Politecnico di Milano
AA 2018-2019

Computer Science and Engineering

Software Engineering 2 Project

By Track Me

D.D.

Design Document

Tommaso Peresson - 845427
Giacomo Ziffer – 920905
Version 1.0 - 10/12/2018

Contents

1 Introduction

1.1 Purpose

The Design Document (DD) contains a functional description of TrackMe's System. This document explains each component that will be inserted into the system, its architecture and the design patterns that will be implemented to ensure that all the requirements are satisfied. Components will be described both at High Level and more in depth, illustrating and explaining all the subcomponents every component is made of. The reader of this document will get a clear idea about its architecture (hardware and software), whether he wants to have a detailed description of the system or a more general one.

1.2 Scope

The purpose of the project is to develop a software-based service allowing third parties to monitor the location and health status of individuals.
By providing a simpler and more convenient way to communicate health data, the purpose of Data4Help is to offer a service that closes the gap between the doctors and patients and even facilitates research and analysis by third parties. A simple to use application will be distributed to all patients (from now on called Private Customers) through which Data4Help will be able to collect their personal health data that will be stored and processed. Afterwards the intent is to be able to offer this data, following all privacy concerns, to doctors and researchers (called business customer) to improve the reach and precision of their studies and diagnosis.
In order to exploit fully the capabilities of this platform Data4Help is interested also in release a sub-service called AutomatedSOS to provide immediate help to people in serious health conditions, by sending to the location of the customer an ambulance, when the parameters are below the threshold.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

- Private Customer: a customer that applies to the service Data4Help as a provider of personal health data and that can subscribe to AutomatedSOS
- Business Customer: a customer that applies to the service Data4Help as a user of the data acquired. It can be a single individual, such as a doctor, but also a company
- Anonymized Request: request made by a Business Customer, who asks for data grouped according to different possible parameters (i.e. place, age, etc.)
- Individual Request: request made by a business customer, who requests data belonging to a specific Private Customer, in this case the request must be accepted by the PC.

1.3.2 Acronyms

- [BC] as Business Customer
- [PC] as Private Customer
- [SM] as System Manager
- [Gn]: n-goal
- [Rn]: n-functional requirement
- [API]: application programming

1.4 Revision history

- 1.0: Initial release of Design Document – 10/12/18

1.5 Reference documents

- RASD document previously delivered

1.6 Document Structure

The paper includes sixth areas. The first one is composed by the introductory information provided in order to give an indicative view of what this document is

about. The second one provides an in-depth description of each Architectural Design aspect which reflects the decisions that the team made, both from a software and a hardware perspective. Then a User Interface Design section presents some examples of the user interface from an actor point of view. The fourth section is about the "Requirements Traceability", which consists in a mapping of the requirements defined in the RASD to the specific module that will guarantee its satisfaction. The sixth section identifies the order in which the subcomponents of the system will be implemented, integrated and tested. Finally, a seventh section allows the reader to learn the effort spent on this project by each team member.

2 Architectural design

2.1 Overview

The system that implement TrackMe relies substantially on two products:

- an *application*, which is used to collect data from Private Customers and provide the AutomatedSOS service, analyzing user data as soon as they will be received in real time
- a *web app*, which is used to receive requests from Business Customers and to show the results of their requests. This web app is also used by the System Manager, which access from another port, and which can verify the data of a registration request as a Business Customer. To further clarify this aspect, Business Customer's and System Manager's web app will be hosted on the same server but they will export their interfaces on two different ports on the same IP (e.g. 99.99.99.99:80 and 99.99.99.99:81)

In general, Data4Help service will be based on a three-tier architecture distribute on multiple physical systems:

- *Presentation layer:* The purpose of this layer is to present data to the user sent by the application layer and gather user inputs and forward them to the layer beneath.
- *Application layer:* Represents the model and the application logic. It makes queries to the database, receives inputs from the adjacent Presentation Layer check their integrity and commits changes to the model.
- *Data layer:* Layer in which information are stored. It satisfies the requests of the application layer in order to provide and store the required data

In order to provide a more flexible and safer platform, it's necessary to move all the computational load of the complex queries from the user's device to Data4Help's server. As can be seen from Figure 1, the only exception is Automated SOS, which represents the logic layer within the application available to Private Customers, this decision was taken in order to provide a much faster and more efficient service. Figure 1 also shows a data layer, which will only be used for caching purposes to ensure offline reliability, when data cannot be sent to the Data4help server at the moment.

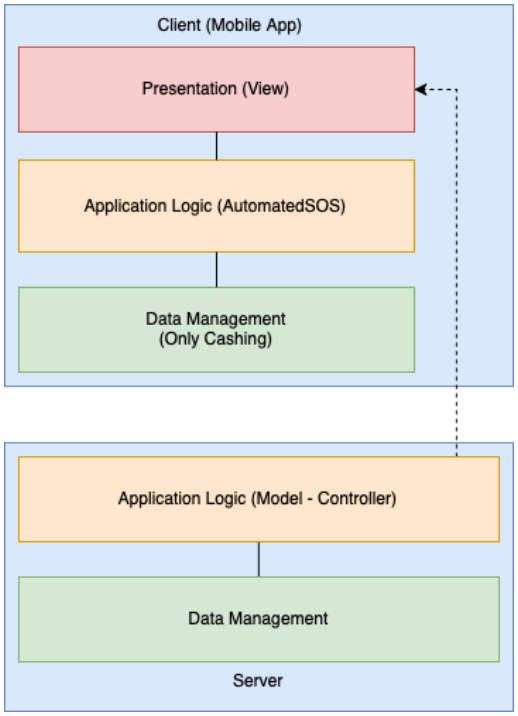


Fig. 1: General tier architecture of Data4Help's Mobile App

Regarding Figure 2 the network will sit between the Presentation Layer and the Application Layer. That it's necessary to move all the computational load of the complex queries from the Business Customer's device to Data4Help's server and to provide a more flexible and safer platform.

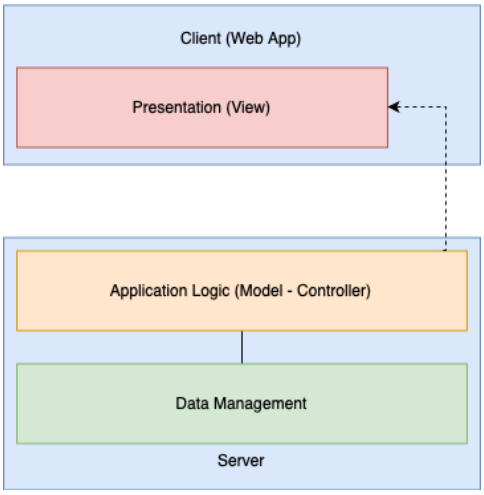


Fig. 2: General tier architecture of Data4Help's Web App

2.1.1 High level components and their interaction

The mobile application located on the client's phone connects to the central server by sending REST API calls in asynchronous mode in a separated thread. The main server replies with the information requested using the JSON encoding, which is reinterpreted locally by the application.

We have chosen AWS Elastic Beanstalk as our deployment platform because highly available and scalable web hosting can be a complex and expensive proposition. Traditional scalable web architectures have not only needed to implement complex solutions to ensure high levels of reliability, but they have also required an accurate forecast of traffic to provide a high level of customer service. Dense peak traffic periods and wild swings in traffic patterns result in low utilization rates of expensive hardware. This yields high operating costs to maintain idle hardware, and an inefficient use of capital for underused hardware. Amazon Web Services (AWS) provides a reliable, scalable, secure, and highly performing infrastructure for the most demanding web applications. This infrastructure matches IT costs with customer traffic patterns in real time.

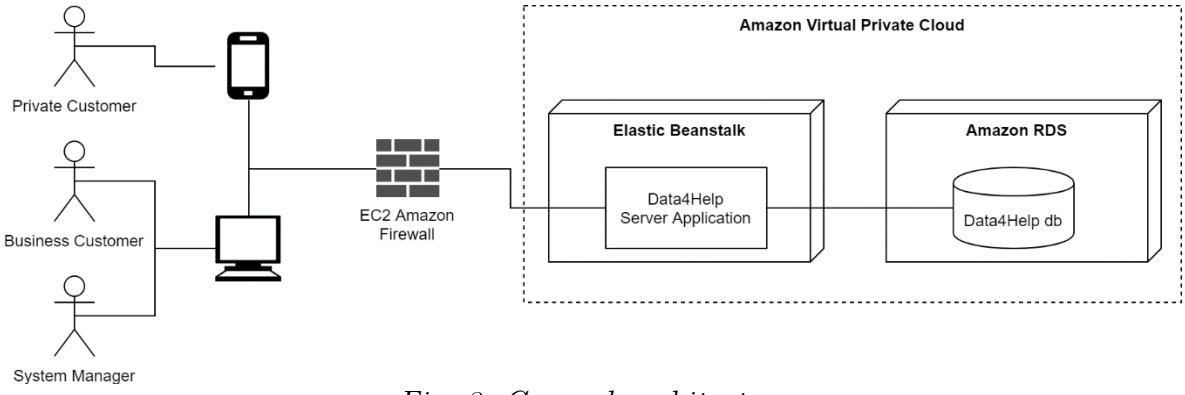


Fig. 3: General architecture

2.2 Component view

In this section we will describe the internal structure and the interconnections of the inner components of Data4Help service.

