

Received July 3, 2019, accepted July 22, 2019, date of publication July 31, 2019, date of current version August 14, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2932196

A Progressive Web Application Based on Microservices Combining Geospatial Data and the Internet of Things

MANEL MENA^{ID}, ANTONIO CORRAL, LUIS IRIBARNE, AND JAVIER CRIADO

Applied Computing Group, University of Almería, 04120 Almería, Spain

Corresponding author: Manel Mena (manel.mena@ual.es)

This work was supported by the EU ERDF and the Spanish Ministry of Economy and Competitiveness (MINECO) under Project TIN2017-83964-R. The work of M. Mena was supported by a Grant of the Spanish Government under Grant FPU17/02010.

ABSTRACT Modern Web applications combine information from different sources, such as Web services, static resources, or real-time sensors data. The Internet of Things (IoT) is increasingly being used in these applications to show useful, updated information. However, the information related to the IoT devices is commonly displayed on dashboards for monitoring and control purposes and is not often combined with other types of data. In addition, it is important to base information on the location displayed in the user context. In this paper, we propose the use of a software architecture based on microservices and micro frontends for assisting the user in the friendly, seamless acquisition of geospatial data and information concerning the IoT. Our solution orchestrates those microservices and a component-based progressive Web application (PWA). The main microservice handles the creation of component configurations using a selection graph consisting of component tags and other descriptive properties and also contextual information about the application user. To demonstrate how the proposed architecture works, we present a scenario in which the Web application is dynamically built up by combining the geospatial information, the data acquired from the IoT sensors, and other complementary data.

INDEX TERMS Geospatial data, IoT, microservices, Netflix OSS, micro frontend, progressive Web application.

I. INTRODUCTION

The importance of geospatial data is growing enormously, and government entities are offering more and more useful related open data [1]. There are many solutions focusing on the extraction, transformation and storage of this kind of information, as well as visualization tools for it. Furthermore, with the increased use of technologies related to the Internet of Things (IoT), real time data can be acquired by measuring and analyzing environmental indicators, *e.g.*, from natural resources, urban environments or wearable devices [2].

Today's geospatial software visualization tools are narrowly focused on a single topic, performing simple tasks, and often just working with geospatial data alone. Such software shows maps with layers upon layers of data, usually so scientifically that it is problematic to understand for users

The associate editor coordinating the review of this manuscript and approving it for publication was Tawfik Al-Hadhrani.

with little or no knowledge in the related field. However, new approaches are emerging in IoT visualization tools for monitoring and diagnostics operations [3], for example, representing indoor devices [4].

From an architectural perspective, geospatial data approaches are generally at a standstill. Software related to such data tends to concentrate on merely trying to integrate different data formats to suit the needs of a particular software [5]. In our solution, we propose the use of microservices [6] and their orchestration [7], which has a series of advantages, such as the use of containers for seamless service replication, correct versioning, and load balancing between service instances, etc. In the IoT domain, some authors have encouraged patterns and best practices used in microservices [8]. We build upon that by making use of those advances in the geospatial domain.

Some of the solutions related to geospatial information also have a service-oriented approach [9]–[11], but only a

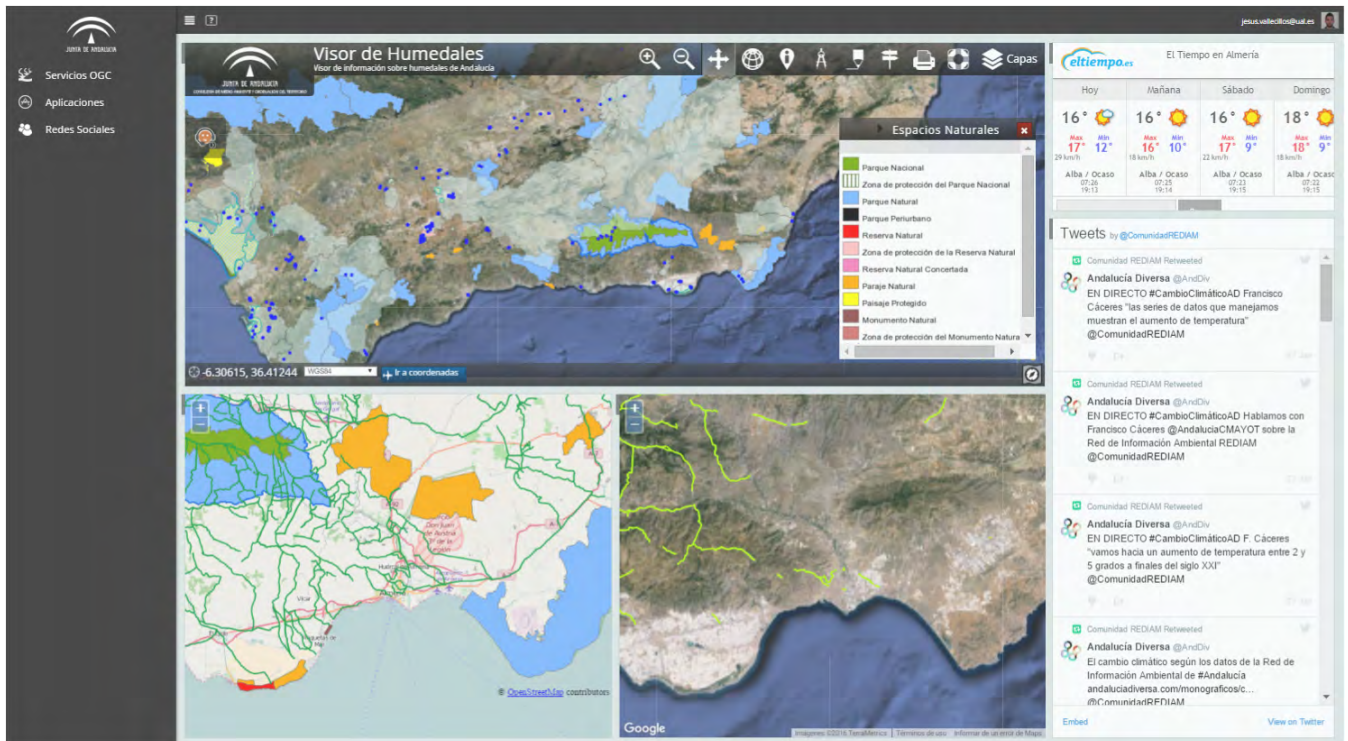


FIGURE 1. ENIA (Environmental Information Agent): An example of a component-based web application.

few apply the advances made in this field. Furthermore, IoT information on geospatial data can mean added value for an application like ours.

For an example of such possible benefits, think of a web application made up of two web components [12]: The first component is a microservice capable of providing a map via Web Map Service (WMS), and the second is a microservice that acquires data from an IoT sensor. We could set an orchestration system that makes it possible to set a trigger that launches a new instance of a service when requests surpass one thousand per minute. The new instance is registered in a discovery service making it part of a load balancer and lessening the system load at run-time. This can help overcome a sudden increase in traffic at a moment's notice.

From the user's perspective, it may be hard for them to meet their needs for information. Acquisition of the data may require a considerable effort by the user. To overcome that, we propose a way to encompass heterogeneous data. Thus, the user experience is the cornerstone of our application.

In this paper we propose a solution based on a multi-platform web application¹ for visualizing geospatial data and related information, which considers the context of the user in the application. The related information is focused on real-time data from IoT devices, but can also include other types of information, such as static resources or third-party web services. Some features like the location, the platform which the information is visualized on and the platform language have contributed to our improving the user experience.

¹<https://app.acg.ual.es/>

A Progressive Web Application (PWA) [13] was developed as part of the proposal. For it we used a component-based approach that we have had experience with, ENIA (ENvironmental Information Agent) [14], an intelligent Web agent for environmental information which was developed with funding from the Regional Government of Andalusia (Spain).

Figure 1 shows ENIA, a mashup interface in which every component is coarse-grained and encapsulates a certain functionality, and those components can communicate with each other. In this use case example, every component of the mashup is working together as a whole, finding a way to show economic activity in the province of Almería based on the environment. Three map components, an external weather component and a twitter component, are used for that. The ENIA web application does not show any real time information from IoT devices and the proposed approach is intended to fill this gap by providing this information through its front end.

The backend of our proposal has a microservice architecture, which takes several system-oriented patterns, such as *gateway* and *circuit breaker* into account [15]. This architecture and the use of context data help decide what information should be used to fill the components. A selection graph is used for this as an underlying data model. This makes it faster to narrow down the search of those components, thus improving the user experience.

