POLITECNICO
MILANO 1863

TrackMe
Software Engineering 2 Project
ITD Document

Stefano Martina, Alessandro Nichelini, Francesco Peressini

A.Y. 2018/2019
Version 1.0.0

January 13, 2019

Contents

1 Introduction

1.1 Scope of the document

The Implementation and Test Document (ITD) for Data4Help describes the features and characteristics of the developed application, the adopted frameworks and testing technique, the structure of the source code and the installation instructions for the final release.

1.2 Definitions, Acronyms, Abbreviations

1.2.1 Definitions

- RESTful API: API that follow the REST paradigm;
- Third-parties: external organisations.

1.2.2 Acronyms

- API: Application Programming Interface;
- DB: Database;
- DD: Design Document;
- CRUD: Create, Read, Update, Delete;
- RASD: Requirement Analysis Specification Document;
- JSON: JavaScript Object Notation.

1.2.3 Abbreviations

- [Gn]: n-th goal
- [Rn]: n-th functional requirement

1.3 Revision history

- 1.0.0 - Initial Versione (13/01/2019)

1.4 Reference documents

- RASD document previously delivered
- DD document previously delivered

2 Functionality implemented

The team has developed and implemented the main functionalities of the core module Data4Help and Automated SOS. In details, the developed framework has fulfilled the following requirements described in the RASD Document previously delivered:

- [R1] Users can create an account with credentials;
- [R3] Users can log manually or automatically their data;
- [R4] Users have to be able to accept/deny access to single data access request;
- [R5] Users have to be able to see current data policies and change them;
- [R6] The machine has to be able to read health and position data;
- [R7] The machine has to be able to recognise below threshold parameters;
- [R8] The machine has to be able to communicate with third parties;
- [R9]* The machine has to be able to recognise data fragmentation level;
- [R10] The machine has to be able to store users' data.

* Requirement number 9 is only partially satisfied in this first release of the project: third-parties are able the make group-requests and retrive data correctly but there's no kind of control on the provided data (group data are supplied divided by individuals that are identified by a unique code from which it is impossible to trace back to the real user); this is because of the lack of an adequate data-set to test this feature.
We've also decided not to implement the functionality regarding requirement number 2 "*Credentials can be retrievable also if forgotten/lost*" to avoid to build an entire email infrastructure.

3 Adopted frameworks

For the backend: Flask framework has been adopted. Flask is a web development framework that let building easy and light web environments. It has been used to both build the API infrastructure and the third-parties' web interface.

3.1 Adopted programming language

The project has been developed using two programming languages:

- iOS application has been developed in the "new" programming language provided by Apple: Swift (target version: 4.2).
- Backend software has been developed using Python (target version: 3.6)

The team has chosen Swift over Objective-C because of the effort Apple is putting in its development. Moreover Swift has been thought to work with iOS in a more specific way than Objective-C.
Python has been chosen as the programming language for the backend because most of the team members has previous proficiency in using it.

3.2 Middleware adopted

Since the communication stack is based on a RESTful API system, the project doesn't use any further middleware technology.

3.3 Libraries

For Apple Swift in iOS the following external libraries has been adopted:

- Alamofire by Alamofire (`https://github.com/Alamofire/Alamofire`) to better handle web request client side;
- SwiftyJSON by SwiftyJSON (`https://github.com/SwiftyJSON/SwiftyJSON`) to better handle JSON data in Swift.

For the backend, the following Python libraries has been adopted:

- Flask: the main library of the previously described framework (see: `http://flask.pocoo.org`);
- Flask HTTP Auth: an extension of Flask framework to implement basic HTTP auth directly in the same environments (see: `https://flask-httpauth.readthedocs.io/en/latest/`);
- MySQLConnector: this library is necessary to handle communication with a SQL database;
- The following standard libraries were also used:

- Python-Sys;
- Python-Secret;
- Python-PPrint;
- Python-Collections;
- Python-Json.

3.4 API used

The system only uses the in-house built APIs.

4 Source code structure

4.1 Python Backend

4.1.1 File and class index

Python backend is composed by three files:

- *API.py*;
- *DbHandler.py*;
- *WebApp.py.*

4.1.2 API.py

This file handles the web-server, useful to build and maintain the API endpoints that serve both Data4Help app and third-parties. It has to be directly executed. It basically defines all the URL endpoints and each associated action
Decorator "*@auth.login required*" states that each marked method is called if and only if the incoming HTTP connection has been authenticated. Authentication method follows rules of Flash-HTTPBasicAuthLibrary as described here: `https://flask-basicauth.readthedocs.io/en/latest/`.

4.1.3 DbHandler.py

This file handles all the necessary operations to connect and submit queries to the data layer. The class *ConnectionPool* has been assigned with the task to generate a new connection to the server each time a method needs it.
DbHandler class is composed of two static and, most important, private methods: *send(query, values)* and *get(query, values, multiple lines)*. This two methods are made in order to fulfil connection requirements of the other methods and to hide connection and cursor handling to them.

4.1.4 WebApp.py

This file handles the web-server, useful to build the web app that serves third-parties for, in example, registration purpose. The structure is very similar to the one of *API.py* since they are both based on Flask framework's rules.

4.2 iOS frontend

iOS app's code is meanly composed by UIView and UIViewController. Here follows the description of the ControllerLogic classes that mainly handle communications with iOS native frameworks.

