- **AmbulanceCallingService** : responsible for calling an ambulance or other emergency services
- **HealthDataMonitoringService** : responsible for monitoring the health data of a user who subscribed to AutomatedSOS

The components shown here communicate with each other and work together to complete certain user requests. The Router has the job of determining whether a request that is received is valid or not and, if it is, to dispatch it to the relevant service component.

The following diagram is provided to further describe the structures used by the services and components (see section 2.6).

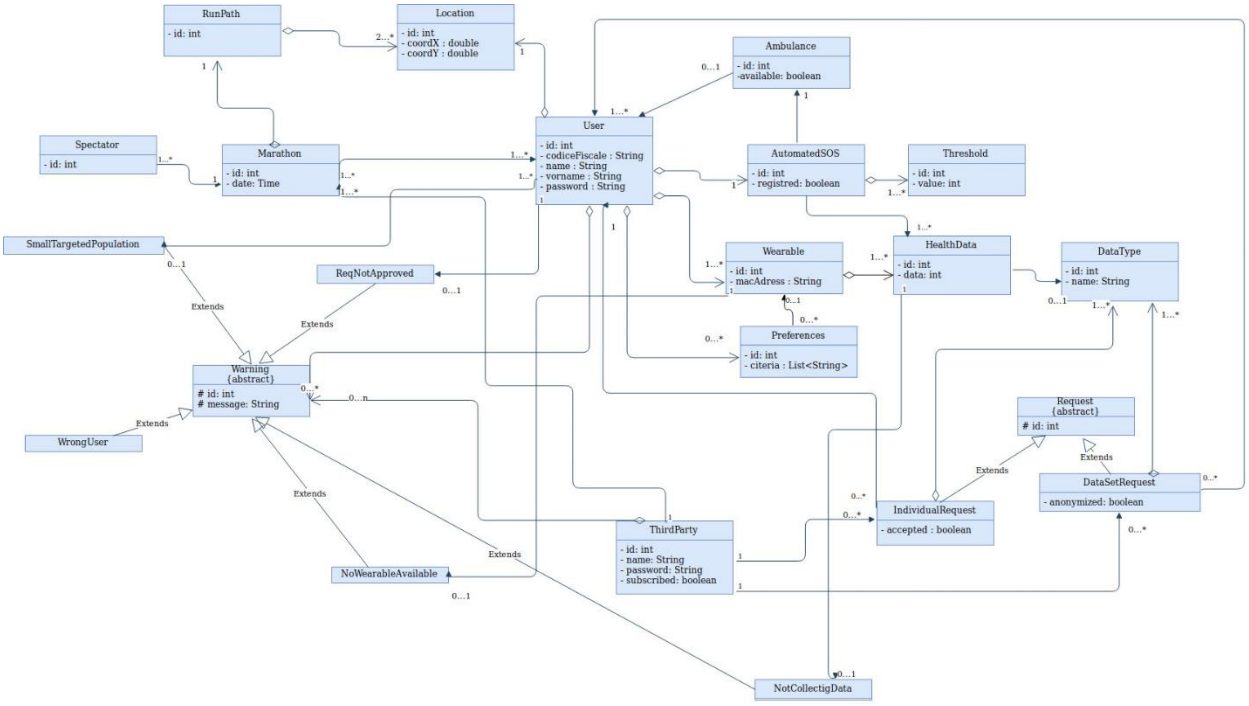


Figure 3 : Class view

2.4 Deployment view

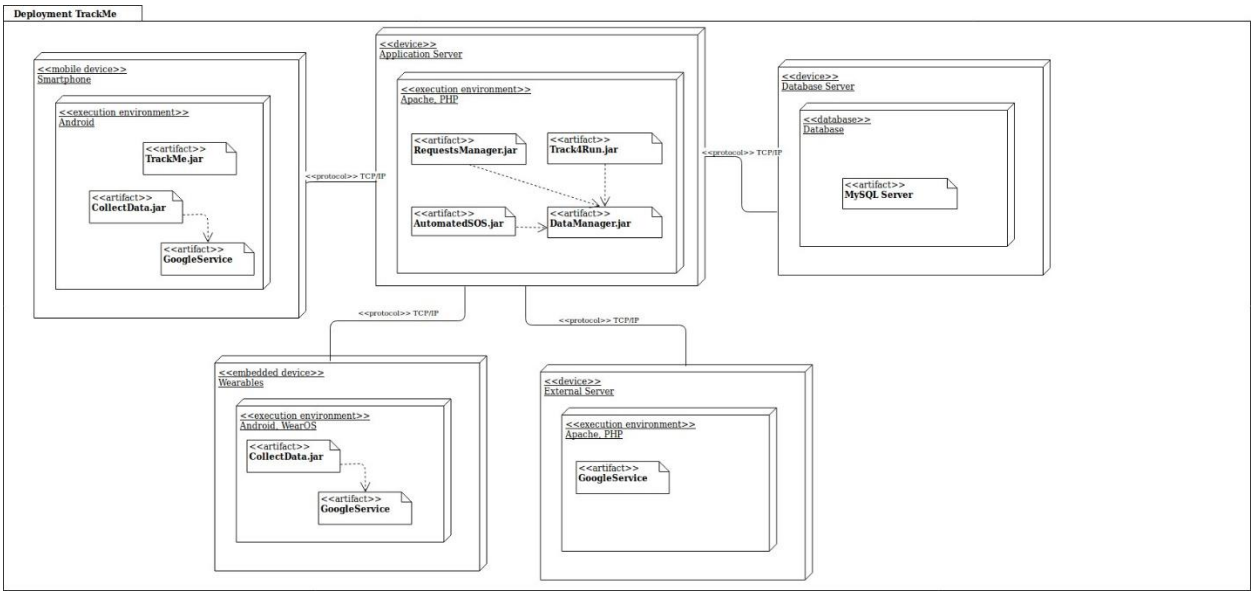


Figure 4 :Deployment Diagram

The deployment diagram shows the architecture of the system as deployment (distribution) of software artifacts to deployment targets. TrackMe requires deployment of software on the following nodes :

- **TrackMe.jar** is the application that will be used by the User/Third Party. The client will be able to get/send information from/to the main Application server.