This article is structured as follows. Section II reviews related work in the literature. Section III presents the background and fundamentals of microservices and PWAs on which our solution is based. Section IV shows the architecture

of our proposed solution from both the back-end and front-end perspectives. At the back-end, we propose division into three layers, edge services, core services and internal-external data. Section V describes a detailed use case of the application based on user context information. Finally, Section VI draws conclusions and proposes some future work. Summarizing, the main contributions are:

- (1) Architecture based on microservices, integrating different technologies and an orchestration method for combining IoT and georeferenced information.
- (2) Integration of IoT devices in a component-based user interface.
- (3) Use of context information to improve the user experience in a multiplatform progressive web application related to IoT and geospatial data.
- (4) The use of a graph-based data model with a selection algorithm able to gather components based on user context information.
- (5) A micro frontend component-based user interface coupled with PWA.
- (6) Setting the foundations (infrastructure, graph and selection algorithm, component-based development, etc.) to which components can be added, taking into account context-related information.

II. RELATED WORK

Our proposal was developed based on three pillars. First, the use of a microservice architecture, secondly the development of a front-end following a *micro front-end* approach, and finally the use of context data to improve the user experience.

Use of microservice architectures has been growing exponentially in the last few years. The microservice concept was first introduced in a case study by Lewis [16] and at about the same time by George [17]. Since then, the concept has evolved into what we now know [18]. For instance, companies like Netflix [19] make impressive use of microservice architectures to meet every single business objective and be able to grow at an unprecedented rate, taking close to forty percent of the bandwidth usage in North America in 2015 [20]. This architecture has been used in many kinds of applications, but not yet in research contexts which require a combination of geospatial data processing and IoT information management.

Recently, researchers in geospatial data have started to realize their advantages. In [21], the authors proposed the use of microservices to orchestrate existing geospatial processing algorithms, and to compose and execute geospatial workflows in a cloud environment. Another study related to the use of spatial data in a microservice setting [22], uses the spatial location to orchestrate the deployment of edge computing distributed services in *fog computing environments*. The approach presented in [23] processes and visualizes sensor data acquired by devices in real time. In that proposal, microservices were used to manage individual services across the complete visualization process workflow. Therefore, only the benefits related to service choreography were applied.

We leveraged the potential of microservice architectures to make the back-end of our application more resilient and provide our users with a faster, robust, secure and more reliable experience.






Novel IoT domain approaches take advantage of the benefits of microservices. For example, in [24] a Web of Objects architecture is proposed for providing and reusing IoT services. In contrast to our approach, that architecture represents each object (or a composition of objects) with a microservice, while we use microservices for managing and accessing our software components. Moreover, those microservices are linked to location data, but that information does not adjust the information to the user context, as proposed in our architecture. The proposal in [25] supports the development and evolution of context-based IoT applications. It is focused on three main aspects of IoT devices: context triggering, geospatial visualization and anomaly detection. The main difference from our approach is front-end communication. Those authors proposed *iframe* components communicating by the *window.postMessage* method, while we do so using the MQTT publish/subscribe policy (Message Queue Telemetry Transport) [26] protocol which makes updating data easier, more flexible and dynamic.

Front-end research is a continuously developing field, and component-based interfaces are already being used as a primary source of user interaction. Some of the most important web development frameworks (Angular, React, Vue, etc.) use a component-based approach as a reliable method of developing a web-based application. We already had experience with component-based mashup interfaces [14], [27], [28], but we wanted to improve them with the use of a Progressive Web Application (PWA) [29], [30].

PWA applications attempt to combine features offered by most modern browsers with the benefits of mobile experiences. At a first glance, they look like native platform applications, but they are developed as normal web pages. The only trait that has to be changed to convert web applications into a PWA is to provide them with unnetworked behavior. Simultaneously, a new paradigm was born, the *micro front-end* [31]. From a developer's perspective, a micro front-end adapts the microservice philosophy to front-end development with concepts like technology agnostic design, team code isolation, high reusability, etc. Our front end was developed based on both concepts (multiplatform and micro front-end), as PWA development leads to this philosophy.

Every major platform (Windows, Android, iOS, Linux, etc.) already supports the use of PWA as a bridge for web development in native platform applications without the drawbacks this kind of application had just a few years ago. With the advances in PWAs, the sensors fitted in devices to help with context awareness can still be used, as this forms part of one of our main objectives. Furthermore, major web browsers are starting to support a variety of those sensors, such as GPS, NFC, and others for PWA. To illustrate the support of certain features available to PWAs using Chrome, Table 1 shows the compatibility of each major platform with

TABLE 1. PWA features supported on Chrome.

	MOBILE		DESKTOP		
					
Progressive Web Apps	Yes	Yes	Yes	Yes	Yes
Service Worker	Yes	No	Yes	Yes	Yes
Web push notifications	Yes	No	Yes	Yes	Yes
Offline browsing	Yes	No	Yes	Yes	Yes
Background synchronization	Yes	No	Yes	Yes	Yes
Add to home screen	Yes	No	-	-	-
App launching screen	Yes	No	-	-	-
Bluetooth	Yes	No	No	Yes	No
Beacons	Yes	No	No	Yes	No
Geolocation	Yes	Yes	Yes	Yes	Yes
Geofencing	No	No	No	No	No
Image capture	Yes	Yes	Yes	Yes	Yes
Video capture	Yes	Yes	Yes	Yes	Yes

key PWA features. For a more in-depth look at the features supported on each browser for PWA see [32].

We developed our system with the user context in mind [33], so context awareness is mandatory for our application. Of the work related to this topic which has already been published, the study most closely related to the contributions examined here is presented in [34], where the system applies context data to improve the user experience. Context awareness is achieved in our application by harnessing sensor and device properties to give the user a personalized experience. Offering the users components personalized for them based on their location can help give those users the information they really need. For example, if they want to know the weather, with the user geoposition we can show the results from IoT devices or third-party Web Map Service (WMS) providing the weather data for their positions and with a granularity suited to their needs.

III. BACKGROUND AND FUNDAMENTALS

Some background knowledge on the main principles used is required to understand our solution. First, it is important to know the microservice architecture paradigm, as well as what a microservice is.

There is no clear definition of a microservice, but it is generally accepted to be a self-contained set responding to small problems, usually through services, which are built around key pieces of the business logic [35]. These microservices are independent processes able to communicate with each other by lightweight mechanisms, such as API (Application Programming Interface) HTTP resources. Most of the time they are written in different programming languages and do not have to be based on the same data persistence technologies.

A *microservice architecture* is based on appropriate orchestration of those microservices. It has the advantage of being able to independently raise replicas of the microservices in high demand immediately, unlike monolithic architectures, in which all the business logic is implemented in a single solution. A generic example of a microservice architecture is presented in Figure 2, showing how the back end is separated into small microservices, each with its own

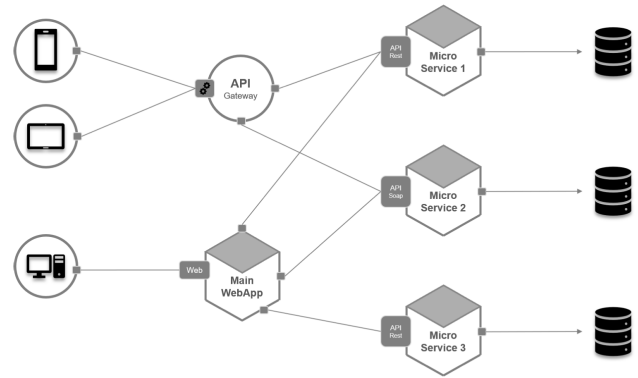


FIGURE 2. General microservice architecture.

database. At the same time, different devices can use different connection patterns, in which case, mobile platforms request information through an API Gateway and the Web Application is connected directly to each microservice. It is important to realize that a microservice architecture is dynamic, and there is no preestablished way of developing it, so we had to decide on the best approach for separating our microservices and how to handle communication between them.

Along with the use of a microservice architecture, the proposed solution takes advantage of PWAs as a way to deploy on multiple platforms with minimum effort. PWAs are normal web applications that provide an app-like user experience using modern web browser capabilities like push notifications, user device information, GPS capabilities, etc. Even so, PWAs are more than just a set of new features, they are a way of building better websites, leading to a set of best practices for site development, regardless of which device they use to visit it.

PWAs use service workers to make it possible to tap into network requests and help build better web experiences [13]. Service workers act as a middleware in each web request, and let the request go through if the conditions are met (e.g., internet connection is up), or will respond with a previous cached response if there are connection problems. Figure 3 shows an example of another service worker action. In this example, the service worker, acting as middleware, intercepts each request, and then, if the request is cached, it responds immediately, or otherwise, it retrieves it from the network.

The service worker generates a thread at the same time as the thread generated by our main Webapp, which provides the distinct advantage of alleviating the workload of the main thread. The service worker of our PWA is not tied to any web page. It cannot modify any element in the web page as it does not have DOM access and has its own global script context. Service workers are event-driven. The number of events that the service workers handle can be increased, so the development of a PWA tends to be progressive. There is no need to do it all at once, and so, converting a Web application into a PWA is a painless process, which can be done by bits and pieces, working on the things considered essential first,

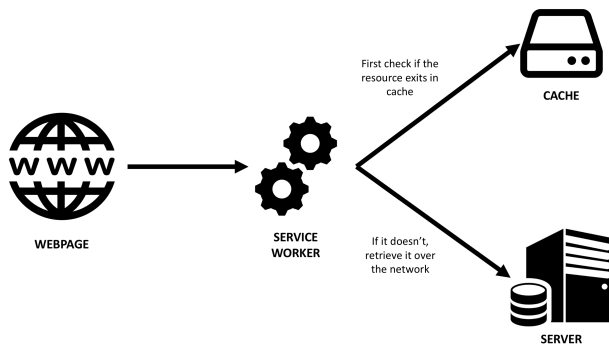


FIGURE 3. Service worker example behavior.

and then selecting more events to improve the behavior of the application step by step.