2.2.1 Main Components

This diagram represents the components in the most general possible way, defining the physical systems and their interconnections.

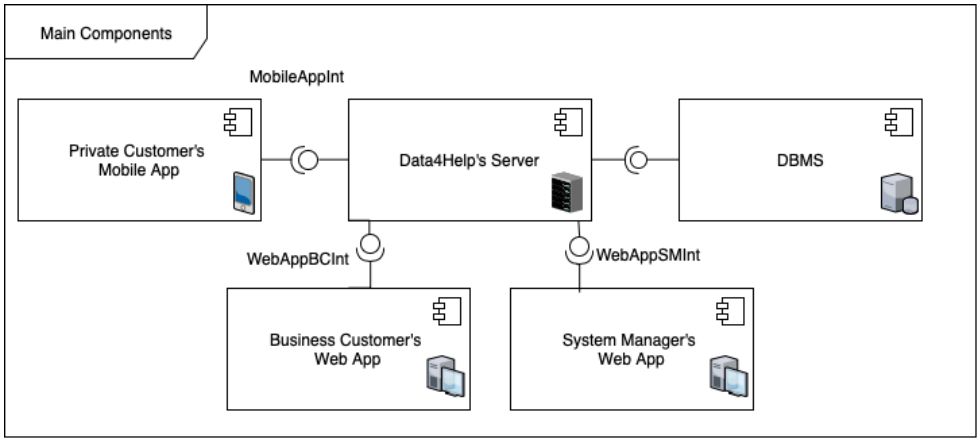


Fig. 4: Main components of Data4Help

2.2.2 Data4Help server's components

As the Fig. 4 shows, the system will be composed of two main subsystems: user services and request services. Both sub-systems will interface with the mobile app and the web app.
Each module that compose the subsystems is interfaced with the Database (specifically the DBMS) in order to let each part work independently and simultaneously.

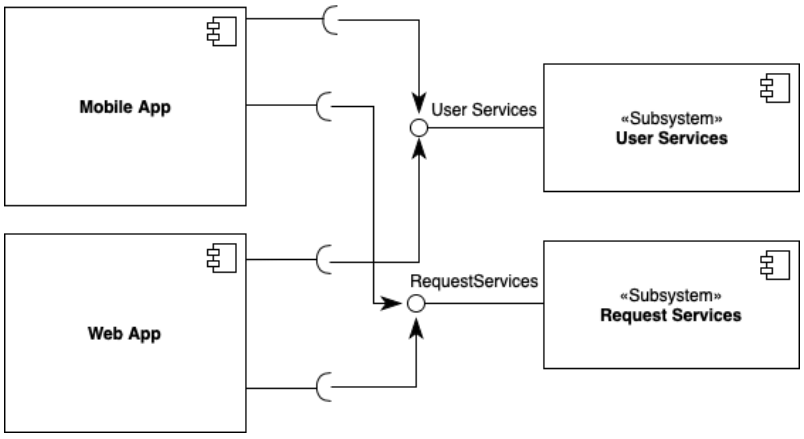


Fig. 5: Server's components.

2.2.3 AutomatedSOS component view

AutomatedSOS will be implemented as a subsystem in the Mobile App, that is necessary in order to achieve the required functionality of immediate rescue in case of emergency. The health data will be analyzed immediately after acquisition, without having to be sent to the server, that prevents any network and server-related downtime. Fortunately, the algorithm performing the analysis of the parameters is light-weight enough to be implemented efficiently on a mobile system.
This subsystem will be using the Health and Phone interface provided by Android OS.

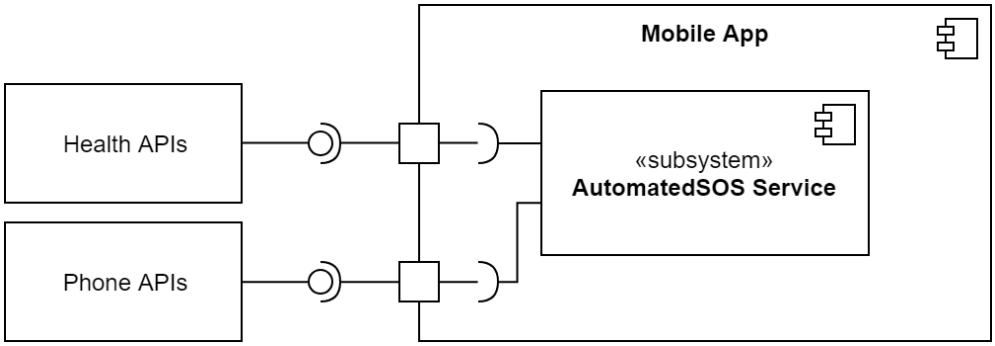


Fig. 6: Mobile App Components.

2.2.4 Component view of the Request Services

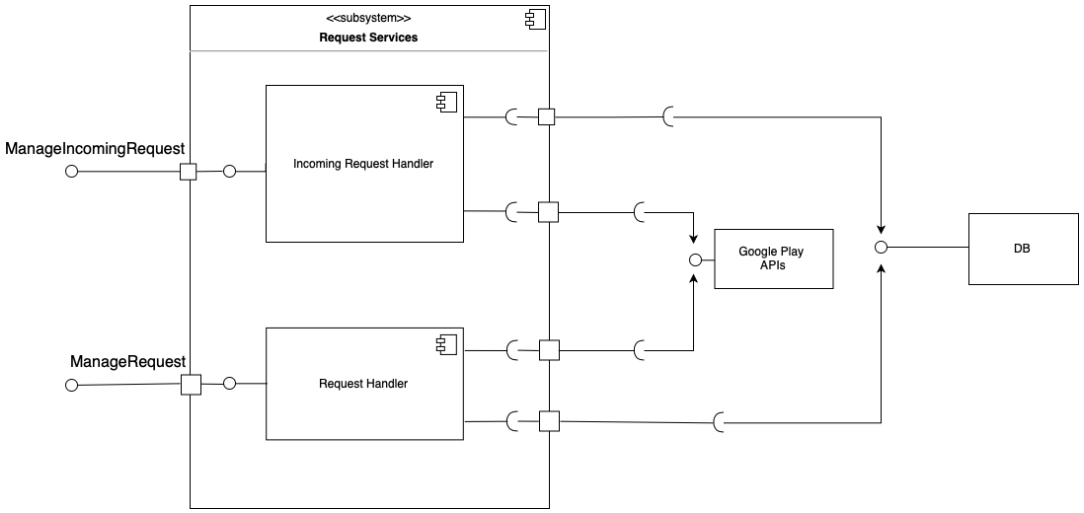


Fig. 7: Request Services component view diagram

The *Request Services* subsystem will consist of 2 main modules:

- *Incoming Request Handler*, that will interface only with the mobile application, providing methods to accept or reject the request from a Business Customer, showing all the information available on the latter.
- *Request Handler*, which will interface only with the web app used by Business Customer, providing methods to see information about made requests (updates or results) and to make others, both individual and anonymized requests.

In addition to interfacing with the database, both modules use the Google Play APIs to send notifications to the Private Customer and the Business Customer.

2.2.5 Component view of the User Services

The *User Services* subsystem will instead consist of 3 main modules:

- *PrivateCustomer Module*, which will interface only with the mobile app, providing all the methods of managing the account (login, register, subscribe, etc.) and which also allows the Private Customer to see his/her personal data.
- *BusinessCustomer Module*, which will interface only with the web app (business customer side) that will provide the methods for managing the account for the Business Customer
- *SystemManager Account*, that will interface only with the web app (system manager side) providing methods to see the business customers who want to register and to accept / reject them

All 3 modules will use an *External Billing Service* to proceed with the service subscription process, furthermore the business customer module and system manager module will also use Google Play APIs to send notifications.

