POLITECNICO DI MILANO

Department of Electronics, Information and Biomedical Engineering

MSc. Course in Computer Science and Engineering

Software Engineering 2

TrackMe

Design Document

Instructor: Prof. Matteo ROSSI

Authors: Buse GUNAY (matr. 903822)

Ezgi TASTI (matr. 917104)

Talip TURKMEN (matr. 915609)

Version 1.0 - 10.12.2018

Politecnico di Milano

Table of Contents

Chapter 1

1. Introduction

1.1 Purpose

The purpose of this document is to provide a detailed description of the TrackMe system. This will be done by a detailed presentation of the proposed solution and its purpose, listing its goals, and the requirements and assumptions through which they will be achieved. The document is meant to be used by individuals such as elderly people and runners; and, by the third parties designated with the task of creating the specified system, mainly the system and requirements analysts, the project managers, software developers and testers.

The TrackMe system is designed as a software application used to facilitate the monitoring of the health status of people and to transmit information to third parties. Also, it ensures that information is kept and shared both individually and anonymously.

Health status monitoring is critical for people who are particularly sensitive to health, want to share health information with their hospital and want to view this information in detail. If necessary, it is vitally important to direct a health service to an individual's position. TrackMe offers all these services together and makes it easier for individuals to track their health status for both themselves and third parties.

TrackMe also provides a platform for running. Thanks to this application, the organizers can specify the run and define the path for the run, while at the same time the spectators can easily track the runners on the map.

1.2 Scope

TrackMe is a software application which is designed to facilitate mechanisms for requesting and monitoring data. It supports different type of users which can be grouped as either “individual” or “third party”. Based on the user type, it offers a wide variety of functionalities. For example, while it provides mechanisms for data request for third parties, it allows individuals to register for a service which is responsible for providing an immediate help in an emergency case.

Health data of the individuals are the skeleton of TrackMe and every functionality that is offered by the application relies on it. To get this huge data, TrackMe works in cooperation with the smart devices. Basically, it collects the data which has been captured by the smart devices of the individuals who are already registered.

Three different services named as; Data4Help, AutomatedSOS and Track4Run are offered by TrackMe. Each of these services provides different functionalities both for the individuals and third parties. As a brief explanation of these services, the first one; Data4Help, is responsible for the data request of third parties, the next one; AutomatedSOS, is responsible for the data monitoring of individuals and providing immediate help to those individuals in an emergency case and the last service; Track4Run, is responsible for providing mechanisms to organize, register and spectate runs which have been organized by the third party users of TrackMe.

1.3 Definitions, Definitions, Acronyms, Abbreviations

API: Application Programming Interface

CPU: Central Processing Unit

DB: Database

DBMS: Database Management System

DD: Design Document

GPS: Global Positioning System

GUI: Graphical User Interface

MVC: Model View Controller is a design pattern used for GUIs

RASD: Requirements Analysis and Specifications Document

UI: User Interface

REST: Representational State Transfer

1.4 Revision history

This is the version 1.0 of DD.

1.5 Reference Documents

- Specification Document: Mandatory Project Assignment AY 2018-2019.pdf
- Software Engineering, 10th edition, Ian Sommerville
- Lecture slides of Software Engineering 2
- Example Documents:
  DD to be analyzed.pdf

1.6 Document Structure

Chapter 1 presents a brief introduction to the design document of the project. Under this chapter, the purpose and scope of the document are discussed. Also, the definitions and acronyms that were used in the document are described.

Chapter 2 gives an architectural design of the project. This chapter includes the general architecture and high-level architectural structure of the system, component view, deployment view and detailed sequence diagrams from this component view.

Chapter 3 describes the algorithm design of the project. This chapter includes algorithm designs about request flow, checking threshold and checking anonymity constraints for anonym requests. All the algorithms are written in pseudo code in order to focus on the most critical points and to make them easier to understand.

Chapter 4 refers to the mock-ups already presented in the RASD document.

Chapter 5 includes how the requirements that have been defined in the RASD map to the design elements that are defined in this document.

Chapter 6 explains the implementation plan of subcomponents of the system. In addition, it includes how to be planned to integrate these subcomponents and test the integrated parts.

Chapter 7 involves the effort spent by each group member while working on this project.

Chapter 8 lists the references that were used in the project.

Chapter 2

2. Architectural Design

2.1 Overview

Since our application will be used by large-scale of users, we have preferred to use a "Multi-Tier Client-Server Architecture"

The application will be composed of 3-tiers which are called as; "Presentation Tier", "Logic (Application Tier)" and finally "Data Tier".

The below diagram is intended to give a brief description of the structure of the tiers. For the simplicity of the diagram, the two "call-response" arcs pairs; (one from "Application Server" and the other from the "Web Server") are represented as a single arc pair that is going from the entire "Application Tier". The purpose of doing is to make the diagram more readable and not to make the diagram messy with the arcs that are going over all the way on the diagram.

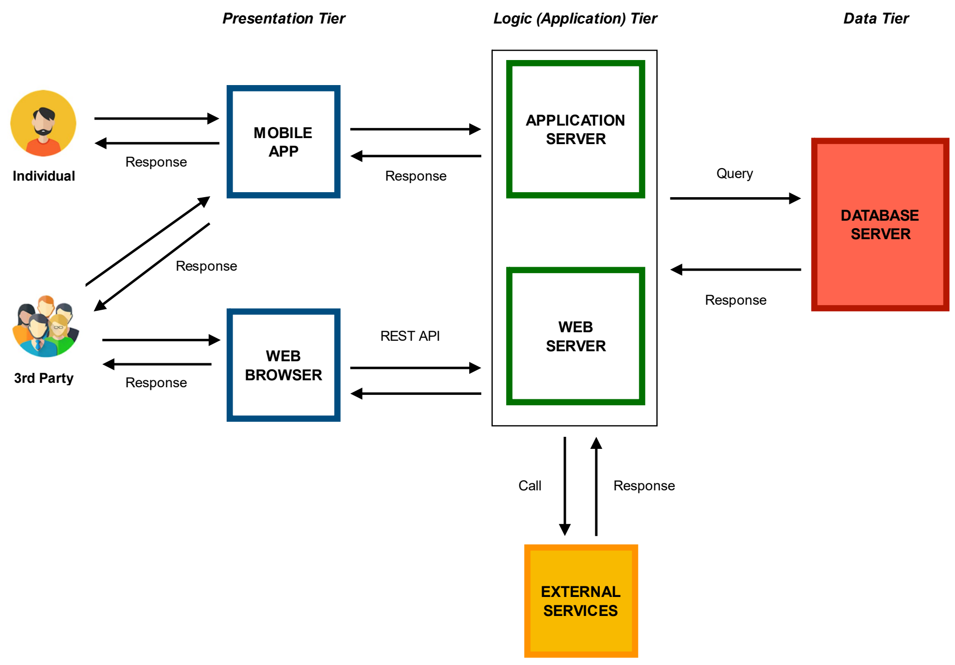


Figure 1 - 3-Tier Archtectural Design

Presentation Tier: Presentation tier is the topmost layer which provides a graphical user interface for the user. Since our application has two different interfaces; one for the mobile application and one for the web browser, we have two sub-components inside this tier. In order to be more precise about these sub-components; we can say that since “Web Browser” will only be used by third parties, we have included only one “request/response” pair between the “3rd Party” and the “Web Browser”. On the other hand, since “Mobile Application” will be used by both the “Individual” and “3rd Party”, it will respond to both type of the clients.

Logic Tier: Logic tier can be referred to as the brain of the whole architecture. This layer is responsible for coordinating the application and processing the commands. It receives the client request from the presentation layer, turns it into a query for the data tier, then sends the respond of the data tier back to the presentation layer. As can be seen from the above diagram, we have preferred to use two different sub-components within this tier. We preferred this because we have presented two different user interface components in our presentation layer and we thought that it will be more accurate to separate the servers that those components will interact with. We have introduced the “Web Server” for the “Web Browser” and “Application Server” for the “Mobile Application”. Since this layer will drive the core capabilities of our application, we have also included the link between our application and the external servers within this tier.

Data-Tier: Data-tier can be named as the layer of data access. It includes the database in which our whole data is stored. It is in connection with the logic layer and responsible for returning responses for the queries that are requested by the application tier. Also, this layer includes mechanisms for data persistence.

2.2 High Level Architecture

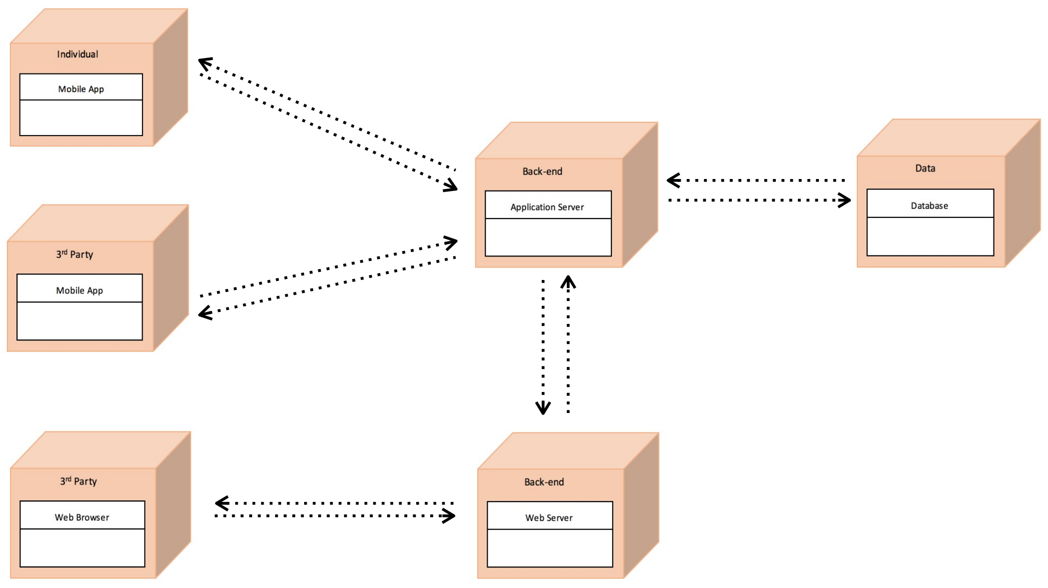


Figure 2- High Level Architectural Design

The above diagram gives the whole representation of the architecture of our application. As can be seen from the diagram two sub-components of our logic tier; the “Application Server” and the “Web Server” are at the central part of the architecture. These components work as a bridge between the client and the database, and they control the interaction between them. Both the “Mobile App” and the “Web Browser” use these components (“Application Server” used by “Mobile App” and “Web Server” used by “Web Browser”) directly in order to perform any operation within the application.