- **RequestManager.jar** is the software that manages the requests generated by the Third Party and targeting one or more Users (Individual Request or targeted Population Request). It allows the Third Party to know if his request is valid and communicates with the smartphone node and database node.

- **Track4Run.jar** is the software that manages a run organized by a Third Party for users. It communicates with the database node and the smartphone node.

- **AutomatedSOS.jar** is the software which manages the monitoring of health data of the elderly users who subscribed. It communicates like the previous ones with the database node for collecting the data to monitor. Besides, it interacts with the external services and especially with the emergency services in the case where the person is in danger.

- **DataManager.jar** manages the sending of data to the database server. It is the link between all the nodes in the architecture.

- **MySQL Server** will store all the persistent data for the users such as usernames, codice fiscale, passwords, personal data as well as their preferences and the data collected by the wearables.

- **CollectData.jar** manages the collect of health data or positions measured by the wearables/smartphone

- **GoogleServices** represents the Google services that can be used to simplify the application (to get the map of the run for example) and to collect of data measured by the wearables with Google Fit or other services.

2.5 Runtime view

Here are some sequence diagrams to show the way component interact for some tasks.

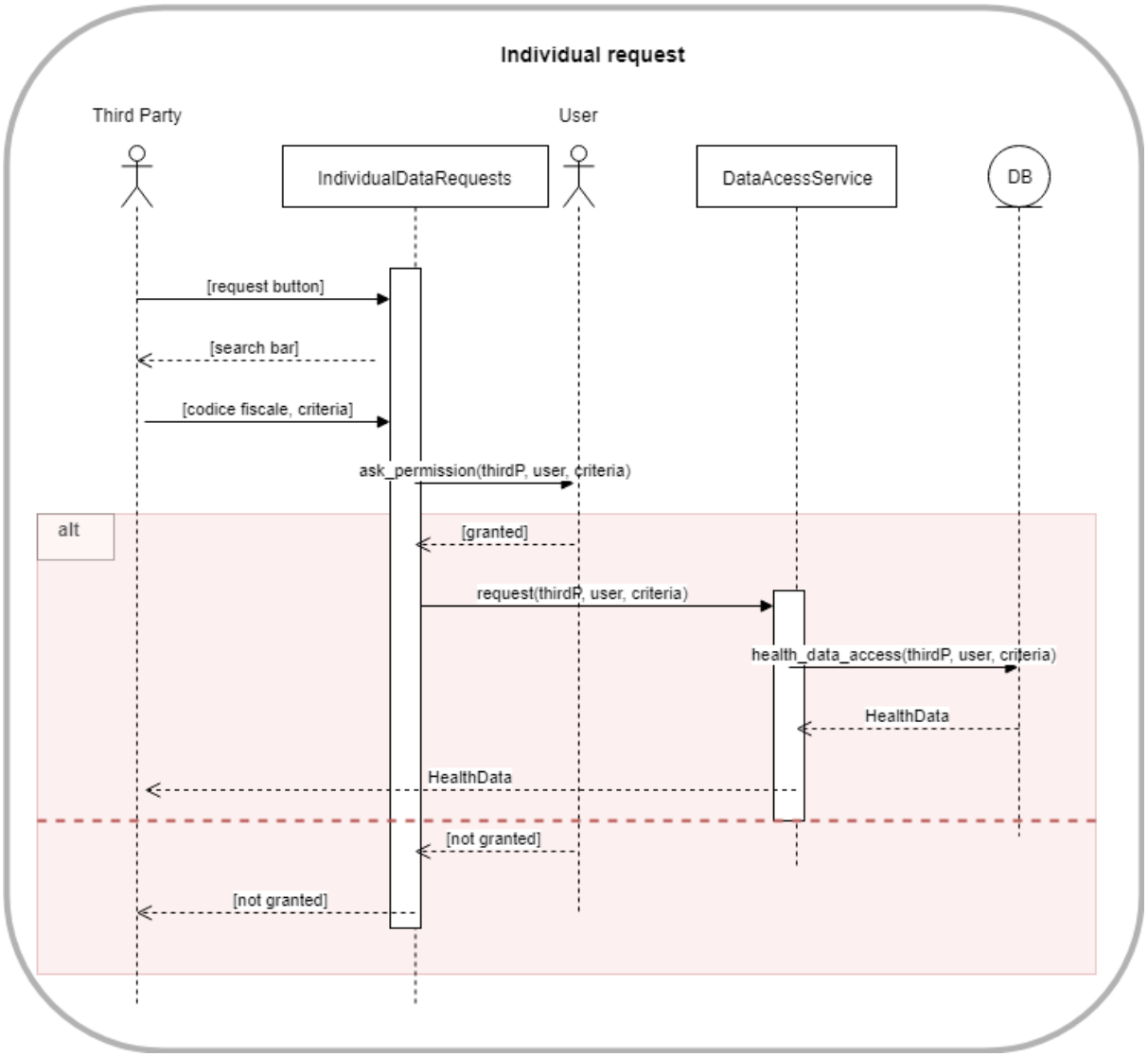


Figure 5 : Sequence Diagram to show runtime view

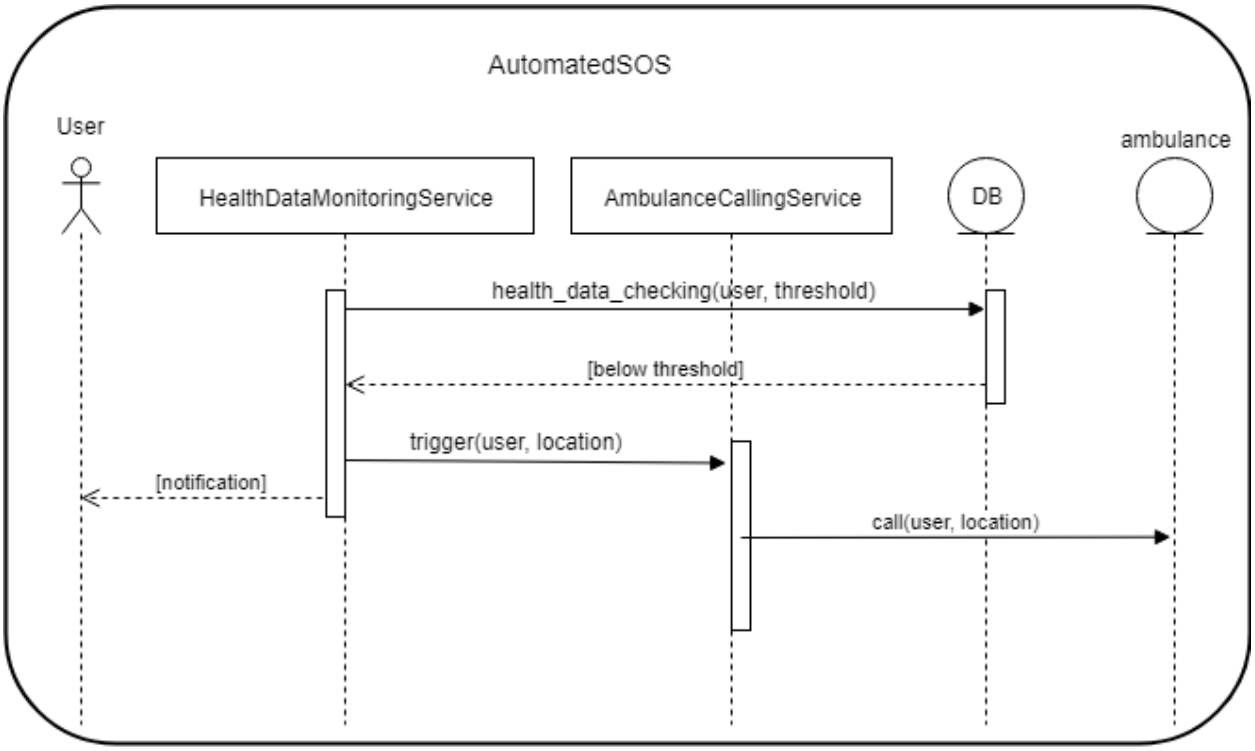


Figure 6 : Sequence Diagram to show an AutomatedSOS service case

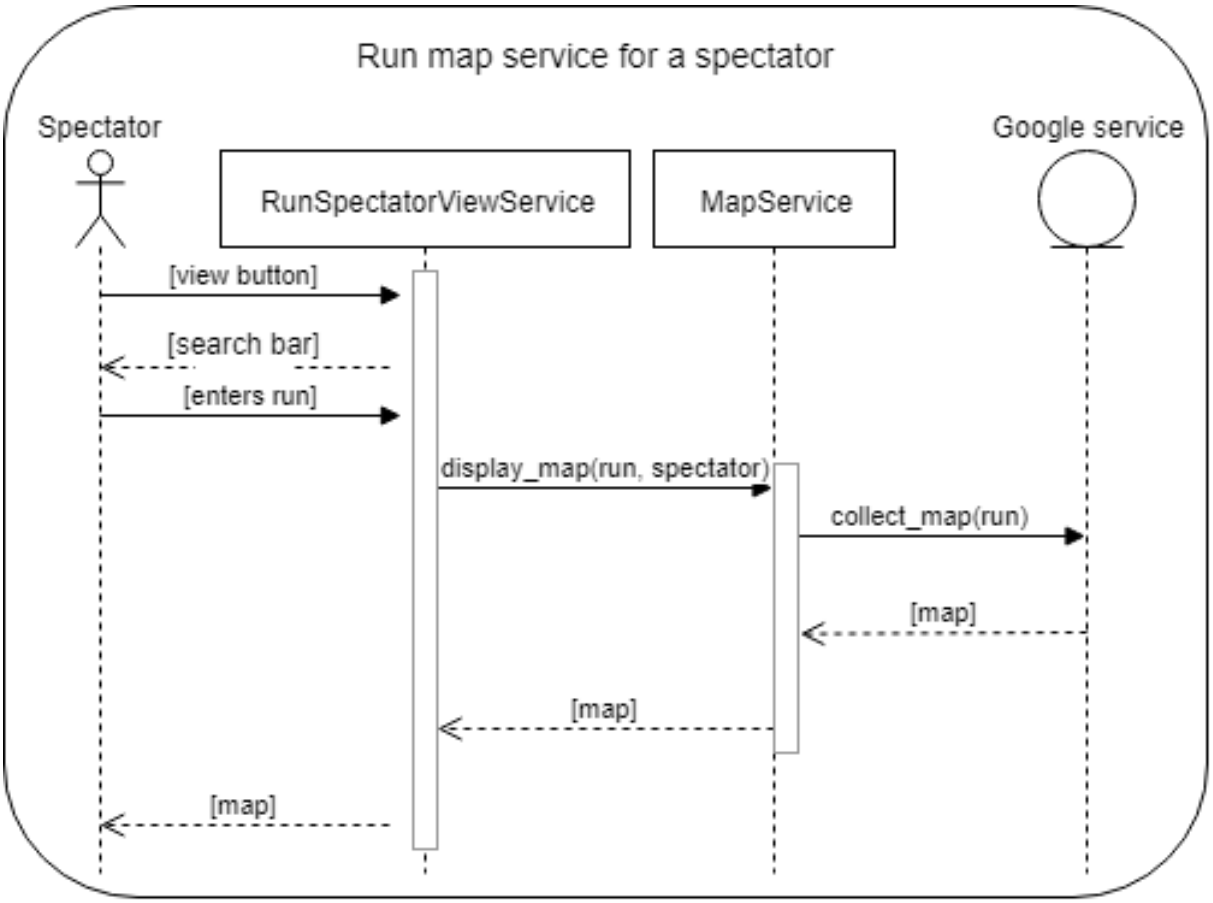


Figure 7 : Sequence Diagram to show how the map is displayed

2.6 Component interfaces

