POLITECNICO
MILANO 1863

Software Engineering 2: Data4Help

Design Document

Mattioli Daniele, Romano Leonardo, Tonelli Alessio

Politecnico di Milano

December 10, 2018

Version 1.0.0

Contents

1. Introduction

1.1.Purpose

The purpose of this document is to provide more technical details than the RASD about Data4Help application system. In particular, the aim of this document is to present a description about high level architecture, components and their interfaces, deployment, run-time processes, design patterns, implementation, integration and test plan.

This document is written for project managers, developers, testers and Quality Assurance team. It can be used for a structural overview to help maintenance and further development.

1.2.Scope

Data4Help is a system provided by TrackMe whose purposes are described in section 1.1 of the RASD. The architecture has to be easily extensible and maintainable in order to provide new functionalities.

Every component must encapsulate a single functionality. The dependency between components has to be unidirectional and coupling must be avoided in order to increase the reusability of modules.

Design patterns will be used in order to simplify the system comprehension and avoid misunderstanding during the implementation phase.

1.3.Definitions, Acronyms, Abbreviations

1.3.1. Definitions

See section 1.3.1 of the RASD

1.3.2. Acronyms

- AES: Advanced Encryption Standard
- API: Application Programming Interface
- ACID: Atomicity, Consistency, Isolation, Durability
- DB: Database
- DBMS: Database Management System
- DD: Design Document
- DMZ: Demilitarized Zone
- GUI: Graphical User Interface
- MVC: Model View Controller
- RASD: Requirements Analysis and Specifications Document
- RDBMS: Relation Database Management System

1.3.3. Abbreviations

- [Gn]: n-th goal
- [Dn]: n-th domain assumption
- [Rn]: n-th functional requirement

1.4. Revision History

The following list contains the changes between version 1.0.0 and 1.0.1 of the RASD

- Functional requirement R21 has been changed.
- Domain assumption D12 has been added.
- Class diagram in figure 1 has been modified: class Data and some attributes were added.
- External Service HERE has been removed.
- G13 has been added.
- Functional requirement 22 has been added.

1.5. Document Structure

The document is divided in seven parts. The first one provides general information about the DD and the system to be developed. The second part shows the main components of the systems, together with subcomponents and their relationships. This section comprehends deployment and run-time views. The third part provides an overview on how the user interface will look like, referring to the mockups already presented in section 3.1.1 of the RASD. The fourth part shows how the requirements that have been defined in the RASD map to the design elements that are defined in this document. The fifth part identifies the order to implement the subcomponents of the system and the order to integrate such subcomponents and test the integration. The sixth part includes information about the number of hours each group member has worked for this document. Finally, the seventh part includes the reference documents.

2. Architectural Design

2.1.Overview

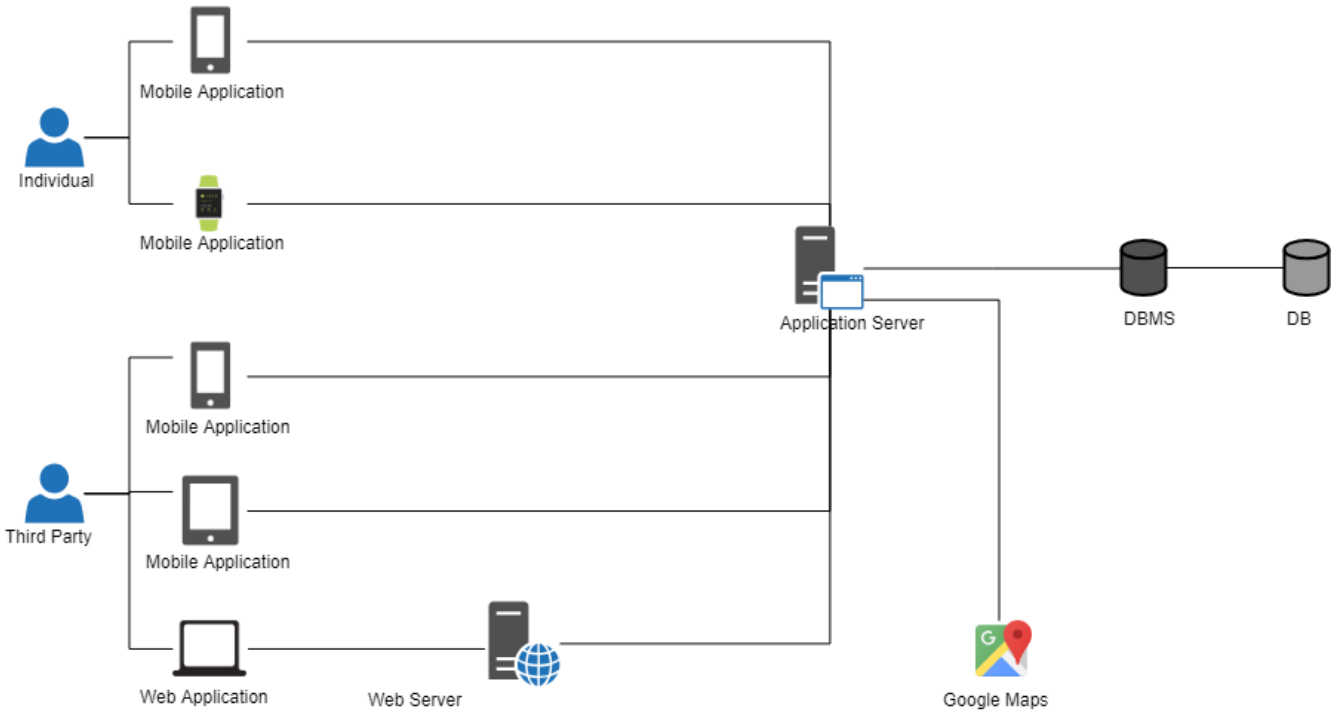


Figure 1

Data4Help is a client server three layered architecture. The logical levels are presentation, application and data.

The presentation level is composed by the mobile application, the client browser (web application) and by the web server that handles clients' http requests. It consists of the user interface (GUI, in this case) which displays content and information useful to an end user.

The application layer is composed by the application server and it contains the functional business logic which drives an application's core capabilities. The application server includes also the logic needed to interface with the external service. The application layer moves and processes data between the presentation and the data layers. The mobile application directly communicates with the application layer, while the web application communicates with this layer through the web server.

The data layer includes all structures and entities responsible for data storage and management. It is composed by the DBMS and the database. It is worth pointing out that no application logic is found at this level, apart from the DBMS one that must guarantee the correct functioning of the data structures while assuring the ACID properties of transactional databases.

External services, in this case Google Maps, communicate with the application server, in order to provide maps for Track4Run service.

2.2.Component View

In this section, the individual components will be discussed in terms of the needed sub-parts and their role. Moreover, this section will show which of said sub-elements is in charge of interfacing with other components of the system.

