



Desarrollo de una aplicación web y móvil para la monitorización, notificación, visualización y predicción de la calidad del aire.

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Juan Bernal Miralles

Tutor/es:

Raquel Martínez España

María Del Carmen Garrido Carrera

26 de Junio de 2024



Facultad
Informática
Universidad
Murcia

Desarrollo de una aplicación web y móvil para la monitorización, notificación, visualización y predicción de la calidad del aire.

Autor

Juan Bernal Miralles

Tutor/es

Raquel Martínez España

Departamento de Ingeniería de la Información y las Comunicaciones

María Del Carmen Garrido Carrera

Departamento de Ingeniería de la Información y las Comunicaciones



Grado en Ingeniería Informática



UNIVERSIDAD DE
MURCIA



Murcia, 26 de Junio de 2024

Preámbulo

Desde un primer momento, mi voluntad siempre era la de realizar este trabajo fin de grado. Los principales motivos que me llevaron a elegir este tema fueron la intención de aprender sobre una plataforma para el desarrollo de aplicaciones inteligentes en entornos reales como es FIWARE. Además, me pareció interesante la posibilidad de aplicar los conocimientos adquiridos durante la carrera en un proyecto real.

Por otro lado, el tema de la monitorización de la calidad del aire me pareció interesante, ya que es un tema de actualidad y de gran importancia para la sociedad. Además, la posibilidad de trabajar con API de obtención de datos y la creación de un modelo de predicción me pareció un reto interesante.

Agradecimientos

Es para mí un placer poder expresar mi agradecimiento a todas aquellas personas que han hecho posible la realización de este trabajo. En primer lugar, quiero agradecer a mis familiares por haberme apoyado en todo momento, especialmente a mis padres, por su paciencia y implicación en mi formación académica. También quiero agradecer a mis compañeros y amigos de clase, por hacer más llevadera la etapa universitaria y ayudarme en todo lo que han podido.

Por otro lado, quiero agradecer a mis tutoras, Raquel Martínez España y María del Carmen Garrido Carrera, por su profesionalidad y dedicación en la realización de este trabajo, así como por su paciencia y apoyo en todo momento. También quiero agradecer a todos los profesores que me han enseñado durante mi etapa universitaria, por su dedicación y esfuerzo en la formación de los alumnos.

Declaración firmada sobre originalidad del trabajo

D. Juan Bernal Miralles, estudiante de la titulación de **Grado en Ingeniería Informática** de la Universidad de Murcia y autor del TF titulado “**Desarrollo de una aplicación web y móvil para la monitorización, notificación, visualización y predicción de la calidad del aire.**”.

De acuerdo con el Reglamento por el que se regulan los Trabajos Fin de Grado y de Fin de Máster en la Universidad de Murcia (aprobado C. de Gob. 30-04-2015, modificado 22-04-2016 y 28-09-2018), así como la normativa interna para la oferta, asignación, elaboración y defensa de los Trabajos Fin de Grado y Fin de Máster de las titulaciones impartidas en la Facultad de Informática de la Universidad de Murcia (aprobada en Junta de Facultad 27-11-2015)

DECLARO:

Que el Trabajo Fin de Grado presentado para su evaluación es original y de elaboración personal. Todas las fuentes utilizadas han sido debidamente citadas. Así mismo, declara que no incumple ningún contrato de confidencialidad, ni viola ningún derecho de propiedad intelectual e industrial

Murcia, a 26 de Junio de 2024



Fdo.: Juan Bernal Miralles
Autor del TF

Resumen

Uno de los problemas más importantes en la actualidad es el aumento de la contaminación del aire en el mundo. Es un problema global que afecta de forma grave a la salud de las personas y al medio ambiente. De ahí la importancia de visualizar, analizar, alertar, predecir y controlar la calidad del aire. A su vez, el aumento de tecnologías y software para la recopilación de datos a través de sensores y dispositivos IoT está permitiendo minimizar este problema. Una de estas tecnologías es la plataforma FIWARE, que proporciona una arquitectura abierta basada en estándares para el desarrollo de aplicaciones inteligentes y sostenibles. En este trabajo se ha desarrollado una aplicación multiplataforma que permite visualizar, alertar y predecir la calidad del aire con el uso de la plataforma FIWARE.

Este Trabajo Fin de Grado documenta todo el proceso seguido para la puesta en marcha de esta aplicación. En primer lugar, se ha realizado un estudio de las tecnologías y herramientas necesarias para el desarrollo de la aplicación. También una comparativa de las diferentes tecnologías disponibles. Las decisiones tomadas son las siguientes:

Spring Boot para el desarrollo del backend, React como lenguaje del frontend, Python para el desarrollo de una machine learning, MySQL se encargará de la persistencia de datos y muy importante, la utilización de la plataforma FIWARE.

En segundo lugar, una vez seleccionadas las tecnologías, se ha procedido a establecer los objetivos y metodología de trabajo para el proyecto que se ha desglosado en varias fases. A continuación, se explica el diseño de la aplicación, donde se han descrito los casos de uso y sus actores; la arquitectura de Fiware y sus modelos; el diseño de la base de datos; la arquitectura de la aplicación y un prototipo de la aplicación tanto en su versión móvil como de escritorio. En la estructura de la aplicación se ha seguido los métodos de arquitectura limpia y el estilo arquitectónico REST con una tabla con todas las rutas de la API.

En la fase de implementación se describe el desarrollo de cada componente de la aplicación. El componente de backend, desarrollado en Spring Boot, es el encargado de gestionar la lógica de negocio y la comunicación con los demás componentes. En la siguiente fase, pruebas y despliegue, se han realizado las pruebas unitarias, integración y sistema. De mismo modo se ha desplegado la aplicación haciendo uso de contenedores Docker para todos los componentes de la misma. Al final, se ha mostrado las conclusiones del trabajo realizado y se han propuesto futuras líneas de trabajo.

Extended Abstract

1. Introduction

One of the most important problems today is the increase in air pollution in the world. Air pollution is a global problem that seriously affects people's health and the environment. With the increase in this problem, we have sought ways to alleviate or minimize its effects. One of the ways to do this is through visualization, analysis, alerting, prediction and control of air quality through computer applications. In turn, the increase in technologies and software for data collection through sensors and IoT devices encourage these developments. One of these technologies is the FIWARE platform. It is an open innovation and entrepreneurship ecosystem based on the internet technologies of the future. This ecosystem is rapidly becoming a base platform for the development of intelligent services internationally.

Taking into account the importance of air quality and the FIWARE platform, in this work a multi-platform application has been developed that allows visualizing, notifying and predicting air quality with the use of the FIWARE platform. The developed application has been called AirQualityNest, it will use an API to obtain air quality data.

2. State of the art

Currently, there are different applications that allow you to view air quality in real time. We can group these applications into two types:

- **Applications specialized in air quality:** These are applications that focus on air quality and offer detailed information about pollutants and air quality in real time.
- **Applications that include air quality as an additional functionality:** These are applications that offer information about the weather and air quality as an additional functionality.

In the first group, we find applications such as IQAir AirVisual, Plume Labs, among others. In the second group, we find applications such as AccuWeather or Aemet.

As tools for development, different programming languages have been analyzed, but the ones selected are:

- **Java:** It is a general-purpose, concurrent, object-oriented programming language that has been used for backend development.
- **Python:** It is an interpreted, high-level, multi-paradigm programming language that has been used for the development of the machine learning model.
- **React:** It is an open source framework created by Facebook that has been used for frontend development.

As a development environment, one that supports all the programming languages used has been selected, such as Visual Studio Code.

A Docker and Docker Compose container system has been used to deploy the application. Data persistence has been carried out in a MySQL relational database. Obtaining air quality data has been done through Google's Air Quality API, which allows a limited number of free requests. This API provides real-time air quality data with an accuracy of 1 hour. It gives us information about the air quality index and the following pollutants: CO (carbon monoxide), NO₂ (nitrogen dioxide), O₃ (ozone), PM₁₀ (10 micron particles), PM_{2.5} (2.5 micron particles) and SO₂ (sulfur dioxide).

3. Objectives and work plan analysis

The general objective of this work is the development of a web and mobile application for the notification, visualization and prediction of air quality. To this end, the following objectives have been set, which have been broken down into phases:

- **OB1** Research and study FIWARE technology.
 - **OB2** Obtain and study APIs for air quality.
 - **OB3** Study technologies and tools for the development of cross-platform applications.
 - **OB4** Analyze the system and raise requirements.
 - **OB5** Design the application architecture.
 - **OB6** Design the data model.
 - **OB7** Develop from the back-end.
 - **OB8** Develop front-end.
 - **OB9** Develop machine learning for prediction.
 - **OB10** Integrate from the application.
-

- **OB11** Validate the application in a test environment.
- **OB12** Document the cross-platform application.

Likewise, this section has had its work planned thanks to the clickup platform, which allows organization in task list format, kanban board, Gantt chart, among others.

4. Design of the work carried out

In this section, the architecture of the application has been designed, the use cases and the data model have been defined. It has been decided that there are three user roles:

- **Observer User:** Corresponds to the user external to the application who can view a summary of the air quality in their location, register and log in.
- **Participatory User:** You must register and log in to view air quality, see news, receive alerts, update your data and view the air quality prediction for the next 7 days.
- **Manager:** He is responsible for managing the news and users of the application.

For this project, the Orion Context Broker will be used, which is a middleware that allows context management and data publication in real time. This component needs a MongoDB database to store the data. The OCB has a REST API that allows you to interact with it, it also has a catalog of entities. This catalog contains different categories of entities. Some of these categories are:

- **Smart City:** Contains entities related to the smart city.
- **Energy:** Entities related to energy.
- **Environment:** Entities related to the environment.

In the Environment category we find the AirQualityObserved entity of the NGSI-V2 standard to represent the observed air quality. Likewise, the Orion Context Broker subscription system will be used for alert notification.

A class diagram for the application data model and a data persistence diagram have been designed and explained. In the data model, the entities and their attributes have been defined, as well as the relationships between them. In these diagrams it is not shown to the observing user since it does not need to store data in the database. The persistence diagram has been obtained directly from the MySQL database thanks to the MySQL Workbench tool.

There are 4 main components in the application architecture:

- **Frontend:** Developed with React, it is responsible for interacting with the user and communicates with the quality API with the endpoints it provides.
- **Quality API:** Programmed in Java with Spring Boot, it is responsible for the business logic and communication with the other components.
- **Prediction API:** Corresponds to a machine learning model developed in Python, it offers the prediction of air quality for the next 7 days to the quality API.
- **FIWARE:** Orion Context Broker and MongoDB are responsible for context management and alert notification.

All these components communicate with each other through HTTP requests and in the case of FIWARE, through NGSI-V2. The development of these components has been carried out following clean architecture methods with SOLID principles. The REST architectural style has been used in the application using the GET, POST, PUT, DELETE and PATCH methods. Additionally, a routing table has been created for the Quality API. Finally, three prototypes of the application have been designed with the Canva tool, both for the web version and the mobile version. These prototypes are:

- **Registration page:** Corresponds to the registration page of the application that is shown to the observing user.
- **Prediction page:** It is a first version of the air quality prediction page corresponding to the participatory user.
- **Air quality page:** The only prototype in mobile version that shows the complete air quality for the participatory user.

5. Development and implementation of the application

In this section the development of the application has been explained. It has been divided into different components and the development of each of them has been explained.

5.1. Air Quality API

The Air Quality API is a REST API developed in JAVA using the Spring Boot framework. This API is the main element of the application, responsible for communication with all other backend services and offers the necessary endpoints for the frontend. This API has a hexagonal architecture, which is divided into 5 layers:

- **Presentation Layer:** It is responsible for receiving HTTP requests and sending HTTP responses. It is made up of API controllers and exception handlers.
-

- **Business logic layer:** Corresponds to the API services, contains the business logic of the application. It is important to note that in this layer there is a service called `ServicioCoordinadorAirQuality` in charge of satisfying users with air quality in real time and a timer that is limited to obtaining air quality data for registered users each day of the Google API. To group coordinates, the latitude and longitude values have been represented as integers to group them by degrees. One of the most important services is the User Service, this service is responsible for managing the users of the application. This has been divided into an abstract service and two concrete services, one for participatory users and another for managers.
- **Persistence layer:** Contains all the application repositories, these extend the `JpaRepository` interface of Spring Data JPA. In addition to the default methods by JPA, custom methods have been created to obtain data from the database.
- **Domain Layer:** All application entities are located in this layer. They are annotated with Lombok's `@Data` for the generation of the getter and setter methods. These have JPA annotations for the creation of the tables in the database and the relationships between them. Persistent entities are identified with a UUID type identifier that is automatically generated in the database. Not all classes have an identifier but some are embedded in others.
- **Support Layer:** Contains different types of classes that support the other layers of the application. These classes are the DTOs, the mappers, generic error with its factory, data validation and the classes in charge of the security of the application. JWT with Spring Security has been used for security.

5.2. Air Quality Prediction API

The prediction API is a machine learning model developed in Python. This model has been trained with air quality data obtained from the Google API during the development of the application. This should return a prediction of the air quality index for the next 7 days at specific coordinates. The libraries used for the development of the model were pandas, numpy, scikit-learn, flask, SQLAlchemy and pickle. The two main components of the prediction API are:

- **model.py:** In this file is the train function, which is responsible for creating, training and saving the machine learning model.
- **api.py:** Sew locates the endpoint that receives the coordinates and returns the air quality prediction for the next 7 days.

Different machine learning algorithms have been tested for predicting air quality, such as linear regression or Support Vector Regression, but the algorithm that has given the

best results is Random Forest Regressor. The characteristics for the model are latitude, longitude, day of the week, month, year and to improve the prediction the air quality index for the previous 6 days has been added.

To estimate the best characteristics, a small study has been carried out. The model has been trained with 90% of the data and tested with the remaining 10% giving the best result with a mean absolute error of 3.845. This result is acceptable since the air quality index ranges from 0 to 100 and the amount of data used for training has been limited. It is important to note that this component is not the main element of the application, but rather a complementary service.

5.3. Fiware Configuration

For the application to function correctly, a subscription to Orion Context Broker is required to notify the Air Quality API when an alert occurs. This happens when the air quality index is less than 40.

5.4. Frontend

It is the application component responsible for interaction with the user. For its development, different libraries have been used to develop the frontend of the application. This component is a Single Page Application, which allows for shorter loading times and a better user experience. These are:

- **React:** It is the programming language used for the development of the frontend of the application. Which allows the development of cross-platform applications.
- **Vite:** It is a module packager that allows fast loading of the application.
- **Bootstrap:** Thanks to this library it has been possible to develop a responsive design of the application quickly.
- **React Router Dom:** Allows navigation between the different pages of the application and is responsible for making the application a Single Page Application.
- **Ag-Grid:** Library used for creating tables in the application.
- **Echarts:** Allows the creation of charts in the application.
- **Prop-Types:** It has been used for validation of component props.

For better organization it has been divided into different packages:

- **Components:** The jsx components of the application are found. These are used in different parts of the application by other jsx components or pages.
-

- **Pages:** Each item in this folder corresponds to a page of the application. They are jsx elements that contain the components of the components folder and different functionalities.
- **Security:** These elements perform authentication and access control to the application.
- **Others:** There are other packages such as styles, configuration, assets, utils and hooks.

Next, a navigation map has been developed and all the pages of the application divided by users with a description of each of them. These are:

- **Unauthenticated**
 - **AirQuality summary:** Displays a summary of the air quality at the user's location.
 - **Registration:** Allows the user to register in the application.
 - **Login:** Allows the user to log in to the application.
- **Authenticated as user**
 - **AirQuality:** Displays the air quality at the user's location. This contains a list of contaminants and some recommendations.
 - **Alerts:** Contains a list of air quality alerts that the user has received.
 - **Prediction:** A list of air quality prediction for the next 7 days. Below the list is a legend with the colors of the prediction.
 - **Update user:** Allows the user to modify their data.
- **Authenticated as manager**
 - **Users:** corresponds to a table with the participating users of the application. In this table you can view user data and delete it.
 - **Create manager:** Allows the manager to create a new manager by entering their data.
 - **Update manager:** Allows the manager to modify its data.

It has not been possible to develop the news functionality in this component due to lack of time.

6. Testing and deployment

White box testing has been carried out on the air quality API. These are unit, integration and system tests. JUnit was used for the unit tests and the other components of the application were mocked. JUnit has also been used for integration tests, these with an H2 in-memory database. Finally, system tests have been carried out with Visual Studio Code's Thunder Client.

To deploy the application, a docker-compose.yml file has been created, which contains the application services and the networks necessary for its operation. For each of the three services developed, a Dockerfile file has been created, which contains the instructions for creating the Docker image.

7. Conclusions and future work

I am satisfied with the work done, since I have managed to develop a multi-platform application that allows you to visualize, notify and predict air quality. It has been a very complete job and I have learned a lot during the development of the application. Although it is important to highlight that this project is the starting point for future improvements and expansions.

Among those improvements that can be made in the application we have the following:

- **Improve air quality prediction:** The machine learning model can be improved to make the prediction more accurate. The drawback is that it is necessary to have more data to train the model.
 - **Improve and add new features:** It can be modified to be a microservices application and add new features such as news.
 - **Implement a monitoring system:** Give users the possibility of purchasing an IoT device to monitor air quality in their location.
-

Índice general

Resumen	xi
Extended Abstract	xiii
1. Introduction	xiii
2. State of the art	xiii
3. Objectives and work plan analysis	xiv
4. Design of the work carried out	xv
5. Development and implementation of the application	xvi
5.1. Air Quality API	xvi
5.2. Air Quality Prediction API	xvii
5.3. Fiware Configuration	xviii
5.4. Frontend	xviii
6. Testing and deployment	xx
7. Conclusions and future work	xx
1. Introducción	1
2. Estado del arte	3
2.1. Aplicaciones similares	3
2.1.1. IQAir-AirVisual	3
2.1.2. Plume Labs	5
2.1.3. Accuweather	6
2.1.4. Aemet	7
2.1.5. Comparación de aplicaciones similares	8
2.2. Tecnologías y herramientas	9
2.2.1. Lenguajes de programación	9
2.2.2. Bases de datos	13
2.2.3. Entornos de despliegue	14
2.2.4. Entornos de desarrollo	15
2.2.5. Api obtención de datos de calidad del aire	16
3. Análisis de objetivos y plan de trabajo	19
3.1. Análisis de objetivos	19
3.2. Planificación temporal	21

4. Diseño del trabajo realizado	23
4.1. Requisitos de la aplicación	23
4.1.1. Casos de Uso: Usuario Observador	24
4.1.2. Casos de Uso: Usuario Participativo	25
4.1.3. Casos de Uso Gestor	26
4.2. Fiware	27
4.2.1. ¿Qué es Fiware?	27
4.2.2. Orion Context Broker	28
4.2.3. NGSIv2	28
4.2.4. Modelo NGSI	29
4.3. Modelado del Dominio	35
4.4. Arquitectura de la aplicación	37
4.4.1. Componentes de la aplicación	37
4.4.2. Arquitectura limpia	39
4.4.3. Arquitectura REST	40
4.5. Prototipo de la aplicación	42
5. Desarrollo e implementación de la aplicación	45
5.1. Desarrollo de la API de Calidad del Aire	45
5.1.1. Descripción	45
5.1.2. Arquitectura	45
5.1.3. Capa de Presentación	46
5.1.4. Capa de Lógica de Negocio	46
5.1.5. Capa de Persistencia	50
5.1.6. Capa de Dominio	51
5.1.7. Capa de Soporte	51
5.2. Desarrollo de la API de predicción de la calidad del aire	53
5.2.1. Descripción	53
5.2.2. Estructura y elementos de la API	54
5.2.3. Análisis de datos y entrenamiento del modelo	55
5.3. Configuración de Fiware	56
5.4. Desarrollo del Frontend	56
5.4.1. Descripción	56
5.4.2. Herramientas y librerías utilizadas	57
5.4.3. Estructura y elementos del Frontend	59
5.4.4. Mapa de navegación	61
5.4.5. Páginas de la aplicación	61
6. Pruebas y despliegue	75
6.1. Pruebas	75
6.1.1. Pruebas unitarias	75
6.1.2. Pruebas de integración	75

6.1.3. Pruebas de sistema	76
6.2. Despliegue	76
7. Conclusiones y vías futuras	79
7.1. Conclusiones	79
7.2. Vías futuras	80
Bibliografía	83
A. Anexo	87
A.1. Modelo de AirQualityObserved	87
A.2. Suscripción a Fiware	88
A.3. Pruebas	89
A.3.1. Pruebas unitarias	89
A.3.2. Pruebas de integración	89
A.4. Docker	90
A.4.1. Docker-compose	90
A.4.2. Dockerfile	91

Índice de figuras

2.1. IQAir pagina principal	3
2.2. Opción de la aplicación móvil de AirVisual	4
2.3. Página web de PlumeLabs	5
2.4. Página web de Accuweather	6
2.5. Página web de Aemet	7
2.6. Logo de Python	10
2.7. Logo de Java	11
2.8. Logo de React	12
2.9. Logo de MongoDB	13
2.10. Logo de Docker	14
2.11. Logo de Visual Studio Code	15
2.12. Logo de Meteosource	16
3.1. Lista de objetivos	21
3.2. Diagrama de Gantt	22
4.1. Casos de Uso del usuario observador	24
4.2. Tarjetas de los casos de uso del usuario observador	24
4.3. Casos de Uso del usuario participativo	25
4.4. Tarjetas de los casos de uso del usuario participativo	25
4.5. Casos de Uso del gestor	26
4.6. Tarjetas del los casos de uso del gestor	26
4.7. Arquitectura de Orion Context Broker	28
4.8. Especificación en OpenAPI de las operaciones sobre entidades en NGSI [1]	29
4.9. Especificación en OpenAPI de las operaciones sobre subscripciones en NGSI	29
4.10. Elementos principales del modelo de datos	30
4.11. Modelo del dominio	35
4.12. Modelo de persistencia	36
4.13. Diseño de la arquitectura del proyecto	38
4.14. Estructura de la arquitectura limpia	39
4.15. Prototipo de la ventana de registro	42
4.16. Prototipo de la predicción de la calidad del aire	43
4.17. Prototipo de la calidad del aire para el usuario participativo	44

5.1. Diagrama de secuencia de la obtención de la calidad del aire	48
5.2. Diagrama de secuencia de la llamada interna del Scheduled	49
5.3. Diagrama de secuencia de la generación de alertas	49
5.4. Diagrama de secuencia de la obtención de la predicción de la calidad del aire	50
5.5. Diagrama de navegabilidad de la aplicación	61
5.6. Mensaje en formato de alerta	61
5.7. Barra de navegación de la aplicación	62
5.8. Página de la calidad del aire para hoy	63
5.9. Página de inicio de sesión de la aplicación	64
5.10. Página de registro de usuario participativo	65
5.11. Página de actualidad de la aplicación	67
5.12. Página de predicción de la aplicación	69
5.13. Página de alertas de la aplicación	70
5.14. Página de perfil de la aplicación	71
5.15. Tabla de usuarios de la aplicación	72
5.16. Modal de eliminación de usuario	73
5.17. Página de creación de un usuario gestor	73

Índice de tablas

2.1.	Comparación de aplicaciones similares	8
2.2.	Comparación de lenguajes	12
2.3.	Comparación bases de datos	14
4.1.	Descripción de los campos una suscripción en NGSiv2	31
4.2.	Descripción de los campos de la entidad AirQualityObserved. La columna "O/Op" indica si el campo es obligatorio (O) u opcional (Op). "R" significa que el campo no puede ser modificado una vez creado. "O*" significa que el campo es obligatorio si no se ha proporcionado el campo address/location respectivamente.	33
4.3.	Operaciones REST de la aplicación	41
5.1.	Resultados de la predicción de la calidad del aire	55

Índice de Códigos

A.1. Ejemplo modelo AirQualityObserved del catalogo Envoironment de FI-WARE	87
A.2. Código de suscripción a Fiware para recibir alertas de calidad del aire .	88
A.3. Código de una prueba unitaria de AirQualityService para obtener la calidad del aire más reciente en una coordenada especifica	89
A.4. Código de una prueba de integración de AirQualityRepository para obtener la calidad del aire por id	89
A.5. Código del archivo Docker-compose del proyecto para desplegar los servicios de este	90
A.6. Código del archivo Dockerfile para la API de calidad del proyecto . . .	91

1. Introducción

Uno de los mayores riesgos ambientales que existen para la salud es la contaminación del aire [2]. El empeoramiento del aire ambiental en las ciudades y zonas rurales de todo el mundo provoca cada año 4,2 millones de muertes prematuras; esta mortalidad se debe a la exposición a la materia particulada fina, que causa enfermedades cardiovasculares y respiratorias, así como cánceres[2]. Una de las medidas recomendadas para reducir las emisiones atmosféricas y el riesgo de la contaminación para la salud es la monitorización, notificación y verificación de emisiones. Para este aspecto los dispositivos Internet de las cosas (IoT) son necesarios para la supervisión/monitorización de la calidad del aire [3].

