1. Introducción

La sociedad actual necesita el uso de la electricidad para hacer la mayoría de las tareas cotidianas, especialmente por la noche para muchas más actividades ya que es cuando no tenemos luz natural. Algunas de las tareas o actividades que necesitamos hacer por la noche podremos hacerlas en casa, en cambio otras, ante la necesidad de usar la calle para desplazarnos de un lugar a otro externo al hogar, no podremos hacerlas si no somos capaces de ver el espacio que nos rodea y es por ello que precisamos del uso de iluminación artificial en las carreteras y vías peatonales. Sabemos que el uso de la iluminación artificial altera el entorno en el que vivimos empeorando la percepción del cielo nocturno desde las ciudades y sus alrededores. Esto es un hecho que tenemos muy normalizado desde los primeros años de nuestras vidas que rara vez nos ha hecho plantearnos si supone algún tipo de problema para los humanos y los animales. Es por ello que este trabajo pretende facilitar el estudio de este fenómeno también conocido como Contaminación Lumínica.

1.1 ¿Qué es la contaminación lumínica?

La contaminación lumínica, también llamada a veces contaminación luminosa, es la iluminación ineficiente de nuestras ciudades durante la noche, la cual emite parte de la luz hacia el cielo provocando que éste tenga una luminosidad antinatural que conlleve una peor percepción del cielo.

Entre las **principales causas** encontramos:

- Mal diseño de las farolas: parte de su luz se emite innecesariamente hacia arriba como las farolas tipo globo. [https://www.naturalizaeducacion.org/2019/05/14/contaminacion-luminica/]
- Exceso de potencia en las luces artificiales. [https://www.naturalizaeducacion.org/2019/05/14/contaminacion-luminica/]
- Existencia de horarios poco eficientes de la iluminación artificial. [https://www.naturalizaeducacion.org/2019/05/14/contaminacion-luminica/]
- Uso de lámparas que emiten en el espectro de luz que no es útil para el ser humano, pero que afecta a otros seres vivos o el desarrollo de actividades como la astronomía.
- La mala instalación de la luminaria no ilumina el objeto o espacio que se necesita.
- Cantidad excesiva de luminarias o una intensidad luminosa inadecuada que hace que se refleje en el cielo.

Las principales consecuencias de este fenómeno son:

- Energéticas: se desaprovecha más de un 25% de la energía, lo que nos lleva a la siguiente consecuencia que es el mayor coste de la luz.
- Económicas: más del 50% por ciento de la factura de la luz de un municipio se debe al alumbrado público, por lo que el desperdicio de la luz hacia zonas que no necesitan estar iluminadas conlleva un aumento de la carga impositiva al ciudadano.
- Ambientales: este desperdicio energético se traduce en mayores emisiones de gases contaminantes o mayores residuos radiactivos según el tipo de centrales eléctricas que provean esta energía. También afecta al ciclo vital de especies animales, vegetales y a los seres humanos.
- Culturales: impide contemplar el cielo estrellado con calidad. [https://www.greenglobe.es/contaminacion-luminica/]

1.2 Motivación del proyecto

Este trabajo pretende monitorizar la calidad lumínica del cielo nocturno de la provincia de Granada realizando una serie de mediciones en distintos puntos de ésta.

Todo esto se realiza mediante una aplicación web que permite a los investigadores que realizan estas mediciones subir sus resultados acompañados de fotografías, interpolaciones gráficas y algunos datos adicionales como la localización, fecha, autoría y su ubicación geográfica en un mapa de la provincia de Granada.

Todo esto se realiza con el objetivo de concienciar a la población sobre el impacto que produce la contaminación lumínica en nuestro ecosistema, las alternativas que podemos seguir para reducir este fenómeno y dar paso a que las autoridades municipales puedan contactar con nosotros para que realicemos mediciones en su localidad dando lugar a la promoción de su calidad de cielo.

1.3 Aparato de medición

Para realizar las distintas mediciones se ha utilizado un dispositivo llamado SQM, Sky Quality Meter (o Medidor de Calidad del Cielo en español) que calcula la cantidad de brillo del cielo nocturno de forma estandarizada. Es un instrumento que acerca a la ciudadanía a una experiencia científica pudiendo mapear una ciudad en distintos puntos localizando los oasis de oscuridad y midiendo los cambios que se producen en la oscuridad del cielo a lo largo del tiempo y del mundo.

Los pasos para realizar una medición son:

i. Determinar las coordenadas geográficas del lugar de la medición, es decir, latitud y longitud.
ii. Empezar a medir una hora después de la puesta de Sol cuando la noche es clara y no hay luna ni nubes. Es muy recomendable dejar el medidor en el lugar al menos 5 minutos para que capte la temperatura del ambiente antes de realizar las mediciones.
iii. Evitar usar el dispositivo cerca de cualquier tipo de iluminación como farolas y en áreas con sombras como debajo de árboles y edificios. En cielos muy oscuros puede tardar hasta un minuto para completar la medida.
iv. Informar de la medición indicando todos los datos.

Tras cada medición se obtiene un archivo de datos mostrados de la siguiente forma (los separadores pueden variar):

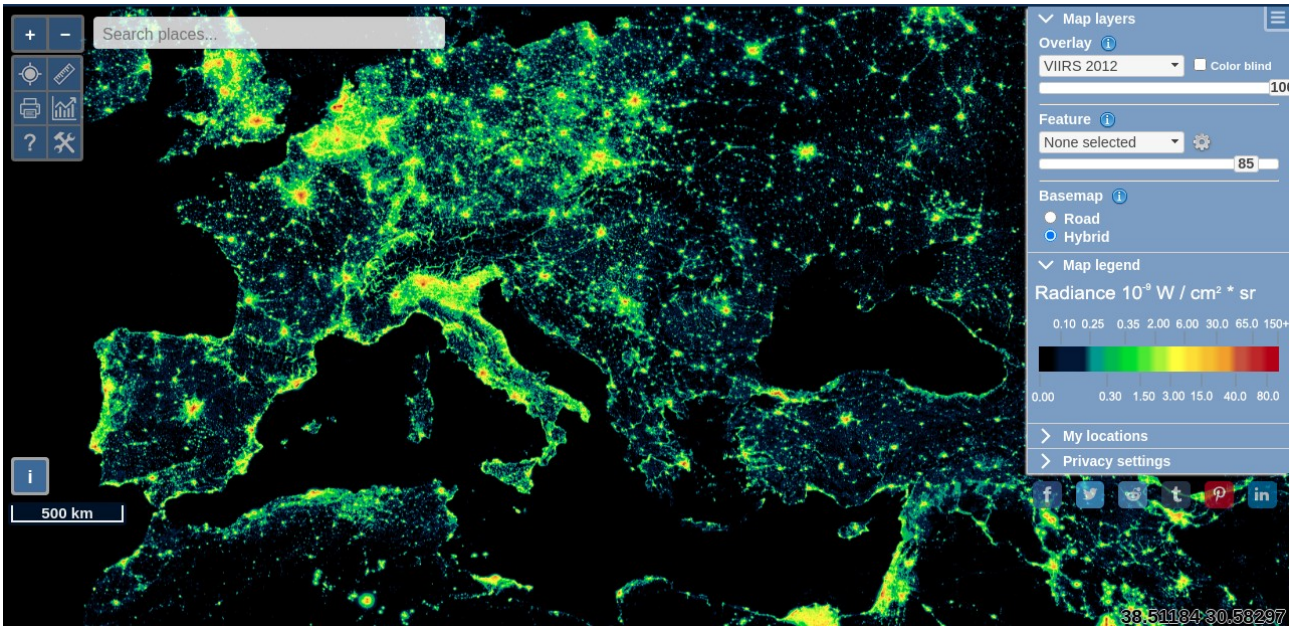
```
…
# UTC Date & Time, Local Date & Time, Temperature, Counts, Frequency, MSAS
# YYYY-MM-DDTHH:mm:ss.fff;YYYY-MM-DDTHH:mm:ss.fff;Celsius;number;Hz;mag/arcsec^2
# END OF HEADER
2013-04-29T18:07:23;2013-04-29T19:07:23;24.10;0.000;567881.400;-0.000
2013-04-29T18:08:25;2013-04-29T19:08:25;24.10;0.000;568008.000;-0.000
2013-04-29T18:09:28;2013-04-29T19:09:28;24.40;0.000;568071.800;-0.000
2013-04-29T18:10:30;2013-04-29T19:10:30;24.40;0.000;567961.200;-0.000
2013-04-29T18:11:33;2013-04-29T19:11:33;24.10;0.000;567939.400;-0.000
…
```

Fuentes: [https://www.globeatnight.org/sqm.php], [https://cielososcuros.org/monitoreo/sqm/], [https://icts-yebes.oan.es/reports/doc/IT-CDT-2013-7.pdf]

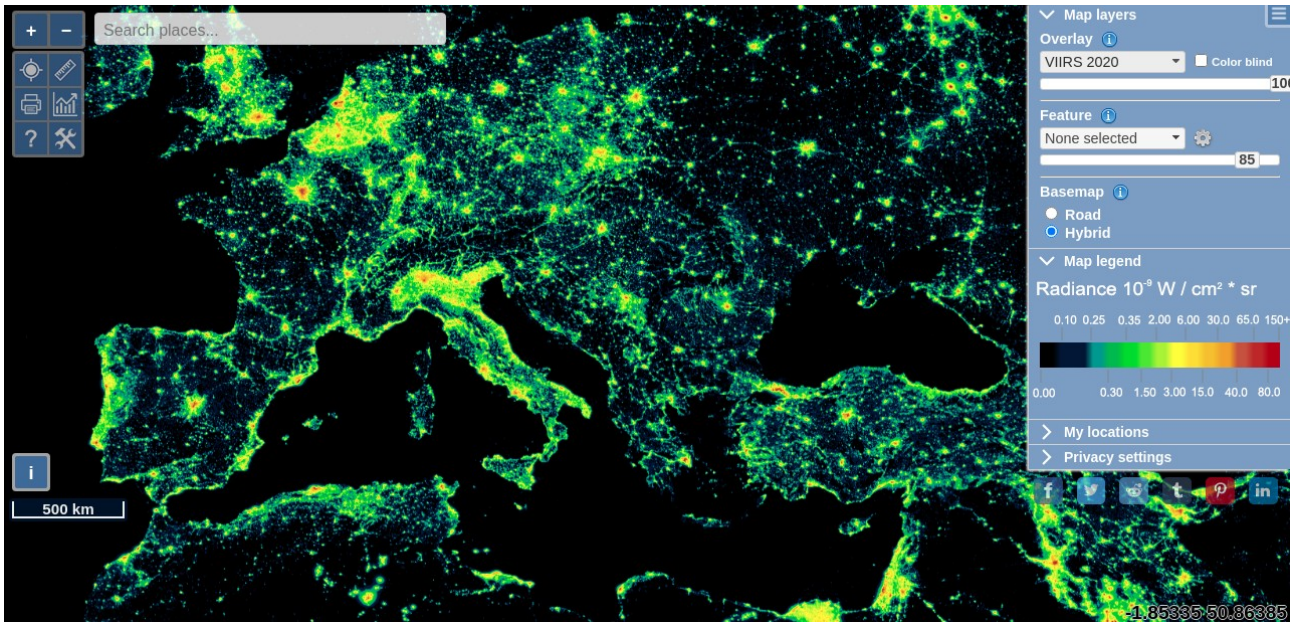
1.4 Contexto actual de la contaminación luminosa

Un patrón común en todo el mundo es el aumento anual de la emisión de luz, salvo en contados lugares, la contaminación lumínica no ha hecho más que aumentar cada año siendo cada vez más complicado encontrar lugares en los que se pueda contemplar el cielo estrellado. Para hacernos una idea de la magnitud de este fenómeno existen webs que nos muestran la calidad lumínica del cielo nocturno en diversas zonas, una de ellas es https://www.lightpollutionmap.info que nos permite interactuar con un mapa del mundo en el que podemos seleccionar el año de medición, ajustar el nivel de transparencia entre mapa normal y mapa lumínico, etc. Gracias a las distintas mediciones a lo largo de los años podemos apreciar cómo este problema ha ido aumentando poco a poco desde 2012, que es el año más temprano en el que esta página tiene medidas, hasta las últimas mediciones en 2020 tal y como se muestran en las siguientes fotografías:

Luminosidad en 2012:



Luminosidad en 2020:

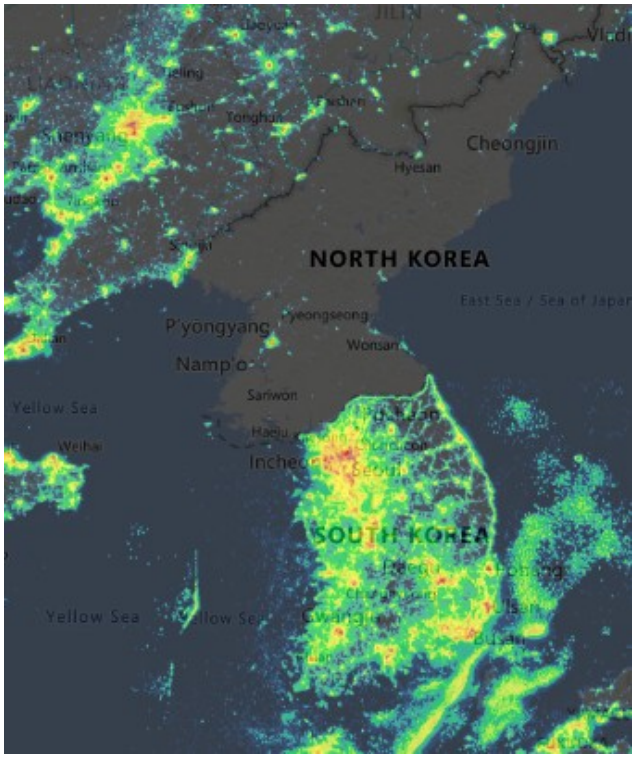


[poner enlace a gif de ambas imágenes]

Podemos apreciar que hay diferencias especialmente en las zonas que en 2012 no tenían mucha iluminación cómo en 2020 están más intensificadas como en el territorio de Turquía.

Podríamos decir que los mapas de luminidad nocturna nos pueden hablar de ciertos acontecimientos internacionales en función de su luminosidad o sus cambios de tendencia en la cantidad de luz emitida. Un caso que se puede apreciar en las anteriores fotografías es la zona norte de Siria y las luces que siguen el curso del río, ocurre justo lo contrario que en los demás lugares del mapa debido al largo conflicto bélico en el que está inmerso puesto que en 2012 tiene bastante más iluminación que en 2020.

Otro caso muy llamativo es el de las dos coreas que viven en sistemas políticos radicalmente opuestos pudiendo apreciar este enorme contraste de luz:



Sin embargo, que unas zonas estén menos iluminadas que otras no tiene por qué implicar siempre un peor desarrollo, en Europa tenemos el sorprendente caso de Alemania que pese a tener un gran desarrollo urbanístico, tiene una mejor calidad de cielo que España. La ciudad de Berlín teniendo casi los mismos habitantes que Madrid emite mucha menos luz al cielo y el 58% de los alemanes pueden ver la Vía Láctea.

En términos globales, el 83% de la población mundial está afectada por la contaminación lumínica, alcanzando el 99% para los europeos y estadounidenses.

Por último creo que es importante mencionar que existen más opciones de páginas que registran esta iluminación artificial nocturna de formas distintas como son
http://www.nightearth.com,
https://blue-marble.de/nightlights/2012,
https://www.arcgis.com/ y
https://jwasilgeo.github.io/esri-experiments/earth-at-night/

[https://elpais.com/elpais/2016/06/10/ciencia/1465550834_950273.html]

1.5 Frameworks de programación web

Para realizar este proyecto utilizo PHP como lenguaje de programación web para páginas dinámicas junto con el framework Symfony que permite de forma muy cómoda crear la arquitectura modelo-vista-controlador, crear rutas, añadir entidades en las que definimos a su misma vez las tablas de base de datos en las que éstas se almacenan, clases, crear formularios, migraciones para actualizar el esquema SQL entre las principales utilidades. También se hace uso del gestor de plantillas Twig que facilita mucho que las páginas mantengan un mismo esquema base sobre el que añadir bloques de etiquetas HTML.

También se hace uso de *qgis* (*geografic information system*) que es una aplicación de SIG de Software Libre y de código abierto. Nos proporciona un mapa de la provincia de Granada en el que situamos todas las mediciones que se hacen a partir de las coordenadas de latitud y longitud de cada una. Además asignamos a cada observación un valor llamado cenit que consiste en el valor de magnitud (u oscuridad) en los 90º de declinación. Este valor lo usamos para relacionar distintas tonalidades de colores en la miniatura en forma de punto de cada ubicación ya que si el cenit es pequeño significa que hay bastante luminosidad (color claro) y en cambio si tiene un valor más elevado estamos hablando de una zona bastante oscura (y por tanto, usamos un color más oscuro). De esta forma, podemos facilitar el filtrado de observaciones en función de la cantidad de luz que haya de media como podemos observar en la siguiente imagen.

1.6 Asignaturas útiles para este trabajo

La realización de este trabajo no sería posible sin contar con las asignaturas que he superado a lo largo de toda la carrera. En especial me gustaría destacar algunas de ellas ya que han resultado mucho más relevantes que el resto para poder enfrentarme al diseño e implementación de este proyecto.

Para poder programar esta aplicación web al igual que cualquier otro tipo de programa informático es esencial manejar bien los contenidos que se estudian en las asignaturas **Fundamentos de Programación (FP) y Metodología de la Programación (MP)**.

