



TRABAJO FIN DE GRADO  
INGENIERÍA EN INFORMÁTICA

# FIND YOUR STAR

---

Aplicación multiplataforma

**Autor**

Lorena Castillo Palomo

**Tutor**

Sergio Alonso Burgos



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

Granada, Septiembre de 2020





# FIND YOUR STAR

---

Aplicación multiplataforma.

**Autor**

Lorena Castillo Palomo

**Tutor**

Sergio Alonso Burgos



# **FIND YOUR STAR: Aplicación multiplataforma**

Lorena Castillo Palomo

**Palabras clave:** Estrella, Docker, API, Flutter, Astrometry, Socket, NodeJS, ascensión recta, declinación, orientación, dimensiones

## **Resumen**

¿Cuántas veces has mirado al cielo en una noche estrellada y te has preguntado, dónde estará “x” constelación o “tal” estrella? Puede que muchas, puede que ninguna, pero ¿Y si pudieses encontrar estas sin necesidad de un gran telescopio con navegación?

Mi aplicación está dedicada a esas personas que les encanta la astronomía ya sea en gran medida o no. Facilitando que, sin necesidad de un equipo altamente cualificado, puedas lograr el objetivo de encontrar un estrella, una constelación o simplemente pasar el rato buscando estrellas.

FIND YOUR STAR te permite realizar esta búsqueda a través de la captación del cielo estrellado en fotos, ya sea con el móvil o con una cámara y un ordenador. Coteja las estrellas que salen en la foto y muestra una indicación de hacia donde tienes que echar la siguiente foto, para encontrar la estrella o constelación que desees.

A lo largo del documento se expone de forma más detallada la realización de dicha aplicación, explicando el porqué del uso de las diferentes aplicaciones, lenguajes y entornos. La descripción de sus características, requisitos principales y un análisis que recogerá todas las fases del proyecto desde el diseño hasta las pruebas. También se incluye la documentación de todas las llamadas de la API, así como cómo desplegar Docker en cualquier máquina.



# FIND YOUR STAR: Multi platform APP

Castillo Palomo, Lorena

**Keywords:** Star, Docker, API, Flutter, Astrometry, Socket, NodeJS, right ascension, declination, orientation, dimensions.

## Abstract

How many times have you looked at the sky on a starry night and wondered where the “x” constellation or “y” star is? What if you could find these without a expensive telescope with navigation?

My application is dedicated to those people who love astronomy a lot or a little. In that way, without the need of a highly qualified team, to achieve the goal of finding a star, a constellation or just to spend time looking for stars.

FIND YOUR STAR allows you to carry out this search by capturing the starry sky in photos, either with your mobile phone or with a camera and a computer. Check the stars in the photo and show an indication of where you have to take the next photo, to find the star or constellation you want.

Throughout the document, the implementation of this application is explained in more detail, explaining why the different applications, languages and environments are used. The description of its characteristics, main requirements and an analysis that will cover all the phases of the project from design to testing. Also included is the documentation of all API calls, as well as how to deploy Docker on any machine.





---

Yo, **Lorena Castillo Palomo**, alumna de la titulación INGENIERÍA DE INFORMÁTICA, de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 75577436B, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Lorena Castillo Palomo

Granada a 8 de Septiembre de 2020.



---

D. **Sergio Alonso Burgos**, Profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

**Informan:**

Que el presente trabajo, titulado ***FIND YOUR STAR, Aplicación Multiplataforma***, ha sido realizado bajo su supervisión por **Sergio Alonso Burgos**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 8 de Septiembre de 2020 .

**El tutor:**

**Sergio Alonso Burgos**



# Agradecimientos

Quisiera aprovechar estas líneas para agradecer a todas aquellas personas que de una manera u otra han hecho posible mi formación como ingeniera.

