HCI project assignment 2019-2020

Project Title: Serverless Cloud Based Air Quality Monitoring Web Application

Student: Francesco Areoluci

Credits: 9 CFUs

Abstract

The main purpose of the proposed system is to allow users to monitor air quality measurements over a certain area of interest. The application has been developed with the open data concept in mind: collected air quality data should be freely accessible and available to everyone. Thus, the system has been built as a web application that allows users to select configured devices to get the collected measurements. The developer point of view has also been taken into analysis: the application is serverless and cloud based.

1 Project overview

1.1 Application overview

The proposed application allows users to evaluate air quality data (such as CO2, PM2.5, PM10 etc.) coming from an area of interest. Data is retrieved from configured air quality stations that use SensorWebHub platform [1]. An user entering the application can select a device using a map that display all the configured devices in their respective locations, so that the device information (such as name and location) and its collected data will be shown to the user.

1.2 Idea source

The idea comes from the developer company I work for, as we have an air quality station that uses the platform mentioned before. We also wanted to explore a new way to develop a web application that can ease the production process and let the developer focus mainly on the frontend side: a serverless development model.
Serveless web development has been growing in popularity and it has been formalized as JAM (Javscript, APIs, Markup) stack [2]. In this case study we decided to adopt a cloud based solution: the role of the server, and all the effort that will be spent on developing a fully functional backend, is completely replaced by all the functionalities that a cloud host can offer.
We decided to use Microsoft Azure as cloud platform.

1.3 Development flow

The application development has followed this flow: the simple request from the company has been transformed in structured requisites by analyzing their needs in order to better

understand what the user's workflow will be and how to consequently build the user interface. This process is shown in chapter 2.
An initial prototype mockup has then been built using Figma, a cooperative and easy to use mockup designer. Once the design has been approved from the company, the development of the frontend side and cloud functions has started and the two sides have been developed (mainly) indipendently. Adopted technologies and application development are described respectively in chapters 3 and 4.

1.4 Expected outcomes

Expected outcomes can be summarized in the following two points:

- a working web application that can be used to monitor air quality, that is easy to use even for those who don't have a solid background in using computers and smartphones and that satisfies the requisites listed in chapter 2.5,
- an analysis of the adopted development method, which can speed up production and maintenance, and so lower development costs.

2 Needfinding, personas and requirements

2.1 The needfinding process

The purpose of the process of needfinding is to better understand user needs in order to better develop a pleasing user experience. For this project we have an expressed need that comes directly from the place I work for. We can formalize this need as: "I need a system that we can use to display data coming from our station and this system should be serverless and cloud based".
Analyzing this user request we can observe that the first part shows us what the system should do while the second part is a requisite about how the system should be built.
As the expressed need has not been further extended we can work on latent needs, i.e. not expressed needs that we should take care of in order to better develop a meaningful application.

We will now focus on the first part of the expressed statement: "I need a system that we can use to display data coming from our air quality station".
We can extend this statement by thinking about what is currently missing. An example of it can be the following sentence: "I have a station that produces some open data but I don't have an easy way to analyze its collected data". So the user has an item that can serve the community by offering data to monitor the air quality of its city but there isn't an easy way to display these values, thus accomplishing a meaningful analysis (for example, how much CO2 is reduced due to traffic limitations caused by the Coronavirus quarantine in the last month).
We can now deduce that this lack of a friendly user interface will not only affect the air quality station owner (and in general all the owners of that kind of instrument) but anyone who wants a simple way to learn about the air quality of a particular zone or city using this data.
The archetype user of our application will be better formalized in chapter 2.4, for now we can express the latent need as having an open platform that anyone can use, even those who do not have a solid IT background, to do either simple or in-depth analysis of data coming from stations.

As said, the second part of the statement tells us a need about how the system should be built. The approach to solve this need will not affect the experience of the user previously described, but it will affect the developer(s) involved in the system implementation. This need will not be taken into account to implement a better user interface, it will be used instead to explore a new way of building a web application. Using a cloud approach can speed up the process of development for this kind of applications: the backend system is replaced by cloud functionalities, such as a database system with auto replication and interfaces that can be used to access it through user developed functions triggered by an http request.
This type of technologies can be used by developers to focus their work in developing the frontend side and taking care only of how data should be requested/transmitted from the cloud to the application.

2.2 Personas and use case scenarios

Identifying personas, i.e. build an ideal user, is an useful approach to build the correct flow within the application.
For this application, two main personas and use case scenarios have been formalized. They will be now briefly summarized.

