

2020

ANEXO 1: Manual de administración



José Granados Rodríguez
IES PUNTA DEL VERDE

Índice

Instalación de herramientas necesarias.....	2
Código relevante.....	11
Claves de plataformas.....	22
Tecnologías utilizadas	22

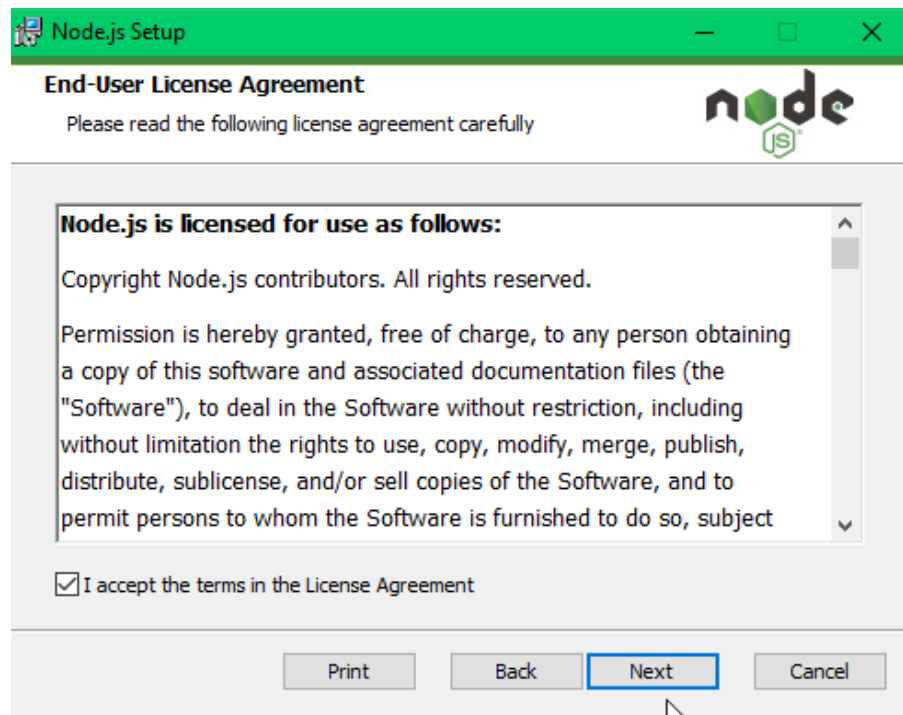
Instalación de herramientas necesarias.

Para hacer la instalación de este proyecto necesitamos tener instalado node.js, un editor de código (como visual code) y una terminal.

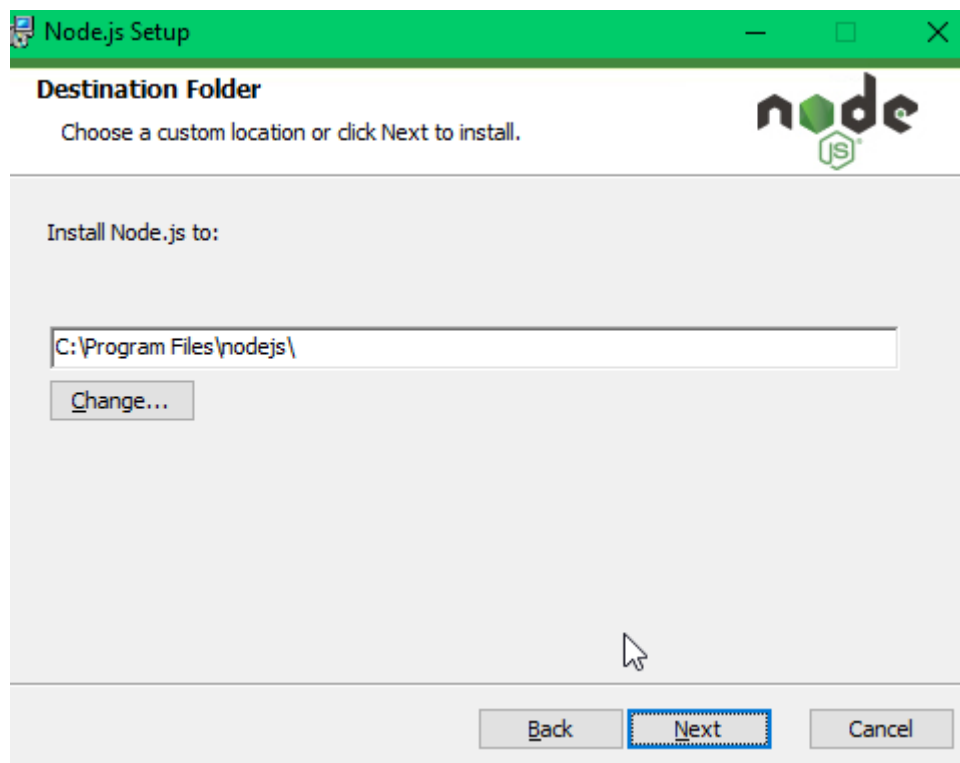
Empezaremos instalando node.js



Hacemos clic donde nos indica la imagen y se iniciará la descarga. La razón por la que elegimos la versión LTS es porque actualmente es la más estable y tiene más soporte técnico.