En especial a mi hermano por ser mi apoyo, mi referencia a seguir y ayudarme a continuar en mis momentos de mayor bloqueo. A mi padre, madre y abuela, por brindarme un gran apoyo y amor incondicional. También dar gracias a mis amigos, los cuales han recorrido conmigo este camino, en los mejores y peores momentos. Sin olvidar a mi novio, que siempre confió en mí y me dio ánimos para seguir aún cuando yo no podía.

Por último dar gracias a mi tutor, Sergio Alonso Burgos, por brindarme la oportunidad de hacer este hermoso TFG y acompañarme durante el proceso.



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Objetivos . . . . .	1
1.2. Fases de realización . . . . .	2
<b>2. Estado del Arte</b>	<b>3</b>
2.1. Antecedentes de trabajo . . . . .	3
2.2. Dominio del problema . . . . .	8
2.3. Metodologías y tecnologías usadas . . . . .	9
2.3.1. Astrometry . . . . .	9
2.3.2. Servidor . . . . .	9
2.3.3. Flutter . . . . .	14
2.3.4. Docker . . . . .	14
2.4. Aplicaciones similares . . . . .	15
<b>3. Análisis</b>	<b>17</b>
3.1. Análisis del sistema . . . . .	17
3.2. Arquitectura del sistema . . . . .	18
3.2.1. Arquitectura del servidor . . . . .	18
3.2.2. Arquitectura cliente . . . . .	19
<b>4. Diseño</b>	<b>21</b>
4.1. Diagrama de interacción . . . . .	21
4.2. Interfaz de usuario . . . . .	22
4.2.1. Home . . . . .	22
4.2.2. Búsqueda . . . . .	23
4.2.3. Comunicación . . . . .	27
4.2.4. Configuración . . . . .	29
<b>5. Implementación</b>	<b>31</b>
5.1. Cliente . . . . .	31
5.1.1. Main . . . . .	31
5.1.2. Controller_home . . . . .	32
5.1.3. Controller_busqueda . . . . .	32
5.1.4. Controller_opciones y controller_comunicación . . . . .	33

5.1.5. Controller_camara . . . . .	33
5.2. Servidor . . . . .	34
5.2.1. main.js . . . . .	34
5.2.2. api.js . . . . .	36
<b>6. Conclusiones y Trabajos Futuros</b>	<b>41</b>
6.1. Conclusiones . . . . .	41
6.2. Trabajos futuros . . . . .	42
<b>Bibliografía</b>	<b>43</b>
<b>A. Anexo</b>	<b>47</b>
A.1. Instalación en local (Desarrollo) . . . . .	47
A.1.1. Requisitos . . . . .	47
A.1.2. Instalación de los archivos FITS para Astrometry . . .	47
A.1.3. Instalación de módulos . . . . .	47
A.1.4. Instalación y ejecución de PM2 . . . . .	48
A.2. Instalación en local (Producción) . . . . .	48
A.2.1. Requisitos . . . . .	48
A.2.2. Instalación de módulos . . . . .	48
A.2.3. Creación de los archivos en producción . . . . .	48
A.2.4. Instalación de los archivos FITS para Astrometry . . .	49
A.2.5. Instalación de módulos . . . . .	49
A.2.6. Instalación y ejecución de PM2 . . . . .	49
A.3. Instalación en Docker . . . . .	49
A.3.1. Requisitos . . . . .	49
A.3.2. Instalación de módulos . . . . .	49
A.3.3. Creación de la imagen en Docker . . . . .	50
A.3.4. Ejecución de Docker con Docker-Compose . . . . .	50