Teniendo capacidad para resolver problemas de programación es muy importante definir con exactitud qué es lo que quiere el cliente que hagamos y a partir de ello saber los requisitos de nuestro programa, hace falta estructurar la solución a diseñar y establecer un buen diseño de clases cuya comunicación entre ellas sea lo más eficiente posible. Esto es algo que nos facilita la asignatura de **Fundamentos de Ingeniería del Software (FIS)** sin la cual sería complicado decidir cómo empezar y por supuesto nos arriesgaríamos a entregar un producto que no esperara el cliente y, probablemente, mal diseñado. Una vez tengamos el diseño, éste no tendría valor si no supiéramos cómo implementarlo en código, es por ello que en **Programación y Diseño Orientado a Objetos (PDOO)** se aprende a escribir código a partir de una serie de diagramas proporcionados.

En algún momento del proyecto será necesario guardar una serie de datos y posteriormente acceder a ellos eficientemente, por lo que es crucial diseñar la base de datos adecuada para la información que necesitaremos almacenar, algo que se aprende en **Fundamentos de Bases de Datos (FBD)** donde no sólo se aprenden los comandos necesarios para manejar la base de datos, sino también a diseñarla previamente mediante los esquemas de Entidad-Relación.

El desarrollo de un programa web debe realizarse con lenguajes adaptados para ello, como HTML para páginas estáticas, CSS para dar mejor presentación y estilo, PHP para programar páginas dinámicas y Javascript para controlar las páginas de forma asíncrona. Todas estas herramientas son contenidos básicos de la asignatura **Tecnologías Web (TW)**.

La creación de una aplicación web normalmente implica el manejo de algún framework que proporcione utilidades para ahorrar demasiado esfuerzo al desarrollador para estructurar archivos, crear entidades, rutas o mejor acceso y modificación de la base de datos entre otras posibilidades. Éste es el caso de **Desarrollo de aplicaciones para Internet (DAI)** en el que se hace una toma de contacto con distintos frameworks, perfeccionamos los conocimientos de PHP, manejamos plantillas similares a Twig con las que mantener una misma estructura básica de presentación de las páginas y desplegamos la aplicación en Internet una vez finalizada.

2. Objetivos

Lista de objetivos principales

1. **Objetivo principal**: crear una aplicación web para representar y almacenar las mediciones del cielo nocturno en la provincia de Granada. Para ello se debe:

    1.1. Sección de mediciones. Mostrar en una sección una lista de las mediciones cada una de forma resumida con la posibilidad de acceder a información más detallada.

    1.2. Sección informativa. Se crea una sección en la que se informa a cualquier visitante acerca de la importancia del problema de la contaminación lumínica.

    1.3. Subida de documentos. Se facilita una interfaz con la que se podrá subir a la aplicación informes con datos sobre las **fotografías del cielo nocturno en 360 grados** y demás archivos relacionados.

1.4. Información meteorológica. Con cada subida de datos se obtendrá información meteorológica del lugar en el que se ha realizado una medición para contextualizarla.

1.5. La información que se presente además de ser mapas indicativos de luminosidad debe poder contener descripciones, estimaciones, recomendaciones y demás consideraciones que estimen oportunas los investigadores.

1.6. Edición de informes. Se debe contar con la opción que permita alterar los datos de las mediciones únicamente para los usuarios registrados.

1.7. Búsqueda de mediciones. Por medio de la interfaz, el usuario o visitante podrá hacer un filtrado con el que obtener el informe o los informes de medición que desea consultar.

2. **Objetivos respecto al uso de la plataforma:**

2.1. Gestionar usuarios. El usuario administrador dará de alta o de baja únicamente a quienes trabajan en este proyecto investigando o subiendo información.

2.2. Facilitar contacto a interesados. Las autoridades municipales interesadas en tener una medición de calidad del cielo nocturno de su localidad tendrán una sección con la que comunicarán su intención mediante un formulario que envía un correo al administrador de la plataforma.

3. Objetivos adicionales

3.1. Ofrecer varios idiomas. Además de que la página tenga por defecto el idioma español, los usuarios y visitantes podrán seleccionar otro idioma extranjero con el que leer todo el sitio web.

3.2. Bot de Telegram. Se creará un bot de Telegram para este programa que informará con un mensaje en el momento en que algún usuario registrado haya subido nueva información.

3.3. Actualizar la plataforma. Se proporcionarán nuevas mejoras a lo largo del tiempo en las que se añadirán nuevas funciones y/o se mejoren las ya existentes.

3.4. Mostrar mediciones en un mapa. Se marcarán todas las ubicaciones donde se ha medido la calidad del cielo en un mapa interactivo de la provincia de Granada. Permitirá ofrecer resúmenes de cada observación y filtrar por luminosidad media.

3. Metodología

Este trabajo de fin de grado se llevará a cabo siguiendo una metodología de **desarrollo incremental con feedback del cliente** o **desarrollo incremental prototipado** en la que se añadirán funcionalidades a lo largo de todo el proceso acompañadas con reuniones con el cliente después de añadir cada funcionalidad. De esta forma podemos resolver errores cometidos recientemente, adaptar cada nueva función y no arriesgamos a una posible discrepancia con el cliente al final de todo el proceso.

Podríamos definir una serie de tareas clave en el desarrollo de este producto, algunas de ellas repetidas en bucle dada la naturaleza iterativa de esta metodología.

1) Obtención de los requisitos con el cliente, refinamiento de los mismos y confirmación con el cliente.
2) Definición de tareas y funcionalidades a añadir al proyecto.
3) Diseñar la funcionalidad. Para ello se diseñan los diagramas de clase, diagramas de secuencia, de actividad, etc.
4) Implementar la funcionalidad creando nuevas clases o modificando las ya existentes proveyendo de métodos y funciones que cumplan con los diagramas y realicen la tarea acordada.
5) Ejecución de pruebas.
6) Reunión con el cliente mostrando el trabajo realizando y comprobando que se cumple con lo acordado. En caso de que la funcionalidad no satisfaga las expectativas del cliente se realizarán las modificaciones oportunas volviendo al paso 4. Si la funcionalidad es correcta y tenemos el visto bueno del cliente, procedemos a realizar la siguiente funcionalidad retomando de nuevo el paso 3.
7) Presentación formal de la aplicación. Una vez terminadas de implementar correctamente las principales funcionalidades se procede a mejorar la capa de presentación de la aplicación mejorando así la forma con la que ve e interactúa el usuario. Para ello se realizan tareas de ajuste de tamaño imágenes, de letra o añadiendo diseño adaptable de elementos HTML al tamaño del dispositivo entre otras tareas.
8) Entrega del producto. Una vez completamente terminado se procede a entregarlo al cliente.

En cuanto a las funcionalidades que se implementan en la aplicación tenemos las siguientes:

- F1: Proveer de un esquema básico para las secciones de Inicio, Mapas y mediciones, Contacto y página con formulario Login. Esto se ha conseguido gracias al gestor de plantillas Twig que como se mencionó anteriormente, facilita mucho la inserción de nuevo código HTML sobre una misma base sin tener que repetirla en cada página.
- F2: Autenticación de usuario admin. Symfony encripta la contraseña usando el algoritmo “bcrypt” y tras acceder a la base de datos del usuario y comprobar que es correcto inicia sesión.
- F3: Lectura y subida de los archivos de medición. El programa es capaz de leer cualquier fichero de datos siguiendo un determinado formato y rechazando aquellos ficheros que no son de texto o que no siguen el formato correcto.
- F4: Modificación de datos del usuario como la biografía, su foto de perfil o el nombre de usuario.
- F5: Mostrar una galería de mediciones en las que poder acceder a los detalles de cada una de ellas.

- F6: Descargar la información meteorológica de Meteoblue del mismo día que se produce la subida.
- F7: Integrar una sección en la que se muestre un mapa de Granada y se marquen las ubicaciones en las que se hayan realizado observaciones del cielo.
- F8: Modificar los datos de las mediciones realizadas por parte de los usuarios registrados.
- F9: Gestión de usuarios del administrador pudiendo dar de alta, dar de baja y alterar a los usuarios registrados.

En el siguiente apartado podemos comprobar cómo se van añadiendo estas funcionalidades a la aplicación.

3.1 Temporización del proyecto

La planificación en un primer momento estaba pensada para terminar el producto a finales de junio. Sin embargo, no ha podido ser así y ha tenido que someterse a modificaciones para tenerlo todo listo para finales de agosto. La distribución de las distintas etapas son las siguientes:

- Parte 1: 15/03/2021 – 21/03/2021. El cliente presenta el producto a desarrollar, que es una aplicación web para la medición de la calidad del cielo nocturno. En este paso se establece con el cliente los requisitos y se deciden las herramientas a utilizar como la biblioteca de plantillas Twig, se investiga la instalación y funcionamiento del framework Symfony y plantillas HTML cuyo diseño base pueda encajar bien con este programa.

- Parte 2: 22/03/2021 – 04/04/2021. Se continúan probando más plantillas para la aplicación, se investigan las funcionalidades de usuarios con Symfony y las partes privadas reservadas para éstos. Se adaptan los templates a la temática del proyecto y se prueban cómo funciona Symfony con las bases de datos. Por último se crean y prueban los primeros formularios.

- Parte 3: 19/04/2021 – 02/05/2021.
  - Se decide que el modelo de datos de las mediciones que constarán de dos tablas: una para los datos genéricos y otra para cada dato individual de la medición, esta segunda tabla relacionada con la primera gracias a una clave foránea.
  - Se crea la tabla del usuario en la base de datos y se conocen el formato de los ficheros de medición que posteriormente leerá este sitio web.
  - Se registra el primer usuario llamado Admin que posteriormente se le proporcionará la capacidad para registrar o dar de baja a los usuarios, así como poder editar toda la información que se suba a la aplicación.
  - Se crea la página de edición de usuario en el que el usuario logueado tiene un formulario con el que puede cambiar su foto de perfil y su biografía.
  - Se comienza a realizar un formulario sin programar su funcionamiento para la subida de ficheros únicamente visible en la sección “Mapas y datos” para los usuarios registrados.

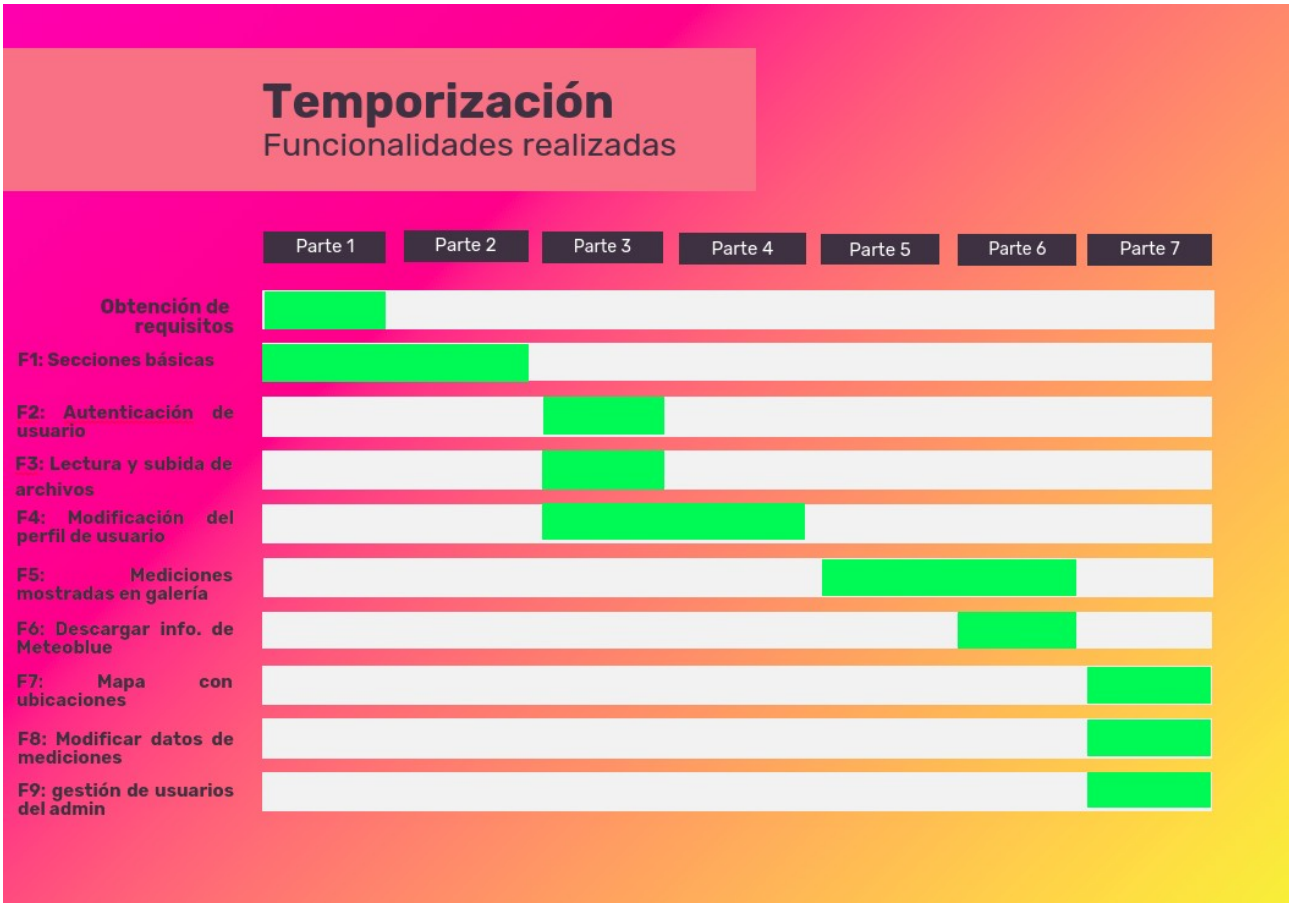
- Parte 4: 03/05/2021 – 16/05/2021. Se crea el modelo de datos para las mediciones anteriormente mencionado, por lo que ya estamos en condiciones de poder implementar la subida de archivos de medición con la condición de que tengan formato “.txt” para que posteriormente se programe la lectura de estos ficheros según el formato en el que aparecen los datos. Cada vez que se sube un nuevo archivo se crea una carpeta de la medición en cuestión y en ella subimos todo lo que tenga que ver ésta. Finalmente, una vez leídos los ficheros, se procede a almacenar la información recogida en la base de datos.

- Parte 5: 17/05/2021 – 30/05/2021. Uso el script de python “interpolador.py” con cada archivo de medición cada vez que lo subo para obtener una imagen .png. Esta imagen indica gráficamente el grado de oscuridad y luminosidad del lugar en el que se ha realizado la medición y se guarda en la carpeta de la medición. Se muestra en la sección de “Mapas y datos” información resumida de cada medición almacenada en forma de galería.

- Parte 6: 01/07/2021 – 18/07/2021. Como se ha comentado en el anterior paso, se presenta en forma de galería la información resumida de cada observación del cielo. En esta parte se le añade un enlace en cada una para acceder a información más detallada en otra página de estructura de presentación similar. Además de mostrar los datos guardados en la base de datos, en la página de detalles también se procede a obtener los datos de la página de Meteoblue respecto a previsión meteorológica los cuales se almacenan en la misma carpeta en la que se suben los archivos de la medición en cuestión. También es importante mencionar que se añade un botón para descargar todos los archivos de la carpeta de la observación en un zip y así pueda consultarlo cualquier usuario, tanto usuarios registrados como visitantes anónimos.

- Parte 7: 01/08/2021 – 22/08/2021(?). En esta última parte se terminan de pulir detalles en la programación del producto como añadir la imagen del segundo tipo de interpolación en los datos de cada medición y añadir las gráficas de la calidad del aire de ese día procedente de Meteoblue. También se proporciona la opción de borrar y de modificar la observación ya subida por parte de un usuario registrado pudiendo cambiar sus datos genéricos. Se añade además un campo extra para anotar observaciones por parte de quien suba los datos y se facilita la subida de más de una fotografía (pueden ser fotografías de la zona, no tienen por qué ser siempre del cielo).

Podemos observar el resumen de la temporización en el siguiente diagrama de Gantt:



3.2 Presupuesto y estimación del gasto

El cálculo del presupuesto inicialmente estaba pensado para entregar el producto a principios de julio. Sin embargo, debido a la falta de tiempo para poder dedicarme a desarrollarlo, se ha tenido que alterar el cálculo en vistas de su entrega a principios de septiembre.

Para dar un presupuesto al cliente he tenido que tener en cuenta una serie de costes que en el día a día podrían pasar desapercibidos y hay que apreciar si queremos que este proyecto sea rentable. Es por ello que he decidido calcular la depreciación del material utilizado, en este caso, mi portátil con su marca y modelo específico debido a que con el paso del tiempo los equipos tienden a estropearse y nos podríamos topar un día con el inconveniente de que deje de ser útil y por tanto es muy importante tener la capacidad para proveerse de otro material similar sin problema. Además no tenemos que olvidar el coste de ciertos tipos de seguros que se deben contratar y la carga impositiva correspondiente del país y la región en la que vivimos ya que estamos haciendo una tarea que hay que declarar a Hacienda. Por supuesto, este trabajo sería imposible hacerlo sin energía eléctrica, por lo que debemos tener en cuenta también su coste ya que mientras que se está utilizando para cargar el portátil o mantener el Internet encendido estamos perdiendo dinero. Dicho sea de paso, también es necesario añadir el coste mensual de Internet puesto que se utiliza para realizar las consultas ante los problemas que surjan y también para poder comunicarme con el cliente tras cada funcionalidad realizada. Finalmente es esencial señalar el coste por hora del tiempo que se dedica a trabajar en este producto que no se está dedicando a otras tareas.

