POLITECNICO

MILANO 1863

Air Quality

Software Design Document

Antonino Natoli, Michele Pilia

June 11, 2020

Summary

1 Introduction

1.1 Goals

This project was realized as the conclusion of the course of "Design and Implementation of Mobile Application" at Politecnico di Milano and this document has the purpose to help understanding all the details of the application and to justify all the taken choices.
Air Quality is an application realized in collaboration with the research group lead by the Professor Fulvio Re Cecconi, in order to build an interface that works with an Arduino board to measure some specific values related to the quality of the air. The application features also an integration of the measured data with the pollutants data taken by the ARPA stations, in the city of Milan. The application should be easy to use, with an intuitive UI. We aimed at giving the users a perspective of the quality of air in the near period, that means the data that can be displayed are the ones recently measured.
In this way many users can contribute in mapping the quality of air, and the results can be seen by all the users of the application.

1.2 Platform

We decided to use React Native since the research group use both Android and iOS as us. We already had some experience together with Javascript, however we hadn't used react-native, so we spent time becoming in learning and becoming familiar with it. The application works correctly both on iOS and Android.

1.3 Stakeholders

The main stakeholder is the Professor Re Cecconi, who should use this application in the future, to complete his measurements and researches, but it may be useful also for other people that could own an Arduino board (or any kind of physical device with the sensors needed) or for general users that want to examine the previously data, even if measured and stored by other people.
For these reasons we tried to build a UI as intuitive as possible, but we introduced also the possibility to edit some specific parameters, aiming at more experienced users.

2. General description

2.1 Intention

Air Quality is a cross-platform application, supported on Android and iOS devices, that allows the user to store in a database the values of some air parameters coming from sensors connected to a physical device, that could be an Arduino board, or to consult the parameters stored in the past.

The project is composed by the following main components:

- Client, the application running on the smartphone, with the task of providing the User Interface to deal with the system, and to query the servers and the board to handle all the data;

- Server, composed by the database containing all the data related to users and air parameters;

- Arduino board, connected to sensors for CO_2, $PM_{0.4}$, $PM_{0.5}$, PM_1, $PM_{2.5}$, PM_{10}, TVOCs, humidity, temperature, pressure, and to a WiFi module needed to send data to the client;

- Arduino simulation server, to simulate data related to the board.

2.2 Functionalities

Here we will list all the functionalities implemented in the app, trying to follow a possible order of interaction:

- Login: it's the first screen, displayed after the app has finished loading. It allows the user to insert the credentials and log into the system;

- Signup: if the user is not registered, the form in this screen is used to create a new user for further authentications;

- Home: it's the screen shown after the authentication. It is shown only once and presents the button to navigate to the other screens composing the application. It features also the descriptions of the application functionalities;

- Real time data: allows to see the user's position and the data taken in real time from the connected physical device acting as an environmental sensor;

- Historical: a screen that shows various statistics about the measured data by different users and allows the integration and comparison of these data with the ones that were taken in the previous days by ARPA stations located in Milan. The statistics of the air quality data measured with the physical device can can be taken from a period that goes from two weeks before the current day up to the current day, in fact we are aiming at giving a recent perspective of the quality of air and therefore the time period from which users can retrieve data shouldn't be long;

- Settings: it's the page that shows all the logged user's informations and allows the change of password, of the sensor device's URL and of the time delay between each data read.

2.3 Features

The application has to be simple to use at the early interactions, but, since it's aimed to experienced users, it should allow a certain degree of customization and in-depth analysis.

2.3.1 Functional Requirements

General requirements

- Since the app has to be used mostly inside the Politecnico, we chose English language, but data from stations in Milan are transmitted in Italian by ARPA;

- The application must work only after the authentication of a user, in order to keep track of all the changes;

- The user should be able to see data from the device in real time, and to store them in a database;

- The user must have the possibility to see the details of the previously stored data, and to see the statistics of the measured data close to each one of the ARPA stations located around Milan, filtering them by period;

- The application must be extensible, since a third part can build its own sensor, sending data in the format requested by the app;

- The application should be connected to different devices, one at a time, simply by changing the URL to receive data from;

- If a board is not available, it should be possible to simulate data;

- After the user has logged into the system, a button to log out must always be present;

- The user must be able to change its password;

- The application should have a user friendly and clean UI and it should be easy to use, without allowing the user to change too many parameters and giving him a general overview of the quality of air, for a certain geographic region.

Login requirements

- Once the app is loaded, a login form must be shown to allow the user's authentication;

- A link to the signup form must be present, to be used by a not yet registered user;

- If the credentials are wrong, an alert message should be displayed;

- The input field for the password must have the "hidden" font, but if the user wants to, it can be displayed clear by pressing an icon.

Signup requirements

- In the registration form the user must insert first name, last name, birthday, gender and password (hidden format, as mentioned above);

- A complexity indicator should be shown while choosing the password;

- If the password and the confirm password do not match, an alert will be shown;

- If the signup process goes successfully, the user should be redirected to the login form, else a proper alert message is displayed (for example in case the email is already taken)

Home requirements

- Once the user is logged in, the Home page should display a menu with all the page to which is possible to navigate. A brief description for each options is present. Home page should be displayed only once, after the user has logged in.

Real Time Data requirements

- A map must be displayed, showing the user's actual position. A follow user option should be enabled.

- A switch should be present, in order to enable the storing of actual data;

- Data caught by the device should be shown in this view;

- By default data are read every 5 seconds;

- In case data aren't received a proper alert message is shown.

Historical requirements

- The user can set the start date, and the end date of the period between which to compute the statistics. It's allowed a margin of two weeks between the two dates;

- A map should be displayed, with a marker (blue color) for each one of the currently active ARPA stations located in Milan, and for each cluster of the daily measured data (green color) by the users;

- Measures that are taken within a distance of 200 meters are grouped together forming a cluster, since we are aiming at giving a more user friendly visualization. For each cluster it is displayed the mean for the selected period. We decided to use the mean since it is a commonly used and easily understandable quantity, from any kind of possible user;

- If the user presses one of the marker corresponding to an ARPA station, statistics for that station will be displayed. It is computed the mean for the received data. If instead the user presses one of the marker corresponding to data measured by other users are shown the mean values for the proper data;

- When navigating on this page, on the map are automatically shown the markers corresponding to ARPA stations in Milan and to the taken measures for the current day.