Gwen

Gwen lives in Florence, she works in public administration as she is responsible for the environment strategies of its city. Air pollution monitoring can be an useful instrument to evaluate her decisions' results.
An example of this could be how much a vehicle traffic block on a particular zone of the city has influenced its air pollution: a good result can support her decision to reduce the traffic.
The resulting use case scenario can be the following.

1. Gwen knows that a monitoring station is placed on a particular zone of Florence and she knows about this application.
2. Gwen enters the application and she can see a list of devices.
3. Gwen chooses the device of interest
4. A new page will be displayed, and she can see exactly where the device is located
5. Gwen can see that there is a section for in-depth data analysis that displays a chart of collected data in a given range of time and grouped by pollution indexes.
6. Gwen chooses the starting and ending date for which she wants to analyze trend values.
7. She can now choose a pollution index value of interest (e.g. CO2).
8. A chart will be shown to Gwen where she can learn how much her decisions have influenced the air pollution.

Leon

Leon lives in Florence and discovers the application by surfing the web. Leon does not have a knowledge about air quality indexes and wants to know how much the air in Florence is polluted w.r.t other cities.
An use case scenario for the described user can be the following:

1. Leon enters the application, he can see a list of devices in various cities and Florence is one of them.
2. Leon clicks on the device located in Florence.
3. A new page will be shown to Leon
4. In this page he can see where exactly the sensor is located and its name, but more importantly he can see a collected data summary of the last weeks

5. He can see that, for each day, a list of average values for certain pollution indexes is shown, but he knows nothing about these indexes.

6. He can now click on an index he wants to know about (e.g CO2) to have a brief description of what it means.

7. Now Leon knows what the C02 index means and he can see a list of values for the last week, but he does not know if these values are good or not. So he decides to compare them to the values of another city.

8. He decides to return to the homepage and selects a device in another city (e.g. Bologna): he can repeat the previous steps to know the CO2 values of the last week in Bologna and how these differ from the values of its city.

2.3 Application requisites

We can now better formalize the requisites using the previously descripted needs and use case scenarios.
Requisites can be grouped in two main categories: functional and non functional. The key difference between the two is that the first group defines what the user can do and how to do it (it defines a set of specific functions), the second group defines instead how the system should be.
Given the previous model, the functional requirements can be formalized as following:

1. The application must be exposed as a web application without a login process.

2. Collected data must be stored and accessible using Microsoft Azure Cloud technologies

3. The application must be responsive on mobile devices.

4. Users must be able to select all the configured devices in order to see their data.

5. By clicking on a device the user must see the following information:

    5.1. Device Name

    5.2. Device Position

    5.3. A summary of the last 14 days of data collection (daily average values)

    5.4. An in-depth analysis of all the data collected: users must be able to query data by date and understand the trend between the start and the end of the requested period

    5.5. The value types used in 5.3 and 5.4 must be at least the following:

        5.5.1. Temperature

        5.5.2. CO2

        5.5.3. PM2.5

        5.5.4. PM10

        5.5.5. RAD

        5.5.6. VOC

6. A basic explanation of air quality indexes must be given to the users, if requested

7. Once a device is selected the users must be able to select another device

Non functional requirements can now help us to better figure out how we can build this system in order to support the functional requirements previously described and to further extend functionalities. This type of requirements can be particularly useful in designing graphical user interfaces.
The identified non functional requirements are the following:

1. The application can be presented as a "two-page application", the homepage will display the configured devices, after the selection of a device a second page will be shown, displaying all the info explained in section 5 of functional requirements.

2. A fixed header should be used in each page in order to easily navigate the site (i.e. return to the homepage once a device is selected)

3. All the configured devices should be displayed on a map to ease the process of device localization for the user.

4. The data summary explained is section 5.3 can be built as a table where each row displays the data values for the types explained in section 5.5 in a given day. For each day of the summary a new row will be created.

5. The data visualization for the in-depth analysis explained in section 5.4 can be built using a chart. This approach can ease the comprehension of the data trend for the user. Two calendar widgets can be used to select the starting and ending days. Only a chart of a particular data type can be shown at a time, the user can select the type of chart to show other value types.

6. Due to data simplicity (and lower costs), a noSQL database can be used to store data in cloud. This simple database will maintain the names of the configured devices and their collected data.