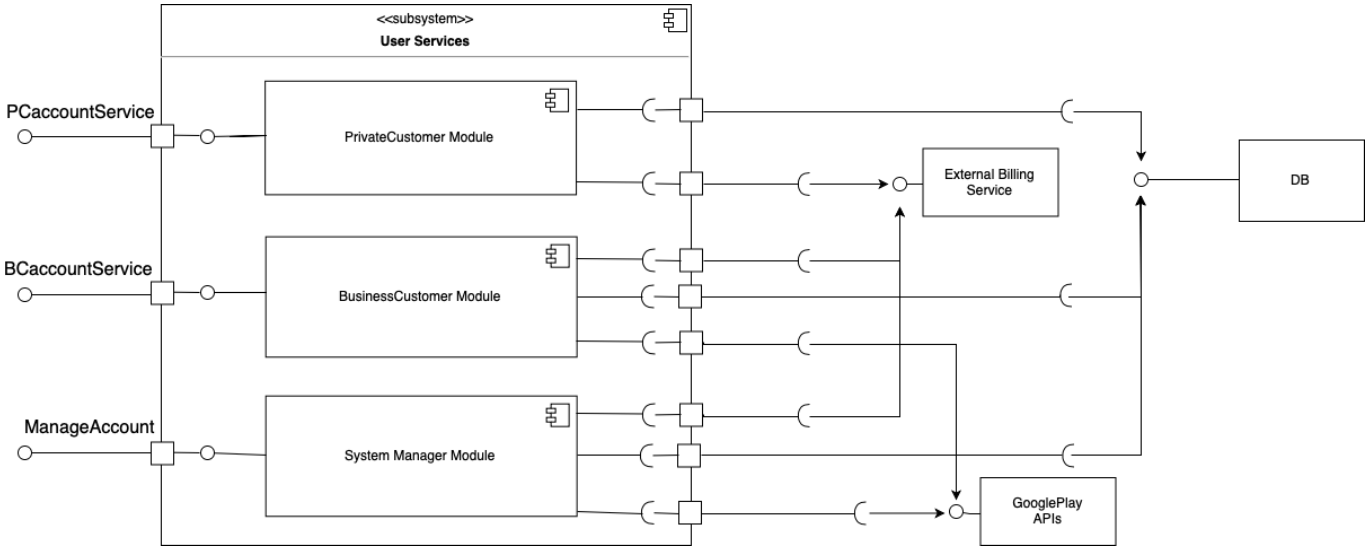


Fig. 8: User Services component diagram

2.2.6 Entity Relationship Diagram

The following diagram provides a graphical representation of the Database Schema that will be adopted by the system.

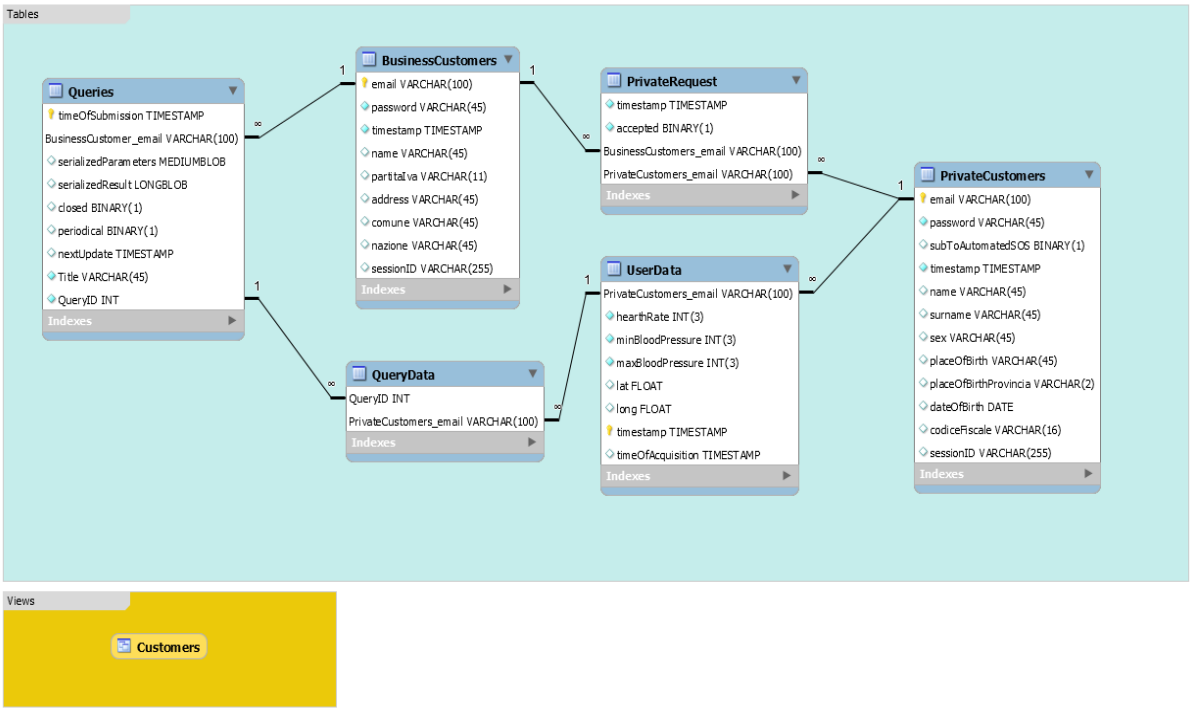


Fig. 9: Entity Relationship Diagram

To further explain fig. 8:

- *QueryData:* represents all the data that matches for an anonymous request.
- *UserData:* each row is a unique telemetry data acquisition from a Private Customer.
- *PrivateRequest:* represents the request for permission to access Private Customer data from a Business Customer
    - *SubToAutomatedSOS:* Binary(1), represents if a Private Customer is currently subscribed to AutomatedSOS.
- *Queries:* each row is an anonymous request.
    - *Closed:* Binary(1), represents the state of the query. Normally this bit is set to zero but will be set to one in case of an expired periodical query.
    - *Periodical:* Binary(1), identifies a query that will be executed periodically.

2.3 Deployment view

As stated before, Data4Help will be composed by a three-tier architecture over Amazon Elastic Beanstalk and Amazon RDS.

- *Presentation Layer:* The mobile app will run on Android Smartphones and the Web App on desktop browsers.
- *Logic Layer:* The main webserver and application logic will be deployed on an instance of elastic beanstalk to provide maximum scalability and efficiency.
- *Data Layer:* A MySQL database will be deployed on an instance of Amazon RDS.

These three layers will be communicating through http protocol and MySQL protocol respectively.