Settings requirements

- All the user informations must be shown (except for the password);

- The user must be able to change its password, with the same requirements described for the signup process;

- The user should be able to set the URL of the physical device that reads data;

- A slider must be present to set the time delay between successive data reads in the "Real Time Data" view.

2.3.2 Non-Functional Requirements

- Portability: the app must be used on both Android and iOS smartphone;

- Extensibility: it is possible to use any kind of device for measuring data, both physical or virtual. The only requirement is that of sending data through WiFi with a specific format, that will be mentioned after in this document.

- Maintainability: the code should be easily-readable;

- Reliability: the system must be safe and not broken by anyone;

- Efficiency: the app should use the less resources as possible.

3. Data Structure

3.1 Internal Organization

The most important portion of data is stored on the server, therefore the needed data are obtained through fetch call and stored temporarily in local and global variables just to be showed in the related views.

3.2 External DB Services

All the data referred to users and measurements are stored on a DB Server offered by Heroku, located at the following URL: https://polimi-dima-server.herokuapp.com/.

The user data are associated to an email address, and contain also the birthday, the gender and the name. When the user logs in, the server will return a token that is carried on along all the execution in order to grant the authorization to store and view the data.

The way to access the DB is through fetch request sent to the server, with the correct set of parameters, as specified in the docs of the server.
The connection with the server is encrypted via HTTPS.

4. Architecture Design

4.1 Client

The client is divided into graphical part and logical part. The graphical part is realized with JavaScript and JSX (with style written with the CSS syntax), while the logical part with all the functions is totally composed by JavaScript code.

4.2 Server

There are two main servers. The first one is the server that sends the measured data to the application, that should be implemented inside the device used for taking measures. Then the other server is the Heroku one, it interfaces with a PostgreSQL database and gives the application access via an API endpoint.

4.3 Arduino

The code stored on the board is written in C++, with the structure required by Arduino. The board interfaces with an environmental sensor that measures temperature, humidity, tvocs, co2, altitude and pressure. The sensors for measuring PM particles is not supported yet. The arduino device is connected to a WiFi shield, that enables it sending data to our application using HTTP protocol.
In the arduino file it is possible to modify the SSID and password of the WiFi access point for the WiFi module.

Arduino code updates the value of the global variables (each one of them corresponds to a specific measure) every 2 seconds. It then launches a server that answers to the HTTP requests sent by the application. The server sends the measured data from the sensors with the following format, in which each value should be separated by a semicolon:

"temperature_value;humidity_value;pressure_value;altitude_value;tvocs_value;co2_value; pm0.5_value;pm1_value;pm2.5_value;pm4_value;pm10_value"

The application provides an interface to any possible sensor, so you can implement your own sensor using any technology, but in order to interface it with the application you must respect the format above when sending data.

For testing the application we used a virtual device (node .js server) that simulates a generic physical device.
The code of both the arduino device and of the simulator are available in the github repository.

4.4 MVC Paradigm

If we want to indicate a separation of the components according to the MVC paradigm we can state that:

- The controller lies both on the client and the server since a lot of controlling functions are implemented in the server part to manage for instance the authentication process, but also in the client there is an important amount of logical functions, regarding the handling of the air parameters;

- The model is updated by the controller and stored on the database server, but is partially replicated locally on the client when some parameter has to be kept along all the execution;

- The view is obviously managed in the client side, taking the values from the model and showing them accordingly.