Depreciación de equipo y herramientas: portátil Lenovo Legion Y-530	- Valor inicial aproximado: 600€ - Vida media: 5 años - Valor residual: 660€/5años = 132€ - Importe de depreciación mensual: (660€-132€)/5años = 105.6€/año = 105.6€/12meses = 8.8€/mes - Importe total de depreciación hasta junio: 8.8€/mes * 4 meses = **35.2€**
Tiempo de media de dedicación al proyecto	- Media jornada: 4h/día
Tiempo total de dedicación hasta septiembre	- N.º total de días: 102 días - Media de horas dedicadas por día: 4h/día - Total horas dedicadas: 102días * 4h/día = **408 horas**
Precio total de trabajo hasta finales de agosto	- Precio por hora de trabajo: 10€/h - **Precio total**: 408h * 10€/h = **4080€**
Seguro de autónomos	- Precio al mes: 300€/mes (jornada completa) - Precio para media jornada: 150€/mes - **Coste total:** 150€/mes * 4meses = **600€**
Seguro de responsabilidad civil	- Coste mensual: 150€/año = 150€/12meses = 12.5€/mes - **Coste total** hasta fin de junio: 12.5€/mes * 4meses = **50€**
Coste del **consumo de luz** utilizado durante el trabajo en el proyecto	- Media del precio del kilowatio hora: 0.36€/kwh - **Coste total** de consumo de luz: 0.36€/kwh * 436h = **156,96€**
Coste del **consumo de Internet**	- Coste mensual: 40.45€/mes - **Coste total** hasta finales de septiembre: 40.45€/mes * 4 meses = **161.8€**
Importe final sin impuestos	- Precio base: **5083’96€**
Gasto en **impuestos**	**- 21% IVA: 1067,63€** **- 15% IRPF: 762,59€**

IMPORTE FINAL con IVA e IRPF incluidos: **5389€**

4. Análisis y obtención de requisitos

Antes de comenzar tenemos que ser capaces de definir de la forma más específica posible el problema en cuestión y obtener toda la información que necesitemos para proceder a crear una solución software lo más adecuada posible. En este capítulo prepararemos el proyecto y acordaremos con el cliente todos los requisitos que se deben satisfacer para que finalmente obtenga el producto tal y como deseaba.

4.1 Planteamiento del problema

La cuestión que nos surge cómo actuar ante el problema de la contaminación lumínica del cielo nocturno. Como se comentó en la introducción, la contaminación luminosa está relacionada con la cantidad de luz o falta de oscuridad que hay en diversas zonas, en nuestro caso concreto, en municipios de la provincia de Granada. Naturalmente, no es posible crear un software que directamente resuelva este problema, pero sí que podemos crear una aplicación web que lo monitorice y pretenda crear conciencia en la población acerca de este fenómeno. Como consecuencia, vamos a tener que ser capaces como mínimo de:

1. Recibir unos datos (en algún tipo de fichero).
2. Leerlos.
3. Procesarlos correctamente.
4. Almacenarlos en una base de datos.
5. Presentarlos de forma comprensible a los usuarios humanos.

También es importante considerar que la monitorización tiene dos partes fundamentales: la medición y el procesamiento de los datos. Ambas no tienen por qué ser realizadas en el mismo dispositivo ni en el mismo programa en nuestro caso, por tanto, podemos separar estas tareas de forma que podamos especializarnos más en el tratamiento de la información recibida haciendo estas 5 labores citadas y delegar la tarea de medición a dispositivos más adecuados para ello como el SQM mostrado en la introducción.

Por tanto, podemos confirmar que es totalmente viable el desarrollo de este trabajo y podemos empezar a concretar todos los quehaceres con el cliente.

4.2 Información necesaria para crear una solución

Llegados a este punto deberíamos profundizar más en cómo debe ser el producto final. Queremos crear un sitio web que reciba datos de distintas ubicaciones en la provincia de Granada que puedan ser leídos por el programa, puedan ser procesados y se detecten los cambios que se hayan producido en el mismo sitio a lo largo del tiempo. Para esta tarea se necesita un equipo de personas que vayan realizando esas mediciones y las suba a la aplicación, por lo que no se le puede dar acceso a cualquiera, debemos tener una serie de usuarios identificados a los que conceder acceso y un usuario administrador que permita dar de alta o dar de baja a las cuentas de quienes participen en este proyecto. Estos usuarios registrados por tanto podrán acceder y usar formularios y secciones al contrario que los usuarios anónimos que son simples visitantes.

Además implicaría el uso de una sección para acceder a las mediciones realizadas en el caso de usuarios visitantes y subir, eliminar o modificar las ya existentes si somos usuarios registrados en la página. Esta sección tendría un formulario para todas las operaciones que impliquen alguna operación con la base de datos incluyendo también el filtrado para ayudar en la búsqueda.

También debemos facilitar que los municipios que así lo deseen puedan solicitar que midamos la calidad de su cielo, por lo que sería necesario establecer una sección de Contacto en la que mediante un formulario se envíe un correo al administrador de la página.

4.3 Casos de uso

Previamente a la definición de los requisitos describo los casos de uso ya que es importante acordar con el cliente cómo serán las distintas interacciones que se producirán entre el sistema y los actores, que en este caso serían dos: los usuarios registrados y visitantes anónimos.

- **C-1:** Subir archivos. Los usuarios registrados podrán subir los archivos con los datos lumínicos obtenidos en una ubicación concreta por la noche.

- **C-2:** Contactar con interesados. Los visitantes anónimos podrán enviar un mensaje que llegará al correo del administrador en el que podrá solicitar una medición del cielo nocturno de su localidad.

- **C-3:** Acceder a mediciones. Tanto los usuarios legales como los visitantes podrán consultar la información subida en la web.

- **C-4:** Modificar mediciones. Los usuarios dados de alta por el administrador podrán modificar la información contenida en la medición que estimen oportuna.

- **C-5:** Borrar mediciones. Al igual que en el anterior caso de uso, los usuarios registrados podrán borrar las mediciones que vean convenientes.

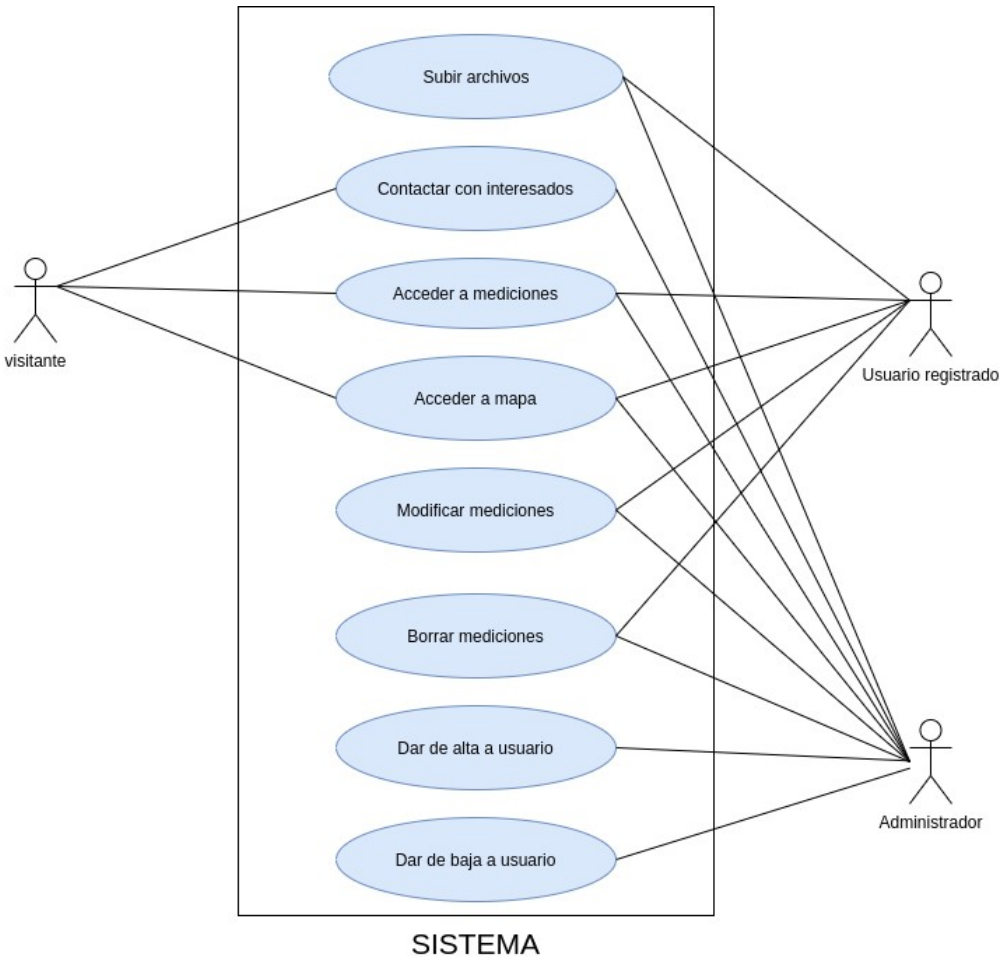
- **C-6:** Acceder a mapa. Tanto usuario logueado como anónimo consultan el mapa de Granada con las ubicaciones de las observaciones señaladas.

- **C-7**: Dar de alta. Sólo el usuario administrador puede dar de alta o de baja a nuevos usuarios.

- **C-8:** Dar de baja. Análogamente, el administrador tiene el poder de eliminar del sistema los usuarios que estime oportunos.

Para mostrar las principales interacciones entre los actores y el sistema tenemos el siguiente diagrama de casos de uso en el que mostramos las relaciones que hay entre cada caso de uso y el

actor o actores que lo activan pudiendo delimitar además la frontera entre el propio sistema y los agentes externos.



4.4 Lista estructurada de requisitos

Una vez conocidos los casos de uso, estamos en condiciones de extraer de ellos una lista estructurada de requisitos. Esta lista es el resultado final de todas las iteraciones que han ido ocurriendo a lo largo del proyecto que han ido refinando estos requisitos. Está dividida en sus principales tipos: **funcionales, no funcionales y de información**.

4.4.1 Requisitos funcionales

Son los requisitos relacionados con la funcionalidad del sistema y cada uno está relacionado con un caso de uso de los citados en el apartado anterior. De esta forma tenemos:

- **RF-1** Formulario de subida. El sistema proporciona un formulario en la sección de Mapas y Datos. Es únicamente visible para usuarios registrados en el que se muestra el

campo para seleccionar el archivo de datos de medición, un campo de texto para escribir la localización, otro campo de texto para comentar observaciones y otro campo para subir imágenes adicionales. Relacionado con **C-1**.
    - **RF-1.1** Crear interpolación gráfica. Cuando se sube un fichero de medición automáticamente se genera una interpolación gráfica mediante un script en Python. Éste lee los datos individuales formando una imagen de colores claros y oscuros en formato PNG que indican las partes claras y oscuras del cielo medido.
    - **RF-1.2** Obtener datos de la página Meteoblue. Cuando se sube un fichero de medición también de forma automática se descargan los ficheros HTML y CSS correspondientes de dicha página. De esta forma el programa accede a estos ficheros para mostrar información del día que se ha subido cuando se consulte la información detallada.
    - **RF-1.3** Escoger visibilidad. Quien suba información al sistema puede decidir qué partes hacer visibles y cuáles ocultar.
    - **RF-1.4** Formato de medición correcto. Cuando el usuario sube un archivo de medición el sistema comprueba que el fichero es de formato de texto, es decir, “.txt” y no se deja subir archivos con otras extensiones para prevenir errores. Relacionado con **C-1**.

- **RF-2** Formulario de contacto. En esta aplicación web se proporciona una sección de contacto en la que se explica a qué se dedica la gente que trabaja en ella y se facilita al final un formulario para que cualquier alcalde o autoridad de un municipio de la provincia de Granada rellene los campos informativos y comente detalladamente su intención de realizar una medición del cielo y la parte concreta de su localidad en la que desea que se haga.

- **RF-3** Acceso a mediciones. En la sección “Mapas y Datos” se muestra una serie de datos resumidos de observaciones subidas, todas mostradas en forma de galería siendo visibles tanto para visitantes anónimos como para registrados. Relacionado con **C-3**.
    - **RF-3.1** Acceso a detalles. Cada medición tiene una vista detallada de todos sus datos con la que se accede por medio de un enlace que se ve en la vista resumida. Al entrar en dicho link se observan los datos que se almacenan en la base de datos, las anotaciones correspondientes y además los datos meteorológicos del día de la medición en esa ubicación procedentes de la página Meteoblue.
    - **RF-3.2** Búsquedas de mediciones. Se proporciona un formulario con el que el visitante o el usuario escriba el nombre del lugar y obtenga todas las mediciones de ese sitio en la misma forma de galería.
    - **RF-3.3** Descargar información. Cualquier interesado puede ver en los detalles de la medición un botón con el que descargar todos los ficheros correspondientes de ésta en un mismo zip.

- **RF-4** Acceder a mapa. Desde la sección de galería de mediciones se puede acceder a un mapa de la provincia de Granada en el que se marcan las ubicaciones donde se ha medido. Relacionado con **C-4**.

- **RF-4.1** Filtrar ubicaciones. Desde el mapa de Granada se permite filtrar las ubicaciones por la media de magnitud (es decir, la “cantidad de oscuridad”). Así pues, mediante una barra podemos establecer el mínimo y máximo de ese valor medio y marcar sólo aquellos lugares que se encuentran dentro de ese rango de magnitud media.
- **RF-4.2** Mostrar foto y pequeño resumen de cada ubicación. Al pulsar en una localización cualquiera se nos muestra un bocadillo con un breve resumen de la medición y una fotografía de su cielo nocturno.
- **RF-4.3** (opcional) Mezclar interpolación gráfica con la fotografía original. Cuando se selecciona una ubicación en el mapa y se muestra la fotografía del cielo de esa ubicación, se procede a mezclar y comparar con la interpolación gráfica para apreciar mejor las zonas más iluminadas y las más oscuras.

- **RF-5** Modificar mediciones. Cualquier usuario registrado que accede a la información detallada de una medición puede ver la opción que permite modificarla. De esta forma se muestra un formulario con todos los campos correspondientes para editarlos al igual que cuando se sube por primera vez. Relacionado con **C-5**.

- **RF-6** Borrar mediciones. De la misma forma que el requisito anterior, cualquier usuario registrado puede contemplar la opción de borrar la medición cuando accede a su información detallada. Al pulsar se muestra una advertencia seguida de los botones “Aceptar” y “Cancelar” con la que confirmar la decisión.

- **RF-7** Editar perfil. Cada usuario registrado puede acceder a su perfil gracias a un enlace que aparece a un lado de la página junto con el resumen de sus datos. Tras pinchar en dicho enlace aparecen en la página todos sus datos y la opción de editar perfil con la que aparece un formulario con cada uno de los campos correspondientes para alterar la información que estime oportuna.

4.4.2 Requisitos no funcionales

- **RNF-1** Escala. Cuando se utilice algún mapa siempre debe estar siempre bajo una escala.

- **RNF-2** QGIS. El framework que se use para interactuar con mapas geográficos debe ser Qgis.

- **RNF-2** Lenguaje de programación. Sistema implementado principalmente en PHP.

- **RNF-3** Tailwind como framework CSS para la presentación de la página.

- **RNF**-4 Páginas dinámicas. Servir de forma dinámica gracias a PHP aquellas páginas cuyo contenido pueda variar.

- **RNF-5** Diseño adaptable. La aplicación debe tener disponibles distintos tamaños con los que mostrar contenido en diversos dispositivos.

- **RNF-6** Páginas estáticas. Servir de forma estática las páginas que no necesitan cambios como la sección de contacto.

- **RNF-7** Scripts. Usar el lenguaje Python para scripts adicionales que utilice el sistema.

- **RNF-8** Idioma. Español como lengua por defecto.

4.4.3 Requisitos de información

- **RI-1** Modelo de datos lumínicos. La información de la medición consiste en dos tablas: una para los datos genéricos y otra para los datos individuales.
  - **RI-1.1** Medición genérica. La tabla genérica se compone de toda la información que contextualiza la propia medición y la diferencia de las demás. Contiene ID, fecha, hora, latitud, longitud, nombre, localizacion, altitud, media de magnitud, bat, temperatura infrarroja, temperatura del sensor, nombre del gráfico y la autoría.
  - **RI-1.2** Medición individual. La tabla de los datos individuales consta de la propia información acerca del cielo nocturno que nos sirve para valorar su contaminación lumínica. Sus valores son: ID, ID de la tabla genérica, declinación, azimut y magnitud.
- **RI-2** Modelo de datos de usuario. Los campos de información que deben almacenarse en la base de datos respecto al usuario deben ser: ID, email, roles, contraseña, nick (el nombre de usuario), nombre completo y nombre de la foto de perfil.
  - **RI-2.1** Roles. Este campo debe tener formato Json y como mínimo debe indicar el rango jerárquico al que pertenece, es decir, si es administrador o normal.
- **RI-3** Ficheros relacionados. Para cada medición hay una carpeta en la que se guarda el propio archivo con los datos, los ficheros de Meteoblue, las interpolaciones gráficas y las observaciones.
- **RI-4** Archivos de usuario. Para cada usuario se crea una carpeta en el sistema donde se almacena su foto de perfil y un archivo de texto que se genera automáticamente conteniendo su biografía.