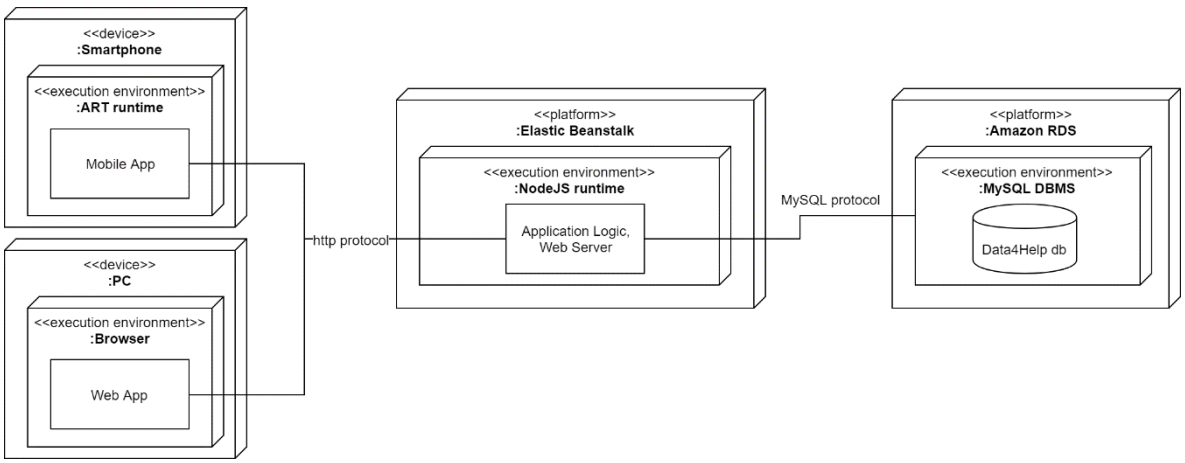
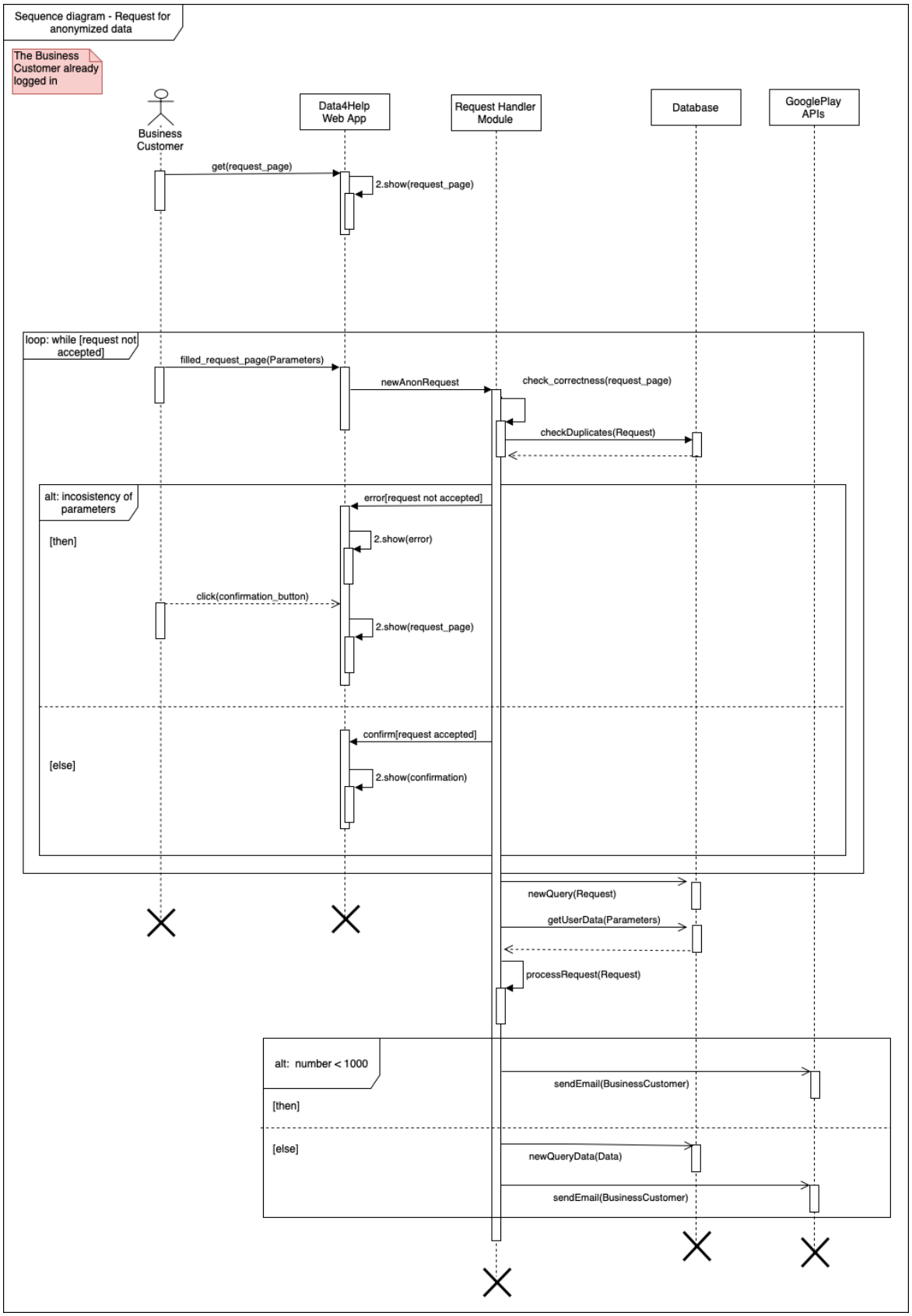