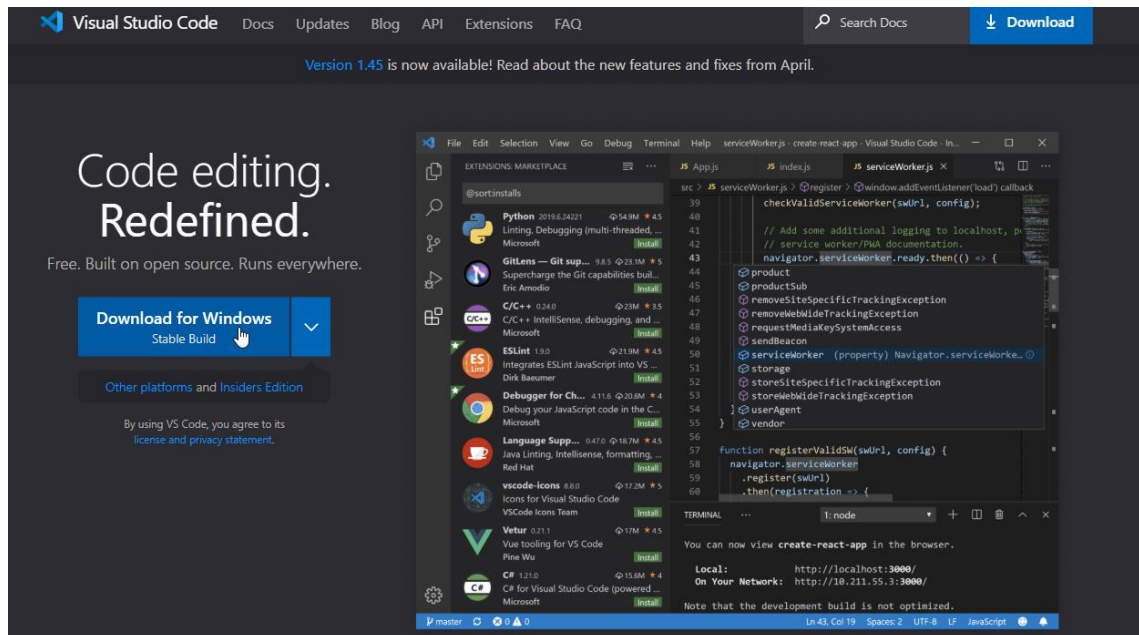
Aquí tenemos la interfaz de instalación de node.js en la que tendremos que aceptar los términos de la licencia y hacer clic en siguiente.



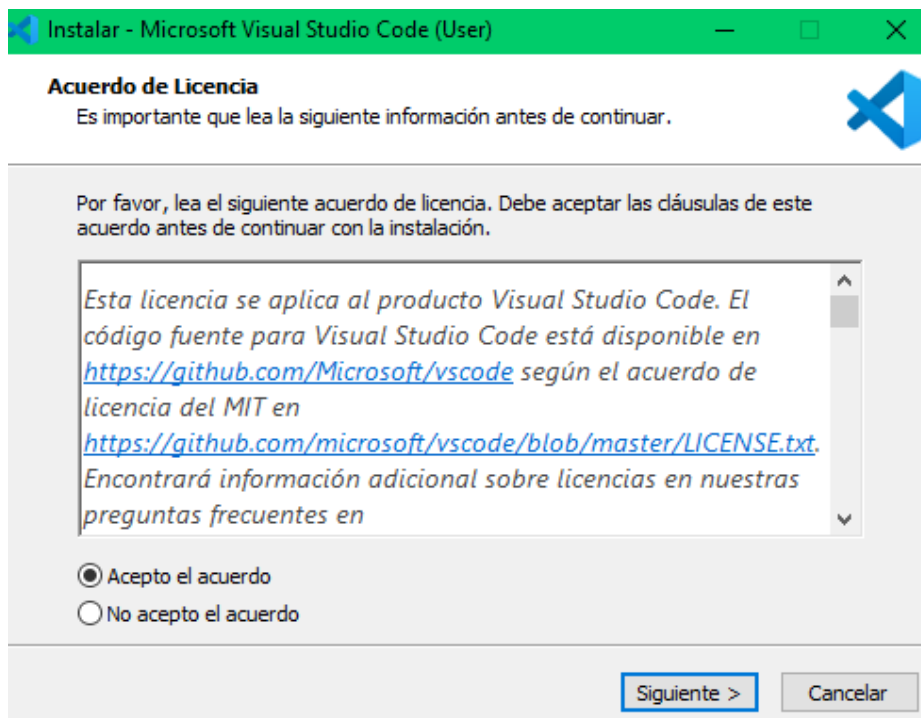
Elegimos la ubicación de donde vamos a instalar el programa y haciendo clic en siguiente ya se iniciará la instalación.

A continuación, vamos a instalar un editor de código como puede ser visual code.

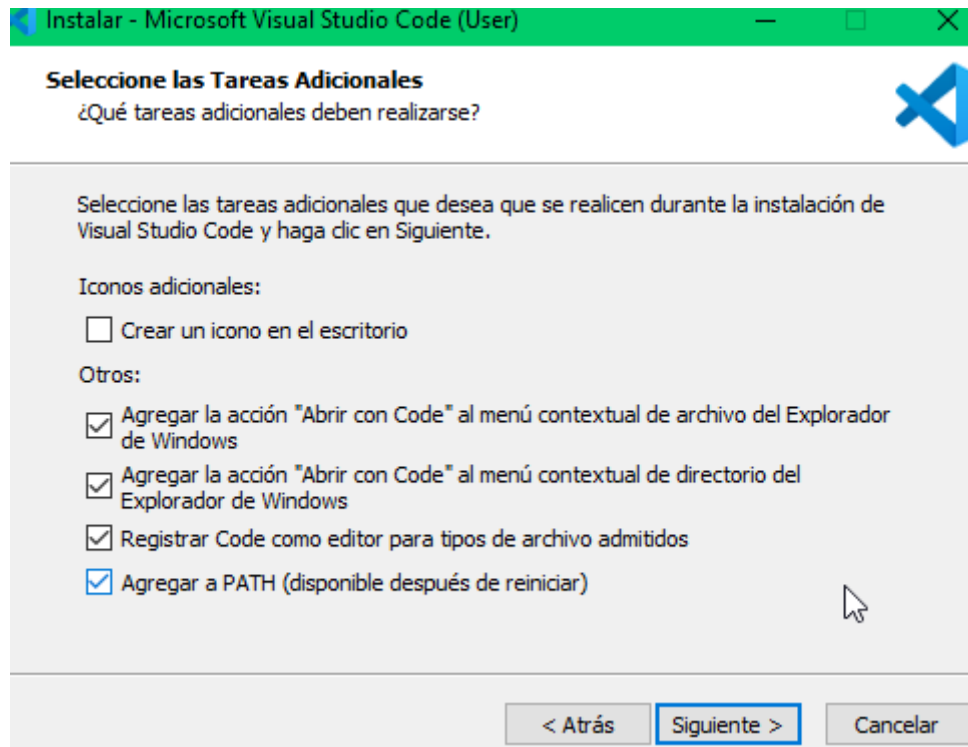
Nos iremos a la página oficial y descargaremos el software.