4.5 Project organization
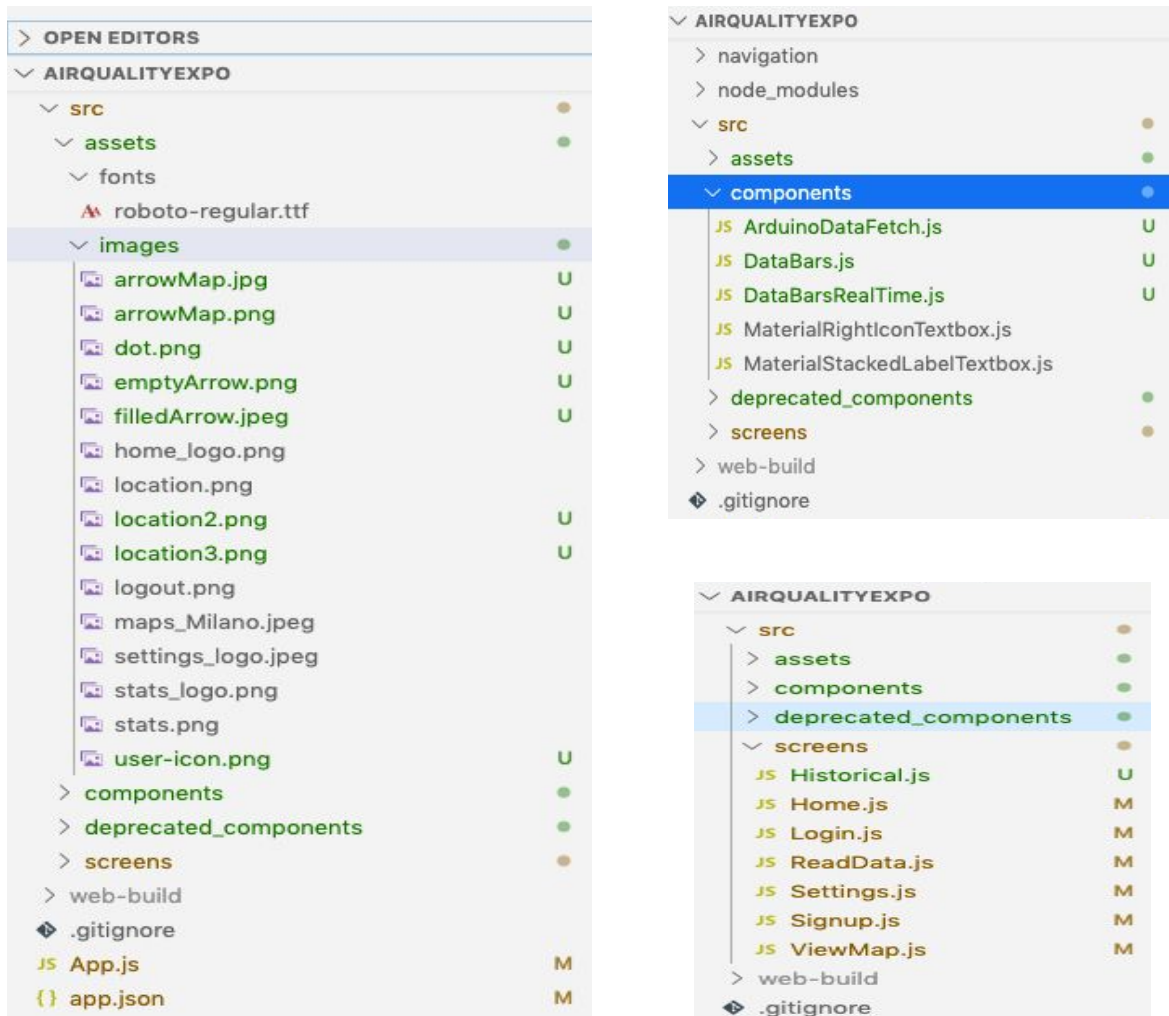
The project contains a folder named "assets", containing the resources used by the applications, a folder containing the custom components, used inside the view, and a "screens" folder, containing all the screens that the user can access with the navigation.
It is also present a deprecated components folder, that contains component we decided not using anymore or that we used as a starting point for further developed ones.
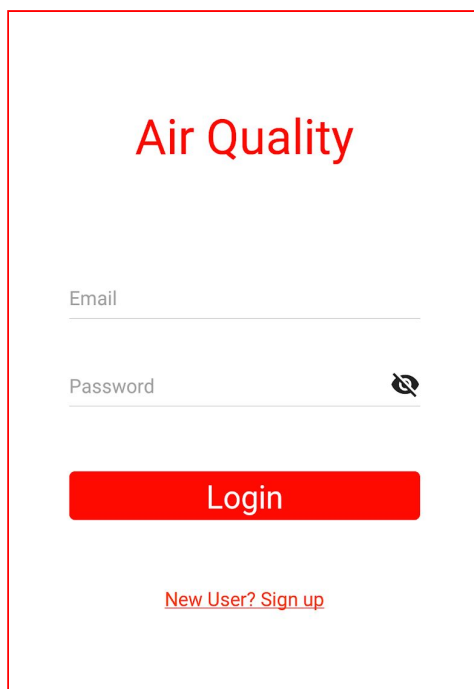
5. User Interface

5.1 Splash

The Splash screen is displayed when the app is loading the required resources.

Air Quality

5.2 Login

The login screen is composed by a form in which the users inserts the credentials to log into the system. If a mistake is made, an alert will be shown.
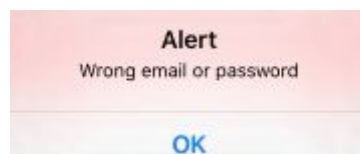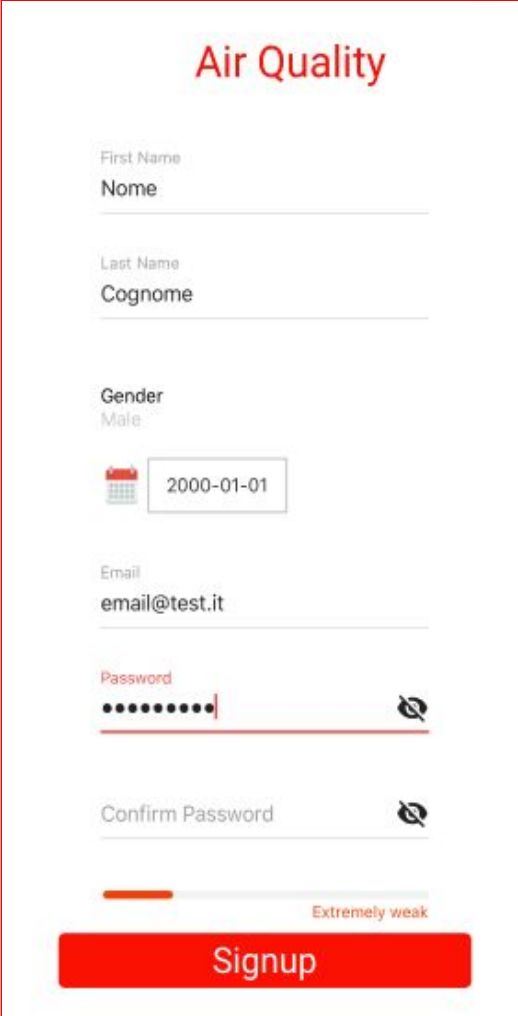
5.3 Sign Up

From the link under the login form is possible to access the signup form, in order to create a new user to log in with in the future sessions. Under the password field a bar is present to give a feedback about its complexity.
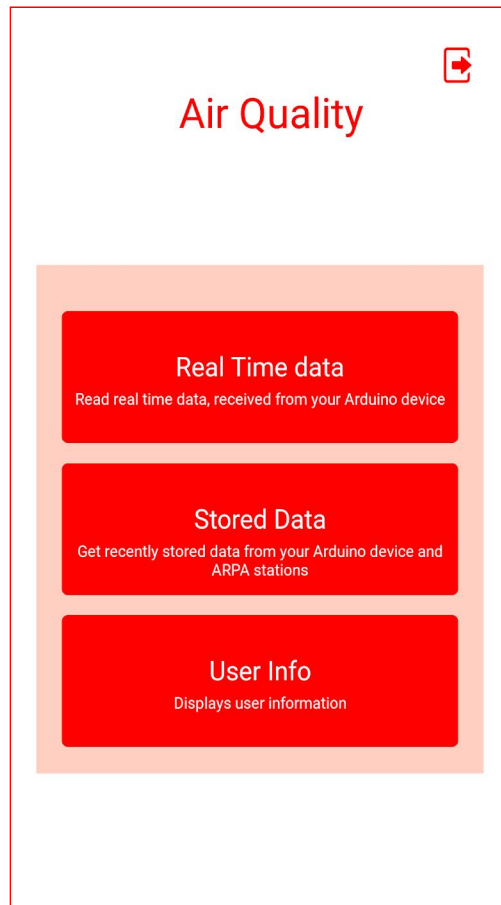
5.4 Home