In the following diagram, the component interfaces are presented and the dependencies between the parts of the application server are shown. This information was already present in the class diagram presenting the services, but here it is shown in more detail.

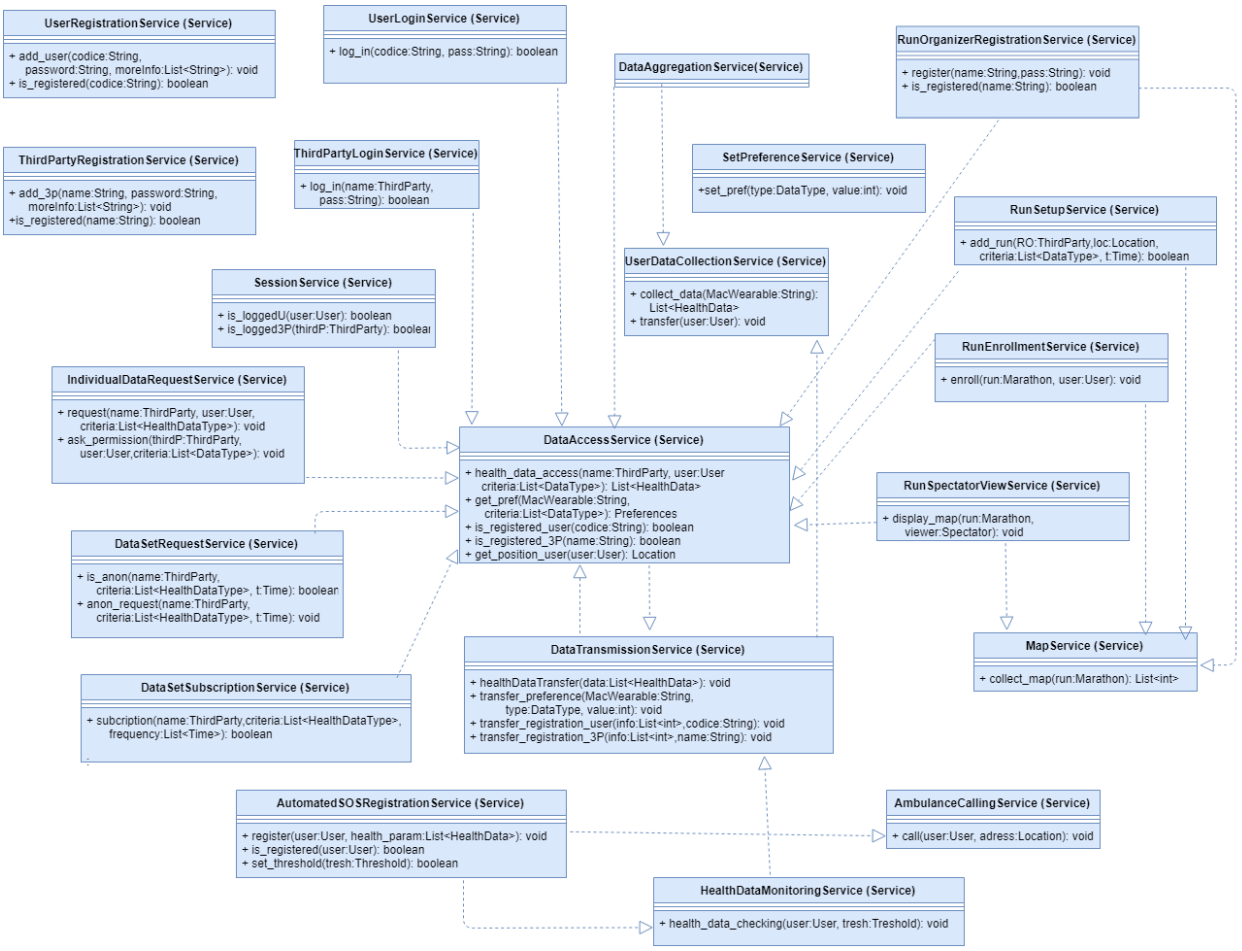


Figure 8 : Detailed Diagram to show services and their dependencies

2.7 Selected architecture style and patterns

As previously mentioned above, a three-tier architecture has been chosen for the application. This client-server architecture separates three layers, the presentation tier, the logic tier and the data storage tier. The application is basically users and third parties giving and requesting data, which justifies the client-server architecture. The main advantage of the three-tier system is that the separation into different modules allows developers to modify a specific layer without interfering with the others, for a change in requirements or in the operating system for instance.