IV. MERGING MICROSERVICES AND IOT INTO A GEOSPATIAL PWA

In this section our solution is described from two perspectives, back-end and front-end, respectively, with special attention to how IoT devices and context information behave in our system. Our own microservice architecture is presented as a fundamental part of the proposed solution. Once the architecture is understood, management of back-end microservices and their behavior in use are explained.

After the description of how the back end works, front-end development is explained as a component-based architecture from a PWA approach.

A. BACKEND ARCHITECTURE

Due to the nature of microservice architectures, a set of small services needs to be managed. At the same time, the use of resources and the possibilities for micromanaging each are evaluated until a start-up order is established based on dependencies (*i.e.*, initiate geoservice after PostGIS is up), using Docker [36]. The Docker software platform makes use of containers to help handle a series of virtualized resources that are completely independent and have their own properties, like name spaces, file systems, libraries, etc. Containers in Docker share a common kernel, meaning that all containers can work over a single Linux instance, which helps avoid the need of running and maintaining several virtual machines independently. A Docker system is based on a layer that describes personalized configuration that has to be laid over the basic images in the container (See Docker Hub [37] for further information about Docker images).

One of the first questions to be answered is how to divide up the microservices, and how many are to be managed. In our case, we distributed all the microservices in the following three layers:

- Edge Services:** These services handle communication with the users. Included in this layer are Gateway Services, Discovery Services, and a Load Balancer.
- Core Services:** This layer contains the business logic. Included in this layer are Component Services, Translation Services, Geo Services and Auth Services.

- Persistence:** This layer is built up from the databases of internal data, as well as external services not managed locally. This data is used to acquire information to feed the components.

Figure 4 presents the proposed back-end architecture. The dark arrows show the flow of a request in our system. The light arrows show the behavior of the individual microservices. Each request sent to the Gateway Service is analyzed and then sent to the appropriate *Core Service*. The figure shows how IoT devices are connected directly to the MQTT broker (Mosquitto).

The Figure 5 represents a sequence diagram where microservice startup and its behavior is observed. In it, the Core Service routes are registered in the Discovery Service when a call is sent to it with the address and port of the single instance every time a microservice starts up. At the same time, the gateway service generates a call every 60 seconds to check microservice status.

1) EDGE SERVICES

This layer is mainly focused on orchestration of communication between microservices. Some Netflix OSS components [38] are used to achieve this orchestration. They are a proven, easy way to implement certain mechanisms inherent to a microservice architecture, and at the same time help implement a series of *system oriented patterns* [39] like the API gateway pattern, circuit breaker, client-side discovery and server-side discovery patterns, among others.

In this service layer, requests are orchestrated on a microservice level, managing external requests and redirecting them to the core services that can provide the appropriate response. This layer has two main services: (a) Discovery Service and (b) Gateway Service.

The **Discovery Service** acts as a registry of other services as well as a discovery server. Its main function is to manage the status of each back-end service that is running. It is also able to use Ribbon, a component that balances loads between services of the same type. The **Gateway Service** is the only one the user can communicate with, *i.e.*, the access point to the back end. It automatically generates a route for each service registered in the Discovery Service. At the same time, it handles requests between the front-end application and the back end. This is therefore the only point where an SSL certificate has to be used to make communications secure.

In addition to these two components, we use the Circuit Breaker pattern with Netflix OSS (Hystrix). This system pattern is implemented in the microservices to avoid error chains. Hystrix is ready to step in if some method is repeatedly causing timeouts in our requests. At the moment it detects an error, Hystrix opens the circuit handling all the requests following that method with a default response. At the same time, Hystrix sends successive messages to that method until it is working again, then it closes the circuit and everything works the way it is supposed to.

These services can be replicated at container level, so by establishing a prefilter in each of the microservices, a script

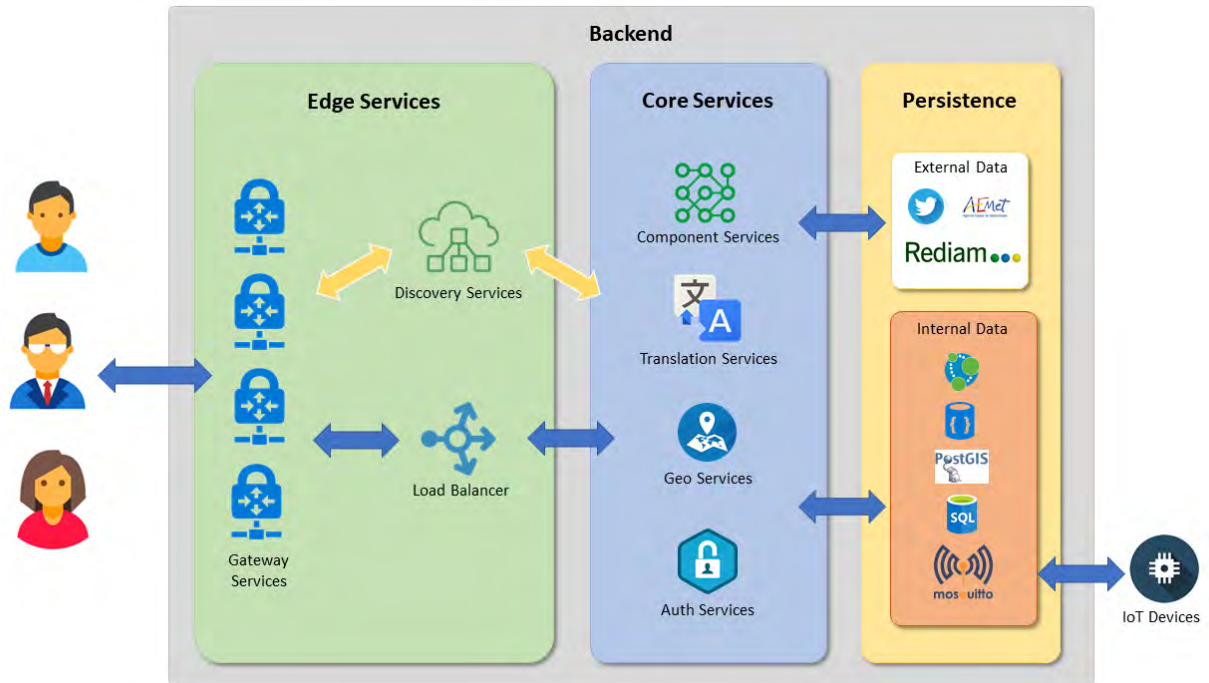


FIGURE 4. Geospatial PWA back-end architecture.

can be directly configured that launches other mirror containers after a certain load threshold has been exceeded. From that moment on, Ribbon will redirect requests to that mirror container as need, depending on the policy stated in the Discovery Service.

2) CORE SERVICES

The main services which implement the business logic, and auxiliary services which handle personalization based on the user context information are in this layer. The context information has four properties: the name of the platform the request comes from, the form factor of the device used to run the application, the language of the platform where the request is made, and the user position. The Core Services layer has four services: (a) Auth Service, (b) Translation Service, (c) Geo Service, and (d) Component Service.

The **Auth Service** manages authentication and authorization of users and applications that make use of the back end, using the OAUTH 2.0 protocol [40]. Each method of every microservice has its own scope, with certain restrictions of users who can access and/or applications that have permission to use them. The **Translation Service** handles the translations of words and units that depend on the user context. For example, if an American user wants to know the state of the beaches, he/she will receive the components translated into English and in the appropriate units (miles, degrees Fahrenheit, etc.). **Geo Service** is a middleware service that can manage geospatial requests. Behind this service, there is a Geoserver which serves maps in multiple formats and with multiple layers, as required by the component. It is worth noticing that the geoservice is used to draw maps with layers

where the position of the IoT devices can be shown. Finally, **Component Service** is the most important service in our back-end architecture. This service manages user needs and responds to them according to the context.

The Component Service states a set of components that fill with information related to the user’s context. Figure 6 shows the data model behind the selection of those components as a tree or a directed graph:

$$G = \{V, E\} \tag{1}$$

where,

- $V = \{Category, Region, Component\}$
- $E = \{Belongs_to, Contains\}$

and,

- $Belongs_to \subseteq \{(a, b) \in Category \times Region\}$
- $Contains \subseteq \{(c, d) \in Region \times Component\}$

The acyclic directed graph representing the data model behind our Component microservice is a set of vertex and edges that we use to select the adequate components to send as a response. The vertex represents the categories, regions and components. While the edges represent the relations between those vertex, like how the categories belongs to a region and how the regions contain different kinds of components.