This screen is shown only once, after a successful login. It features the navigation options

5.5 Real Time Data

The Real Time Data screen shows the data caught in that specific moment by the connected device, if the URL is set accordingly.
If the "Store Data" switch is ON, the app will start saving these data on the database server.
If the app cannot receive any data from the specified URL an alert will be shown.
For each measure it is displayed a progress bar with two labels, one for the minimum value and one for the maximum value. We've chosen the interval of possible values accurately for each measure. In this way even a non experienced user can easily have a feedback whether the values he is measuring (in particular the ones referring to the pollutants) are above the mean values and if they are near the maximum values it means that there can be a risk for the health.
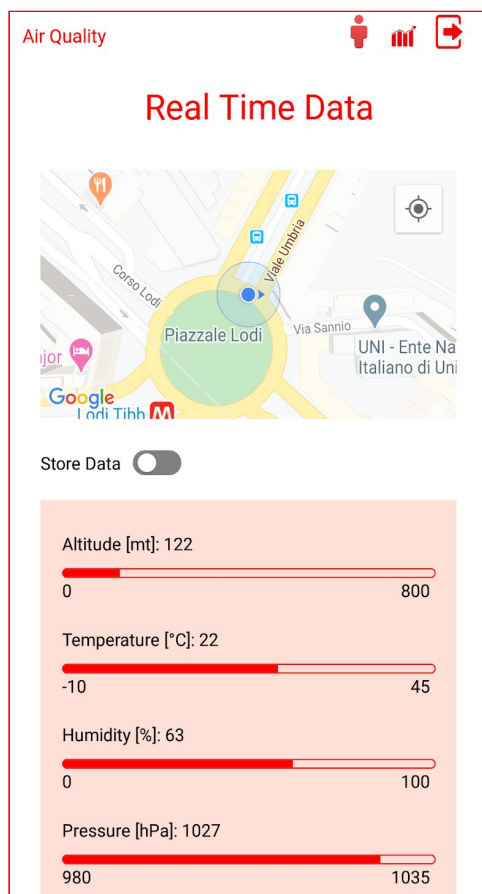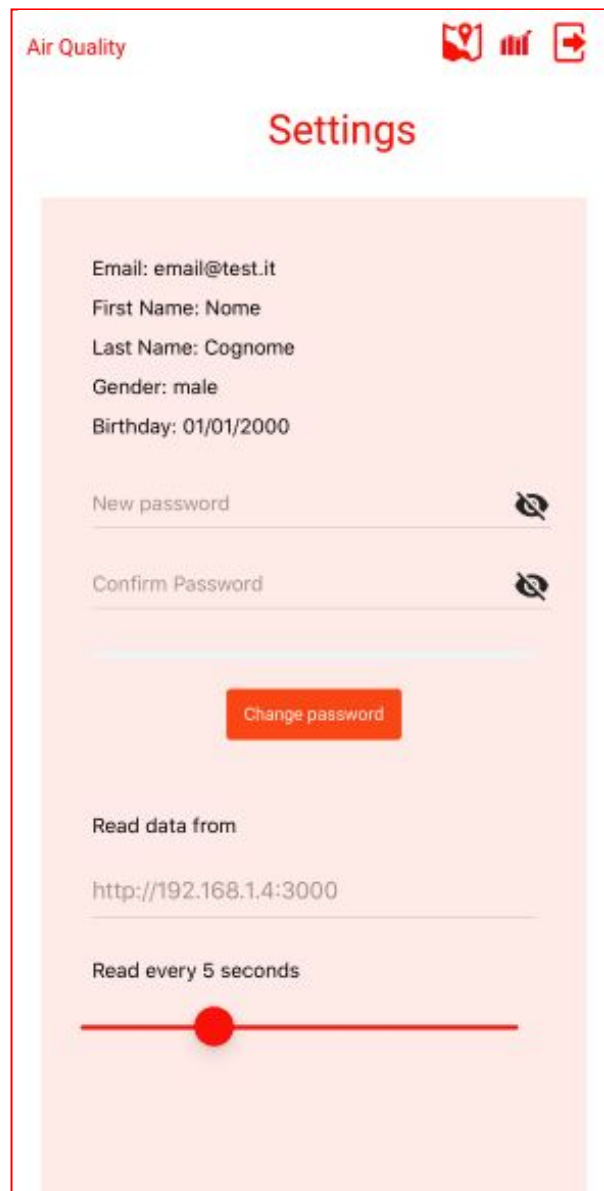
5.6 Historical