2.3 Component View

In the component diagram shown below, the components in the project are examined in more detail together with the application server. In order to make the display more general and understandable, the interface provided by each component and the interface it requires are shown together. This will also help to better understand the relationship between the components. Other external services except for the database server and DBMS are not included in this diagram, because they are as black-boxes for this diagram and at this stage, their structure does not concern us. We are only interested in the services they provide.

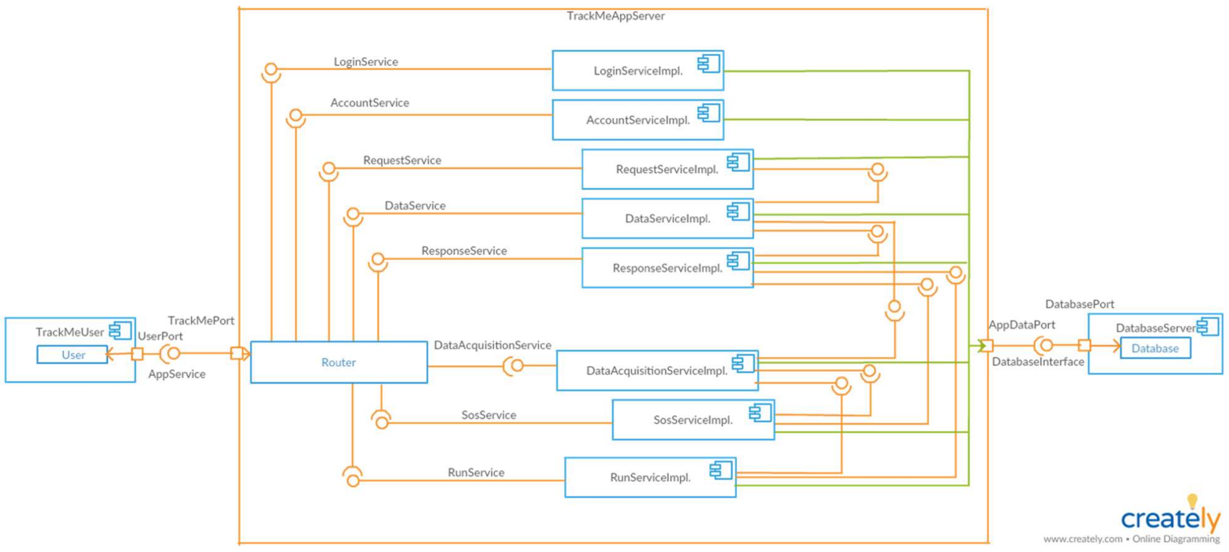


Figure 3- Components

Component Diagram Services:

- **LoginService**: responsible for the authentication of the user.
- **AccountService**: provides functionalities to view his/her own history, track own data, update and delete own account for users.
- **RequestService**: responsible for receiving the request and identifying the request type. Then sends the request to data service for data preparation.
- **DataService**: provides functionalities for preparing to requested data.
- **ResponseService**: provides functionalities for receiving the prepared data from data service. Then sends this data.
- **DataAcquisitionService**: provides functionalities for normalizing the data coming from smart device and store data. Also, provides data for the other services.
- **SosService**: provides functionalities for getting individual data, comparing the data with the threshold and informing hospital based on the results of the comparison.
- **RunService**: provides functionalities for registering the run, defining a path and spectating runners.

The requests received by the user to the system are evaluated by the router. The router determines whether the received request is valid and if this request is intended to a component in the system, it sends the request to the component. Components in the diagram communicate with each other and work together to respond to requests from the user.

The following two diagrams, created from the component diagram, are designed to further describe the components in the application server. The relationship of the application server with the service components is clearly seen.

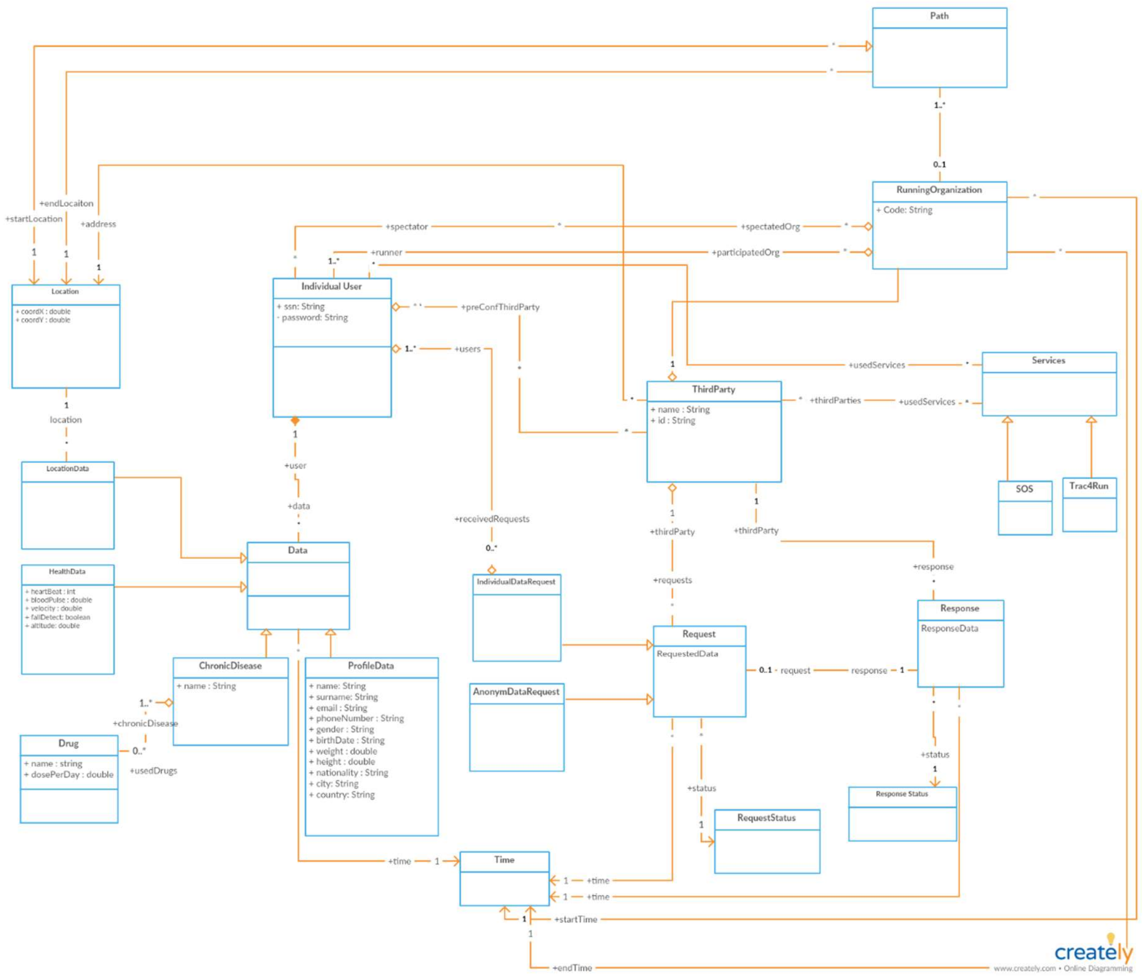


Figure 4- Class Diagram

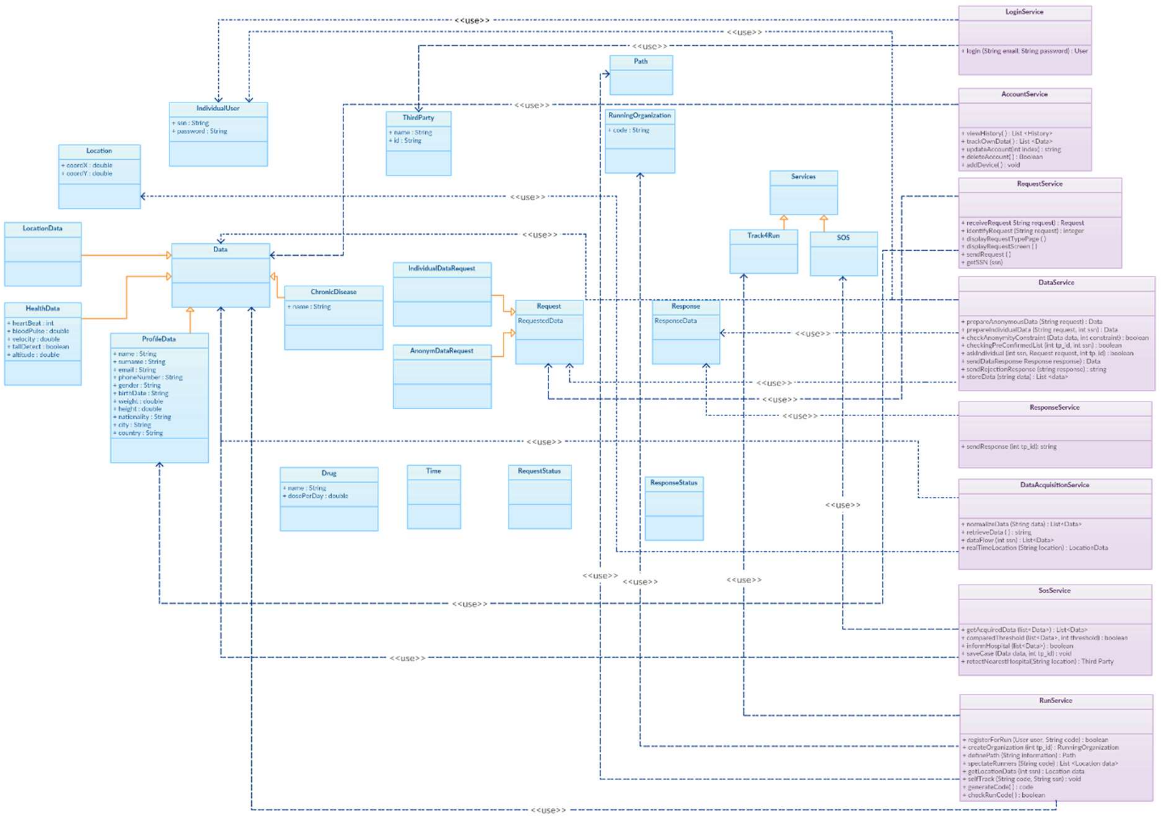


Figure 5- Component View

2.4 Deployment View

The deployment diagram shows the architecture of the system. As mentioned before, the system architecture is 3-tier architecture. However, in addition to presentation, logic and data layer there is also intermediary web tier to support third parties to access the system by the web browser. So, we can divide system architecture into 4 tiers.

- The first tier is the presentation tier: It's composed by the mobile applications and web application which is used by the third parties. The mobile application has Data Acquisition Component to retrieve data from smart devices.
- The web tier contains web server which is implemented by Apache Tomcat and communicates the application tier using RMI.