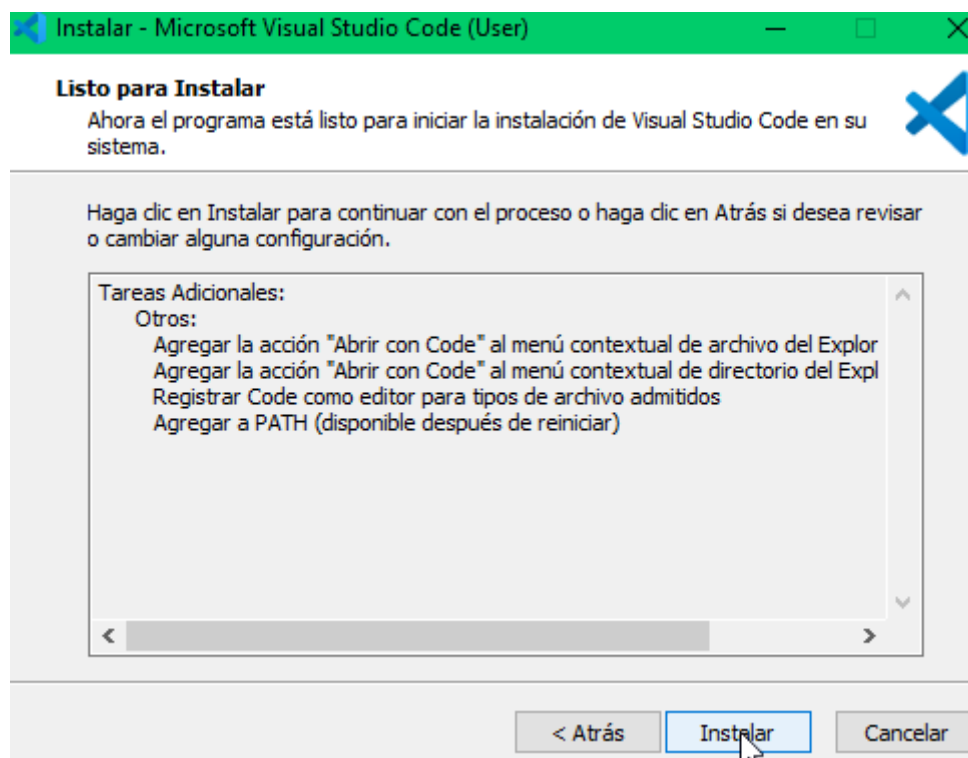
Ahora abriremos el archivo que se nos ha descargado e iniciaremos la instalación



Aceptamos los términos de la licencia

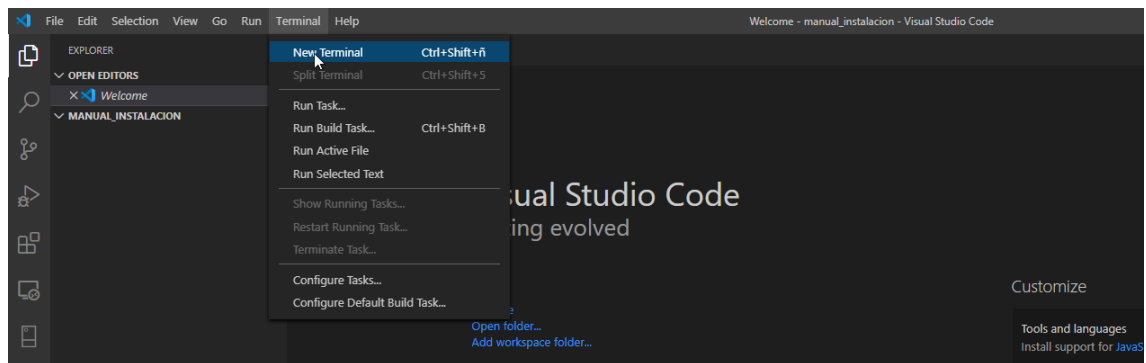


En esta ventana elegiremos según nuestras necesidades, personalmente dejaría las que están marcadas pero sobre todo la última ya que permite que utilicemos las terminales dentro de visual code.

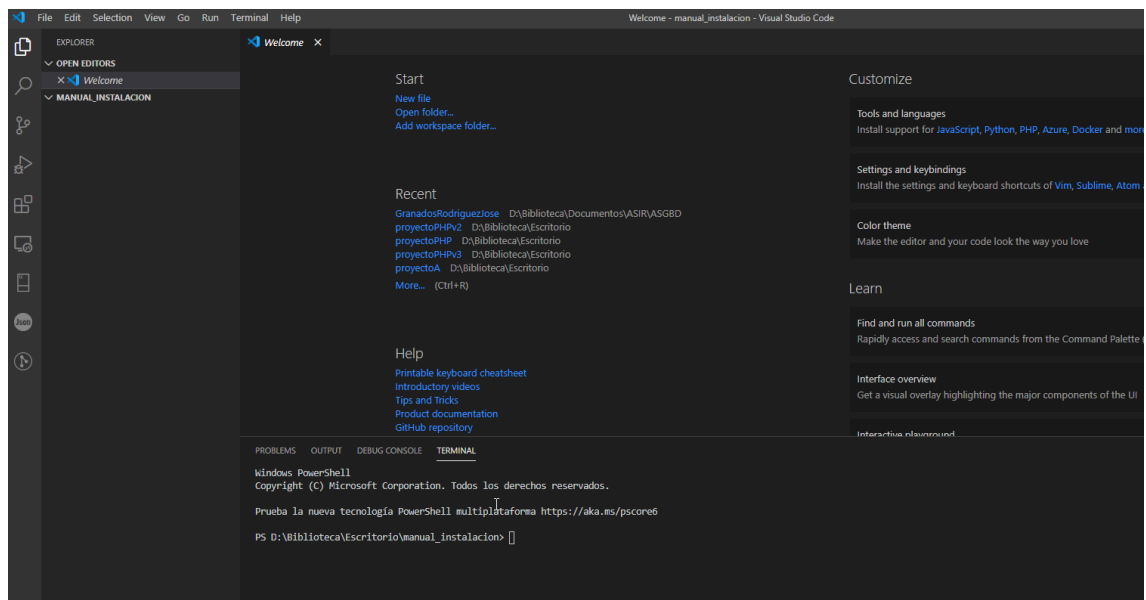


Una vez hecho todo lo anterior nos aparecerá esta ventana que nos dirá que hemos elegido y si queremos instalarlo.

La terminal la abriremos dentro de visual code.



Dentro de la carpeta que queremos abrir la terminal entraremos en visual code y para abrir la terminal nos iremos a terminal/new Terminal como se muestra en la imagen anterior.



Aquí en la parte inferior de la imagen vemos como tenemos una terminal abierta, que en el prompt nos indica donde estamos situados.