7. Data retrieval from cloud should be done using a restful approach. Each resource (list of configured devices, summary data, in-depth data) should be accessible on a particular endpoint, using JSON as resource format.

3 Underlying technologies

3.1 Figma [3]

Figma is a collaborative mockup platform. This application has been used to build the first prototype of the application.
This tool is particularly useful due to its nature: it is accessibile via web, it is easy to use and multiple people can collaborate on the same mockup at the same time from remote. Moreover it is possibile to do dynamic mockups using transitions between pages and to simulate a "fully functional" mockup on configurable screens and devices.

3.2 React [4]

React is a JavaScript library designed for building user interfaces.
A React application is built of smaller entities, called Components. Components are stateful (each Component has a state), and they can communicate by passing to each others variables, called props. Props and state management is done by React using the Component lifecycle and it offers various hooks that developers can use to handle component life scenarios.
Creating small components and then composing them to build the user interface is an efficient method that makes the code more readable and mantainable.

3.3 Redux [5]

Redux is a JavaScript state container. It is particularly useful when used with React in order to manage components' props, which can grow exponentially as the application gets bigger.
By using this library we can design our application in order to follow the Model View Controller pattern, as React itself does not separates the state logic and the view.
Redux is composed of three basic components: centralized state, actions and dispatcher. React components can register to the Redux store to either receive updates on centralized state changes or to update it using actions. Once an action is fired by a component, the dispatcher will update the cetralized state following the developed logic and will dispatch the changes to the registered components.
With respect to the MVC pattern, the centralized state and the dispatcher are the Model, the actions are the Controller and the React components are the View.

3.4 Microsoft Azure Cloud [6]

Microsoft Azure is a cloud computing platform that offers a lot of functionalities and applications.
For this project the needed functionalities are:

- a storage application that stores data coming from our sensors
- tools that put data from the sensors in the storage
- tools that retrieve data from the storage and send it to our application

The first requirement can be met by Azure Table Storage, a simple noSQL database, while the second and third requirements can be met using Azure Functions: simple applications that can be written in a variety of programming languages (Python in our case) that can be fired by http or cron triggers.

4 Application development

4.1 Development flow

In this section the development flow of the entire application will be described.
The process is started from the application mockup, that is done using the requirements and use case scenarios previously described. Then an analysis of how the Azure cloud platform works has been taken into account to design how the data storage should be built.
Once these fundamental blocks have been consolidated the real application development is started: the frontend side and the rest functions can be built indipendently and then the two parts will be connected together.

4.2 Mockup

The mockup is a fundamental starting point to design the graphical user interface and the navigability of the application. Moreover, it is a very useful tool to enhance cooperative work and to show to the application's stakeholders how the GUI and user interactions will be developed.
For this case study the used mockup platform is Figma. This platform enables the user to build a fully functional project directly via its web page, and multiple users can work on the same project at the same time. This platform provides very useful features, such as page transitions and the rendering of the mockup on multiple screen sizes and devices.
The application has been designed using a material design approaches. Each block of information is placed on a card: this approach will ease to process of control mappings by grouping together the same functionalities. Using the functional and non functional requirements, the application has been designed as following.

4.2.1 Home page

The home page is the entry point for the user.
It displays an introduction title and a map with all the configured devices. The user can click on one of them to navigate to the selected device page. These two elements are placed on two distinct cards.
As shown on the **Figure 1**, the signifier "Click on a device to start" has been placed to let the user understand what to do to show the device information.