- The application tier consists of Glassfish which handles all the business logic, connecting external services and data tier.
- The data tier is mainly composed by the Database Server. The connection between the application tier and data tier is performed via JDBC connector.

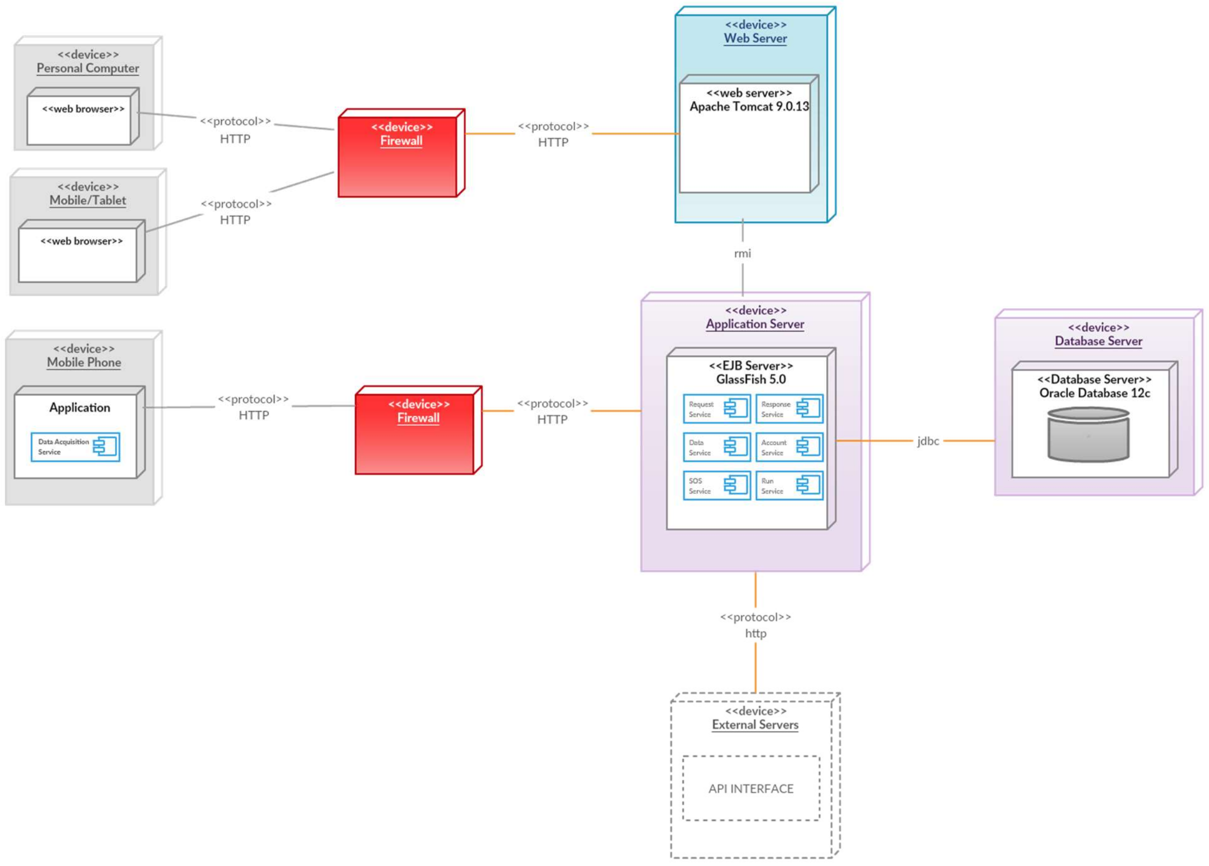


Figure 6 - Deployment View

Recommend Implementation

- **Client Tier**: The mobile application can be implemented using a cross-platform development tool such as Cordova which is an is an open-source mobile development framework. It allows you to use standard web technologies - HTML5, CSS3, and JavaScript for cross-platform development.

- **Web tier**: Frontend of the web application can be implemented using HTML5, CSS(Bootstrap), Angular.
- **Application tier**: The EJB application server uses java beans to implement business logic and stores data to a persistent database using JPA.
- **Data-tier**: The database can be implemented with Oracle database 12c.

2.5 Runtime View

2.5.1 Data Acquisition

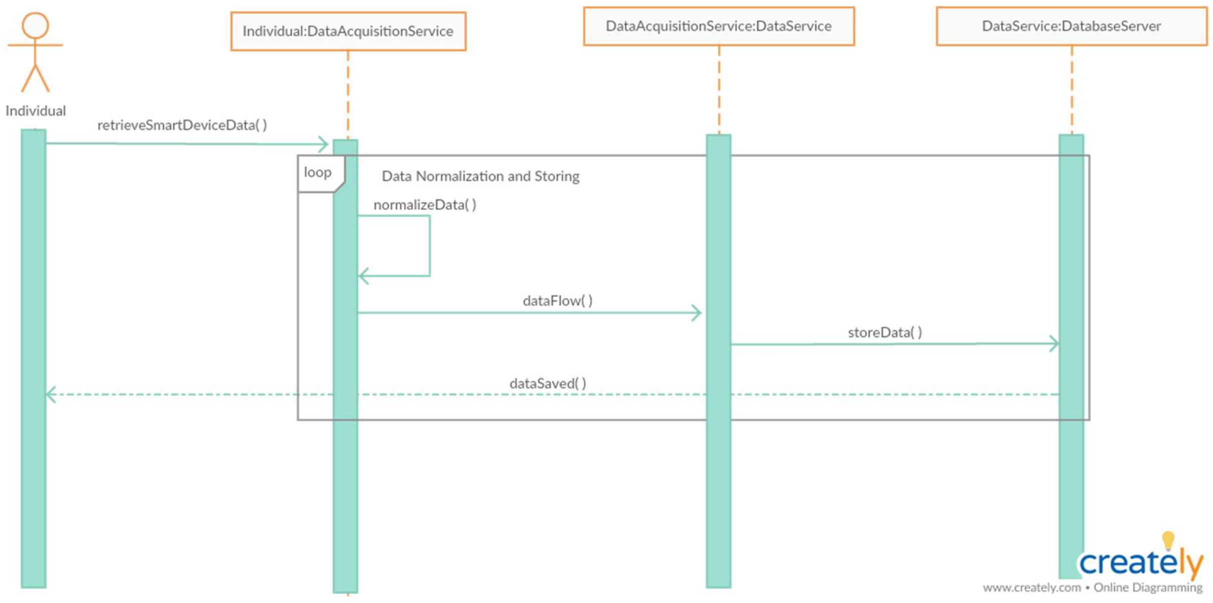

Individual
Individual:DataAcquisitionService
DataAcquisitionService:DataService
DataService:DatabaseServer
retrieveSmartDeviceData( )
loop
Data Normalization and Storing
normalizeData( )
dataFlow( )
storeData( )
dataSaved( )
creately
www.creately.com • Online Diagramming

Figure 7- Sequence Diagram of Data Acquisition

In the first part of the operation, data received via the smart device on the individual, it is transmitted to the DataAcquisitionService. The data received is normalized in the DataAcquisitionService. Then, it is sent to the DatabaseServer for storage. After that, the message of the received data is sent to the individuals.

The process of retrieving and saving the data is shown in a loop because there will be a continuous flow of data from the smart device and the incoming data will be stored in the database.

2.5.2 Request Individual Data

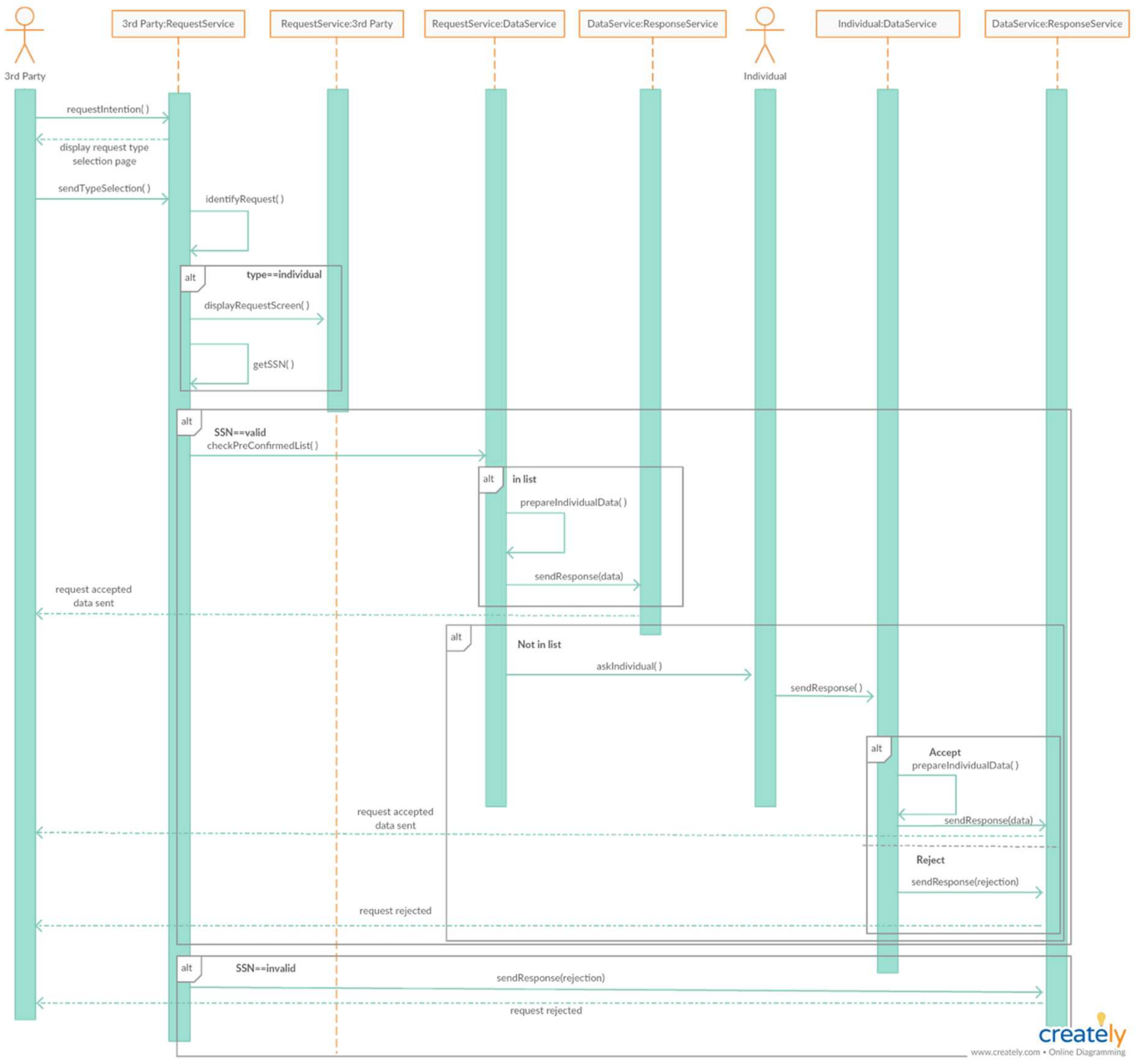


Figure 8- Sequence Diagram of Individual Data Request

This diagram describes the section on the transfer of data from individuals to third parties. First of all, the request intention from the third party is sent to the request service. Then, request service shows the request type selection page to the third party. In request service, this request is identified, because this request can be individual or anonymous. In this diagram, we are only interested in the individual request. SSN of the individual is checked whether it is valid or not in request service. If the SSN is not valid, a notification is sent to the third party that the request was rejected. If the SSN is valid, it should be checked whether the third party sending the request is on the individual's pre-confirmed list. If this third party is on this list, the data service prepares the requested data and sends it to the third party. If the third party is not in this list, the consent of the individual is required to send the data by the system. The response of the individual is sent to the data service. If the individual accepts to submit his/her own data to the third party, the data is prepared and sent to the third party. Otherwise, if the individual rejects data sharing with them, this rejection is sent to the third party.