A continuación, vamos a pasar con la instalación del proyecto, es decir, instalar los módulos necesarios de node.js para el correcto funcionamiento del proyecto.

```
PS D:\Biblioteca\Escritorio>manual_instalacion> npm install @angular/cli
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142

> @angular/cli@9.1.7 postinstall D:\Biblioteca\Escritorio>manual_instalacion\node_modules\@angular\cli
> node ./bin/postinstall/script.js

npm WARN saveError ENOENT: no such file or directory, open 'D:\Biblioteca\Escritorio>manual_instalacion\package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open 'D:\Biblioteca\Escritorio>manual_instalacion\package.json'
npm WARN manual_instalacion No description
npm WARN manual_instalacion No repository field.
npm WARN manual_instalacion No README data
npm WARN manual_instalacion No license field.
```

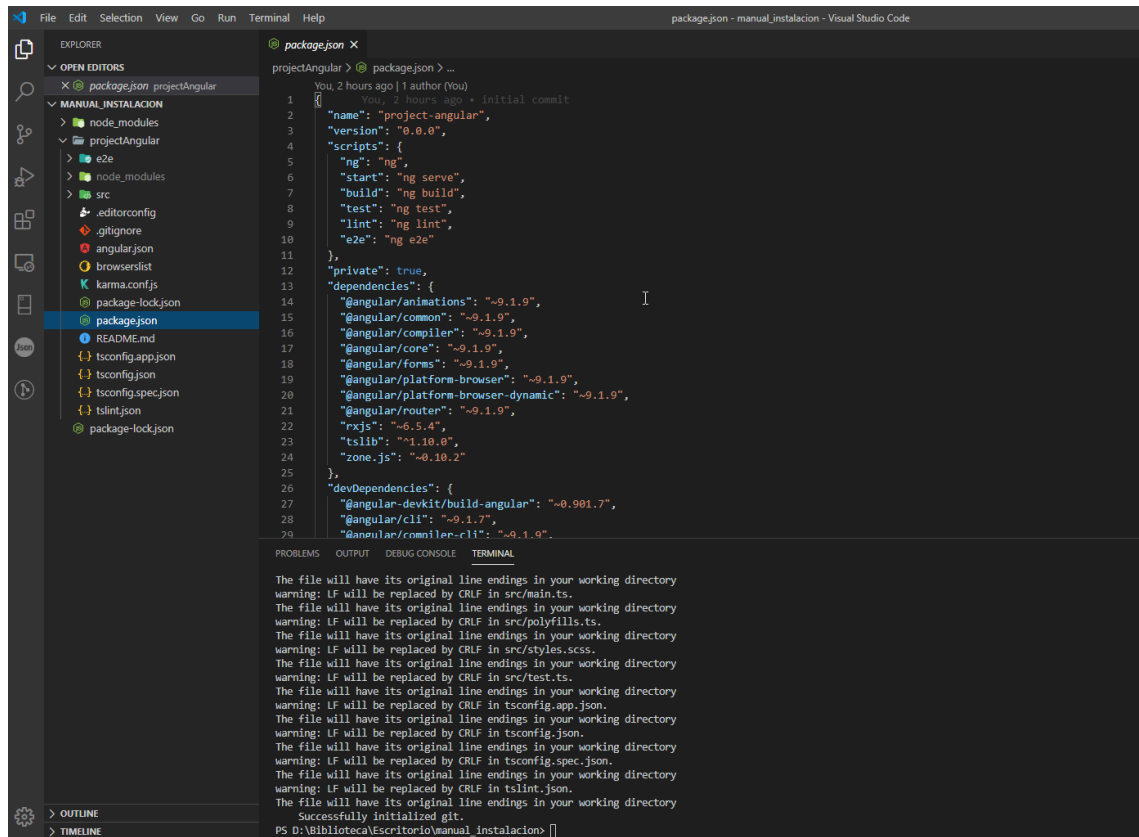
Ejecutamos el comando que se muestra en la imagen para hacer la instalación de angular/cli que nos instalará todo lo necesario para que funcione angular.

Ahora tenemos que crear un proyecto, por tanto, tenemos que ejecutar los siguientes comandos en la terminal.

```
PS D:\Biblioteca\Escritorio>manual_instalacion> ng new projectAngular
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? SCSS [ https://sass-lang.com/documentation/syntax#scss
CREATE projectAngular/angular.json (3718 bytes)
CREATE projectAngular/package.json (1248 bytes)
CREATE projectAngular/README.md (1031 bytes)
CREATE projectAngular/tsconfig.json (489 bytes)
CREATE projectAngular/tslint.json (3125 bytes)
CREATE projectAngular/.editorconfig (274 bytes)
CREATE projectAngular/.gitignore (631 bytes)
CREATE projectAngular/browserslist (429 bytes)
CREATE projectAngular/karma.conf.js (1026 bytes)
CREATE projectAngular/tsconfig.app.json (210 bytes)
CREATE projectAngular/tsconfig.spec.json (270 bytes)
```

Una vez ejecutado el comando anterior, la instalación puede demorarse bastante.

Una vez finalice el comando que lanzamos anteriormente, nos debería de haber creado una carpeta con el nombre que le hayamos dado en la creación del proyecto con todos los archivos que aparecen en la parte izquierda de la imagen.