Por otra parte, tenemos el avance en las ciudades que buscan aumentar la calidad de vida de los ciudadanos, optimizar la gestión de los recursos e implementar nuevas tecnologías de la información y comunicación. Bajo este concepto surgen las ciudades inteligentes, siendo una de sus bases las Tecnologías de la Información y las Comunicaciones (TIC). La evolución de la tecnología y sus aplicaciones ha sumado nuevas posibilidades a las ciudades inteligentes que llegan de la mano de sistemas IoT, algoritmos, big data, inteligencia artificial o blockchain [4].

Con el avance tecnológico y la necesidad de modernizar las ciudades surgió Future Internet Ware (FIWARE). FIWARE es una iniciativa, plataforma y conjunto de tecnologías de código abierto, que nació bajo el paraguas del VII Programa Marco de la Unión Europea en colaboración público-privada para el desarrollo del Internet del Futuro [5]. En un primer momento FIWARE fue pensada para el dominio de ciudades inteligentes, pero también se aplica a puertos y fabricas inteligentes, entre otros. Cualquiera aplicación Smart se caracteriza por recoger información relevante para la aplicación de diferentes fuentes sobre lo que está pasando en un momento dado [6].

El proyecto se basa en el desarrollo de una aplicación multiplataforma para la monitorización, notificación, visualización y predicción de la calidad del aire. Este proyecto hará uso de las diferentes tecnologías de FIWARE. Se utilizará una Interfaz de Programación de Aplicaciones (API) para la obtención de datos simulando un dispositivo IoT. El nombre de la aplicación es **AirQualityNest**.

El resto del documento se divide en las siguientes secciones:

Sección 2: Estado del arte. En esta sección se realiza una investigación sobre aplicaciones similares y un estudio de las tecnologías y herramientas que se sirven para el proyecto. Se exponen tablas comparativas de las diferentes opciones así como la explicación de la elección de las tecnologías y herramientas utilizadas.

Sección 3: Análisis de objetivos y metodología. En esta se establecen los objetivos del proyecto y se detalla la metodología de trabajo que se va a seguir, así como la planificación del proyecto a través de un diagrama de Gantt.

Sección 4: Diseño del trabajo. Se detallan los casos de uso y los usuarios del proyecto, el diseño de la arquitectura de la aplicación, bases de datos y un prototipo de la aplicación.

Sección 5: Desarrollo e implementación de la aplicación Se explica la implementación y configuración de los diferentes componentes de la aplicación.

Sección 6: Pruebas y Despliegue Se describe las diferentes pruebas realizadas y el despliegue de la aplicación.

Sección 7: Conclusiones y vías futuras En este apartado se presentan las conclusiones del proyecto y posibles vías futuras.

2. Estado del arte

El estudio del estado del arte en el contexto del proyecto requiere analizar aplicaciones similares y, por otro lado, el análisis y comparación de las diferentes tecnologías disponibles.

2.1. Aplicaciones similares

2.1.1. IQAir-AirVisual

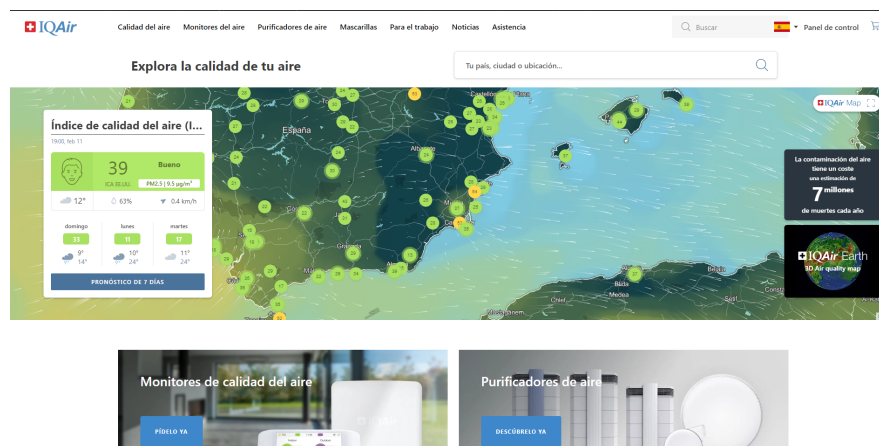


Figura 2.1: IQAir pagina principal

IQAir[7] fue fundada en 1963 por los hermanos Manfred y Klaus Hammes, quienes introdujeron un sistema de filtro de aire para hornos de carbón residenciales en Alemania.

Actualmente, IQAir es una empresa suiza de tecnología de calidad del aire, especializada en la protección contra contaminantes transportados por el aire, desarrollando productos de limpieza y control de la calidad del aire.

IQAir es una multiplataforma, que tiene una web llamada iqair.com y una aplicación llamada AirVisual. En la Figura 2.1 IQAir pagina principal se observa que la web nos proporciona información sobre la calidad del aire en tiempo real, pronósticos y

recomendaciones para mejorar la calidad del aire. La aplicación permite a los usuarios ver la calidad del aire en tiempo real para más de 10 000 ciudades en todo el mundo. La aplicación también proporciona pronósticos de calidad del aire para los próximos 7 días y recomendaciones para mejorarla. Además, tiene una función de alertas inteligentes para cuando la calidad del aire empeora a tu alrededor.



Figura 2.2: Opción de la aplicación móvil de AirVisual

Asimismo tiene una función de monitorización de la calidad del aire en interiores, que permite a los usuarios verla en interiores en tiempo real y recibir recomendaciones para mejorarla en espacios interiores. Para esta función se necesita un dispositivo de monitorización de la calidad del aire en interiores compatible con la aplicación. Estos se pueden comprar en la pagina web o aplicación de IQAir, y la adquisición de otros dispositivos. En la Figura 2.2. Opción de la aplicación móvil de AirVisual se observa la posibilidad de monitorización de la calidad del aire, para lo que se necesita añadir un dispositivo de monitorización.

La principal diferencia entre IQAir y el proyecto es que IQAir es una empresa que vende dispositivos de monitorización de la calidad del aire y el proyecto es una aplicación que utiliza dispositivos de monitorización de la calidad del aire ya existentes.

Una reseña muy frecuente por parte de los usuarios de la aplicación es que la aplicación es muy lenta, la información no es muy precisa y tiene un pequeño rango de cobertura.

2.1.2. Plume Labs

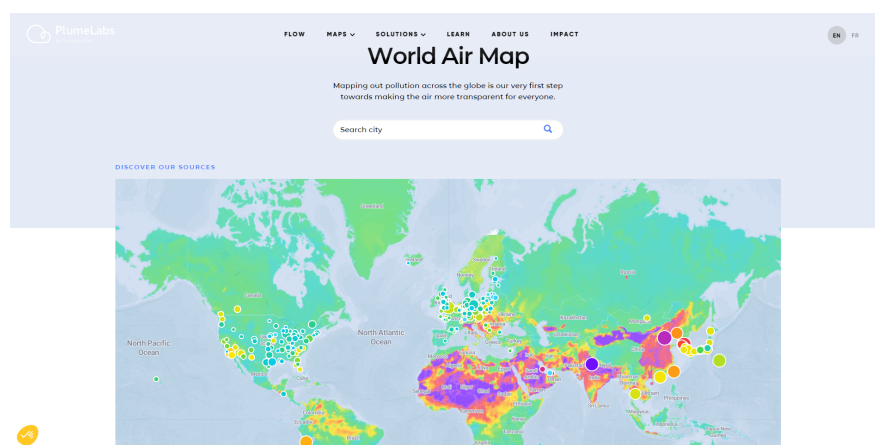


Figura 2.3: Página web de PlumeLabs

Plume Labs[8] es una empresa parisina que ha desarrollado una aplicación para conocer la contaminación del aire en el mundo. En el marco de la COP21 de París de 2015 presentaron su mapa de contaminación en tiempo real en la Figura 2.3. Página web de PlumeLabs se observa este mapa a través de la web. La metodología que han desarrollado combina inteligencia artificial y aprendizaje automático (machine learning), tecnologías en torno a las cuales han creado Plume Air Cloud, una plataforma de datos que recopila, agrega, normaliza y predice los niveles de contaminación atmosférica[9] .

Al igual que IQAir, Plume Labs es una multiplataforma con una web llamada plume.com y una aplicación llamada Plume Air Report, que tienen la misma funcionalidad y estética.

Como sucedía con la anterior alternativa, Plume Labs no tiene una gran cobertura fuera de zonas muy urbanas. También su cobertura se reduce a ciudades de Europa y Estados Unidos.

2.1.3. Accuweather

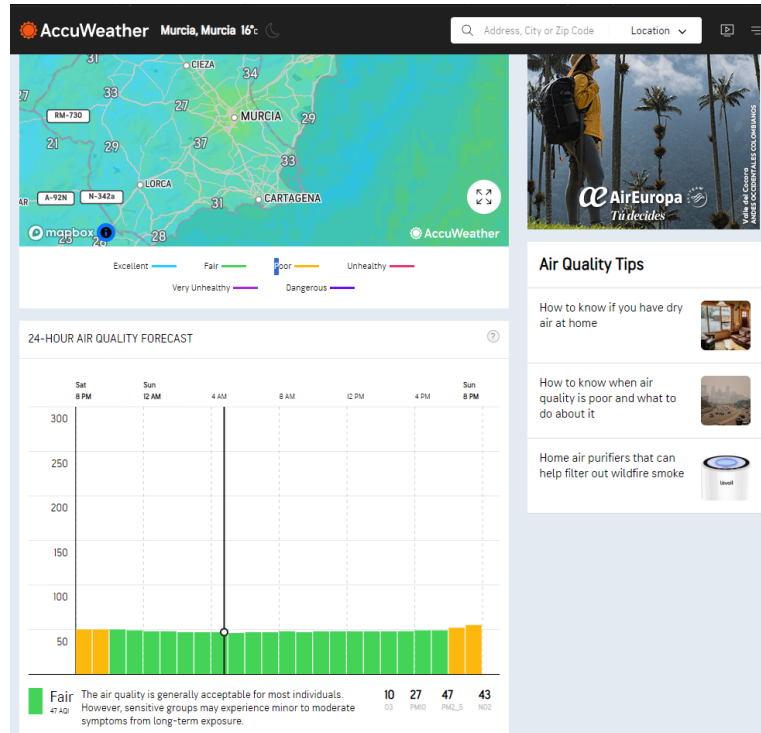


Figura 2.4: Página web de Accuweather

AccuWeather[10] es una empresa de pronóstico del tiempo con sede en State College, Pensilvania, que fue fundada en 1962. La empresa proporciona pronósticos de tiempo para más de 2.7 millones de ubicaciones en todo el mundo. En la Figura 2.4. Página web de Accuweather se observa la página principal de la web donde se aprecia un mapa con la calidad del aire actual en todo el mundo y una pequeña previsión de calidad del aire para mi localización actual.

Al igual que las otras dos alternativas, Accuweather es otra multiplataforma, llamándose su web [accuweather.com](https://www.accuweather.com) y dispone de una aplicación llamada Accuweather. Esta herramienta no se centra en exclusiva en la calidad del aire a diferencia de las otras dos alternativas. Aquí la calidad del aire es una opción más dentro de la aplicación y de la web.

En definitiva, Accuweather es una empresa de pronóstico general la cantidad y calidad de información proporciona es menor en cuanto a la calidad de aire se refiere. Este es otro posible enfoque para el proyecto.

2.1.4. Aemet

Aemet[11] es la Agencia Estatal de Meteorología de España. Es un organismo autónomo, adscrito al Ministerio para la Transición Ecológica y el Reto Demográfico, a través de la Secretaría de Estado de Medio Ambiente. Aemet es el principal proveedor de datos meteorológicos y climáticos en España. En la Figura 2.5. página web de Aemet se observa la página principal de la web donde se aprecia un mapa con la calidad del aire previsto para el día actual en España.

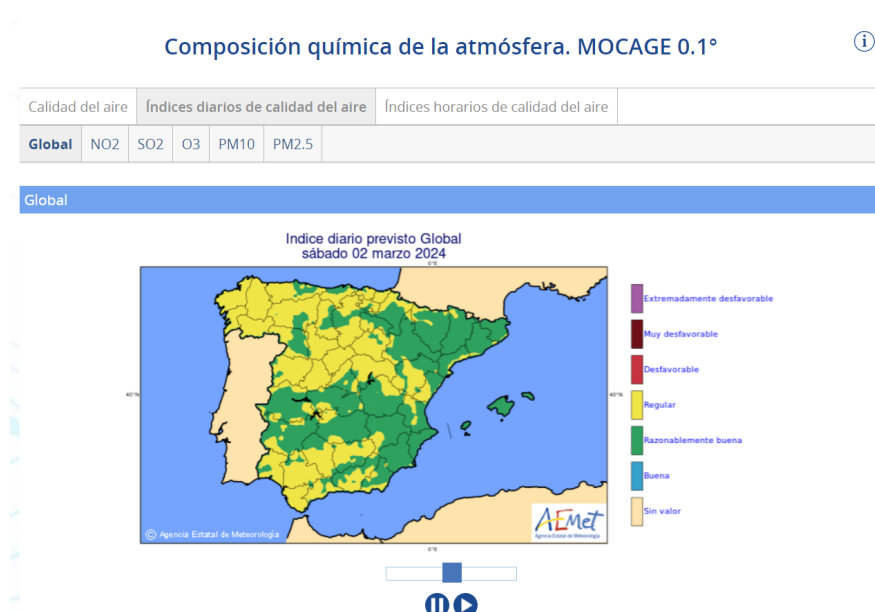


Figura 2.5: Página web de Aemet

Se repite el mismo patrón que los anteriores, Aemet es una multiplataforma, llamándose su web aemet.es y dispone de una aplicación llamada Aemet. Esta herramienta tiene una cobertura muy limitada, ya que se centra en exclusiva en la información meteorológica en España. Pero, por contra parte, la calidad de la información es la más alta de todas las alternativas. La mayoría de las aplicaciones de calidad del aire utilizan los datos de Aemet para la calidad del aire en España.

2.1.5. Comparación de aplicaciones similares

En la tabla 2.1 se muestra una comparación de las aplicaciones similares. Se observa que todas las aplicaciones tienen una funcionalidad muy similar, que es la de proporcionar información sobre la calidad del aire en tiempo real y pronósticos de calidad del aire para los próximos días. La principal diferencia entre las aplicaciones es la cobertura, la precisión de los datos y la estética de la aplicación.

Puntuación del 0 al 4						
	Cobertura a nivel nacional y mundial	Calidad software	Populari- dad	Calidad de la informa- ción	Opciones disponi- bles	Total
IQAir	3	3	3	3	3	15
PlumeLabs	3	4	2	3	3	15
Accuweather	4	3	4	3	4	18
Aemet	2	2	2	4	2	12

Tabla 2.1: Comparación de aplicaciones similares

La aplicación más valorada en la tabla 2.1 es Accuweather, esta va a ser una de las aplicaciones a seguir para el desarrollo del proyecto. La aplicación de Aemet es la menos valorada, pero es la que tiene la información más precisa. Mientras las dos aplicaciones IQAir y PlumeLabs son aplicaciones mas similares centradas totalmente en la calidad del aire. Esto provoca que sea menos populares y tengan una menor cobertura.

2.2. Tecnologías y herramientas

En esta sección se analizarán las diferentes tecnologías y herramientas que se pueden y se van a utilizar en el proyecto. Al ser una aplicación multiplataforma, se analizarán las tecnologías y herramientas para el desarrollo de aplicaciones web y móviles. Y por otro lado, se indicarán las tecnologías y herramientas para el despliegue y desarrollo del proyecto.

2.2.1. Lenguajes de programación

Los lenguajes de programación son muy importantes en el desarrollo de cualquier proyecto informático. Los lenguajes de programación más utilizados en el desarrollo de aplicaciones web y móviles son Node.js, React, Python, Java, y C#. A continuación, se analizarán estas tecnologías y se elegirá la más adecuada para el desarrollo de la aplicación web y móvil.

Node.js

Node.js [12] es un lenguaje que sirve para crear sitios web dinámicos muy eficientes, escritos con el lenguaje de programación JavaScript. Normalmente, los desarrolladores se decantan por este entorno de ejecución cuando buscan que los procesos se ejecuten de forma ágil y sin ningún tipo de bloqueo cuando las conexiones se multiplican.

Este entorno fue creado por los desarrolladores de JavaScript con el objetivo de ir un paso más allá con este lenguaje de programación. Node.js utiliza un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, Node Package Manager (NPM), es el ecosistema de bibliotecas de código abierto más grande del mundo. Node.js es una tecnología muy popular y es utilizada por empresas como Netflix, Uber, LinkedIn, PayPal, eBay, Walmart y Medium, entre otras. Esta tecnología se enfoca en la parte del servidor, por lo que es ideal para el desarrollo de la parte del servidor de la aplicación web.

Python

Python[13] es un lenguaje de programación ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML). Los desarrolladores utilizan Python porque es eficiente y fácil de aprender, además de que se puede ejecutar en muchas plataformas diferentes. El software Python se puede descargar gratis, se integra bien a todos los tipos de sistemas y aumenta la velocidad del desarrollo.



Figura 2.6: Logo de Python

Este lenguaje se puede usar para diferentes tareas dentro del proyecto, como el desarrollo web del lado del servidor, realizar tareas de ciencia de datos y machine learning, automatización con scripts de Python y la automatización de tareas. Python es un lenguaje interpretado, lo que significa que el código se ejecuta línea por línea, también es un lenguaje de alto nivel, lo que significa que es fácil de leer y escribir y es fácil de aprender. Python es utilizado por empresas como Google, Facebook, Instagram, Spotify, Netflix, Dropbox, Reddit, Quora, y muchas otras.

Java

Java [14] es una plataforma informática de lenguaje de programación creada por Sun Microsystems en 1995. Ha evolucionado desde sus humildes comienzos hasta impulsar una gran parte del mundo digital actual, ya que es una plataforma fiable en la que se crean muchos servicios y aplicaciones.



Figura 2.7: Logo de Java

Para este lenguaje se puede utilizar el framework Spring, que es un framework de código abierto para el desarrollo de aplicaciones en Java. Este framework se ha convertido en el más popular para el desarrollo de aplicaciones web en Java.

Spring[15] es un framework muy popular y es utilizado por empresas como Netflix, Amazon, LinkedIn, y muchas otras. Spring es un framework muy completo y es ideal para el desarrollo de la parte del servidor de la aplicación web.

C#

C# [16] es un lenguaje de programación moderno, basado en objetos y con seguridad de tipos. C# permite a los desarrolladores crear muchos tipos de aplicaciones seguras y sólidas que se ejecutan en .NET. Proporciona construcciones de lenguaje para el desarrollo de software muy popular y es utilizado por empresas como Microsoft, Stack Overflow, Dell, y muchas otras. C# es un lenguaje ampliamente utilizado e ideal para el desarrollo de la parte del servidor de aplicaciones web.

React

ReactJS [17] es una de las librerías más populares de JavaScript para el desarrollo de aplicaciones móviles y web. Creada por Facebook, React contiene una colección de fragmentos de código JavaScript reutilizables utilizados para crear interfaces de usuario (UI) llamadas componentes.

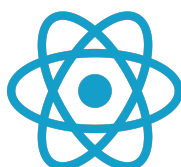


Figura 2.8: Logo de React

React es una librería de código abierto que se utiliza para crear UI en aplicaciones web. Es utilizada por empresas como Instagram, WhatsApp, Airbnb, y muchas otras. React es válida para el desarrollo de la parte del cliente de la aplicación web.

Comparación y elección de lenguajes de programación

Puntuación del 0 al 4						
	Código abierto	Popularidad	Documentación y curva de aprendizaje	Escalable y rendimiento	Experiencia	Total
Node.js	4	3	2	3	0	12
Python	4	4	4	3	2	17
React	4	4	3	3	2	16
Spring Java	4	3	4	4	4	19
C#	2	3	3	3	1	12

Tabla 2.2: Comparación de lenguajes

En la Tabla 2.1. Comparación de lenguajes se observa que el lenguaje de programación con mayor puntuación es Java, seguido de Python y React. Por lo tanto, se va a utilizar Java para el desarrollo de la parte del servidor de la aplicación web. React para el desarrollo de la parte del cliente de la aplicación web y Python para el desarrollo de un machine learning model para la predicción de calidad del aire.

2.2.2. Bases de datos

Las bases de datos son el elemento principal para la persistencia en el desarrollo de aplicaciones web y móviles. Las bases de datos más utilizadas en el desarrollo de aplicaciones web y móviles son MySQL, MongoDB y PostgreSQL. A continuación, se analizarán estas tecnologías y se elegirá la más adecuada para el desarrollo de la aplicación web y móvil.

MySQL

MySQL [18] es un sistema de código abierto de administración de bases de datos que es desarrollado y soportado por Oracle. MySQL es un sistema de administración de bases de datos relacional, lo que significa que almacena datos en tablas y utiliza claves primarias para identificar de forma única cada fila dentro de una tabla. MySQL esta muy extendida. Se utiliza por empresas como Google, Facebook, Twitter, LinkedIn, y entre otras.

MongoDB

MongoDB [19] es un sistema de gestión de bases de datos no relacionales y de código abierto, que utiliza documentos flexibles en lugar de tablas y filas para procesar y almacenar varias formas de datos.



Figura 2.9: Logo de MongoDB

Al ser una solución de base de datos NoSQL, MongoDB no requiere un sistema de gestión de bases de datos relacionales, por lo que proporciona un modelo de almacenamiento de datos elástico que permite a los usuarios almacenar y consultar fácilmente tipos de datos multivariados. Esto no solo simplifica la gestión de la base de datos para los desarrolladores, sino que también crea un entorno altamente escalable para aplicaciones y servicios multiplataforma. Un aspecto a tener en cuenta es que MongoDB es necesaria, ya que se va a utilizar para el almacenamiento de datos en la nube de FIWARE.

PostgreSQL

PostgreSQL [20], comúnmente pronunciado "Post-GRES", es una base de datos de código abierto que goza de una sólida reputación por su fiabilidad, flexibilidad y compatibilidad con estándares técnicos abiertos. A diferencia de otros sistemas de gestión de bases de datos relacionales, PostgreSQL admite tanto tipos de datos no relacionales como relacionales. Esto la convierte en una de las bases de datos relacionales más compatibles, estables y maduras disponibles en la actualidad. PostgreSQL es una base de datos ampliamente aceptada para el desarrollo de la aplicación web y móvil.

Comparación y elección de bases de datos

Puntuación del 0 al 4						
	Código abierto	Popularidad	Equilibrio de carga	Escalable y rendimiento	Experiencia	Total
MySQL	4	3	3	3	4	17
MongoDB	4	4	3	4	3	18
PostgreSQL	4	4	4	3	1	16

Tabla 2.3: Comparación bases de datos

En la Tabla 2.3 se observa que no hay una base de datos que destaque sobre las demás. Las tres opciones son muy óptimas y se pueden utilizar para el desarrollo de la aplicación. En este caso, voy a elegir MySQL, porque es la base de datos que más he utilizado a lo largo de la carrera y con la que más familiarizado estoy.