# Índice de figuras

2.1. Coordenadas . . . . .	4
2.2. Orientación Norte . . . . .	5
2.3. Orientación Sur . . . . .	5
2.4. Constelación de la Osa Mayor . . . . .	6
2.5. Trozo izquierdo constelación de la Osa Mayor . . . . .	7
2.6. Trozo derecho constelación de la Osa Mayor . . . . .	7
2.7. Logo Astrometry . . . . .	9
2.8. Logo NodeJS . . . . .	9
2.9. Diagrama de comunicación de SocketIO. . . . .	10
2.10. Logo SocketIO . . . . .	10
2.11. Logo NodeJS-Express . . . . .	11
2.12. Estructura servidor . . . . .	11
2.13. Logo PM2 . . . . .	12
2.14. pm2.json. . . . .	13
2.15. pm2-prod.json. . . . .	13
2.16. Logo Flutter . . . . .	14
2.17. Logo Docker . . . . .	14
2.18. Aplicación Mapa Estelar . . . . .	15
2.19. Aplicación Sky Map . . . . .	16
3.1. Arquitectura Servidor . . . . .	18
3.2. Arquitectura Cliente . . . . .	19
4.1. Diagrama interacción vistas . . . . .	21
4.2. Página principal . . . . .	23
4.3. Página principal fallo de búsqueda . . . . .	23
4.4. Página de búsqueda . . . . .	24
4.5. Página de búsqueda con imagen . . . . .	24
4.6. Imagen cotejada con éxito . . . . .	25
4.7. Error al cotejar la imagen . . . . .	26
4.8. Página búsqueda móvil . . . . .	26
4.9. Fallo de conexión . . . . .	27
4.10. Página de comunicación . . . . .	28

4.11. Mensajes emergentes . . . . .	28
4.12. Página de comunicación . . . . .	29
4.13. Mensaje de delete . . . . .	30
5.1. Main.dart . . . . .	32
5.2. controller_camera.dart . . . . .	33
5.3. helmet . . . . .	34
5.4. Morgan y Chalk . . . . .	35
5.5. Error 404 . . . . .	35
5.6. Redirect . . . . .	35
5.7. Variables . . . . .	36
5.8. Entorno del tiempo real . . . . .	36
5.9. Conexión de un cliente web con el servidor . . . . .	37
5.10. Envío de valores de la imagen cotejada . . . . .	38
5.11. Recibo de dimensiones . . . . .	39
5.12. Calculo imagen sin dimensiones . . . . .	39
5.13. Calculo imagen con dimensiones . . . . .	39
5.14. Calculo imagen con dimensiones . . . . .	39
5.15. Función para la conversión de píxeles a escala . . . . .	40

# Capítulo 1

## Introducción

En la actualidad existen telescopios con navegación asistida muy útiles a la hora de encontrar los satélites, estrellas, constelaciones y planetas que quieras, pero estos telescopios son muy caros. Si no eres realmente un apasionado de la astronomía o sí lo eres, pero no te lo puedes permitir, esto puede ser un problema.

Con la aplicación FIND YOUR STAR puedes realizar la búsqueda de lo antes mencionado sin necesidad de un telescopio con GPS, simplemente con tu teléfono móvil o con una cámara y un ordenador, además de tu telescopio habitual.

### 1.1. Objetivos

El desarrollo de FIND YOUR STAR viene motivado por querer brindar una alternativa más asequible para aquellas personas a las que le apasiona la astronomía en mayor o menor medida y no se puede permitir un telescopio con navegación asistida.

FIND YOUR STAR es una aplicación multiplataforma diseñada para el reconocimiento de las estrellas que has capturado en una foto con tu móvil o cámara, indicándote a través de ella la dirección que debes tomar para realizar tu siguiente foto y así encontrar lo que estás buscando.

## 1.2. Fases de realización

En este apartado se da una visión general de las fases seguidas a lo largo de la realización del proyecto, que son las correspondientes a la mayoría de un desarrollo software.

- **Estudio:** Investigación sobre las posibles tecnologías a utilizar, posible estudio de mercado y de viabilidad. Indagación en el modo de funcionamiento de Astrometry [1].
- **Análisis:** Definición de requisitos y arquitectura del servidor y el cliente con el objetivo de ver qué es lo que se desea construir exactamente.
- **Diseño:** Tras finalizar las fases de estudio y análisis, se procede a diseñar la arquitectura de la aplicación. Incluye varios diagramas que definirán el diseño la arquitectura del servidor y el cliente y una explicación detallada de la interfaz de usuario.
- **Implementación:** Programación de la aplicación multiplataforma en Flutter y del servidor en NodeJS.
- **Documentación:** La documentación es algo indispensable para el correcto mantenimiento de cualquier producto software. Tanto a nivel de código como de documento, con el fin de dejar constancia por escrito de las fases llevadas a cabo en el ciclo de vida del producto software.

