Politecnico di Milano
AA 2018-2019

POLITECNICO
MILANO 1863

Computer Science and Engineering
Software engineering 2

Data4Help DD

Design Document
Version 1.0
14/11/18

Diego Piccinotti, Umberto Pietroni, Loris Rossi

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to further analyze the design and architectural choices for the system to be. Whereas the RASD presented a general view of the system and its features, this document will detail those concepts by showing components of the system, its run-time behavior, deployment plan and algorithm design.

The document will therefore contain a presentation of:

- Overview of the high-level architecture
- Main components and their interfaces provided one for another
- Runtime behavior
- Design patterns
- Algorithm design of the most critical parts of the application
- Implementation plan
- Integration plan
- Testing plan

1.2 Scope

(An SDD shall identify the design stakeholders for the design subject.
An SDD shall identify the design concerns of each identified design stakeholder.
An SDD shall address each identified design concern.

Stakeholders are those people, groups, or individuals who either have the power to affect, or are affected by the endeavor you're engaged with.
)

1.3 Definitions, Acronyms, Abbreviation

1.3.1 Definitions

- **User**: individual who allows *Data4Help* to monitor his location and health status.
- **Third party**: individual or organization registered to *Data4Help* which can request users' data.
- **Data collection**: gathering of users' data through a wearable device.

- **Anonymized data**: data about more than 1000 users whose personal information has been previously removed so that they are not directly relatable to the system's users.
- **Elderly**: user who is subscribed to *AutomatedSOS* and is older than 60 years old.
- **Risk threshold**: Set of boundary health parameters defined for each elderly. If monitored values of the user's health parameters get below these boundaries, an SOS request is placed to an external ambulance provider.
- **Athlete**: user who participates in a run.
- **Run organizer**: third party who can manage runs for athletes.
- **Spectator**: non-registered individual who follows a run through a map with runners' positions.
- **Wearable**: a personal device provided with biometric sensors and GPS given to each user for free after the registration process.

1.3.2 Acronyms

- **API**: Application Programming Interface
- **DB**: Database
- **DBMS**: Database Management System
- **DD**: Design Document
- **GUI**: Graphical User Interface
- **RASD**: Requirements Analysis and Specifications Document
- **GPS**: Global Positioning System
- **GSR**: Galvanic Skin Response

1.3.3 Abbreviations

- **Gn**: n^{th} goal.
- **Dn**: n^{th} domain assumption.
- **Rn**: n^{th} functional requirement.
- **Rn-NF**: n^{th} nonfunctional requirement.

1.4 Reference Documents

- IEEE Standard on Software Design Descriptions (*IEEE Std 1016-2009*)

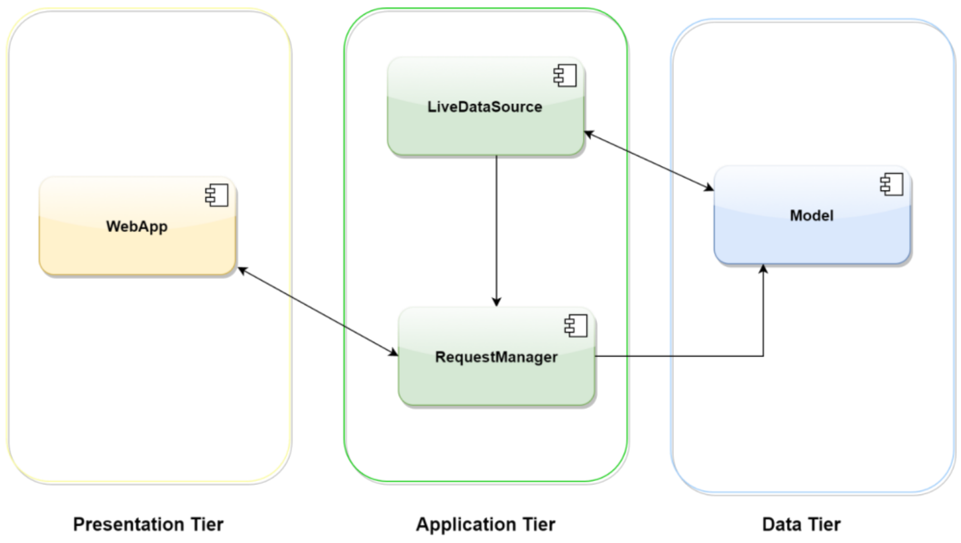
1.5 Document Structure

2 Architectural Design

2.1 Overview:

(High-level components and their interaction)

This system is based upon a three-tier architecture, divided in three "layers" of logical computing. The first one is Presentation Tier which is the front-end and consist of the user interface of the web app and all the client logic. Secondly the Application Tier which contains the functional business logic and lastly the Data Tier that is responsible for the database storage system and data access.