The user of the application only deals with the presentation tier, the top layer of the mobile application. The user never has access directly to the data tier, which is better to preserve data privacy . Health data must always be secured, and thus not accessible to anyone using the application. The second layer is the logic layer, which is the core of the application. It links the two other layers, and is responsible for communicating with the external services, that is to say all devices used as wearables and map/ambulance services. Indeed, the logic layer contains all the logic needed to decide whether or not sending anonymous data required by third parties or calling an ambulance in the case of the AutomatedSOS service. Plus, all

the data collected from the users' wearables are sent to the second layer first, and then to the third layer. The third layer is basically the data layer. It only interacts with the second layer and contains the complete database of the application.

Besides a strong flexibility, this architecture is up to a faster development because of the division of work. More, it leads to an enhanced security because the clients do not interact with the health data directly, it provides less risk and confliction with unauthorized data.

2.8 Other design decisions

- A router as a linchpin

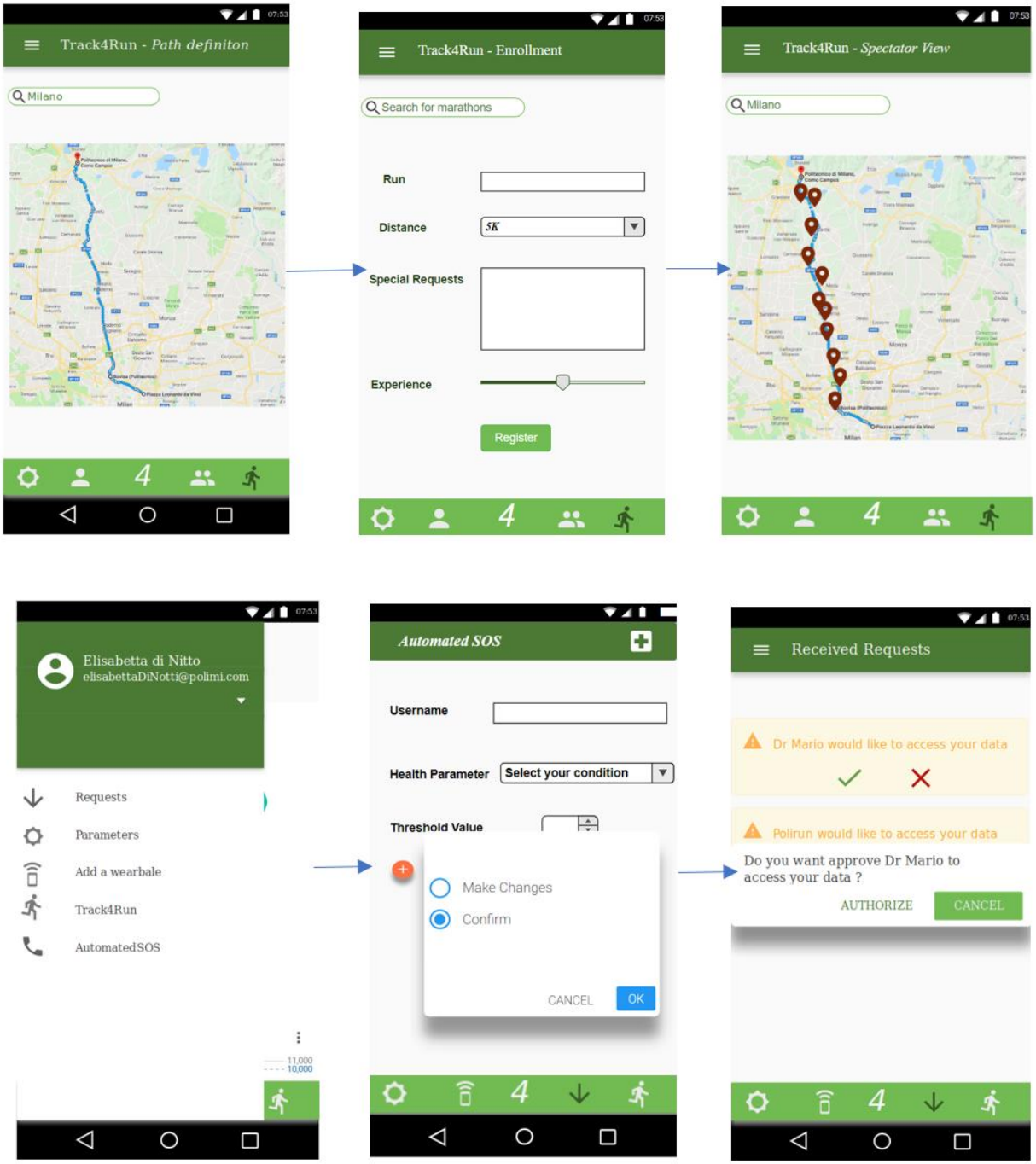
A router has been selected to play the role of a linchpin, as seen in the components diagram. It makes the link between every component of the TrackMe application, except the registration component. Its role of linchpin between components is very important to forward data between the components and to simplify the architecture.