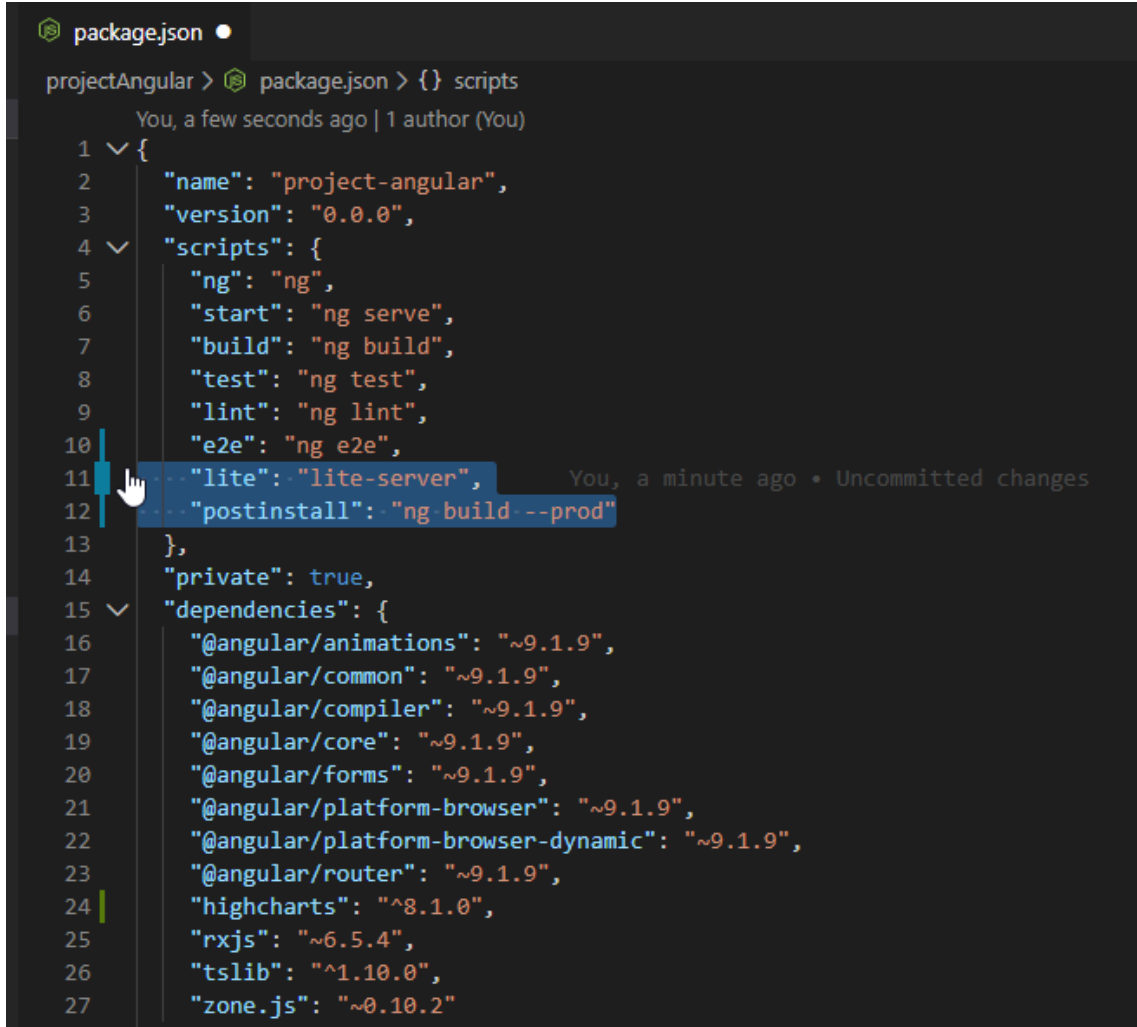
Para instalar los módulos que queramos añadir a angular tendremos que cambiarnos al directorio que acabamos de crear, por tanto, tendremos que lanzar el comando de la siguiente imagen:

```
PS D:\Biblioteca\Escritorio>manual_instalacion\projectAngular> npm install highcharts
[.....] / rollbackFailedOptional: verb npm-session 38f3ac98aa364a1e
```

Una vez hayamos instalado esto ya tendríamos todo lo necesario para que funcione el proyecto o para poder desarrollarlo

Ahora vamos a preparar el proyecto para poder subirlo a Heroku o a cualquier servidor que sea capaz de correr en él node.js

Para ello tendremos que añadir dos scripts al archivo de package.json que esta situado dentro del proyecto de angular.



```
package.json
projectAngular > package.json > {} scripts
You, a few seconds ago | 1 author (You)
1 {
2   "name": "project-angular",
3   "version": "0.0.0",
4   "scripts": {
5     "ng": "ng",
6     "start": "ng serve",
7     "build": "ng build",
8     "test": "ng test",
9     "lint": "ng lint",
10    "e2e": "ng e2e",
11    "lite": "lite-server",
12    "postinstall": "ng build --prod"
13  },
14  "private": true,
15  "dependencies": {
16    "@angular/animations": "~9.1.9",
17    "@angular/common": "~9.1.9",
18    "@angular/compiler": "~9.1.9",
19    "@angular/core": "~9.1.9",
20    "@angular/forms": "~9.1.9",
21    "@angular/platform-browser": "~9.1.9",
22    "@angular/platform-browser-dynamic": "~9.1.9",
23    "@angular/router": "~9.1.9",
24    "highcharts": "^8.1.0",
25    "rxjs": "~6.5.4",
26    "tslib": "^1.10.0",
27    "zone.js": "~0.10.2"
```

Los dos scripts que hemos añadido son los que se muestran en la imagen.

El primer script su función es hacer menos pesado el servidor.

El segundo script su función es construir la versión de producción.

Crearemos un fichero dentro de el proyecto de angular, que se llamará server.js y su contenido será el siguiente:



```
server.js
frontaquiv1 > .\server.js > ...
Unsaved changes (cannot determine recent change or authors)
1  const express = require('express');
2  const path = require('path');
3
4  const app = express();
5
6  app.use(express.static(__dirname+'dist/frontaquiv1'));
7  app.get('/',function(req,res){
8      res.sendFile(path.join(__dirname+'dist/frontaquiv1/index.html'));
9  });
10
11  app.listen(process.env.PORT || 8080);
```

Lo importante de este archivo es indicarle donde se encuentra el index.html para redirigirlo y mostrar la página web

Código relevante.

En esta parte vamos ver la parte más importante del código, es decir, los métodos utilizados que permiten que la aplicación funcione correctamente.

Para ello empezaremos explicando que angular 9 funciona mediante pequeños módulos que se van cargando a medida que los necesitamos. Tendremos que hacer uso de componentes y servicios para cargar partes de la página, o una página entera, ya sean para código HTML o TypeScript. Para entender que es lo que hacemos necesitaremos saber que es un componente y un servicio:

- Un **componente** en Angular es una combinación de un archivo HTML con un TypeScript y algunas veces scss para crear un elemento con características propias tanto de comportamiento como de apariencia que se puede mostrar en un navegador.
- Un **Servicio** en Angular es el mecanismo para compartir funcionalidad entre componentes.

Dado por sabido esto, pasaremos a la parte más relevante del código de la página web.