2.5.3 Request Anonymous Data

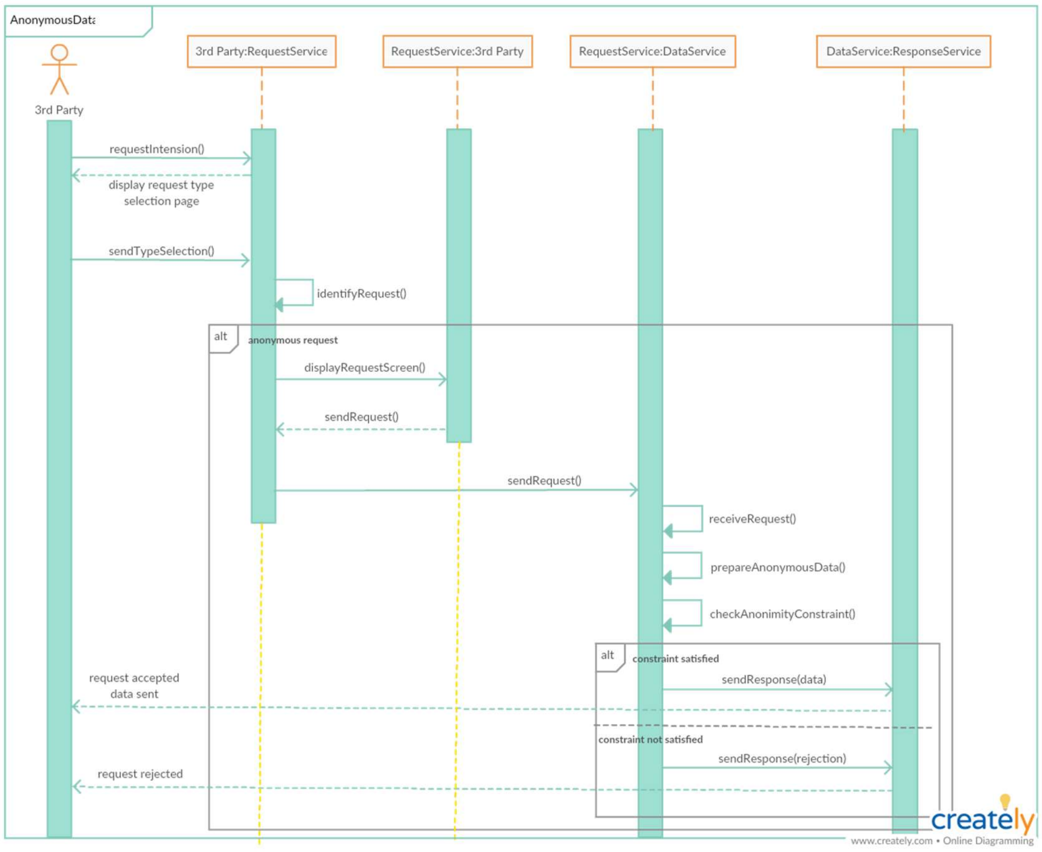


Figure 9- Sequence diagram of Anonym Data Request

The above figure represents the sequence diagram of Anonymous Data Request. In order to make an anonymous data request, it is first expected from the 3rd party to perform a request intention. By making this request intention, 3rd party informs the system that it wants to request data. Then, this intention of the 3rd party is captured by the RequestService and a page asking for the request type; individual or anonymous data is displayed to the 3rd party. After this step, the selection of the 3rd party is identified by the RequestService and if the request type is anonymous, the complete version of the request that 3rd party wants to perform is asked. Once again, RequestService receives this request and forwards it to DataService. DataService prepares the requested data and checks its anonymity constraint. If the requested data satisfies the anonymity constraint, then data is sent to the ResponseService in order to be shared with the 3rd party. However, if the anonymity constraint does not satisfy, then the knowledge of request rejection is sent to ResponseService and ResponseService notifies the 3rd party about the rejection of its request.

2.5.4 Automated SOS

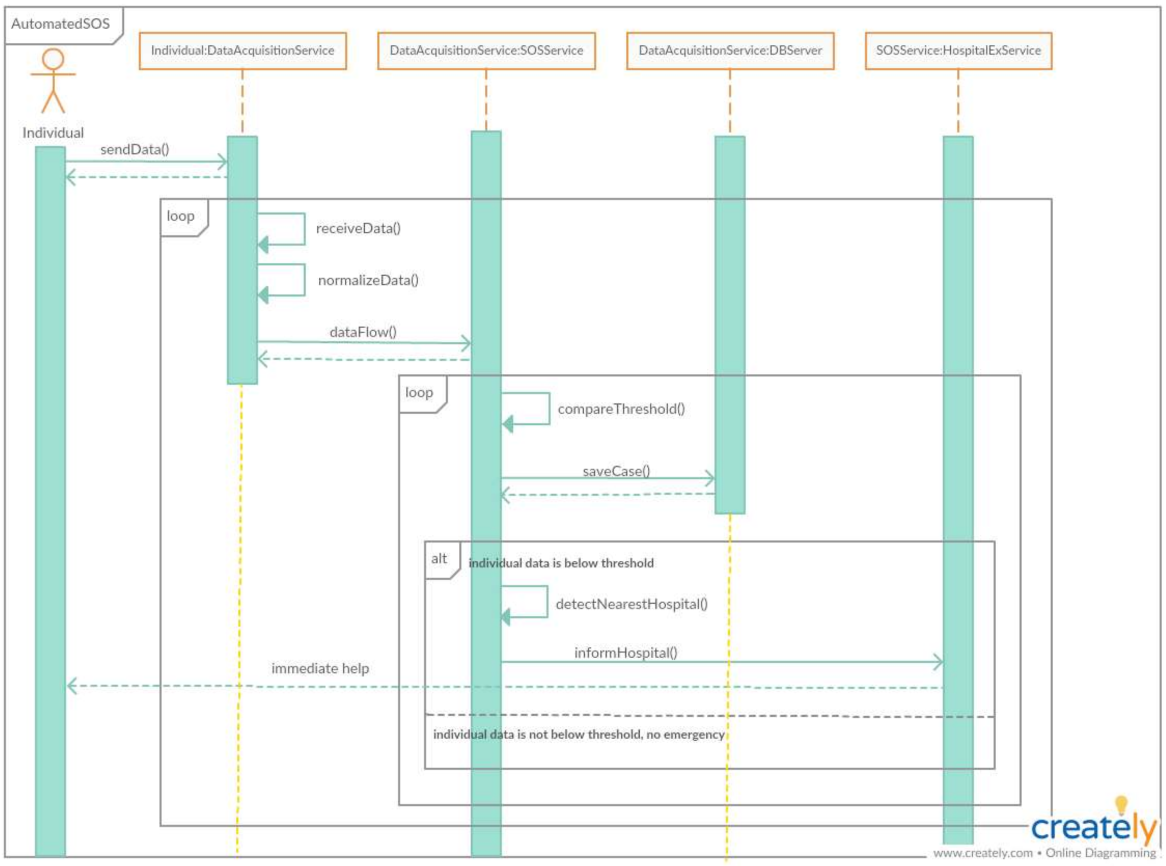

AutomatedSOS
Individual:DataAcquisitionService
DataAcquisitionService:SOSService
DataAcquisitionService:DBServer
SOSService:HospitalExService
Individual
sendData()
loop
receiveData()
normalizeData()
dataFlow()
loop
compareThreshold()
saveCase()
alt
individual data is below threshold
detectNearestHospital()
informHospital()
immediate help
individual data is not below threshold, no emergency
creately
www.creately.com • Online Diagramming

Figure 10- Sequence diagram of AutomatedSOS

The above figure represents the sequence diagram of AutomatedSOS. On the AutomatedSOS, there is a continuous data flow between the individual and the system.

During the AutomatedSOS, the data of the individual is first received by the DataAcquisitionService. Then, this unstructured data is normalized by the DataAcquisitionService and forwarded to the SOSService in order to decide whether there is an emergency case or not. As the very first step of the SOSService, compareThreshold() is invoked and the current health data of the individual is compared with a threshold value. Based on the result of the comparison, SOSService branches into 2 conditions. If it detects an emergency case, it detects the nearest hospital and informs the hospital through an external service called as HospitalExService. On the other hand, if it does not detect an emergency, it will continue to go on with data monitoring.

2.5.5 Register for Run

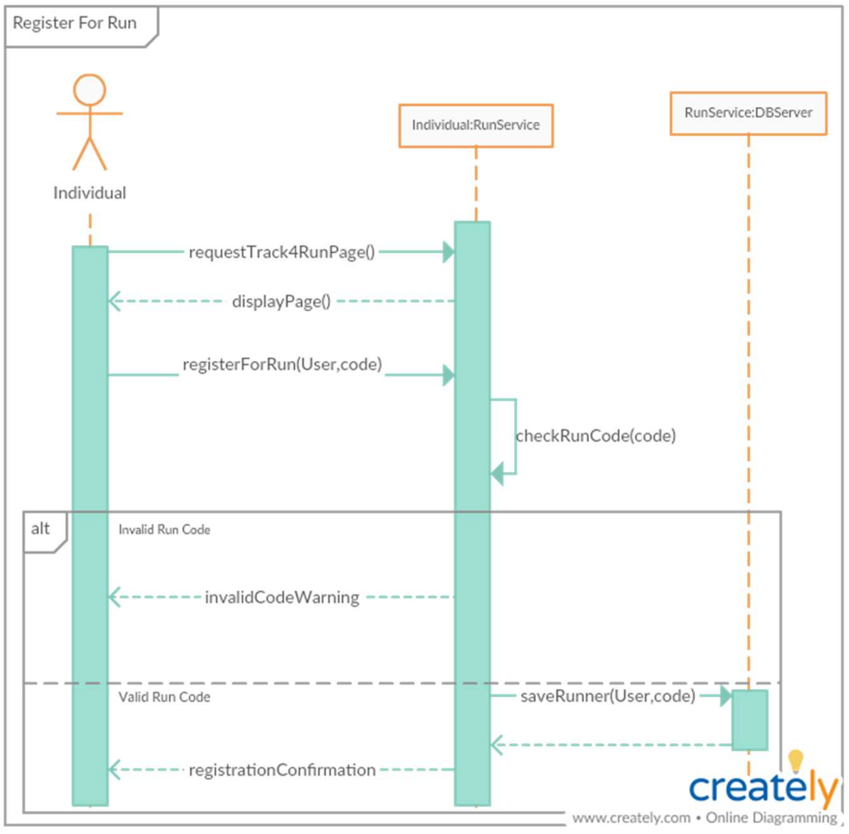


Figure 11 – Sequence diagram of register to runners

The above figure represents the sequence diagram of “Register for the run”. Before individual register to run s/he must have a code of run. System checks code which is entered by the user and according to validation process user is either registered run or warned by invalid code warning.