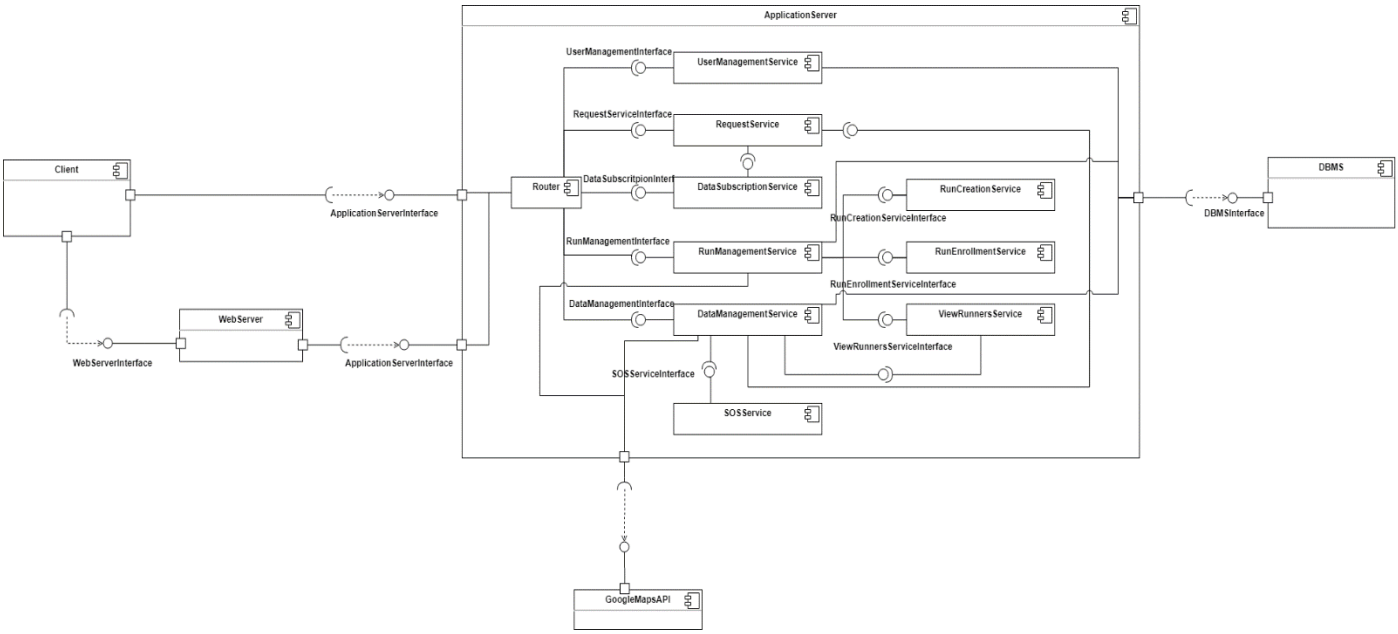


Figure 2: component diagram

Database and DBMS:

The data layer must include a DBMS component, in order to manage the insertion, modification and deletion of transaction on data inside the database. The DBMS must guarantee the correct functioning of concurrent transactions and the ACID properties. Moreover, the DBMS must be relational (RDBMS), since the application doesn't require more complex structures than the relational one to store its data.

The data layer must be accessible only through the application server via a dedicated interface.

Sensible data, such as passwords and personal information, must be encrypted properly before being stored.

Application Server:

The server must handle the business logic, the connections with the data layer, the interaction with the external service and the multiple ways of accessing the application from different clients. In Figure 2 is possible to observe that the components of the application server are the following:

- UserManagementService: handles user sign up and log in.
- RequestService: handles both individual and group requests
- DataSubscriptionService: handles functionalities concerning third party subscriptions to individual's specific data.
- DataManagementService: contains the logic to manipulate and manage individual's data.
- SOSService: manages the sending of the SOS message when the health status parameters go below their thresholds.
- RunCreationService: handles the creation of a run.
- RunEnrollmentService: handles the enrollment to a run.
- ViewRunnersService: handles functionalities to allow individual to see the position of all runners during a run.
- RunManagementService: handles functionalities concerning runs, such as show the list of all runs.
- Router: receives requests (in the general sense of the term, not only individual and group requests) from the client and dispatch it to the relevant service component.

Web Server:

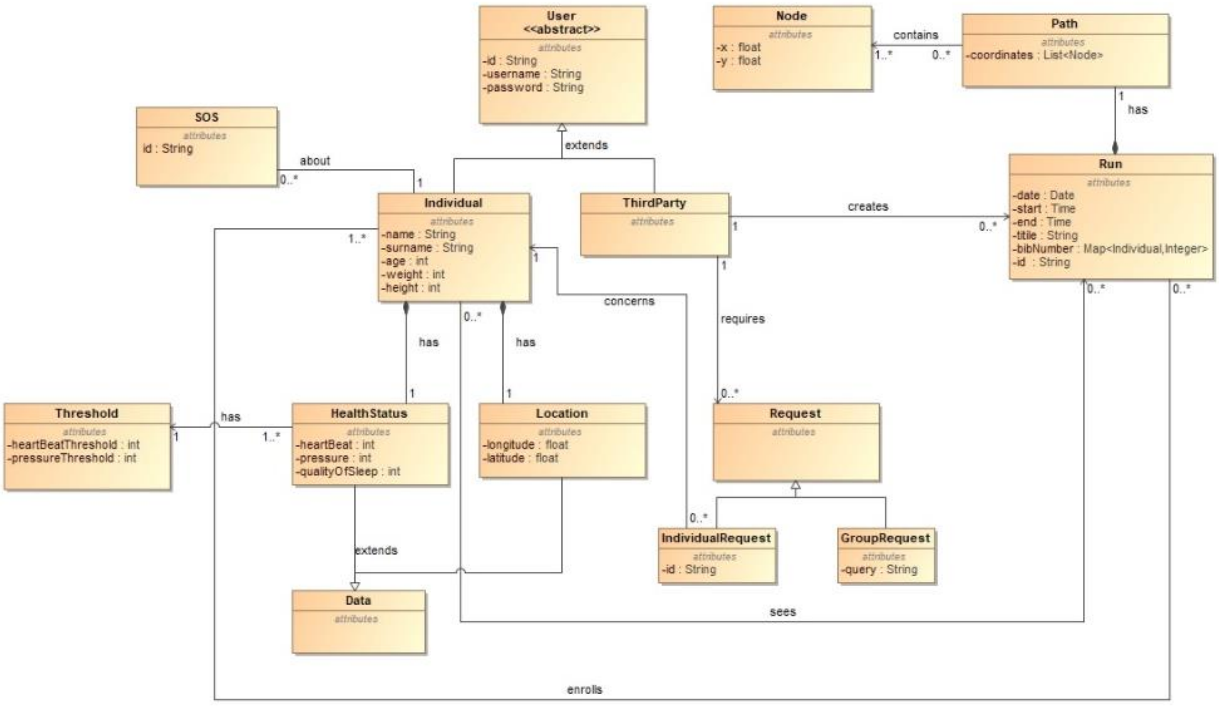
The web server connects clients using Data4Help on the web app with the application layer. The main functions to be implemented in this layer will essentially consists in interfaces, since there won't be any logic implemented within the web server besides the presentation of pages.

Client:

The mobile application GUI must be designed following the guidelines provided by the Android and iOS producers.

The application must keep track of locations, heartbeat and pressure and quality of sleep measurements providing it to the application server.

Since the Model is made up of the classes represented in the Figure 1 of the RASD and it's not a component per se, the following diagrams aims to represent it and its relationship with the components of the application server.