## Capítulo 2

# Estado del Arte

### 2.1. Antecedentes de trabajo

La idea de este TFG surgió del tutor, Sergio Alonso Burgos, que propuso el tema y dio como referencia la página web [Astrometry.net](http://Astrometry.net) para basarse en ella. Por tanto, se realizó una investigación previa sobre dicha web para saber como funcionaba y cómo se podía usar su información y documentos para dar forma a la aplicación.

En primera instancia y puesto que estamos hablando de localizaciones de estrellas en el cielo, había que familiarizarse con los significados de las coordenadas ecuatoriales. Estas se basan en 2 centros (ascensión y declinación recta) [2] 2.1 y una orientación:

- **Declinación DEC:** Es el equivalente de la Latitud Terrestre y tiene su origen en el Ecuador Celeste. La Declinación se mide de  $0^{\circ}$  a  $90^{\circ}$  hacia el Norte, y de  $0^{\circ}$  a  $-90^{\circ}$  hacia el Sur. Su símbolo de representación es ( $\delta$ )
- **Ascensión Recta RA:** Es el equivalente de la Longitud Terrestre, el cual tiene su origen en el Equinoccio de Primavera. La Ascensión Recta está dada en Horas y minutos. Tiene un máximo de 24 Horas. Su símbolo de representación es ( $\alpha$ )



Figura 2.1: Coordenadas

- **Orientación:** La orientación se puede medir de dos maneras en el cielo, una para la búsqueda del sur y otra para la búsqueda del norte:

- **Búsqueda del norte:** Para buscar el Norte se recurre a la Osa Mayor. Una vez que la encontremos, sólo tenemos que mirar hacia las dos estrellas opuestas al rabo de la Osa y, en la dirección que marca la flecha del dibujo 2.2, a cinco veces la distancia que hay entre esas dos estrellas, nos encontramos con la punta de la cola de la Osa Menor. Esta estrella se llama Estrella Polar y es la estrella del Norte, que se encuentra muy cerca del polo norte celeste. Ella nos marca el norte

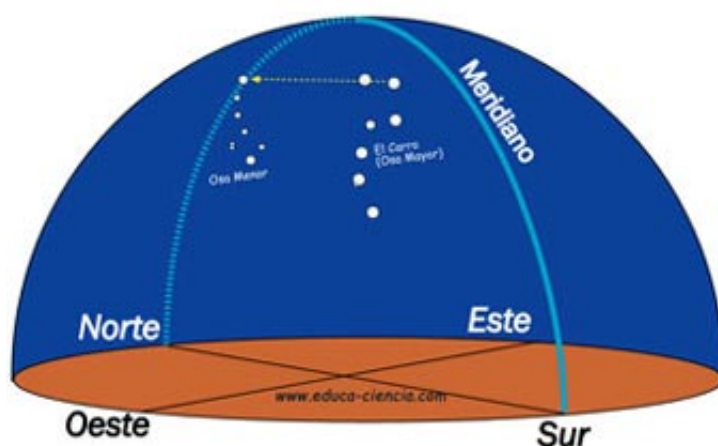


Figura 2.2: Orientación Norte

- **Búsqueda del Sur:** La orientación mediante las estrellas resulta más complicada en el Sur. La única constelación que ayuda un poco es la Cruz del Sur. Así, las dos estrellas del eje mayor de esta cruz señalan, aproximadamente, en la dirección del polo sur celeste, que queda en las cercanías de las tres estrellas más brillantes de la constelación de El Octante 2.3.