Empezaremos por la página de inicio que ofrecerá los datos del día de hoy, ya sea, de los datos ofrecidos por la Unión Europea o por los datos propios que obtenemos de nuestros sensores. Para ello empezaremos por el código de los datos que nos ofrece la Unión Europea.

```
import { Component, OnInit } from '@angular/core';
import * as Highcharts from 'highcharts';
import { GraficaHistoricosService } from '../../../Services/pagina-inicio/grafica-historicos/grafica-historicos.service';
```

Empezaremos importando los componentes necesarios, como son HighCharts y el servicio encargado de ofrecernos los datos llamado "GraficaHistoricosService".

En la siguiente imagen creamos las opciones de la gráfica para cargarla posteriormente, con los datos que obtenemos mediante el servicio incluido anteriormente.

```
public options : any = {
  chart: {
    type: 'column'
  },
  title: {
    text: 'Datos Externos'
  },
  xAxis: {
    categories: [] //nombre de los contaminantes
  },
  yAxis: {
    title: {
      text: 'µg/m3'
    }
  },
  //Define el contenido de la leyenda
  tooltip: {
    formatter: function() {
      switch (this.point.category) {
        case "PM10":
          return "<small>PM10 ~ < 35 µg/m3 = BUENO <br> <- 50 µg/m3 = ACEPTABLE <br> > 50 µg/m3 = MALO</small><br>" + this.series.color + "<b>";
          break;
        case "PM25":
          return "<small>PM25 ~ < 20 µg/m3 = BUENO <br> <- 25 µg/m3 = ACEPTABLE <br> > 25 µg/m3 = MALO</small><br>" + this.series.color + "<b>";
          break;
        case "NO2":
          return "<small>NO2 ~ < 30 µg/m3 = BUENO <br> <- 40 µg/m3 = ACEPTABLE <br> > 40 µg/m3 = MALO</small><br>" + this.series.color + "<b>";
          break;
        case "CO":
          return "<small>CO ~ < 10 µg/m3 = BUENO <br> > 10 µg/m3 = MALO</small><br>" + this.series.color + "<b>";
          break;
        case "SO2":
          return "<small>SO2 ~ < 15 µg/m3 = BUENO <br> <- 20 µg/m3 = ACEPTABLE <br> > 20 µg/m3 = MALO</small><br>" + this.series.color + "<b>";
          break;
        case "O3":
          return "<small>O3 ~ < 75 µg/m3 = BUENO <br> <-100 µg/m3 = ACEPTABLE <br> > 100 µg/m3 = MALO</small><br>" + this.series.color + "<b>";
          break;
      }
    }
  },
  credits: {
    enabled: false
  },
  series: [
    //nombre(name) del país y valores(data) de los contaminantes
  ]
}
```

```
1 import { Injectable } from '@angular/core';
2 import { HttpClient, HttpHeaders } from '@angular/common/http';
3
4 You, a month ago | 1 author (You)
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class GraficaHistoricosService {
9
10   constructor(private _http: HttpClient) { }
11
12   ObtenerDatos(pais){
13     let apiSpain = "https://api.waqi.info/feed/@8495/?token=2925a12d6716caa9e5eff975b281dd6eb985552c"
14     let apiBulgarian = "https://api.waqi.info/feed/@8084/?token=2925a12d6716caa9e5eff975b281dd6eb985552c"
15     let apiGreece = "https://api.waqi.info/feed/@12410/?token=2925a12d6716caa9e5eff975b281dd6eb985552c"
16     switch (pais) {
17       case "spain":
18         return this._http.get(apiSpain)
19         break;
20       case "bulgarian":
21         return this._http.get(apiBulgarian)
22         break;
23       case "greece":
24         return this._http.get(apiGreece)
25         break;
26       default:
27         break;
28     }
29   }
30 }
```

En esta imagen visualizamos el servicio que hemos incluido en el componente anterior, por tanto, su funcionamiento es que recibe un parámetro del componente en el que realizará una opción u otra según el país que haya introducido mediante parámetros y realizará una petición a una API REST u otra dentro de la misma API, es decir, lo único que cambia de la URL que aparece en la imagen solo es la estación a la que pertenece (de ahí la necesidad de utilizar un switch)

En la imagen vemos como inicializamos un array con los tres países que vamos a obtener sus datos. Para realizar la petición de los tres países que queremos, tendremos que recorrer el array de países, según tantos países hayamos incluido. Cuando obtengamos los datos dependiendo de cada país los iremos almacenando en `datosSpain`, `datosBulgarian` y `datosGreece` según la estructura de datos que se muestra en la imagen, ya que es la estructura que tiene que tener cuando se lo pasemos a las opciones de la gráfica posteriormente.

```
let paises = ["spain","bulgarian","greece"]
let i = 0
let datosSpain = []
let datosBulgarian = []
let datosGreece = []

let nameComun = []

paises.forEach(k => {

  this._historicos.ObtenerDatos(k).subscribe( data => {

    for (const key in data["data"]["iaqi"]) {
      if(k == 'spain'){
        datosSpain.push(
          {
            name: key,
            data: data["data"]["iaqi"][key]["v"]
          }
        )
      } else if (k == 'bulgarian'){
        datosBulgarian.push(
          {
            name: key,
            data: data["data"]["iaqi"][key]["v"]
          }
        )
      } else {
        datosGreece.push(
          {
            name: key,
            data: data["data"]["iaqi"][key]["v"]
          }
        )
      }
    }
  })
})

i++
```

Cuando el acumulador “i” tenga el tamaño del array países establecemos la estructura de datos mostrada en la imagen, y será almacenada en el array “datos”.