User
<<abstract>>
attributes
-id : String
-username : String
-password : String
Node
attributes
-x : float
-y : float
contains
1..*
0..*
Path
attributes
-coordinates : List<Node>
has
SOS
attributes
id : String
about
0..*
1
extends
Run
attributes
-date : Date
-start : Time
-end : Time
-titile : String
-bibNumber : Map<Individual,Integer>
-id : String
Individual
attributes
-name : String
-surname : String
-age : int
-weight : int
-height : int
ThirdParty
attributes
creates
1
0..*
1..*
1
concerns
requires
0..*
has
has
1
1
0..*
Threshold
attributes
-heartBeatThreshold : int
-pressureThreshold : int
has
1
1..*
HealthStatus
attributes
-heartBeat : int
-pressure : int
-qualityOfSleep : int
Location
attributes
-longitude : float
-latitude : float
Request
attributes
0..*
IndividualRequest
attributes
-id : String
GroupRequest
attributes
-query : String
extends
Data
attributes
sees
enrolls

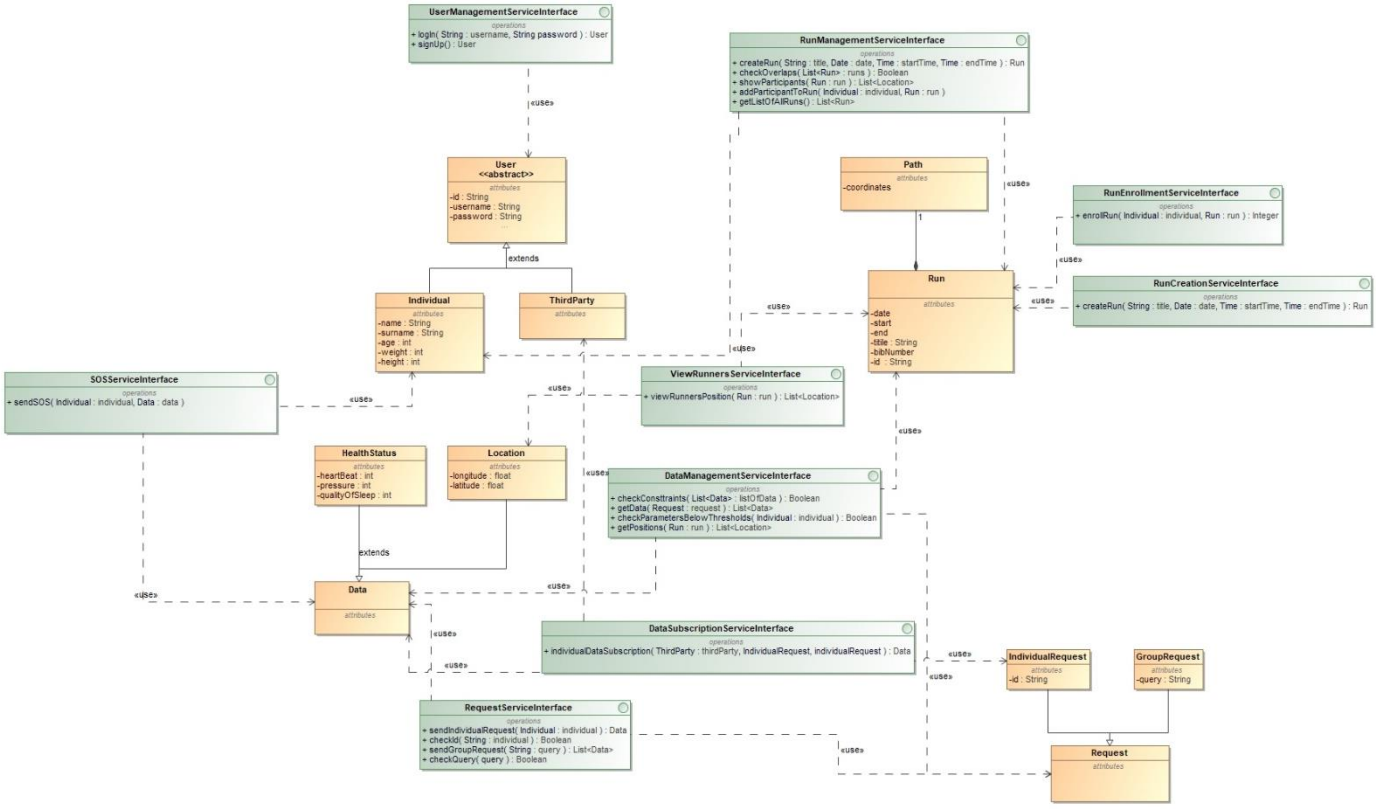

UserManagementServiceInterface
operations
+ login(String : username, String password) : User
+ signUp() : User
RunManagementServiceInterface
operations
+ createRun(String : title, Date : date, Time : startTime, Time : endTime) : Run
+ checkOverlaps(List<Run> : runs) : Boolean
+ showParticipants(Run : run) : List<Location>
+ addParticipantToRun(Individual : individual, Run : run)
+ getListOfAllRuns() : List<Run>
«use»
User
<<abstract>>
attributes
-id : String
-username : String
-password : String
Path
attributes
-coordinates
RunEnrollmentServiceInterface
operations
+ enrollRun(Individual : individual, Run : run) : Integer
extends
Individual
attributes
-name : String
-surname : String
-age : int
-weight : int
-height : int
ThirdParty
attributes
Run
attributes
-date
-start
-end
-titile : String
-bibNumber
-id : String
RunCreationServiceInterface
operations
+ createRun(String : title, Date : date, Time : startTime, Time : endTime) : Run
SOSServiceInterface
operations
+ sendSOS(Individual : individual, Data : data)
ViewRunnersServiceInterface
operations
+ viewRunnersPosition(Run : run) : List<Location>
HealthStatus
attributes
-heartBeat : int
-pressure : int
-qualityOfSleep : int
Location
attributes
-longitude : float
-latitude : float
DataManagementServiceInterface
operations
+ checkConstraints(List<Data> : listOfData) : Boolean
+ getData(Request : request) : List<Data>
+ checkParametersBelowThresholds(Individual : individual) : Boolean
+ getPositions(Run : run) : List<Location>
extends
Data
attributes
DataSubscriptionServiceInterface
operations
+ individualDataSubscription(ThirdParty : thirdParty, IndividualRequest, individualRequest) : Data
IndividualRequest
attributes
-id : String
GroupRequest
attributes
-query : String
RequestServiceInterface
operations
+ sendIndividualRequest(Individual : individual) : Data
+ checkId(String : individual) : Boolean
+ sendGroupRequest(String : query) : List<Data>
+ checkQuery(query) : Boolean
Request
attributes

Figure 3

2.3.Deployment View

The deployment diagram in Figure 4 describes the physical deployment of information generated by the software program, called artifact, on hardware components.

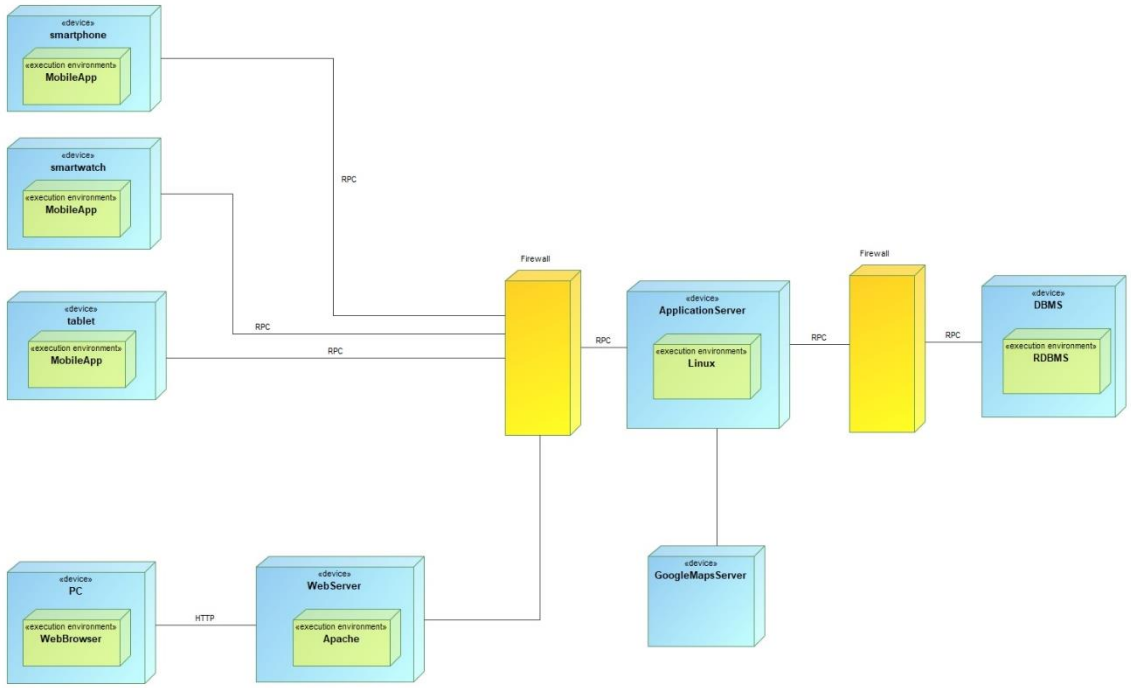


Figure 4: deployment diagram

It is possible to observe the presence of firewalls in order to guarantee protection of data and information exchange

2.4.Runtime View

This section describes the dynamic behavior of the system in the most relevant cases.