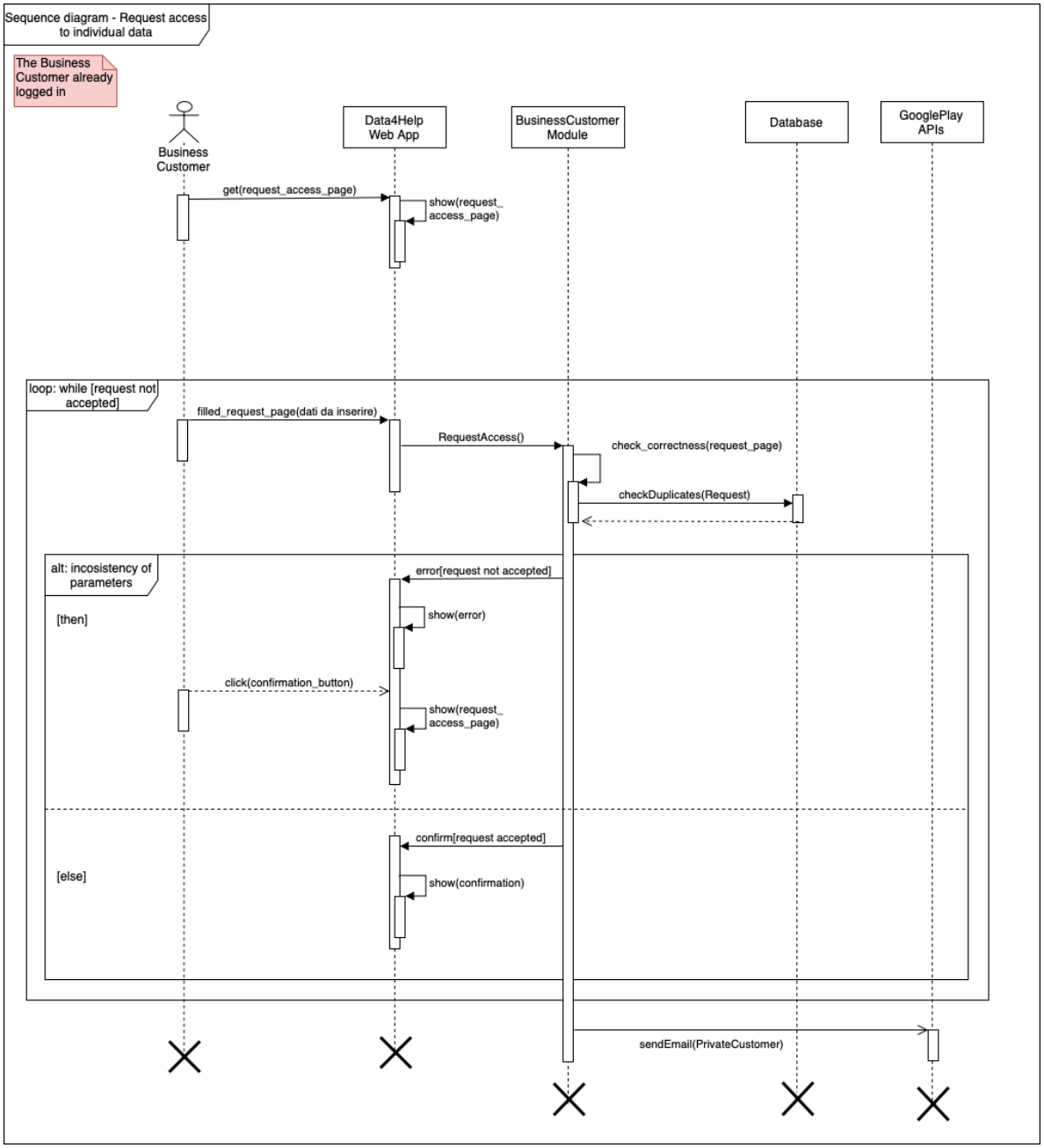
Fig. 10: Deployment Diagram

2.4 Runtime view

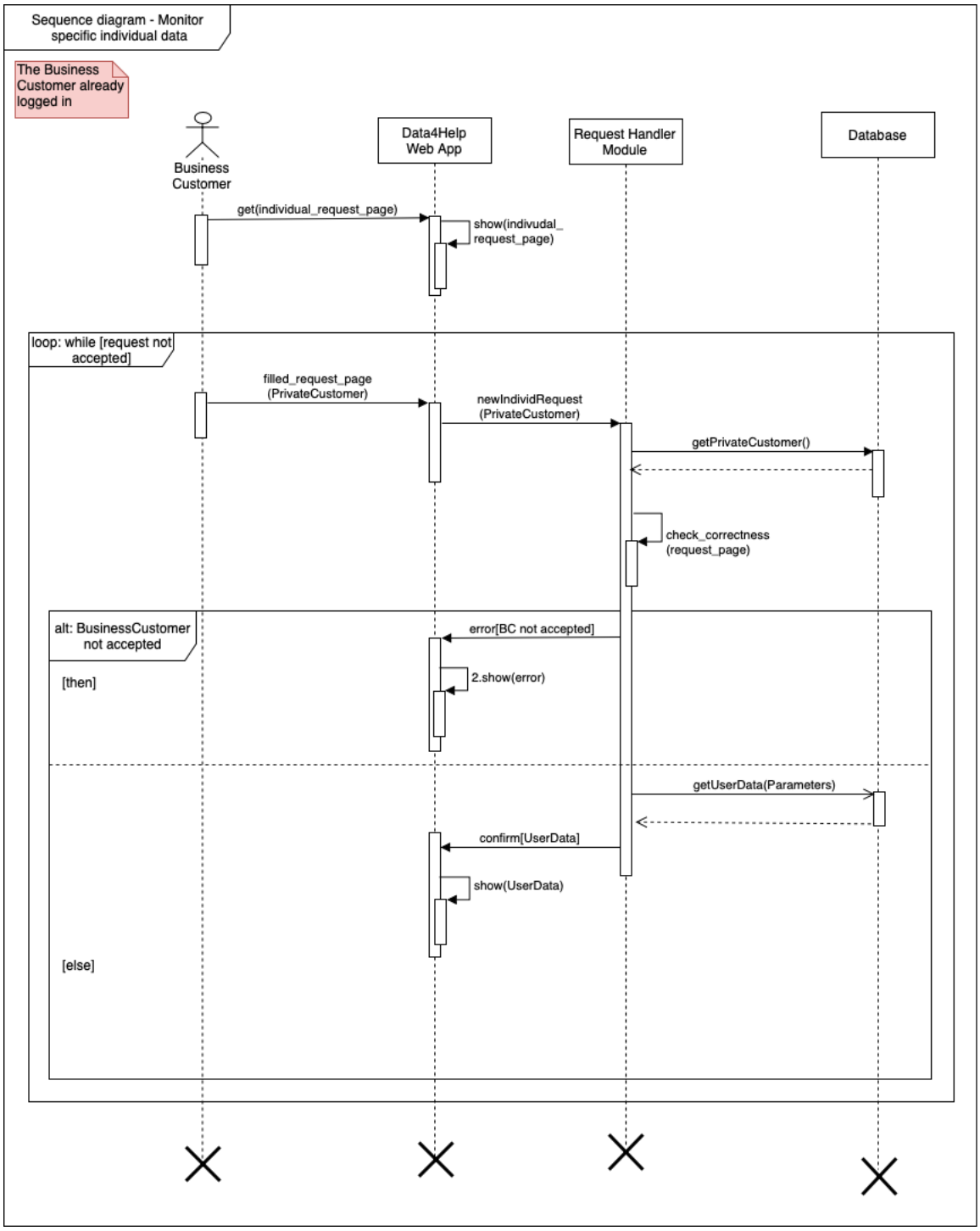
2.4.1 Request for anonymized data



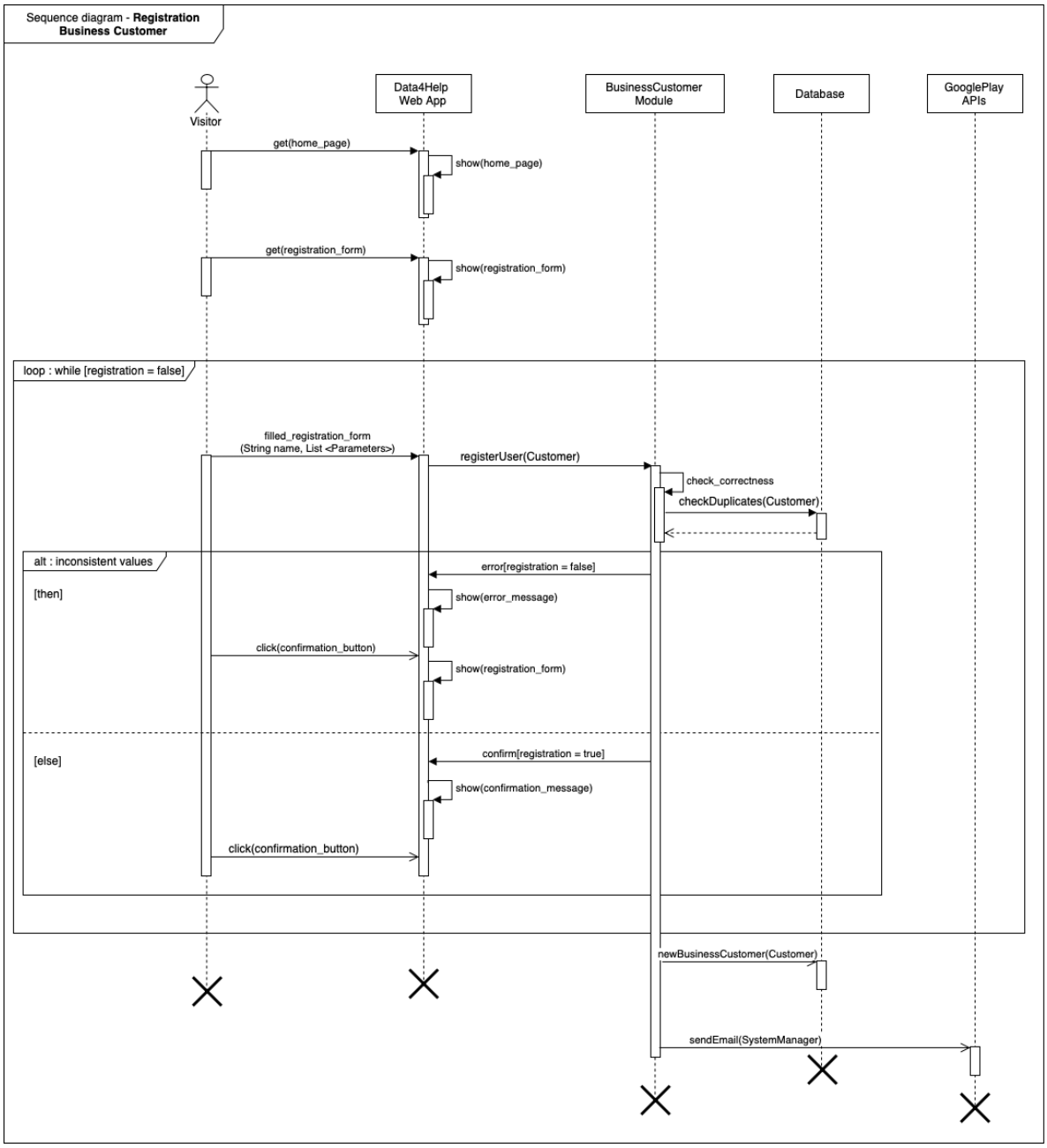
2.4.2 Request for individual data



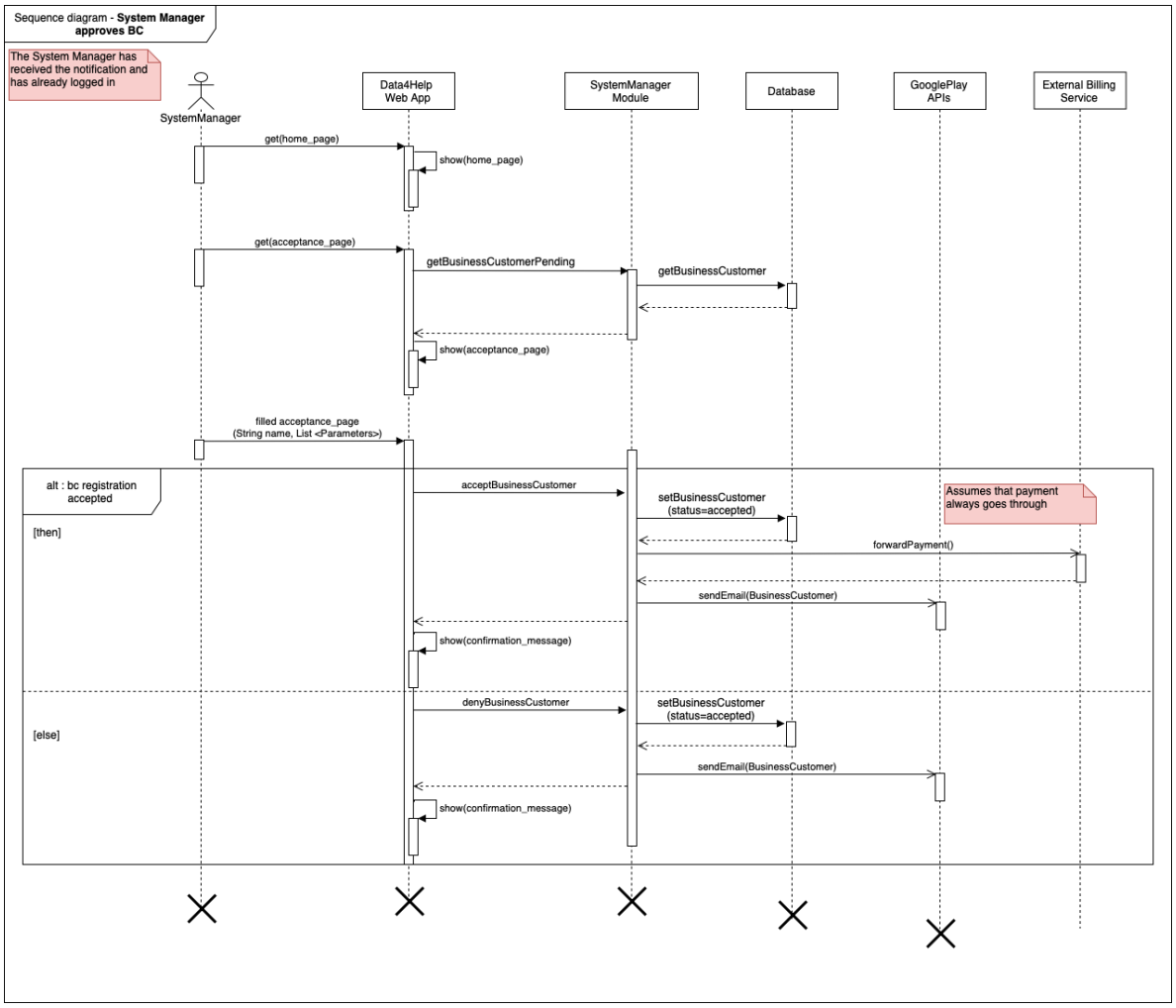
2.4.3 Monitor of individual data



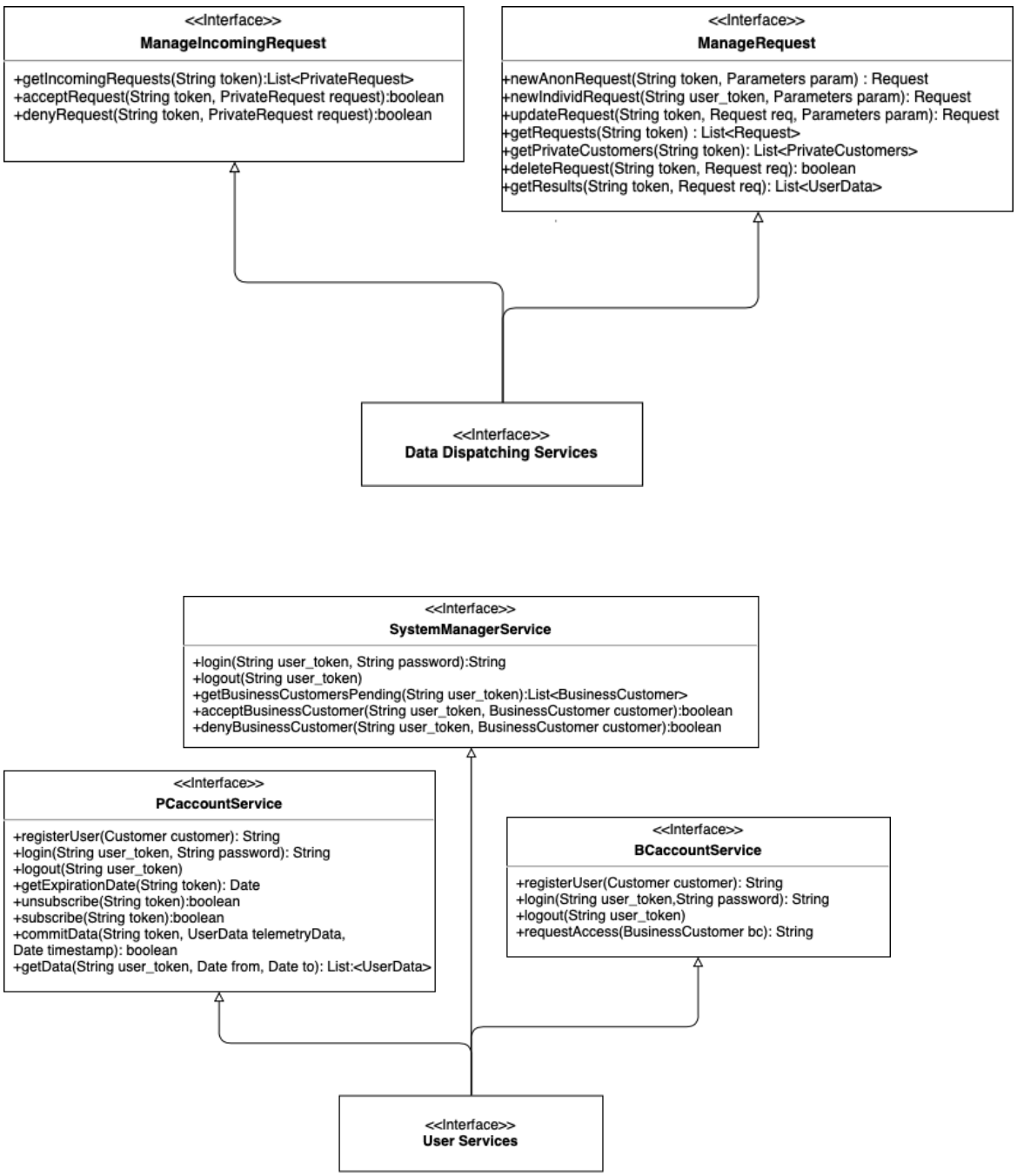
2.4.4 Business Customer registration



2.4.5 Business Customer registration approval



2.5 Component interfaces



2.5.1 RESTful API

Our presentation layer will be connected through a network that will use a JSON RESTful Application Programming Interface to the application layer. The API will lie on a Web server and will be called on a HTTP channel with a classic TLS encryption layer. All the methods will require an authentication, except for the login one.

Private Customer

- /api/pc/login
  POST: Allow the user to get an authentication token.
  * Parameters: user_token A user token that is generated client side by the application
  * Return: device_token A token that will be used for future authentication on that device with other APIs

- /api/pc/register
  PUT: Add new PrivateCustomer - registerUser(Customer customer): String
- /api/pc/data
  GET: Retrieve the user data - getData(·): List<UserData>
  PUT: Add new user data - commitData(·): boolean
- /api/pc/request
  GET: Retrieve the incoming request - getIncomingRequest: List<PrivateRequest>
- /api/pc/request/{request_id}
  PATCH: Update the status of the individual request - accept/deny Request