First are the *Categories*, which are basically the subjects the user can search for and receive an appropriate response. Next, spatial *Regions* defined have a bounding box linked to the categories by a *Belongs_to* relationship. This bounding box shows whether the user is making the request from a position inside the region. This way the components can be

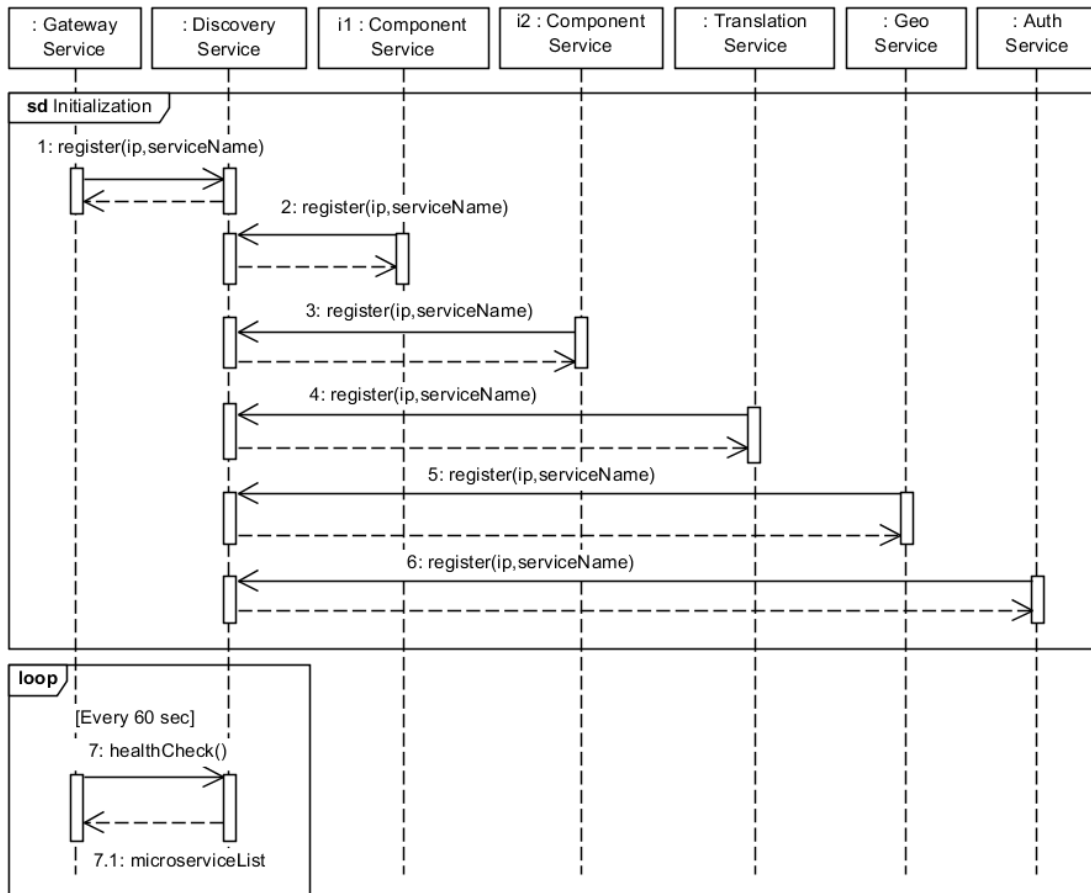


FIGURE 5. Back-end microservice startup sequence.

personalized based on data from the region itself. Each region has a series of *Components* compatible with the information from that region, defined by subject (*Contains* relationship), which can be filled with static information, such as a map divided into regions, or more dynamic data sources, such as a table with a data source suggesting a topic in the MQTT broker which is updated in real time.

Within this relationship, a field is defined with the scores of each compatible type of device that can be used to retrieve those components. A triplet (*smartphone, tablet, desktop*) where each field is a real value in the interval [0, 1], defines the affinity of that component on each device. For example, (0.1, 0.5, 0.9) means that the example has low affinity for smartphones, medium for tablets, and high for desktops, so the component is likely to be used when a user makes the request from a desktop. That way, their selection priority can be reorganized, taking into account possible incompatibilities between them in a device (for example, by definition, our application cannot show two maps on the same window in smartphones, but does on a desktop). These scores are assigned by expert knowledge when a new component is generated in the system.

The number of components returned in response is limited by the device, e.g., mobile devices only receive a

maximum of the four scoring the highest. Furthermore, the same component can be defined for different categories or regions, but with different scores and data origins. At the time of writing, our application only supports eight different regions in Andalusia (Spain), but it can be extended according to user needs. For external resources to feed defined components we use WMS and the Web Feature Service (WFS) offered by the Rediam (Environmental Information Network of Andalusia) [42] and data in XML format from AEMET (Spanish Agency of Meteorology) [43]. Use of external data is flexible, so more sources may be added.

To explain the concept of a bounding box and components shared by different regions, Figure 7 shows a map representing the graph above it. As an example, *Region 3* is composed of components #2, #3 and #4; whereas *Region 4* is only composed of Component #4, so both regions share component #4 between them.

The data model behind our Component Service is an important piece that solves the underlying structure used to store component data. As it is a graph, those components can be selected much faster by exploring the graph and working with the subgraph that the context information leads to. The component selection process is shown in Algorithm 1.

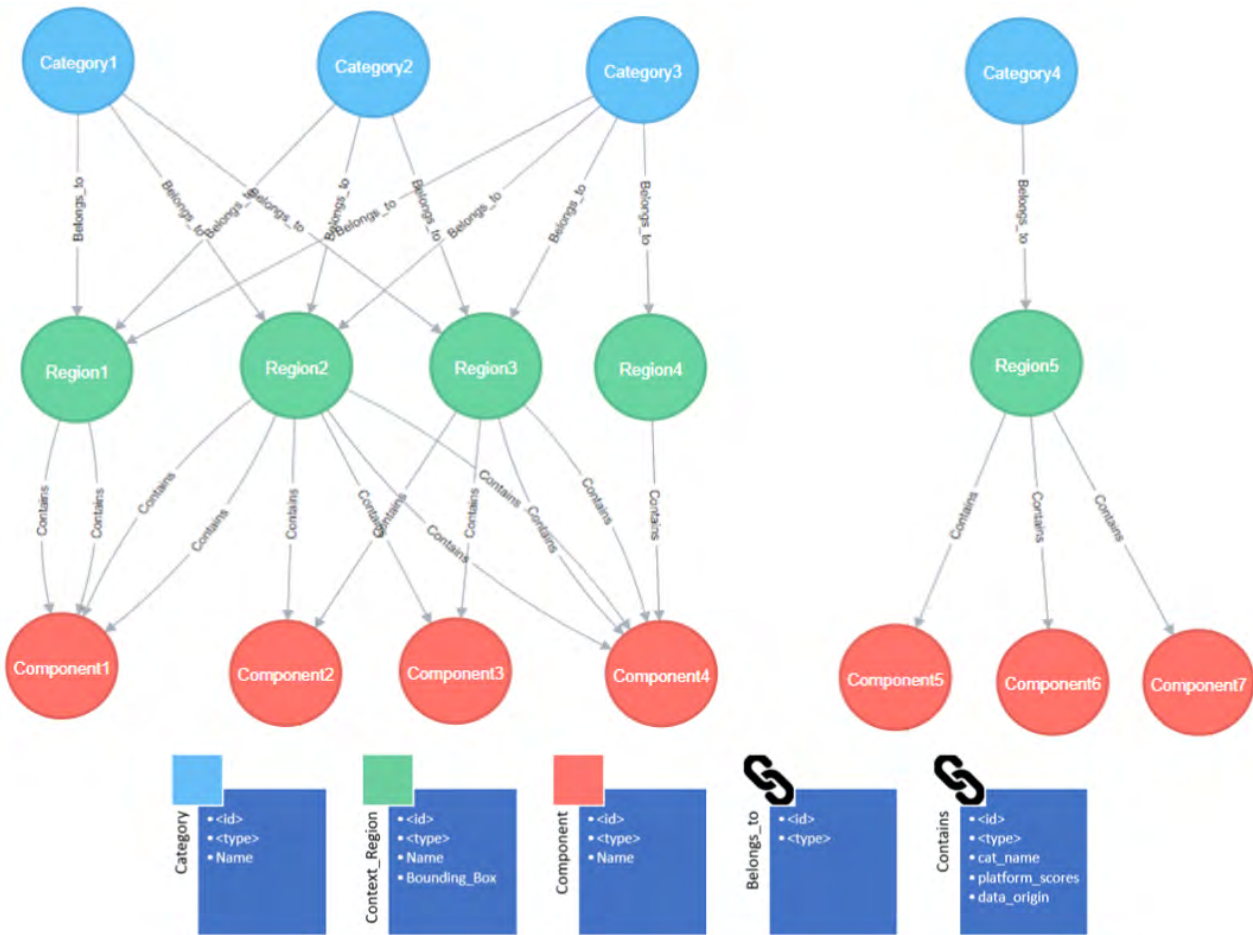


FIGURE 6. Component service graph.