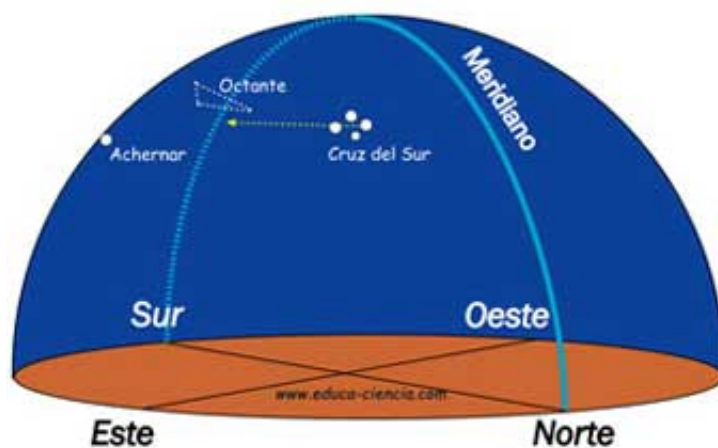


Figura 2.3: Orientación Sur

[3]

Una vez que familiarizado un poco con las coordenadas, se pasó a hacer pruebas con imágenes que se subían a la web de Astrometry. Estas pruebas se realizaron con una foto de la Osa Mayor y el recorte de la misma imagen en varias más pequeñas, para ver los cambios en sus centros y orientación.

## Osa Mayor

2.4

- Center (RA, Dec): (181.157, 55.095)
- Center (RA, hms): 12h 04m 37.749s (13h = 180°)
- Center (Dec, dms): +55° 05' 41.139"
- Orientation: Up is 354 degrees E of N
- Size: 39 x 26.1 deg



Figura 2.4: Constelación de la Osa Mayor

## Osa Mayor cuadrante superior izquierdo

2.5

- Center (RA, Dec): (201.447, 59.596)
- Center (RA, hms): 13h 25m 47.227s (13h = 195°)
- Center (Dec, dms): +59° 35' 47.085"
- Orientation: Up is 345 degrees E of N



- Size: 18.1 x 13.4 deg



Figura 2.5: Trozo izquierdo constelación de la Osa Mayor

### Osa Mayor cuadrante inferior derecho

2.6

- Center (RA, Dec): (201.447, 59.596)
- Center (RA, hms): 13h 25m 47.227s (13h = 195°)
- Center (Dec, dms): +59° 35' 47.085"
- Orientation: Up is 345 degrees E of N
- Size: 18.1 x 13.4 deg

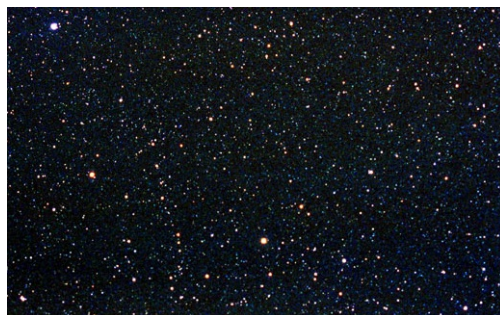


Figura 2.6: Trozo derecho constelación de la Osa Mayor

Como se puede observar hay pequeñas diferencias en sus centros y orientación con respecto a la imagen completa. Cabe destacar, que además de los

centros y la orientación hay otro parámetro que nos dan de la imagen, estas son sus **dimensiones** y nos servirán para una optimización de tiempo en la búsqueda de imágenes en la aplicación, la cual se explicará más adelante con más detalle.

Adentrándonos más en la aplicación, esta iba a estar destinada a ser una aplicación de escritorio, por lo que solo podría funcionar en un ordenador.

## 2.2. Dominio del problema

FIND YOU STAR estaba destinada a ser una aplicación de escritorio únicamente, donde para poder usarla el usuario debería descargarse Astrometry, con todas sus dependencias y, además de ello, 120GB de archivos FITS, que son los correspondientes al catálogo de estrellas para poder cotejar las imágenes. Esto es una gran carencia en la aplicación, ya que un usuario promedio, no tiene los conocimientos necesarios para realizar dicha tarea y esto llevaría a que no le fuese atractivo el querer usar la aplicación. Por ello, se decidió resolver dichas carencias.