- The DataAccessService: a key service

One of the most important service is definitely the DataAccessService, as it is the one other services call when they need to get some data. It can be seen clearly in the services class diagram, where lays in central position. DataAccessService is called by the log in and session services, which need information about users and third parties to decide whether log in some user. It is then used by most of services of Data4Help, AutomatedSOS and Track4Run. DataAccessService works together with the DataTransmissionService, which is in charge of transferring data, and have a central role to simplify the application.

3. USER INTERFACE DESIGN

Mockups for the UI were earlier presented in RASD. Here we show a control flow between the user screens for some of the functionalities.



4. REQUIREMENTS TRACEABILITY

The design of this application aims to meet all the goals and requirements that have been previously specified in RASD. Below listed are the design components to which requirements and goals are mapped:

R1: System should allow registration of users with codice fiscale and personal data (age, gender, blood type)

- UserRegistrationService

R2: System should collect and store date of the registered users

- UserDataCollectionService

R3: System should allow registration of third party with valid ID

- ThirdPartyRegistrationService

R4: System should pass requests from third party to individual based on codice fiscale

R5: System should make the requested data available to the third party only if the individual approves the request

- IndividualDataRequestService

R6: System should be able to retrieve data based on category

R7: System should be able to accept or refuse a request based on the size of data

- DataSetRequestService

R8: System should allow subscription for data requests from third party

R9: System should send requested data as soon as they are produced

- DataSetSubscriptionService

R10: System should allow subscription for elderly people by entering personal preferences (thresholds for health parameters)

- AutomatedSOSserviceRegistration

R11: Monitor health parameters of subscribed elderly people continuously

R12: Call an ambulance only when health parameters go below threshold

- HealthDataMonitoringService
- AmbulanceCallingService

R13: System should allow registration of run organizers

- RunOrganizerRegistrationService
- RunSetupService

R14: System should allow creating an enrollment process

R15: System should allow run participants to enroll for a run

- RunEnrollmentService

R16: system should create a view of the map with all run participants positions

R17: system should allow spectators to have access to the view

- RunSpectatorViewService

5. IMPLEMENTATION

5.1 Component Description with relevant services

This section explains in detail the strategy for the implementation of the complete Data4Help application along with AutomatedSOS and Track4Run services. As we have seen before, the application is divided into 8 components, where each component has a set of services that helps in its interaction with other components and in satisfying the functionalities of the system. All the components and the services it deals with are described below.

5.1.1.Registration Component

This component consists of *UserRegistrationService* and *ThirdPartyRegistrationService.* Both the services involve in the registration process of the two main clients of the system (i.e.) User and Third Party. The separation of the clients in the system is important for the application to differentiate the clients during the data request process which shall be discussed in the upcoming components.

5.1.2.Login Component

This component consists of *UserLoginService, ThirdPartyLoginService* and *SessionService.* These services involve in the login procedure of the clients and handle their respective sessions. It takes care of authorized access and keeps the functionalities disjoint. Also, it works with the Registration Component indirectly to make use of the credentials provided at the time of registration

5.1.3.Preferences Component

This component consists of *GetPreferenceService* and *SetPreferenceService.* These services help in fetching and setting of user preferences with respect to their data collection. These services come of use while collecting data from the wearable and also while data requests are being made by third parties.

5.1.4.Data Collection Component

This component consists of *UserDataCollectionService* and *DataAggregationService*. Both the services help in collecting various data from the user with the help of wearable and in finally sending the aggregated data to the database.

5.1.5. Data Request Component

This component consists of *IndividualDataRequestService*, *DataSetRequestService* and *DataSetSubscriptionService.* These services come into play when the Third-Party user requests for individual specific data or anonymized data sets. It also takes care of subscriptions made by Third-Party users for data sets.

5.1.6.Automated Sos Component

This consists of *AutomatedSOSRegistrationService*, *HealthDataMonitoringService* and *AmbulanceCallingService.* These services form the value added, AutomatedSOS functionality of the application. *AutomatedSOSRegistrationService* takes care of subscription of the user for the SOS service by taking in various health parameters and their respective thresholds. *HealthDataMonitoringService* continuously monitors the health parameters and interacts with *AmbulanceCallingService* when the data goes beyond specified thresholds and makes sure of an ambulance arriving at user location with a response time of 5 minutes. It interfaces with external ambulance calling facility for the same.

5.1.7.Run Organization Component

This component forms the next value added, Track4Run functionality of the application. It consists of *RunOrganizerRegistrationService, RunSetupService ,RunEnrollmentService , RunSpectatorViewService* and *MapService.* The first two services deals with the registration of the run organizer and the setup process for the run including, setting up the run path which interacts with external maps facility using the *MapService.* The enrollment of the run participants is done by the *RunEnrollmentService* and *RunSpectatorViewService* takes care of getting positional data of the participants during the run to create a map view for the spectators.

5.1.8.Database Component

This component handles all services related to fetching and transmitting data to the system database. It consists of *DataAccessService* and *DataTransmissionService* which work predominantly with most of the other components in the application. This is the most crucial component in the system and will be external to the application, hosted in an outside data server.