5. Diseño

La fase que nos va a garantizar una mejor implementación más ordenada y con menor probabilidad de ocasionar grandes problemas es la fase de diseño. Debemos tener en cuenta los requisitos descritos anteriormente ya que el sistema tiene que estar lo suficientemente adaptado

para satisfacerlos. En este capítulo se van a describir las clases que necesitamos, la interacción entre éstas, los modelos de datos y el flujo de trabajo.

5.1 Patrón Modelo-Vista-Controlador (MVC)

Esta aplicación está formada por el patrón de diseño Modelo-Vista-Controlador que es con la que trabaja Symfony. Organiza las clases y los archivos según sus funcionalidades concretas, por lo que es muy utilizado especialmente en aplicaciones que requieren una interfaz de usuario.

5.1.1 Modelo

Las clases que pertenecen al modelo son las que tratan los datos y acceden a la base de datos ya sea para guardar, borrar o actualizar. En este caso también dan forma a las tablas de las entidades que deseamos almacenar gracias a la herramienta ORM de Doctrine que es una abstracción que trabaja sobre la base de datos y permite el ahorro de sentencias SQL en el código. Las clases que implementan la capa Modelo son User, MedicionGenerica y MedicionIndividual. Contienen todos los datos que se almacenan en las tablas de la base de datos y los métodos set y get para su consulta y manipulación.

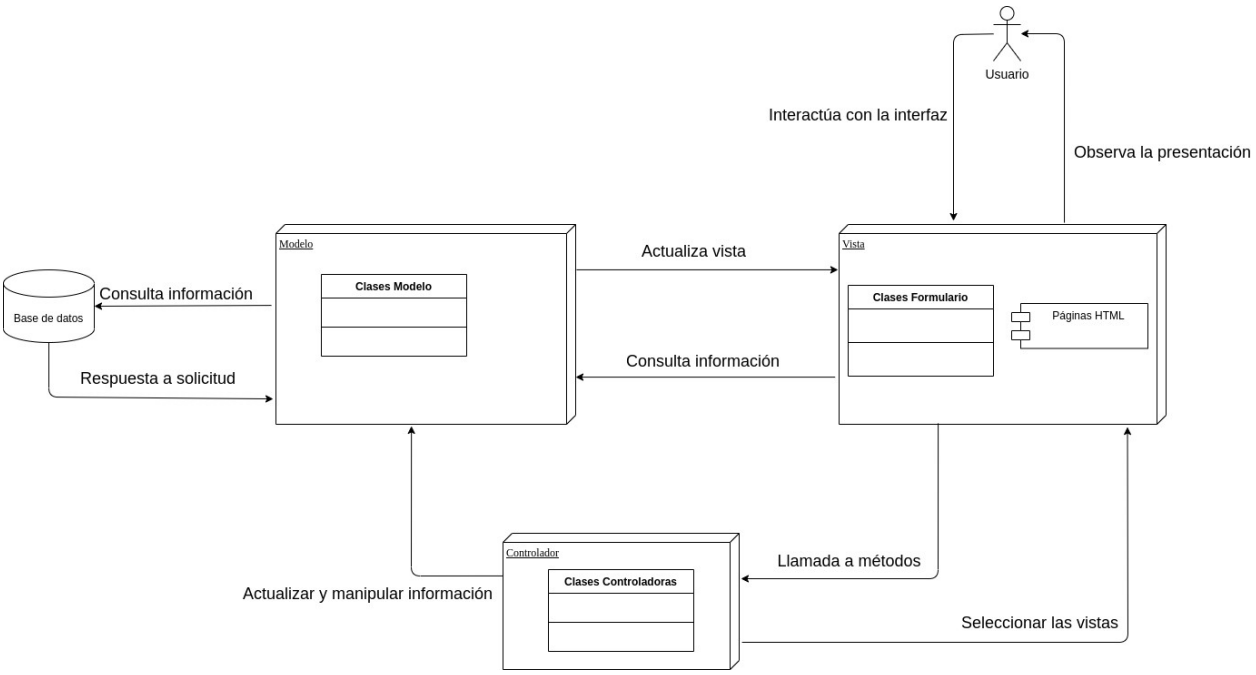
5.1.2 Vista

La presentación de la aplicación se consigue por medio de los archivos html y archivos que implementan los formularios. Nos muestran las imágenes, crean las formas, los textos y proporcionan un estilo que observamos al usar un sitio web. También es donde alojamos las plantillas que dan un aspecto profesional a las distintas páginas sobre las cuales incrustamos los elementos oportunos según a lo que estemos accediendo. A continuación se muestran los esqueletos de las plantillas que he utilizado para las páginas.

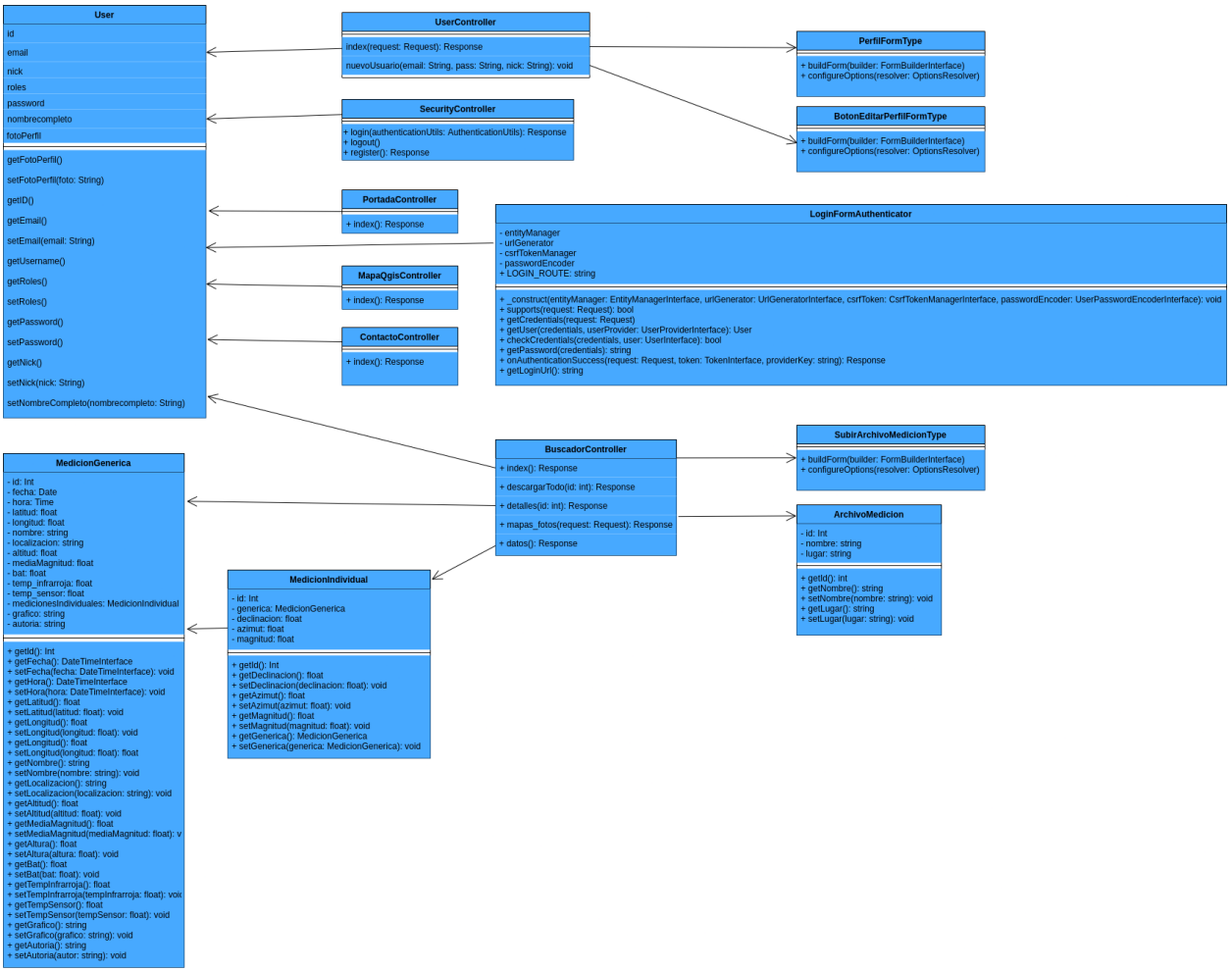
5.1.3 Controlador

Las clases que forman este módulo manejan el comportamiento de cada sección y la información que se manda a las vistas, y alteran o consultan la información almacenada en la base de datos. Por tanto, hacen posible que las páginas sean dinámicas y éstas actúen o cambien según las circustancias. En definitiva, dotan de la funcionalidad que da sentido al sitio web sin la cual nos quedaría un programa sin utilidad con páginas estáticas.

En el caso de nuestro programa, la interacción entre estos módulos principales se podría resumir a través del siguiente esquema:



Y tendríamos el siguiente diagrama de clases UML:

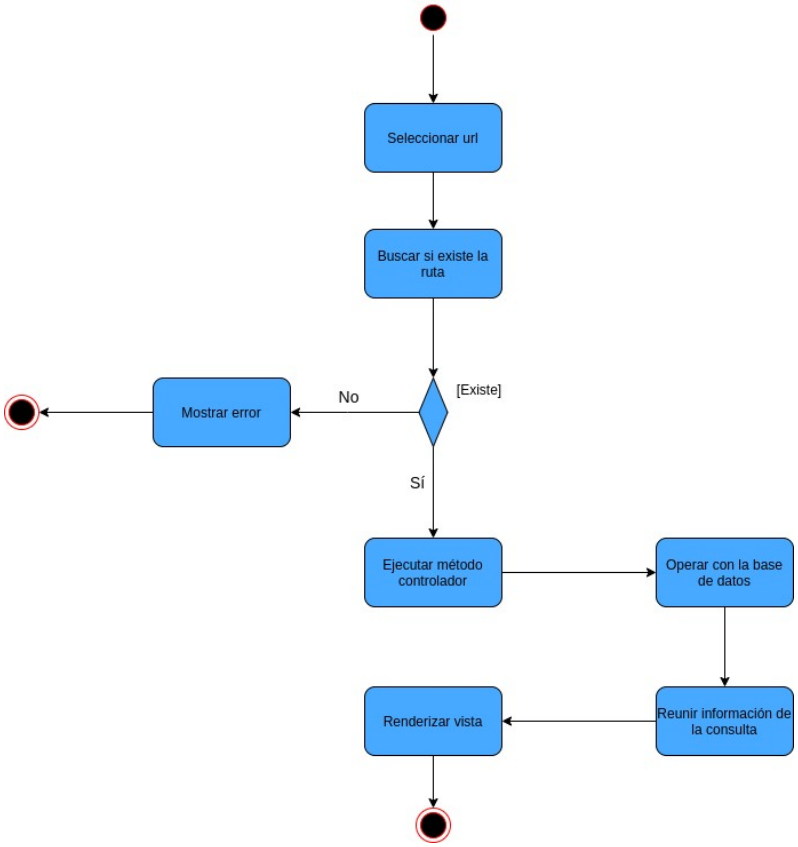


5.2 Funcionamiento

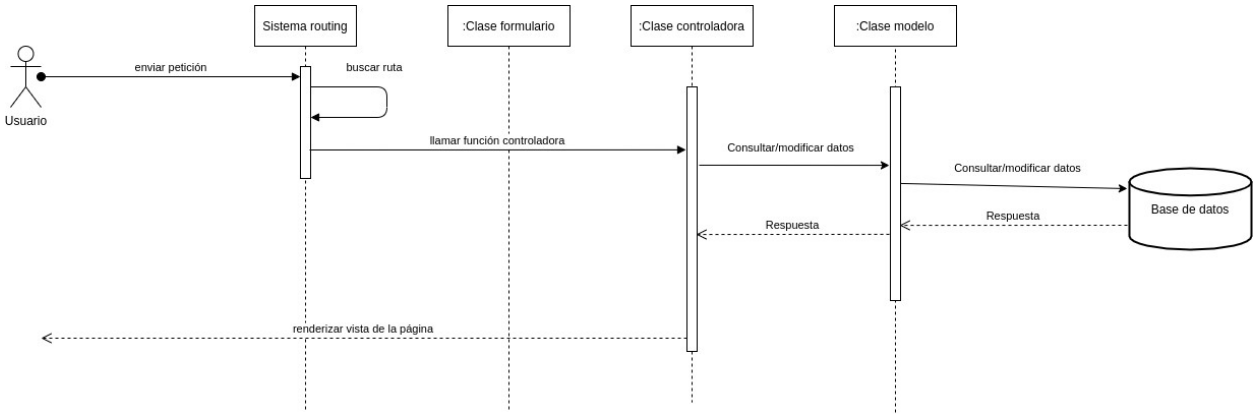
La actividad básica que ejecuta la aplicación consiste en mostrar una vista, detectar la url que ha solicitado el usuario a través de la interfaz, ejecutar funciones internas y mostrar esa página solicitada. Cada vez que el usuario accede a una sección cualquiera dentro de la página se realiza un proceso en el que la url correspondiente se compara con una lista de rutas en un archivo llamado routes.yaml. En este archivo cada ruta presenta tres campos:

1) Nombre. Es simplemente el identificador que la distingue del resto
2) Path. Es la ruta o url propiamente dicha.
3) Controller. Indica la clase controladora y el método de ésta que la maneja.

Posteriormente se accede al método de la url seleccionada que realiza las operaciones pertinentes y finalmente renderiza la vista accediendo al archivo html que corresponde. En esta renderización se pueden mandar los valores que deseemos para que sean considerados por las etiquetas de twig que forman las plantillas. Todo este proceso se resume en el siguiente diagrama:



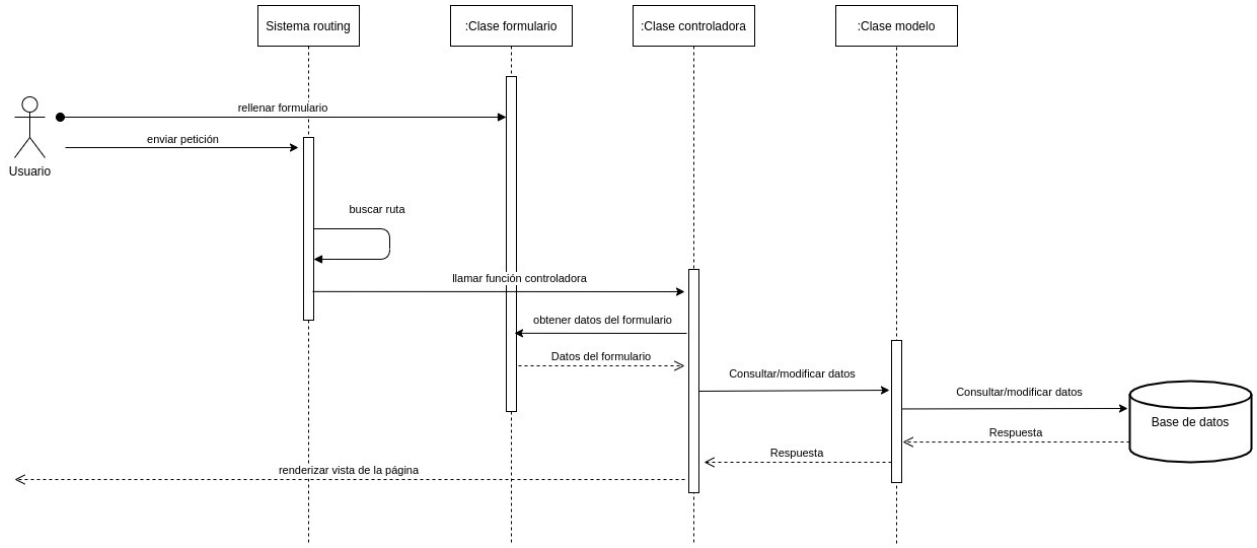
Este diagrama de actividad también vale para los casos en los que el usuario desea modificar su perfil o subir algún tipo de archivo, ya que posteriormente hay que volver a renderizar la vista para actualizarla. Sin embargo, todas estas actividades deben estar bien coordinadas entre las clases, hay que seguir un orden lógico de operaciones entre las distintas capas del patrón MVC y es por ello que en el siguiente diagrama dejamos descrita esta secuencia necesaria para el correcto comportamiento de la aplicación.



Observamos que, para empezar, siempre hay que iteractuar con el sistema routing. Este módulo interno de Symfony es el que detecta la petición que solicita el usuario y busca en la lista de rutas la clase y el método para satisfacerla.

[https://symfony.com/doc/5.2/routing.html]
[https://borrowbits.com/2013/04/que-es-como-funciona-symfony2-conceptos-claves/]

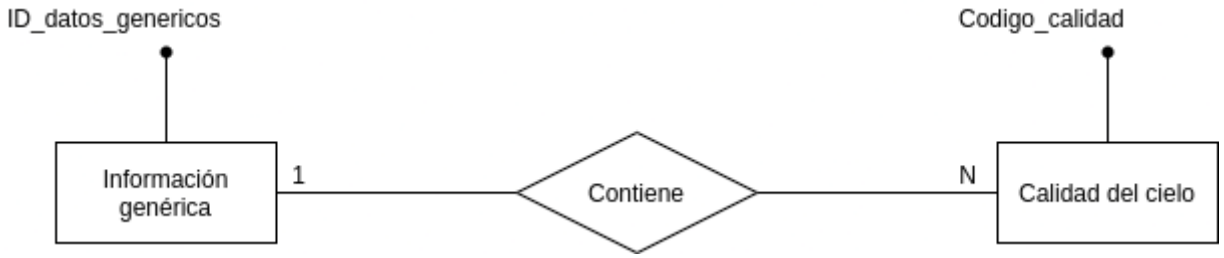
Cuando se realizan operaciones con formularios el orden de las acciones cambia ligeramente ya que necesitamos a la clase que crea el formulario para rellenar datos y obtener de ella la información que haya depositado el usuario. En el siguiente diagrama de secuencia se aprecian estas operaciones adicionales que se realizan con las clases que forman los formularios.