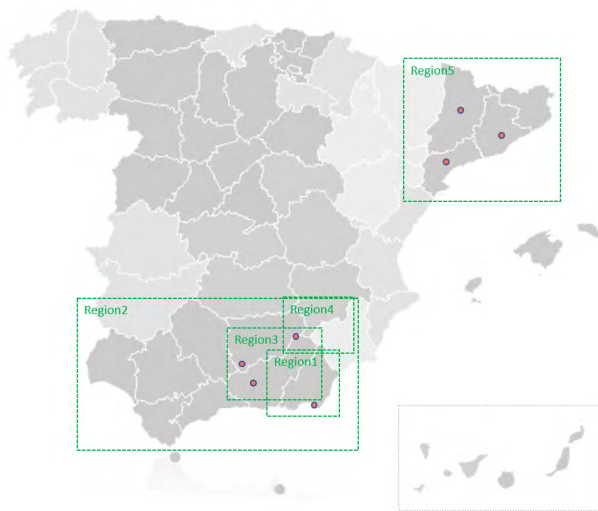


FIGURE 7. Component service region map.

The user context information (position, language, platform and form factor) and the category are necessary to execute the component selection algorithm. First, it translates the search

subject if necessary (line 1-5). Then it finds the subgraph related to the category search (line 6). Immediately thereafter, it picks the subgraphs in the regions the user is in related to the category graph which was recovered in the previous point (lines 7-11). It retrieves the components in the region graph in descending order based on the score defined in the data model that each component has in the particular platform employed by the user, at the same time it chooses the number of components necessary to fill the form factor of the main view in the front-end application (line 12) (see Section IV-B for more details). Then it translates those components if necessary (lines 13-15) and the application sends a response with the appropriate components (line 16).

Thus, the data model (selection graph shown in Figure 6) behind the components and their retrieval (Algorithm 1) form a system enabling new regions, components and/or categories to be introduced very dynamically and flexibly. At the same time, it enables them to be deleted or updated without impacting on the overall system.

To help understand its execution, Section V shows how this algorithm and component selection based on user context data work in an example scenario.

Algorithm 1 Component Selection Algorithm

Require: $c = \text{contextInfo}$ and $cat = \text{category}$
Ensure: $\exists c.\text{position}$ and $\exists c.\text{language}$ and $\exists c.\text{formfactor}$
 and $\exists c.\text{platform}$

- 1: **if** $c.\text{language} \neq \text{eng}$ **then**
- 2: $category \leftarrow \text{TranslationServ.translate}(cat)$
- 3: **else**
- 4: $category \leftarrow cat$
- 5: **end if**
- 6: $catGraph \leftarrow \text{getCatGraph}(category)$
- 7: **for each region in** $catGraph$ **do**
- 8: **if** $region.\text{contains}(c.\text{position})$ **then**
- 9: $regGraph \leftarrow regGraph \cup region$
- 10: **end if**
- 11: **end for**
- 12: $components \leftarrow \text{getComp}(regGraph, c.\text{platform}, c.\text{formfactor})$
- 13: **if** $c.\text{language} \neq \text{eng}$ **then**
- 14: $components \leftarrow \text{TranslationServ.transComp}(component)$
- 15: **end if**
- 16: **return** $components$

3) PERSISTENCE

The use of microservices makes it possible to select different data sources for each microservice. This has the advantage of using technologies that suits the need of each service. The internal data sources chosen for our services are databases which use *polyglot persistence* in the back-end to fit their main use case [41] in our back-end. In this persistence layer, we have deployed ($a \models b$) databases for each of the microservices as follows:

- $AuthService \models MySQL$: The schemas used for OAUTH2 follow a relational pattern, as it provides excellent performance and database integrity [44].
- $GeoService \models PostGIS$: This database was chosen because it is the most widely used for handling spatial petitions [45].
- $TranslationService \models MongoDB$: We needed a flexible schema, as each component is different. A document database is a flexible approach to this problem, and leaves open the possibility of adding more languages to the document and handling different components with different schemas in the same collection [46].
- $ComponentService \models Neo4jSpatial$: Like PostGIS, Neo4j Spatial [47] needs to know if a user is inside the same region to handle the spatial queries. At the same time, when a relationship becomes important in a relational database, a graph is much more efficient, as the graph can be crossed much faster than having to look up all the rows in the relational database.
- $Mosquito$: Apart from the databases used by the services, each IoT device has to be able to communicate with the rest of the system. We make use of an

MQTT broker for that. This broker enables topics to be published and subscribed to help acquire real-time information from every IoT device managed by the system. Each device generates a topic that follows the $/\text{kind}/\text{region}/\text{id}$ pattern, at the same time those IoT devices publish their values in at least another two $/\text{kind}/\text{region}$ and $/\text{kind}$ topics, so if necessary, those devices can be grouped by region and kind respectively (e.g., $/\text{temp}/\text{almeria}/001$).

For external data sources we use REDIAM, AEMET and the Twitter API and others as possible examples of information resources.

B. FRONTEND ARCHITECTURE

One of the main objectives of the front end was to make it available to as many people as possible, and at the same time make the experience as easy as possible for them. For this aim, a hybrid technology was the most appropriate solution, not only to be able to have versions of the application for mobile platforms, but also even in desktop environments if so desired. In the search for a technological solution or framework for developing our microservice application, we realized that in most cases, existing frameworks (Ionic, Cordova, Appcelerator, etc.) focus mainly on the development of mobile platforms.

We wanted to develop a hybrid application using a component-based user interface (UI), as we would need to accommodate the new components added to the back end without impacting negatively on the development time of their front-end counterparts. Another advantage of this approach is the possibility of using multiple component configurations at runtime, thereby providing dynamic flexible interfaces. With that in mind, we decided on development of a PWA, since it is a hybrid solution that is not only prepared to run on mobile platforms, but can also run on desktops. It is relatively easy to adapt a normal Web Application to a PWA just by adjusting the behavior of the web page when the device is offline and adding a few files, avoiding the need to learn new languages or technologies. Another advantage is that there is a single code base for all the platforms, and this is always more manageable than having completely separate applications in different languages for each platform.

In addition to the decision to use a PWA to develop the front end, we used a micro front-end approach, which attempts to take parts of the microservice concept to the front end. Especially important for our use is the isolated development of each front-end component, making it easy for every member of the development team to be working on different components at the same time without code conflicts. In a spatial context, different view components can be developed based on how the spatial data are to be shown. For example, the same spatial data can be shown in a table component, or as a component with a map with Google tiles as a base, or a list of places in order by name, etc. The component used is defined in the Component Service, so it is important to know that the front-end application requires at least one micro front-end

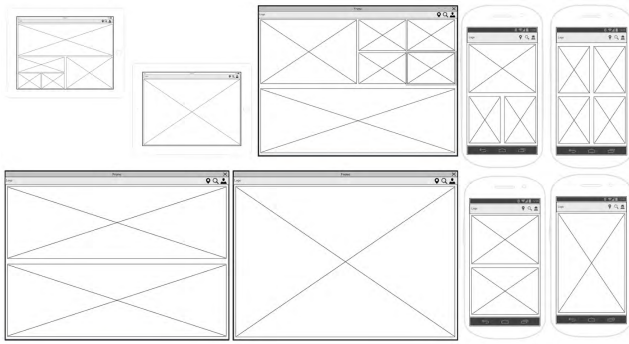


FIGURE 8. Different front-end configurations for the proposed PWA.

component for each one defined in the Component Service. The fact that each back-end component has its counterpart in the front end makes this solution very dynamic, and provides the possibility of having multiple data sources in each component, e.g., a humidity and temperature sensor (DHT22) can be connected to two topics at the same time in the MQTT broker, enabling the component to use those two topics at the same time, alone or in combination with other different components that handle the data sources in totally different ways.

Figure 8 shows different application configurations based on the device that is running it. The configuration will be based on the number of components chosen to be displayed by our Component Service, e.g., a category could have five components defined in a region, but in a mobile platform only three of them will be shown. Moreover, a different category may have just one component defined in a region. If this happens the component will always try to fill all available space in the display area of the platform. The number of components and the size of each one is defined by the context of the platform and the category of the search, but as a reference, the maximum number of components each platform can have is defined: a) Smartphones, a maximum of 4 components distributed in 2 columns and 2 rows, b) Tablets and Desktops (portrait mode), a maximum of 8 components, 2 columns and 4 rows, c) Tablets and Desktops (landscape mode), a maximum of 16 components, 4 columns and 4 rows.

It is important to notice that two different maximum layouts have been defined for tablet and desktops. The device is considered to be in portrait mode when the height is greater than the width, and in landscape mode when the width is greater than the height. This information forms part of the context. Such component organization facilitates the user view without overloading it with information.