5.3 Dependencies among components

Implementation of the components followed by integration needs to take into consideration, the dependencies among the components. The various links between components, including the interfaces between the external facilities are discussed below

1.The basic data functionality satisfied by the Data collection component interfaces with the external wearable facility to collect the data from the user.

2. To store the collected data from the user, the Data collection component interfaces with the Databases component. This sets base for all the other components of the system to interact with the Databases component to fetch data for their respective functionalities.

3. Registration and Login components are indirectly linked in the sense that they both make work with client credentials for identification and authorization. Their implementation is better to be done in parallel to set up client profiles and data segregation.

4.Data Request component is dependent on the Databases component to fetch the requested data. Moreover, it needs Registration and Login components to differentiate Third-Party from User to complete the data request process.

5. Preferences component is linked to Data collection component which helps in collecting data from the user based on the preferences set and aggregating the collected data to be stored in the database.

6.AutomatedSOS component is linked to Databases component to fetch the health parameters and respective thresholds. It interfaces with the external ambulance facility to call an ambulance and the GPS to get the user's location.

7.Run Organization component is dependent on the external Maps facility to set up a path and also to create a spectator view for the run

The dependencies are further explained with the help of the diagram below

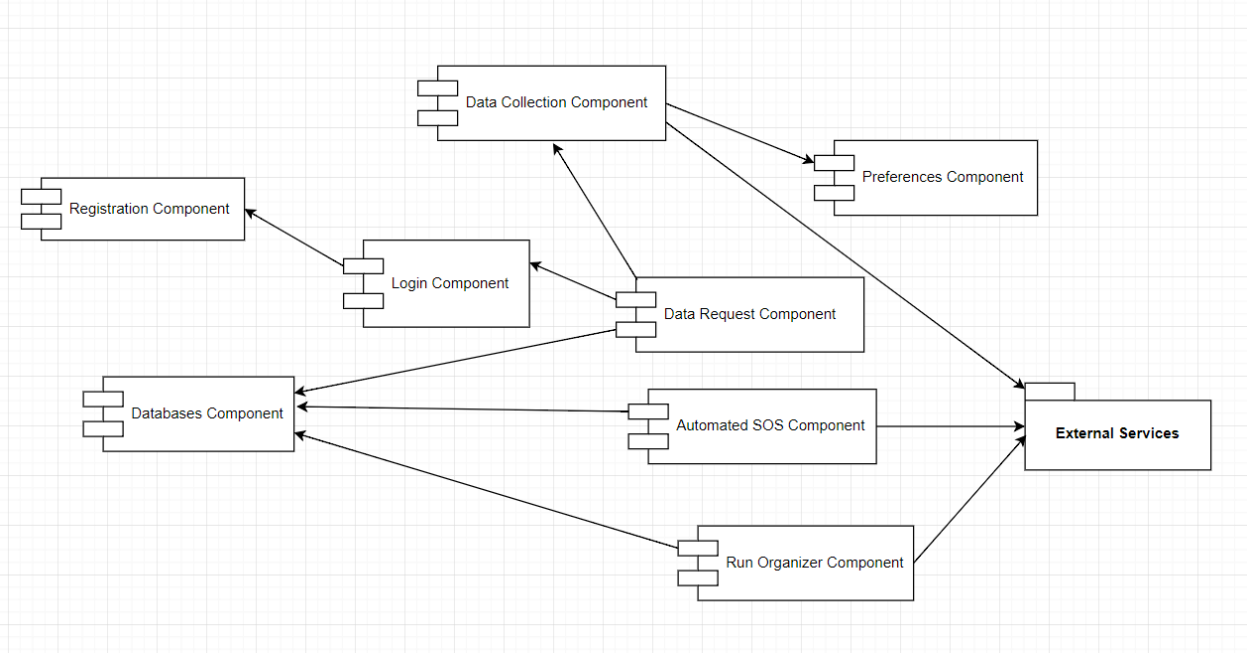


Figure 7: Component Dependency Diagram

5.4 Implementation Strategy

The strategy chosen for the implementation of the system is Top-Down implementation, which is developing the core of the system and build the pieces on top of it with additional modules each satisfying a functionality. This helps in code flow control and keeping a check on the dependencies between modules. This strategy would be further useful during integration followed by system testing approach along with prototyping. The Top-Down strategy is further explained with respect to each component.

Data collection component is the base with which the data driven application can build to form a complete system. This component requires services to interface with the wearable facility and collect data from the user. To begin with the implementation, it should collect data from the wearable and send it to the application server. It needs to work real time and the data need not be filtered.

Databases component needs to be implemented next in order to be able to store the collected data. This storage is the most important requirement as the data will be used across components and also requested by third parties at any time based on individual or anonymized data sets.

Registration and Login components are both required in order to differentiate the two main users of the application (i.e.) Third-Party and user. The registration component collects user credentials which will be authenticated during login process which then starts a new session based on the

client privileges and sets the authorization . This separation of control flow needs to be maintained throughout the application and it involves the data confidentiality between clients

Data Request component satisfies the first feature of the application which takes requested from Third-Party for data based on specific individual or anonymized data sets. This implementation makes use of the differentiation of clients and involves transferring data requests from the Third-Party to the user for approval and the requested data to Third-Party once approved. It also needs to fetch categorical data based on collected user data, access the anonymity of the data sets requests and be able to decide to approve or reject the request based on size of the data set requested.

Preferences component builds on the data collection component in helping the system to filter the data while collection from wearable based on user preferences. The collected data will have to be based on the preferences set during registration process and thus has to be specific to user profile. This addition reduces the data collection and avoid storing irrelevant data keeping the confidentiality of the user data secure.