2.5.6 Spectate Runners

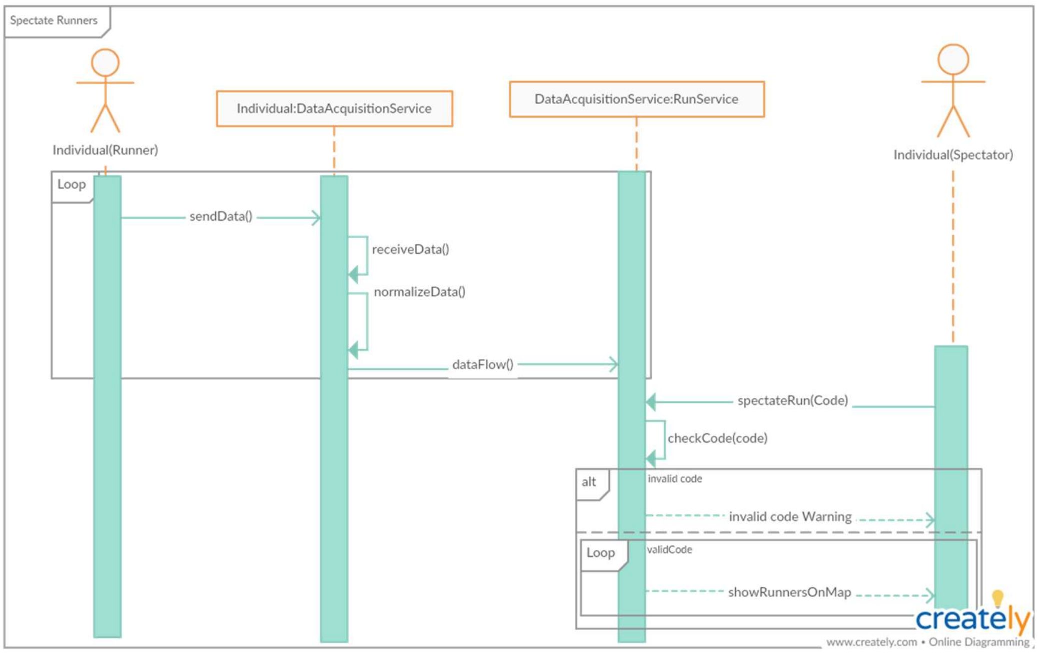


Figure 12- Sequence Diagram of spectate runners

The above figure represents the sequence diagram of “Spectate Runners”. During the run system retrieves location data by “Data Acquisition Service”. When another individual or third-party request to spectate run s/he enters the code and “Run Service” displays runner current location on the map.

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 more clearly.

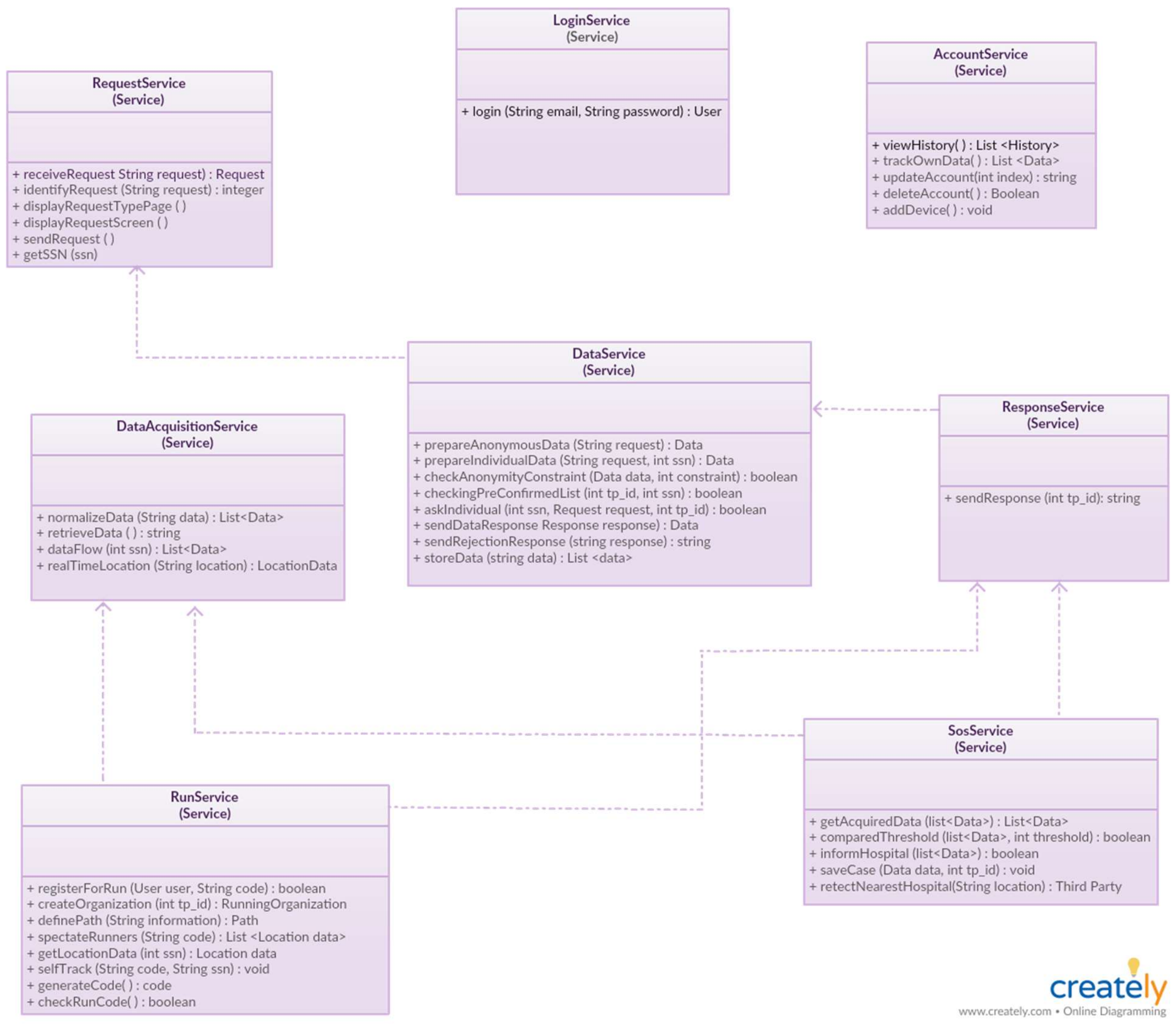

LoginService (Service)
+ login (String email, String password) : User
AccountService (Service)
+ viewHistory( ) : List <History>
+ trackOwnData( ) : List <Data>
+ updateAccount(int index) : string
+ deleteAccount( ) : Boolean
+ addDevice( ) : void
RequestService (Service)
+ receiveRequest String request) : Request
+ identifyRequest (String request) : integer
+ displayRequestTypePage ( )
+ displayRequestScreen ( )
+ sendRequest ( )
+ getSSN (ssn)
DataService (Service)
+ prepareAnonymousData (String request) : Data
+ prepareIndividualData (String request, int ssn) : Data
+ checkAnonymityConstraint (Data data, int constraint) : boolean
+ checkingPreConfirmedList (int tp_id, int ssn) : boolean
+ askIndividual (int ssn, Request request, int tp_id) : boolean
+ sendDataResponse Response response) : Data
+ sendRejectionResponse (string response) : string
+ storeData (string data) : List <data>
DataAcquisitionService (Service)
+ normalizeData (String data) : List<Data>
+ retrieveData ( ) : string
+ dataFlow (int ssn) : List<Data>
+ realTimeLocation (String location) : LocationData
ResponseService (Service)
+ sendResponse (int tp_id): string
SosService (Service)
+ getAcquiredData (list<Data>) : List<Data>
+ comparedThreshold (list<Data>, int threshold) : boolean
+ informHospital (list<Data>) : boolean
+ saveCase (Data data, int tp_id) : void
+ retectNearestHospital(String location) : Third Party
RunService (Service)
+ registerForRun (User user, String code) : boolean
+ createOrganization (int tp_id) : RunningOrganization
+ definePath (String information) : Path
+ spectateRunners (String code) : List <Location data>
+ getLocationData (int ssn) : Location data
+ selfTrack (String code, String ssn) : void
+ generateCode( ) : code
+ checkRunCode( ) : boolean
creately
www.creately.com • Online Diagramming

Figure 13-Component Interfaces

2.7 Architectural Styles and Design Patterns

2.7.1 Design Patterns

Since we are using 3-tier architecture separation of concerns is very important when implementing software. Separation of concerns is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern. A concern is a set of information that affects the code of a computer program. Each component is concerned with only one concern. Also, separation of concerns eases developers to reuse and maintain components.

MVC is one of the architectures which ensures separation of concerns. There are three main parts of MVC architecture. These are Model, View and Controller:

Model is the component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. In TrackMe, Model includes all data related logic.

View is a component which is used for all the UI logic of the application. In TrackMe, View includes user interfaces which clients interact with.

Controller acts as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. In TrackMe, Controller includes router which directs the request to a related component.

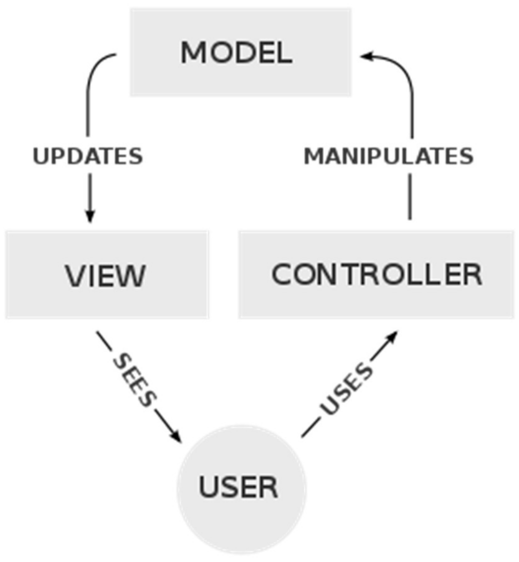


Figure 14-Diagram of interactions within the MVC pattern

In order to increase reusability of components, ease maintenance and ensure separation of concerns MVC is the main architecture of TrackMe.