In this screen the user can see the average of the data measurements stored by other users, selecting a period over which the average has to be computed. By default when navigating on this screen are shown the active ARPA stations in Milan (blue markers), together with the clusters of measured data (green markers) for the current day. Each time data measured by the users are retrieved they are grouped together in clusters that have a radius of 200 meters: we are supposing that the quality of air for these regions is homogeneous.
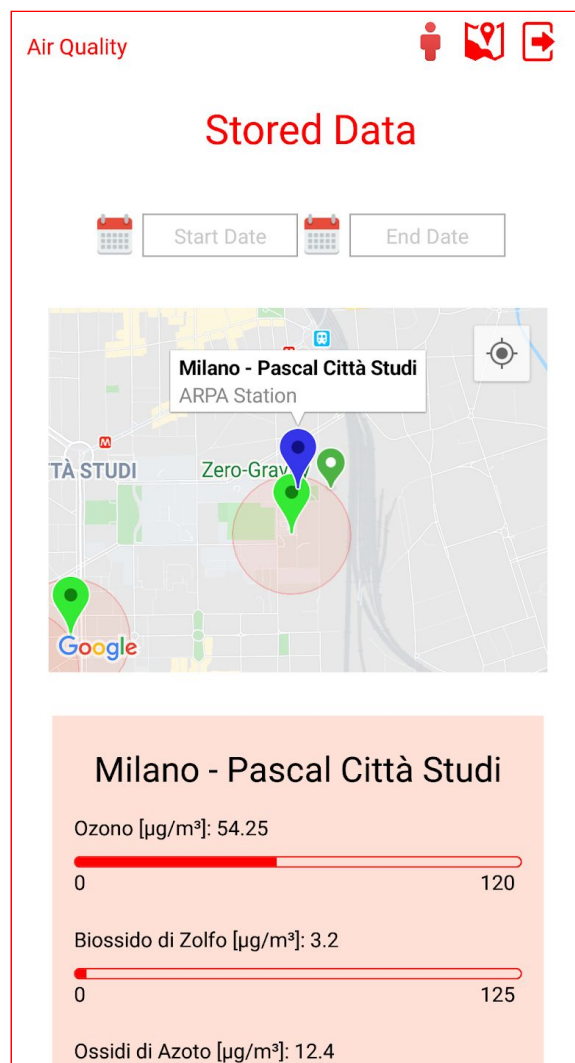
5.7 Settings

The Settings screen shows the user informations, as well as the form to change the current password, an input field to set the URL of the device that should be used for measuring data, and a slider to set the amount of seconds between each read from Arduino.

5.8 Upper Navigation Bar

Each screen features some icons, by clicking them it is possible to navigate to the proper pages, here are listed all the icons with their meaning

By pressing on this icon it is possible to navigate to the screen "Settings"

By pressing on this icon it is possible to navigate to the screen "Real Time Data"

By pressing this icon it is possible to navigate to the screen "Historical"

By pressing this icon it is possible to logout from the application

6. External Services

We decided to use several external services to realize the application, because some of them were necessary to achieve the final result, while other revealed to be very useful to improve the final user experience.

6.1 Heroku

Heroku has been used to build our DBMS, in order to store data in a persistent way just by using HTTPS fetch requests.

6.2 Maps

Since we needed to show informations about user's location, we needed to add a map view to the application, thus, through the use of "react-native-maps" package the user can deal with iOS Maps or Google Maps (Android) interface to see its position.

6.3 External Libraries

Several external libraries have been installed to achieve specific goals:

- expo-permissions to manage the authorizations for user's location;
- react-native-progress to show data values as progress bars;
- react-native-maps, as cited before;
- react-native-hide-show-password-input, to insert the password in the form with the hidden font;
- react-native-picker-select, to use the picker selector in the form;
- react-native-password-strength-meter, to show the password complexity in the form;

7. UML

7.1 Class Diagram

7.1.1 Screens

7.1.2 Components

cmp

Component

ArduinoDataFetch
- + year: Date
- + month: Date
- + day: Date
- + dat: json

- + dataByDate(Str, Fnct): nul
- + addMeasure(Json): null
- + getDataByMean(): Num[]
- + getData(): Num[]

Extends

DataBars
- + okSensors: Number[]
- + maxValues: {Number 13}
- + minValues: {Number 13}

- + normalizeOutp(Num, Num, Num): Num
- + getViewSensorsByStation(Num): Comp
- + computeMeanForSensor(Num): Num
- + getSensorMeasur(Num): Component
- + render(): null

Extends

DataBarsRealTime
- + minValues: {Number 13}
- + maxValues: {Number 13}
- + okSensors: Number[]

- + normalizeOutp(Num, Num
- + render(): null

Extends

7.2 Sequence Diagrams

7.2.1 User Signup

```
        ┌──────────┐   ┌──────────────┐
        │  Signup  │   │ Server Heroku│
        └──────────┘   └──────────────┘
  ┌cmp─────────────────────────────────────┐
  │        ○                                │
  │       ╱│╲                               │
  │       ╱ ╲                               │
  │      User                               │
  │        ──Insert Credentials──>          │
  │                                         │
  │        ──Submit Credentials──Submit Credentials──>
  │                                         │
  │        <----------User Created----------│
  │                                         │
  └─────────────────────────────────────────┘
```