En lugar de que fuese una aplicación de escritorio únicamente se ha usado Flutter para realizar una aplicación multiplataforma, para que pudiese ser usada en Android, Web e IOS, ampliando así la cantidad de dispositivos en los que podría tenerse la aplicación y no solo en el ordenador.

Se hizo un servidor con tiempo real, para las comunicaciones entre cliente y servidor. Así ya fuese por la web o por el móvil, una vez que se subiese una imagen, sería cotejada en el momento en el que el servidor la recibiese y mandaría la respuesta al cliente; Si ha encontrado estrellas en la imagen manda las coordenadas, la orientación y las dimensiones, además de actualizar la imagen mostrando en ella donde salen las estrellas que tiene y en caso contrario además de la imagen, con puntos de referencia, manda un mensaje de error.

Por último, para solucionar el hecho de tener que descargarse las dependencias, archivos, etc. Se ha creado una imagen con Docker, de esta manera esa imagen tiene todo lo necesario para que con unos simples comandos se tenga el servidor corriendo y todas las dependencias y ficheros instalados. Permitiendo así que la aplicación tenga una larga vida y que sea fácil de usar para un usuario promedio.

## 2.3. Metodologías y tecnologías usadas

### 2.3.1. Astrometry

Astrometry.net es una página web, que da un servicio de calibración astrométrica para crear metadatos astrométricos correctos y que cumplan con los estándares para cada imagen astronómica útil que se haya tomado. Con esto los creadores esperan ayudar a organizar, anotar y facilitar la búsqueda de toda la información astronómica del mundo [1]. En ella está basada la aplicación, ya que es de donde se han sacado los catálogos de estrellas y los comandos para el procesamiento de las imágenes en un servidor local.



Figura 2.7: Logo Astrometry

### 2.3.2. Servidor

A continuación se especifican los servicios usados para la creación del **servidor** de la aplicación.

#### NodeJS

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. [4]



Figura 2.8: Logo NodeJS

### Socket.IO

Socket.IO es una biblioteca que permite la comunicación en tiempo real, bidireccional y basada en eventos entre el navegador y el servidor. Consiste en:

- un servidor Node.js
- una biblioteca cliente Javascript para el navegador (que también se puede ejecutar desde Node.js): 2.9



Figura 2.9: Diagrama de comunicación de SocketIO.

También hay varias implementaciones de clientes, que son mantenidos por la comunidad, como Dart [5] que es el lenguaje de Flutter, el cual usamos para desarrollar la aplicación [6]



Figura 2.10: Logo SocketIO

## Express

Express es un módulo de NodeJS para crear fácilmente servicios web.



Figura 2.11: Logo NodeJS-Express

Para la formación del servidor se han dividido en carpetas la parte de desarrollo **src** y la parte de producción **dist**. Dentro de **src** se ha dividido el servidor en:

- **main.js**: Servicio principal para ejecutar el servicio web.
- **routes**: Directorio para dividir las peticiones del servidor por categoría.
- **views/web**: Directorio para mostrar las vistas de la aplicación y directorio donde se encuentra la aplicación de flutter.
- **utils/nUtils.js**: Componente con funciones globales para usar dentro del servidor web.