System Manager

- /api/sm/login
  POST: Allow the user to get an authentication token.
  * Parameters: user_token A user token that is generated client side by the application
  * Return: device_token A token that will be used for future authentication on that device with other APIs
- /api/sm/businesscustomer
  GET: Get all pending business customer - getBusinessCustomerPending(): List<BusinessCustomer>
- /api/sm/businesscustomer/{businesscustomer_id}
  PATCH: Update business customer - accept/denyBusinessCustomer

Business Customer

- /api/bc/login
  POST: Allow the user to get an authentication token.
  *Parameters: user_token A user token that is generated client side by the application
  * Return: device_token A token that will be used for future authentication on that device with other APIs
- /api/bc/register
  PUT: Add new PrivateCustomer - registerUser(Customer customer): String
- /api/bc/access
  PUT: Add new AccessRequest - requestAccess

2.6 Selected architectural styles and patterns

2.6.1 Overall Architecture

The system will be composed of three tiers:

1.1 A client executable running on Android OS

1.2 A client web app running on a generic desktop browser

2. A server application running on an Elastic Beanstalk NodeJS instance
3. A MySQL server instance running on Amazon RDS

The mobile client [1.1] will cash in the file system of the smartphone all data that can't immediately be sent to Data4Help's server. All acquired data will be stored by the server application on the MySQL database instance.