2.2.3. Entornos de despliegue

Existen dos opciones muy populares para el despliegue de aplicaciones web y móviles, que son Docker y servicios en la nube como AWS, Google Cloud, y Azure.



Figura 2.10: Logo de Docker

Docker [21] es un sistema operativo para contenedores. De manera similar a cómo una máquina virtual virtualiza el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos sencillos que puede utilizar para crear, iniciar o detener contenedores. Los contenedores son ligeros y rápidos, por lo que son ideales para el despliegue de aplicaciones web y móviles.

Los servicios en la nube son plataformas que ofrecen servicios de computación, almacenamiento y bases de datos a través de Internet. Los servicios en la nube son ideales para el despliegue de aplicaciones web y móviles, ya que ofrecen una gran cantidad de recursos y son muy fáciles de usar.

La opción elegida es Docker. Es una tecnología muy interesante para el despliegue de aplicaciones web y móviles. El próximo paso del proyecto en el futuro es el despliegue de la aplicación en la nube de FIWARE. Es una gran opción o cualquiera de los servicios de nube ya mencionados.

2.2.4. Entornos de desarrollo

Los entornos de desarrollo son muy importantes en el desarrollo de aplicaciones. Los entornos de desarrollo más utilizados son Visual Studio Code, PyCharm y Eclipse.



Figura 2.11: Logo de Visual Studio Code

Visual Studio Code [22] se caracteriza por ser un editor de código fuente independiente que se ejecuta en Windows, macOS y Linux. Representa la elección principal para desarrolladores web y JavaScript, con extensiones para admitir casi cualquier lenguaje de programación. Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Visual Studio Code es un editor de código fuente muy conocido y es ideal para el desarrollo de la aplicación web y móvil.

Mientras Visual Studio Code es más genérico, PyCharm es un entorno de desarrollo integrado utilizado en programación informática, específicamente para el lenguaje de programación Python. También Eclipse es otro entorno de desarrollo integrado, muy utilizado en el desarrollo de aplicaciones de software de Java.

Mi elección favorita es Visual Studio Code para el desarrollo del proyecto. Primero,

es un editor de código fuente muy popular. Personalmente lo he utilizado mucho a lo largo de la carrera. Segundo, se va a utilizar diferentes lenguajes de programación en el proyecto y Visual Studio Code es compatible con todos ellos. Y por último, este editor es aceptado por una gran comunidad debido a la enorme posibilidades de extensiones que se pueden utilizar para el desarrollo del proyecto.

2.2.5. Api obtención de datos de calidad del aire

Para esta tecnología se han valorado dos opciones principales Meteosource [23] y API Air Quality de Google [24].

Meteosource



Figura 2.12: Logo de Meteosource

La API de calidad del aire [23] ofrece datos actuales y previsiones de la contaminación y el índice de calidad del aire para cualquier ubicación del mundo. La calidad se ve afectada por los contaminantes, que se emiten de manera directa a la atmósfera (contaminantes del aire principales) como materiales de partículas (PM) y óxidos de sulfuros, o se forman en la atmósfera a través de reacciones químicas (contaminantes secundarios) como el ozono o dióxido de nitrógeno.

API Air Quality de Google

La API Air Quality[25] proporciona datos sobre la calidad del aire para 100 países, incluidos mapas de calor de contaminación y detalles sobre tipos específicos de contaminantes presentes. Para mantenerse actualizado, la API valida y organiza terabytes de datos por hora de fuentes como estados de monitoreo gubernamental, satélites, datos meteorológicos y sensores.

Comparación y elección de API de calidad del aire

La diferencia más importante entre las dos opciones consiste en que la API de Google es gratuita y la de Meteosource, de pago. Por lo tanto, se va a utilizar la API de Google para la obtención de datos de calidad del aire. En cuanto a la calidad de los datos, ambas opciones son muy buenas y se pueden utilizar para el desarrollo de este proyecto.

3. Análisis de objetivos y plan de trabajo

3.1. Análisis de objetivos

El objetivo general del presente trabajo es el desarrollo de una aplicación web y móvil para la monitorización, notificación, visualización y predicción de la calidad del aire. Podemos dividir el objetivo en los siguientes objetivos específicos:

OB1 Investigar y estudiar la tecnología FIWARE.

OB2 Obtener y estudiar API para la calidad del aire.

OB3 Estudiar tecnologías y herramientas para el desarrollo de aplicaciones multiplataforma.

OB4 Analizar el sistema y plantear requisitos.

OB5 Diseñar la arquitectura de la aplicación.

OB6 Diseñar el modelo de datos.

OB7 Desarrollar el back-end.

OB8 Desarrollar el front-end.

OB9 Estudio de modelos de machine learning para la predicción.

OB10 Integrar la aplicación.

OB11 Validar la aplicación en un entorno de prueba.

OB12 Documentar la aplicación multiplataforma.

Indicar que la versión esperada de la aplicación no debe ser considerada como un producto final, sino como un prototipo funcional que puede ser mejorado y ampliado en el futuro. Por otro lado, también se establece como objetivo de este Trabajo Final de Grado (TFG) la realización de una memoria en que se describan las principales características del trabajo realizado.

A continuación, se detallan las fases del proyecto y los objetivos asignados a cada una de ellas.

Fase 1 : Objetivo(s): OB1, OB2, OB3

1. Consulta y estudio de la información sobre el origen, propósito y arquitectura de FIWARE.
2. Estudio sobre los componentes y estándares de FIWARE.
3. Análisis de requisitos sobre las API para la calidad del aire.
4. Obtención y estudio sobre API para la calidad del aire.
5. Estudio del estado del arte sobre herramientas para la monitorización, notificación, visualización y predicción de la calidad del aire.
6. Estudio de las tecnologías para el desarrollo del back-end.
7. Estudio de las tecnologías para el desarrollo del front-end.
8. Estudio de las tecnologías para el desarrollo de machine learning.

Fase 2 : Objetivo(s): OB4

1. Definición del alcance del sistema.
2. Desglose de objetivos y tareas.
3. Planificación para el desarrollo del proyecto.

Fase 3 : Objetivo(s): OB5, OB6

1. Diseño del modelo de dominio del proyecto.
2. Diseño del componente back-end del sistema a partir de servicios REST.
3. Diseño del componente front-end del sistema siguiendo un modelo responsivo.

Fase 4 : Objetivo(s): OB7, OB8, OB9

1. Desarrollar los servicios que constituyen la lógica de negocio (back-end).
2. Desarrollar los componentes que constituyen la interfaz de usuario adaptable (front-end).
3. Desarrollo de machine learning para la predicción.

Fase 5 : Objetivo(s): OB10

1. Integración de los componentes del sistema front y back.
2. Integración de los componentes de machine learning.
3. Integración de los componentes de FIWARE.
4. Integración de los componentes de API para la calidad del aire.

Fase 6 : Objetivo(s): OB11

1. Realización de las pruebas de integración.
 2. Realización de las pruebas de usabilidad.
-

- Fase 7 : Objetivo(s): OB12
- 1. Documentación del proyecto.
 - 2. Realización de la memoria del proyecto.
 - 3. Corrección de la memoria del proyecto con las tutoras.
 - 4. Realización de la presentación del proyecto.

3.2. Planificación temporal

Para la planificación he utilizado la herramienta clickup [26], que me ha permitido organizar las tareas y sub-tareas de forma sencilla y visual. Con esta herramienta he podido establecer las fechas de inicio y fin de cada tarea, así como el estado de cada tarea. He podido trabajar con el tablero de tareas de kanban y el diagrama de Gantt.

A continuación, en la Figura 3.1 se muestra la planificación temporal del proyecto en formato lista.

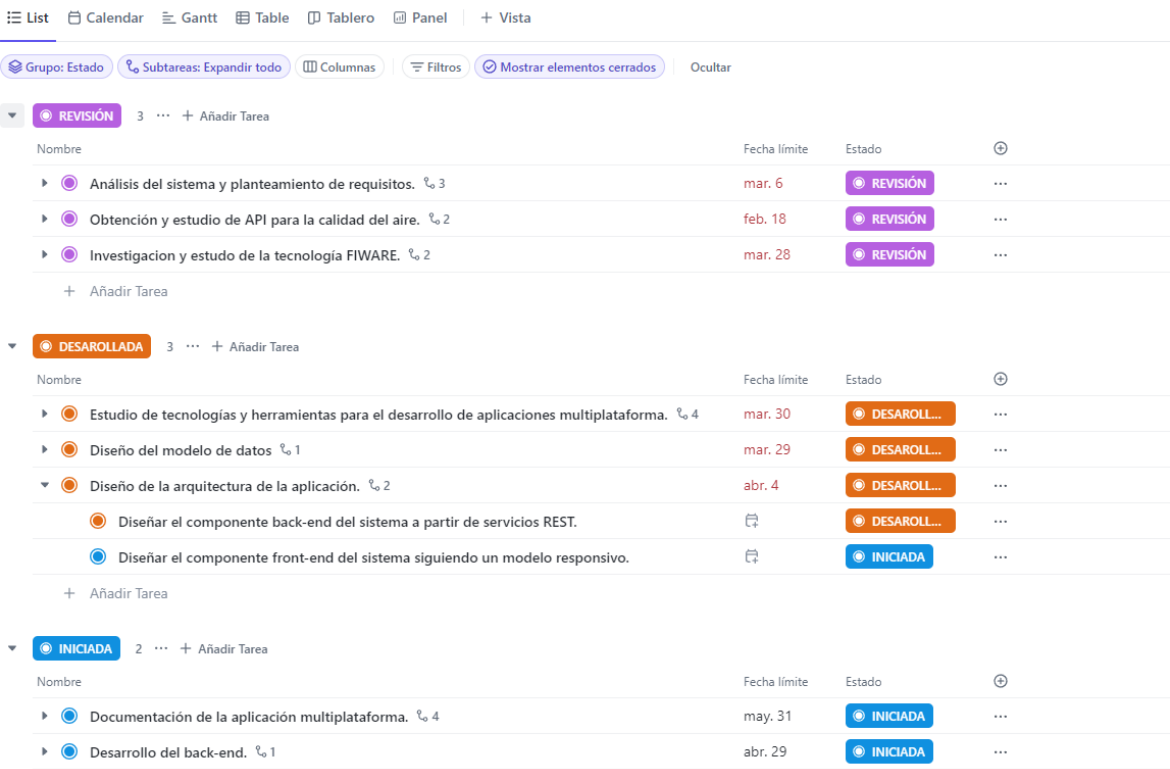


Figura 3.1: Lista de objetivos

En esta lista se observan todas las tareas con sus correspondientes subtareas. Cada tarea tiene una fecha límite. Además, se puede observar el estado de cada tarea y subtarea.

En la Figura 3.2 se muestra la planificación temporal del proyecto en formato diagrama de Gantt realizado en la herramienta clickup. Desde este diagrama he podido llevar una organización precisa del proyecto.

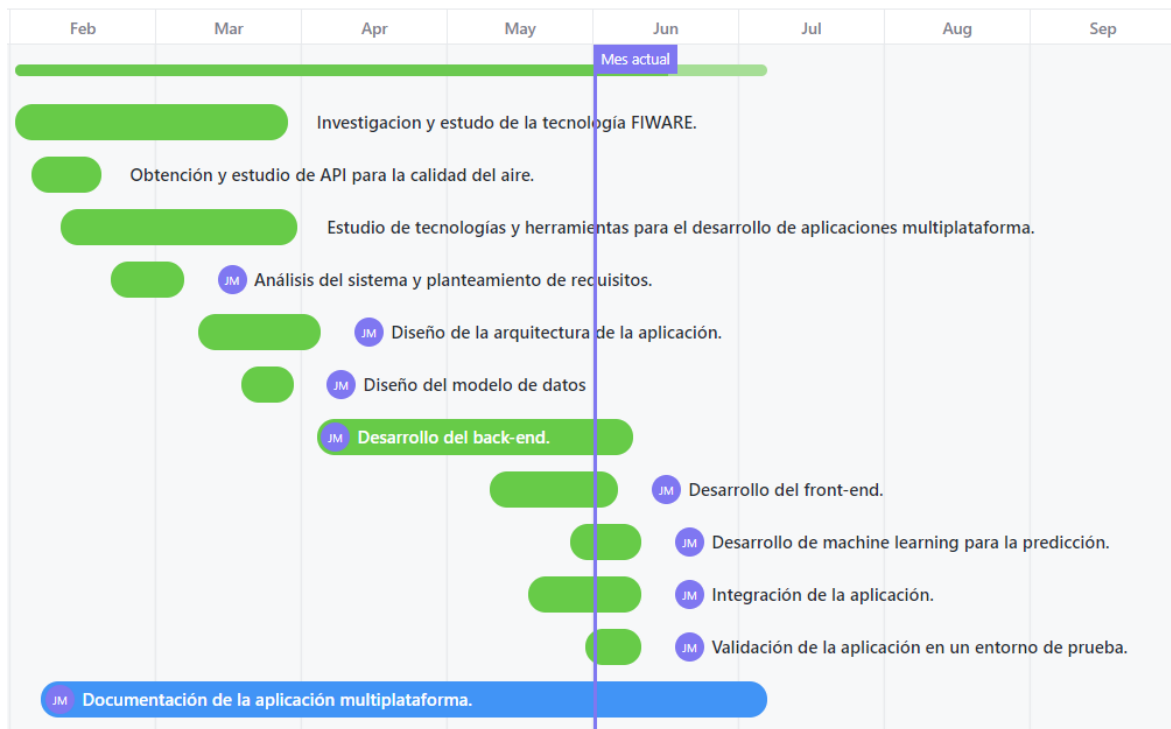


Figura 3.2: Diagrama de Gantt

4. Diseño del trabajo realizado

En este capítulo se detallan las decisiones tomadas en el diseño de la aplicación. Se recopilan los requisitos del sistema en forma de diagramas de casos de uso, se describen las tecnologías y herramientas utilizadas dentro del marco de trabajo de FIWARE y por último, se muestra la arquitectura completa de la aplicación mostrando las relaciones entre los distintos componentes de la misma.

4.1. Requisitos de la aplicación

Se ha elegido ilustrar los requisitos de la herramienta creada por medio de casos de uso en lenguaje natural para facilitar la comprensión y comunicación entre todas las partes interesadas [27]. En primer lugar, cabría señalar que hablaremos ahora de tres tipos de actores:

- **Usuario observador:** Este usuario representa a cualquier persona que quiera visitar la página web sin tener que registrarse. Podrá visualizar un resumen de la calidad del aire en tiempo real para su ubicación actual. Además, puede registrarse o iniciar sesión en la aplicación para obtener información más detallada convirtiéndose en un usuario participativo.
- **Usuario participativo:** Corresponde al usuario principal de la aplicación. Para poder utilizar la aplicación, este usuario debe de iniciar sesión o registrarse. Este puede visualizar la calidad del aire en tiempo real, recibir alertas de calidad del aire, visualizar noticias de su país, consultar la predicción de la calidad del aire para los próximos 7 días, cerrar la sesión y actualizar sus datos personales. Entre estos datos personales se encuentran la ubicación.
- **Gestor:** Estos usuarios también tienen control de sesión como los usuarios participativos a diferencia de que para crear un nuevo gestor debe de ser creado por un gestor. Además, de crear gestores pueden actualizar su información personal, consultar la información de los usuarios participativos y eliminar de la aplicación a los usuarios participativos.

4.1.1. Casos de Uso: Usuario Observador

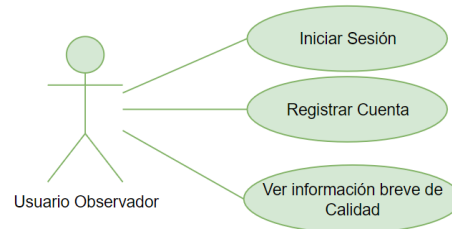


Figura 4.1: Casos de Uso del usuario observador

Las tarjetas de requisitos de los anteriores casos de uso son las siguientes:



Figura 4.2: Tarjetas de los casos de uso del usuario observador

4.1.2. Casos de Uso: Usuario Participativo

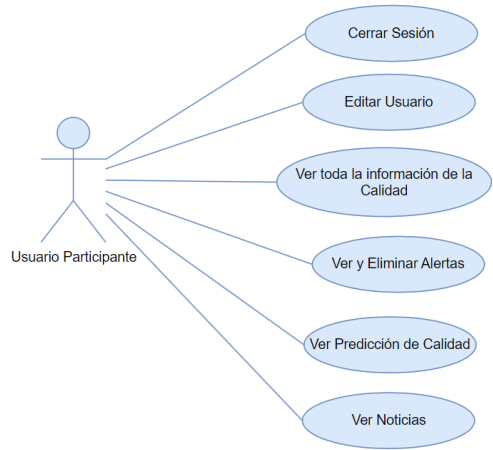


Figura 4.3: Casos de Uso del usuario participativo

Las tarjetas de requisitos de los anteriores casos de uso son las siguientes:

CDU: 4 Prioridad: 3 Complejidad: 1	Cerrar Sesión El usuario participativo puede cerrar sesión en la aplicación.
CDU: 5 Prioridad: 2 Complejidad: 2	Editar Usuario El usuario participativo puede editar su perfil modificando los campos disponibles.
CDU: 6 Prioridad: 5 Complejidad: 3	Ver toda la Información de la Calidad El usuario participativo puede visualizar la calidad del aire en formato tabla con todos los datos disponibles.
CDU: 7 Prioridad: 5 Complejidad: 3	Ver registro de Alertas El usuario participativo puede ver en formato lista un registro de las alertas. Este usuario puede eliminar cada alerta de manera individual o eliminarlas todas.
CDU: 8 Prioridad: 1 Complejidad: 3	Ver Noticias El usuario participativo puede visualizar las noticias sobre la calidad del aire.
CDU: 9 Prioridad: 3 Complejidad: 5	Ver Predicciones El usuario participativo puede visualizar la predicción de la calidad del aire en formato tabla de información.

Figura 4.4: Tarjetas de los casos de uso del usuario participativo

4.1.3. Casos de Uso Gestor

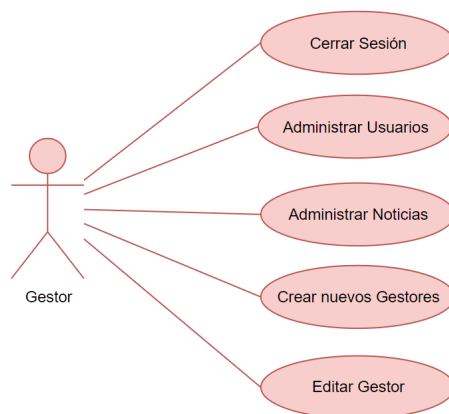


Figura 4.5: Casos de Uso del gestor

Las tarjetas de requisitos de los anteriores casos de uso son las siguientes:

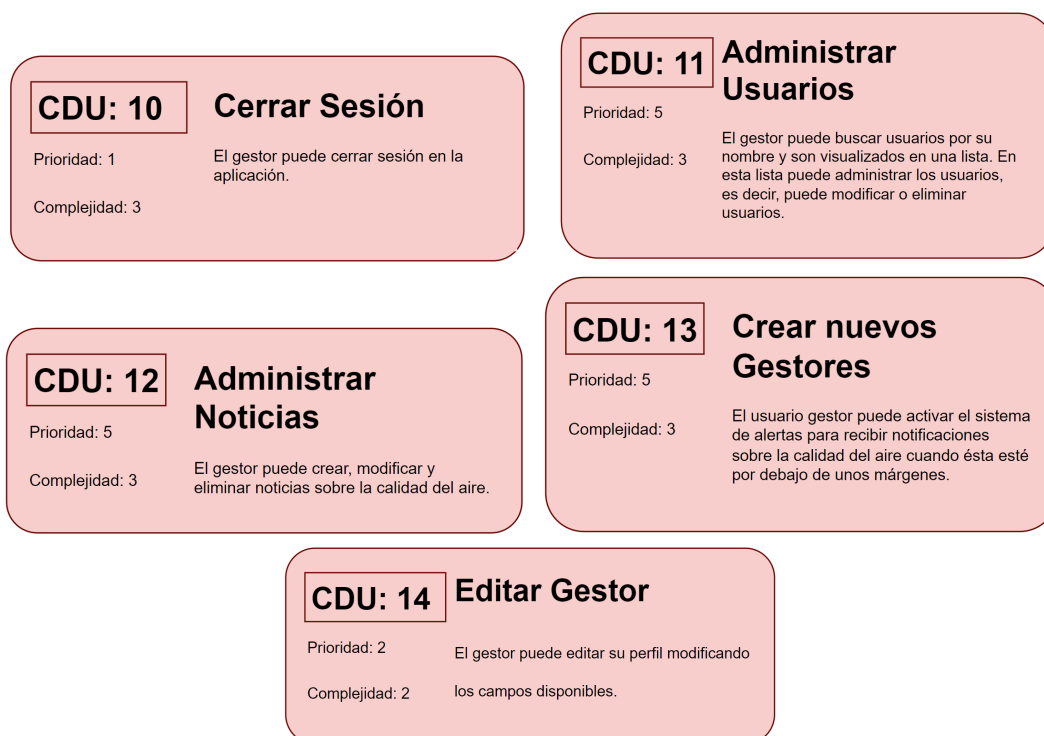


Figura 4.6: Tarjetas del los casos de uso del gestor

4.2. Fiware

4.2.1. ¿Qué es Fiware?

FIWARE es un ecosistema de innovación y emprendimiento abierto basado en las tecnologías del internet del futuro. Este ecosistema se está convirtiendo rápidamente en una plataforma base para el desarrollo de servicios inteligentes a nivel internacional. La plataforma FIWARE fue desarrollada por el programa europeo para el desarrollo de Internet del Futuro (FI-PPP), quien se planteó el objetivo de construir un ecosistema sostenible alrededor de estándares abiertos donde las oportunidades pueden ser capturadas por proveedores de aplicaciones y proveedores de plataformas. Las soluciones inteligentes tienen como principal característica la recopilación de datos del entorno en el que se desenvuelve (son conscientes de su contexto), desde diferentes fuentes como usuarios finales, redes de sensores, aplicaciones móviles y todo tipo de sistemas de información como los sistemas Customer Relationship Management (CRM) y las redes sociales [28].

Para recopilar y administrar información de contexto, FIWARE cuenta con un componente central llamado Context Broker, a través del cual se interactúa con otras plataformas o aplicaciones utilizando la API restful FIWARE Interfaces de Servicio de Nueva Generación (NGSI). Esta API es abierta y estándar, por lo que ofrece a los desarrolladores, la capacidad de portar sus aplicaciones a través de diferentes plataformas desarrolladas también con FIWARE y un marco estable para futuros desarrollos.

La plataforma FIWARE se compone de diferentes componentes, cada uno de ellos con una funcionalidad específica. Algunos de los componentes más importantes son: IoT Agent (para la gestión de dispositivos IoT), Keyrock (para la gestión de usuarios y roles), Cygnus (para la persistencia de datos), Wilma (para la gestión de notificaciones) y Wirecloud (para la creación de dashboards).

4.2.2. Orion Context Broker

El principal y único componente obligatorio de cualquier plataforma o solución desarrollada con FIWARE es el Orion Context Broker (OCB), el cual aporta una función fundamental en cualquier solución inteligente. Orion es una implementación en C++ del enlace API REST NGSIv2 desarrollado como parte de la plataforma FIWARE.

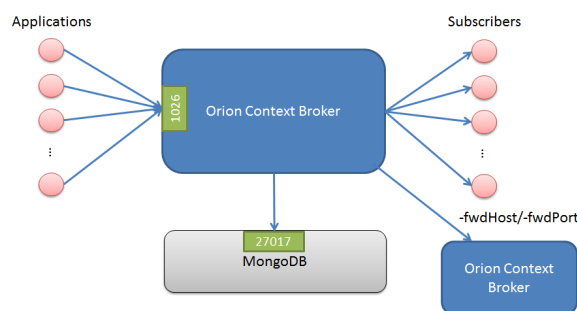


Figura 4.7: Arquitectura de Orion Context Broker

Orion Context Broker le permite gestionar todo el ciclo de vida de la información de contexto, incluidas actualizaciones, consultas, registros y suscripciones. Es una implementación de servidor NGSIv2 para gestionar la información de contexto y su disponibilidad. OCB, puede crear elementos de contexto y administrarlos mediante actualizaciones y consultas. Además, puede suscribirse a información de contexto para que cuando ocurra alguna condición reciba una notificación [29].