En introducción de los primeros datos, introduciremos los datos que son comunes en los tres países ya que las diferentes estaciones de cada país pueden ofrecer diferentes contaminantes.

```
if (l == pais.length) {  
    let categorias = []  
    let datos = [  
        {  
            name: "Spain",  
            data: []  
        },  
        {  
            name: "Bulgarian",  
            data: []  
        },  
        {  
            name: "Greece",  
            data: []  
        }  
    ]  
    You, a month ago • la buena  
  
    //Introducción de primeros datos  
    datosSpain.forEach(eSpain => {  
        datosBulgarian.forEach(eBulgarian -> {  
            datosGreece.forEach(eGreece -> {  
                if( ""+eSpain["name"] == ""+eBulgarian["name"] && ""+eSpain["name"] == ""+eGreece["name"]){  
                    if( ""+eSpain["name"] != "p" && ""+eSpain["name"] != "h" && ""+eSpain["name"] != "dew" && ""+eSpain["name"] != "w" && ""+eSpain["name"] != "wg" && ""+eSpain["name"] != "v" && ""+eSpain["name"] != "vg") {  
                        datos[0]["data"].push(eSpain["data"])  
                        datos[1]["data"].push(eBulgarian["data"])  
                        datos[2]["data"].push(eGreece["data"])  
                        categorias.push(eSpain["name"].toUpperCase())  
                        nameComun.push(eSpain["name"])  
                    }  
                }  
            });  
        });  
    });  
});  
  
Algoritmo ESPAÑA  
let only = []  
  
datosSpain.forEach(eSpain -> {  
    let encontrado = false  
    nameComun.forEach(key => {  
        if( ""+eSpain["name"] == ""+key){  
            encontrado = true  
        }  
    });  
  
    if(!encontrado){
```

```
// Debido a que cada país puede mostrar distintos contaminantes, en la introducción de datos anterior solo se han introducido los contaminantes comunes
// Para ello realizaremos diferentes algoritmos para cada país      You, a few seconds ago • Uncommitted changes
let only = []

// Algoritmo ESPAÑA
datosSpain.forEach(eSpain => {
  let encontrado = false
  nameComun.forEach(key => {
    if(`${eSpain["name"]} == "${key}") {
      encontrado = true
    }
  })
  if(!encontrado){
    only.push(
      {
        name: eSpain["name"],
        data: [eSpain["data"], 0, 0]
      }
    )
  }
});
```

En el enunciado que aparece en la imagen se explica el porqué de los algoritmos. El algoritmo de España funciona de la siguiente manera:

Hacemos un `foreach` para recorrer `datosSpain`, establecemos el flag `encontrado` y lo inicializamos a `false`. Recorremos la variable `nameComun` para ver si esta incluido, en caso de que este incluido encontrado pasa a `true` y en la siguiente condición no entrará. En caso de encontrado se `false`, se añadirá al array `only` según la estructura establecida en la imagen.

```
// Algoritmo BULGARIA

datosBulgarian.forEach(eBulgarian => {
  let encontrado = false
  nameComun.forEach(key => {
    if(""+eBulgarian["name"] == ""+key){
      encontrado = true
    }
  });
  if(!encontrado){
    let encontradoB = false
    let indice1E = 0
    let indice1 = 0
    only.forEach(element2 => {
      if(""+eBulgarian["name"] == ""+element2["name"]){
        encontradoB = true
        indice1E = indice1
      }
      indice1++
    });

    if(encontradoB){
      only[indice1E]["data"][1] = eBulgarian["data"]
    } else {
      only.push(
        {
          name: eBulgarian["name"],
          data: [0, eBulgarian["data"], 0]
        }
      )
    }
  }
});
```

En este algoritmo realizaremos la misma operación que en el anterior pero con una comprobación adicional, ya que tenemos que encontrar que si el contaminante se encuentra añadido en only tenemos que añadir el valor del contaminante y en caso contrario tendremos que añadirlo a la el array push.

El algoritmos de Grecia, es igual al de Bulgaria pero cambiando las variables de Bulgaria por Grecia


```
only.forEach(element => {
  if(!"+element["name"] != "p" && "+element["name"] != "h" && "+element["name"] != "dew" && "+element["name"] != "w" && "+element["name"] != "wg" && "+element["name"]
```

Aquí realizamos la introducción de los datos, pero realizamos una comprobación de que no sean los contaminantes que aparecen en la imagen para que no aparezcan en la gráfica, ya que son contaminantes que se miden con otra medida, en caso de que no sea ninguno de los indicados en la comprobación se añadirán los datos.

```
//Asignación de los datos a la gráfica
this.options.xAxis["categories"] = categories
this.options.series = datos
Highcharts.chart('container', this.options)
```

Las dos primeras líneas son los datos que los añadimos a las opciones de la gráfica, y en la última línea es cuando hacemos la llamada a la gráfica y entonces es cuando se pintaría el componente.

La representación de los datos propios se realiza un procedimiento similar, cambiando el algoritmo de la organización de los datos, pero la filosofía es la misma, primero se inician las opciones de la gráfica, hacemos una petición de los datos, los organizamos, se lo indicamos a las opciones de la gráfica y hacemos llamada a la gráfica. Lo único que tendríamos diferente respecto a los datos externos es que utiliza un servicio diferente y la API REST a la que apunta es la nuestra en vez la api de los datos externos, pero la petición sigue siendo get.

A continuación, pasaremos a la página de los datos propios, es aplicable también a los datos externos. Los métodos que se utilizarán serán los siguientes:

- Importación de módulos y servicios.
- Introducción de opciones de la gráfica.
- Creación del esquema del formulario
- Creación de una función para ejecutar al enviar el formulario.
- Petición POST mediante un servicio creado
- Recepción y organización de datos
- Indicar datos a las opciones y ejecutar la gráfica.

```
import { Component, OnInit } from '@angular/core';
import { PropiosServiceService } from '../../Services/pagina-propios/propios-service.service';
import { FormControl, FormGroup } from '@angular/forms';
import * as Highcharts from 'highcharts'
```

Empezamos con la importación de los módulos o servicios necesarios, son exactamente iguales a diferencia del módulo FormControl y FormGroup, que son necesarios para la recepción y tratamiento de los datos obtenidos en el formulario para hacer una consulta acorde a los datos introducidos.

```
public options:any = {
  title: {
    text: 'Media de valores de cada contaminante'
  },
  subtitle: {
    text: ''
  },
  yAxis: {
    title: {
      text: 'µg/m3'
    }
  },
  xAxis: {
    categories: []
  },
  tooltip: {
    formatter: function() {
      switch(this.series.name) {
        case "PM10":
          return "<small>PM10 ~ < 35 µg/m3 = BUENO <br> <= 50 µg/m3 = ACEPTABLE <br> > 50 µg/m3 = MALO</small><br>" + "<table><tr><td style='color:' + this.series.color+'><b>'+this.series.name+'</b></td><td style=' + this.series.color+'><b>'+this.series.name+'</b></td></tr></table></td>";
          break;
        case "PM25":
          return "<small>PM25 ~ < 20 µg/m3 = BUENO <br> <= 25 µg/m3 = ACEPTABLE <br> > 25 µg/m3 = MALO</small><br>" + "<table><tr><td style='color:' + this.series.color+'><b>'+this.series.name+'</b></td><td style=' + this.series.color+'><b>'+this.series.name+'</b></td></tr></table></td>";
          break;
        case "NO":
          return "<small>NO ~ <= 30 µg/m3 = BUENO <br> > 30 µg/m3 = MALO</small><br>" + "<table><tr><td style='color:' + this.series.color+'><b>'+this.series.name+'</b></td><td style=' + this.series.color+'><b>'+this.series.name+'</b></td></tr></table></td>";
          break;
        case "CO":
          return "<small>CO ~ < 10 µg/m3 = BUENO <br> > 10 µg/m3 = MALO</small><br>" + "<table><tr><td style='color:' + this.series.color+'><b>'+this.series.name+'</b></td><td style=' + this.series.color+'><b>'+this.series.name+'</b></td></tr></table></td>";
          break;
        case "CO2":
          return "<small>CO2 ~ < 500000 µg/m3 = BUENO <br> > 500000 µg/m3 = MALO</small><br>" + "<table><tr><td style='color:' + this.series.color+'><b>'+this.series.name+'</b></td><td style=' + this.series.color+'><b>'+this.series.name+'</b></td></tr></table></td>";
          break;
      }
    }
  },
  legend: {
    layout: 'vertical',
    align: 'right',
    verticalAlign: 'middle'
  },
  series: []
}
```