The following sequence diagrams highlight the runtime interactions between clients, servers and the database.

Individual Request

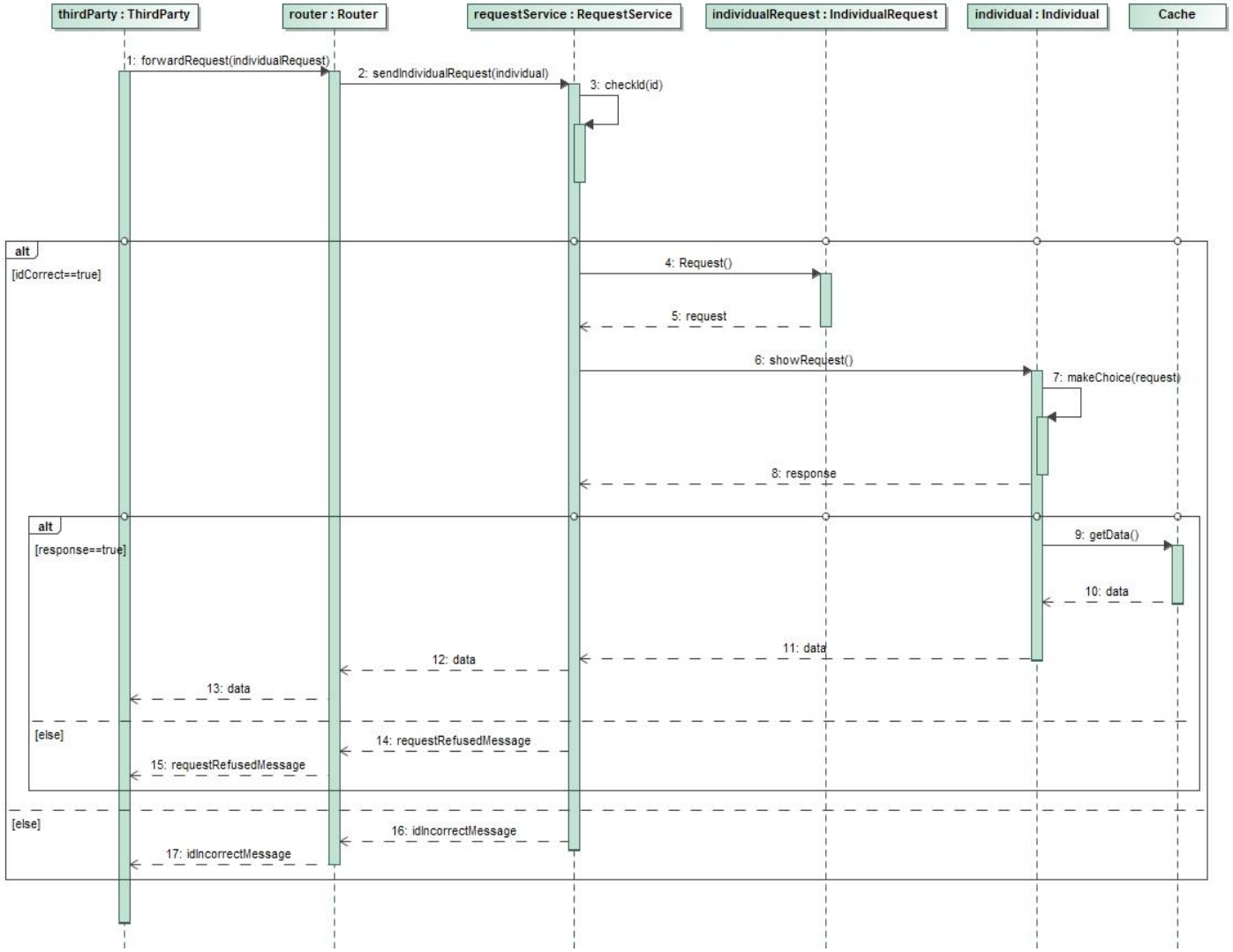


Figure 5

In Figure 5 is represented the sequence diagram concerning an individual request. It is possible to observe that, if the individual accepts the request, his/her data are retrieved directly from the cache.