4.2.3. NGSIv2

NGSI (Next Generation Service Interface) es una API que permite la gestión de información de contexto. NGSIv2 es la versión más reciente de esta API, que se ha convertido en un estándar de facto para la gestión de información de contexto. NGSIv2 define una serie de operaciones que permiten a las aplicaciones crear, leer, actualizar y eliminar información de contexto, así como suscribirse a notificaciones de cambios en la información de contexto. NGSIv2 se basa en HTTP y utiliza JSON como formato de intercambio de datos.

NGSIv2 tiene dos tipos de operaciones: operaciones de entidad y operaciones de suscripción [1].

Las operaciones de entidad permiten a las aplicaciones crear, leer, actualizar y eliminar entidades y atributos de entidades como se aprecia en la figura 4.8.

Entities		
GET	/v2/entities	List Entities
POST	/v2/entities	Create Entity
GET	/v2/entities/{entityId}	Retrieve Entity
DELETE	/v2/entities/{entityId}	Remove Entity
GET	/v2/entities/{entityId}/attrs	Retrieve Entity Attributes

Figura 4.8: Especificación en OpenAPI de las operaciones sobre entidades en NGSI [1]

Las operaciones de suscripción permiten a las aplicaciones suscribirse a notificaciones de cambios en la información de contexto como se aprecia en la figura 4.9.

Subscriptions		
GET	/v2/subscriptions	List Subscriptions
POST	/v2/subscriptions	Create Subscription
GET	/v2/subscriptions/{subscriptionId}	Retrieve Subscription
DELETE	/v2/subscriptions/{subscriptionId}	Delete subscription

Figura 4.9: Especificación en OpenAPI de las operaciones sobre suscripciones en NGSI

4.2.4. Modelo NGSI

Toda la comunicación entre los distintos componentes de la arquitectura de alto nivel del OCB se realiza a través de la API RESTful NGSI v2, la cual es inspirada y basada en la especificación Open Mobile Alliance (OMA) NGSI. Por lo tanto, la información de contexto en FIWARE está representada a través de estructuras de datos genéricos referidos como elementos de contexto. Un elemento de contexto se refiere a la información que es producida, recopilada u observada y que puede ser relevante para su procesamiento, análisis y extracción de nuevo conocimiento[30].

FIWARE soporta un conjunto de tipos de datos básicos, así como la posibilidad de definir tipos de datos estructurados; vectores y mapas clave (cuyos elementos pueden ser otros vectores, mapas clave o tipos de datos simples). Un elemento de contexto debe de contener al menos un atributo que actúe como identificador único, el cual es conocido como el Entity ID y el tipo de entidad al que pertenece llamado Entity Type.

El modelo de datos NGSI se compone de tres elementos principales:

- Un EntityId y un EntityType único que identifica la entidad a la cual los datos de contexto hacen referencia.
- Una secuencia de uno o más atributos de elementos de datos.
- Metadatos opcionales vinculados a atributos.

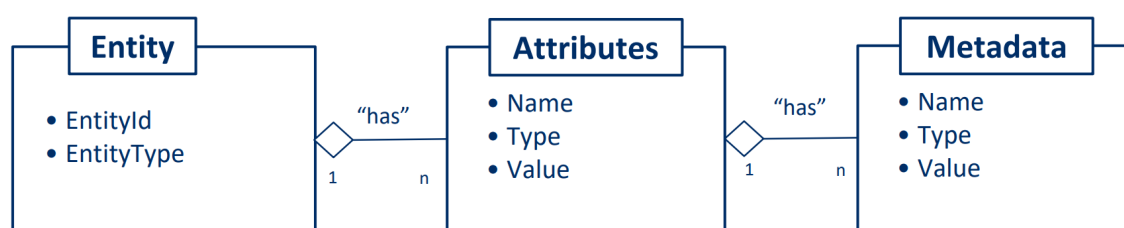


Figura 4.10: Elementos principales del modelo de datos

En la estructura de FIWARE existen diferentes tipos de datos como Text (cadenas de texto), Integer (números enteros), Float (números decimales), Boolean, DateTime (fechas y horas), GeoPoint (puntos geográficos) y GeoJSON (datos geográficos). Pero también existen entidades estandarizadas para la representación de datos de contexto.

Estas son algunas de las categorías disponibles en el modelo de datos NGSI:

- **Smart City**: Entidades relacionadas con la ciudad inteligente.
- **Agrifood**: Entidades relacionadas con la agricultura y la alimentación.
- **Water**: Entidades relacionadas con el agua.
- **Energy**: Entidades relacionadas con la energía.
- **Environment**: Entidades relacionadas con el medio ambiente.
- **Aeronautics**: Entidades relacionadas con la aeronáutica.
- **Health**: Entidades relacionadas con la salud.
- **Logistics**: Entidades relacionadas con la logística.

Para este proyecto se ha utilizado la entidad estandarizada AirQualityObserved que se encuentra en la categoría Environment. Esta entidad se utiliza para representar la calidad del aire observada en un lugar y momento determinados. Mas adelante se explicara con más detalle esta entidad.

Suscripciones

Las suscripciones ya mencionadas permiten a las aplicaciones recibir notificaciones de cambios en la información de contexto. Una aplicación puede suscribirse a un conjunto de entidades y especificar un conjunto de condiciones que deben cumplirse para que se envíen notificaciones.

La estructura de una suscripción en NGSIv2 en formato JavaScript Object Notation (JSON) se puede apreciar en la tabla 4.1

Nombre	Representación	Descripción	Obligatorio/Opcional
id	String	Identificador único de la suscripción	Creación automática
description	String	Descripción de la suscripción	Opcional
subject	Atributo compuesto	Especifica el conjunto de entidades a las que se refiere la suscripción	Obligatorio
notification	Atributo compuesto	Especifica la información necesaria para enviar una notificación	Obligatorio
expires	DateTime	Fecha y hora de expiración de la suscripción	Opcional
status	String	Estado de la suscripción	Solo lectura
throttling	Integer	Número mínimo de segundos entre notificaciones	Opcional

Tabla 4.1: Descripción de los campos una suscripción en NGSIv2

El atributo `subject` es un atributo compuesto que contiene los campos `entities` y `condición`. El campo `entities` es un array de entidades a las que se refiere la suscripción y es obligatorio. Este campo se compone del `id` de la entidad y el tipo de entidad.

El campo `condición` es un atributo compuesto que especifica las condiciones que deben cumplirse para que se envíen notificaciones. Esta compuesto de los campos de atributo, operador y valor.

El atributo `notification` es un atributo compuesto, que contiene los campos de atributos que se enviarán en la notificación, el formato de los atributos en la notificación, la cabecera Hypertext Transfer Protocol (HTTP) y otros campos que no se van a utilizar

en este proyecto.

Los atributos de la notificación se pueden definir de dos formas, con la cabecera `attrs` que especifican los atributos a transmitir o la cabecera `execpAtts` que especifica los atributos que no se van a retransmitir.

Para el atributo HTTP también existen dos tipos, `http` y `httpCustom`. La diferencia es que en el primero solo se especifica la url de la notificación y en el segundo se especifica la url, el token de autenticación y más opciones disponibles.

Las subscripciones van a ser utilizadas en este proyecto para la notificación de cambios en la calidad del aire observada, cuando ésta esté por debajo de un umbral establecido. De esta forma se podrá notificar a los usuarios participativos de la aplicación.

AirQualityObserved

Como se ha mencionado anteriormente, dentro de la categoría Environment de FIWARE se encuentra **AirQualityObserved**. Esta es una entidad estandarizada en el modelo de datos NGSI. Esta es una observación de las condiciones de calidad del aire en un lugar y momento determinados. Este modelo de datos se ha desarrollado en cooperación con operadores móviles y la Global System for Mobile Communications Association (GSMA). Este modelo de datos esta representado por los siguientes campos:

Nombre	Representación	Descripción	O/Op
id	String	Identificador único de la observación	O
type	String	Tipo de entidad debe de ser "AirQualityObserved"	O
dateProvider	URL	Especifica la URL a información sobre el proveedor de esta información.	Op
dateModified	DateTime	Fecha y hora de la última modificación de la observación	R
dateCreated	DateTime	Fecha y hora de creación de la observación	R
location	GeoPoint	Coordenadas geográficas de la observación	O*

address	Atributo compuesto	Dirección cívica de observación de la calidad del aire. En ocasiones corresponde a la dirección de la estación de calidad del aire.	O*
dateObserved	DateTime	Fecha y hora de la observación en formato UTC ISO860	O
areaServed	String	Área geográfica a la que se aplica la observación. Se puede utilizar para agrupar mediciones de calidad del aire por distrito, barrio, etc.	Op
source	String/Url	Fuente de la observación	Op
airQualityLevel	String	Nivel de calidad del aire observado. Los valores posibles son good, moderate, unhealthyForSensitiveGroups, unhealthy, veryUnhealthy y hazardous. Son obligatorios los metadatos que especifican la referencia de los valores	Op
airQualityIndex	Integer	Índice de calidad del aire observado. Metadatos opcionales que especifican la referencia de los valores	Op
reliability	Float[0,1]	Fiabilidad de la observación (porcentaje, expresada en partes por uno).	Op
refDevice	Device	Identificador único de la entidad que representa el dispositivo que ha realizado la observación.	Op
refPointOfInterest	PointOfInterest	Identificador único de la entidad que representa el punto de interés al que se refiere la observación.	Op

Tabla 4.2: Descripción de los campos de la entidad AirQualityObserved. La columna "O/Op" indica si el campo es obligatorio (O) u opcional (Op). "R" significa que el campo no puede ser modificado una vez creado. "O*" significa que el campo es obligatorio si no se ha proporcionado el campo address/location respectivamente.

Además de los campos mencionados, la entidad `AirQualityObserved` contiene una representación de los contaminantes medidos. Estos contaminantes tienen la siguiente estructura:

- Primero son identificados con un nombre. Este nombre tiene que ser exactamente igual a la fórmula química que representa al contaminante. Por ejemplo, el dióxido de nitrógeno se representa como `NO2`.
- Tienen el tipo de atributo que corresponde a `number`. Y el valor que corresponde a la cantidad de contaminante medida.
- Por último, tienen tres tipos de metadatos:
 - **timestamp**: Fecha y hora de la medición en formato ISO8601. Es opcional ya que puede estar disponible en el atributo `dateObserved`.
 - **unitCode**: Unidades de medida del contaminante representadas con un máximo de tres caracteres usando el Código Común UN/CEFACT. Este campo es obligatorio.
 - **description**: Breve descripción del mensurando. Metadato opcional.

Además de los campos mencionados, la entidad `AirQualityObserved` puede contener el campo `refWeatherObserved` que hace referencia a otro modelo llamado `WeatherObserved`. Aunque este campo es opcional y no se va a utilizar en este proyecto, su uso para vías futuras sería recomendable. `WeatherObserved` es una entidad que representa la observación de las condiciones meteorológicas en un lugar y momento determinados. Es muy útil para el análisis de la calidad del aire ya que las condiciones meteorológicas influyen en la calidad del aire así como en la predicción de la calidad del aire.

- **Alerta:** Corresponde a las alertas de calidad del aire que se envían a los usuarios participativos.

A continuación, en la figura 4.12 se muestra el modelo de persistencia de la aplicación.

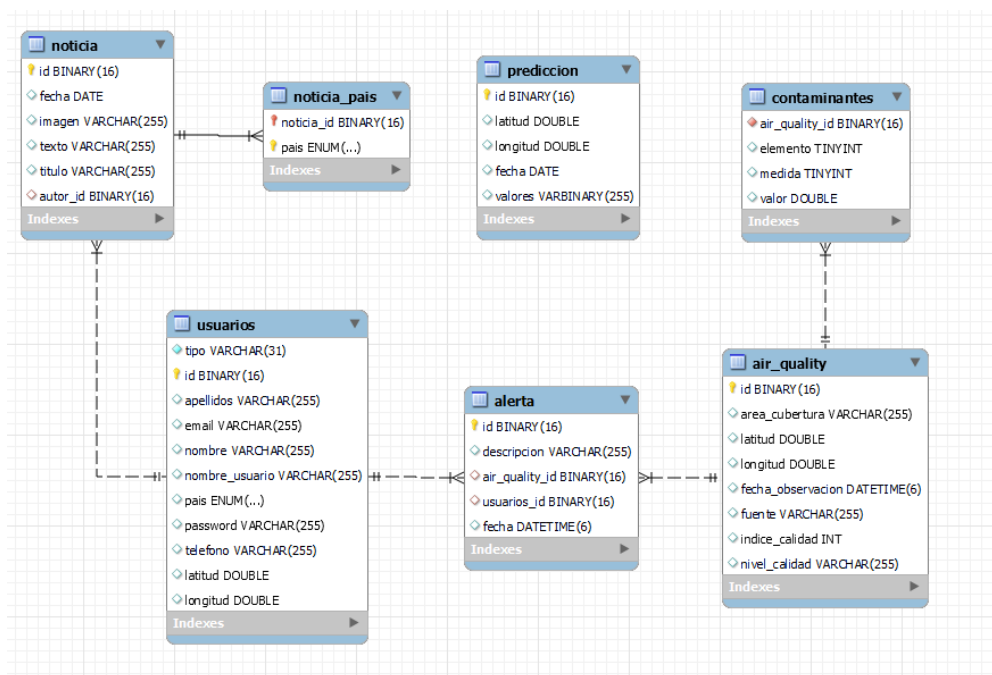


Figura 4.12: Modelo de persistencia

El modelo de persistencia se ha realizado con la herramienta de modelado de bases de datos MySQL Workbench. En este modelo se pueden observar las tablas de la base de datos y las relaciones entre ellas. Las diferencias entre el modelo de dominio y de persistencia son:

- **Jerarquía de usuarios:** Se ha decidido crear una tabla para todos los usuarios persistidos de la aplicación. En esta se encuentra un discriminante denominado tipo que especifica la clase de usuario.
- **Integración de coordenada y país:** No se ha creado una tabla para la entidad de coordenada y país. Se ha decidido integrar estos campos en otras tablas del sistema con la excepción de la lista de países en una noticia. Para este caso se ha creado una tabla donde la clave es una foreign key de la tabla de noticias.
- **Contaminantes:** Al igual que con país y coordenada, no se ha creado una tabla para las medidas y los elementos. Se ha decidido integrar estos campos en la tabla de contaminantes. Esta tabla no tiene primary key, porque se ha decidido que su identificador sea una foreign key de la tabla de AirQuality.

- **Alerta:** Se ha decidido crear una tabla para las alertas de calidad del aire la cual tiene dos relaciones many to one uni-direccional con la tabla de usuarios y la tabla de calidad del aire. Se ha decidido realizar un borrado en cascada con las dos relaciones.

4.4. Arquitectura de la aplicación

En este apartado se describe la arquitectura de la aplicación desarrollada. Se detallan los componentes de la aplicación, como se comunican entre ellos, la estructura de la aplicación y algunos métodos de diseño utilizados.

4.4.1. Componentes de la aplicación

El sistema está compuesto de cuatro componentes principales:

- **Frontend:** Será la interfaz entre los usuarios y el sistema. En este caso, se ha desarrollado una aplicación web con REACT que se ejecuta en el navegador del usuario.
- **API de calidad:** Es el componente principal encargado de la lógica de negocio de la aplicación. En este caso, se ha desarrollado una API SPRING que se ejecuta en un servidor Apache Tomcat (Tomcat). Esta API se comunica con la base de datos, la API de Google, la API de predicción y el ecosistema de FIWARE.
- **Api de predicción:** Es el componente que se encarga de obtener la predicción de la calidad del aire. Esta programado en Python donde se implementa un modelo de Machine Learning (ML) para la predicción de la calidad del aire.
- **Fiware:** Es el componente que se encarga de gestionar la información de contexto de las calidades del aire y también se encarga de la notificación en alertas de calidad del aire. En este caso, se ha utilizado el OCB de FIWARE.

La persistencia de los datos se realiza en una base de datos MySQL utilizada por la API de calidad y de predicción. Mientras que la información de contexto es persistida en una base de datos MongoDB utilizada por el OCB.

En la siguiente figura.4.13 se muestra el diseño de la arquitectura del proyecto y sus interacciones.

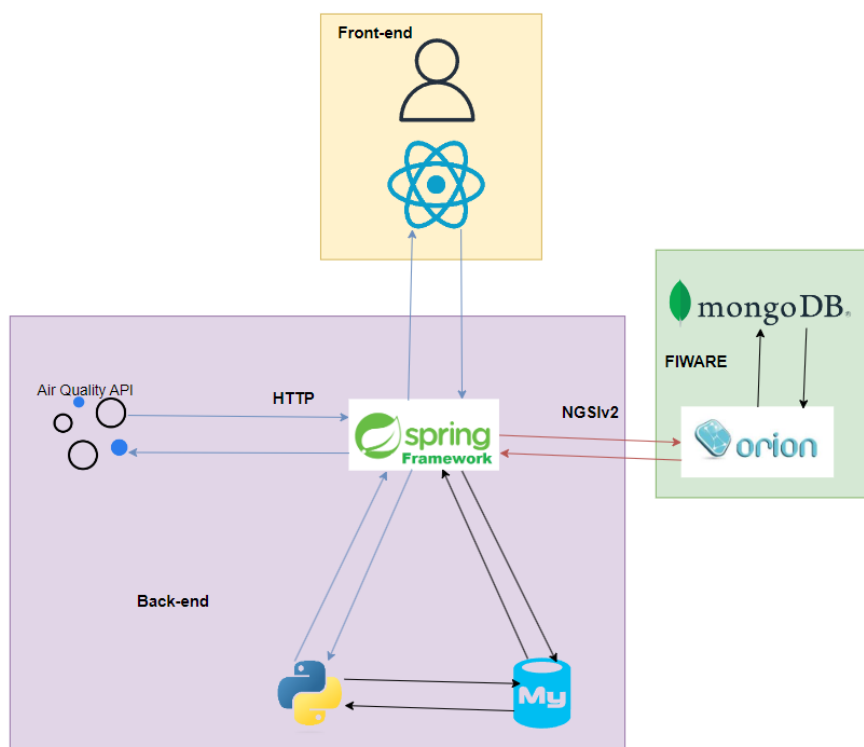


Figura 4.13: Diseño de la arquitectura del proyecto

Para la comunicación entre los diferentes componentes se ha utilizado HTTP y NGSI. La comunicación entre el frontend y la API de calidad tiene lugar mediante peticiones HTTP Representational State Transfer (REST). La comunicación entre la API de calidad y el OCB se produce mediante la API NGSIv2. La comunicación entre la API de calidad y la API de predicción se lleva a cabo mediante peticiones HTTP REST. También se ha aplicado HTTP para la comunicación entre la API de calidad y la API de Google.

El frontend y la API de predicción son componentes independientes de la API de calidad. Esto nos otorga mayor flexibilidad, mantenibilidad y eficiencia en el desarrollo de la aplicación. Además, al ser independientes, se pueden sustituir por otros componentes sin afectar al resto de la aplicación.

4.4.2. Arquitectura limpia

En este proyecto se ha seguido los métodos de la arquitectura limpia. La arquitectura Limpia es un enfoque de diseño de software que se especializa en la calidad; así como en la mantenibilidad y la evolución del software a lo largo del tiempo. La idea fundamental detrás de arquitectura limpia, es que la estructura y diseño de un software deben estar enfocados en separar las preocupaciones del negocio de las preocupaciones técnicas.

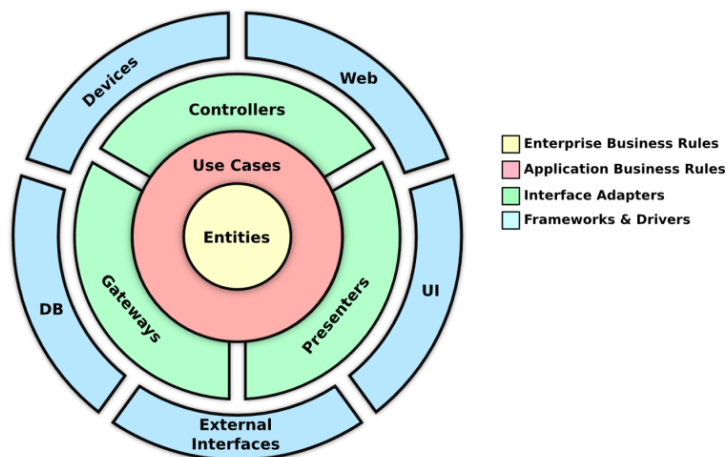


Figura 4.14: Estructura de la arquitectura limpia

[31]

La arquitectura limpia se basa en principios de diseño Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion (SOLID), y utiliza patrones de diseño para lograr una separación clara de las responsabilidades y una estructura modular. Esto se logra mediante la creación de capas de abstracción, donde las capas internas contienen la lógica de negocio; mientras que las capas externas contienen la infraestructura técnica.

4.4.3. Arquitectura REST

REST es un estilo arquitectónico para aplicaciones distribuidas inspirado en los principios de la web. REST se basa en la idea de que los recursos son identificados por Uniform Resource Locator (URL) y que las operaciones sobre estos recursos se realizan mediante los métodos HTTP estándar [32]. Los métodos HTTP utilizados son los siguientes:

- **GET:** Obtención información de un recurso.
 - **POST:** Creación de un recurso.
 - **PUT:** Modificación de un recurso.
 - **DELETE:** Eliminación de un recurso.
 - **PATCH:** Modificación parcial de un recurso.
-

En la siguiente tabla 4.3 se muestra todas las operaciones REST que se han utilizado en la aplicación [33].

Tipo	Ruta
Usuario Observador	
POST	/login
POST	/usuarios/participativos
GET	/airquality/{longitud}/{latitud}/resumen
Usuario Participativo	
GET	/prediccion/{latitud}/{longitud}
GET	/noticias/paises/{pais}
GET	/airquality/{latitud}/{longitud}
GET	/airquality/{id}/alertas
GET	/alertas/{idUserario}
DELETE	/alerta/{idAlerta}
DELETE	/alertas/{idUserario}
GET	/usuarioparticipativo/{idUserario}
PUT	/usuario/participativo/{idUserario}
Usuario Gestor	
GET	/usuarios/gestores/{id}
POST	/usuarios/gestores
PUT	/usuarios/gestores/{id}
GET	/usuarios/participativos
DELETE	/usuarios/participativos/{id}
GET	/usuarios/{id}/noticias
POST	/usuarios/{id}/noticias
PUT	/noticias/{id}
DELETE	/noticias/{id}

Tabla 4.3: Operaciones REST de la aplicación

4.5. Prototipo de la aplicación

En este apartado se muestra el prototipo de la aplicación AirQualityNest. Se han desarrollado los prototipos de las ventanas de registro, predicción de la calidad del aire y calidad del aire. Estos prototipos se han desarrollado con la herramienta Canva[34].

En la figura 4.15 se muestra el prototipo de la ventana de registro de sesión. En esta ventana se puede observar los campos necesarios para el registro de un usuario. También se puede ver las diferentes opciones disponibles en la ventana, así como el nombre y el logotipo de la aplicación.



El prototipo de la ventana de registro de sesión de la aplicación AirQualityNest presenta una interfaz con un encabezado que incluye el logotipo de la aplicación y el nombre 'AirQualityNest'. A la derecha del encabezado se encuentran cinco iconos representando funcionalidades: 'Actualidad', 'Noticias', 'Mapa', 'Predicción' y un botón 'Regístrate'. A la izquierda del encabezado se encuentra un botón 'Iniciar Sesión'. El cuerpo principal de la ventana tiene un fondo verde oscuro con el título 'Empieza' en blanco. Debajo del título, se encuentran cinco campos de entrada con sus respectivos labels: 'Nombre Usuario*' (con el valor 'Ejemplo de prototipo'), 'Contraseña*' (con caracteres ocultos por asteriscos), 'Nombre Completo*' (con el valor 'Prototipo TFG Ejemplo'), 'Correo Electrónico' (con el valor 'ejemplo@um.es') y 'Teléfono' (con el valor '657 891 234'). Debajo de estos campos, se encuentra un botón 'Regístrate' con un borde blanco y fondo verde oscuro.