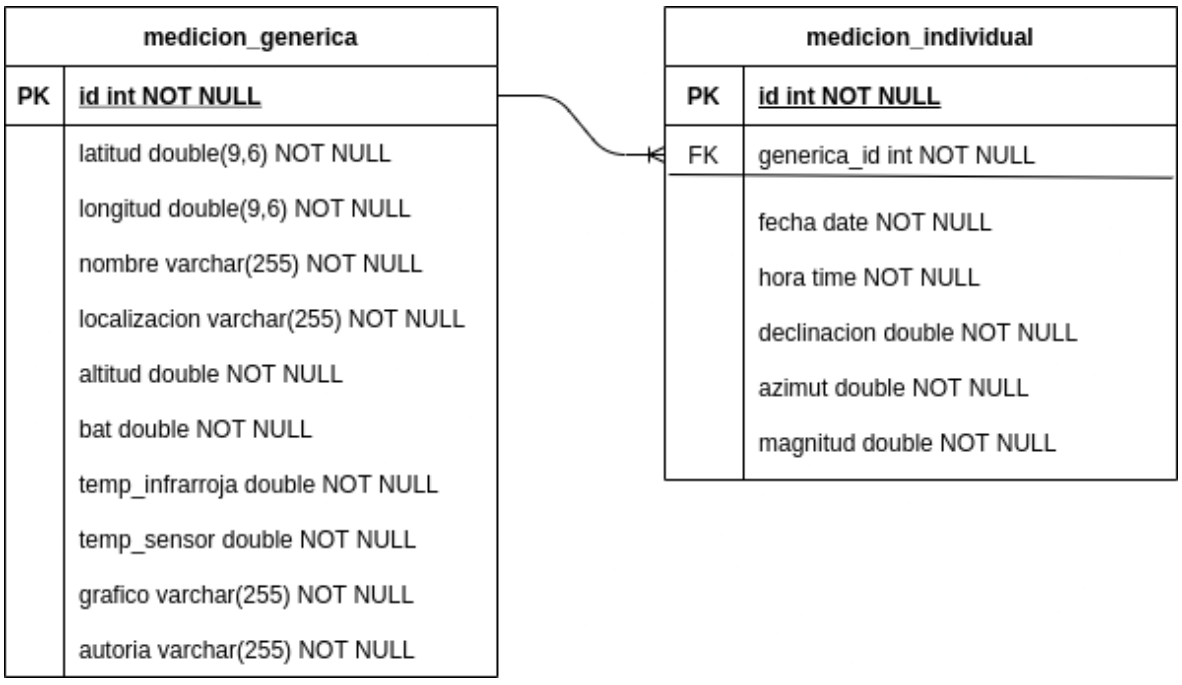
5.3 Modelo de datos

5.3.1 Datos de medición

Tal y como se explica en el requisito **RI-1**, para el almacenamiento de información se considera adecuado separar los datos en dos tablas: una para la información genérica y otra para la información luminosa que enlace con la primera a través de una clave externa. Por tanto, inicialmente nos surge el siguiente esquema Entidad-Relación en el que tendríamos una relación 1 a muchos ya que en cada lugar dadas unas coordenadas geográficas con el correspondiente nombre de la zona pueden producirse varias mediciones del cielo en distintos días o a distintas horas en un mismo día.

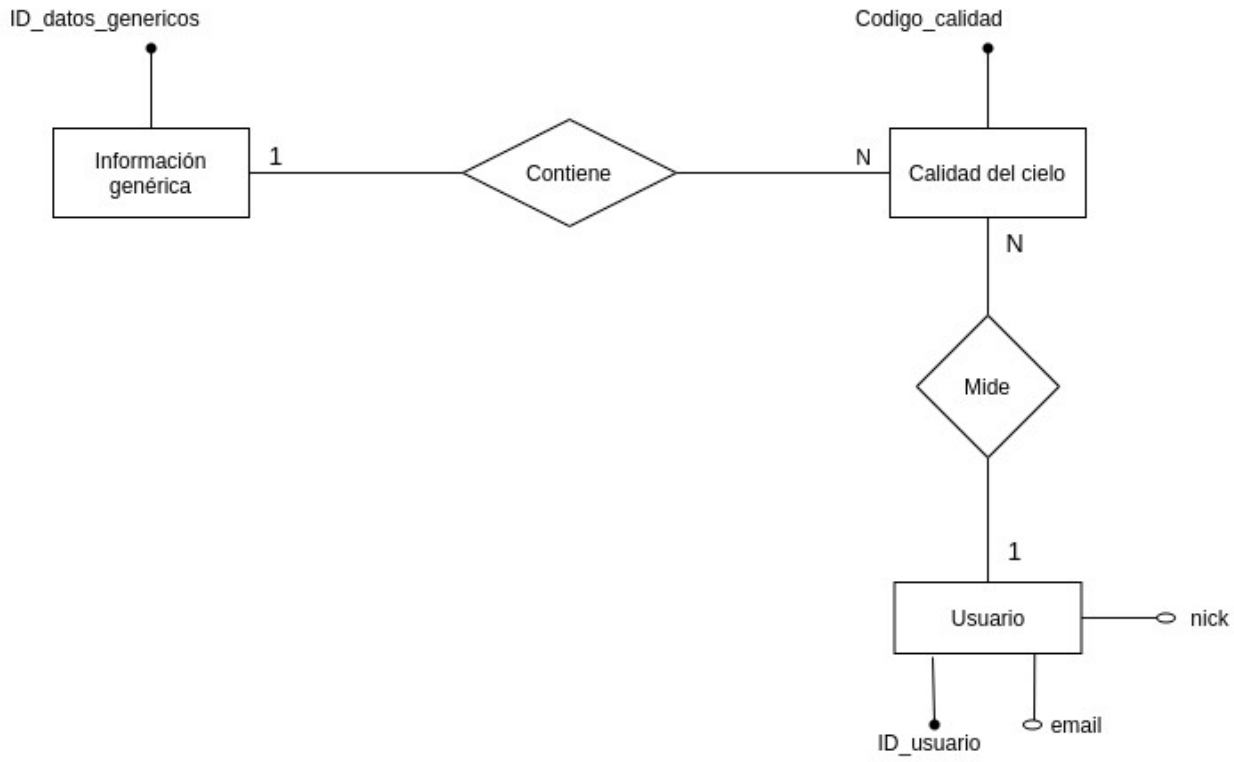


Como tenemos una relación 1 a muchos, sólo la entidad Calidad del cielo necesitará disponer de una clave foránea que nos indique la Información genérica a la que pertenece. De esta forma, añadiendo los campos de información que corresponden, tenemos las siguientes tablas que definirán nuestra base de datos.

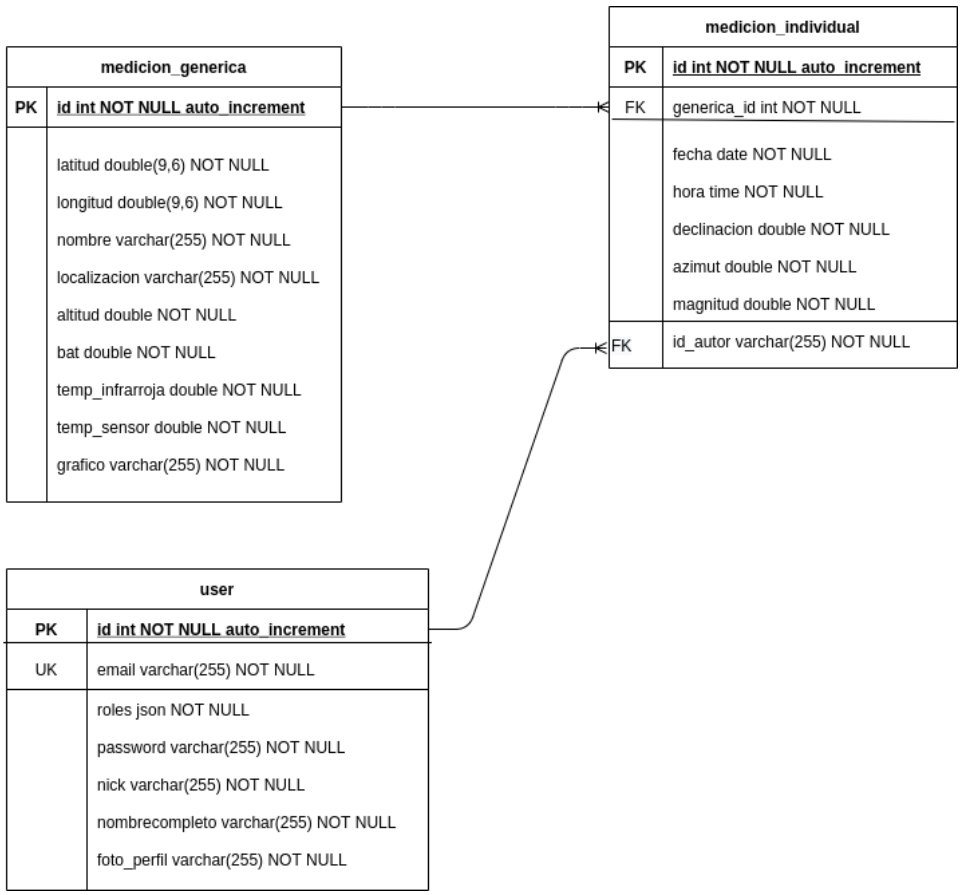


5.3.2 Usuario

En el caso del usuario registrado debemos tener en cuenta además de los datos que se procesan en la aplicación, su relación con las mediciones que ha subido puesto que cada una es medida por alguien que participa en esta web. De esta forma podemos relacionar cada usuario con la medición individual que ha hecho en un lugar ya que para un mismo sitio pueden haber muchas mediciones individuales cada una realizada por una persona distinta, y por supuesto, cada persona puede hacer todas las mediciones que estime oportunas. Así pues, ampliamos el anterior esquema entidad-relación de la siguiente forma añadiendo esta lógica que acabo de describir.



Y por consiguiente se elabora el siguiente diagrama de tablas también como ampliación del anterior.



6. Implementación

6.1 Instalación y herramientas

Antes de explicar en detalle cómo he programado esta aplicación, es importante mencionar las herramientas de las que he tenido que disponer. Además he visto conveniente comentar el motivo por el que he decidido utilizarlas frente a otras de naturaleza similar.

A continuación explico en qué consisten cada una de ellas.

6.1.1 Composer

He decidido hacer uso de este gestor de paquetes y dependencias para PHP frente a otros como PEAR o PECL ya que proporciona un mejor acceso a todas las dependencias permitiendo listarlas e instalarlas con mayor facilidad. Para comenzar con el desarrollo del sistema web procedo a usar el gestor de paquetes para PHP *Composer* el cual utilizo para instalar los siguientes elementos:

- Framework *Symfony.* Proporciona facilidades para crear la estructura de directorios que se va a manejar, en este caso crea una estructura del tipo *Modelo-Vista-Controlador.* También incorpora una serie de comandos para diversas opciones de funcionamiento, como crear las clases controladoras, hacer migraciones o arrancar el propio sitio web entre otras muchas funciones. Está hecho para PHP y lo he considerado mejor que otros

frameworks que manejan otros lenguajes ya que tengo bastante experiencia con él y creo que sería idóneo sacarle el máximo provecho para este proyecto.

- Motor de plantillas *Twig*. Proporciona un motor de plantillas para PHP que facilita el diseño de la vista de las páginas. Existen otros motores de plantillas con sintaxis similar como Jinja, no obstante, este último está hecho para trabajar con lenguaje Python y no sería válido para este caso en el que se utiliza PHP. Además, Symfony incluye suporte para Twig, por lo que ahorra la búsqueda de otros motores de plantilla.

6.1.2 Tailwind

Fuera de *Composer* procedo a instalar *Tailwind* que es un framework CSS que mejora la vista de la aplicación de cara al usuario teniendo acceso a diversas formas y colores que doten de un aspecto más ágil, profesional y cómodo con el que interactuar. Se usa sobre las etiquetas HTML de las páginas en las que se incrustan los comandos o palabras clave que reconoce esta herramienta, la mayoría de las veces en el atributo *class* de la etiqueta HTML en cuestión. A diferencia de otros como Bootstrap, considero que Tailwind da un estilo más natural sin que se note demasiado que se utiliza un framework para el estilo y considero además importante conocer nuevos frameworks de estilo sobre los que poder escoger en futuros proyectos.

6.1.3 Symfony

Para crear y levantar el servidor de desarrollo hago uso del framework *Symfony*, concretamente de su versión *5.2.6.* Éste monta desde el principio la estructura Modelo-Vista-Controlador y contiene una serie de comandos que ayudan y permiten ahorrar tiempo durante el desarrollo de la aplicación web como puede ser para la creación y gestión de archivos, creación de clases en PHP, migrar cambios a la base de datos, así como crear automáticamente los archivos básicos para los controladores junto con su correspondiente vista básica en HTML entre otras muchas más opciones.

Para poder ver la lista de comandos que ofrece Symfony junto con una breve descripción de lo que hace cada uno hay que ejecutar en la terminal desde la raíz del proyecto:

bin/console.

Para ejecutar alguno de los comandos de dicha lista escribimos

bin/console <comando>

Además gracias a Composer podemos añadir extensiones al framework según necesitemos realizar distintas gestiones como puede ser la autenticación de usuarios. Para ello se ejecuta desde la raíz del proyecto

composer add <extensión>.

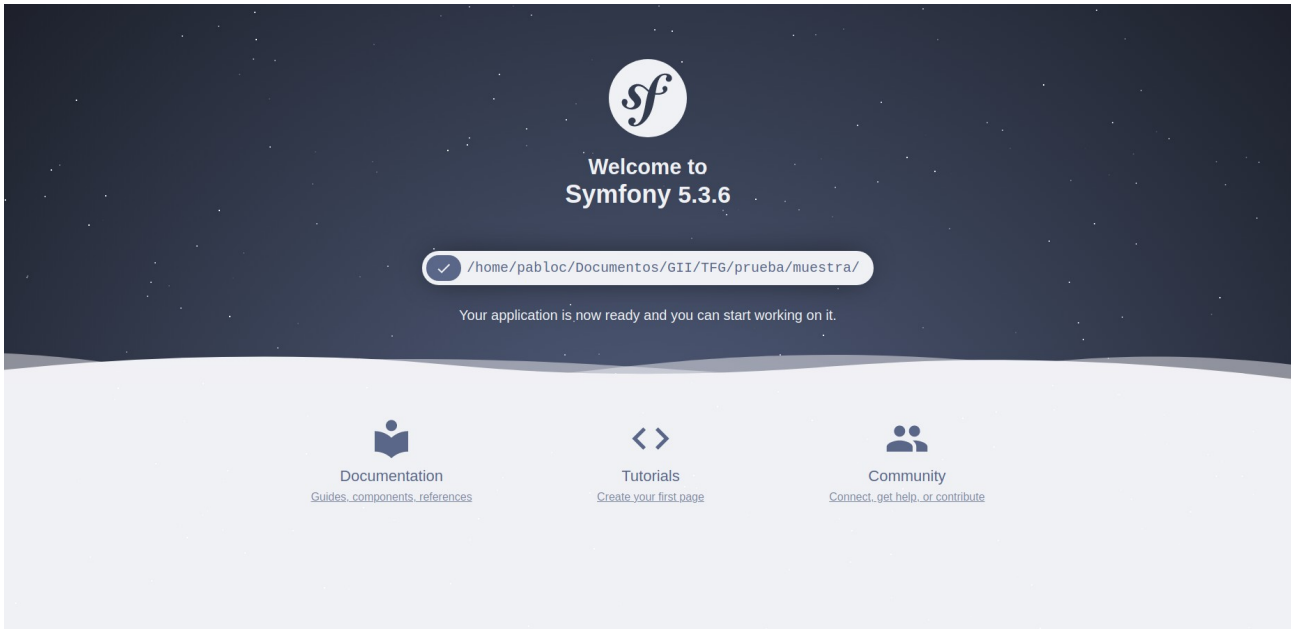
Una vez tengamos preparadas las herramientas a utilizar, debemos crear primero la estructura básica de archivos de esta aplicación web. Para ello hay que situarse en una carpeta cualquiera desde la terminal y ejecutar el siguiente comando:

symfony new –full <nombre del proyecto>.

De esta forma automáticamente se nos instalarán los paquetes básicos que necesitaremos para construir una aplicación web, se crearán la estructura de directorios y los archivos necesarios para poder arrancar el servidor. Una vez se haya instalado todo, nos ubicamos desde la terminal en el nuevo directorio que se ha creado, es decir, en la raíz del proyecto, ya estamos listos para levantar el servidor: basta con ejecutar

symfony server:start, o simplemente **symfony serve** (ambos son equivalentes).

Inmediatamente después se empezará a ver que en la terminal se están ejecutando las instrucciones internas correspondientes hasta que nos avisa de que podemos acceder al servidor desde la dirección 127.0.0.1:8000 desde un navegador. Dirigiéndonos a esa dirección contemplamos la pantalla por defecto que ha creado Symfony de un aspecto similar a la de la siguiente imagen que indica que, efectivamente, el servidor básico está en correcto funcionamiento.