2.6.2 Data Definition Language

The following DDL in a SQL-like language is proposed for the database:

```
-- -----------------------------------------------------
-- Schema Data4Help
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `Data4Help` DEFAULT CHARACTER SET utf8 ;
USE `Data4Help` ;

-- -----------------------------------------------------
-- Table `Data4Help`.`BusinessCustomers`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Data4Help`.`BusinessCustomers` (
  `email` VARCHAR(100) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `timestamp` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `name` VARCHAR(45) NULL DEFAULT NULL,
  `partitaIva` VARCHAR(11) NULL DEFAULT NULL,
  `address` VARCHAR(45) NULL DEFAULT NULL,
  `comune` VARCHAR(45) NULL DEFAULT NULL,
  `nazione` VARCHAR(45) NULL DEFAULT NULL,
  `sessionID` VARCHAR(255) NULL DEFAULT NULL,
  PRIMARY KEY (`email`),
  UNIQUE INDEX `partitaIva_UNIQUE` (`partitaIva` ASC) VISIBLE)


-- -----------------------------------------------------
-- Table `Data4Help`.`PrivateCustomers`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Data4Help`.`PrivateCustomers` (
  `email` VARCHAR(100) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `subToAutomatedSOS` BINARY(1) NULL DEFAULT NULL,
  `timestamp` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `name` VARCHAR(45) NULL DEFAULT NULL,
  `surname` VARCHAR(45) NULL DEFAULT NULL,
  `sex` VARCHAR(45) NULL DEFAULT NULL,
  `placeOfBirth` VARCHAR(45) NULL DEFAULT NULL,
  `placeOfBirthProvincia` VARCHAR(2) NULL DEFAULT NULL,
  `dateOfBirth` DATE NULL DEFAULT NULL,
  `codiceFiscale` VARCHAR(16) NULL DEFAULT NULL,
  `sessionID` VARCHAR(255) NULL DEFAULT NULL,
  PRIMARY KEY (`email`),
  UNIQUE INDEX `codiceFiscale_UNIQUE` (`codiceFiscale` ASC) VISIBLE)
```

```
-- -----------------------------------------------------
-- Table `Data4Help`.`PrivateRequest`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Data4Help`.`PrivateRequest` (
  `timestamp` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `accepted` BINARY(1) NOT NULL DEFAULT '0',
  `BusinessCustomers_email` VARCHAR(100) NOT NULL,
  `PrivateCustomers_email` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`BusinessCustomers_email`, `PrivateCustomers_email`),
  INDEX `fk_PrivateRequest_BusinessCustomers1_idx` (`BusinessCustomers_email` ASC)
VISIBLE,
  INDEX `fk_PrivateRequest_PrivateCustomers1_idx` (`PrivateCustomers_email` ASC)
VISIBLE,
  CONSTRAINT `fk_PrivateRequest_BusinessCustomers1`
    FOREIGN KEY (`BusinessCustomers_email`)
    REFERENCES `Data4Help`.`BusinessCustomers` (`email`)
  CONSTRAINT `fk_PrivateRequest_PrivateCustomers1`
    FOREIGN KEY (`PrivateCustomers_email`)
    REFERENCES `Data4Help`.`PrivateCustomers` (`email`)



-- -----------------------------------------------------
-- Table `Data4Help`.`Queries`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Data4Help`.`Queries` (
  `timeOfSubmission` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `BusinessCustomer_email` VARCHAR(100) NOT NULL,
  `serializedParameters` MEDIUMBLOB NULL DEFAULT NULL,
  `serializedResult` LONGBLOB NULL DEFAULT NULL,
  `closed` BINARY(1) NULL DEFAULT '0',
  `periodical` BINARY(1) NULL DEFAULT NULL,
  `nextUpdate` TIMESTAMP NULL DEFAULT NULL,
  `Title` VARCHAR(45) NOT NULL,
  `QueryID` INT(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`timeOfSubmission`, `BusinessCustomer_email`),
  UNIQUE INDEX `QueryID_UNIQUE` (`QueryID` ASC) VISIBLE,
  INDEX `BusinessCustomer_email_idx` (`BusinessCustomer_email` ASC) VISIBLE,
  CONSTRAINT `BusinessCustomer_email`
    FOREIGN KEY (`BusinessCustomer_email`)
    REFERENCES `Data4Help`.`BusinessCustomers` (`email`)


-- -----------------------------------------------------
-- Table `Data4Help`.`UserData`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Data4Help`.`UserData` (
  `PrivateCustomers_email` VARCHAR(100) NOT NULL,
  `hearthRate` INT(3) NOT NULL,
  `minBloodPressure` INT(3) NOT NULL,
  `maxBloodPressure` INT(3) NOT NULL,
  `lat` FLOAT NULL DEFAULT NULL,
  `long` FLOAT NULL DEFAULT NULL,
  `timestamp` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `timeOfAcquisition` TIMESTAMP NULL DEFAULT NULL,
  PRIMARY KEY (`PrivateCustomers_email`, `timestamp`),
  CONSTRAINT `fk_UserData_PrivateCustomers`
    FOREIGN KEY (`PrivateCustomers_email`)
    REFERENCES `Data4Help`.`PrivateCustomers` (`email`)


-- -----------------------------------------------------
-- Table `Data4Help`.`QueryData`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Data4Help`.`QueryData` (
```

```
`QueryID` INT(11) NOT NULL,
`PrivateCustomers_email` VARCHAR(100) NOT NULL,
PRIMARY KEY (`QueryID`, `PrivateCustomers_email`),
INDEX `fk_UserData_idx` (`PrivateCustomers_email` ASC) VISIBLE,
CONSTRAINT `fk_Queries`
  FOREIGN KEY (`QueryID`)
  REFERENCES `Data4Help`.`Queries` (`QueryID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT `fk_UserData`
  FOREIGN KEY (`PrivateCustomers_email`)
  REFERENCES `Data4Help`.`UserData` (`PrivateCustomers_email`)
```

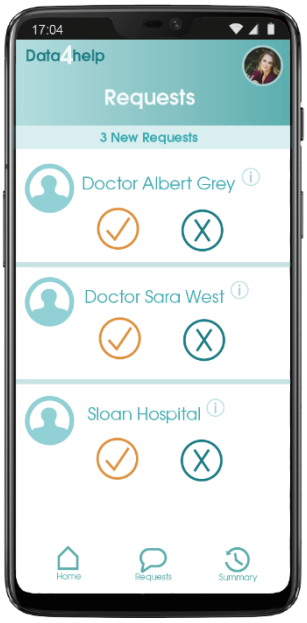
2.6.3 Design Patterns

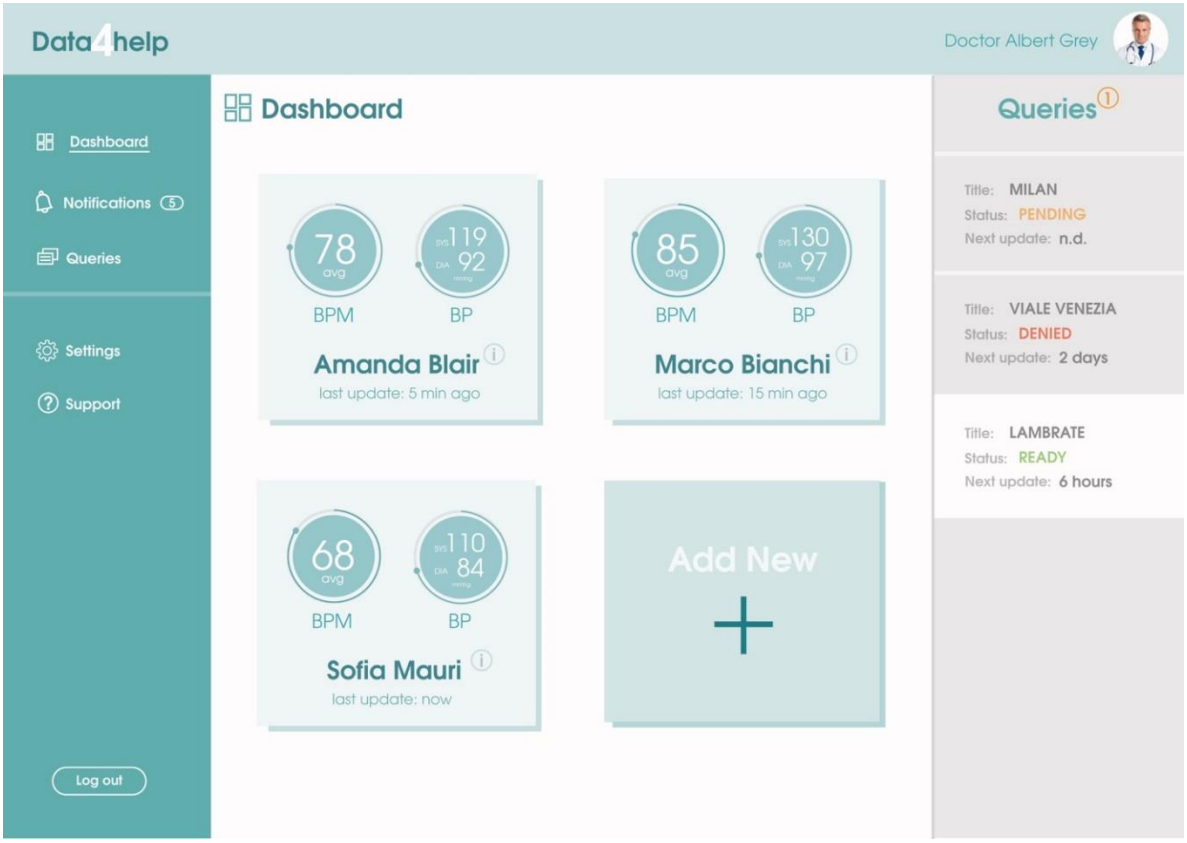
- Publish/Subscribe: Used for the push notifications between the clients and the server
- Facade: Allows the clients to worry less about the different requests that must be made. This pattern hides the complexities of the larger system and provides a simpler interface.
- MVC: Used to separate data structures from the logic and the presentation.
- Client/server: Allows to distribute the computational load from the clients to the server. This contributes to a simpler and easier to maintain client and a more secure and powerful service.