Figura 4.15: Prototipo de la ventana de registro

En la figura 4.16 se muestra el prototipo de la ventana de predicción de la calidad del aire para el usuario participativo. En esta ventana se puede visualizar la predicción de la calidad del aire para la ubicación del usuario. Esta información se presenta como tabla sobre información de la calidad del aire para los próximos días.

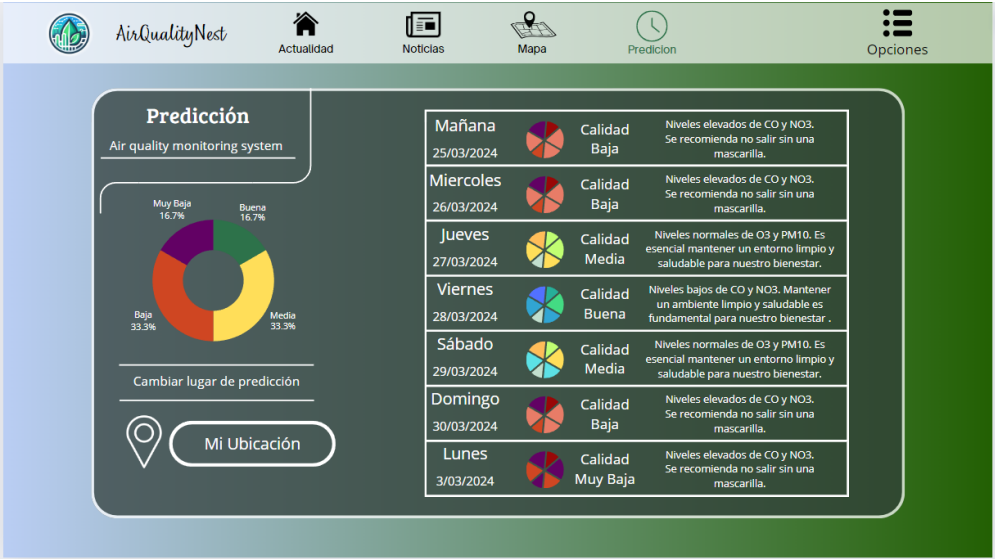


Figura 4.16: Prototipo de la predicción de la calidad del aire

En la figura 4.17 se muestra el prototipo de la ventana de calidad del aire para el usuario participativo. En esta ventana se puede visualizar la calidad del aire en tiempo real para la ubicación del usuario. Esta información se muestra de diferentes formas y medidas. Así como consejos para mejorar la calidad del aire.

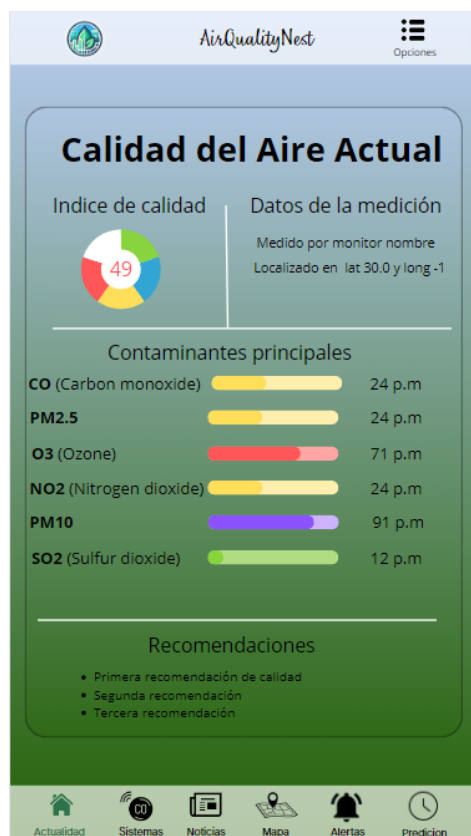


Figura 4.17: Prototipo de la calidad del aire para el usuario participativo

5. Desarrollo e implementación de la aplicación

En este capítulo se describirá el desarrollo e implementación de la aplicación. Esta sección se divide en cuatro secciones principales: API de Calidad del Aire, API de predicción de la calidad del aire, configuración de Fiware y desarrollo del Frontend.

5.1. Desarrollo de la API de Calidad del Aire

5.1.1. Descripción

La API de Calidad del Aire es una API REST desarrollada en JAVA haciendo uso del framework Spring Boot. Esta API es el elemento principal de la aplicación, la encargada de la comunicación con todos los demás servicios del backend y ofrece los endpoints necesarios para el frontend. Además de la comunicación con los demás servicios, la API de Calidad del Aire es la responsable de la persistencia de los datos en la base de datos, de la lógica de negocio y de la autenticación de los usuarios. Este es un proyecto Maven donde se ha utilizado el plugin de Spring Boot para la creación del proyecto y otras dependencias como Spring Security, Spring Data JPA, Spring Web, Lombok, entre otras.

5.1.2. Arquitectura

La arquitectura de la API de Calidad del Aire está basada como se ha explicado anteriormente en una arquitectura por capas/ arquitectura limpia. Con esta arquitectura se busca separar las responsabilidades de cada capa y hacer que cada capa sea independiente de las demás. La arquitectura de la API de Calidad del Aire se divide en las siguientes capas:

- **Capa de Presentación:** Es la que recibe y envía las peticiones del frontend. También recoge las respuestas del sistema de alertas.
- **Capa de Lógica de Negocio:** Esta capa contiene la lógica de negocio de la aplicación. En esta se encuentran los servicios encargados de la lógica de negocio de la aplicación.

- **Capa de Persistencia:** Se ubican los repositorios de la aplicación. Es la responsable de la persistencia de los datos en la base de datos.
- **Capa de Dominio:** Se encuentran las entidades que representan los objetos de dominio de la aplicación.
- **Capa de Soporte:** Se hallan los DTOs, los mappers y los servicios de autenticación que sirven de soporte para la aplicación.

5.1.3. Capa de Presentación

En esta capa encontramos tres componentes esenciales: el `FrontRestController`, el `BackRestController` y el `RestExceptionHandler`. El `FrontRestController` es el ocupante de recibir las peticiones del frontend y enviar las respuestas al frontend. El `BackRestController` es el encargado de recibir las respuestas del sistema de subscripciones de FIWARE. El `RestExceptionHandler` es el responsable de manejar los errores de la aplicación. En los dos controladores se hace uso de los servicios de la capa de lógica de negocio para realizar las operaciones necesarias, los mapeadores y DTOs de la capa de soporte para la conversión de los objetos.

5.1.4. Capa de Lógica de Negocio

Es la capa responsable de la lógica de negocio de la aplicación. Esta capa se encuentra dividida en diferentes paquetes, cada uno de ellos con una responsabilidad diferente.

Paquete de servicio

En este paquete se encuentran los servicios de la aplicación. Estos son los encargados de la lógica interna de la aplicación. Aquí tenemos un servicio por cada entidad de la aplicación. En cada servicio se encuentra su interfaz, implementación y excepción. En la implementación de los servicios se hace uso de los repositorios de la capa de persistencia y, si es necesario, de otros servicios.

De estos servicios podemos destacar el de usuario que se divide en tres elementos diferentes:

- **Servicio Usuario:** Es el encargado de la mayoría de las operaciones de los usuarios. En este se encuentran las operaciones de creación, actualización, eliminación y consulta de los usuarios. Es abstracto debido a que la lógica de la aplicación diferencia entre usuarios participativos y usuarios gestores. En este se localizan dos operaciones abstractas que especifican el tipo de usuario que pertenece el servicio.
 - **Servicio Usuario Participativo:** Extiende el servicio de usuario e implementa las dos funciones abstractas que especifican el tipo de usuario que pertenece el
-

servicio. También se sobrescribe la función `comprobarCampos` y se implementa una nueva funcionalidad para obtener usuarios por coordenadas.

- **Servicio Usuario Gestor:** Solamente extiende y implementa las dos funciones del servicio usuario al igual que el anterior.

El diseño de estos servicios se ha realizado de esta manera para que en un futuro se puedan añadir nuevos tipos de usuarios sin tener que modificar el código ya existente. También, si fuese necesario, se podrían añadir nueva funcionalidad a los servicios de usuario participativo y gestor.

Otro servicio a destacar es `ServicioCoordinadorAirQuality` que asume toda la gestión de la lógica de obtención de datos cuando son solicitados por el usuario. De igual modo el servicio de predicción es el encargado de la gestión de la lógica de obtención de la predicción de la calidad del aire. Esta lógica se explicará en la sección de la lógica de negocio más importante.

Paquete de mensajería

Aquí se encuentran las clases encargadas de la comunicación con las APIs externas y una simulación de sensor para mantener a los usuarios participativos actualizados.

Dentro del paquete se encuentra la carpeta `service` que contiene los servicios de FIWARE, de la API de predicción de la calidad del aire y de Google. Al igual que en el paquete de servicio anterior, en este paquete se encuentran las interfaces, las implementaciones y las excepciones de los servicios. La comunicación es gracias a la librería `WebClient` de Spring. El servicio de Google y de predicción solo cuentan con la función solicitar los datos de la calidad del aire y de la predicción de la calidad del aire respectivamente para una coordenada dada. Estos servicios utilizan mapeadores y DTOs para la conversión de las calidades de aire. Además de esta funcionalidad, FIWARE tienen también la funcionalidad de almacenar una calidad, actualizar una calidad y obtener la fecha de la última calidad almacenada para una coordenada dada.

En el paquete mensajería se encuentran la clase `AppConfig` con la configuración de la librería `WebClient` y una clase llamada `SensorCalidad`. Esta clase tiene `Scheduled` que está programado para que cada media hora se obtengan los datos de la calidad del aire de la API de Google para todos los usuarios participativos. Al disponer de la API de Google que tiene un límite de peticiones gratuitas, se ha decidido que la obtención de datos sea solo dos usuarios participativos. La ubicación de estos usuarios se ha decidido que sean en la ciudad de Murcia y la capital de India, Nueva Delhi.

La segunda funcionalidad es la llamada interna del Scheduled de la clase SensorCalidad. 5.2 muestra el diagrama de secuencia de esta funcionalidad:

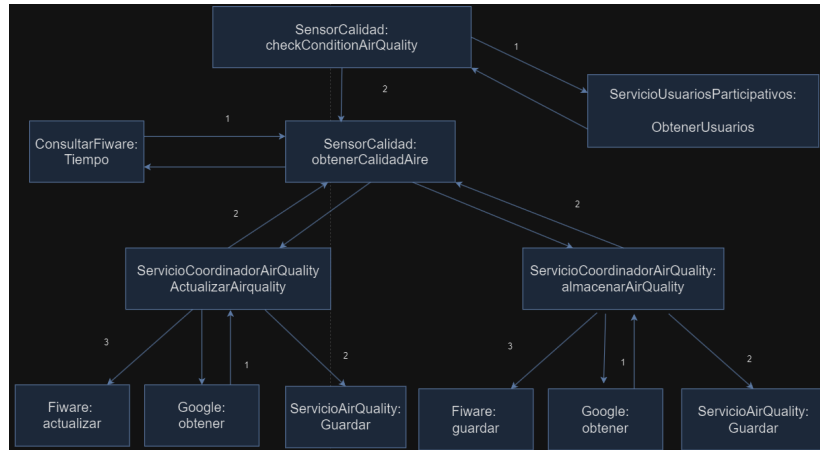


Figura 5.2: Diagrama de secuencia de la llamada interna del Scheduled

En esta función se debería de obtener todos los usuarios participativos para mantener actualizada la calidad del aire de cada usuario. Pero debido a que la API de Google tiene un límite de peticiones gratuitas, se ha decidido que la obtención de datos sea solo de dos usuarios participativos. Después de obtener los usuarios participativos, se obtiene la fecha de la calidad del aire almacenada en la base de datos de Fiware para la coordenada especificada del usuario. Si la fecha es nula o es antigua se llama a las funciones anteriores del servicio de coordinador de calidad del aire.

La tercera funcionalidad es la generación de alertas. 5.3 se muestra el diagrama de secuencia de esta funcionalidad:

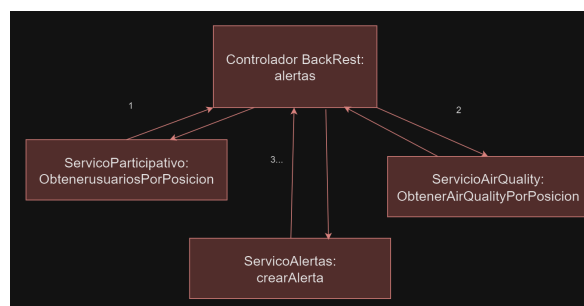


Figura 5.3: Diagrama de secuencia de la generación de alertas

Cuando se llame a la función definida en el controlador back por parte de FIWARE, se obtiene todos los usuarios participativos en las coordenadas de la calidad del aire. También se obtiene la última calidad del aire para esas coordenadas. Esta calidad nunca puede ser nula debido a que debe de estar almacenada en base de datos. Una vez se

tiene la información anterior se recorren todos los usuarios participativos creando una alerta para cada uno de ellos.

La cuarta funcionalidad es la obtención de predicción de la calidad del aire cuando es solicitada por el usuario. 5.4 se muestra el diagrama de secuencia de esta funcionalidad:

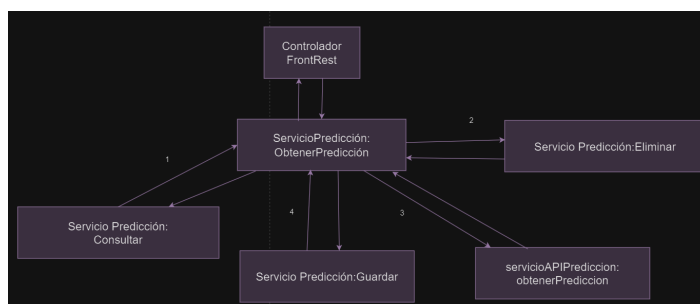


Figura 5.4: Diagrama de secuencia de la obtención de la predicción de la calidad del aire

Esta función es muy similar a la de la obtención de la calidad del aire. Se consulta en la base de datos local si la predicción de la calidad del aire es nula o antigua. Si es antigua, se elimina la predicción de la base de datos local y se obtiene la predicción de la calidad del aire de la API de predicción de la calidad del aire y se almacena en la base de datos local al igual que si es nula. Lo mismo sucede en la obtención de la calidad del aire, pues se devuelve la predicción de la calidad del aire al front controller.

5.1.5. Capa de Persistencia

En esta otra capa se hallan los repositorios de la aplicación, que son interfaces extendidas de la interfaz JpaRepository de Spring Data JPA. Además de las operaciones básicas ofrecidas por JpaRepository, se han añadido otras operaciones personalizadas. Estas son:

- **RepositorioUsuario:** Se han adjuntado operaciones para obtener un listado de los usuarios por tipo, por coordenadas, por nombre de usuario y por nombre de usuario y contraseña.
- **RepositorioPrediccion:** Solo se ha agregado una operación para obtener la predicción de la calidad del aire por coordenadas.
- **RepositorioNoticia:** Contiene dos funciones para obtener las noticias por país y autor.
- **RepositorioAlerta:** Tiene dos consultas adicionales: una para obtener las alertas por usuario y otra para eliminar las alertas por usuario.
- **RepositorioAirQuality:** Se han añadido una operación para obtener la calidad del aire más reciente por coordenadas.

Para las funciones que obtienen una lista de entidades, se devuelve un elemento Page para la paginación de los datos en la aplicación.

5.1.6. Capa de Dominio

En esta capa se encuentran las entidades de la aplicación. Son las clases que representan los objetos de dominio de la aplicación. Cada entidad está usando la anotación @Data de Lombok para la generación de los métodos getter y setter. Así mismo se han añadido las anotaciones de JPA para la persistencia de los datos en la base de datos. Destacar que los ids de las entidades son de tipo UUID para garantizar la unicidad de los datos y son generados de forma automática por la base de datos. Los elementos Coordinada y Contaminante son entidades embebidas; ambas se diferencian, ya que Contaminante tiene una tabla propia en la base de datos mientras Coordinada se almacena dentro de sus referencias.

La entidad Usuario es una clase abstracta que tiene dos clases hijas UsuarioParticipativo y UsuarioGestor. Para esta jerarquía se ha decidido almacenar en una sola tabla con una columna que indica el tipo de usuario. Esto es debido a que el único campo que diferencia a los dos tipos de usuarios es el campo coordenadas perteneciente al usuario participativo.

En cuanto a la entidad Alerta contiene dos campos ManyToOne unidireccionales; uno con la entidad Usuario y otro con la entidad CalidadAire. Estos campos están anotados con el atributo @JoinColumn para indicar la columna de la base de datos que se encarga de la relación y el atributo @OnDelete para indicar que al eliminar un usuario o una calidad del aire se eliminan todas las alertas asociadas a ellos.

5.1.7. Capa de Soporte

Es una capa que contiene diferentes paquetes y cada uno tiene una responsabilidad diferente.

Paquete de DTO

En este paquete se encuentran los DTOs de la aplicación. Los DTOs son clases que se utilizan para la conversión de los objetos de la aplicación. Dependiendo de la necesidad del objeto se pueden tener diferentes DTOs para un mismo objeto. En la aplicación se han utilizado DTOs de petición y de respuesta. Los de petición se utilizan para la conversión de los objetos que se reciben en las peticiones del frontend y backend. Los DTOs de respuesta se utilizan para la conversión de los objetos que se envían en las respuestas del frontend. La mayoría de los DTOs de la aplicación tienen la anotación @Data de Lombok y mapeadores para la conversión de los objetos. Todos los DTOs de petición del frontend tienen validación a través de la librería de javax.validation.

Paquete de Mapper

En este paquete se encuentran los mapeadores de la aplicación. Los mappers son clases que se utilizan para la conversión de objetos en la aplicación. Al ser una aplicación con una arquitectura por capas, se necesita convertir los objetos de una capa a otra. En la aplicación se han usado algunos mapeadores complejos para la conversión de los objetos de la aplicación a objetos de FIWARE y Google o viceversa.

Paquete de Seguridad

En este paquete se encuentran las clases encargadas de seguridad de la aplicación. En la aplicación se ha utilizado Spring Security para la autenticación de los usuarios. También se ha utilizado JWT para la generación de los tokens de autenticación de la librería de jsonwebtoken. En este paquete encontramos los 4 componentes esenciales de Spring Security:

- **SecurityConfig:** Esta clase es la responsable de la configuración de Spring Security. Aquí se han configurado los cors, las rutas y el manejador de excepciones de seguridad.
- **JwtRequestFilter:** Es la encargada de la validación del token de autenticación. Se ha sobrescrito el método `doFilterInternal` para la validación del token ayuda de la clase `JwtTokenUtil`.
- **JwtTokenUtil:** Esta clase es la responsable de la generación y validación del token de autenticación. Implementa los métodos de generación y validación del token.
- **UserAuthenticationProvider:** Se ocupa de la autenticación de los usuarios. Se ha sobrescrito el método `authenticate` para la autenticación de los usuarios. Para la autenticación se ha utilizado el servicio de usuario participativo de la capa de lógica de negocio. Debido a que el método `obtenerUsuarioPorNombreUsuarioYPassword` es genérico de la clase `Usuario` se puede usar cualquiera de las dos implementaciones.

Además de estos componentes, en el paquete se encuentran dos directorios llamados `dto` y `rest` que contienen los DTOs y los controladores necesarios para la autenticación de los usuarios. Este controlador tiene dos funciones: una para iniciar sesión y otra para registrar un usuario.

Paquete Utils

En este paquete se encuentran diferentes clases de utilidades de la aplicación. Las más importantes son las clases:

- **ResponseFactory:** Responsable de la creación de las respuestas de error de la aplicación.
- **UtilsJPA:** En esta clase se permite realizar varias operaciones de persistencia en la misma transacción.
- **ErrorResponse:** Es el objeto que se devuelve en caso de error en la aplicación.
- **Clases de validación:** En este paquete se encuentran las clases de validación de enumerados de la aplicación.

5.2. Desarrollo de la API de predicción de la calidad del aire

5.2.1. Descripción

La API de predicción de la calidad del aire es una API desarrollada en Python haciendo uso de diferentes librerías. Esta ofrece un endpoint REST para obtener la predicción de la calidad del aire de los próximos 7 días para una coordenada dada. Se ha desarrollado de manera independiente a la API de Calidad del Aire. Se ha decidido así para que en un futuro se pueda cambiar el modelo de predicción sin tener que modificar la API de Calidad del Aire. Es importante destacar que no es el elemento principal del TFG, sino un elemento complementario.

Las librerías utilizadas para el desarrollo de la API de predicción son las siguientes:

- **Pandas:** Necesario para la manipulación y análisis de los datos.
 - **Flask:** Es una librería de Python utilizada para ofrecer un endpoint REST.
 - **SQLAlchemy:** Sirve para la comunicación con la base de datos SQL.
 - **Scikit-learn:** Es una de las librerías más utilizadas para el aprendizaje automático en Python, la que se ha utilizado para la creación un modelo RandomForestRegressor para la predicción de la calidad del aire.
 - **Numpy:** Utilizada para el manejo y transformación de datos numéricos.
 - **Pickle:** Se ha utilizado para almacenar el modelo de predicción en un archivo dentro de la carpeta "models".
-

5.2.2. Estructura y elementos de la API

La API de predicción de la calidad del aire se divide en dos directorios diferentes: El directorio app donde se encuentra la lógica de la aplicación y el directorio models donde se halla el modelo de predicción. En la carpeta raíz de la API está el archivo run.py que es el encargado de ejecutar la aplicación y el archivo requirements.txt que contiene las dependencias de la aplicación.

Dentro de app tenemos dos elementos: El archivo `__init__.py` y el `main.py`. Estos, junto a run.py, se encargan de iniciar la aplicación. También en la misma carpeta se ubican los dos elementos más importantes de la API que son `model.py` y `api.py`.

model.py

En este archivo se encuentra la función train que es la encargada de crear, entrenar y guardar el modelo de predicción. El primer paso del modulo es la obtención de los datos de la base de datos. Estos son obtenidos a través de SQLAlchemy y los datos que se obtienen son la latitud, la longitud, el índice de calidad y la fecha de obtención. Con la fecha se obtiene el mes, el día de la semana y del año que son los utilizados como características. Para mejorar el predictor se ha decidido añadir como características el índice de calidad del aire de los 6 días anteriores después de hacer un pequeño estudio.

Una vez se tienen los datos se definen las características con los datos anteriores y la etiqueta con el índice de calidad del aire, que es el objetivo de la predicción. Se escalan los datos con StandardScaler y se define el modelo de predicción.

Se ha decidido utilizar un modelo supervisado de regresión. Este es el más adecuado para este problema, ya que se busca predecir un valor numérico. Se ha probado con diferentes modelos como el modelo de regresión lineal y el modelo Support Vector Regression (SVR). El modelo que mejor resultado ha dado ha sido el modelo RandomForestRegressor. Este es un modelo de regresión que está basado en un bosque aleatorio con 250 árboles y una semilla aleatoria para reproducibilidad.

Por ultimo se entrena el modelo con los datos y se guarda en un archivo con la librería pickle.

api.py

En este modulo encontramos la función predict que es la encargada de llamar al modelo de predicción y obtener la calidad del aire de los próximos 7 días. Para llamar a esta función se ha creado un endpoint REST que recibe la latitud y la longitud de la coordenada gracias a Blueprint de Flask.

El proceso una vez se recibe la latitud y la longitud es la siguiente:

- Para mantener la misma lógica que en FIWARE se convierte la latitud y la longitud en valores enteros para agrupar por grados. Se obtienen los índices de
-

calidad del aire de los últimos 6 días para el intervalo de un grado de latitud y longitud.