He utilizado una misma plantilla para las secciones de Inicio, Contacto y Login y he usado otra distinta para sección en la que se buscarán y gestionarán los mapas y gráficos que utilicemos.

Symfony contiene utilidades para la creación de usuarios. Por medio de

php bin/console make:user

crea las clases User.php y UserRepository.php. Dentro de User podemos ver que contiene los datos miembro **id, email, roles** y **password** que serán exactamente las columnas de la tabla Users que estará en la base de datos mysql. Además de esto, Symfony permite crear usuarios especificando email y contraseña.

Para la autenticación de usuarios Symfony proporciona el comando

bin/console make:auth

el cual crea las rutas /login y /logout, el controlador *SecurityController.php* y el controlador *LoginFormAuthenticator.php*.

[https://symfony.com/doc/5.2//index.html]

6.1.4 Doctrine

Es un conjunto de librerías para PHP que proporcionan funcionalidad y persistencia a la aplicación que se está desarrollando. Está basado en ORM (*Object-relational mapping*) el cual permite convertir datos entre sistemas incompatibles usando la orientación a objetos. Viene incorporado con Symfony y permite ejecutar una amplia variedad de funciones, entre las que considero más importantes o destacadas son las que permiten interactuar con las migraciones, crear o eliminar una base de datos y ejecutar sentencias SQL. Para poder contemplar todas ellas basta con ejecutar desde la terminal en la raíz del proyecto el comando

bin/console

mostrándonos toda la lista de comandos incluyendo también los que no pertenecen a Doctrine. Permite además usar el objeto entityManager en el código PHP con el que poder preparar y ejecutar sentencias SQL.

6.1.5 Git

Para tener un buen control de versiones además de una copia de seguridad que me permita continuar en caso de problemas con el equipo, utilizo el sistema de control de versiones distribuido Git. Al ser una herramienta que he utilizado en numerosas ocasiones he preferido hacer uso de ella frente a otras de naturaleza similar que también habría podido considerar, sin embargo, sabiendo utilizar ésta me ahorro un tiempo valioso en aprender a manejar otra nueva.

Comienzo creando un repositorio de este programa desde la raíz del proyecto ejecutando

git init.

De esta forma permito que Git detecte cada cambio que se haya producido en cualquiera de los archivos y en las subcarpetas para que pueda actualizarse el repositorio cada vez que ejecute y suba un commit. Es importante destacar que gracias a esta herramienta, ante cualquier actualización problemática siempre puedo volver a anteriores versiones para solucionar los problemas. Además junto con Git hago uso de la página Github la cual muestra mis repositorios con una interfaz gráfica a través de la que el cliente puede navegar fácilmente y estar atento a los cambios que voy añadiendo con cada commit.

El repositorio para este trabajo se encuentra en el siguiente enlace: https://github.com/pcerezo/tfgApp

6.1.6 Lenguaje de programación

Para desarrollar este sitio web he utilizado el lenguaje PHP que es con el que trabaja Symfony, pero además he tenido una buena experiencia con éste en anteriores asignaturas como en Tecnologías Web, por lo que considero que es una buena idea utilizarlo además de su facilidad para declarar variables y crear clases que sigan el patrón Modelo-Vista-Controlador dentro del paradigma de orientación a objetos.

6.2 Estructura de directorios de Symfony

Una vez creado un nuevo proyecto con Symfony disponemos de las siguientes carpetas:

- **config**: como su propio nombre indica, contiene todos los archivos de configuración en los que indicamos las rutas, los servicios y los paquetes.
- **src**: es donde se aloja todo el código PHP. Destacan las siguientes subcarpetas:
  - → Controller: clases controladoras
  - → Entity: son las clases que dan forma a los datos que se guardan y crean las tablas de la base de datos el modelo
  - → Form: son los formularios que se generan mediante los comandos de Symfony.
- **templates**: contiene todas las plantillas html.twig que forman la vista de la aplicación.
- **public**: esta carpeta aloja archivos accesibles, por lo que la gran mayoría de las ocasiones funcionará como la raíz del documento desde la que parten las rutas de los ficheros que se descargan o acceden.

Éstos son los principales directorios a los que se acceden durante el desarrollo de la aplicación. No obstante, también es importante tener en cuenta el resto de directorios ya que permiten el correcto funcionamiento del sistema:

- **bin**: contiene ficheros ejecutables, entre ellos está el famoso archivo *bin/console* ya mencionado anteriormente que proporciona una lista de comandos útiles para el desarrollo del proyecto.
- **var**: aloja principalmente archivos de caché y logs.
- **vendor**: almacena las librerías que se instalan desde Composer.
- **migrations:** contiene todos los cambios en la base de datos. Cada vez que se crea o modifica un archivo de Entity y se prepara la migración se anotan todas estas modificaciones en un fichero dentro de este directorio. Posteriormente cuando se ejecuta la migración se aplican todos los cambios necesarios en la base de datos.

[https://symfony.com/doc/5.2/page_creation.html]

A continuación explico en los siguientes apartados cómo he procedido a implementar cada una de las funcionalidades que he ido añadiendo al proyecto, el funcionamiento de las clases programadas y los comandos.

6.3 Creación de las secciones básicas

En esta iteración se generan simplemente las secciones fundamentales del proyecto creando las clases que correspondan e incorporando las plantillas que den una estética lo más profesional posible.

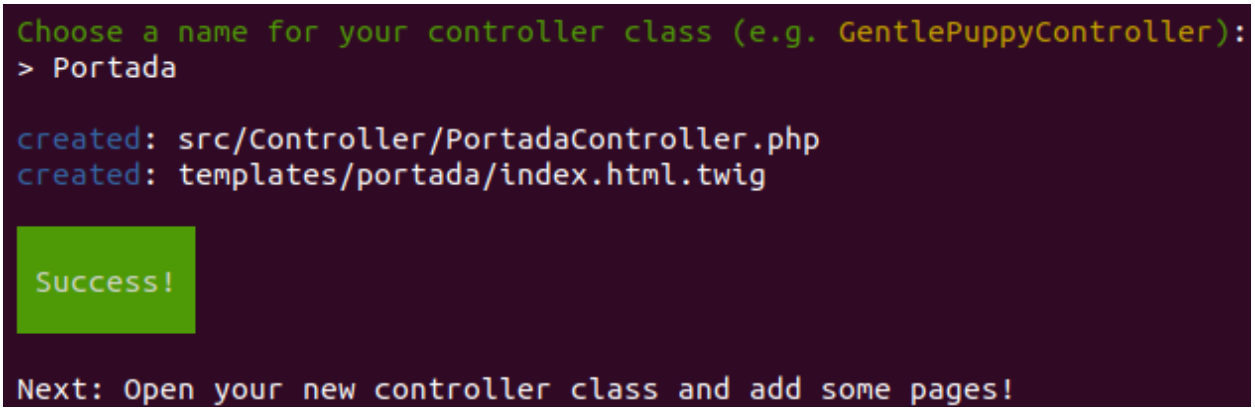
6.3.1 Sección de inicio

Llegados a este punto la aplicación está lista para ser programada y es por ello que comenzaríamos creando la página inicial. En Symfony la renderización de cada página debe hacerse desde una clase controladora, por lo que gracias a los comandos que nos facilita este framework ejecutamos lo siguiente

bin/console make:controller.

Nos pregunta cómo queremos llamar a la nueva clase controladora, en este caso le doy como nombre “Portada” y de esta forma Symfony nos crea automáticamente la clase controladora

añadiéndole el sufijo “Controller”, llamándose así *PortadaController,* y en una subcarpeta de templates cuyo nombre es *Portada* tenemos la plantilla de nombre *index.html.twig* que da la vista a esta sección.



Dentro de cada clase controladora se crean los distintos métodos que realizan las operaciones pertinentes, por defecto se crea inicialmente el método index(). Al finalizar el método se retorna la renderización de la vista indicando la ruta del fichero html que será uno de los que se guardan en */templates* enviándole las variables que necesite utilizar para que sean detectadas por el lenguaje de plantillas Twig.

En un primer momento, el funcionamiento de index() de PortadaController consiste en devolver una función que renderiza el fichero *index.html.twig* indicando su ruta y las variables que serán leídas por el lenguaje Twig.

```
return $this->render('portada/index.html.twig', [
     'controller_name' => 'PortadaController',
     'activeInicio' => 'active',
     'activeContacto' => '',
     'activeLogin' => '',
]);
```

En este caso los valores que comienzan por active es utilizado por la plantilla base para darle valor al atributo class de cada elemento de la barra de navegación.

Con Twig es muy recomendable crear una plantilla base que forme el esqueleto básico e indique los distintos bloques que se añadirán sobre ésta cada página.

Para darle un aspecto más profesional al programa he decidido disponer de algunas plantillas para sitios web que he encontrado en internet. En el caso de la página inicial he utilizado el código HTML de una plantilla ya creada sobre la que he colocado las siguientes partes básicas:

a) *header*. Cabecera de la página con el título y barra de navegación.
b) *page*. Cuerpo de la página que contiene toda la información.
c) *footer*. Parte inferior con información adicional.

Dentro de estas partes básicas indico los distintos bloques de HTML con la sintaxis de Twig que se añadirán al renderizar las distintas páginas que usen esta plantilla. Son:

- **title** para indicar el nombre en la pestaña del navegador. Puede cambiar si se usa la misma plantilla para más páginas.
- **link_meta**. Se indican los distintos enlaces y metadatos.
- **banner**. Usado para poner una fotografía principal ilustrativa.
- **resumen_login**. Muestra un resumen de la información del usuario si ha iniciado sesión.
- **main_container**. Contiene la información principal de la página.
- **footer**. Información adicional.

Dado que el archivo que mostrará la página inicial de forma completa es *index.html.twig*, hago que extienda el esquema de *base.html.twig* escribiendo al comienzo

```
{% extends 'portada/base.html.twig' %}
```

y ahora sólo hay que construir los bloques anteriormente señalados. De esta forma, evitamos repetir el mismo esqueleto para cada página o sección que se añada y sólo nos preocupamos de crear cada bloque de los que se añaden.

No debemos olvidar que para poder observar cada página en el navegador debemos registrar la ruta que se debe poner en la barra de búsqueda en el archivo de configuración *routes.yaml* escribiendo un nombre y con él la url en el campo *path* y la clase controladora seguida del método manejador en el campo *controller*. En este caso añadiríamos a *routes.yaml* lo siguiente:

```
index:
    path: /
    controller: App\Controller\PortadaController::index
```

6.3.2 Sección de mediciones y datos

Para esta parte he decidido utilizar otra plantilla HTML más enfocada para mostrar datos ya que en esta sección se mostrarán las observaciones y el mapa o mapas con los que tratemos si los hubiera. Se comienza de la misma manera que la anterior sección, creamos mediante Symfony una nueva clase controladora que nos cree además el fichero de vista. He decidido ponerle de nombre Buscador, por lo que como resultado se obtiene */src/BuscadorController.php* y */templates/buscador/index.html.twig*. Análogamente creo un fichero en la misma carpeta (*/templates/buscador*) para formar la estructura básica. A este fichero que le llamo también *base.html.twig*, le que incorporo las etiquetas HTML de la plantilla externa recientemente mencionada y los bloques de código que me preocuparé de implementar en el archivo index. Estos bloques son:

- Title.
- Titular. Muestra el nombre de la sección. Puede variar ya que para la misma plantilla podemos añadir más páginas.
- Contenido. Es donde se mostrará la lista de mediciones y toda la información relacionada.
- Sidebar. Es una barra de navegación vertical dentro del cuerpo de la página que permite acceder a otra página que utilice esta misma plantilla base. De todos sus elementos se mantiene iluminado aquél en cuya ruta estamos.

Dado que se van a usar distintos archivos HTML para esta sección, decido eliminar el nombre de *index* y cambiarlo por *mediciones_fotos* para que sea más explicativo.

Seguidamente comienzo a implementar la clase *BuscadorController.php*. Creo un método llamado mediciones_fotos() en el que simplemente retorno la renderización del archivo de mismo nombre sin enviarle más información ya que no es necesario por el momento. De esta forma el método queda como

```
public function mediciones_fotos(): Response {
     return   $this→render('buscador/mediciones_fotos.html.twig',
[
          'controller_name' => 'BuscadorController',
     ]);
}
```

Por supuesto, añadimos una nueva ruta en el archivo *routes.yaml*:

```
buscador_mediciones:
     path: /busqueda/mediciones_fotos
     controller: App\Controller\BuscadorController::mediciones_fotos
```

6.3.3 Contacto

La sección de contacto es sencilla, puesto que es una página informativa podría bastar con que fuera estática. Procediendo de la misma forma, para facilitar las cosas por si en el futuro se decidiera añadir algo más, se crea mediante nuestro framework la clase controladora *ContactoController* y su HTML dejando el nombre que viene por defecto puesto que no se añadirán más vista para esta parte.

En *ContactoController* nos preocupamos de momento en devolver la renderización de la vista en su único método indicando la ruta al igual que en la anterior sección:

```
return $this->render('contacto/index.html.twig', [
     'controller_name' => 'PortadaController',
     'activeInicio' => '',
     'activeContacto' => 'active',
     'activeLogin' => '',
]);
```

En cuanto a la vista he decidido usar la misma plantilla que la portada, por lo que en su archivo de vista añado también `{% extends 'portada/base.html.twig' %}` e implemento los bloques correspondientes adaptándolos a lo que se vaya a mostrar.

6.3.4 Login

En cuanto al Login, el procedimiento es algo distintos a los anteriores. Para esto Symfony nos proporciona un comando específico que nos crea la clase que genera el formulario en *src/Security/LoginFormAuthenticator.php,* se actualiza el archivo de configuración en *config/packages/security.yaml* y por supuesto, la clase controladora *SecurityController.php y* el fichero de la vista *login.html.twig*.

La clase *SecurityController.php* costa de un par de funciones:

- **`login(AuthenticationUtils $authenticationUtils): Response`**. Tal y como su propio nombre indica, es la función que permite iniciar sesión al usuario. De momento sólo renderiza la página *login.html.twig.*
- **`logout()`**. Puede estar en blanco ya que la llamada a este método es interceptada y cierra la sesión actual.
- **`register()`**. Carga la página que permite registrar un nuevo usuario. Esta acción sólo estará permitida para el usuario administrador.

Si nos vamos al archivo */config/security.yaml* vemos que en el apartado firewalls tenemos una serie de secciones dentro de main entre la que encontramos logout. Para indicar hacia dónde nos debe redirigir el sitio web una vez se cierre sesión debemos indicar el nombre de la ruta correspondiente (de las que se guardan en *routes.yaml*). Para ello, basta con añadir

```
target: index
```

para que la aplicación nos lleve a la página de inicio.

Para que cualquier usuario se pueda autentificar es necesario un formulario que le facilite introducir los datos, es por ello que Symfony nos ha creado automáticamente el fichero *LoginFormAuthenticator.php.* Contiene 4 atributos:

1. **entityManager**. Objeto que podremos usar para preparar y ejecutar sentencias SQL.
2. **urlGenerator**. Tal y como indica, proporciona la ruta para el login.
3. **csrfTokenManager**. Es el manejador de un objeto CSRF que son las siglas de *Cross-site request forgery*. Permite a los usuarios de la plataforma realizar operaciones seguras cada vez que utilicen algún formulario ya que va oculto en ellos. De esta forma, sólo el usuario logueado conoce su valor y no puede ser conocido por ningún otro intruso y asegura que sea el propio usuario y no un impostor quien envíe los datos.

[https://symfony.com/doc/5.2/security/csrf.html]

Además destacan los siguientes métodos:

1. **`getCredentials(Request $request)`**. Lee los campos del formulario de login tras pulsar el botón de enviar y los devuelve en un vector.
2. **`getUser($credentials, UserProviderInterface $userProvider)`**. Devuelve el objeto usuario si la autenticación ha sido exitosa. En otro caso lanza una excepción.
3. **`checkCredentials($credentials, UserInterface $user)`**. Comprueba que la contraseña es válida.
4. **`onAuthenticationSuccess(Request $request, TokenInterface $token, string $providerKey)`**. Permite redirigirnos a una ruta concreta una

vez el usuario se haya autentificado con éxito. Por defecto nos lleva a la página inicial y lo mantendré así.

Para la vista de esta sección he decidido emplear la misma plantilla base que en la portada, por lo que hago que extienda de ésta y sólo necesito rellenar los bloques *title* y *main_container* incorporando en este último el formulario de inicio de sesión.

6.4 Creación de usuario y autenticación

En esta parte se define la base de datos del usuario para poder almacenar sus datos, se permitirá su autenticación, la encriptación de la contraseña y, por supuesto, iniciar sesión al primer usuario de la aplicación que será el administrador.

6.4.1 Base de datos

Antes de configurar y crear la base de datos es necesario instalar ciertos componentes de Composer que nos permita conectar con el gestor de base de datos. Para ello usando la terminal hay que ejecutar los siguientes comandos:

```
composer require symfony/orm-pack
composer require --dev symfony/maker-bundle
```

Una vez ya instalado mysql, tenemos que establecer en Symfony la dirección de nuestra base de datos, su puerto, usuario y contraseña. Para ello editamos el archivo oculto *.env* que está en la raíz del proyecto y le doy valor a la variable *DATABASE_URL* de la siguiente forma:

```
DATABASE_URL="mysql://<usuario>:<contraseña>@0.0.0.0:3306/app_db"
```

Una vez que ya está ajustada esta variable, creamos la base de datos para el proyecto a través de Symfony. Para ello se ejecuta en la terminal lo siguiente:

```
php bin/console doctrine:database:create
```

En este caso se crea una base de datos de nombre *app_db*.