2.7 Other design decisions

As described above, we will use Google Play APIs as an external notification service, this will allow us to send a notification to the mobile phone, if the recipient was a private customer, and to send an email if the recipient was instead a Business Customer.

3 User interface design





These are just few examples of the mockups, in order to get an idea on how the application will look like at the final stage. Further examples can be found at chapter 3.1.1 “User Interfaces” of the RASD document previously delivered.

4 Requirements traceability

The choices presented in this document have the aim to achieve the goals described on the *Requirements Analysis and Specification Document,* Follow a list of goals with their requirements and the components that satisfy them.

- [G1] Allow a Visitor to become a Private Customer.
  - [R1][R2][R3][R4] User Services: PrivateCustomer Module through *PCaccountService*

- [G2] Allow a Visitor to become a Business Customer.
  - [R2][R3][R4][R5][R6] User Services: BusinessCustomer Module through *BCaccountService*

- [G3] Allow a Private Customer to subscribe to AutomatedSOS.
  - [R3][R7] User Services: PrivateCustomer Module through *PCaccountService*

- [G4] Allow a Private Customer to review personal data.
  - [R3][R8] User Services: PrivateCustomer Module through *PCaccountService*

- [G5] Allow a Business Customer to monitor data from Data4Help.
  - [R3] User Services: BusinessCustomer Module through *BCaccountService*
  - [R9][R10][R11] Request Services: Request Handler through *ManageRequest*

- [G6] Allow a Business Customer to request data from Data4Help.
  - [R3] User Services: BusinessCustomer Module through *BCaccountService*
  - [R12][R13] Request Services: Request Handler through *ManageRequest*

- [G7] Allow a Private Customer to share his real time position and health status by a Business Customer.
  - [R3][R8] User Services: PrivateCustomer Module through *PCaccountService*
  - [R14] User Services: BusinessCustomer Module
  - [R15][R16] Request Services: Incoming Request Handler through *ManageIncomingRequest*

- [G8] Allow a Business Customer to subscribe to a data source like a specific PC or a geographical area.
  - [R3] User Services: BusinessCustomer Module through *BCaccountService*
  - [R12][R17][R18][R19] Request Services: Request Handler through *ManageRequest*

- [G9] Allow a PC in serious health conditions to receive an ambulance in the shortest possible time.
  - [R3] User Services: PrivateCustomer Module through *PCaccountService*
  - [R20][R21] AutomatedSOS (in the Mobile App)

- [G10] Allow a System Manager to do operations of system maintenance.
  - [R22][R23][R24] User Services: SystemManager Module through ManageAccount

5 Implementation, integration and test plan

In this section it will be described the most efficient way to implement, integrate and test Data4Help and AutomatedSOS. We assume that before the start of the developing process an instance of MySQL will be available, with already loaded our ER schema defined in.
Following an MVC pattern bottom-up approach to this implementation problem, the model should be implemented first, shortly followed by the controller and application logic, including the client-side logic (AutomatedSOS), finally network connectivity and view will be the last parts to be completed this to simply the process of unit testing. This pattern should be applied to each subsystem, following the proposed timeline.

Integration between modules should be implemented and tested as soon as they are complete.

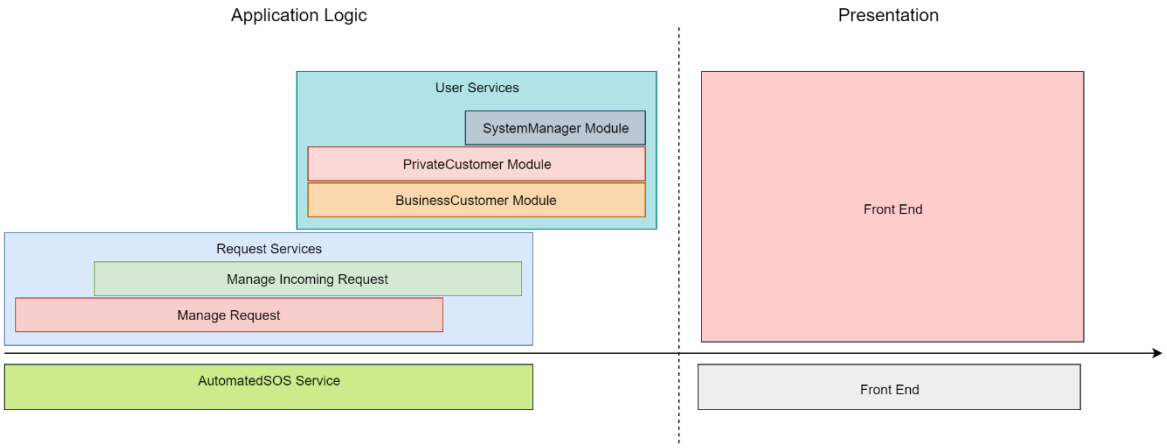


Fig. 11: Development timeline.

5.1 Request Services

The Manage Request Module will be the first implemented, since this will be the most challenging part of Data4Help from an algorithmic prospective. Giving it maximum priority in development we can eliminate bugs and malfunctions rapidly without worrying about all the dependencies from the other modules. Shortly after we should start to develop alongside the Manage Incoming Request module. The similar nature of these two modules makes them easily implementable in parallel.

5.2 User Services

When the request services will be mostly completed, we assume possible to shift some effort into the development of User Services subsystem. All the modules will be implemented in parallel since they all share the same underlying CRUD api layout. This will allow a more streamline and precise testing process.

5.3 Automated SOS service

This subsystem should be implemented as soon as possible, since it will serve a vital purpose to many customers and it will need to be tested in the most extensive possible way during all the development cycle.

5.4 Testing

The testing will be performed with the paradigm of the Test-Driven Development, that consist in writing all the tests before their relative code, giving to the developers an already made benchmark to test their code as soon as it's produced, helping immediately to find bugs and errors. The TDD is described in Figure 11.

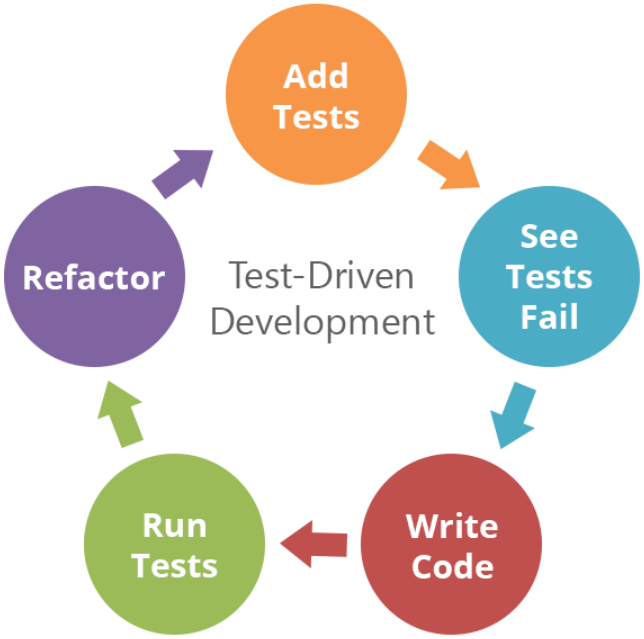


Fig. 12: Test Driven Development Flow chart.

The server-side code unit testing will be performed with the framework mocha, while on the client jUnit will be used.

6 Effort spent

- Peresson Tommaso: 30h
- Ziffer Giacomo: 30h

7 References

- RASD document previously delivered