In addition to MVC pattern following patterns is used in some components for specific purposes:

Factory Pattern: is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. In our system in order to support different types of database server factory pattern is used as DB Factory.

Builder Pattern: is a design pattern designed to provide a flexible solution to various object creation problems in object-oriented programming. The intent of the Builder design pattern is to separate the construction of a complex object from its representation. Builder Pattern is used in creating Anonym and Individual Data Requests by third parties.

Observer Pattern: is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. Observer pattern is used in SOS service to implement checking threshold values and send a notification.

Facade Pattern: is a structural pattern which hides the complexities of the system and provides an interface to the client using which the client can access the system. For routing requests of the web server to application server, facade pattern is used in TrackMe.

2.7.2 Other Design Choices

TrackMe uses Map services in order to display users' location on the map, especially Track4Run service. For this reason, the external map service should be integrated. Google Maps is our design choice to be integrated services since it provides accurate and maintainable location services.

In order to increase the reliability of the application server load balancer will be used to distribute requests according to servers' workloads. In any case of any shortage on a server, this load balancing will prevent inaccessibility to services.

Chapter 3

3.Algorithm Design

3.1 Request Flow

“Request Flow” is the main functionality of Data4Help. Within this service, registered third parties are allowed to request both individual and anonymous data. The overall process of “Request Flow” is composed of steps that are needed to be performed in order to make a successful data request.

Pseudocode of the process;

```
3rd party indicates its intention of data request and states the
type of request it wants to make

if(individual data is requested)

     system asks the SSN of the individual

     if(SSN is valid)

     if(3rd party is on the pre-confirmed list)
          provide the requested data
               save the request and data sharing info into the
               request history of the individual

          /*not on the pre-confirmed list*/
          else
               ask for the individual permission

               if(individual accepts the request)
                    provide the requested data
                    save the request and data sharing info into
                    the request history of the individual

               /*if individual not accept*/
               else
                    reject the request of 3rd party


     /*SSN not valid*/
     else
     reject the request of 3rd party


/*anonymous data is requested*/
else

     prepare the requested data
     check the anonymity constraint of data

     if(anonymity constraint is satisfied)
          provide the requested data

     /*anonymity constraint is not satisfied*/
     else
          reject the request of 3rd party
```

3.2 Checking Threshold

“Checking Threshold” is the core operation of AutomatedSOS. The whole flow of AutomatedSOS relies on getting data of the individual, comparing this data with a threshold value, and then based on the result of the comparison determining whether there is an emergency case to inform the nearest hospital.

Pseudocode of the process;

```
while (individual data flows)

     monitor individual data

     if(data < threshold)

          detect the location of the individual
          find the nearest hospital
          notify the hospital
          save SOS info of the individual to the application db
          go on with data monitoring

     /*no emergency case, everything is normal*/
     else

          go on with data monitoring
```

3.3 Checking Anonymity Constraint

Checking anonymity constraint is the core operation of Anonym Data Request. After an anonym data request is received. Data manager prepares requested data according to defined filter criteria and after query is executed if number of clients is lower than 1000 request is rejected, and a rejection notification is sent to the third party; else data is sent.

Pseudocode of the process;

```
Anonym data request is received

     Data := prepareData(requestedData,filter);

     if (size (Data)<1000)

          sendRejection(ThirdParty,request)

     else

          sendData(ThirdParty,Data)
```

Chapter 4

4.User Interface Design

The mock-ups for mobile application were presented in the RASD Document in section 3.1.1. User interfaces. In addition to mobile user interfaces in this document, we present mock-ups of the web application for third parties.

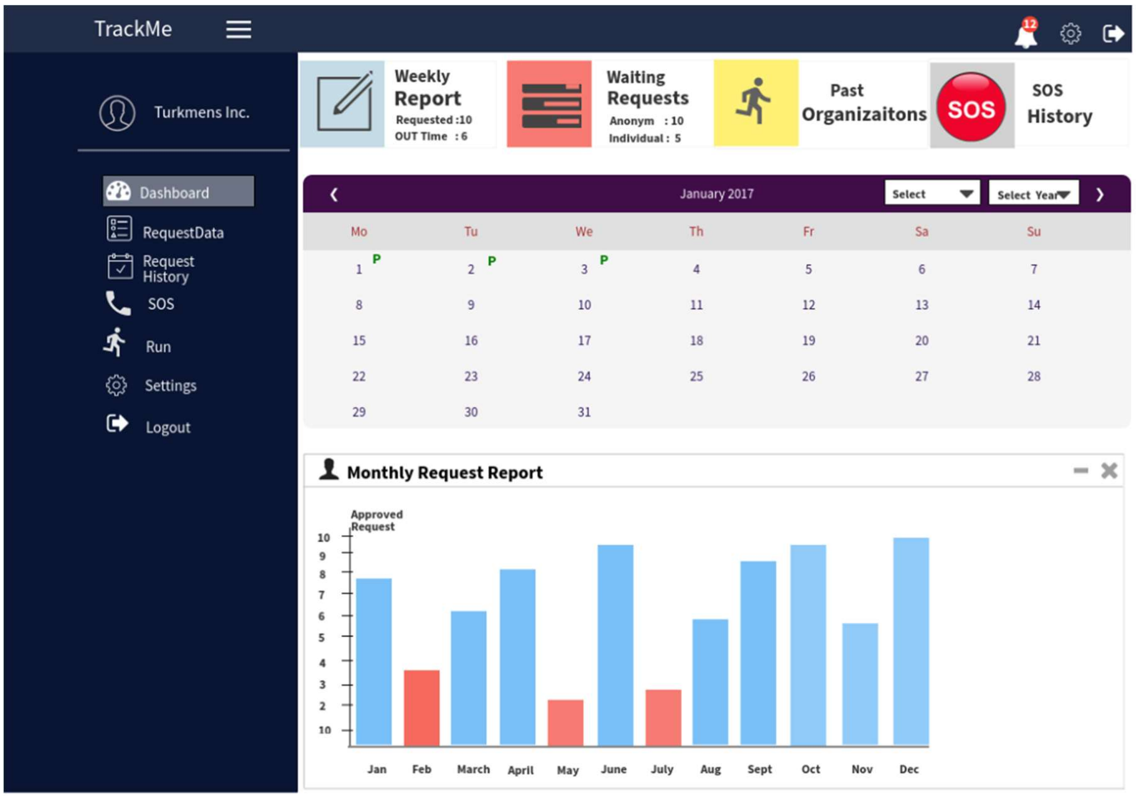
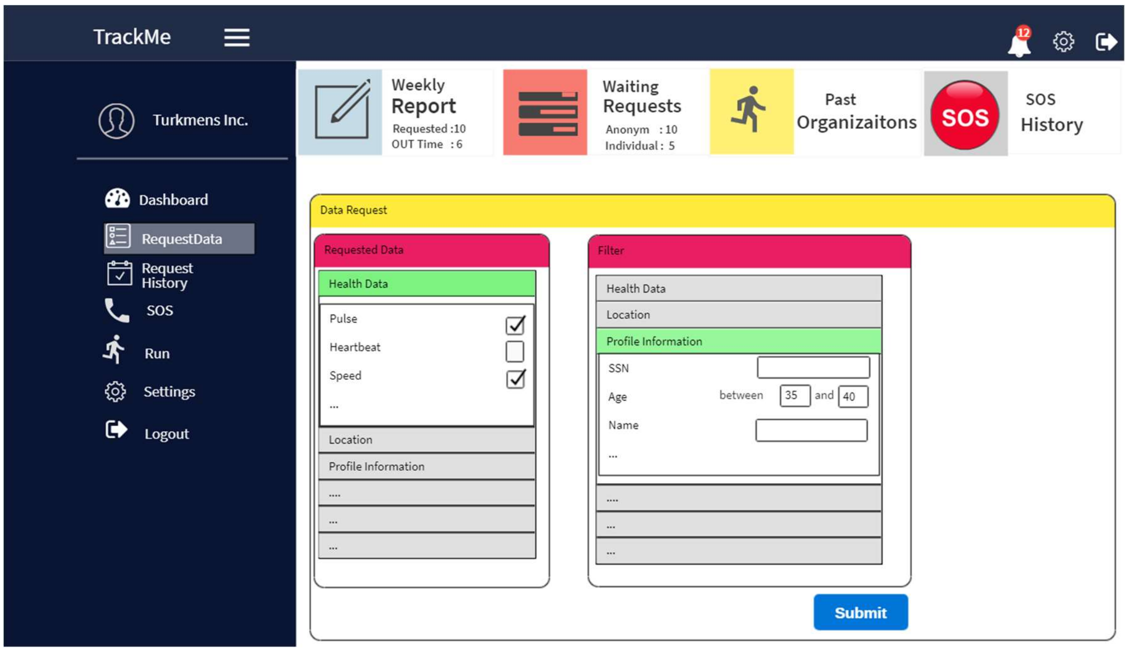


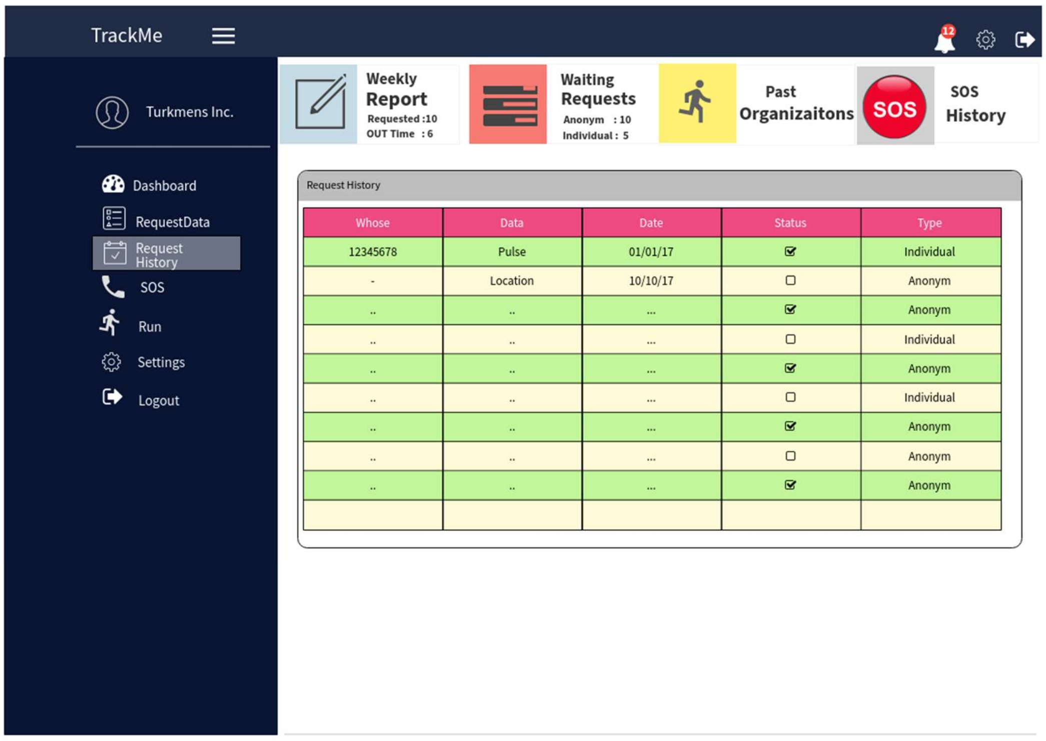
Figure 15-Dashboard



Requested Data	
Pulse	☑
Heartbeat	☐
Speed	☑
...	