Aquí tenemos las opciones que tendrá y recibirá la gráfica.

```
createFormGroup() {
  return new FormGroup({
    estacion: new FormControl(''),
    fechaInicial: new FormControl(''),
    fechaFinal: new FormControl(''),
  })
}
contactForm: FormGroup;
```

Ejecutamos la función `createFormGroup()` que permite crear el esquema del formulario según nuestras necesidades, en este caso solo pediremos la estación, fecha inicial y fecha final.

```
onSaveForm(){
  let contVal = [
    {
      name : "",
      data:[]
    },
    console.log(this.contactForm.value)
  ]
  this._propios.postAPI(this.contactForm.value).subscribe( data => {
    console.log(data)
    let fechas = new Array();
    let contaminantes = new Array();
    let contVal = new Array();
    let valCO = new Array();
    let valPM10 = new Array();
    let valNO = new Array();
    let valCO2 = new Array();
    let valPM25 = new Array();
    for (const key in data) {
      let enc = false;
      let encUno = false;
      //key son numeros
      //empieza comprobacion de repetidos de fecha

      fechas.forEach(element => {
        if (data[key]["fecha"].substr(0,10) == element) {
          enc = true
        }
      })
    }
  });
}
```

Creamos la función `onSaveForm()` para que cuando el formulario sea rellenado y ejecutado haga lo que nosotros deseamos. Al hacer la petición nosotros le pasamos los valores que hemos obtenido del formulario y nos devolverá los valores que deseamos, según lo que hayamos introducido en él.

En la imagen anterior hemos inicializado todas la variables que nos harán falta en más adelante.

```
for (const key in data) {  
  let enc = false;  
  let encUno = false;  
  //key son numeros  
  //empieza comprobacion de repetidos de fecha  
  fechas.forEach(element => {  
    if (data[key]["fecha"].substr(0,10) == element) {  
      enc = true  
    }  
  });  
  
  // añadimos la fecha en caso de que no este repetido  
  if (!enc) {  
    fechas.push(data[key]["fecha"].substr(0,10))  
  }  
  
  // comprobacion de repetidos en contaminantes  
  contVal.forEach( element => {  
    if (data[key]["contaminante"] == element["name"]) {  
      encUno = true;  
    }  
  })  
  //añadimos los contaminantes en caso que no esté repetido  
  if (!encUno) {  
    contaminantes.push(data[key]["contaminante"])  
  }  
}
```

Recorremos los datos obtenidos y para pasarle los datos a la gráfica necesitamos introducirle las fechas sin repetir, por tanto se realiza la comprobación como se indica en la imagen anterior y también haremos lo mismo con los contaminantes y en caso de que no esté repetido se añadirá a sus arrays correspondientes

```
// almacenamos la cantidad de datos
switch(data[key]["contaminante"]){
  case "CO":
    if (data[key]["valor"] == null){
      valCO.push(0)
    } else {
      valCO.push(Math.round(data[key]["valor"]))
    }
    break;
  case "PM10":
    if (data[key]["valor"] == null){
      valPM10.push(0)
    } else {
      valPM10.push(Math.round(data[key]["valor"]))
    }
    break;
  case "NO":
    if (data[key]["valor"] == null){
      valNO.push(0)
    } else {
      valNO.push(Math.round(data[key]["valor"]))
    }
    break;
  case "CO2":
    if (data[key]["valor"] == null){
      valCO2.push(0)
    } else {
      valCO2.push(Math.round(data[key]["valor"]))
    }
    break;
  case "PM25":
    if (data[key]["valor"] == null){
      valPM25.push(0)
    } else {
      valPM25.push(Math.round(data[key]["valor"]))
    }
  default:
    break;
}
```

Aquí añadiremos los valores según si el contaminante es uno u otro para luego introducirlos con una estructura como se mostrará en la siguiente imagen

```
contVal = [  
  {  
    name: "CO",  
    data: valCO  
  },  
  {  
    name: "PM10",  
    data: valPM10  
  },  
  {  
    name: "NO",  
    data: valNO  
  },  
  {  
    name: "CO2",  
    data: valCO2  
  },  
  {  
    name: "PM25",  
    data: valPM25  
  }  
]
```

En la variable contVal vemos los contaminantes que queremos sacar, con sus datos correspondientes y ordenados por fechas.

```
this.options.xAxis["categories"] = fechas  
this.options.series = contVal  
this.options.subtitle["text"] = "Estacion: "+data[0]["estacion"]  
Highcharts.chart('propios', this.options)
```

En esta imagen le pasamos al eje X de la gráfica las fechas ordenadas y sin repetir. También añadimos donde van los datos dentro de las opciones de la gráfica y por último para saber que estación es la que se está mostrando en la gráfica pues le añadimos el subtítulo con la estación que se haya añadido en el formulario.

Hacemos llamada al gráfico y se mostrará la gráfica en caso de que los datos introducido existan.

Claves de plataformas

Las credenciales del correo como de Heroku son las siguientes:

Correo: proyectoreecalidaddelaire@gmail.com

Contraseña: REEyCA_2020

Tecnologías utilizadas

Las tecnologías utilizadas han sido:

- Heroku
- Angular 9
- MongoDB
- HighCharts
- TypeScript
- Node.js