6.4.2 Usuario

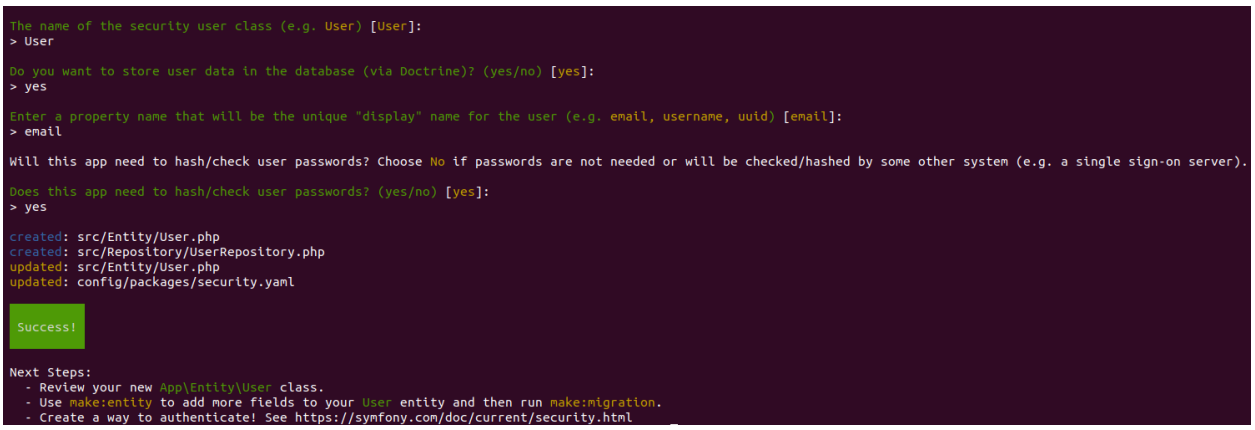
Ahora que ya está hecha, estamos en condiciones de crear las distintas tablas que necesitaremos. Comenzamos primero por la creación del usuario. Symfony tiene también una funcionalidad para ello, pero antes debemos instalar de nuevo un componente de seguridad de Composer con el siguiente comando:

```
composer require symfony/security-bundle
```

Después podemos usar ejecutar la funcionalidad que nos crea el usuario:

```
php bin/console make:user
```

Seguidamente respondemos a una serie de preguntas que formarán la tabla del usuario dependiendo de cómo nos interese que sea, en mi caso es:



Acto seguido preparo la migración a la base de datos y la ejecuto para aplicar los nuevos cambios. Se realiza de la siguiente manera:

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

Finalmente queda indicar en el archivo de configuración *config/packages/security.yaml* el algoritmo de hasheo de las contraseñas que es *bcrypt*.

```
security:
    encoders:
        App\Entity\User:
            algorithm: bcrypt
```

Dado que en esta aplicación no se puede dar de alta a cualquier usuario sino que debe tener el visto bueno del administrador, introduzco manualmente por ser el primer usuario los datos del administrador en la base de datos (incluyendo la contraseña hasheada) rellenando todos los campos.

Además de los típicos campos id, email, password y roles que nos crea por defecto nuestro framework, he visto conveniente añadirle los siguientes:

- **nick**. Es el nombre de usuario que los demás usuarios verán principalmente.
- **nombrecompleto**. En la información detallada se mostrará por si hubiera alguna duda de quién sea el usuario.
- **foto_perfil**. Es el nombre de la foto de perfil que suba el usuario y que se almacenará en el sistema de archivos.

```
mysql> describe user;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| id             | int          | NO   | PRI | NULL    | auto_increment |
| email          | varchar(180) | NO   | UNI | NULL    |                |
| roles          | json         | NO   |     | NULL    |                |
| password       | varchar(255) | NO   |     | NULL    |                |
| nick           | varchar(255) | NO   |     | NULL    |                |
| nombrecompleto | varchar(255) | NO   |     | NULL    |                |
| foto_perfil    | varchar(255) | NO   |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
7 rows in set (0,01 sec)

mysql>
```

```
mysql> select * from user;
+----+---------------------+-----------------+--------------------------------------------------------------+------+---------------+--------------------------------------+
| id | email               | roles           | password                                                     | nick | nombrecompleto | foto_perfil                          |
+----+---------------------+-----------------+--------------------------------------------------------------+------+---------------+--------------------------------------+
|  1 | correoadmin@gmail.com | {"rango": "admin"} | $2y$13$K1honx0pXCiA.DwFahz03ehwVKWaORp/VHROzVbNOXYtRv3zaVKu2 | Admin | Administrador | c4ca4238a0b923820dcc509a6f75849b.jpg |
+----+---------------------+-----------------+--------------------------------------------------------------+------+---------------+--------------------------------------+
1 row in set (0,00 sec)
```

Como se mostró anteriormente, Symfony ha creado varios archivos, uno de ellos es *src/Entity/User.php*. Este archivo pertenece a la capa de modelo dentro del patrón MVC que comenté en previos apartados. Los atributos de esta clase coinciden con los campos de la tabla del usuario (id, email, roles, password, nick, nombrecompleto, foto_perfil) y sus métodos son los set y get que permiten consultarlos y modificarlos.

Otra de las clases generadas es *src/Repository/UserRepository.php* la cual contiene

- *`__contruct(ManagerRegistry $registry)`* . Contructor de la clase.
- *`upgradePassword(UserInterface $user, string $newEncodedPassword)`*. Actualiza la contraseña actual.

6.4.3 Login

Una vez que tenemos la base de datos preparada y los datos del usuario correctamente cargados en su tabla comprobamos que el inicio de sesión es exitoso. Es muy importante añadir en el formulario de login el campo oculto del csrftoken de tipo *hidden*

```
<input type="hidden" name="_csrf_token"
value="{{ csrf_token('authenticate') }}">
```

No olvidemos que es necesario enviar el csrftoken que comentamos anteriormente para proporcionar seguridad al usuario.

Finalmente por medio de una plantilla externa para este formulario nos quedaría de la siguiente forma

6.5 Lectura y subida de archivos de medición

Este apartado marca la esencia de este proyecto puesto que es la iteración en la que permitimos a los usuarios subir los ficheros con los datos de iluminación del cielo nocturno. Principalmente incorporaremos código sobre la capa controladora, en concreto sobre la clase *BuscadorController*, creando nuevas funciones y ampliando las ya existentes que nos garantice este funcionamiento. También ampliaremos esta interacción entre humano y sistema formando la vista que mostrará la galería de mediciones y los formularios que necesitemos.

6.5.1 Archivos de medición

Para empezar debemos preguntarnos si las mediciones del cielo que se realicen serán siempre de la misma forma, si se usará el mismo aparato y por tanto, si usaremos algún formato. La respuesta para todo esto es afirmativa, por lo que se ha establecido un mismo formato de lectura de archivos para cada medición y así facilitar la programación de esta funcionalidad. Los archivos tendrán estos campos:

- **#.** Es el número de la medida. Comienza desde el 1 y va incrementándose fila a fila.
- **TASD00.** Es la fecha en la que se realiza la medida. Se tendrá en cuenta la fecha al inicio de la operación aunque durante la medición se sobrepasen las 23:59.
- **ci:20.48.** Indica la hora, minuto y segundo en el momento de la medida.

- **T IR**. Temperatura infrarroja, es decir, la temperatura que se capta del cielo en cada instante en el que se mide.

- **T Sens**. Análogamente, la temperatura que tiene el sensor en cada instante.

- **Mag**. Es el valor que nos indica la luminosidad del cielo en la dirección en la que se esté midiendo.

- **Hz**. Hercios.

- **Alt**. Es el ángulo de inclinación respecto a la horizontal en la que se está realizando la medida en cada preciso instante.

- **Azi**. Azimut. Es la inclinación respecto al norte magnético.

- **Lat**. Latitud geográfica.

- **Lon**. Longitud geográfica.

- **SL**. Altura sobre el nivel del mar en la que estamos.

- **Bat**. Presión atmosférica (?)

6.5.2 Subida de archivo

Una vez que ya tenemos bien aclarado el formato, el siguiente paso consiste en subirlo al sistema de archivos de la aplicación donde se almacenará y desde donde se leerá. Para ello debemos crear un nuevo formulario y existen dos formas de hacerlo: implementándolo en una clase controladora o creándolo en una clase a parte a través de los comandos que nos facilita Symfony. Me he decantado por esta última forma ya que considero que es más preferible separar el código según la funcionalidad que aporte, por lo que ejecuto el siguiente comando:

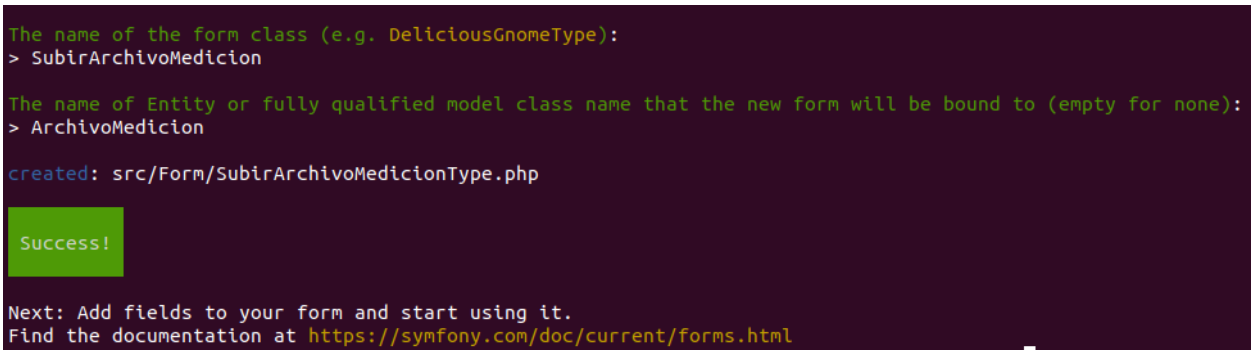
```
bin/console make:form
```

Al igual que con la creación de clases controladoras, nos hace unas preguntas en la que tenemos que indicar el nombre de esta nueva clase para el formulario y la clase entidad en la que está basada (si la hubiera). Para facilitar las cosas previamente creo una clase entidad llamada

ArchivoMedicion que servirá para almacenar los datos que se envíen del formulario en un objeto y poder manejarlos con mayor facilidad. Esta clase contiene los siguientes atributos:

- **id**. Se genera siempre por defecto para diferenciar con el resto de objetos.
- **nombre**. Es un string pero contendrá el nombre del archivo de medición que seleccionemos.
- **lugar**. El nombre de la localización o del municipio donde se ha realizado la medición.

Sin embargo, este no será el modelo de datos para las mediciones, sino que será algo más complejo tal y como se comentó en la sección de diseño. Cuando se programe correctamente la lectura del formato de los datos será entonces cuando se proceda a crear las tablas para su almacenamiento. En la siguiente captura se aprecia este rápido proceso:



y así es como se presenta la clase:

```
class SubirArchivoMedicionType extends AbstractType
{
public function buildForm(FormBuilderInterface $builder, array $options)
{
      $builder
            ->add('nombre')
            ->add('lugar')
      ;

}

public function configureOptions(OptionsResolver $resolver)
{
      $resolver->setDefaults([
      // Configure your form options here
      ]);
}
}
```

Como se puede apreciar, se añaden como campos los atributos de esa clase entidad auxiliar que acabo de crear, por lo que los modificamos indicando los tipos y añadimos un botón para el envío de datos. La función buildForm queda de esta manera.

```
$builder
      ->add('nombre', FileType::class, array('label' => 'formato .txt',
      'data_class' => null))
      ->add('lugar', TextType::class)
      ->add('subir', SubmitType::class, array('label' => 'Subir'))
      ->getForm();
;
```

Se puede indicar el tipo para cada campo e incluso añadir etiquetas en las que incluir alguna información, como por ejemplo que los archivos que se suban han de ser de extensión *.txt*.

Posteriormente en el método mediciones_fotos() de BuscadorController hacemos que se muestre este formulario únicamente a quien esté registrado en el sistema. Primero creamos un objeto de ArchivoMedicion y lo pasamos a la función *createForm*() cuyos parámetros son SubirArchivoMedicionType::class y dicho objeto. Una vez que el usuario haya rellenado los campos y haya pulsado el botón de Subir, esos datos se almacenan en ese objeto de tipo ArchivoMedicion() y de él extraemos todos los datos necesarios gracias a sus funciones de consulta *get* que se implementan por defecto.

Previamente a la programación de la subida de archivos al sistema, se establece un parámetro global que indica la ruta de la carpeta a la que se subirán los archivos. Este parámetro está definido en *services.yaml* llamado “directorio_mediciones” y se hace uso de él con cada operación en la que el usuario introduzca un nuevo fichero.

Se realizan las siguientes tareas:

1. Se obtiene el archivo enviado en el formulario.
2. Se comprueba su extensión. Si es *.txt* se procede a subir el archivo poniéndole antes como título `<lugar>_<id>` y finalmente la extensión.
3. Se genera el nombre de la carpeta destino cuyo nombre es de la misma forma que el archivo: `<lugar>_<id>`. Si esa carpeta no existe, se crea.
4. Finalmente se sube a dicho directorio.

6.5.3 Creación del modelo de datos de las mediciones

Tal y como se comentó en la sección de diseño, los datos que se van a analizar estarán separados en dos tablas según sean los datos de las condiciones geográficas del lugar y las de la contaminación lumínica. Por lo tanto, serán necesarios dos archivos entidad que estén enlazados y se usarán los comandos de Symfony para ello.

Se considerarán como datos genéricos a los siguientes:

- id
- latitud
- longitud
- localización
- altitud
- bat

y éstos serán los datos individuales:

- id
- id medición genérica
- fecha
- hora
- nombre del archivo
- declinacion
- azimut
- magnitud
- temperatura infrarroja
- temperatura del sensor
- autoría

Como se puede apreciar, estas tablas tienen una relación de 1 a muchos ya que para una misma zona geográfica se pueden realizar diversas mediciones en distintos días y en distintas horas en un mismo día.

Se ejecuta el siguiente comando en la terminal desde la raíz del proyecto:

`bin/console make:entity`

Nos vuelve a hacer preguntas respecto a cada dato y su tipo como se muestra en la imagen e indicamos los datos genéricos recientemente citados.

```
Add another property? Enter the property name (or press <return> to stop adding fields):
> longitud

Field type (enter ? to see all types) [string]:
> float

Can this field be null in the database (nullable) (yes/no) [no]:
> no

updated: src/Entity/MedicionGenerica.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> localizacion

Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 255

Can this field be null in the database (nullable) (yes/no) [no]:
> no

updated: src/Entity/MedicionGenerica.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> altitud
```

De nuevo volvemos a ejecutar el mismo comando haciendo lo mismo para la clase entidad MedicionIndividual. Nótese que para el id de la medición genérica en la medición individual debe hacerse usando el tipo de dato *ManyToOne* que proporciona Symfony.

Para terminar bastaría con crear la migración y ejecutarla para dejar constancia en la base de datos de las nuevas tablas y a partir de ahora poder almacenar información en ellas.

6.5.4 Lectura de los archivos

Teniendo el archivo guardado en el sistema se comienza a implementar la lectura del mismo para obtener sus datos. Primero se obtiene el descriptor del archivo a leer por medio de la función *file*() pasándole la ruta como parámetro. Justo después se procede a leer línea por línea distinguiendo cada campo separado por tabulador gracias a *explode*() y los guardamos en un array llamado *registros* del que extraemos cada dato. Algunos de estos datos como la fecha o la hora no tiene sentido leerlos más de una vez. Sin embargo otros como las temperaturas sí es más preferible realizar medias por si existiera cierto margen de error en la percepción.

Dado que tenemos un modelo de datos para las mediciones separados en dos tablas, separo la lectura también en dos partes y creo los objetos entidad *medicionGenerica* y *medicionIndividual*.

Una vez obtenidos los datos más genéricos y almacenados en variables, los inserto en el objeto *medicionGenerica* por medio de sus métodos *set* y seguidamente lo inserto en la base de datos gracias al objeto *entityManager* que interactúa con ésta invocando los métodos *persist,* que prepara al objeto para ser almacenado, y *flush* que ejecuta las sentencias SQL para guardar todos los objetos preparados.

De la misma forma lo hacemos para los datos más específicos del cielo leyendo desde la segunda línea hasta la última en bucle e introduciendo en cada iteración los datos leídos en el objeto *medicionIndividual* que después se pasa a la tabla de la medición individual.

6.6 Modificación de datos del usuario

Una parte importante de un sistema web es que sus usuarios puedan acceder a una interfaz que les permita consultar y alterar sus datos. Es por ello que el usuario una vez logueado debe tener la opción de acceder a su perfil, observar sus datos y pulsar algún botón o enlace que le lleve a algún formulario en el que edite los campos. De la misma forma que hemos creado anteriormente los otros formularios, creamos ahora el que nos permita hacer todo esto.

Lo primero de todo sería crear la página del perfil de usuario. Para ello creamos la vista HTML extendiendo alguna plantilla de las ya existentes, por ejemplo decidí hacerla sobre el esquema básico de la portada. En esta vista únicamente he considerado implementar los bloques de etiquetas *title*, en el que pongo de título a la página “Mi Perfil”, y *main_container* donde por medio del lenguaje de plantillas muestro los datos de la persona y condicionalmente muestro el botón que nos lleve al formulario o muestro el propio formulario porque se haya pulsado anteriormente dicho botón.