Filter: Profile Information		
SSN		
Age	between	35 and 40
Name		
...		

Figure 17-Data Request Page



Whose	Data	Date	Status	Type
12345678	Pulse	01/01/17	☑	Individual
-	Location	10/10/17	☐	Anonym
..	..	...	☑	Anonym
..	..	...	☐	Individual
..	..	...	☑	Anonym
..	..	...	☐	Individual
..	..	...	☑	Anonym
..	..	...	☐	Anonym
..	..	...	☑	Anonym

Figure 16 - Request History Page

Chapter 5

5.Requirements Traceability

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

[G1] Individuals can collect and store their data on Data4Help. The desired data from individuals are profile data, health data, location data and chronic disease.

- DataAcquisitionService
- DataService

[G2] Individuals can share their data with third parties which they allowed.

- RequestService
- ResponseService
- DataService

[G3] Individuals can update their personal information they have used while registering and delete their own account.

- AccountService

[G4] Individuals can see and select which information is shared with which third parties when information of these individuals is shared.

- AccountService

[G5] Individuals can see and track their own data.

- AccountService
- DataService
- DataAcquisitionService

[G6] Third parties can access specific individuals' data.

- RequestService
- ResponseService
- DataService

[G7] Third parties can access anonymized data of a group of individuals.

- RequestService
- ResponseService
- DataService

[G8] Sends a notification to the nearest hospital in 5 seconds when health values of individual fall below a certain threshold.

- DataAcquisitionService
- SosService

[G9] Individuals can keep their SOS history.

- SosService

[G10] Organizers can specify paths for running.

- RunService

[G11] Individuals can register to run as a runner with a unique code of the organization.

- RunService

[G12] Runners can track the path and their location on the map in real-time during the running.

- RunService
- DataAcquisitionService

[G13] Organizers and spectators can see runners' instant locations on a map.

- RunService
- DataAcquisitionService

Chapter 6

6.Implementation, Integration and Test Plan

6.1 Implementation Plan

The overall process of implementation is divided into multiple stages and it will be done component by component.

While deciding the implementation order, we have examined several properties of the components like; their criticality, their size, the number of integrations that they will likely to have and etc. Then, based on these examinations we have given a number to each component indicating its priority on the implementation sequence. Then, from this ordering, we have identified some of the components which can be implemented at the same time and grouped them as one stage of the implementation phase. Finally, after following all these steps we have ended up with the final version of our implementation plan.

The sequence that will be followed during the implementation plan will be like;

1. Model
2. DataService and DataAcquisitionService
3. RequestService and ResponseService
4. SOSService
5. RunService
6. LoginService and AccountService
7. Router

A brief reasoning for this type of order can be given as;

Model

"Model" can be referred to as the center of the application. It is likely that every other component in the system will somehow connect or interact with the model. For this reason, we have preferred to implement the model as a first step. We thought that having an accurate model before implementing and integrating the other components will be better for the overall correctness and the wellness of the system.

DataService and DataAcquisitionService

Since "data" forms a base for the main functionality of the system, it will be true to say that data is one of the most important thing in our application. So, the way data is collected, stored, transferred or in a more abstract manner, the operations that will deal with the data itself and the services that provide these operations are also very crucial. Also, most of the components in the application are in a direct interaction with these two components. As a consequence of this, we thought that it will be better to implement these two components at the early stages of the implementation.

RequestService and ResponseService

Similar to the services that are planned to be implemented before this stage, these two services also form a skeleton for the other components of the application and play a critical role over them. That's why it will be better to implement these components as early as possible. Also, implementing these components earlier will be helpful for us in order to decide the performance of the system in terms of the request-response relation between the client and the server. Because, if we encounter with the decrease in the overall efficiency of the system, we will likely to have some time to find a way to make it better.

SOSService

This component is responsible for an important service called as AutomatedSOS which is offered by TrackMe. When its scope is thought, AutomatedSOS is a crucial service which is least tolerable for a mistake. Because it is in charge of providing an immediate help to the individuals in an emergency situation and in case of a mistake, the results that we will encounter might be more than we can imagine. That's why, after implementing all the previously mentioned "skeleton services", we have decided to start the implementation of SOSService. Because in that case, we will have the chance to devote more time to the implementation and possible improvement of this component.

RunService

This component is responsible for the Track4Run service. Since the job of this component is not as critical as the job of the SOSService, we thought it will be appropriate to implement it right after the implementation of SOSService. The important thing that we need to consider in this stage is to check the accurate connection between this component and the external map service which we have previously called as MapExService.

LoginService and AccountService

Compared with the other components of the application, these two components are among those components which have less priority within the implementation sequence. These two components are simpler to implement and unlike the services which are mentioned above, they do not require too much integration with other components. So, the implementation of these components can be done after finishing the implementation of other components.

<u>**Router**</u>

Router is the least critical component within the system. The only responsibility of this component is to identify the request that is coming from the client and forward it to the correct component that is in charge of this operation. So, its implementation can be shifted to the last position at the implementation plan.

6.2 Integration and Testing

6.2.1 Entry Criteria

Integration of components and testing should start as soon as possible. However, several conditions must be met before these procedures begin. The most important one of these conditions, as already mentioned earlier SmartDeviceExService, HospitalExService, MapExService and the application will run on the DBMS and server.

Then, the first stage is testing the unit for each component. Thus, we can be sure that each component works smoothly. In addition, integration testings after integrating the components, we can be sure that the problems that aren't caused by the component itself.

Next step should be to integrate our components that will work together in an integrated manner and to test the composite part of the integration. After the components have already passed unit tests and have integrated each other, if the integrated parts fail in the integration testing, the problem lies in the integration test.

Before starting the testing level, we may not be able to give a very precise information about what component's and how many percentages should be absolutely finished. But before the integration test of the components that will be integrated with each other, we can definitely say that these components should be completed.

Our external services should be available in 90% to 100% since they will be used directly without having to implement them.

6.2.2 Elements to be Integrated

The TrackMe system consists of a number of components as already shown, and the integration process can be grouped for these components as follows:

- Integration of components with the DBMS
- Integration of components with the (other) external services
- Integration of the components of the application server
- Integration of the client (mobile application) and the application server

Integration of components with the DBMS - User uses AppService. AppService can be referred to the connection between the client and the application server. AppServer is connected with Router inside the application server, which identifies the type of request that is made and decides the component that is needed to be used to perform the requested operation.

- LoginService, DBMS
- AccountService, DBMS
- RequestService, DBMS
- DataService, DBMS
- ResponseService, DBMS
- DataAcquisitionService, DBMS

- SosService, DBMS
- RunService, DBMS

Integration of components with the (other) external services - In this group, we cover the combination of every part of the TrackMe system with an already existing and functional external service (without counting the DBMS). They are the following ones:

- DataAcquisitionService, SmartDeviceExService
- SosService, HospitalExService
- RunService, MapExService

Integration of the components of the application server - Here, the integration between the parts of the application server is conducted. It is composed of:

- RequestService, DataService
- DataService, ResponseService
- DataService, DataAcquisitionService
- ResponseService, RunService
- ResponseService, SosService
- DataAcquisitionService, SosService
- DataAcquisitionService, RunService

Integration of the client and the application server - User can use all components offered by application service.

6.2.3 Order of Component Integration

In this section, we have introduced the visual representations of our component integrations. For this purpose, we have positioned our components next to each other and connected them with an arrow which is pointing from left component to the right one. We have chosen the direction of the arrow on purpose because, by creating an arrow from the first component to the second component, we tried to mean that; “the second (right) component” uses the methods provided by the first (left) component.” Also, by this representation, we have tried to say that the components that are given on the right side use the components on the left side.

The figures that are listed below shows the integration between the components.

Integration of Components with DBMS:

The figures that are given below show the integration between the DBMS and other components of the application. Each component that is given on the right side use the methods that are provided by DBMS. The way each component interact/use DBMS may differ from each other but in the overall case, most of them use it for data storage.

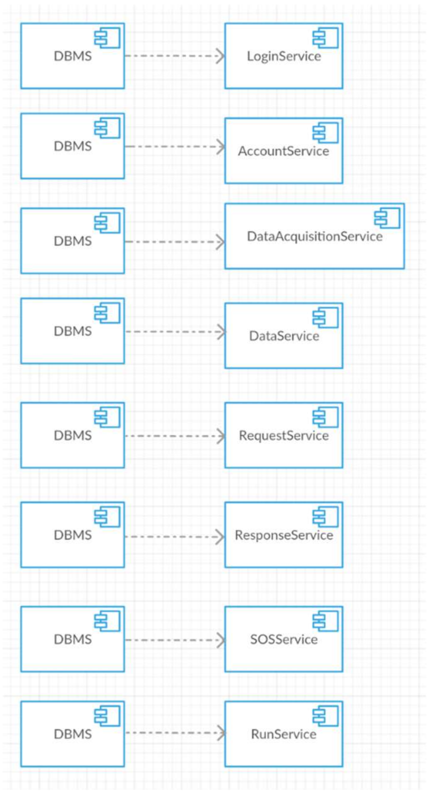


Figure 18 - Integration between DBMS and application components

<u>Integration of Components with External Services:</u>

The figures that are given below show the integration between the application components and the external services that that are being used by them.