The high-level component related to the presentation layer is the WebApp component, whose task is to show the user interface and react to user input.
The data tier corresponds to the Model component that store data in the database and make queries.
The application tier is composed by two main component: LiveDataSource, which is responsible for collecting users' data and then redistributing them immediately to third parties or storing it in the database, and RequestManager which contains all the logic related to third parties' requests and subscriptions.

2.2 Component View

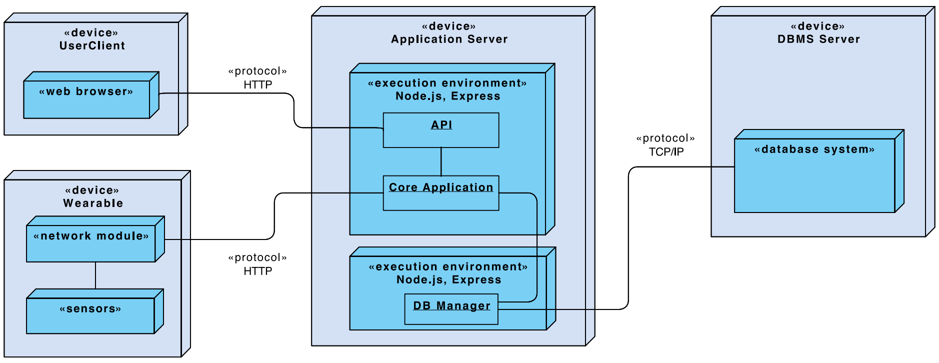
2.3 Deployment View

The deployment diagram shows the physical topology of the system.
We can see the three-tier architecture explicitly.

In the Application Server, there is a dedicated node to deploy the DB Manager. This way it is possible to maintain and update the DB Manager without shutting off the whole Application Server.

Commentato [LR1]: Actually, in a Continuos Integration environment, the system never shuts down. We can discuss this together

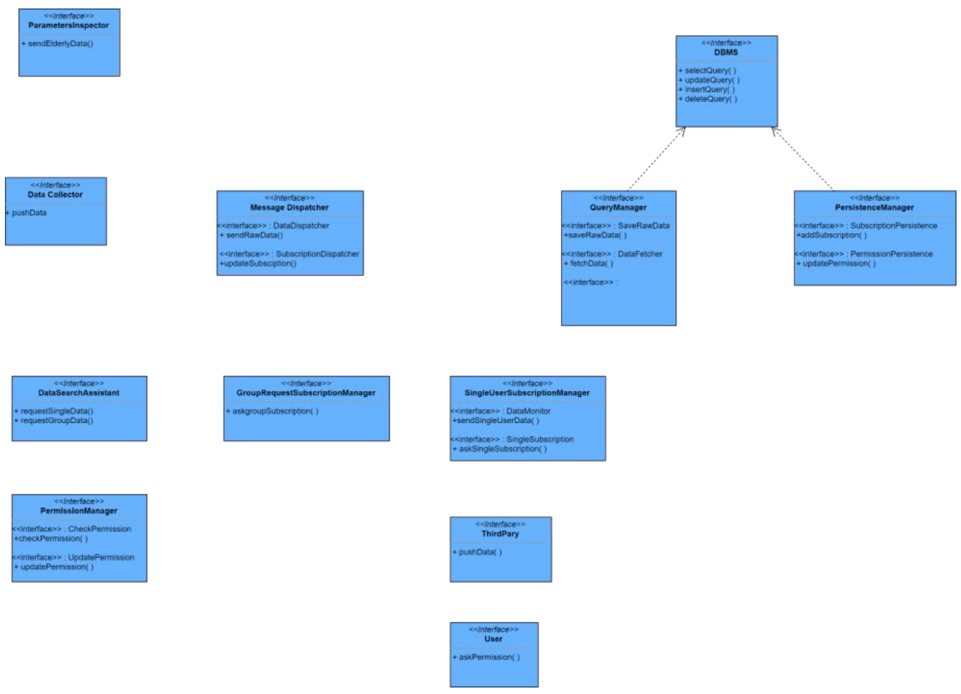
(more to be added)



2.4 Runtime View

(You can use sequence diagrams to describe the way components interact to accomplish specific tasks typically related to your use cases)

2.5 Component Interfaces



(not completed, must be modified and added new interfaces and dependencies)

2.6 Selected Architectural Styles And Patterns

(Please explain which styles/patterns you used, why, and how)

2.6.1 General architecture

Our system is a three-tier application.
The client is used to enabling interaction between users and the system. The client is basically a representational tier, which asks for data and shows it when available.
The business logic is handled by the Application server. It manages every kind of interaction between the system and the external world (users' clients, wearable, external services), and communicates with the DBMS, which stores all the application data.
The three tiers communicate in a client-server fashion. The Application server exposes an API for the client, and the DB manager inside the application server makes requests to the DBMS.