6.6.1 Formularios

La forma que nos permite Twig para acceder a un formulario o a otro cómodamente es la siguiente sentencia:

```
{{ form(<nombre formulario>) }}
```

De modo que ahora hay que crear estos formularios que se van a mostrar y se procede de la misma forma que en los anteriores con el comando

`bin/console make:form`.

El primer formulario que se encuentre el usuario será aquel que lo lleve a la página de edición. Este formulario será muy sencillo ya que bastará con ser un botón llamado “Editar” que envíe una petición al método manejador de la clase controladora.

Posteriormente creo el otro formulario que falta, que es el que permite editar los campos de los datos. A éste lo he llamado *PerfilFormType* y en él de momento se permitirá modificar la foto de perfil y enviarla. Más adelante en futuras mejoras se le añadirán el resto de campos.

6.6.2 UserController

Para poder interactuar y acceder a los formularios es necesario programar la clase controladora *UserController* para las operaciones relacionadas con el usuario. Creamos este archivo con el mismo comando con el que se crearon las anteriores clases controladoras que además nos generará la carpeta *templates/user* donde se alojará su correspondiente HTML. Como siempre, dentro de cualquier clase controller tenemos por defecto el método *index*() el cual se usará para renderizar dicha página de usuario y realizará las siguientes tareas.

1. Se comienza obteniendo los datos del usuario una vez haya iniciado sesión. Después se accede al archivo de su biografía que está contenida en la carpeta del usuario del sistema de archivos. Cada usuario tiene un directorio cuya ruta es

   `public/uploads/<id usuario>/`

   en la cual se suben todos los archivos relacionados con él. En el caso de la biografía se crea ahí el fichero **`bio.txt`** si no existiera.

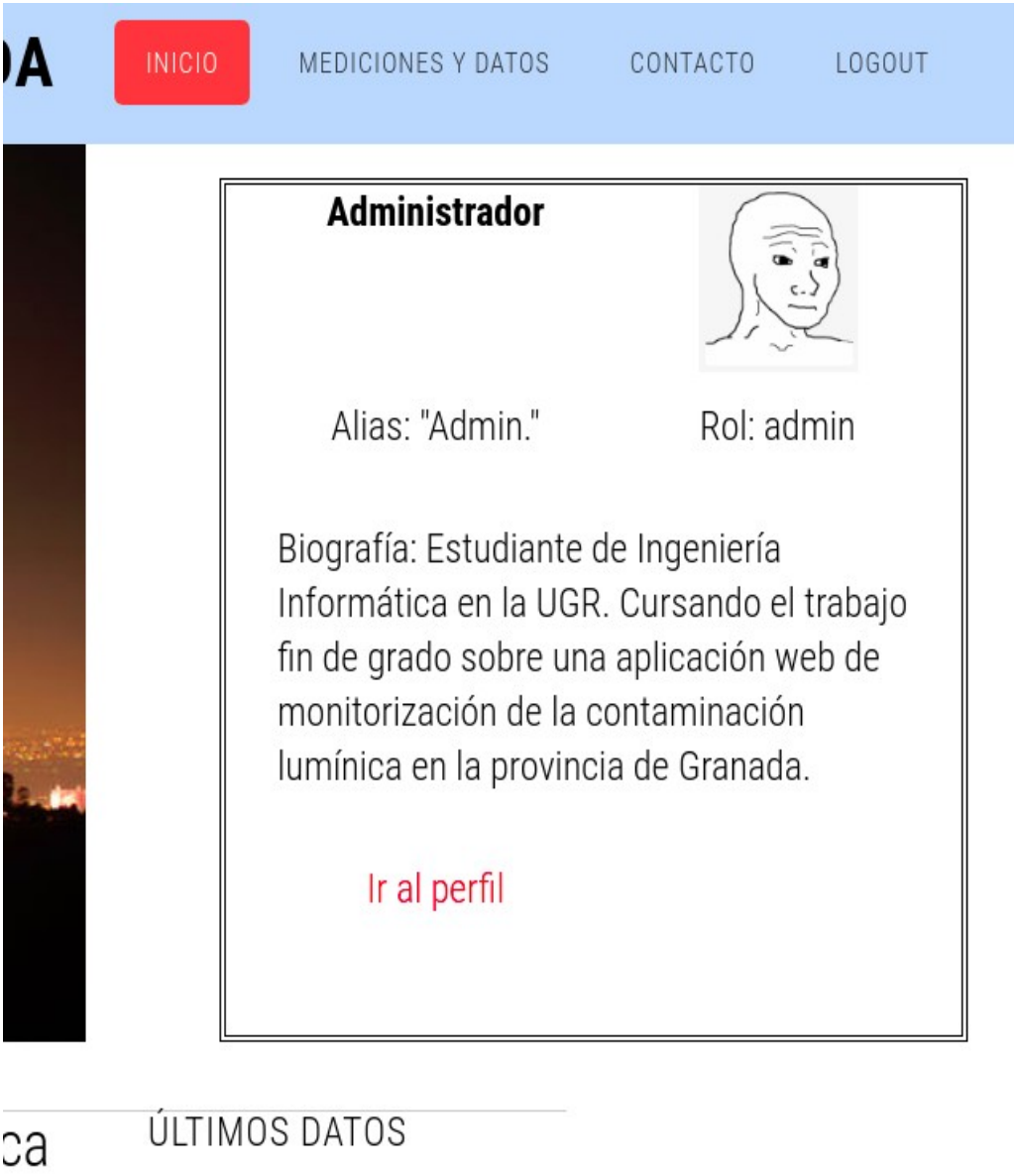
2. Se crean variables a las que se asignan mediante la función *createForm* los formularios con los que nos encontraremos.

3. Se detecta si se ha pulsado el botón enviar de alguno de ellos.
   a. Si es el caso del formulario *BotonEditarPerfilType* entonces le damos el valor **true** a la variable editar para indicar en la renderización que se muestre el formulario de edición.
   b. Si es el caso del formulario *PerfilFormType*, obtenemos la foto de perfil seleccionada y la subimos a su carpeta correspondiente. Finalmente volvemos a la página de Inicio.
   c. Si no se ha enviado datos por ningún formulario entonces simplemente se renderiza la página que muestra el perfil del usuario enviando sus datos y los formularios almacenados en las variables. El fichero HTML se encarga de seleccionar qué formulario se muestra en cada momento.

6.6.3 Acceso a perfil desde Inicio

Finalmente, ya creada la página del perfil, es necesario tener alguna forma de acceder a ella desde alguna de las secciones principales cuando se inicie sesión. Es por ello que señalo en la

plantilla base mediante el lenguaje de Twig un bloque llamado `resumen_login` que luego en el fichero de Inicio se le añaden las etiquetas y datos proporcionando un resumen de los datos del usuario mostrando su foto, nick y biografía tal y como se muestra en la siguiente fotografía. Justo debajo introduzco un enlace que invita a entrar en el perfil que será la página que se acaba de crear.



6.7 Galería de mediciones con página de detalles

Para esta funcionalidad se ha ampliado el comportamiento del método *mediciones_fotos* de la clase *BuscadorController* explicado anteriormente. Se obtendrán todas las mediciones almacenadas en la base de datos por medio del objeto *entityManager* que conecta con la base de datos ejecutando las siguientes sentencias de consulta que nos devolverá los datos en un array.

```
// Obtención del manejador y conexión con la BD
$entityManager = $this->getDoctrine()->getManager();
$conn = $entityManager->getConnection();
```

```
$sql = 'SELECT * FROM medicion_generica ORDER BY medicion_generica.fecha';
$sentencia = $conn->prepare($sql);
$sentencia→execute();

$datos = $sentencia->fetchAll();
```

Este array lo pasamos a la lista de objetos que se envía en la función *render* que carga la vista de la página. A estos elementos se acceden a través del lenguaje de plantillas. Gracias a este lenguaje podemos mostrar los datos por grupos en iteraciones de bucle for de la siguiente manera.

```
{% for row in datos %}
<div ...>
     {{ row.localizacion }}
     {{ row.fecha }}
     {{ row.autoria }}
     .
     .
     .
</div>
{% endfor %}
```

Esto facilita mucho crear agrupaciones separadas para cada medición indicando sus datos más genéricos a modo de resumen como la localización, la fecha, la hora y la autoría. Debajo de esa información sitúo además un enlace que contiene el id de esta medición y nos lleva a la página que nos muestra los detalles para dicho objeto contiene dicho identificador.

Al igual que en el método *mediciones_fotos* se obtenía información de la base de datos, en el método *detalles* se actúa de la misma manera mostrando el resto de datos genéricos cargando un nuevo fichero llamado *detalles.html.twig* en la carpeta de las vistas para BuscadorController. En la siguiente iteración de la implementación de este proyecto se ampliará el uso de

6.8 Ejecución del script y muestra de la información de Meteoblue

Cuando se muestran los detalles de cada medición, unos datos que se deberían tener en cuenta, además de la información genérica que se ofrece, es la información respecto a la calidad del cielo. Esta información se podría mostrar con números mediante una simple consulta a la base de datos; no obstante, es difícil de apreciar para cualquier humano la calidad del cielo si sólo se muestran números y no alguna referencia gráfica que facilite su interpretación. Por ello, en este apartado se van a emplear imágenes gráficas respectivas a la luminosidad y datos de la página de meteorología Meteoblue.

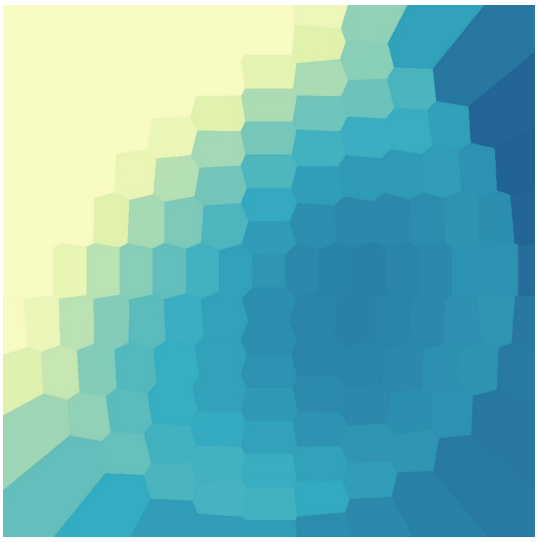
6.8.1 El script *interpolador.py*

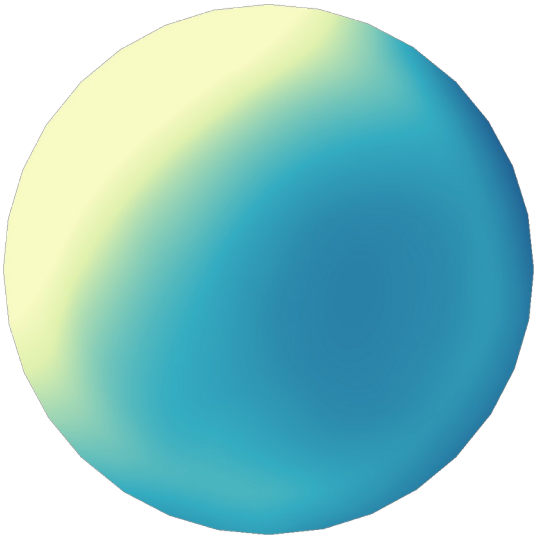
Para mostrar una aproximación gráfica del cielo nocturno hago uso de un script externo escrito en Python llamado *interpolador.py* el cual lee estos datos del archivo de medición y le asigna un color más claro o más oscuro en función del valor de la magnitud (es decir, su luminosidad), lo sitúa más en el centro de la imagen o más en la periferia en función de su declinación y lo posiciona a un lado o a otro según su azimut (es decir, ángulo respecto al norte). Con esto obtenemos una imagen muy ilustrativa de la contaminación lumínica que se entenderá mucho mejor a primera vista que los simples datos numéricos.

El script se ejecuta con el siguiente comando:

```
python3 [ruta interpolador.py] [archivoMedición] [archivoSalida.png] [opción]
```

Donde opción es un parámetro que puede estar o no. Si está y vale 1, suaviza los colores haciendo una media con los valores de su alrededor: De esta forma se imita mucho más a una fotografía del cielo nocturno pues de lo contrario las distintas partes de la imagen aparecerían cada una con un color plano distinguiéndose demasiado las distintas zonas. En este ejemplo de gráfico se aprecian estas diferencias:





En estos gráficos relativos al cielo del Canal de la Espartera, es la imagen de la derecha la que tiene la opción de suavizado.

Una vez creadas se guarda en la tabla de datos genéricos el nombre básico de estas imágenes para poder acceder a ellas en posteriores ocasiones (la imagen suavizada tiene el mismo nombre que la imagen por defecto añadiendo "_1"). En la página de detalles de la medición mostramos siempre estas dos imágenes.

6.8.2 Información de *Meteoblue*

Para contextualizar la información que tenemos sobre la calidad del cielo en un lugar determinado podemos acceder a la información que nos ofrece la página web www.meteoblue.com en el apartado de la información astronómica. Esta web puede

proporcionar datos en función de las coordenadas geográficas de latitud y longitud del lugar al que nos referimos las cuales se introducen en la misma url en el siguiente formato:

`www.meteoblue.com/es/tiempo/outdoorsports/seeing/[latitud]N[longitud]E`

Esta web nos mostrará los datos correspondientes al día en el que hayamos subido la medición puesto que es cuando será consultada. Por tanto, desde el método *mediciones_fotos* de *BuscadorController,* damos valor a la latitud y la longitud gracias a los datos leídos del archivo de medición y seguidamente asignamos esta url a la variable *$enlaceMeteo*. Después ejecutamos el comando *wget* pasándole este enlace para obtener todos los archivos y carpetas que forman dicha página. El comando completo es el siguiente:

```
wget –p –k –E $enlaceMeteo –P $directorioMediciones
```

Donde

- **-p**. Descarga todos los archivos necesarios para poder ver la página de forma local, incluyendo las imágenes, archivos de sonido y hojas de estilo.
- **-k**. Convierte los enlaces del documento o de los documentos para permitir acceder a ellos de forma local.
- **-E**. Añade la extensión *.html* al final del nombre de los ficheros que se descarguen.
- **-P**. Indicamos el directorio en el que se guardarán todos los archivos, carpetas y subcarpetas.
- **$directorioMediciones**. Es la misma carpeta en la que se ha subido el archivo de mediciones del lugar en cuestión.

Este mismo comando se pasa a la función *escapeshellcmd*() de PHP que devuelve el comando preparado como comando válido para finalmente ser ejecutado por medio de la función *shell_exec*(). Esto ocurrirá automáticamente cada vez que el usuario suba un nuevo archivo de medición al sistema.

6.9 Integrar mapa interactivo de Granada en la aplicación

En esta iteración se proporciona la posibilidad de interactuar con un mapa de la provincia de Granada en el que se marcan todas las mediciones realizadas en su localización geográfica exacta. Para ello he utilizado un software adicional de código abierto multiplataforma llamado ***QGIS*** *(Quantum Geographic Information System)* especializado en la vista, edición y análisis geoespacial de datos.

Para mostrarlo en una página de este sistema web he decidido crear de nuevo otra clase controladora llamada MapaQgisController en la que implementar el método que obtenga los datos necesarios para renderizarlo. El método que contiene esta clase es el que se crea por defecto, *index*, y desde él extraigo toda la información genérica e individual de la base de datos al igual que se realizó en anteriores ocasiones. Una vez tengo toda la información de la consulta, voy añadiendo cada fila obtenida en un array en el cual tengo que introducir los siguientes atributos que solicita:

- **`Fid`**. Da un número de identificación a cada elemento que se añada al mapa. Basta con establecer un contador que vaya aumentando con cada inserción.

- **`Cenit`**. Es el valor medio de la magnitud sobre el que se clasifican las medidas. En este caso he decidido asignarle el valor de la magnitud para una declinación de 90º, es decir, la luminosidad que hay mirando al cielo con inclinación completamente vertical.
- **`Nombre`**. Municipio en el que se ha medido.
- **`Imagen`**. Fotografía que aparecerá minimizada tras seleccionar la ubicación.
- **`Geometry`**. Vector con coordenadas geográficas exactas del lugar.

```
array('type' => 'Feature', 'properties' =>
array('fid' => $fid,
'cenit' => $cenit['magnitud'],
'nombre' => $datos['localizacion'],
'imagen'=>$datos['grafico'].'.png'),
'geometry' =>
array('type'=>'Point',
'coordinates' => [$datos['longitud'], $datos['latitud']]))
```

Para cada fila de los datos genéricos se crea un vector como el anterior donde se introducen los datos requeridos el cual se concatena con los vectores añadidos anteriormente.

Cuando ya se haya introducido toda la información necesaria, se concatena este conjunto de vectores dentro de otro array más grande de valores predefinidos de QGIS que no necesito editar. Toda esta información se almacena en una variable y se escribe sobre el fichero *medicionesCielo_5.js* en la carpeta *public/data.*

Finalmente se renderiza como siempre la página que va a alojar este mapa. En este caso he considerado crear un archivo HTML que es *mapa.html.twig* que extiende la plantilla usada anteriormente para el acceso a las mediciones. Como esta plantilla tiene un lateral con enlaces que permite acceder a otras páginas, he hecho que uno de esos enlaces lleve a este archivo. En este archivo he rellenado los bloques de etiquetas HTML añadiendo los links y scripts que el software necesita y poniendo en el bloque de contenido un div con identificador llamado ***map***. Las hojas de estilo y scripts detectan este elemento y sobre él plasman el mapa interactivo.