Quedando la estructura del servidor 2.12 de la siguiente forma :

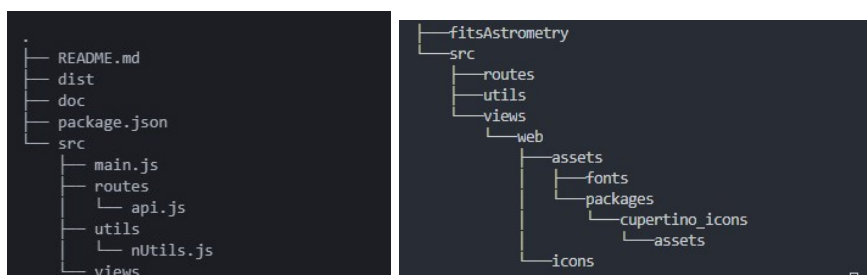


Figura 2.12: Estructura servidor

## PM2

PM2 es un gestor de procesos en producción para las aplicaciones Node.js que tiene un balanceador de carga incorporado. PM2 permite mantener siempre activas las aplicaciones y volver a cargarlas evitando los tiempos de inactividad, a la vez que facilita tareas comunes de administrador del sistema. PM2 también permite gestionar el registro de aplicaciones, la supervisión y la agrupación en clúster.



Figura 2.13: Logo PM2

Se ha creado el archivo **pm2.json** 2.14 y **pm2-prod.json** 2.15 para dicha tarea.

- **script:** Ejecución del servicio especificado.
- **watch:** Refresco del servicio cuando se cambia un archivo dentro de la ruta.
- **ignore\_watch:** Ignora los archivos dentro de la carpeta especificada.
- **args:** Array de argumentos
  - **-color:** se le aplica color a la consola del pm2
- **instances:** Número de instancias que se pueden ejecutar en paralelo ( siempre que esté en modo cluster)
- **exec\_mod:** Para que se ejecute en modo cluster
- **env:** Variables de entorno
  - **NODE\_ENV:** Indica si estoy en *desarrollo* o en *producción*
  - **PORT:** Indica el puerto
  - **REDIS:** Indica la IP del servidor Redis

```
{
  "apps": [{
    "name": "worker",
    "script": "./src/main.js",
    "watch": ["./src"],
    "ignore_watch": ["views"],
    "args": ["--color"],
    "instances": "2",
    "exec_mode": "cluster",
    "env": {
      "NODE_ENV": "development",
      "PORT": 8080,
      "REDIS": "localhost"
    }
  }]
}
```

Figura 2.14: pm2.json.

```
{
  "apps": [{
    "name": "find-you-star",
    "script": "web-api.js",
    "args": ["--color"],
    "env": {
      "NODE_ENV": "production",
      "PORT": 80,
      "REDIS": "redis-server"
    },
    "instances": "max",
    "exec_mode": "cluster"
  }]
}
```

Figura 2.15: pm2-prod.json.

### 2.3.3. Flutter

Flutter es un SDK de código abierto de desarrollo de aplicaciones móviles creado por Google para realizar hermosas aplicaciones, compiladas nativamente, para móvil, web y escritorio desde una única base de código. Se caracteriza por la posibilidad de un desarrollo rápido, una UI expresiva y flexible y un rendimiento nativo [7].

Por todo lo dicho anteriormente es por lo que se ha escogido este framework para la realización de la aplicación multiplataforma.



Figura 2.16: Logo Flutter

### 2.3.4. Docker

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos [8].

Se ha usado Docker para crear una imagen con todos los catálogos de estrellas y su instalación para que el usuario no tenga que hacerlo manualmente.

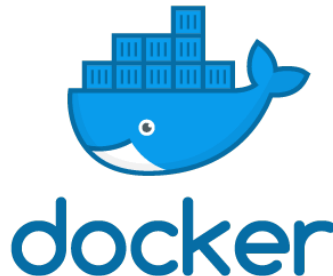


Figura 2.17: Logo Docker







Figura 2.19: Aplicación Sky Map

Aunque todas estas aplicaciones lo que hacen fundamentalmente es pintarte el cielo, sin determinar exactamente a donde estás apuntando. Algunas aplicaciones de móvil hacen uso de su GPS y los giróscopios para indicarte “donde estás apuntando” pero en general su fiabilidad es muy pobre. Mientras que la finalidad de FIND YOUR STAR es a partir de una foto saber donde estás apuntando.