4.2.1 Global.swift

This file contains a class with the same name that declares all global parameters needed by the application.

4.2.2 HTTPManager.swift

This file contains a class with the same name that handles all communications through the network.

4.2.3 HealthKitManager.swift

This file contains two classes, *AutomatedSOS* that handles heartbeats monitoring in order to implement Automated SOS functionalities, and *HealthkitManager* that handles all the setup operations needed to let the app communicates with Apple HealthKit framework.

4.2.4 NotificationCenter.swift

NotificationCenter contains a class with the same name which handles all the setup operations needed to trigger and dispatch local notifications for the iOS application.

4.3 Component mapping

All the names of the components refer to the ones associated with the components presented and described in DD document.

4.3.1 DatabaseLink Manager

Database link manager is basically mapped on code of file *DbHandler.py*.

4.3.2 HealthSharing Manager

4.3.2.1 Access Policy Manager Module

Access Policy Manager Module is composed by the following set of methods implemented in *API.py*:

- ***user_subscription()***: that handles responses to HTTP GET requests for subscriptions;
- ***update_subscription_status()***: that handles responses to HTTP PUT requests for updating subscriptions status;
- ***subscribe()***: that handles subscription operations for third parties.

4.3.2.2 Data Manager Module

Data Manager Module is composed by the following set of methods implemented in *API.py*

- ***heart()***: that handles heart data storing operations;
- ***get_heart_rate_by_user()***: that handles heart data retrieval operations;
- ***user_location()***: that handles location data storing operations;
- ***get_user_location()***: that handles location data retrieval operations;

4.3.2.3 Data elaboration module

Data elaboration module can be mapped in the following set of methods of *API.py*, which functionalities are self-explained:

- ***groups_heart_rate_by_birth_place()***;
- ***groups_heart_rate_by_year_of_birth()***;
- ***groups_location_by_birth_place()***;
- ***groups_location_by_year_of_birth()***.

4.3.3 SOS Manager

SOS manager has been implemented client-side in class ***AutomatedSOS*** of file *HealthKitManager.swift*

4.3.4 Access Manager

4.3.4.1 Login Module

Login Module is composed by the following set of methods implemented in *API.py*:

- ***login()***: that handles login operations for users;
- ***user_register()***: that handles registration process for users.

5 Testing

Information about testing procedures are contained in section 6.3 of the Design Document previously delivered.
Instead, in this paragraph we describe the main test cases that we have considered and their outcome.

5.1 Test concerning single users

Of particular importance are the tests made on the location insertion and retrieval and heart rate insertion and retrieval. For testing these two features of Data4Help application we simulate the registration of a new user in the database; after checking the HTTP POST request has been made correctly, we proceed to insert some location/heart rate information for the user just registered in the system. Lastly, with an HTTP GET request we retrive the data just inserted in the database and confront them with the information initially inserted.

5.2 Test concerning third-parties

Concerning the testing cases regarding third-parties, of particular interest are the tests made on location and heart rate retrieval based on parameters provided to the endpoint. For testing these features, in the first instance we simulate the registration of a new third party and of a new user in the database; after this operation, we insert some location/heart rate information for the user previously created and than we simulate the permission of subscription request made by the third party and direct to the user.
After the user acceptance, the third-party make an HTTP GET request to the specified endpoint providing its username, its secret and the parameters of interest. The test phase is performed by confronting the information retrieved by the database and the information initially inserted.

5.3 How to test the code

In order to test the code, MySQL server has to be installed on your testing machine and there has to be a running instance of it. First run API.py with "–testing" option, then execute the unit tests contained in Test.py.
You can download MySQL server from here: `https://dev.mysql.com/downloads/mysql/`.

6 Installation instructions

6.1 iOS mobile app installation instructions

Since an Apple Developer Program affiliated account is needed to distribute an iOS application, in order to run and test our application you need to clone our repository and manually compile and run the entire XCode project. A device running Mac OS 10.14 or later and XCode 9 or later is required.
The application can run on a simulator or on a real iOS device running at least iOS 11 (target Apple device: iPhone 5 or later).

Application won't work without a running backend instance, however we are going to keep alive an instance of it at `http://data4help.cloud:5000`. If you like to run your own backend instance during testing to support app activities you will be able to test them using XCode embedded simulator: as matter of fact the application tries to connect to localhost in first instance.

6.2 Python backend installation instructions

Python backend code should run on each Python 3.x distribution. More in details, the target version in which the whole project has been developed is Python 3.6.
Once installed Python, it's recommended to build a Python virtual environment to run the code (you can check the official Python documentation here: `https://packaging.python.org/guides/installing-using-pip-and-virtualenv/`).
The following libraries have to be installed (Python-pip is the preferred method to install these packages):

- Flask (target version: 1.0.2);
- Flask-HTTPAuth (target version: 3.2.4);
- mysql-connector (target version: 2.1.6);
- mysql-connector-python-rf (target version: 2.2.2).

Once the virtual environment has been built and activated, and the repository has been closed, it's enough to call the python interpret from a console to run the backend server: the main file is *Api.py* located in *Path/to/repository/Server/*

7 Effort spent

Time spent on ITD document: 12.00h

- Stefano Martina: 4.00h
- Alessandro Nichelini: 4.00h
- Francesco Peressini: 4.00h

Time spent on Data4Help code: 150.00h

- Stefano Martina: 50.00h
- Alessandro Nichelini: 50.00h
- Francesco Peressini: 50.00h