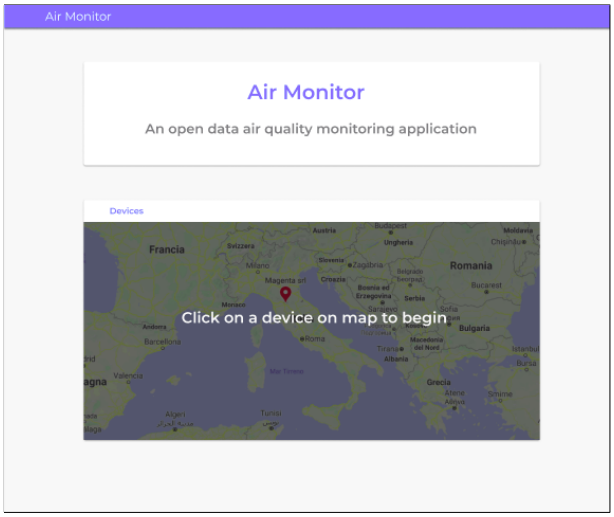

Figure 1: Homepage mockup

4.2.2 Device information

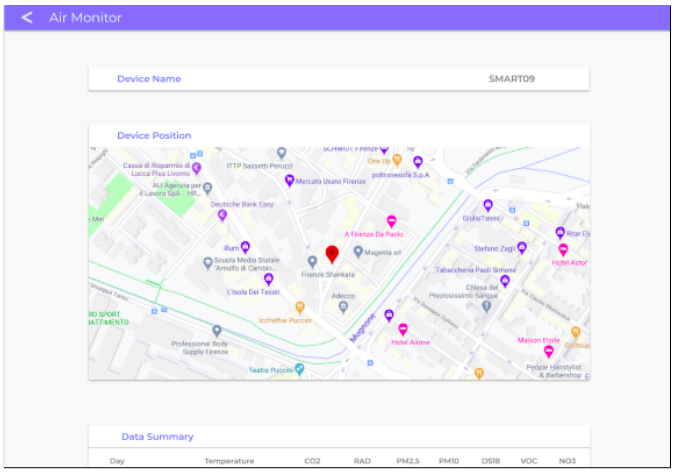

Figure 2: Mockup of the device information's cards

Once a device has been selected, this page will be displayed.
It shows four cards: each one groups different information and commands. The first two cards presented to the user, shown in **Figure 2**, are informative: they will show the device name and its zoomed location on a map. This let the user understand what device has been selected and where is exactly located.
The next card it's the device data summary. This tool will be used by the user to have a fast view of the trend of daily mean values for the last weeks. The last card contains the tool that will let the user analyze the device data. As shown on the images, two calendar widgets will let the user control the starting and ending dates of the chart. Moreover, the charts will be separated by indexes, so a group of labels will let the user select which kind of values should be displayed.
It is important to focus on control mapping for this card. These two types of widgets are grouped separately as they are used for different purposes. Moreover, the display order of these widgets is important: the user will firstly select the range of time and then will choose the proper type value. This will let the user understand how to manipulate the chart using the displayed controls.

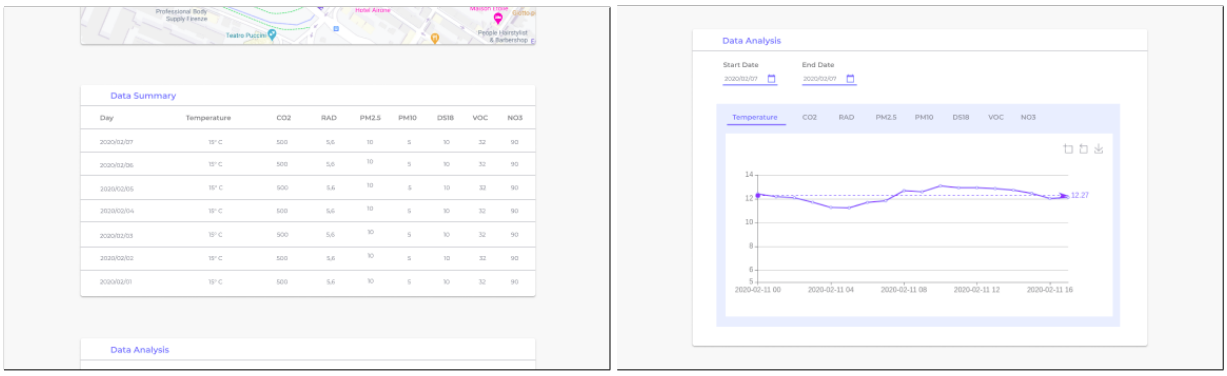

Figure 3: Mockup of summary data and data analysis cards

4.3 Cloud data design

A fundamental step to develop this kind of application is to understand how the data is produced from the sensors, and so how to store it in a proper way.
For this study case, the sensors use an online platform to store their data. This data can be retrieved with a REST API specifying the device, and the starting/ending date. The outcomes of this request will be a csv file where for each row a set of values is given, such as the position, the detection time and the air quality values. A detection is done approximately every 2 minutes, for a resulting of 700 rows per day.
A study of the Azure cloud platform has been done to choose the right resources to use for our application. The chosen resources are Azure Tables and Azure Functions. The first one is a NoSql database that will store our data, while the seconds allows the developer to write functions, in various programming languages, that can be launched via an HTTP request or by a cron trigger.
The data flow is now described.