- Al no disponer de suficientes datos en algunas coordenadas, se ha decidido que si no se obtienen 6 días se completan con los datos existentes. En una aplicación con un monitoreo más amplio este problema no existiría.
- Se realiza un bucle para obtener la predicción de la calidad del aire de los próximos 7 días. Para cada iteración/día se simula el paso del tiempo introduciendo al modelo la información correspondiente a la iteración/día. Esto se consigue gracias a la función predict del modelo de predicción que devuelve la predicción de la calidad del aire para los valores introducidos. Este valor se almacena en una lista de predicciones que conforman la respuesta y se actualiza la información de los días anteriores simulando el paso del tiempo.
- Por último se devuelve la lista de predicciones.

5.2.3. Análisis de datos y entrenamiento del modelo

Para estimar las mejores características para el modelo de predicción se ha realizado un pequeño análisis de los resultados de la calidad del aire para las diferentes características. Para esto se ha entrenado el modelo con el 90% de los datos y se ha probado con el 10% restante. Los resultados obtenidos han sido los siguientes:

Parámetros de entrada	MAE
Sin Ordenar 0 días	11,3356
Ordenado 0 días	7,7642
1 día anterior	5,7613
2 días anteriores	4,9886
3 días anteriores	6,454
4 días anteriores	5,2666
5 días anteriores	6,236
6 días anteriores	3,8459
7 días anteriores	7,1319
8 días anteriores	5,9296
9 días anteriores	9,9552

Tabla 5.1: Resultados de la predicción de la calidad del aire

Como se puede observar en la tabla 5.1 la tendencia inicial es favorable conforme aumenta el número de días anteriores. Ahora bien la predicción empeora cuando se sobrepasa el séptimo día y anteriores introducidos como características. Por lo tanto, se ha decidido que la mejor predicción obtenida es con los 6 días anteriores. El Mean Absolute Error (MAE) obtenido es de 3,8459 que es un valor aceptable sabiendo que el índice de calidad del aire va de 0 a 100 lo que supone un error del 3,85%.

5.3. Configuración de Fiware

La configuración se debe de realizar una vez se inicia el contenedor de Fiware Orion por primera vez o si se elimina la base de datos utilizada por Fiware Orion. Esta configuración consiste en crear una suscripción en Fiware Orion. Cuando se actualice la entidad de calidad del aire y cumpla una condición, entonces se envía una petición a la API de Calidad del Aire para la creación de las alertas. En un primer momento se ha intentado realizar la suscripción con una condición compleja comprobando los valores de todos los contaminantes, pero el sistema de suscripciones de Fiware no lo permite. Por lo tanto, la condición que se ha implementado es que cuando el índice de calidad del aire sea menor a 40, se remita una petición para crear las alertas. El formato de la suscripción se muestra en el anexo A.2.

5.4. Desarrollo del Frontend

5.4.1. Descripción

Es el elemento de la aplicación que permite a los usuarios interactuar con la funcionalidad de AirQualityNest. Se ha desarrollado en React haciendo uso de diferentes librerías. Los dos principales objetivos de la interfaz son: la sencillez y su facilidad de uso y que sea responsiva. Es muy importante que la aplicación sea responsiva, ya que la mayoría de los usuarios utilizarán la aplicación en dispositivos móviles. El Frontend corresponde a una Single Page Application (SPA) que se divide en diferentes componentes. Una SPA es una aplicación web que se carga una sola vez y se actualiza dinámicamente sin necesidad de recargar la página. Esto se consigue gracias a la librería de React Router Dom como se explicará más adelante.

Es importante destacar que no se ha podido implementar la funcionalidad relacionada con el apartado de noticias debido a falta de tiempo. Esta funcionalidad era una funcionalidad opcional y no afecta a la funcionalidad principal de la aplicación.

5.4.2. Herramientas y librerías utilizadas

Para el desarrollo del Frontend se han utilizado diferentes herramientas y librerías.

Vite

Vite es una herramienta de compilación que tiene como objetivo proporcionar una experiencia de desarrollo más rápida y sencilla para proyectos web modernos [35]. Además de su popularidad, Vite es una herramienta más potente y rápida que otras herramientas de compilación. Por esta razón, he decidido utilizar Vite en el desarrollo del proyecto.

Bootstrap

En el proyecto se ha utilizado la versión 5 de React Bootstrap. Bootstrap es un framework de diseño web que nos permite crear interfaces web de forma rápida y sencilla. Bootstrap otorga una serie de estilos CSS y componentes HTML que permiten crear una aplicación web responsive y con un diseño atractivo [36].

Los componentes de Bootstrap que se han utilizado en el proyecto son los siguientes:

- **Nav y Offcanvas:** Utilizados para la creación de la barra de navegación.
- **Modal:** Requeridos para el desarrollo del modal de eliminar usuario.
- **Card:** Empleado en la creación de las tarjetas de diferentes elementos.
- **Form:** Necesario para la construcción de los formularios de la aplicación.
- **Spinner:** Utilizado para la creación del spinner de carga.
- **Alert:** Usado para los mensajes tanto de error como de información.
- **Button:** Empleado en toda la aplicación para los botones.
- **Acordeón:** Requerido para el uso de la leyenda.
- **ProgressBar:** Utilizado para la representación de los contaminantes de la calidad del aire.

Los componentes Bootstrap más utilizados en nuestra aplicación han sido Col, Row y Container. Estos componentes son la implementación de Bootstrap de CSS Grid y Flexbox, que permiten crear una estructura de página responsive y con un diseño atractivo.

React Router Dom

React Router Dom es una librería que permite agregar enrutamiento en una aplicación web de React. Esta ofrece a los usuarios navegar entre diferentes páginas de la aplicación sin necesidad de recargar la página [37] debido a que es una SPA. Además de esta funcionalidad, React Router Dom ha permitido navegar entre diferentes rutas de la aplicación dependiendo de la opción seleccionada por el usuario gracias a la función `UseNavigate`.

CSS

CSS (Cascading Style Sheets) parten de un concepto simple pero muy potente: aplicar estilos a uno o varios documentos de forma automática y masiva. Se le denomina estilos en cascada porque se lee, procesa y aplica el código desde arriba hacia abajo y en el caso de existir ambigüedad, se siguen una serie de normas para resolver dicha ambigüedad [38]. En la aplicación se ha utilizado CSS para la personalización de los estilos de la aplicación.

JavaScript

JavaScript es un lenguaje de secuencias de comandos que te permite crear contenido de actualización dinámica, controlar multimedia, animar imágenes, etc [39]. En la aplicación se ha utilizado JavaScript para la creación de funciones que permiten la interacción del usuario con la aplicación.

Otras librerías

- **Ag-Grid:** Es una librería de React que permite crear tablas de datos de forma sencilla y con un diseño atractivo. Esta librería ha posibilitado la creación de la tabla de usuarios. Este componente permite paginar, filtrar y ordenar los datos de forma sencilla. Además, hemos personalizado el diseño de las tablas de datos para que se adapten al diseño de nuestra aplicación. El punto negativo es que es parcialmente de pago y sus componentes no son responsivos.
 - **Echarts:** Es una librería de React que permite crear gráficos de datos de forma sencilla y con un diseño atractivo. Esta librería ha sido utilizada para la creación de los gráficos de calidad del aire. Sus puntos fuertes son: su gratuidad y la gran cantidad de gráficos personalizables. El punto negativo de esta librería es que no es responsiva.
 - **Spinners:** Es una librería de React que permite crear spinners de carga con una estética diferente. Se utiliza como acompañante del componente `Spinner` de Bootstrap.
-

- **Prop-Types:** Corresponde a una librería de React que permite la validación de las propiedades de los componentes. Se ha usado para la validación de las propiedades de los componentes de la aplicación.

Destacar que se ha servido de una API externa a la aplicación. Esta ha permitido la obtención del nombre de la ciudad más cercana a unas coordenadas dadas. Esta API es la de Nominatim de OpenStreetMap [40].

5.4.3. Estructura y elementos del Frontend

A continuación se describen los principales elementos y la estructura del Frontend. Se divide en diferentes carpetas que contienen una serie de elementos con un uso específico.

Elementos iniciales

- **index.html:** Es el único archivo HTML de la aplicación. En este archivo se encuentra el elemento div con el id root donde se renderiza la aplicación y las cabeceras que especifican los estilos, el logotipo y el título de la aplicación.
- **index.css:** Se encuentran los estilos del atributo body y root de la aplicación. Estos son los que dan la estructura, estética y color al fondo de la aplicación.
- **main.jsx:** Este archivo es el punto de entrada principal para una aplicación React.
- **App.jsx:** Es el componente donde se enruta la aplicación. Aquí se encuentran todas las rutas de la aplicación con sus respectivos componentes gracias a la librería de React Router Dom. Además, se autentifican las rutas/componentes de la aplicación como se verá más adelante.
- **App.css:** La mayoría de los estilos globales de la aplicación se encuentran en este archivo.

Paquete component

En este paquete se encuentran los componentes que son utilizados por otros componentes de la aplicación. Algunos de estos son reutilizables y otros son específicos de la aplicación.

Los componentes más importantes de este paquete son los siguientes:

- **Menu:** Este componente incorpora a **PersonalOffcanvas**, que juntos son los encargados de la creación de la barra de navegación de la aplicación. Para esta creación se necesita introducir como propiedad un objeto con las rutas disponibles.

- **GradeComplejo y GradeSimple:** En estos componentes se encuentran las gráficas de la librería Echarts. Son utilizados para mostrar el índice de calidad del aire actual y el de las predicciones.
- **AlertaGenerico:** Es un componente genérico al que se le introduce el mensaje y tipo de alerta que se quiere mostrar.

Paquete pages

Es el paquete donde se encuentran los componentes utilizados por las rutas de la aplicación. Cada componente de este paquete es una o más páginas de la aplicación gracias al renderizado condicional de React. En todas estas páginas se han utilizado el componente **Menu** y **Footer** para la creación de la barra de navegación y el pie de página respectivamente. También se ha utilizado el hook personalizado llamado `useFech` para la comunicación con el backend. Más adelante se explicará en detalle el contenido de cada página y su funcionalidad.

Otros paquetes

- **hooks:** Un hook es una función donde se realizan otros hooks o funciones de la aplicación. Existen dos hooks personalizados de la aplicación. El primero es `useFetch`, que se encarga de la comunicación con el backend. El segundo es `useGetCityFromCoordinates`, que se encarga de obtener la ciudad de unas coordenadas dadas.
 - **utils:** En este paquete se encuentran las funciones de utilidad de la aplicación. Algunas de estas funciones son el tratamiento de mensajes, tratamiento de errores y la obtención de la localización actual del usuario.
 - **styles:** La mayoría de los estilos css de la aplicación se encuentran en este paquete. Los componentes que han necesitado estilos específicos se han creado en esta carpeta.
 - **security:** Se encuentran los componentes de control y autenticación de sesión de la aplicación. Utilizando `localStorage` se ha creado un sistema de autenticación de sesión. Tenemos dos componentes encargados de esta tarea: `PrivateComponent` y `Autenticacion`. El primero utiliza el hook de `useAuth` y comprueba si el usuario está autenticado; si no lo está, redirige a la página de inicio de sesión. El segundo es el encargado de la autenticación del usuario con las funciones de login y logout.
 - **config:** Contiene el archivo `constast` que establece todas las constantes de la aplicación. Aquí se encuentran iconos, rutas, entre otras.
 - **assets:** En este paquete se encuentran el logotipo de la aplicación.
-

5.4.4. Mapa de navegación

El mapa de navegación de la aplicación se muestra en la figura 5.5. La aplicación se divide en tres secciones principales, dependiendo de la autenticación necesaria.

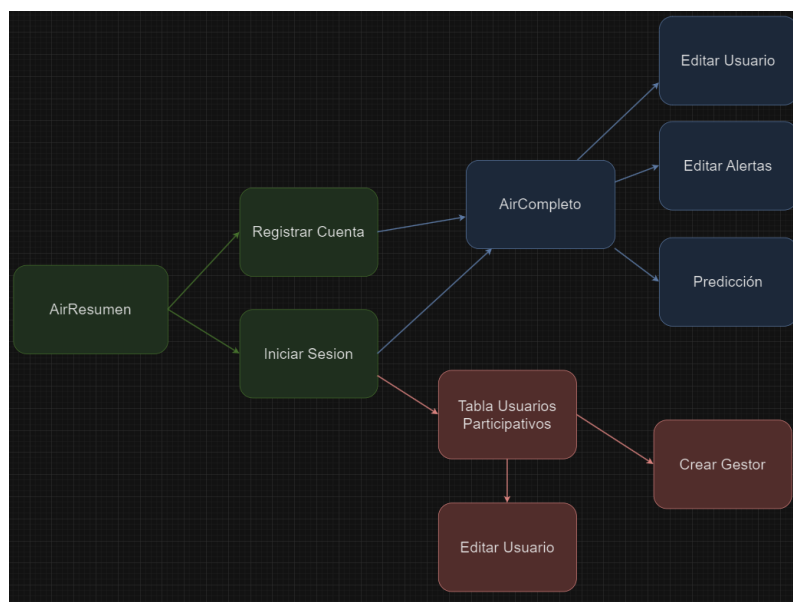


Figura 5.5: Diagrama de navegabilidad de la aplicación

5.4.5. Páginas de la aplicación

A continuación se describen las páginas de la aplicación divididas por tipo de autenticación. Es importante indicar que se informa al usuario de cualquiera transacción realizada por este o del posible error retornado por la aplicación en formato de alerta. Esta se abre en la parte superior derecha de la pantalla durante unos segundos. A continuación se muestra un ejemplo de alerta en la figura 5.6.

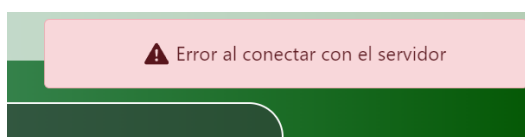


Figura 5.6: Mensaje en formato de alerta

Uno de los componentes más importantes es la barra de navegación de la aplicación. Esta es común en todas las paginas, solamente se modifica el contenido disponible dependiendo del usuario. Destacar que para la visualización de la barra de navegación hay que pulsar el icono de la hamburguesa situado en la parte superior derecha de la aplicación.

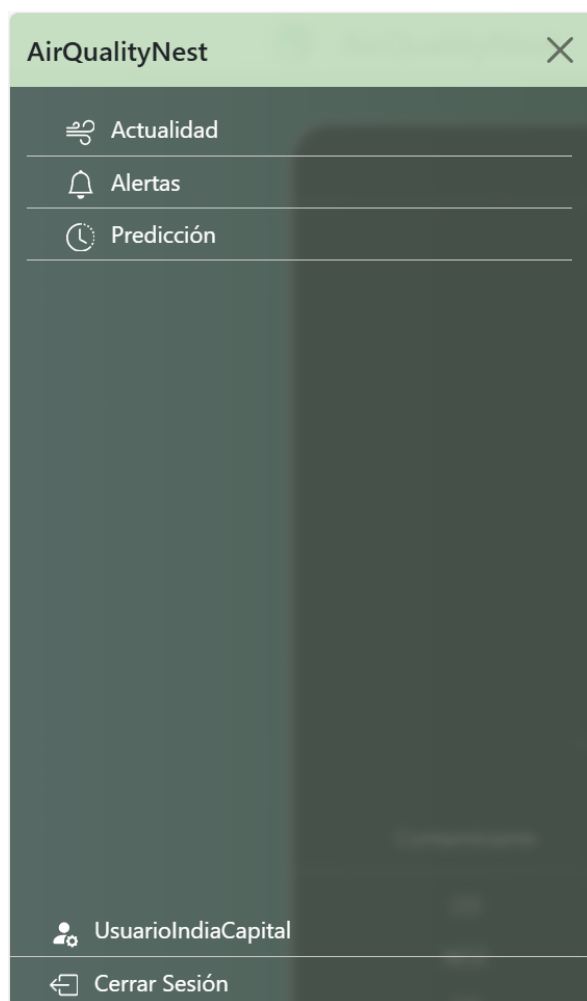


Figura 5.7: Barra de navegación de la aplicación

Como se puede observar en la figura 5.7, la barra de navegación contiene las diferentes rutas con un icono y un texto asociado. En la parte inferior del Offcanvas se muestra el nombre de usuario para modificar su perfil y un botón para cerrar sesión en caso de que el usuario este autenticado.

No autenticado

Actualidad

Consiste en la página inicial de la aplicación. Se presenta la calidad del aire de hoy en las coordenadas actuales del usuario. Si el usuario no ha permitido la localización en el navegador, se muestra un error solicitando que se permita la localización para poder mostrar la información.

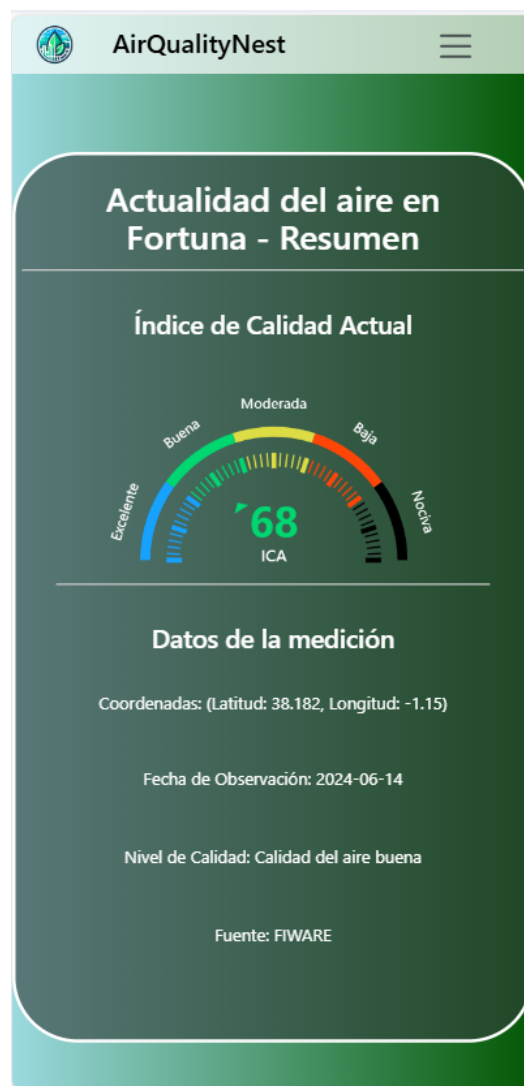


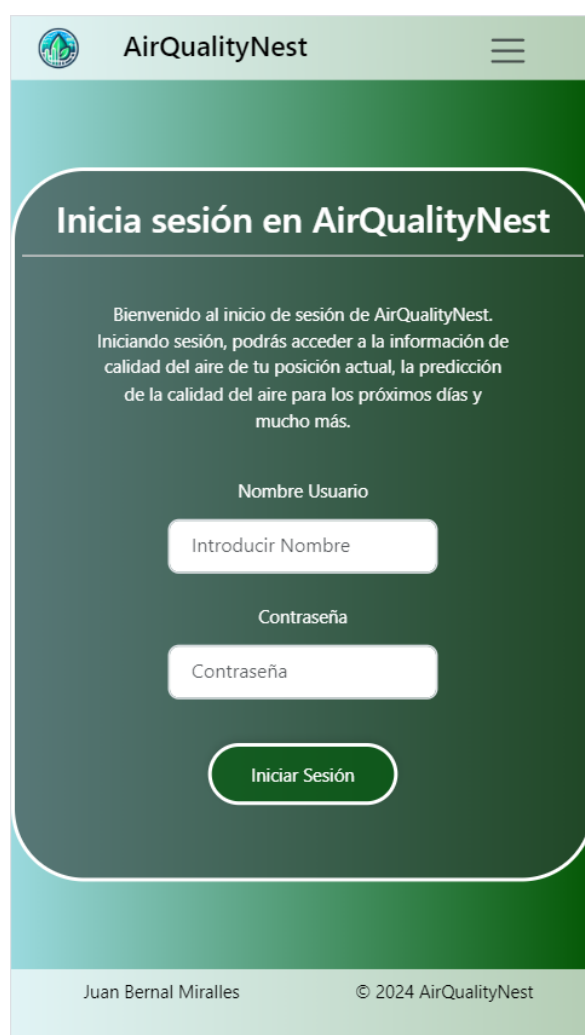
Figura 5.8: Página de la calidad del aire para hoy

Como se puede observar en la figura 5.8, se expone el índice de calidad del aire en un medidor y datos relevantes de la medición. Además, en el título de la página contiene

el nombre de la ciudad más cercana a las coordenadas actuales del usuario.

Inicio de sesión

El inicio de sesión de la aplicación corresponde a esta página. Cualquier error producido por autenticación de rutas y caducidad de sesión son redirigidos a esta página.



AirQualityNest

Inicia sesión en AirQualityNest

Bienvenido al inicio de sesión de AirQualityNest.
Iniciando sesión, podrás acceder a la información de
calidad del aire de tu posición actual, la predicción
de la calidad del aire para los próximos días y
mucho más.

Nombre Usuario

Introducir Nombre

Contraseña

Contraseña

Iniciar Sesión

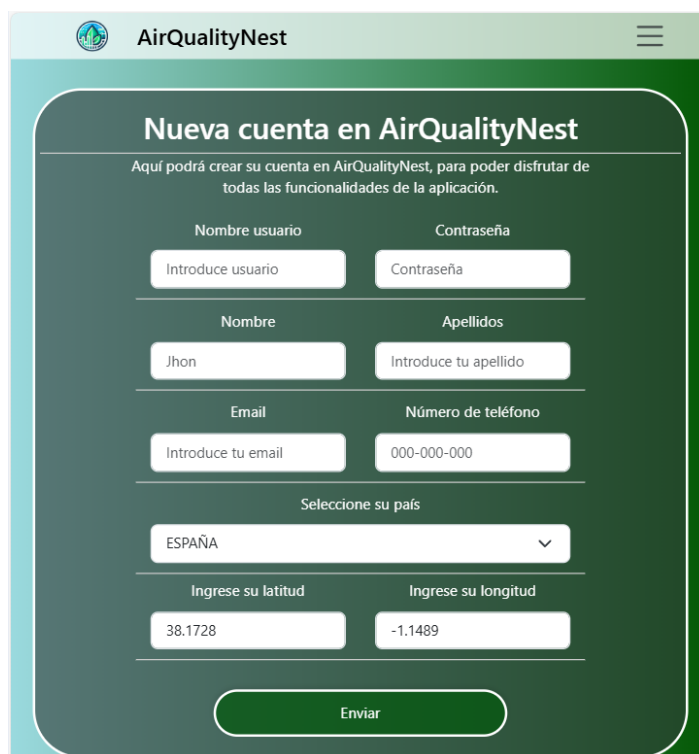
Juan Bernal Miralles © 2024 AirQualityNest

Figura 5.9: Página de inicio de sesión de la aplicación

En la figura 5.9 se puede observar los dos campos disponibles: el nombre de usuario y la contraseña. Esta página permite autenticarse a los usuarios con cuentas en la aplicación, siempre que las credenciales introducidas sean válidas.

Registro usuario participativo

Aquí los usuarios pueden registrarse como usuarios participativos y acceder a su vista. En esta página se solicita toda la información necesaria para el registro, destacando la latitud y longitud que se obtienen de la localización actual del usuario, aunque es modificable.



The image shows a web form titled "Nueva cuenta en AirQualityNest". The form is set against a dark green background with a light green header. The header contains the "AirQualityNest" logo and a hamburger menu icon. Below the title, a subtitle reads: "Aquí podrá crear su cuenta en AirQualityNest, para poder disfrutar de todas las funcionalidades de la aplicación." The form fields are arranged in a grid:

- Nombre usuario**: Input field with placeholder "Introduce usuario".
- Contraseña**: Input field with placeholder "Contraseña".
- Nombre**: Input field with placeholder "Jhon".
- Apellidos**: Input field with placeholder "Introduce tu apellido".
- Email**: Input field with placeholder "Introduce tu email".
- Número de teléfono**: Input field with placeholder "000-000-000".
- Seleccione su país**: Dropdown menu with "ESPAÑA" selected.
- Ingrese su latitud**: Input field with placeholder "38.1728".
- Ingrese su longitud**: Input field with placeholder "-1.1489".