Group Request

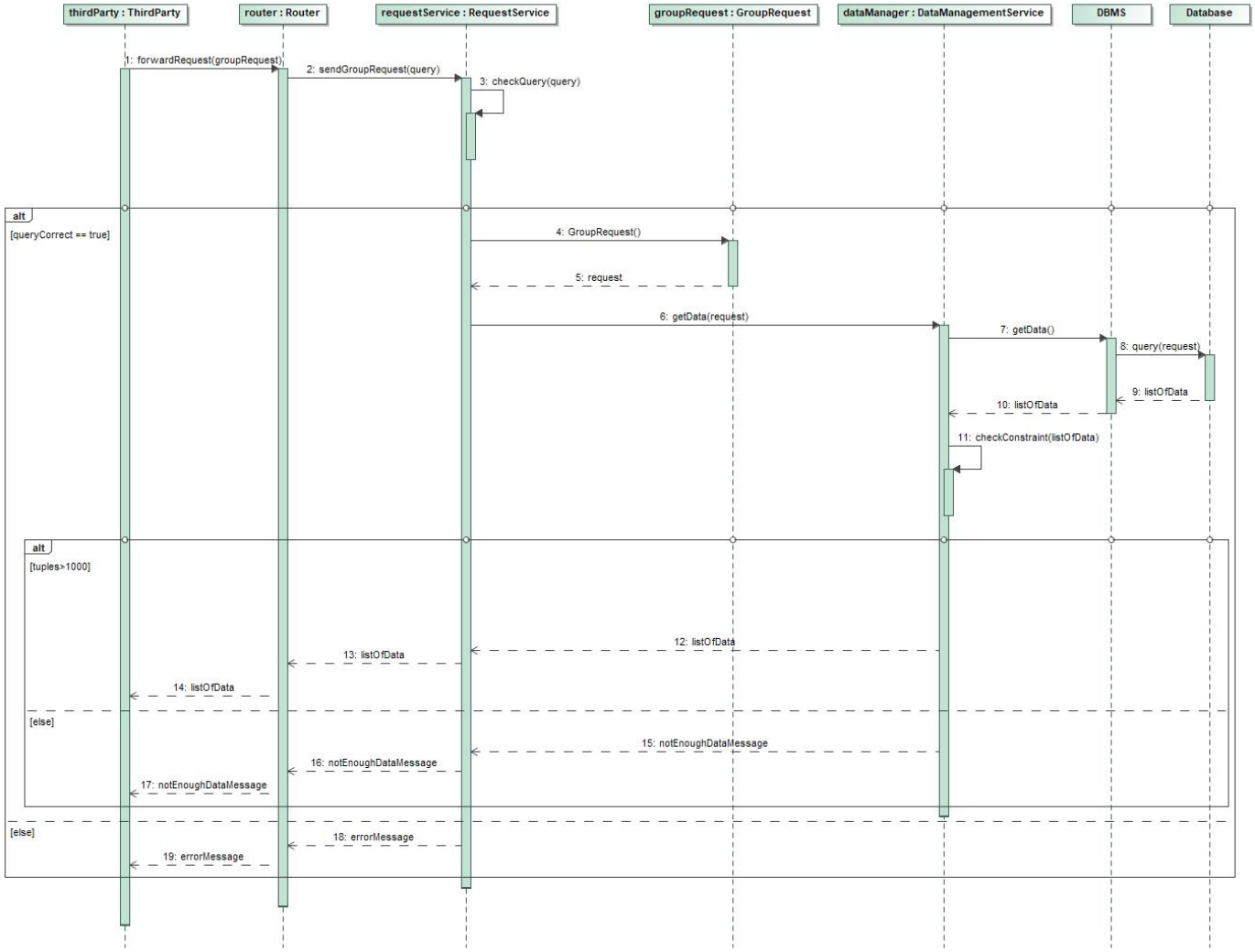


Figure 6

In Figure 6 is represented the sequence diagram concerning a group request.

Monitor health status

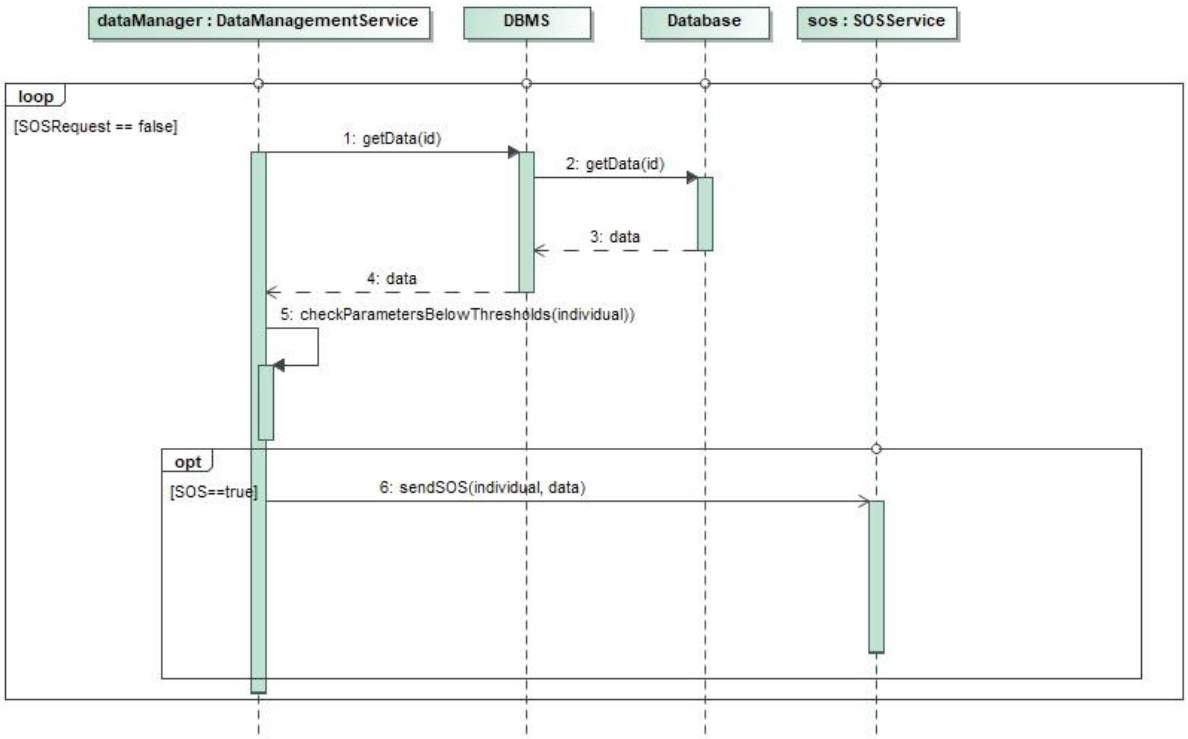


Figure 7

In Figure 7 is represented the sequence diagram concerning monitoring of individual's health status. Individual's data, retrieved from the database, are sent to the DataManagementService component, that checks if the parameters are below their thresholds. If it is so, SOSService component sends a notification to the external partner.

Create Run

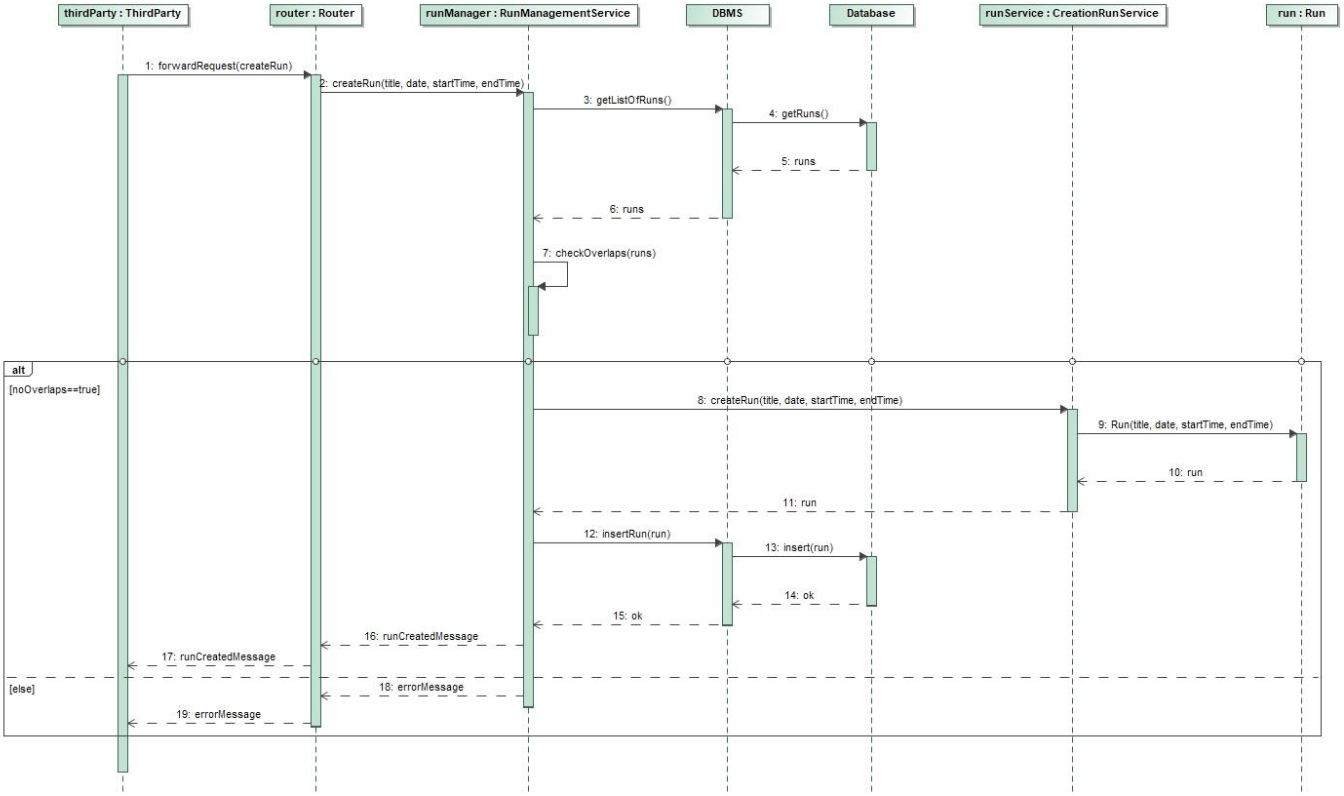


Figure 8

In Figure 8 is represented the sequence diagram concerning the creation of a run by a third party. It is possible to observe that RunManagementService checks if there exists already a run, with the same date and starting time, whose path overlaps with that of the run the third party is trying to create.