1. At the end of each day a cron function will grab data from configured sensors by calling the previously described REST API;

2. The csv is rearranged to evaluate the average values for each hour and any missing data is filled with null values;

3. A JSON is then produced: it contains the name of the device, its position, its average daily values and its average hourly values for each kind of index;

4. This JSON will be stored on Azure Table on a new row with timestamp as the row identificator;

5. The data in Azure Table can be retrieved by calling the HTTP Azure functions exposed on certain REST endpoints. These will manage the access to the database and execute queries on it. The resulting data will be sent to the requester in JSON format. These functions will be described in chapter **4.5**

4.4 Frontend

Once the mockup and the data design has been accomplished, the real frontend application has been developed.
It is developed using React library and Redux as state manager. The source code is available in the following repository: `https://github.com/francescoareoluci/air-monitor`.
The project structure is the following:

- /src/
  - /src/components/
  - /src/css/
  - /src/js/
  - /src/images/
  - /src/fonts/
- /public/

- /package.json
- /webpack.config.js

The key concept of React library, which is oriented to user interface building, is the usage of components. Each component has a state, methods and can pass variables to its child components using props. The developed components can be found in /src/components/ folder.
This model can be enhanced using Redux state management: the state will be centralized and accessible/modificable from the components that need it. This state management is done with the actions and the reducers. Actions can be used by components to update the centralized state. Components registered to a subset of the state will receive notification (i.e. new props will be passed to the component) every time the state is updated thanks to the reducers. The actions and the reducer can be found in /src/js/ folder.
As said, this kind of approach will result in an implementation of the MVC pattern, where the Model is the centralized state, the Controller are the action and the reducer, and the components are responsible to implement the View.
The application uses npm as dependency manager and webpack to build the project. The configuration of the two tool can be found respectively in /package.json and /webpack.config.js files.
The user interface has been reproduced very closely to the mockup showed in section 4.2, with some little modification to improve usability and displaying on smaller devices.
Feedback is a crucial part in designing graphical user interfaces. For this reason all the async components, which require a request/response to display data, provide loading and error handling. This way, the users will always be aware of the state of these components.
The frontend has been deployed using Github Pages, which let the user to manage the source on the master branch and push the application builds on another branch that will be hosted on a public domain.
The application is accessibile here: `https://francescoareoluci.github.io/air-monitor`

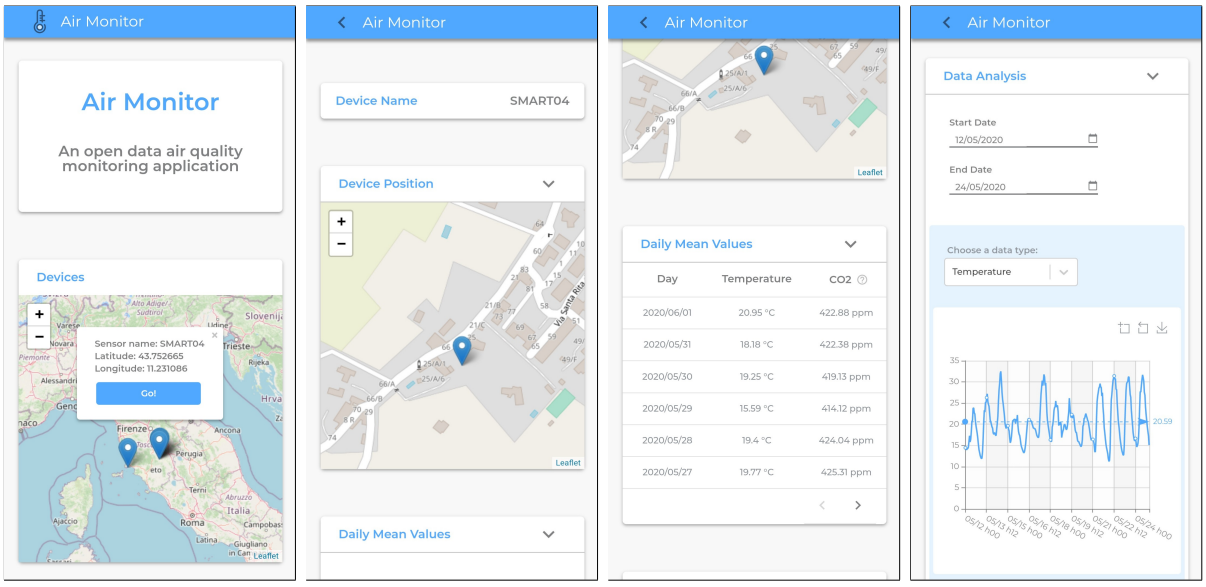

Figure 4: Responsive frontend on mobile devices

4.5 Azure rest functions

Azure Functions tool has been used to develop stateless functions in Python that are exposed on rest endpoint to access Azure Table database and send data to the requester. The endpoints are the following:

- https://pullairmonitordata.azurewebsites.net/api/configured-devices
- https://pullairmonitordata.azurewebsites.net/api/summary-data
- https://pullairmonitordata.azurewebsites.net/api/device-data

Each endpoint has a function associated. A get request on one of these endpoints will trigger the function.
The structure of the three HTTP function is very similar between them:

1. Parameters are evaluated from the requested api
2. Access the database
3. Query the database using parameters previously evaluated
4. Format the query response
5. Send JSON to the requester
6. If something goes wrong during the database connection, an HTTP 500 response will be sent. An HTTP 400 response will be sent if a failure happens during requested parameters' evaluation.

The responsabilities for these functions are the following:

- **configured-devices**: will return a JSON containing all the devices with their name and positions.
- **summary-data**: will return a JSON containing daily averages values of the last 14 days, for a requested device.
- **device-data**: will return a JSON containing hourly average values for a requested device in a requested range of time.

4.6 Frontend - Cloud Data connection

Once the frontend and the Azure functions has been developed, the frontend is ready to implement the get requests to retrieve data from the cloud.
Before the connection to the cloud, the frontend has been firstly implemented with random generated values to test functionalities of data visualization, loading and error handling. Then it has been tested with local runs of Azure functions to test the endpoints and JSON responses. Once everything has been tested, Azure functions have been deployed to the cloud.
Endpoints will be asynchronously called in the following scenarios:

- An access to the homepage will request **configured-devices** function

- An access to the device information page will request **summary-data** and **device-data** functions, for the configured device and the default range of time
- A change of the chart data range of time will request **device-data** function.
- A change of device will trigger again **summary-data** and **device-data** functions

An issue discovered after the functions' deploy and the connection with the frontend, has been addressed to the Azure Functions **Cold Start** [7]. This type of issue is caused by the modalities of how the functions are run on unaware servers and how resources are allocated/deallocated on them.
This issue causes a noticeable jitter on request/response if functions are in cold start state. A naive solution to this problem has been done by deploying a new cron function that will run every 5 minutes to try to mantain the warm state of the group of functions. A less naive solution consints in deploying functions using Azure Service App. By doing this, functions will reside in a Virtual Machine and will always be in a warm state.