Commentato [DP2]: I would put it in a more hypothetic way, or provide some reasons for the choice

2.6.2 Subsystems architecture

- **Data collector**
  The data collector can be implemented following the event-based paradigm. The data collector itself is the event dispatcher. It receives data from users' wearables, and then it broadcasts this data to the DBMS (in order to storage the data) and to the Message Dispatcher.
  This paradigm allows a high modularity between different components of the systems, significantly reducing coupling and enhancing flexibility.
  Since all the data has its own timestamp, it's not relevant if the data collector dispatches data out of order.
  The weakness of this paradigm is that the data collector can be a bottleneck of the system. In order to address this problem, it is possible to set a policy for which, once a defined amount of wearable is subscribed to the data collector, a new instance of data collector is spawn, and all new wearables will subscribe to this new one.

- **Message dispatcher**
  The message dispatcher has the same architecture of the data collector. Its role is to sends real-time data to the components that need them, specifically the ParametersInspector, SingleUserSubscriptionManager and the MapManager. So, the message dispatcher is an event dispatcher, receives data from the data collector and broadcasts this data to the components mentioned above.
  The message dispatcher stores internally a list of all the AutomatedSOS users, single user subscriptions and the list of active runners. When there are some modifications on the DB, the message dispatcher is updated accordingly. The message dispatcher can also ask for this data to the DB in case of reboot.
  The message dispatcher shall be very reliable, since the ParametersInspector depends on it. For this reason, it is necessary to have parallel components in order to increase robustness to failures.
  Regarding scalability, it is possible to use a Load Balancer which receives data from the data collector, and distributes this data to multiple instances of message dispatcher, accordingly to their requests load. These multiple instances shall have a shared memory, so they all know to which component they have to redirect data.

Commentato [LR3]: This is just a general draft, all the text must be reviewed and improved with a better English haha

Commentato [LR4]: Change this name with the one defined in the component diagram

- **Parameters inspector**
  (here is very important to say that availability should be really high, so we need to instantiate multiple components in parallel)
  The parameters inspector is an event listener. It waits for users' data from the message dispatcher, and then check if this data goes beyond the owner's risk thresholds.
  If some data goes beyond risk thresholds, the parameters inspector begins a rescue procedure, where it contacts the ambulance provider and provides the health and position data of the ill user.

Commentato [LR5]: It needs to know the risk thresholds. It needs to communicate with the permanent storage

2.7 Other Design Decisions

3 User Interface Design

(Provide an overview on how the user interface(s) of your system will look like; if you have included this part in the RASD, you can simply refer to what you have already done, possibly, providing here some extensions if applicable.)

4 Requirements Traceability

(Explain how the requirements you have defined in the RASD map to the design elements that you have defined in this document.)

G1: The user can be recognized by providing a form of identification.

- DataCollector [R2]
- User

G2: Allow third parties to monitor data about location and health status of individuals.

G3: Allow third parties to access data relative to specific users.

G4: Allow third parties to access anonymized data of groups of users.

G5: Allow third parties to offer a personalized and non-intrusive SOS service to elderly people so that an ambulance arrives to the location of the customer in case of emergency.

G6: Allow athletes to enroll in a run.

G7: Allow organizers to manage runs.

G8: Allow spectators to see on a map the position of all runners during the run.

5 Implementation, Integration And Test Plan

(Identify here the order in which you plan to implement the subcomponents of your system and the order in which you plan to integrate such subcomponents and test the integration.)

6 Effort Spent

(In this section you will include information about the number of hours each group member has worked for this document.)

6.1 Piccinotti Diego

Description of the task	**Hours**
Purpose, Scope, Definition	1
High-level Components and their Interaction	3
Component View	2
Deployment View	
Runtime View - Sequence Diagrams	
Selected Architectural Styles and Patterns	
Component Interfaces	
Algorithm Design	
User Interface Design	
Requirements Traceability	
Implementation, Integration and Test Plan	

6.2 Pietroni Umberto

Description of the task	**Hours**
Purpose, Scope, Definition	1
High-level Components and their Interaction	3
Component View	2,5
Deployment View	
Runtime View - Sequence Diagrams	
Selected Architectural Styles and Patterns	
Component Interfaces	2
Algorithm Design	
User Interface Design	
Requirements Traceability	
Implementation, Integration and Test Plan	

6.3 Rossi Loris

Description of the task	**Hours**
Purpose, Scope, Definition	1
High-level Components and their Interaction	3
Component View	2
Deployment View	1.5
Runtime View - Sequence Diagrams	
Selected Architectural Styles and Patterns	1
Component Interfaces	
Algorithm Design	
User Interface Design	
Requirements Traceability	
Implementation, Integration and Test Plan	

7 References