Furthermore, the user can press a particular component for a full screen view (which may have more information for the user). In keeping with our goal making it more comfortable and flexible for the user, in a small component it is better to have a quick view, and then instinctively access more information without too much effort if necessary.

V. AN EXAMPLE SCENARIO OF THE PROPOSED PWA

This section explains a use case of our Progressive Web Application following the application flow from the moment the user makes a request to the system. It takes the component selection algorithm (Algorithm 1) and Figure 9 representing the sequence diagram of an example request, into account.

In this example, a user wants to know the condition of the beaches in his/her area (Almería, Spain), and he/she uses a search bar to find the category (var *cat*) “Playas” (Beaches). While the user is performing the search, the application generates the context information that is added to the header of each request sent to the Gateway Service. The context information (var *c*) that is added to the request is:

- Platform → Android (var *c.platform*).
- Form factor → Smartphone (var *c.formfact*).
- Language → Spanish (var *c.language*).
- Position → (36.8293, -2.4044) that corresponding to the (*lat, long*) of the University of Almería (var *c.position*).

Once the Gateway Service (see Figure 4) receives the request, it is redirected to the Component Service. This Component Service checks whether the user has permission for that resource by looking up the user’s role in the Auth Service. Then it translates the search term into the Component Database main language (*line 2* of Algorithm 1), which in this request is Spanish. As the database components are in English, the Translation Service provides the translation of “Playas”. When the translation has been received, the Component Service retrieves the subgraph in the translated search category (*line 6*). Using the user’s position, the subgraph containing the region components finds the region the user is in (*line 7-8*), as there may be several regions with different granularity (e.g., Spain - Andalusia - Almería) and different components and data sources.

In this example, only one region, Andalusia, is retrieved, so now only the subgraph for that region in particular is worked with (*line 9*). As the context information indicates that the user device is an Android smartphone, the service only has to retrieve the four components with the highest score (*line 12*), as explained with regard to component size constraints based on the front-end form factor platform. Once the components have been received, the service fills them from the appropriate data source. In this case, the *map component* retrieves the data from the Geoservice as a WMS, and the *category image* is shown simply as a static image, a *document* populated by a static text about the 16 best beaches of Andalusia, and a *table* with a network of devices connected to their related topics that retrieves the weather conditions interpreted by our front-end application. As a fail-safe mechanism, the data from an external service, in this case AEMET (Spanish Agency of Meteorology) is sent as backup in case there is a problem with the devices. Then, after filling all the components, the service translates them back into the context language, if possible (*line 14*), again using the Translation Service. When the request process has been completed, the Component Service issues a response to the front-end application (*line 16*).

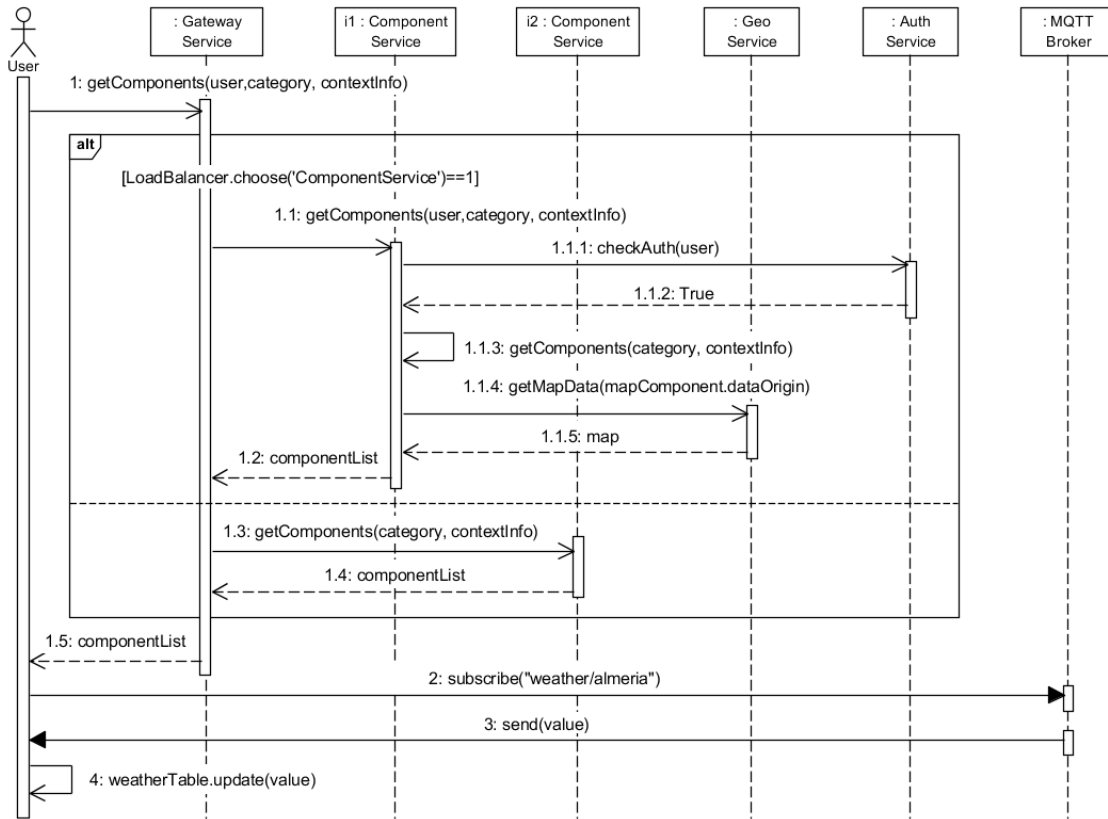


FIGURE 9. Sequence example for a Request in the geospatial PWA.

Once the application has received the response, it composes the main view with the components up to the maximum platform size, in this case size four. As one of the components is size two, only three of the four components included in the response can be used, as seen in Figure 10. The *document component* is discarded as it has a lower score than the other three. Notice that the component has a score defined for each platform as shown in Figure 6. The front-end component can be static, with the data embedded in the response given by the back end, or dynamic, like the table in which topics are connected to the one in the component response and the data received (wind speed, humidity and temperature) to show what the weather is like in real-time at those beaches.

Besides the main view, the user can access each component by pressing it longer for the full screen view. Sometimes the component contains more information in full screen mode than in the default size, because there is much more space available to work in, making it possible to increase the information shown to the user. In our example, the table component contains the current weather conditions in the user region, and in full screen mode, predictions for the next day (Figure 11a) acquired directly from AEMET are added. Another interesting feature of the application is that in a response from a category, a link to related categories is also usually sent, in this example “tapas” (Figure 11b), which can be accessed just by swiping to the left in the main view of the search category. This action makes it easy to find new

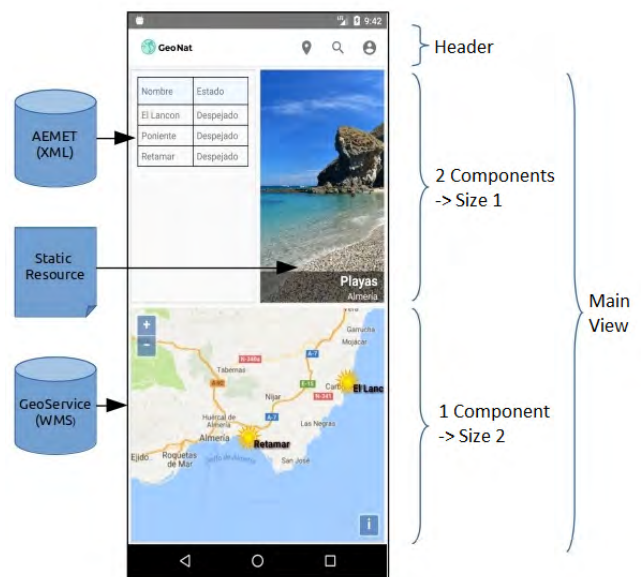


FIGURE 10. Main User view of the PWA example.

categories that could be of interest to the user, for discovery and easy access to new information.

Table 2 summarizes application behavior with different categories searched or different context information, as well as the instances where the components shown are based on IoT devices. For example, when User 3 searches for the traffic

TABLE 2. Example scenarios with different users and contexts.

User	Category	Context Information			
		Position	Language	Platform	Form Factor
Response					
User 1	Playas (Beaches)	(36.834047, -2.463714)	Spanish	Windows	Landscape
1. Map Component with the weather in Almería. 2. Table Component with the temperatures and the wind speed in Almería. (Celsius + Km/h). IoT 3. Map Component with wind speed at nearby beaches. (Km/h). IoT Related category → Tapas.					
User 2	Beaches	(36.834047, -2.463714)	English	iOS	Smartphone
1. Map Component with the weather in Almería. 2. Table Component with the temperatures and the wind speed in Almería. (Fahrenheit + M/h). IoT 3. List Component with beaches with danger flags within 100Km. Related category → Tapas.					
User 3	Tráfico (Traffic)	(36.721274, -4.421399)	Spanish	Android	Portrait
1. Map Component with color coded traffic density in Málaga. 2. List Component with accidents within 100Km. (Spanish). 3. Table Component with pollution levels in Málaga.(Air Quality Index, ICA) Related category → Pollution.					