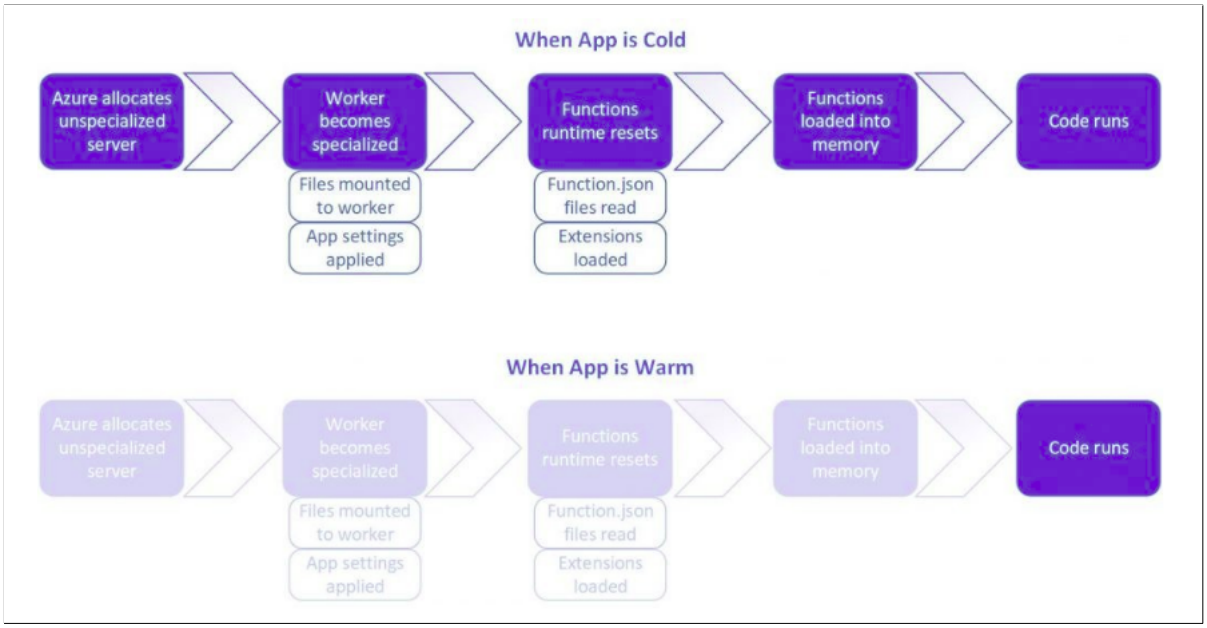

Figure 5: Azure Cold Start

5 Usability Testing

Usability testing has a fundamental role in designing graphical user interface, especially during the prototype development. Thanks to usability testing, flaws and defect of the developed product can be detected and fixed. Various approach has been developed over time to get the best feedback from the users under testing and to obtain more useful results.

5.1 Testing Protocol

The adopted testing protocol is composed of two tests. Each test is subdivided in tasks. Tasks have been designed using the developed Persona use case scenarios descripted in chapter **2.2**. Users have been asked to take part of some scenarios to perform the actions for which we want a feedback, instead of simply execute a set of actions. This way the user will immedesimate in the role of the expected Persona and its results will be more accurate.
The developed testing protocol has been structured this way: two sets of task will be given to the users, each task will be timed to understand what operation requires the most effort for the user and a success rate will be evaluated for each task. These values will give us a quantitative analysis for the analyzed operations. The two sets of test are executed on a Desktop environment.
Participants under test are not familiar with the proposed project and they have different levels of experience when interacting with web interfaces. The two test cases will be now briefly described.

Test 1

1. You live in Florence. You have discovered this application by surfing the web and you are interested in knowing the air quality indexes for your city. Entering the application you see that you can select a device from a map. You choose the device SMART09 that is located very close to your home.

2. Now that you have selected the desired device, a new page is shown to you. In this page you can see a list of information, such as the device name and its location. You have heard about some air quality indexes such as C02 and particulates and you want to know the trend values of these indexes for the last two weeks. Looking at the page you see a section that displays the daily mean values for the indexes you want to know about.

3. In this section you can see a list of values. You are particularly interested in CO2 values, but you don't know exactly what does this index mean. You want to know more about this index and you see that the section offers an informative dialog, so you decide to open it.

4. You have read the explanation for the CO2 index and now you want to check the indexes values of the last two weeks.

5. Now that you know the CO2 data trend for your city, you still don't know if these values are good or not. To have a better understanding of these values you decide to look at the data of another city. So you decide to return to the homepage and select another device.

6. You see that the device SMART07 is placed in Livorno and you want to know about its data. You decide to select this device and repeat the same steps that you did with the previous device to know the CO2 trend of the last two weeks.

Hints can be given to the user, for steps 1-5. Timing of step 6 will determine the learnability for this set of operations.

Test 2

1. You live in Florence. You have discovered this application by surfing the web and you are interested in knowing how much the quarantine caused by the COVID-19 has influenced the CO2 values in your city. Entering the application you see that you can select a device from a map. You choose the device SMART01 because its location is close to your workplace.

2. Now that you have selected the desired device, its information page is shown to you. Looking at the page you can see that there is a section that you can use to perform data analysis based on time ranges and indexes.

3. You want to use this tool to check the CO2 trend during the quarantine period. To do this you need to change the default data range. To perform a meaningful analysis of the trend you may want to set the start date at 15/03/2020 and the end date at 15/05/2020

4. Now that you have selected the desired range of time, you want to have a look at the CO2 trend, but the default index (temperature) is showed, so you select the CO2 index.

5. You can now identify the CO2 data trend during the quarantine using the chart. Is it ascending, descending or constant?

Hints cannot be given to the user.

The following tables contains the mean values and the standard deviations in seconds for each executed task.

