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 11, 2019

Contents

1 Introduction and scope

1.1 Introduction

1.2 scope

2 Functionality implemented

The team has developed and implemented the main functionalities of the core Data4help module and Automated SOS. In details the developed framework has fulfilled the following requirements described in RASD Document:

- (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 to 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.
- (R10) The machine has to be able to store users' data.

We excluded requests of anonymous data because of the lack of an adequate data set to test this feature (Requirement number 9). We've also decided not to implemented 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 the building of 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 language.

- iOS application has been developed in the "new" programming language by Apple: Swift (target version: Swift 4.2).

- Backend software has been developed using Python (target version: Python 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 HTTP REST 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 Flask.
- Flask HTTP Auth: an extension of Flask framework to implemented basic HTTP auth directly in the same environments.
- 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 project has not used external API than the ones directly developed in-house.

4 Source code structure

4.1 Python Backend

4.1.1 File and class index

Python backend is composed by three file:

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

4.1.2 API.py

It 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 define all the URL endpoint and the action associated with enabled HTTP actions.
Decorator "*@auth.login required*" states that each following 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

It handles all the necessary operations to connect and submit query 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 need it. *DbHandler* class is composed of two static and most important private method: *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 connections and cursor handling to them.

4.1.4 WebApp.py

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

4.2 iOS frontend

4.3 Component mapping

All components names 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 request for subscriptions.
- ***update_subscription_status()***: that handles responses to HTTP PUT request for updating subscriptions status.
- ***subscribe()***: that handles subscriptions 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

As described in section 2 of this document we only implemented aggregated data requests. The whole component 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

4.3.3.1 Anomaly Detection Module

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 operation for users.
- ***user_register()***: that handles registration process for users.

5 Testing

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 ore later and XCode 9 or later is required.
The application can run on a simulator or on a real iOS device running iOS 11 or later (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/*