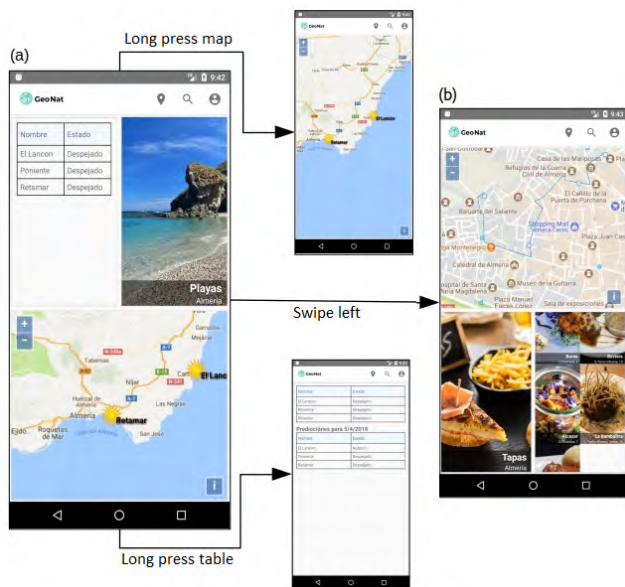


FIGURE 11. User interaction with the PWA example.

in his/her zone (Málaga) he/she receives a map component, a list of the accidents near his/her location and a table with pollution levels in the zone. “Pollution” is also sent as a possible category related to traffic that could be relevant.

This use case shows the advantages of the main features of our proposal, the use of microservices architectures and the use of a directed graph to present component information considering the context information of the user. Thanks to these improvements we can, for example, scale services to respond in times of higher demand (e.g., Geoservice in summer, in behalf of “playas” category), while in the case of those low-use services we have the possibility to reduce their level of replication to a minimum. Another of the main advantages is the possibility to include new categories or

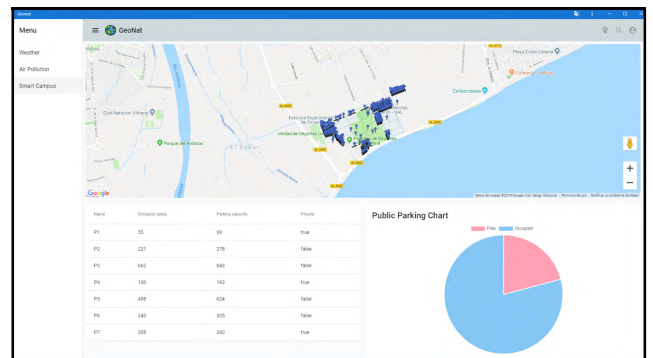


FIGURE 12. PWA Desktop interface. University of Almería.

components on the fly without affecting the performance of our application, making them available to the users as soon as they are added to our system (e.g., adding new locations to the “tapas” category).

To test some user experiences, a PWA demo called GeoNat has been implemented (available at <https://app.acg.ual.es>). This example has a static user context, so the reader can experience how the application would behave if he were near the University of Almería (Spain). The categories shown are those related to that position, “weather”, “air pollution” and the “smart campus” categories. Those categories were chosen by our selection algorithm. Figure 12 shows the Smart Campus layer for that particular context on a desktop computer user interface. This view is built around three components, a map with the university locations, a table with the free parking spaces in each section, and a chart with the total of free vs occupied spaces.

VI. CONCLUSION AND FUTURE WORK

In this article, we reported on how we were able to overcome several problems, such as developing a

microservice-based architecture, choosing the optimal number of those microservices, integrating the different technologies we used to construct them, and orchestrating them in such a way that our back end would be secure and reliable. At the same time, we show real-time data based on IoT devices in a component-based interface. Additionally, we had to build the foundations upon which components could be added and expanded based on new categories or regions to be supported.

The use of context information in this setting helps provide the user with the information that is really needed. At the same time, the use of a data model based on graphs makes our component selection algorithm quite fast, as in this setting, the relationships between entities are more important than the entities themselves. Pruning the graphs by picking subgraphs is also faster than having to relay each record by querying in a more traditional relational component database. With PWAs, the technology barrier of front-end application compatibility with mobile devices is easily overcome just by adjusting a web application and making it hybrid, so our scope could be broadened, thus reaching more users. The micro frontend approach made it possible to build the user interface dynamically and develop visual components independently, finding different ways to show the user data.

As future work, the proposed solution could still be improved upon. First, to improve the user experience, with time, we will add more categories, components and data sources. One of our next implementation targets is how to score components automatically as soon as they are added in the component service. Another field we would like to explore is how to improve the performance and the reliability of our microservice architecture even further. This can be done, for example, by optimizing the microservice scale, changing the policy that handles the traffic at the back end, creating new middleware microservices, and other solutions. One of the problems with our architecture is the fact that our MQTT broker (Mosquitto) cannot be replicated. As a possible solution we are trying to implement a middleware microservice, which could handle subscriptions based on the topic requested and redirect them to the appropriate MQTT broker instance. Focusing on the last point, we could add value to our solution by decoupling the real-time data from the MQTT protocol, in an attempt to find a solution that could handle different types of communication protocols. One last interesting field that is worth considering is making use of data analytics, as every request made in our system and the data generated by our IoT devices is a source of knowledge about both user behavior (how they use our application) and the environment (the data recovered by our IoT sensors).

REFERENCES

- [1] B. Ubaldi, "Open government data: Towards empirical analysis of open government data initiatives," Paris, France, OECD Working Papers 22, 2013.
- [2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.
- [3] E. Ahmed, I. Yaqoob, I. A. T. Hashem, I. Khan, A. I. A. Ahmed, M. Imran, and A. V. Vasilakos, "The role of big data analytics in Internet of Things," *Comput. Netw.*, vol. 129, pp. 459–471, Dec. 2017.
- [4] Y. Jeong, H. Joo, G. Hong, D. Shin, and S. Lee, "AVIoT: Web-based interactive authoring and visualization of indoor Internet of Things," *IEEE Trans. Consum. Electron.*, vol. 61, no. 3, pp. 295–301, Aug. 2015.
- [5] A. Jolma, D. P. Ames, N. Horning, H. Mitasova, M. Neteler, A. Racicot, and T. Sutton, "Chapter ten free and open source geospatial tools for environmental modelling and management," *Develop. Integr. Environ. Assessment*, vol. 3, pp. 163–180, Sep. 2008.
- [6] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. Newton, MA, USA: O'Reilly Media, 2015.
- [7] A. Sill, "The design and architecture of microservices," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 76–80, Sep/Oct. 2016.
- [8] B. Butzin, F. Golasowski, and D. Timmermann, "Microservices approach for the Internet of Things," in *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2016, pp. 1–6.
- [9] C. Granell, L. Díaz, and M. Gould, "Service-oriented applications for environmental models: Reusable geospatial services," *Environ. Model. Softw.*, vol. 25, no. 2, pp. 182–198, 2010.
- [10] P. Zhao, L. Di, and G. Yu, "Building asynchronous geospatial processing workflows with Web services," *Comput. Geosci.*, vol. 39, pp. 34–41, Feb. 2012.
- [11] P. Zhao, T. Foerster, and P. Yue, "The geoprocessing Web," *Comput. Geosci.*, vol. 47, pp. 3–12, Oct. 2012.
- [12] A. Khalili, A. Loizou, and F. van Harmelen, "Adaptive linked data-driven Web components: Building flexible and reusable semantic Web interfaces," in *Proc. 13th Int. Conf. (ESWC)*. Cham, Switzerland: Springer, 2016, pp. 677–692.
- [13] D. A. Hume, *Progressive Web Apps*. Shelter Island, NY, USA: Manning Publications, 2017.
- [14] J. Vallecillos, J. Criado, N. Padilla, and L. Iribarne, "A cloud service for COTS component-based architectures," *Comput. Standards Interfaces*, vol. 48, pp. 198–216, Nov. 2016.
- [15] F. Montesi and J. Weber, "Circuit breakers, discovery, and API gateways in microservices," 2016, *arXiv:1609.05830*. [Online]. Available: <https://arxiv.org/abs/1609.05830>
- [16] J. Lewis, "Micro services—Java, the unix way," in *Proc. 33rd Degree Conf. Java Masters*, Kraków, Poland, Mar. 2012, pp. 19–21.
- [17] F. George, "Micro service architecture," Melbourne, VIC, Australia, 2012. [Online]. Available: <https://yowconference.com/melbourne/2012/>
- [18] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Softw.*, vol. 35, no. 3, pp. 24–35, May/June. 2018.
- [19] T. Mauro. (Feb. 2015). Adopting microservices at netflix: Lessons for architectural design. NGINX Inc. Accessed: Apr. 23, 2019. [Online]. Available: <https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices>
- [20] D. Pariag and T. Brecht, "Application bandwidth and flow rates from 3 trillion flows across 45 carrier networks," in *Proc. Int. Conf. Passive Active Netw. Meas.* Cham, Switzerland: Springer, 2017, pp. 129–141.
- [21] M. Krämer, D. W. Fellner, and J. Boehm, "A microservice architecture for the processing of large geospatial data in the cloud," M.S. thesis, Dept. Comput. Sci., Technische Universität, Darmstadt, Germany, 2018.
- [22] M. Villari, A. Celesti, G. Tricomi, A. Galletta, and M. Fazio, "Deployment orchestration of microservices with geographical constraints for edge computing," in *Proc. IEEE Symp. Comput. Commun.*, Jul. 2017, pp. 633–638.
- [23] P. Volland and H. Asche, "Geospatial visualization of automotive sensor data: A conceptual and implementational framework for environment and traffic-related applications," in *Proc. Int. Conf. Comput. Sci. Appl.* Cham, Switzerland: Springer, 2017, pp. 626–637.
- [24] M. A. Jarwar, M. G. Kibria, S. Ali, and I. Chong, "Microservices in Web objects enabled IoT environment for enhancing reusability," *Sensors*, vol. 18, no. 2, p. 352, 2018.
- [25] P. Bak, R. Melamed, D. Moshkovich, Y. Nardi, H. Ship, and A. Yaeli, "Location and context-based microservices for mobile and Internet of Things workloads," in *Proc. IEEE Int. Conf. Mobile Services (MS)*, Jun./Jul. 2015, pp. 1–8.
- [26] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-S—A publish/subscribe protocol for wireless sensor networks," in *Proc. 3rd Int. Conf. Commun. Syst. Softw. Middleware Workshops (COMSWARE)*, Jan. 2008, pp. 791–798.