Test No.	Mean	σ
1.1	35.6	15.27
1.2	9.2	1.92
1.3	5.6	1.67
1.4	17.6	4.56
1.5	6.2	3.35
1.6	24.6	3.78

Test No.	Mean	σ
2.1	20.2	11.58
2.2	12.8	2.95
2.3	24.4	10.48
2.4	3.8	1.64
2.5	4.6	0.55

Table 1: Mean and Standard Deviation for each task

While the test completion give us a quantitative analysis about executed scenarios, a qualitative analysis can be performed by submitting some questions to the users through a questionnaire. 11 Single Ease Questions have been submitted to the users, which can be answered in a 7-point Likert scale, where 1 means "Strongly disagree" and 7 "Strongly agree". Moreover, some questions are written in negative logic: this encourage the user to not give the same answer for more questions and can be used to isolate false positive. The following table displays the questions submitted to the users after the execution of the previously described tests. Mean and standard deviation have been evaluated for each question.

No	Question	Mean	σ
1	Tasks completion required too much effort	1.4	0.55
2	I was able to complete the tasks very quickly	6.2	0.84
3	The device selection in the homepage was easy to use	6.4	0.55
4	After the device selection, I find useful to know the name and the location of the devices	6	0.71
5	After the device selection, I can easily check the average CO2 values for the last two weeks	6.2	0.84
6	I easily found the informative dialog for the CO2 index	6.8	0.45
7	I did not find the data analysis chart management easy to use	1.6	0.55
8	Once a device has been selected I can't easily select another device	1.6	0.89
9	I would like to use a feature to directly compare devices' data	5.8	1.30
10	Overall, I found the user interface easy to use	6.4	0.55
11	Overall, I am satisfied with the AirMonitor platform	6.6	0.55

Table 2: Submitted questions and their results

5.2 Test results

The quantitative results can be used to detect the tasks that took longest execution time and their deviation between the users under test. The most time consuming task is the device selection in the homepage, as we can see from the tasks **1.1** and **2.1**. Moreover, the high standard deviation of this task is caused by the experience level difference between the users: less experienced users have had some trouble in using the map and in the identification of a device. In particular, when the map is totally zoomed out, nearly located devices are not easily distinguishable. This can be improved by adjusting the default zoom if devices are nearly grouped or by using different colors to distinguish them.
Tasks **1.3** and **1.4** refer to the summary data section usage. Results show that the informative dialog can be easily found on the user interface, while the table pagination usage has not been enough clear for all the users: a possible improvement can be achieved

by adding the signifiers "Next/Previous page" near the page arrows.
Results of task **1.6** show a good level of learnability of the first test scenario between all the users.
Tasks **2.3**,**2.4** and **2.5** refer to the data chart management and visualization. Their results show that the data range selection is the most consuming task of the three, while the index selection and the chart reading can be accomplished very quickly. An issue on date selection button has been highlighted during the test: the widget was not showing on the click of the calendar icon.
Notes from the users has suggested an insufficient feedback for the user on summary data page change caused by row's values not so much different between them. This problem can be fixed by adding an animation on the transition of table pages.
The qualitative analysis perfomed on the submitted questions shows that the users are enough satisfied of the platform and have found the graphical interface enough easy to use. Question 9 has been submitted to know if a specific new feature should be included: its results show that a tool to directly compare devices data can be useful and will be appreciated by the users.

6 Conclusions

The developed application is an open platform that enables the users to check air quality values coming from public devices. The platform has been developed as a serverless cloud-based web application: this approach let the developer to focus more on the graphical user interface by saving time in developing a fully functional backend.
The cloud functions and the pre-built system that handle the database management can improve the quality of the development by lowering the written code and the mantainance time. Moreover, the cost of the used cloud features is very low: adding more sophisticated features can enable us to delivery an improved version of our application, for example to handle the cold start of the cloud functions. The benefits in building small applications using this kind of approach are noticeable and this development process should be further explored in the making of more complex applications.
The usage of a frontend framework has improved the developing experience by adding a structured approach in the development process of the application. This has made the code much more robust and mantainable.
The process of needfinding and the Personas designing, followed by the requirements' definition has contributed by formalizing the ideas behind this platform.
Finally, the usability testing has highlighted application's flaws and issues regarding the graphical user interface. The correction of these problems can be used to further improve the application usability.

References:

[1] http://www.sensorwebhub.org/

[2] https://jamstack.org/

[3] https://www.figma.com/

[4] https://it.reactjs.org/

[5] https://react-redux.js.org/

[6] https://azure.microsoft.com/

[7] https://azure.microsoft.com/en-us/blog/understanding-serverless-cold-start/