Monitor Runners

In Figure 9 is represented the sequence diagram concerning monitoring of runners during a run. The individual selects the run he/she is interested in; RunManagementService retrieves the run and the list of all its participants on the database; DataManegementService retrieves the position of each runner and returns it to the RunManagementService. It is possible to observe that the RunManagementService interacts with the external service to provide the individual the map with all the positions.

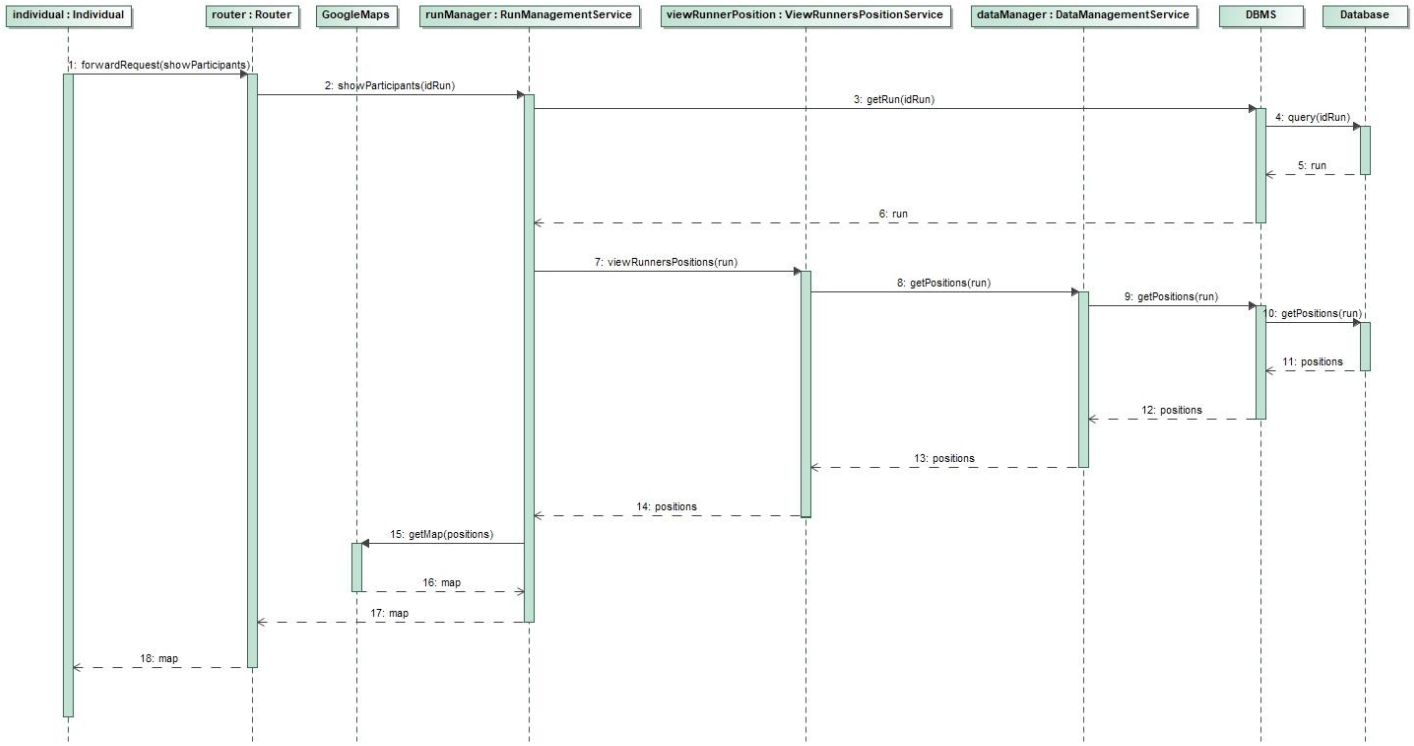


Figure 9

2.5. Component Interfaces

In the following diagram are shown the dependencies between the components of the application server. This information was already present in Figure 2 but here it is shown more clearly.

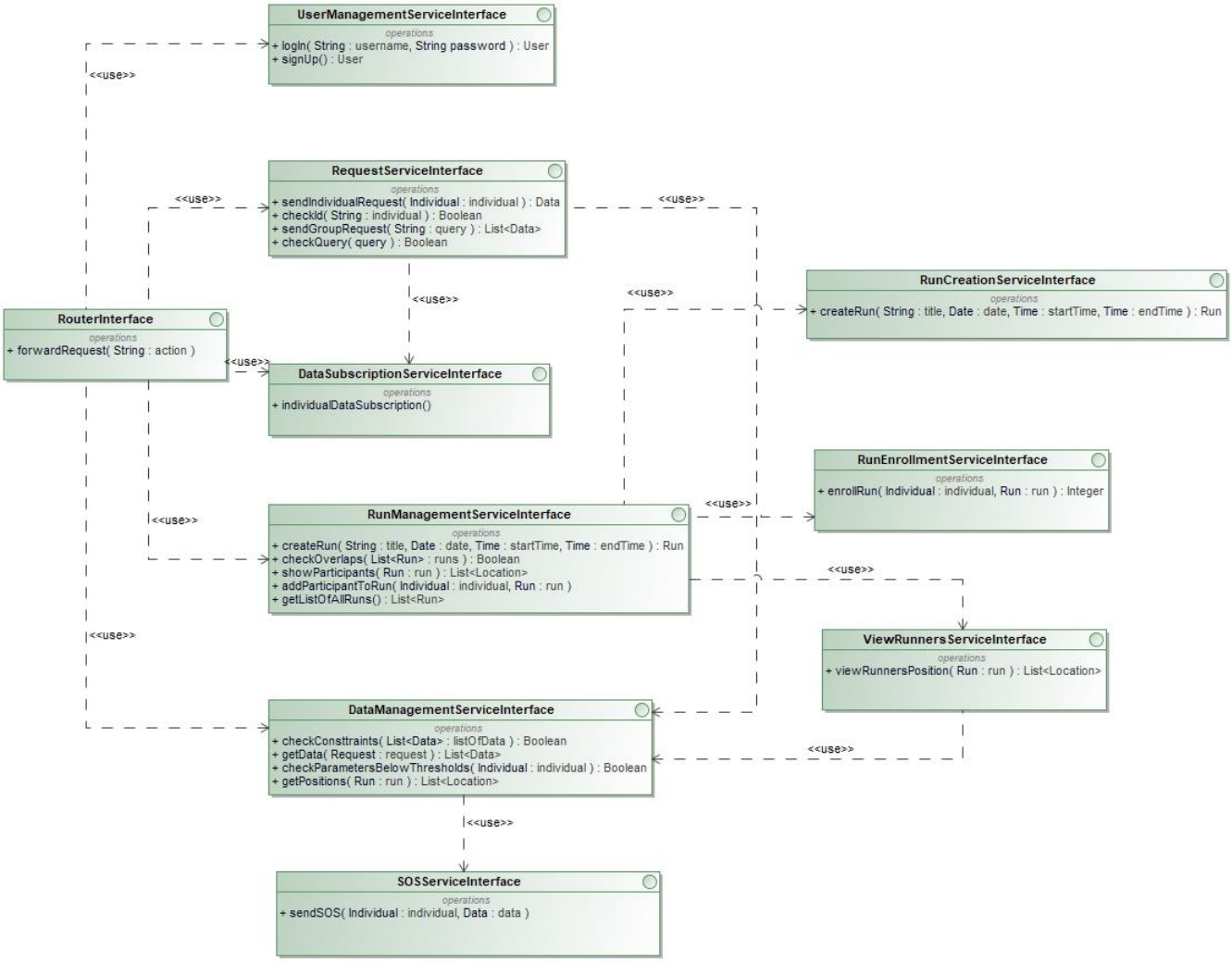


Figure 10

2.6. Selected Architectural Styles and Patterns

Client - Server

This type of architectural style has been chosen due to the fact it is very useful in different occasions:

- The mobile application is the client with respect to the application server that receives and elaborates the requests.
- The users' browser is the client with respect to the web server, that is in charge of providing the requested web page.
- The web server is the client with respect to the application server, that indeed processes users' requests.
- The application server is the client with respect to the database, that is responsible of fetching query results.

Three Tier Architecture

Three tier architecture allows to physically separate presentation, application processing and data management operations. This architecture is intended to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology.

Thin client

Thin client approach has been followed while designing the interaction among users' machine and the system: all the main logic is implemented by the application server, that has enough computing power and can manage concurrency issues in an efficient way, leaving the mobile application and the web application in charge of presentation only.

This choice allows users to use the service via devices with limited computing power obtaining however acceptable performances. It also makes software updates easier.

Model – View – Controller

The MVC pattern has been chosen in order to separate the application into three communicating and interconnected parts fulfilling the design principle of the separation of concerns. The model directly manages the data, logic and rules of the application, the view comprehends the components of the user interface, the controller accepts input and converts it to commands for the model or view. The model and the controller are server side; the view is client side.

Façade Pattern

This pattern is used to hide the complexities of the application server and to provide a simpler interface to the client. Indeed, the client interacts only with the router, that forwards the request to the appropriate component on the application server.

2.7.Other design decisions

Maps

Google Maps is used to support the definition of the path of a run and to show individuals the position of all runners during a map. This choice is motivated by the fact that manually developing maps for each city is not a viable solution due to the tremendous effort of coding and data collection required.

Moreover, this service is well-established, tested and reliable software components already used by millions of people around the world.

Availability and redundancy

In the RASD it is stated that high degree of availability must be guarantee. Figure 11 shows a more detailed architecture than that of Figure 1: it is possible to observe the presence of a multiple components with load balancing instead of a single component in order to increase availability through redundant servers.

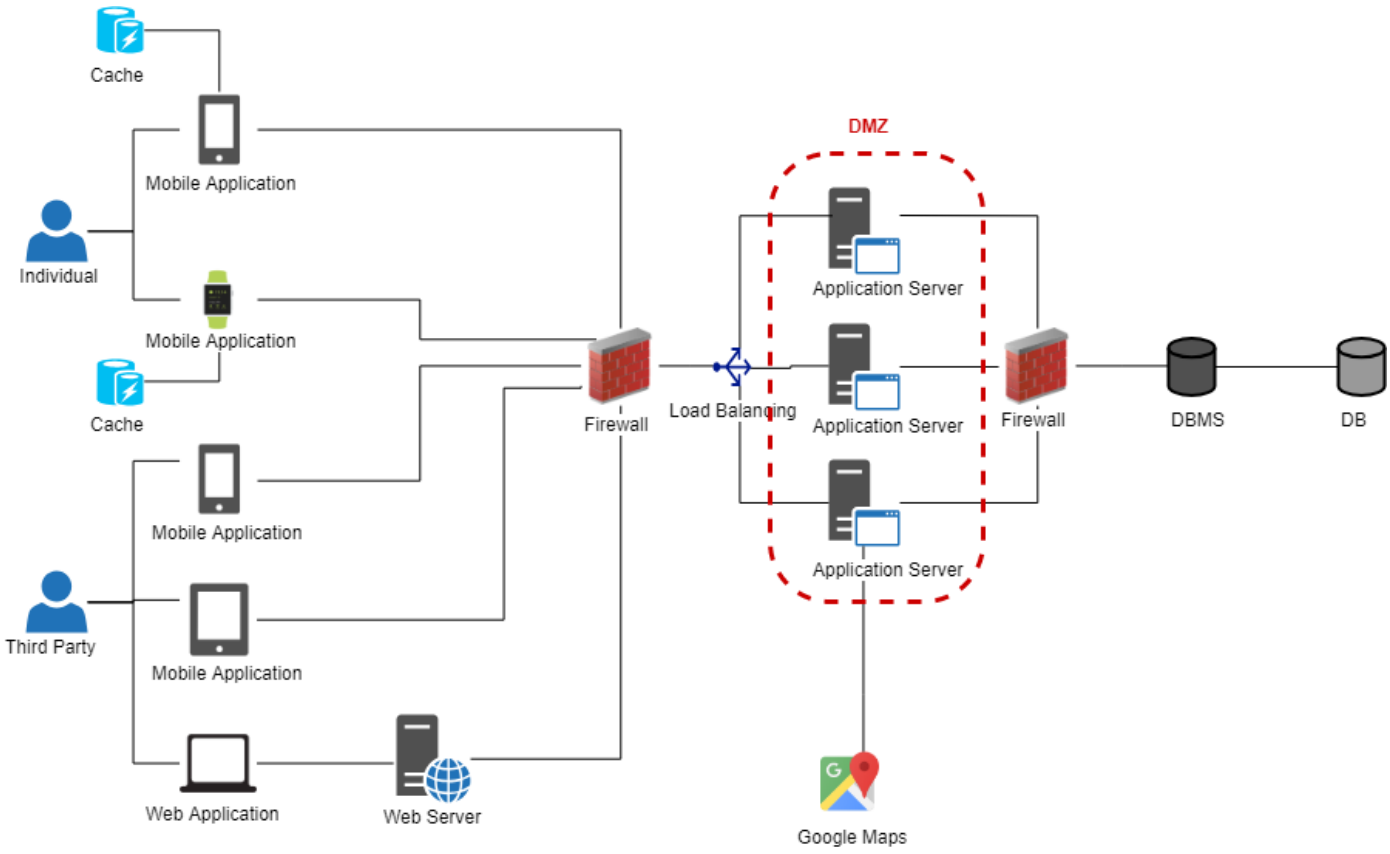


Figure 11

Note that in the diagram Google Maps is linked only to one application server, in order to be more readable and not have too many links.

Encryption and security

Users' passwords are not stored in plain-text, but they are hashed and salted with cryptographic hash functions. This provides a last line of defense in case of data theft.

AES can be used to encrypt individuals' location and health parameters.

In Figure 11 is possible to observe that by using a couple of firewalls, a DMZ containing the application server is created, in order to add an additional level of security.

3. User Interface Design

The mock-ups for this application were presented in the RASD Document in section 3.1.1. User Interfaces. At this point, there are no new functionalities that can be presented with new mock-ups for the application.

4. Requirements Traceability

In the table below is specified how the design of the application meets all the goals and requirements already specified in section 3.2 of the RASD. It is worth noticing that all the application components are included in this mapping, also those contributing indirectly in the satisfaction of the requirements.