- [27] J. Vallecillos, J. Criado, L. Iribarne, and N. Padilla, "Dynamic mashup interfaces for information systems using widgets-as-a-service," in *On The Move To Meaningful Internet Systems OTM 2014 Workshops* (Lecture Notes in Computer Science), vol. 8842. Berlin, Germany: Springer, 2014, pp. 438–447.
- [28] J. Criado, D. Rodríguez-Gracia, L. Iribarne, and N. Padilla, "Toward the adaptation of component-based architectures by model transformation: Behind smart user interfaces," *Softw., Pract. Exper.*, vol. 45, no. 12, pp. 1677–1718, 2015.
- [29] B. Frankston, "Progressive Web apps [bits versus electrons]," *IEEE Consum. Electron. Mag.*, vol. 7, no. 2, pp. 106–117, Mar. 2018.
- [30] R. Fransson and A. Driaguine, "Comparing progressive Web applications with native Android applications: An evaluation of performance when it comes to response time," Ph.D. dissertation, Dept. Comput. Sci., Linnaeus Univ. Växjö, Sweden, 2017. Accessed: Apr. 23, 2019. [Online]. Available: <http://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-64764>
- [31] M. Geers. *Micro Frontends*. Accessed: Apr. 23, 2019. [Online]. Available: <https://micro-frontends.org>
- [32] M. Santoni. (2018). *Progressive Web Apps: Feature Compatibility Based on the Browser*. [Online]. Available: <https://bit.ly/2N1YIBL>
- [33] G. Chen and D. Kotz, "A survey of context-aware mobile computing research," Dept. Comput. Sci., Dartmouth College, Hanover, NH, USA, Tech. Rep. TR2000-381, 2000.
- [34] Q. Zhu, S. Wang, B. Cheng, Q. Sun, F. Yang, and R. N. Chang, "Context-aware group recommendation for point-of-interests," *IEEE Access*, vol. 6, pp. 12129–12144, 2018.
- [35] J. Lewis and M. Fowler. (2014). *Microservices*. [Online]. Available: <https://www.martin.fowler.com/articles/microservices.html>
- [36] Docker. *Official Docker Documentation*. Accessed: Apr. 23, 2019. [Online]. Available: <https://docs.docker.com>
- [37] Docker Hub. Accessed: Apr. 23, 2019. [Online]. Available: <https://hub.docker.com>
- [38] Netflix. Accessed: Apr. 23, 2019. [Online]. Available: <https://netflix.github.io>
- [39] C. Richardson. *Microservice Architecture Patterns and Best Practices*. Accessed: Apr. 23, 2019. [Online]. Available: <http://microservices.io>
- [40] D. Hardt, *The OAuth 2.0 Authorization Framework*, document RFC 6749, 2012.
- [41] P. J. Sadalage and M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. London, U.K.: Pearson, 2013.
- [42] REDIAM. *Environmental Information Network of Andalusia (Spain)*. Accessed: Apr. 23, 2019. [Online]. Available: <http://www.juntadeandalucia.es/medioambiente/rediam>
- [43] AEMET. *Spanish Agency of Meteorology*. Accessed: Apr. 23, 2019. [Online]. Available: <http://www.aemet.es>
- [44] MySQL. Accessed: Apr. 23, 2019. [Online]. Available: <https://www.mysql.com>
- [45] PostGIS. *Spatial and Geographic Objects for PostgreSQL*. Accessed: Apr. 23, 2019. [Online]. Available: <https://postgis.net>
- [46] MongoDB Atlas. Accessed: Apr. 23, 2019. [Online]. Available: <https://www.mongodb.com>
- [47] Neo4j Spatial. Accessed: Apr. 23, 2019. [Online]. Available: <https://neo4j-contrib.github.io/spatial>



MANEL MENA received the master's degree in computer engineering from the University of Almería. He is currently pursuing the Ph.D. degree, with a focus on the Internet-of-Things (IoT) systems, software architectures, and the Web of Things. He has been a member of the Applied Computing Group (TIC-211), University of Almería, since 2016. He works alongside his peers in the national research project CoSmart (TIN2017-83964-R). Since 2018, he has been supported by an FPU Grant (FPU17/02010). His research interests include data engineering, software engineering, big data, cloud computing, machine learning, the IoT, and the Web of Things.



ANTONIO CORRAL received the Ph.D. degree in computer science from the University of Almería, Spain, in 2002, where he is currently an Associate Professor with the Department of Informatics. He has participated actively in several research projects in Spain, such as CoSmart, INDALOG, vManager, and ENIA, and in Greece, such as CHOROCHRONOS and ARCHIMEDES. He has published in referred scientific international journals, such as *Data & Knowledge Engineering*, *Geoinformatica*, *The Computer Journal*, *Information Sciences*, *the Journal of Systems and Software*, *Computer Standards & Interfaces*, *Knowledge-Based Systems*, *Computing*, and UAIS, conferences, such as SIGMOD, SSD, ADBIS, SOFSEM, PADL, DEXA, OTM, MEDI, and SAC, and book chapters. His main research interests include access methods, algorithms, query processing, spatial databases, and distributed query processing.



LUIS IRIBARNE received the B.S. and M.S. degrees in computer science from the University of Granada and the Ph.D. degree in computer science from the University of Almería and conducted from the University of Málaga, Spain. From 1991 to 1993, he was a Lecturer with the University of Granada and collaborated as an IT Service Analyst at the University School of Almería. Since 1993, he has been a Lecturer with the Advanced College of Engineering, University of Almería. From 1993 to 1999, he has participated in several national and international research projects on distributed simulation and geographic information systems. Since 2006, he has been serving as the Main Coordinator of five Research and Development projects funded by the Spanish Ministry of Science and Technology and the Andalusian Ministry ST. In 2007, he has founded the Applied Computing Group (ACG). He has also acted as an Evaluator for funding agencies in Spain and Argentina. He is currently an Associate Professor with the Department of Informatics, University of Almería. He has published in referred JCR scientific international journals, such as *The Computer Journal*, *Computer Standards & Interfaces*, *the Journal of Logical and Algebraic Methods in Programming*, *Software Practice and Experience*, *Simulation Modelling Practice and Theory*, the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, *the Journal of Neuroscience Methods*, *Information Systems Management*, *Behavioural Brain Research*, *Computers in Industry*, the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS, and *the Journal of Visual Languages and Computing*, among others. He has also published in referred scientific international conferences, such as ICMT, ICSOC, ICSSOFT, SOFSEM, ICAART, PAAMS, SEAA, and EUROMICRO, among others, and book chapters. His main research interests include simulation and modeling, model-driven engineering, machine learning, and software technologies and engineering.



JAVIER CRIADO received the degree in computer science engineering and the master's degree in advanced computer techniques from the University of Almería, Spain, where he is currently pursuing the Ph.D. degree in CS. In 2009, he joined the Applied Computing Group (TIC-211), a research group in computer science at the University of Almería. Since 2009, he has participated in three national research projects (TIN2007-61497, TIN2010-15588, TIN2013-41576-R, and TIN2017-83964-R) and a regional research project (P10-TIC6114). From 2011 to 2015, he was supported by an FPU Grant (AP2010-3259). His research interests include model-driven engineering, component-based software engineering, model transformations, mashups, model-driven development for advanced user interfaces, COTS components, trading, agents and multi-agent systems, ontology-driven engineering, and UML design.

•••