At the bottom of the form is a large green button labeled "Enviar".

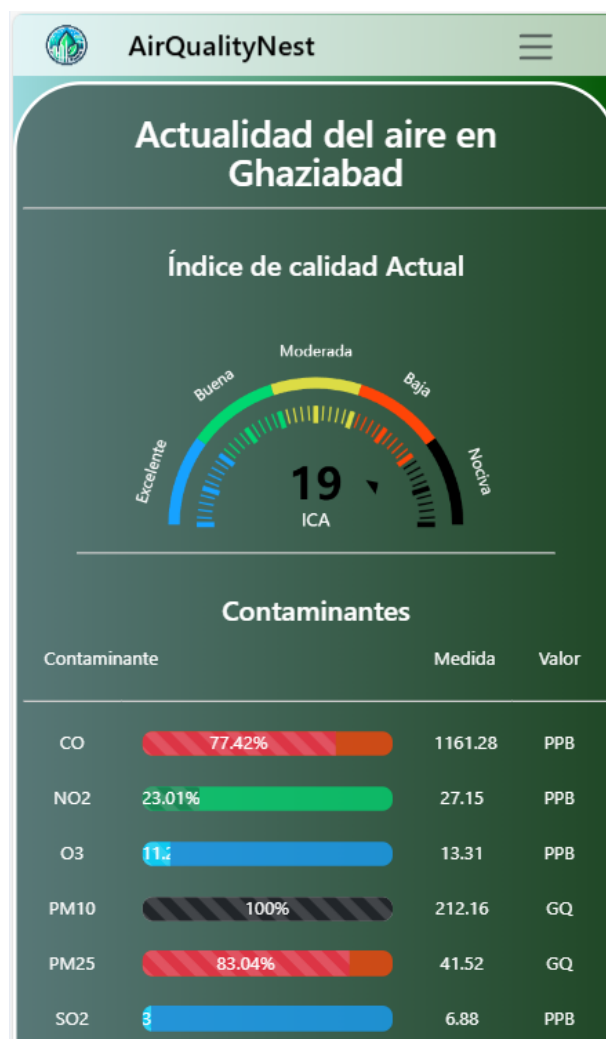
Figura 5.10: Página de registro de usuario participativo

En la figura 5.10 se muestra la página de registro de usuario participativo. Cuando el usuario rellena todos los campos y selecciona el botón correspondiente, podrá registrarse.

Autenticado como usuario

Actualidad

Corresponde a la página principal para el usuario participativo. A diferencia de la página de Actualidad del usuario observador/ no autenticado, en esta se muestra la calidad del aire actual en las coordenadas registradas del usuario participativo.



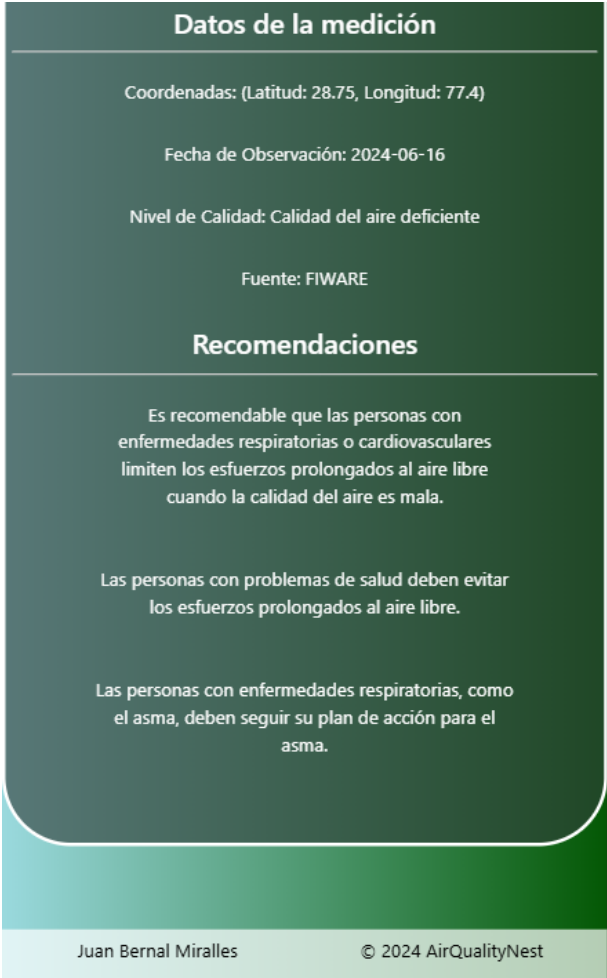
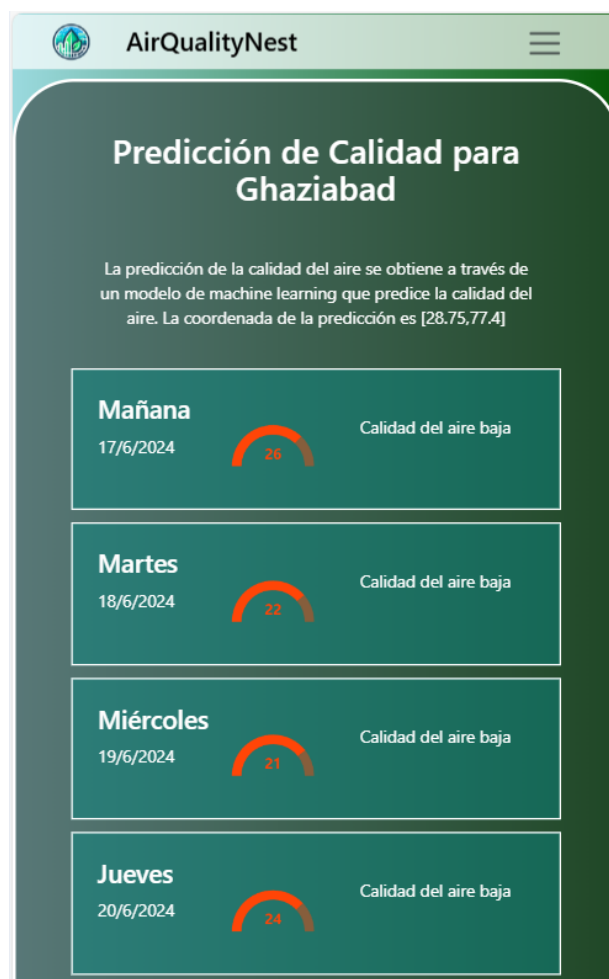


Figura 5.11: Página de actualidad de la aplicación

En la figura 5.11 se aprecia la misma información que se mostraba al usuario observador, pero con la diferencia de que se muestra una lista de contaminantes con su valor, medida y porcentaje de contaminación. Al final de la página se distingue un apartado con las recomendaciones de la calidad del aire.

Predicción

Con esta página el usuario participativo puede ver la predicción de la calidad del aire de los próximos 7 días para su ubicación.



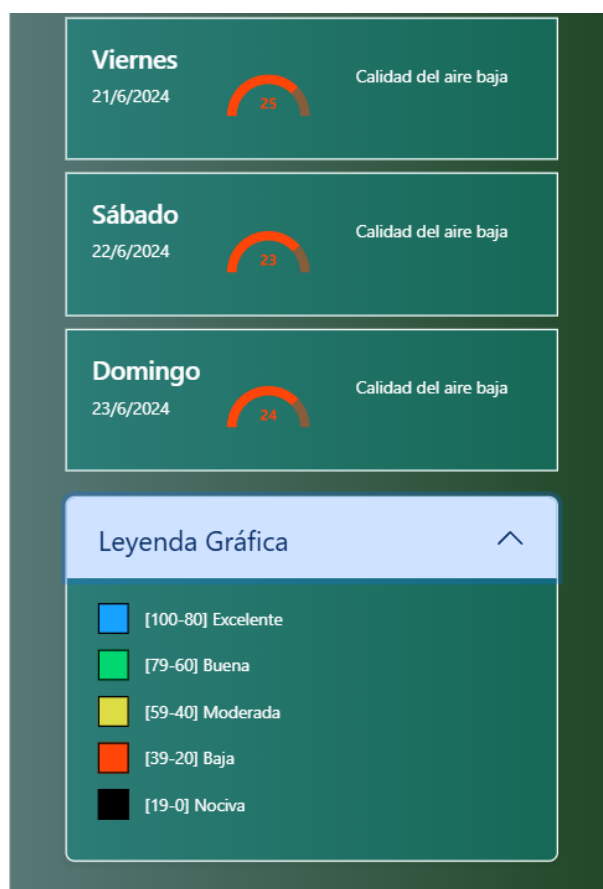


Figura 5.12: Página de predicción de la aplicación

Como se puede observar en la figura 5.12, la página tiene una lista de 7 días con su correspondiente índice de calidad del aire. En cada elemento se muestra el día de la semana, la fecha, el índice de calidad del aire con un medidor más simple que el de la página de actualidad y su representación con texto. Al final de la página se expone una leyenda con los diferentes niveles de calidad del aire y sus colores.

Alertas

Se otorga un registro de las alertas que se han generado para el usuario participativo. Cada alerta contiene la ciudad a la que pertenece, la fecha de la alerta, el índice de calidad del aire, las coordenadas de la alerta y un botón para eliminar la alerta.

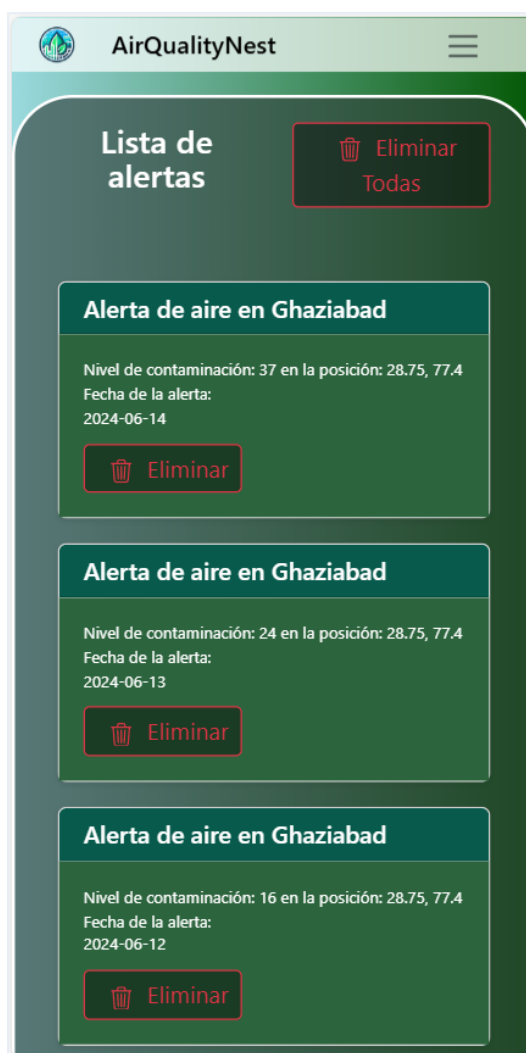
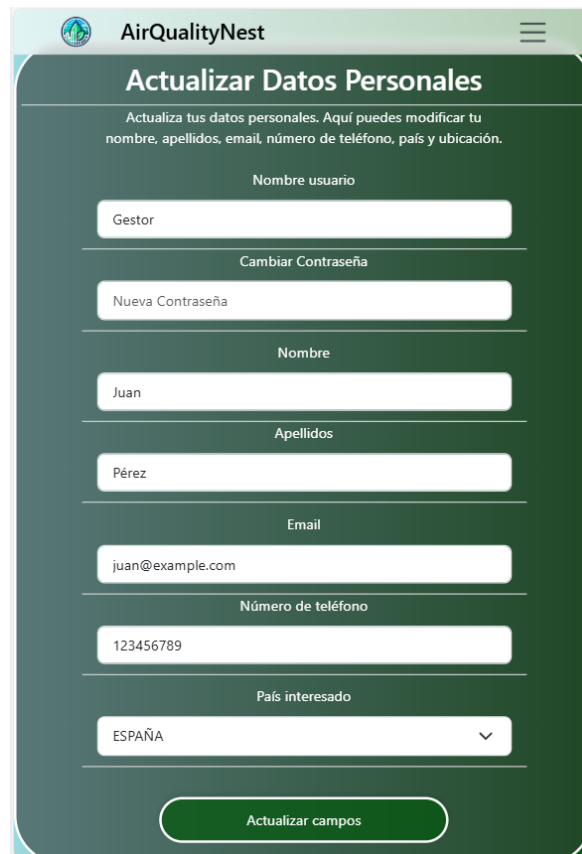


Figura 5.13: Página de alertas de la aplicación

En la figura 5.13 se presenta la página de alertas. Además de la lista de alertas, se muestra un botón para eliminar todas las alertas en la parte superior de la página. Se destaca la información al usuario cuando elimina una o todas las alertas a través de un mensaje. Si el usuario no tiene alertas, se muestra un texto informando de ello y no se dispone de botones para eliminar alertas.

Perfil

El usuario puede ver y modificar su perfil en esta página. Dispone de la facultad de actualizar cualquier dato de manera independiente.



AirQualityNest

Actualizar Datos Personales

Actualiza tus datos personales. Aquí puedes modificar tu nombre, apellidos, email, número de teléfono, país y ubicación.

Nombre usuario

Gestor

Cambiar Contraseña

Nueva Contraseña

Nombre

Juan

Apellidos

Pérez

Email

juan@example.com

Número de teléfono

123456789

País interesado

ESPAÑA

Actualizar campos

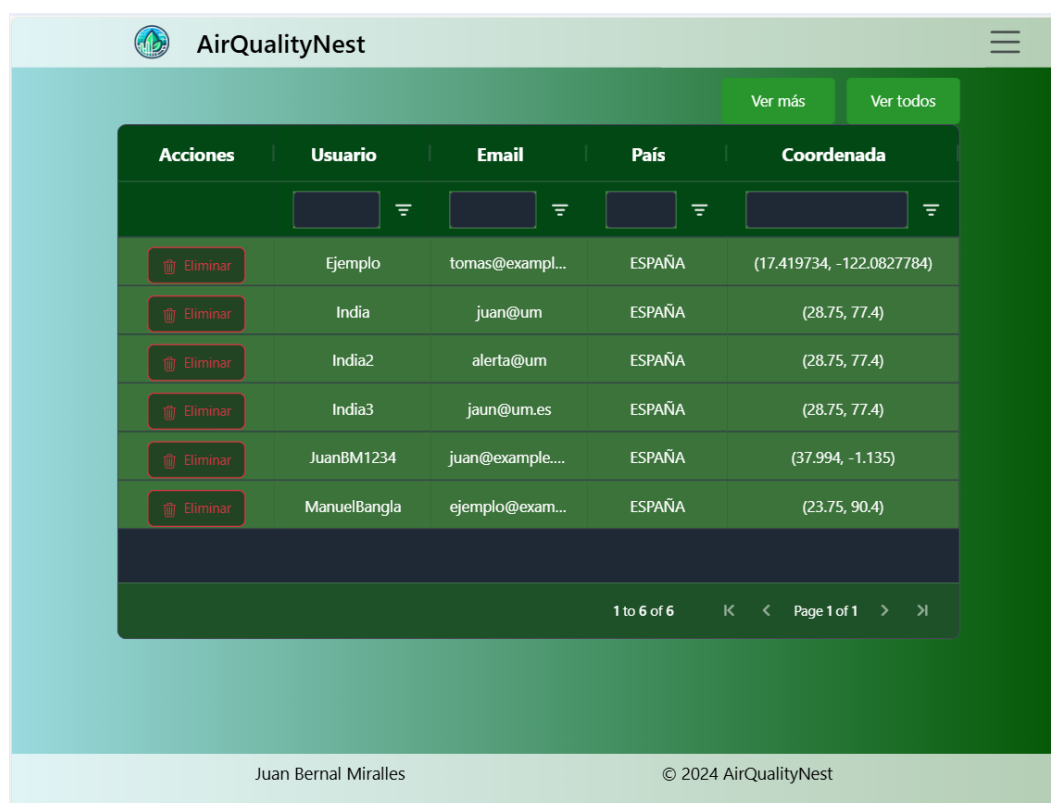
Figura 5.14: Página de perfil de la aplicación

En la figura 5.14 se muestra la página de perfil. Se muestran todos los datos del usuario menos la contraseña. Aunque no se muestre, se puede actualizar rellenando el campo, pero no es necesario para actualizarla.

Autenticado como gestor

Tabla de usuarios

Aquí el usuario gestor puede ver todos los usuarios registrados en la aplicación en formato tabla. En esta se muestra el nombre de usuario, email, país, la latitud, longitud y un botón para eliminar el usuario. La tabla es paginada solicitando solamente 6 usuarios. Se puede obtener mas usuarios gracias a los dos botones de la parte inferior de la tabla que realizan más peticiones al back introduciendo más datos.



Acciones	Usuario	Email	País	Coordenada
 Eliminar	Ejemplo	tomas@exampl...	ESPAÑA	(17.419734, -122.0827784)
 Eliminar	India	juan@um	ESPAÑA	(28.75, 77.4)
 Eliminar	India2	alerta@um	ESPAÑA	(28.75, 77.4)
 Eliminar	India3	jaun@um.es	ESPAÑA	(28.75, 77.4)
 Eliminar	JuanBM1234	juan@example....	ESPAÑA	(37.994, -1.135)
 Eliminar	ManuelBangla	ejemplo@exam...	ESPAÑA	(23.75, 90.4)

1 to 6 of 6 < > Page 1 of 1

Juan Bernal Miralles © 2024 AirQualityNest

Figura 5.15: Tabla de usuarios de la aplicación

En la figura 5.15 se muestra la tabla de usuarios. Destacar que permite el filtrado de datos ofrecidos por la librería Ag-Grid. Además, se informa al usuario cuando se elimina un usuario a través de una ventana emergente de confirmación.

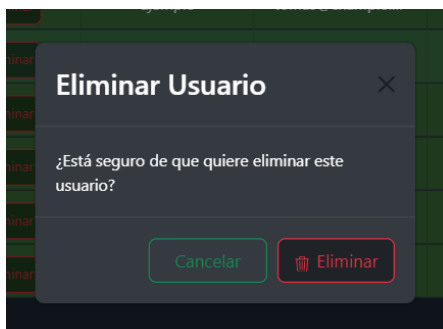


Figura 5.16: Modal de eliminación de usuario

En la figura 5.16 se presenta el modal de eliminación de usuario. En este solicita la confirmación para la eliminación del usuario.

Crear usuario Gestor

Esta pagina es muy similar a la de registro de usuario participativo. La diferencia es que en esta no se solicita la latitud ni longitud.

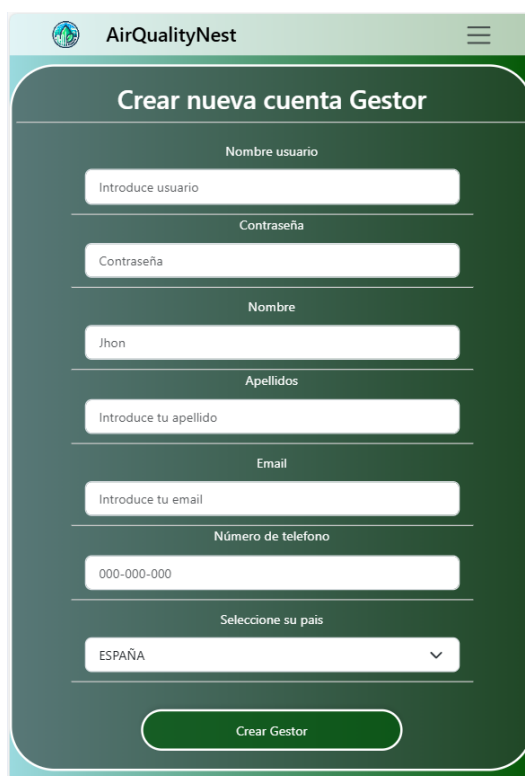
A mobile app screen titled "AirQualityNest" with a hamburger menu icon. The main heading is "Crear nueva cuenta Gestor". The form contains the following fields: "Nombre usuario" (placeholder: "Introduce usuario"), "Contraseña" (placeholder: "Contraseña"), "Nombre" (placeholder: "Jhon"), "Apellidos" (placeholder: "Introduce tu apellido"), "Email" (placeholder: "Introduce tu email"), "Número de telefono" (placeholder: "000-000-000"), and "Seleccione su pais" (dropdown menu showing "ESPAÑA"). At the bottom is a green button labeled "Crear Gestor".

Figura 5.17: Página de creación de un usuario gestor

En la figura 5.17 se aprecia los mismos campos que el registro menos los dos comentados anteriormente.

Perfil

Esta página de perfil es la misma que para el usuario participativo. Al igual que sucede en la creación de usuario gestor la única diferencia con el usuario participativo es que no existe la latitud y longitud.

6. Pruebas y despliegue

En este capítulo se describen las pruebas realizadas al sistema y el despliegue de la aplicación.

6.1. Pruebas

Se han realizado diferentes pruebas de caja blanca. Estas pruebas ejecutan la lógica interna y las interfaces de cada componente software conociendo de antemano su estructura interna [41]. Se han llevado a cabo durante el desarrollo de la aplicación, para comprobar el funcionamiento de cada componente. Se han implementado tres tipos de pruebas de caja blanca: unitarias, de integración y de sistema. Las dos primeras se encuentran en el directorio `tests` del proyecto del backend de la aplicación. Y la ultima en una colección de peticiones de la herramienta **Thunder Client** de Visual Studio Code.

Para el desarrollo de la aplicación se ha creado dos perfiles de Spring, uno para pruebas y otro para producción. Los perfiles se componen de un archivo de configuración general, `application.properties`, y dos archivos de configuración específicos, `application-prod.properties` y `application-test.properties` este último, ubicado en la carpeta `resources` del directorio `test` del proyecto [42].

6.1.1. Pruebas unitarias

Situadas dentro de una carpeta `unit` en el directorio `tests`, las pruebas unitarias se han realizado para comprobar la correcta ejecución de los servicios de la aplicación de forma aislada. Para ello, se han utilizado las librerías `unittest` y `unittest.mock` de `JavaSpring` [43] [44]. Por ejemplo, para el servicio de `airQuality` se han creado 15 pruebas unitarias. En el anexo A.3 se puede encontrar un ejemplo de una de estas pruebas.

6.1.2. Pruebas de integración

Estas pruebas son implementadas para comprobación de los servicios con el resto de componentes de la aplicación. Tenemos dos tipos de pruebas de integración:

- Pruebas de integración con una base de datos en memoria: Utilizando el perfil de test de Spring, se ha configurado la base de datos de H2 para que se ejecute en

memoria. Se han realizado gracias a la librería **SpringBootTest** de **JavaSpring** [43], ubicadas en la carpeta **integration** del directorio **tests**. Ejemplo de ello, se ha procedido a la implementación de 13 pruebas de integración para el servicio y el repositorio de calidad del aire. En los anexos ?? se puede encontrar un ejemplo de una de estas pruebas.

- Pruebas de integración con una base de datos real: Estas pruebas se ubican en la carpeta **program** del directorio **tests**. Corresponden a programas de prueba que se han utilizado en una fase inicial del desarrollo de la aplicación, para comprobación de diferentes funciones.

6.1.3. Pruebas de sistema

Validan la correcta ejecución de la aplicación en su conjunto. Gracias a la librería **Thunder Client** de **Visual Studio Code**, se ha probado todos los **endPoint** de la aplicación a través de llamadas **REST**. Las peticiones son agrupadas por controlador y por tipo de usuario en el caso del controlador **Front**.