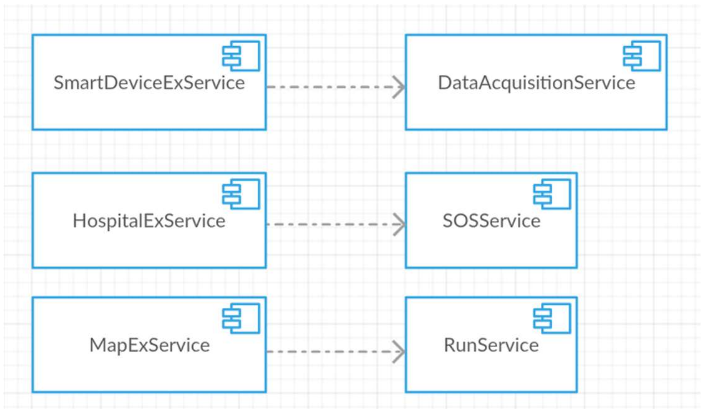


Figure 19 - Integration between the external services and application components

Integration of Components within the Application Server:

The figures that are given below show the internal integration of the components that are introduced under the application server. It is likely that this part will work as the brain of the application. After the integration of these components, it is possible to say that we will end with a fully integrated application server.

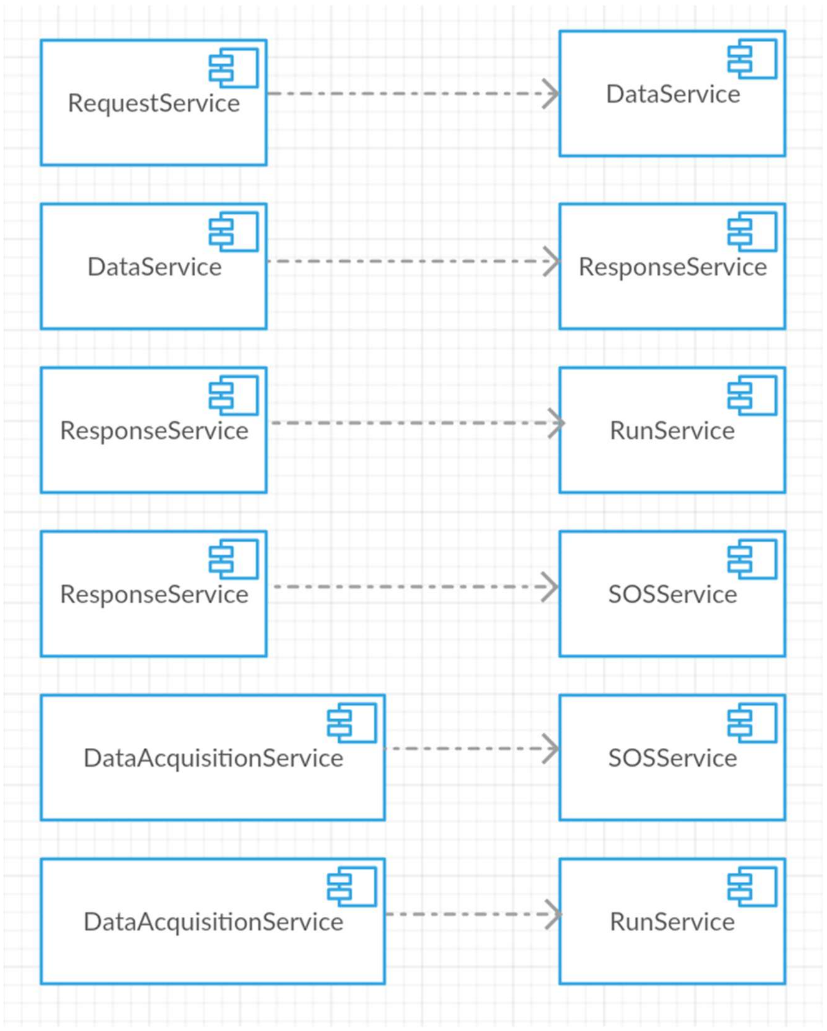


Figure 20 - Integration between the application server components

Integration of Client and Application Server:

The figure that is given below, shows the final component integration. After successfully integrating the internal application server components and application server with DBMS, we are now ready to integrate our client component with the application component.

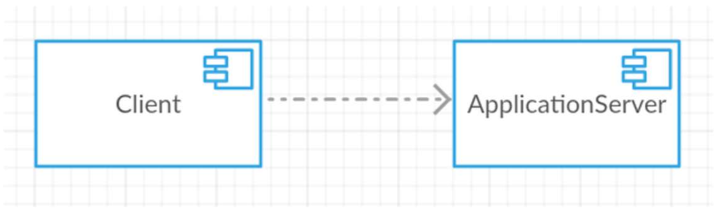


Figure 21 - Integration between the client and the application server

6.2.4 Integration and Testing Strategy

When the overall architecture of the application considered, we have concluded to use the “Bottom-Up Approach” for the integration and testing. By using this approach, we have first tested our low-level components. Then, we have used these components to test the following higher-level components. We have preferred this approach in order to make the localization of the possible faults easier.

We have used the below figure, to briefly explain the hierarchical representation of the components in our application. On the figure, the numbers that are given in the right corner of each component identifies the testing order of the component and the arrow between each component represents the “uses relation” between the two components. (the arrow is pointing from the component to the one which it uses)

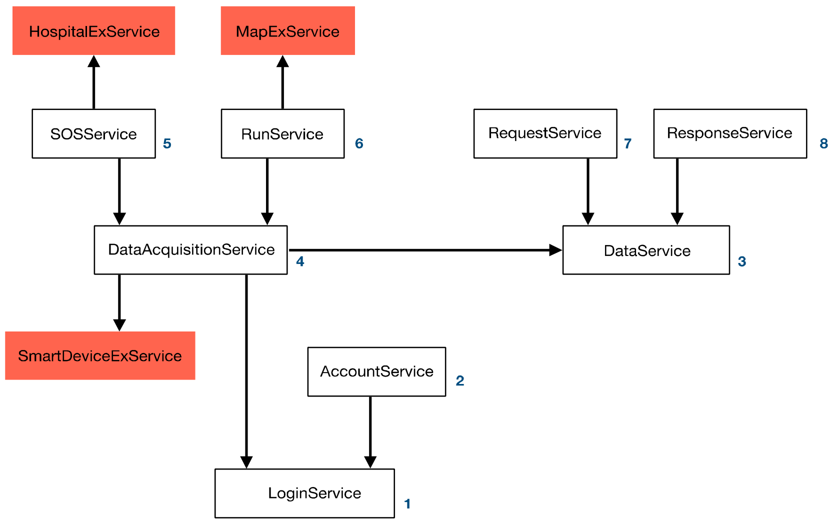


Figure 22 - Overview of the component testing order

Since we are following “Bottom-Up Approach”, we need to use drivers for each module. These drivers represent a higher-level component for the module which has been currently going through the testing process. Then, for all the following steps, we will replace those drivers with the module itself and use its drivers to proceed with the testing.

The figure which is included below gives a brief representation of the testing procedure by simulating the integration testing of the components; “LoginService” and “DataAcquisitionService”.

Integration Testing of LoginService:

During the integration testing of “LoginService”, two components; “DataAcquisitionService” and “AccountService” which uses it are represented as drivers.

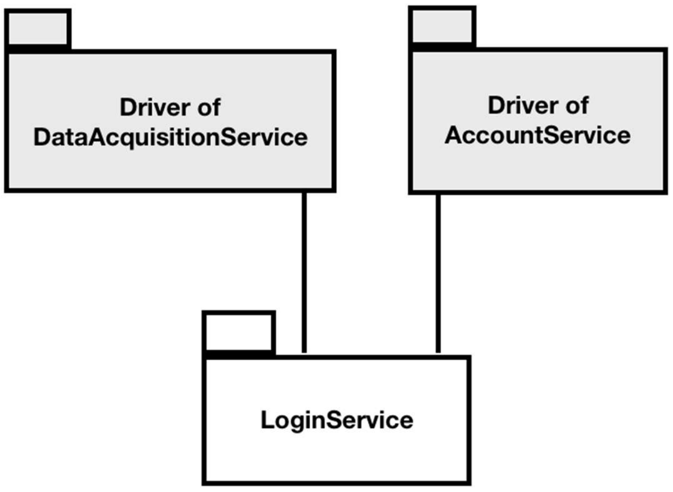


Figure 23 - Integration testing of LoginService

<u>Integration Testing of DataAcquisitionService:</u>

During the integration testing of “DataAcquisitionService”, two components; “SOSService” and “RunService” which use it are represented as drivers. On the other, the other two components; “LoginService” and “DataService” which are used by “DataAcquisitionService” are represented as normal components since they were already tested.

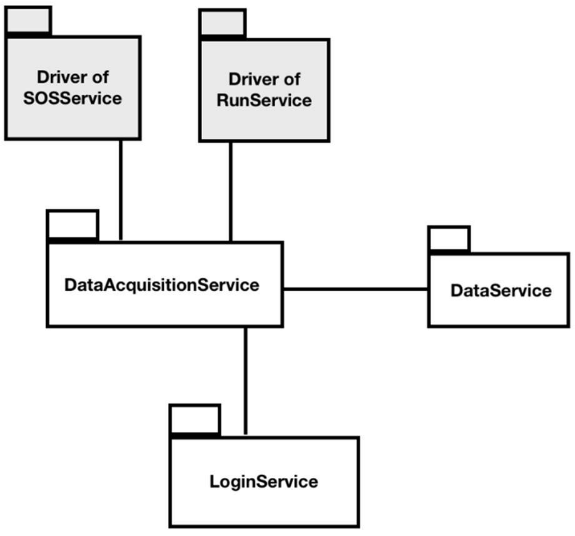


Figure 24 - Integration testing of DataAcquisitionService

Chapter 7

7. Effort Spent

Meeting Dates:

22/11/18: 2h -> DD to be analyzed, the component interface defined.

25/11/18: 2h -> Component and deployment interfaces are discussed.

29/11/18: 2h -> We made brainstorming on design patterns, implementation order, external services and algorithm design.

04/12/18: 2h -> We discussed integration and testing.

06/12/18: 2h -> We made a general overview of the project and merged the documents.

In addition to meeting hours, each group member worked additional following hours.

Buse GUNAY:	~ 30h
Ezgi TASTI:	~ 30h
Talip TURKMEN:	~ 29h

Chapter 8

8. References

Github: Used for sharing project document and version control

Creately: Used for drawing diagrams

WireFramePro: Used for drawing user interfaces

Microsoft Word 2016: Used for creating the document

Pages: Used for creating figures given under the integration testing.

Reference links:

https://cordova.apache.org/docs/en/latest/guide/overview/index.html

https://en.wikipedia.org/wiki/Factory_method_pattern

https://en.wikipedia.org/wiki/Observer_pattern

https://en.wikipedia.org/wiki/Builder_pattern

https://www.tutorialspoint.com/design_pattern/facade_pattern.htm

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

https://en.wikipedia.org/wiki/Integration_testing

https://www.quora.com/What-is-bottom-up-integration-testing

https://en.wikipedia.org/wiki/Multitier_architecture

https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/