AutomatedSOS component is a complete component by itself and makes of the user data collection service that is already implemented. The subscription service of the component has to be developed to take health data and thresholds to be monitored for the elderly user. These should then be compared to the real time data collected from the user and a trigger call should be made to call an ambulance using the external ambulance calling facility when the monitored health parameters go beyond threshold. The interface with the external facility is one of the important modules in this component plays the role of satisfying the feature.

Run Organization component has multiple modules to be implemented and can begin with registration of the run organizer which would involve setting up a feature to interface with maps service and set up a path for the run. The run organizer should also be able to setup the enrollment process which would later be made available for run participants to register for the run. Moreover, as an additional functionality, a spectator view of the run should be developed to show the run situation during the run by using the real time positional data from user and showing it on a map. The spectator as a user of the application should be given appropriate access to view the same.

5.5 Integration Strategy

Once the components are implemented as discussed before, in order to bring them together and integrate to form a complete system, the dependencies between components have to be taken into account. Each component interfaces with one another with the help of the services that link them and hence the completion of these services plays a major role in deciding the order of components to be integrated.

1. Data collection component needs application interfaces to be able to talk to wearable which is an external component and collect data
2. Databases component needs access and transmission services to be complete in order to be able to store collected data
3. Registration and Login components need to be able to get information from clients and form user profiles while updating the database with corresponding data.
4. Data Request component needs separate services for each type of data request made by Third-Party. It needs services from databases component to be able to fetch data according to request and also look for data that will be produced in future to match Third-Party dataset subscriptions.
5. Preferences component works as an add on functionality for data collection component. The preferences will be stored in user profiles during registration and this needs to be applied at data collection in order to filter the data being collected and also categorize them. At this phase data4help can work as an application with single data service and this could be used as a first prototype for system testing which shall be explained in detail in the next section.
6. AutomatedSOS component needs basic user registration followed by data collection in order to begin integration. The service to monitor the data continuously needs to be complete before integrating it to the data4help service. The integration also involves the external ambulance calling facility that should be in place to call an ambulance given a user's location. This point marks the completion of the next prototype available for system testing.
7. Run Organization component should be able to setup a run with its organizer and participant's enrollment and spectator view. Existing map facilities can be leveraged for this component and integration with data collection component to fetch data specific to the run. Once the run organizer component, the application is complete and ready to be sent for end to end solution testing.

The integration strategy followed is called as incremental integration as it progressively aggregates the functionality of the system. This kind of integration also facilitates development and quality assurance team to work in parallel with prototyping. This setup works in finding errors at each level of code implementation which increases quality of the application.

5.6 Testing

Test strategy to be followed is Top-Down. This works well with implementation strategy adopted for the system. When each module gets completed with the services to link with the next component, one round of unit testing followed by integration testing after the integration of two components should be done. The unit testing will ensure the verification of functionality of each individual component and the integration testing done at each step of integration will ensure

verification of errors that might crop with interfacing of components. The results of this testing phase should be well documented because they will used at a later stage for debugging purposes.

Here below are few important feature functionalities to be verified at each integrated module.

1. Real time data is collected from user
2. Collected data is sent to database and made retrievable
3. Client authentication and authorization
4. Data requests handled without violating application constraints
5. Data collection filtered based on preferences
6. Continuous monitoring of health data for subscribed users
7. Ambulance calling functionality with real time location
8. Setup run path and run enrollment procedure
9. Spectator view for clients with real time positional data

Unit testing followed by integration testing gives the complete application with all modules implemented and integrated as discussed.

This final prototype of the application needs to undergo one round of system testing and solution to check the functionality of the entire system and as a complete solution. This also includes negative testing to make sure exceptions are gracefully handled by the application.

Here below are few negative scenarios to be handled by the application.

1. Wearable not sending across data from the user
2. Database issues like integrity and consistency.
3. User profiles mismatched or authentication authorization failures from server
4. Data request lacking information headers to comprehend
5. Dropped data requests due to over demand
6. Timing issues related to data subscription
7. Health data fluctuation
8. Unavailability of ambulance
9. GPS errors

Solution testing can use data simulation as a tool to test the reliability and scalability of the application. Data driven application needs sample data to check for robustness and correctness at each functionality. These simulations completely emulate a deployment and thus helps in finding defects earlier.

6. EFFORTS SPENT

Work Description	Sankari Gopalakrishnan	Louis Lesieur	David Brellmann
Purpose, scope, definitions	1	1	1
Components and Services	3	4	6
Class View	2	5	4
Runtime View	3	5	4
Implementation	6	3	2
Integration and testing	5	2	3

7.REFERENCES

Specification Document “Mandatory Project Assignment AY 2018-2019”

Remy-manuo.no :

http://remy-manu.no-ip.biz/UML/Cours/coursUML8.pdf

http://remy-manu.no-ip.biz/UML/Cours/coursUML3.pdf

http://remy-manu.no-ip.biz/UML/Cours/coursUML9.pdf

developpez.com :

https://laurent-audibert.developpez.com/Cours-UML/?page=diagrammes-composants-deploiement

https://laurent-audibert.developpez.com/Cours-UML/?page=diagramme-classes

https://laurent-audibert.developpez.com/Cours-UML/?page=diagrammes-composants-deploiement

UML-diagram.org

https://www.uml-diagrams.org/deployment-diagrams-overview.html

Wikipedia :

https://fr.wikipedia.org/wiki/Architecture_trois_tiers