Requirement	Components
[R1] - Individuals' health status is provided to Data4Help by their smartwatches	DataManagementService Router
[R2] - Individuals' location is provided to Data4Help by their smartwatches	DataManagementService Router
[R3] - Third parties can identify a specific individual entering on Data4Help his/her unique ID	RequestService Router
[R4] - Third parties can send a specific individual the request for his/her data through Data4Help	RequestService Router
[R5] - Third parties can send a request for anonymized data of groups of individuals through Data4Help	RequestService DataManagementService Router
[R6] - Third parties' request is sent to the interested user by Data4Help	RequestService Router
[R7] - Individuals can view on Data4Help the request they've received	RequestService

	Router
[R8] - Individuals can accept or refuse the request using Data4Help	RequestService Router
[R9] - Third parties can view on their devices data provided by Data4Help	RequestService DataManagementService DataSubscriptionService Router
[R10] - Data4Help has access to the requested data	DataManagementService DataSubscriptionService Router
[R11] - Data4Help must take into account individuals' response	RequestService Router
[R12] - Third parties can view requested data on their devices	RequestService DataManagementService Router
[R13] - The application sends the users' data to the external partner	SOSService
[R14] - The users are shown the map	RunManagementService Router GoogleMaps
[R15] - Third parties can create a run by entering the title, the date, the starting and ending time and the path	RunManagementService RunCreationService Router
[R16] - Individuals are shown a list of athletic runs	RunManagementService Router
[R17] - Individuals can select a specific run from the list	RunManagementService Router
[R18] - Track4Run shows the location only of the individuals who actually enrolled the run	RunManagementService ViewRunnersService Router GoogleMaps
[R19] - Users can create a Data4Help account	UserManagementService
[R20] - Users can log in to the application by providing the combination of a username and a password that match an account	UserManagementService Router

[R21] - Users can sign up to the application by providing the required information	UserManagementService Router
[R22] - Individuals can enroll to a run through Data4Help	RunManagementService RunEnrollmentService Router

5. Implementation, Integration and Test Plan

5.1. Implementation Plan

The implementation of Data4Help will be done component by component and will take into account the dependence between components and their complexity.

The implementation will start from the simplest components, that should not take a long time, and will proceed with the most complex ones. This will allow to devote more time to their implementation and to better focus on them.

The components of Data4Help will be developed in the following order:

1. Model
2. UserManagementService, SOSService, DataSubscriptionService
3. DataManagementService
4. ViewRunnersService, RunCreationService, RunEnrollmentService
5. RequestService, RunManagementService
6. Client and Router

The model will be implemented first, since some components of the application server use some element of it. Once the model has been created, there will be the setup of the DBMS.

The second components to be implemented will be UserManagementService, SOSService, DataSubscriptionService. These components are not too complex and their implementation should not require too much time.

The third component to be implemented will be DataManagementService, that contains the logic to manipulate and manage individuals' data.

The fourth components to be implemented will be those concerning specific features of the runs. These components will be used by the RunManagementService, thus they're implemented earlier.

The fifth components to be implemented will be RequestService and RunManagementService. Since they handle both requests and runs, their development is expected to require more effort and time.

The last components to be implemented will be the Client and the Router because they are mainly used, respectively, to provide users with the GUI and with a way to communicate with the components on the application server and to redirect users' request to the correct component.

5.2.Entry Criteria

The integration of components and their testing should start as soon as possible, but first the two following conditions must be met:

- The external service, Google Maps, and its APIs must be available
- The modules integrated with each other should have at least the operations concerning one another created, if not completed completely. Moreover, these components should pass their unit tests, in order to be sure that possible failures depend only on the integration and not on the components their selves.

5.3.Integration Testing Strategy

The chosen strategy for integration is the bottom up one. The choice of the bottom-up testing strategy is natural since the integration testing can start from the smallest and lowest-level components, that are already tested at a unit level and do not depend on other components or not-already-developed components. In this way no stubs will be needed, but temporary programs for higher-level modules (drivers) will be necessary to simulate said modules and invoke the unit under test.

The integration will proceed in the following order:

1. **Integration of components with the DBMS:**

   First, the integration between the application server components that uses the database and the DBMS will be carried out. The specific integrations to be done are the following:

   - DataManagementService – DBMS
   - RunManagementService – DBMS
   - UserManagementService – DBMS

2. **Integration of components with the external service:**

   - GoogleMapsAPI – ViewRunnersService
   - GoogleMapsAPI – RunCreationService

3. **Integration of the components of the application server:**

   Exploiting the "use" hierarchy represented in Figure 12, the application server components will be integrated in the following order:

   - SOSService-DataManagement
   - RunCreation-RunManagement
   - RunEnrollment- RunManagement

- DataManagement-ViewRunners
- ViewRunners- RunManagement
- DataManagement-RequestService
- Router - DataManagementService
- Router - UserManagementService
- Router - RequestService
- Router - DataSubscriptionService
- Router – RunManagementService

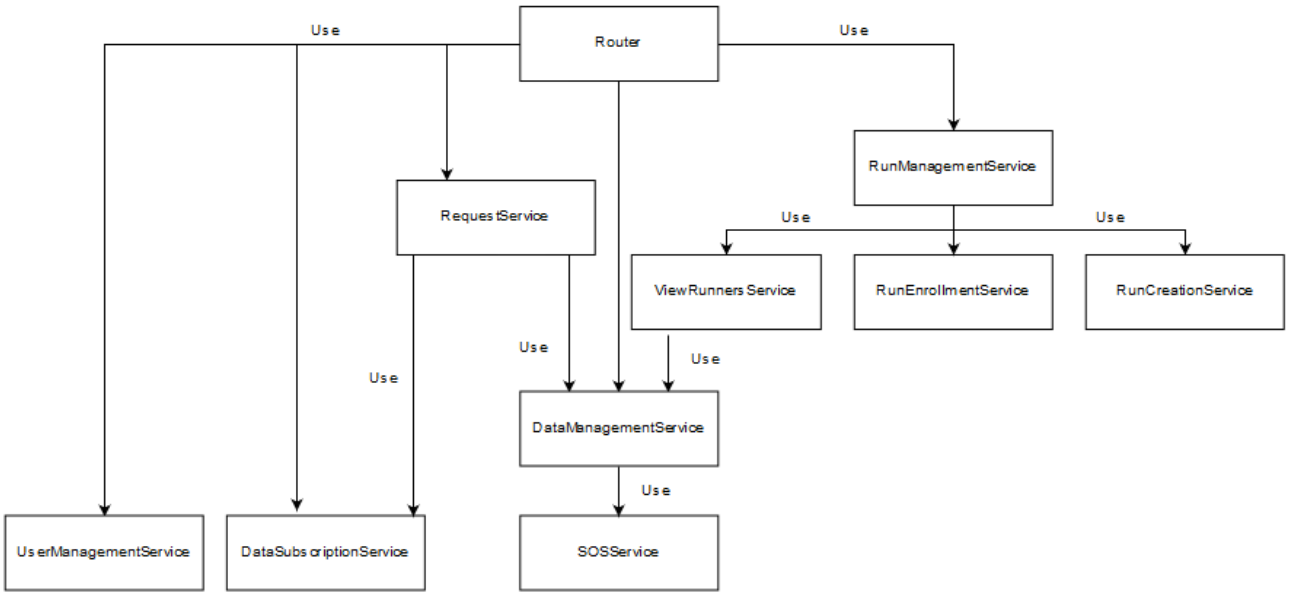


Figure 12

4. Integration of the client application with the web server and the application server:

This last integration is carried out to enable the sending of requests to the server via the mobile application or the web application which the user will use

6. Effort Spent

- Mattioli Daniele: 54.5 hours
- Romano Leonardo: 30 hours
- Tonelli Alessio: 20.5 hours

7. References

- Specification document “Mandatory Project Assignment AY 2018-2019”
- “Design” from Beep
- “Verification and Validation” from Beep