6.2. Despliegue

Para el despliegue de la aplicación se ha utilizado la plataforma de contenedores **Docker**. Se ha creado un archivo **docker-compose.yml** A.5 en el directorio raíz del proyecto, que contiene la configuración de los servicios de la aplicación. Este archivo se compone de los siguientes servicios:

- **frontend**: Contiene la imagen de la aplicación **Front** desarrollada en **React**.
- **backend**: Se halla la imagen de la aplicación **Back** desarrollada en **JavaSpring**.
- **mongo**: Abarca la imagen de la base de datos **MongoDB** utilizada por **Fiware**.
- **orion**: Incluye la imagen de **Orion Context Broker**.
- **MySQL**: Contiene la imagen de la base de datos **MySQL** utilizada por el **backend** y la **API** de predicción.
- **Predicción**: Tiene la imagen de la **API** de predicción desarrollada en **Python**.

Además de los servicios, se han configurado las redes de los servicios y los volúmenes de las bases de datos. Para la base de datos **MySQL** se ha creado un iniciador con los datos obtenidos y utilizados durante el desarrollo de la aplicación. Este archivo se encuentra también en el directorio raíz del proyecto y es necesario para el correcto funcionamiento del componente de predicción. Es imprescindible exponer estos tres puertos que son:

- **Front 4800:** Corresponde al puerto donde se conectara el cliente.
- **API de calidad 8080:** Este puerto es obligatorio, porque el proyecto de front-end se ejecuta en el cliente. Hace peticiones a este componente debiendo estar accesible para el usuario.
- **Servicio de Orion Context Broker 1026:** Solo es inexcusable para crear la suscripción de generación de alertas.

Se ha creado un archivo **Dockerfile** en el directorio raíz del proyecto para cada componente desarrollado: Front, API de calidad y API de predicción. Estos archivos contienen las instrucciones necesarias para la creación de las imágenes de los servicios. En el anexo A.6 se puede encontrar un ejemplo de uno de estos archivos.

Para el despliegue de la aplicación únicamente es necesario ejecutar estos dos comandos en el directorio raíz del proyecto:

- **docker-compose up** para la creación de las imágenes y el despliegue de los servicios.
 - **Suscripción a Orion Context Broker** necesaria para la creación de alertas.
-

7. Conclusiones y vías futuras

7.1. Conclusiones

Una vez finalizado el desarrollo de la aplicación, se puede concluir que se han cumplido los principales objetivos propuestos en el proyecto. Se ha desarrollado una aplicación que permite a los usuarios conocer la calidad del aire en tiempo real, consultar la predicción de la calidad del aire para los próximos 7 días y recibir alertas en caso de superar los umbrales establecidos. Todo ello, gracias a la integración y desarrollo de diferentes tecnologías y servicios, como Fiware, la API de predicción y la API de calidad del aire. Sin olvidar el desarrollo de una interfaz de usuario orientada a dispositivos móviles, que permite a los usuarios interactuar con la aplicación de forma sencilla e intuitiva. Hay que destacar la importancia de una tecnología como Fiware, no vista en la carrera, que permite trabajar con datos en tiempo real y facilita la creación de aplicaciones inteligentes. También es importante dar los primeros pasos con dispositivos o sistemas de IoT, que son una parte fundamental del futuro de nuestra sociedad.

Es evidente que es de un alto interés el desarrollo del proyecto, ya que se ha podido aplicar la mayoría de los conocimientos adquiridos durante la carrera, desde la gestión de proyectos, desarrollo de software, bases de datos, técnicas de testing, hasta la creación de un machine learning. Al principio del proyecto, había un reto muy sugerente y a la vez complicado, pero a medida que se avanzaba en el desarrollo, aumentaba la curiosidad de este. Otra parte que ha sido ilustrativa ha sido la utilización por primera vez de tecnologías ofrecidas por Google Cloud, como Air Quality API, que me ha permitido trabajar con datos en tiempo real y trabajar con una API de uso real.

Aunque se ha implementado una versión inicial de la aplicación, no se ha podido desarrollar con toda la funcionalidad deseada. Por ejemplo, no se ha podido implementar la funcionalidad de noticias en la interfaz de usuario, ni una idea bastante interesante como es la simulación de monitores de calidad del aire. Esta funcionalidad se ha quedado fuera del alcance del proyecto, pero es muy atractiva como demuestra el mercado actual de aplicaciones de calidad del aire. Uno de los grandes problemas de la sociedad actual es la contaminación del aire, y cada vez más se están desarrollando aplicaciones que permiten a los usuarios conocer la calidad del aire en tiempo real.

7.2. Vías futuras

Como se ha mencionado anteriormente, la aplicación que se ha desarrollado es una versión inicial. Este proyecto podría escalar a una aplicación más completa y con más funcionalidades. A continuación, se detallan algunas de las posibles vías futuras que se podrían seguir para mejorar la aplicación:

- **Implementación del sistema de monitoreo de calidad del aire:** Se podría implementar una opción para los usuarios participativos con la finalidad de que pudieran obtener/comprar un monitor de calidad del aire. Este monitor puede ser un dispositivo simulado o un dispositivo real. En el caso de ser un dispositivo real, se podría utilizar las tecnologías de Fiware para la integración del dispositivo con la aplicación. Estos dispositivos deberían ser capaces de enviar datos a la aplicación, para que los usuarios pudieran conocer la calidad del aire en tiempo real donde se encuentren estos dispositivos.
 - **Mejora de la aplicación:** Sería el siguiente paso en la evolución de la aplicación: introducir el sistema de noticias, dar la posibilidad de diferentes idiomas y temas, mejorar temas de rendimiento para el ahorro de ancho de banda y modificar el tratamiento de las coordenadas. Actualmente, la aplicación recoge las coordenadas como latitud y longitud en un campo, pero sería interesante que el usuario pudiera seleccionar su ubicación en un mapa. También se podría mostrar la calidad del aire en tiempo real en un mapa, pero para esto es necesario el siguiente punto.
 - **Obtención de datos en tiempo real:** Actualmente, la aplicación obtiene los datos de la calidad del aire de la API de Air Quality que ofrece un límite de peticiones gratuitas. El siguiente paso sería obtener más cantidad de datos. Bien se podría seguir la misma estrategia del proyecto, es decir, obtener los datos una vez al día, pero la opción más óptima sería obtener los datos en un rango de tiempo más corto y en más posiciones geográficas. El problema que abarcaría esta vía es que se necesitaría una API de pago o una gran cantidad de dispositivos propios.
 - **Mejora de la predicción:** Esta mejora perfectamente podría ser un TFG en sí mismo. En la actualidad, este componente es complementario a la aplicación. Uno de los elementos necesario para mejorar la predicción es obtener más datos de calidad del aire. Cuando se hizo operativo el modelo de machine learning, se utilizó un dataset de 55 datos, que no es suficiente para obtener una predicción fiable.
 - **Transformación de la aplicación en una aplicación de microservicios:** A día de hoy, la aplicación tiene diferentes componentes que se comunican entre
-

sí. Sería fundamental transformar la aplicación en una aplicación de microservicios, para que cada componente sea independiente y se pueda escalar de forma independiente.

- **Despliegue en la nube:** A fecha de hoy, la aplicación se despliega en un contenedor Docker en local. Sería loable desplegar la aplicación en la nube cuando se tenga una versión más estable de la aplicación. Para ello, se podría utilizar Google Cloud o Amazon Web Services para el despliegue de la aplicación.
-

Bibliografía

- [1] Swagger. Swagger ui. 2024. URL <https://swagger.lab.fiware.org/?uri=https://raw.githubusercontent.com/Fiware/specifications/master/OpenAPI/ngsiv2/ngsiv2-openapi.json#/>. Accedido el 19 de marzo de 2024.
- [2] Organización Mundial de la Salud. Contaminación del aire ambiente. 2019. URL [https://www.who.int/es/news-room/fact-sheets/detail/ambient-\(outdoor\)-air-quality-and-health#:~:text=Mediante%20la%20disminuci%C3%B3n%20de%20los,agudas%2C%20entre%20ellas%20el%20asma.](https://www.who.int/es/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health#:~:text=Mediante%20la%20disminuci%C3%B3n%20de%20los,agudas%2C%20entre%20ellas%20el%20asma.)
- [3] Envira. Cómo reducir la emisión de gases contaminantes. 2020. URL <https://envira.es/como-reducir-emision-gases-contaminantes/>.
- [4] Gobierno de España. Innovación y tecnología al servicio del ciudadano. ¿qué son las smart cities? 2024. URL <https://www.red.es/es/actualidad/noticias/innovacion-y-tecnologia-al-servicio-del-ciudadano-que-son-las-smart-cities>.
- [5] Puerto de Huelva. Aún no sabes que es fiware? 2022. URL <https://lalonga.tech/que-es-fiware/>.
- [6] Cátedra Telefónica de la UEx. Fiware, el estándar que necesita el internet de las cosas. URL <https://catedratelefonica.unex.es/fiware-el-estandar-que-necesita-el-internet-de-las-cosas/>.
- [7] IQAir. Iqair - monitor de calidad del aire en tiempo real. s/f. URL <https://www.iqair.com/es/>. Accedido el 11 de febrero de 2024.
- [8] Plume Labs. Plume labs - monitor de calidad del aire en tiempo real. 2020. URL <https://air.plumelabs.com/en/>. Accedido el 17 de febrero de 2024.
- [9] Urquijo J. Plume air report, la contaminación del aire en tu móvil. 2015. URL <https://www.greenappsandweb.com/android/plume-air-report-la-contaminacion-del-aire-en-tu-movil/>. Accedido el 17 de febrero de 2024.
- [10] Accuweather. Accuweather - monitor de calidad del aire en tiempo real. 2024. URL <https://www.accuweather.com/es/es/murcia/308556/air-quality-index/308556>. Accedido el 18 de febrero de 2024.

- [11] Agencia Estatal de Meteorología. Composición química de la atmósfera - calidad del aire - no2. 2024. URL https://www.aemet.es/es/eltiempo/prediccion/calidad_del_aire. Accedido el 17 de marzo de 2024.
 - [12] LucusHost. Qué es node.js y para qué sirve. 2022. URL https://www.lucushost.com/blog/que-es-node-js/#Que_es_Nodejs. Accedido el 23 de febrero de 2024.
 - [13] Inc Amazon Web Services. ¿qué es python? - explicación del lenguaje python - aws. 2023. URL <https://aws.amazon.com/es/what-is/python/>. Accedido el 23 de febrero de 2024.
 - [14] Java. ¿qué es java y para qué sirve? 2024. URL https://www.java.com/es/download/help/whatis_java.html. Accedido el 23 de febrero de 2024.
 - [15] Yanina Muradas. Qué es spring framework y por qué usarlo. 2018. URL <https://openwebinars.net/blog/conoce-que-es-spring-framework-y-por-que-usarlo/>. Accedido el 23 de febrero de 2024.
 - [16] Bill Wagner. Un paseo por c: información general. 2023. URL <https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/>. Accedido el 24 de febrero de 2024.
 - [17] Deyimar A. Qué es react: definición, características y funcionamiento. 2020. URL <https://www.hostinger.es/tutoriales/que-es-react>. Accedido el 24 de febrero de 2024.
 - [18] Kinsta®. ¿qué es mysql? una explicación para principiantes. 2022. URL <https://kinsta.com/es/base-de-conocimiento/que-es-mysql/>. Accedido el 24 de febrero de 2024.
 - [19] IBM. ¿qué es mongodb? 2024. URL <https://www.ibm.com/es-es/topics/mongodb>. Accedido el 24 de febrero de 2024.
 - [20] IBM. ¿qué es postgresql? 2021. URL <https://www.ibm.com/es-es/topics/postgresql>. Accedido el 24 de febrero de 2024.
 - [21] Inc Amazon Web Services. Contenedores de docker | ¿qué es docker? | aws. 2023. URL <https://aws.amazon.com/es/docker/>. Accedido el 25 de febrero de 2024.
 - [22] Microsoft. Visual studio: Ide y editor de código para desarrolladores de software y teams. 2020. URL <https://visualstudio.microsoft.com/es/#:~:text=Visual%20Studio%20Code,casi%20cualquier%20lenguaje%20de%20programaci%C3%B3n>. Accedido el 25 de febrero de 2024.
-

-
- [23] Meteosource.com. Api de calidad del aire y contaminación: Datos para cualquier ubicación del mundo. 2024. URL <https://www.meteosource.com/es/api-calidad-aire>. Accedido el 25 de febrero de 2024.
- [24] Google for Developers. Descripción general de la api de air quality. 2024. URL <https://developers.google.com/maps/documentation/air-quality/overview?hl=es-419>. Accedido el 20 de abril de 2024.
- [25] Redacción Maplink. Conoce las nuevas apis de sostenibilidad de google. 2023. URL <https://maplink.global/blog/es/conoce-los-detalles-de-las-nuevas-apis-de-sostenibilidad-de-google/>. Accedido el 25 de febrero de 2024.
- [26] ClickUp. Clickup. 2024. URL <https://app.clickup.com>. Accedido el 2 de marzo de 2024.
- [27] Rodrigues A. Applying natural language processing (nlp) in user stories on agile software development: Uses, techniques and challenges. 2021. URL <https://medium.com/applying-natural-language-processing-nlp-in-user-stories-on-agile-software-dev>. Accedido el 14 de junio de 2024.
- [28] Aprende FIWARE en Español. Introducción. 2014. URL <https://fiware-training.readthedocs.io/es-mx/latest/introduction/>. Accedido el 10 de abril de 2024.
- [29] Fiware-Orion. Fiware-orion. 2024. URL <https://fiware-orion.readthedocs.io/en/master/>. Accedido el 18 de marzo de 2024.
- [30] Orion Context Broker (OCB). Orion context broker (ocb) - aprende fiware en español. 2016. URL <https://fiware-training.readthedocs.io/es-mx/latest/ecosistemaFIWARE/ocb/#api-restful-ngsi-v2>. Accedido el 19 de marzo de 2024.
- [31] Jiménez F. Arquitecturas limpias y ddd, un recorrido práctico. parte 1. 2021. URL <https://sixphere.com/blog/arquitecturas-limpas-ddd-parte-1/>. Accedido el 10 de abril de 2024.
- [32] Marcos Menarguez Tortosa. Tema 2: Servicios rest - introducción. 2024.
- [33] Philipp Hauer. Restful api design. best practices in a nutshell. 2015. URL <https://phauer.com/2015/restful-api-design-best-practices/>. Accedido el 18 de junio de 2024.
- [34] Canva. Canva. 2020. URL <https://www.canva.com/>. Accedido el 14 de junio de 2024.
-

- [35] vitejs. Introducción. 2023. URL <https://es.vitejs.dev/guide/>. Accedido el 14 de junio de 2024.
 - [36] Santander Universidades. Qué es bootstrap | blog santander open academy. 2023. URL <https://www.santanderopenacademy.com/es/blog/que-es-bootstrap.html>. Accedido el 14 de junio de 2024.
 - [37] Javier J. La última versión de react router dom es la versión 6.1. 2023. URL [https://es.linkedin.com/pulse/react-router-dom-v6-reactjs-junior-javier-duque-valera#:~:text=La%20%C3%BAltima%20versi%C3%B3n%20de%20React,una%20sola%20p%C3%A1gina%20\(SPA\)](https://es.linkedin.com/pulse/react-router-dom-v6-reactjs-junior-javier-duque-valera#:~:text=La%20%C3%BAltima%20versi%C3%B3n%20de%20React,una%20sola%20p%C3%A1gina%20(SPA)). Accedido el 14 de junio de 2024.
 - [38] Lenguajecss.com. ¿qué es css? - css en español. 2024. URL <https://lenguajecss.com/css/introduccion/que-es-css/>. Accedido el 24 de junio de 2024.
 - [39] MDN Web Docs. ¿qué es javascript? - aprende desarrollo web. 2023. URL https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript. Accedido el 24 de junio de 2024.
 - [40] Nominatim. Nominatim demo. 2024. URL <https://nominatim.openstreetmap.org/ui/search.html>. Accedido el 16 de junio de 2024.
 - [41] Marcial Pamies Berenguer. Tema 5: Técnicas de pruebas. 2023.
 - [42] Łukasz Ryś. Spring tests - override properties. 2019. URL <https://www.baeldung.com/spring-tests-override-properties>. Accedido el 17 de junio de 2024.
 - [43] baeldung. Spring boot testing. 2017. URL <https://www.baeldung.com/spring-boot-testing>. Accedido el 17 de junio de 2024.
 - [44] freeCodeCamp.org. How to test services, endpoints, and repositories in spring boot. 2019. URL <https://www.freecodecamp.org/news/unit-testing-services-endpoints-and-repositories-in-spring-boot-4b7d9dc2b772/>. Accedido el 17 de junio de 2024.
 - [45] Domain Logic. Clean architecture: qué es, importancia y beneficios para tu empresa - domain logic. 2023. URL <https://domainlogic.io/clean-architecture-que-es-importancia-y-beneficios-para-tu-empresa/>. Accedido el 30 de marzo de 2024.
-

A. Anexo

A.1. Modelo de AirQualityObserved

```
1
2 {
3   "id": "US-AirQuality-API",
4   "type": "AirQualityObserved",
5   "CO": {
6     "type": "Number",
7     "value": 312.12,
8     "metadata": {
9       "unitCode": {
10        "type": "Text",
11        "value": "MILLIGRAMS_PER_CUBIC_METER"
12      }
13    }
14  },
15  "airQualityIndex": {
16    "type": "Number",
17    "value": 71,
18    "metadata": {}
19  },
20  "airQualityLevel": {
21    "type": "Text",
22    "value": "Good air quality",
23    "metadata": {}
24  },
25  "areaServed": {
26    "type": "Text",
27    "value": "United States",
28    "metadata": {}
29  },
30  "dateObserved": {
31    "type": "Text",
32    "value": "2024-03-22T19:00:00Z",
33    "metadata": {}
34  },
35  "location": {
36    "type": "geo:json",
37    "value": {
38      "type": "Point",
39      "coordinates": [
40        -98.5795,
41        39.8283
42      ]
43    },
44    "metadata": {}
45  }
46 }
```

Código A.1: Ejemplo modelo AirQualityObserved del catalogo Environment de FIWARE

A.2. Suscripción a Fiware

```
1  {
2    "description": "Suscripción FIWARE",
3    "subject": {
4      "entities": [
5        {
6          "idPattern": ".*",
7          "type": "AirQualityObserved"
8        }
9      ],
10     "condition": {
11       "attrs": [
12         "airQualityIndex"
13       ],
14       "expression": {
15         "q": "airQualityIndex<40"
16       }
17     }
18   },
19   "notification": {
20     "httpCustom": {
21       "url": "http://host.docker.internal:8080/airqualitynest/api/back/alertas",
22       "type": "withData"
23     }
24   }
25 }
26 }
```

Código A.2: Código de suscripción a Fiware para recibir alertas de calidad del aire

A.3. Pruebas

A.3.1. Pruebas unitarias

```
1  @Test
2  public void testObtenerAirQualityPorLatitudLongitudMasReciente() throws ↵
3      ↵ ServicioAirQualityException {
4      // Arrange
5      Double latitud = 1.0;
6      Double longitud = 2.0;
7      AirQuality airQuality = new AirQuality();
8      when(repositorioAirQuality.↵
9          ↵ findFirstByCoordenadaLatitudAndCoordenadaLongitudOrderByFechaObservacionDesc(↵
10             ↵ latitud,
11             ↵ longitud))
12         .thenReturn(Optional.of(airQuality));
13
14     // Act
15     AirQuality result = servicioAirQuality.obtenerAirQualityPorLatitudLongitudMasReciente(↵
16         ↵ latitud, longitud);
17
18     // Assert
19     assertEquals(airQuality, result);
20     verify(repositorioAirQuality, times(1))
21         .findFirstByCoordenadaLatitudAndCoordenadaLongitudOrderByFechaObservacionDesc(↵
22             ↵ latitud, longitud);
23 }
```

Código A.3: Código de una prueba unitaria de AirQualityService para obtener la calidad del aire más reciente en una coordenada específica

A.3.2. Pruebas de integración

```
1  @Test
2  public void testObtenerAirQualityPorLatitudLongitudMasReciente() throws ↵
3      ↵ ServicioAirQualityException {
4      // Arrange
5      Double latitud = 1.0;
6      Double longitud = 2.0;
7      AirQuality airQuality = new AirQuality();
8      when(repositorioAirQuality.↵
9          ↵ findFirstByCoordenadaLatitudAndCoordenadaLongitudOrderByFechaObservacionDesc(↵
10             ↵ latitud,
11             ↵ longitud))
12         .thenReturn(Optional.of(airQuality));
13
14     // Act
15     AirQuality result = servicioAirQuality.obtenerAirQualityPorLatitudLongitudMasReciente(↵
16         ↵ latitud, longitud);
17
18     // Assert
19     assertEquals(airQuality, result);
20     verify(repositorioAirQuality, times(1))
21         .findFirstByCoordenadaLatitudAndCoordenadaLongitudOrderByFechaObservacionDesc(↵
22             ↵ latitud, longitud);
23 }
```

Código A.4: Código de una prueba de integración de AirQualityRepository para obtener la calidad del aire por id

A.4. Docker

A.4.1. Docker-compose

```

1  version: '3.8'
2  services:
3
4      frontend-airqualitynest:
5          build:
6              context: .
7              dockerfile: dockerfile_front
8          depends_on:
9              - backend-airqualitynest
10         ports:
11             - "4800:80"
12
13     backend-airqualitynest:
14         build:
15             context: .
16             dockerfile: dockerfile_back-airquality
17         ports:
18             - "8080:8080"
19         environment:
20             MYSQL_URI: "jdbc:mysql://docker-mysql-2:3306/tfg-basedatos"
21             MYSQL_USER: "admin"
22             MYSQL_PASSWORD: "password"
23             PREDICTOR_URL: "http://predictor:5000"
24             ORION_URL: "http://orion:1026"
25         depends_on:
26             - docker-mysql-2
27             - predictor
28             - orion
29
30     predictor:
31         build:
32             context: .
33             dockerfile: dockerfile_predictor
34         environment:
35             DATABASE_URI: "mysql+pymysql://admin:password@docker-mysql-2:3306/tfg-basedatos"
36         depends_on:
37             - docker-mysql-2
38
39     mongo:
40         image: mongo:3.2
41         volumes:
42             - mongodata:/data/db
43
44     orion:
45         image: fiware/orion:1.13.0
46         ports:
47             - "1026:1026"
48         command: -logLevel DEBUG -noCache -dbhost mongo
49         depends_on:
50             - mongo
51         healthcheck:
52             test: ["CMD", "curl", "-f", "http://0.0.0.0:1026/version"]
53             interval: 1m
54             timeout: 10s
55             retries: 3
56
57     docker-mysql-2:

```

```
58 image: mysql:latest
59 container_name: docker-mysql-2
60 environment:
61     MYSQL_ROOT_PASSWORD: password
62     MYSQL_DATABASE: tfg-basedatos
63     MYSQL_USER: admin
64     MYSQL_PASSWORD: password
65 volumes:
66     - ./mysql_data:/var/lib/mysql
67     - ./init.sql:/docker-entrypoint-initdb.d/init.sql
68
69 networks:
70     default:
71
72 volumes:
73
74     mongodata:
```

Código A.5: Código del archivo Docker-compose del proyecto para desplegar los servicios de este

A.4.2. Dockerfile

```
1  # Usar una imagen base con JDK 21
2  FROM eclipse-temurin:21-jdk as build
3
4  WORKDIR /app
5  # Instala Maven
6  RUN apt-get update && \
7      apt-get install -y maven
8  # Copia el archivo pom.xml y descarga las dependencias del proyecto
9  COPY ./back-airqualitynest/pom.xml .
10 RUN mvn dependency:go-offline
11
12 # Copia el resto del código fuente
13 COPY ./back-airqualitynest/src /app/src
14
15 # Compilar el proyecto y empaquetarlo en un archivo JAR
16 RUN mvn clean package -DskipTests
17
18 # Utilizar una imagen base más ligera para el tiempo de ejecución
19 FROM eclipse-temurin:21-jre
20 WORKDIR /app
21
22 # Copia el archivo JAR compilado de la etapa de construcción
23 COPY --from=build /app/target/*.jar app.jar
24
25 # Define el comando de inicio de la aplicación
26 ENTRYPOINT ["java", "-jar", "app.jar"]
```

Código A.6: Código del archivo Dockerfile para la API de calidad del proyecto