7.2.2 User Login

```
        ┌──────────┐   ┌──────────────┐
        │  Login   │   │ Server Heroku│
        └──────────┘   └──────────────┘
  ┌cmp─────────────────────────────────────┐
  │        ○                                │
  │       ╱│╲                               │
  │       ╱ ╲                               │
  │      User                               │
  │        ──Insert Credentials──>          │
  │                                         │
  │        ──Submit Credentials──Submit Credentials──>
  │                                         │
  │        <----------Access Granted--------│
  │                                         │
  └─────────────────────────────────────────┘
```
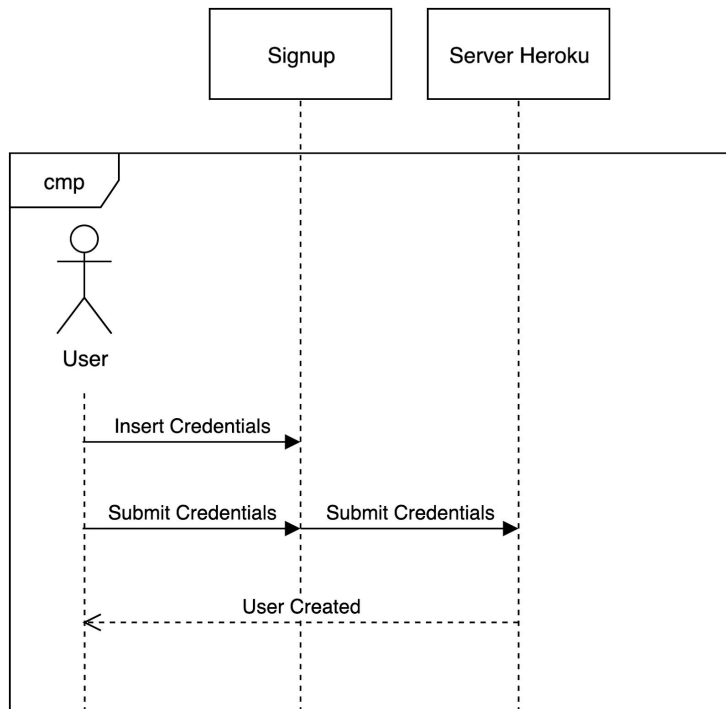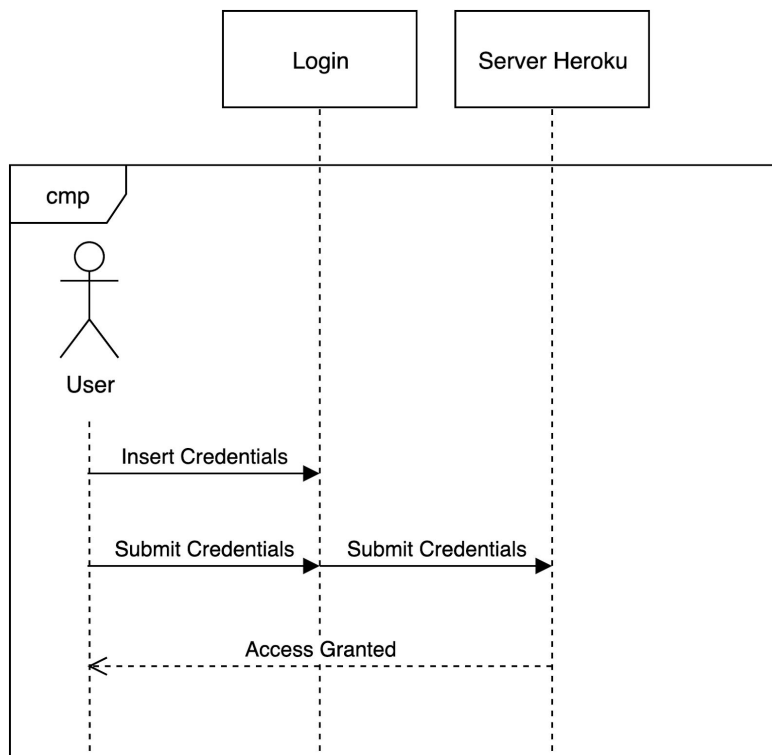
7.2.3 Change Password
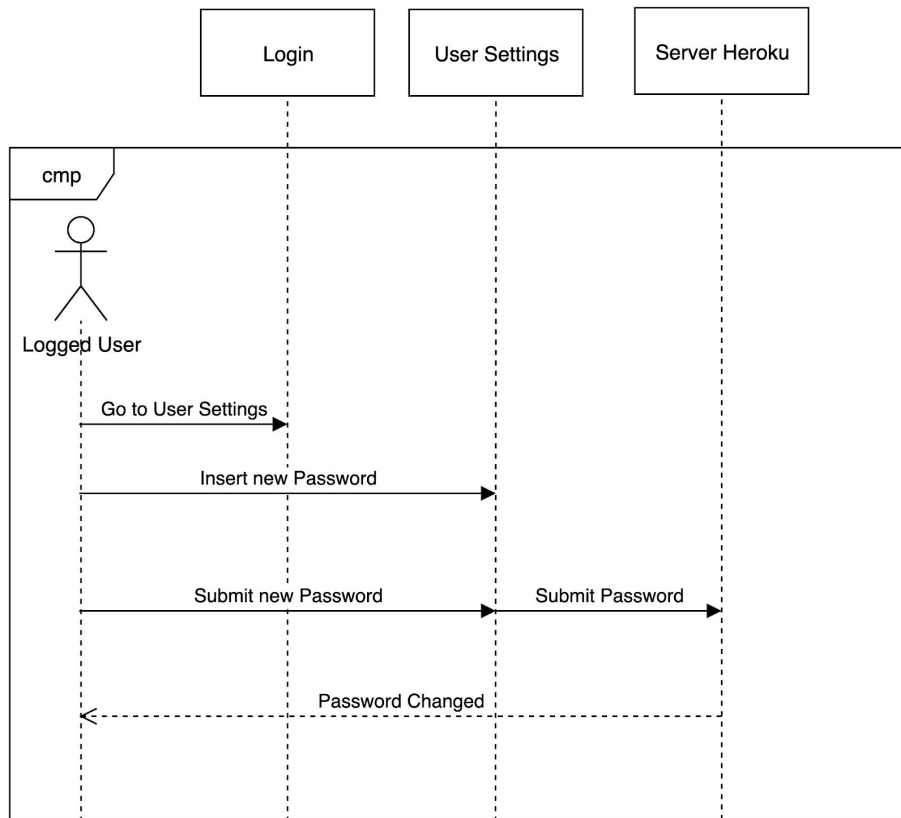
7.2.4 View Real Time Data

7.2.5 View Stats

7.3 Integration Diagram
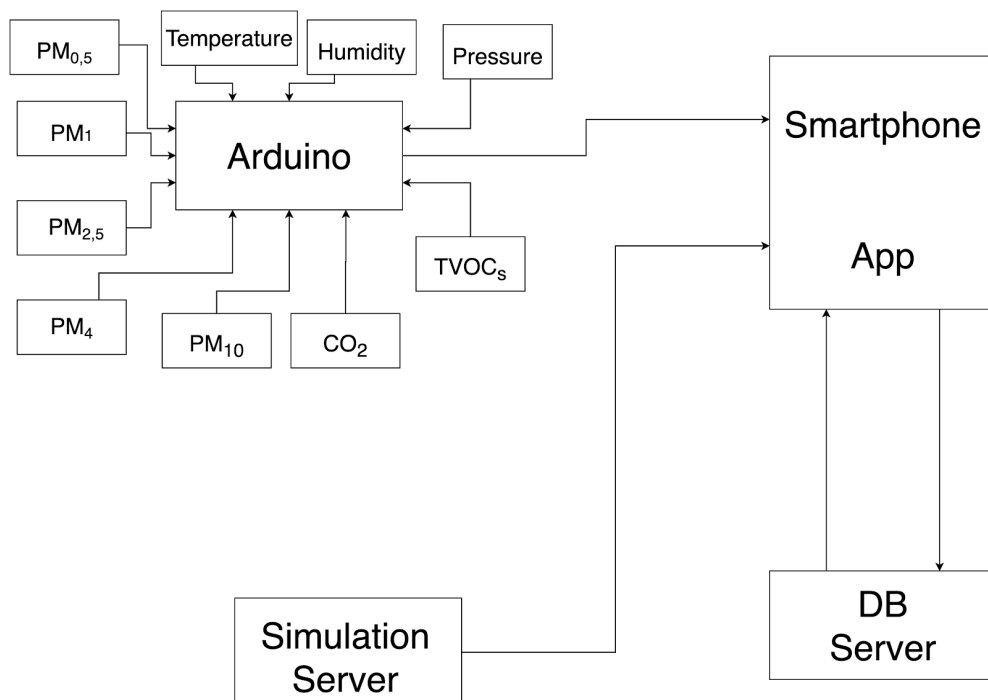
8. Tests

Test Case	Correct Signup
Goal	Save a new user into the database
Input	First Name, Last Name, Gender, Birthday, Email and Password inside a Signup form
Expected Outcome	Navigate to the Login Screen
Actual Outcome	CORRECT: the Login page is correctly loaded

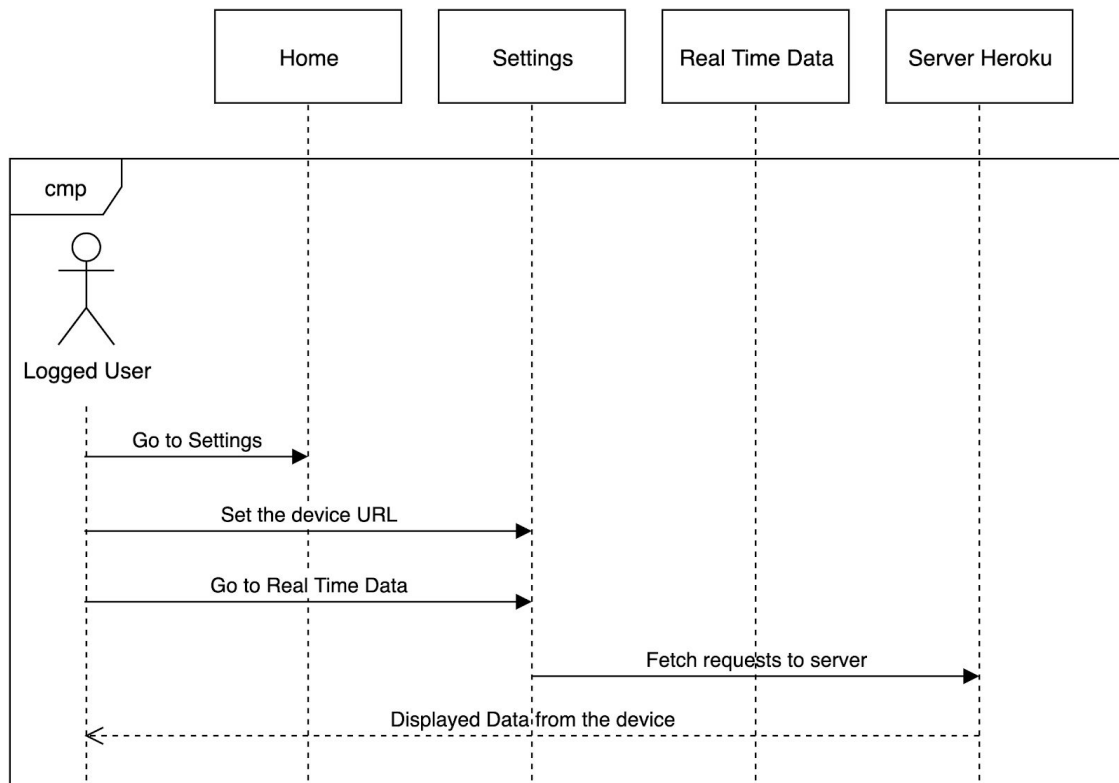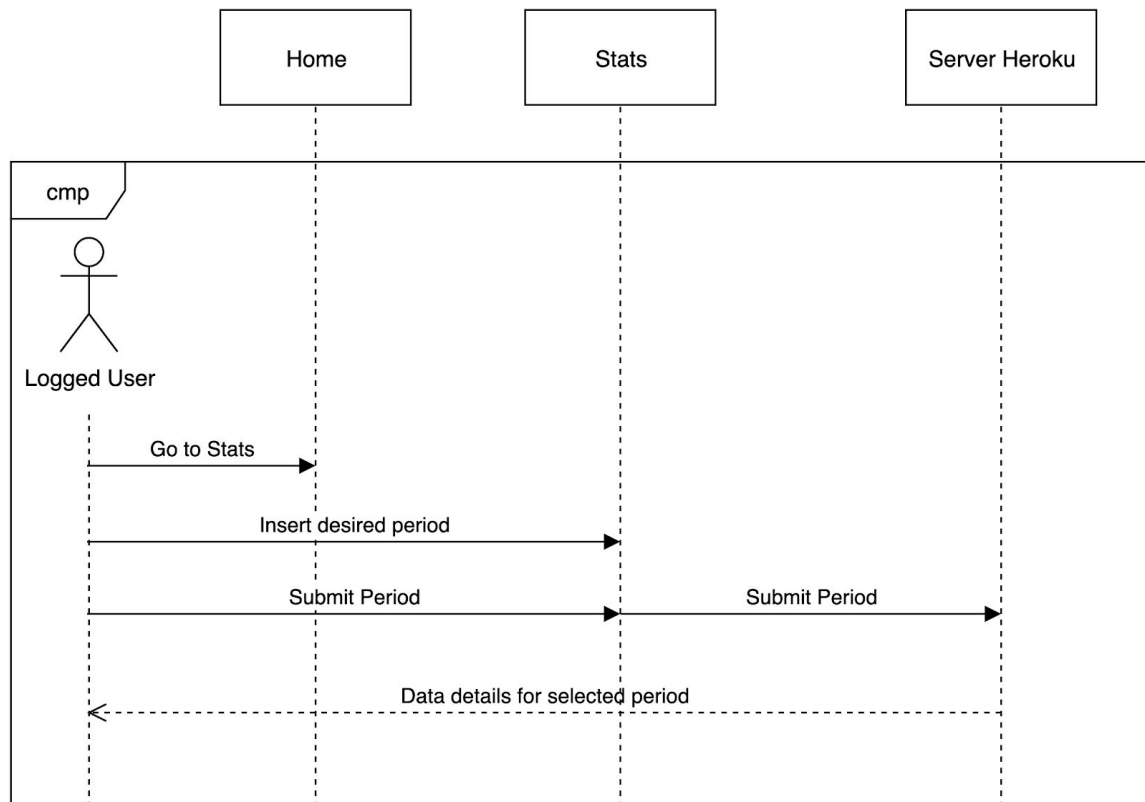| Test Case	Wrong Sign Up
Goal	Avoid creating an already existing user
Input	First Name, Last Name, Gender, Birthday, Email (already existing) and Password inside a Signup form
Expected Outcome	Display an alert of the wrong operation
Actual Outcome	CORRECT: an alert is shown to stop the user from creating a new user with an already existing email address

Test Case	Correct Login
Goal	Get the access to the application
Input	Email and password of a registered user
Expected Outcome	Get the authorization and navigate to the "Home" screen
Actual Outcome	CORRECT: the Home page is correctly viewed

Test Case	Wrong Login
Goal	Get an alert if the user inserts wrong credentials
Input	Wrong email address or password

Expected Outcome	An alert is shown
Actual Outcome	CORRECT: A message "Wrong email or password" is shown

Test Case	Correct Logout
Goal	End the session with a logout request to the database server
Input	Pressing the right "logout" button
Expected Outcome	The user is logged out and the login view is shown
Actual Outcome	CORRECT: the Login page is correctly displayed

Test Case	Display correct user location
Goal	When the user navigates to the "Real Time Data" screen the correct position has to be marked in the map.
Input	Press the "Real time data" button
Expected Outcome	The map should show the actual position
Actual Outcome	CORRECT: the map is centered around the actual user's location

Test Case	Display user's informations
Goal	The user must have the possibility to see the informations indicated during the signup process
Input	Pressing the "settings" button
Expected Outcome	The screen must show the correct credentials
Actual Outcome	CORRECT: the email, first name, etc. are clearly visible

Test Case	Change password
Goal	The user should be able to change the

	password inside the "Settings" menu
Input	The new password and its confirmation
Expected Outcome	The system must store the new password in the database
Actual Outcome	CORRECT: the new password is now accepted in the login process, while the old one is not

Test Case	Change Wrong password
Goal	If the user insert a confirmation password that does not match the new one, an alert message must be shown
Input	A password, and a different one confirmation password
Expected Outcome	An alert message is shown
Actual Outcome	CORRECT: A message is shown to alert the user

Test Case	Change URL data
Goal	The user must be able to change the URL to take the data from, inside the settings menu
Input	The desired URL
Expected Outcome	In the read data screen, the data taken from the correct device should be shown
Actual Outcome	CORRECT: when the URL of Arduino is set, data are shown correctly; when the URL of the simulation server is set, simulation data are displayed

Test Case	Change the time delay between reads
Goal	Time delay between successive reads must be selected with a slider
Input	Selection of the time delay (x) with a slider
Expected Outcome	The data have to be updated each x seconds.
Actual Outcome	CORRECT: When 5 seconds delay is set,

	exactly 5 seconds pass between one read and the next one

Test Case	**Display ARPA station's data**
Goal	The system must show the information about the data taken by each ARPA station located in Milan
Input	Pressing the "Historical" button, then one of the marker on the map
Expected Outcome	The map in the screen must show the ARPA station in Milan, then if one of them is pressed, the respective data must be shown
Actual Outcome	CORRECT: When "Milano Pascal" is selected, data caught by that station are shown

Test Case	**Display data in a period**
Goal	The user must have the possibility to see the average of the data measured in a specific period
Input	The "start date" and the "end date" of the desired period
Expected Outcome	The screen must show the data referring to the selected period
Actual Outcome	CORRECT: if 23/05/2020 - 27/05/2020 is selected, data from that period are shown

Test Case	**Disconnect and connect the board**
Goal	If the board is disconnected while reading data and then it is connected again, it must keep reading data
Input	Disconnect the board while reading, then navigate to another page, connect the board and navigate to "Real Time Data" page
Expected Outcome	The app must keep reading data from the board
Actual Outcome	CORRECT: The app kept showing the real

	time data

Test Case	**Alert disconnected board**
Goal	If no data is coming from the specified URL an alert should be displayed
Input	Navigate to Real Time Data with no board connected
Expected Outcome	The app should alert the user that there might be no board connected
Actual Outcome	CORRECT: The alert is correctly shown

Test Case	**Back button after logout**
Goal	If the user presses the back button (Android) after the logout, the app must quit
Input	Go to the login page (at the beginning or after logout), then press the back button
Expected Outcome	The app must close
Actual Outcome	CORRECT: The app has been closed

Test Case	**Back button when logged in**
Goal	If the user presses the back button (Android) when logged in, the app should navigate to the previous screen
Input	Back button
Expected Outcome	The app must navigate to the prior page
Actual Outcome	CORRECT: The app displays the preceding screen

Test Case	**Correct visualization of Historical page**
Goal	When navigating on the page, on the map are automatically shown the markers corresponding to ARPA stations in Milan and to the taken measures for the current

	day.
Input	Navigate to Historical page
Expected Outcome	The app must show markers of ARPA stations in Milan and measures for the current day.
Actual Outcome	CORRECT: The markers and the daily measures are correctly displayed