## Capítulo 3

# Análisis

En este capítulo se va a exponer el análisis realizado para la implementación de FIND YOU STAR, así como su arquitectura, facilitando diagramas para una mejor comprensión de la misma.

### 3.1. Análisis del sistema

El requisito principal para que la aplicación funcione sin ningún problema es la instalación de Astrometry, con todas sus dependencias y sus archivos FITS, que son los que contienen el catálogo de estrellas.

Ahora bien, dicho esto, se hace necesario la creación de un servidor el cual sea el que acceda a estos archivos para poder cotejar las imágenes que pase el cliente. Por lo que se necesita, además, una interfaz de usuario intuitiva para que el usuario pueda proveer dichas imágenes. Se debe tener en cuenta, que puesto que es una aplicación multiplataforma, hay diferencias en la comunicación entre cliente y servidor y en el diseño de la interfaz dependiendo de si estamos en un dispositivo móvil o en la web.

Analizando todas las partes, tanto por separado como juntas, se decide utilizar las últimas tecnologías web, para hacer una aplicación que satisfice cada una de las necesidades planteadas:

- Utilizar Node.js, Express y PM2 para la parte del servidor. El servidor será el encargado de recuperar los datos de los archivos FITS, una vez que haya encontrado estrellas en la imagen pasada por el cliente.
- Utilizar Socket.io para la sincronización en tiempo real entre los cliente y los diferentes cluster del servidor, es decir, sincronizar la cantidad de usuarios conectados en cada uno de los diferentes cluster, para que

se puedan cotejar diferentes imágenes sin problemas, desde diferentes dispositivos..

- Utilizar Flutter para crear una aplicación web progresiva además de responsive. Permitiendo así que puedan seguir ejecutándose en segundo plano sin tener que vivir dentro del navegador y ofreciendo que las aplicaciones web puedan verse de forma correcta tanto en diferentes dispositivos como en diferentes resoluciones, respectivamente. Así como aplicaciones IOS y Android nativas.

## 3.2. Arquitectura del sistema

Como se ha comentando la aplicación tiene dos sistemas bien diferenciados: servidor y cliente. En esta sección se exponen las arquitecturas de cada una de ellas.

### 3.2.1. Arquitectura del servidor

El servidor se ha dividido en varios módulos, el principal de ellos es el módulo routes, donde se definen los puntos de acceso al servidor para hacer consultas, ejecutar comandos, etc. Este módulo interactúa con Astrometry a través de la herramienta shelljs para poder ejecutar comandos del mismo, obtener lo deseado y comunicarse con la API la cual hace uso de Socket.io para comunicarse con el cliente. En la figura 3.1 se muestra un esquema de la arquitectura del servidor.

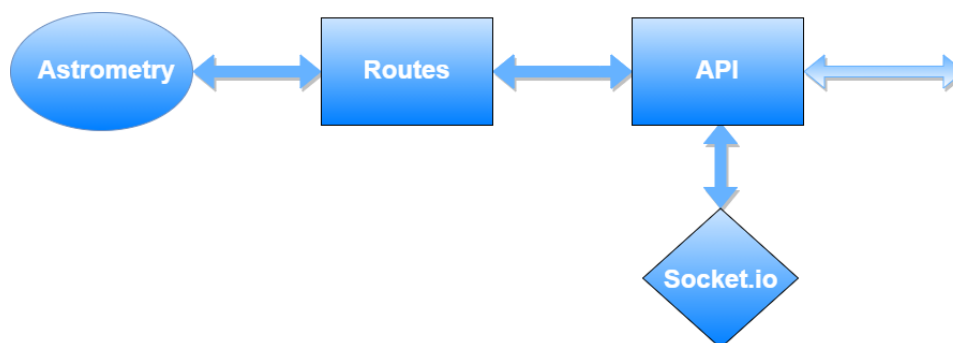


Figura 3.1: Arquitectura Servidor

### 3.2.2. Arquitectura cliente

En la parte del cliente se pueden diferenciar claramente 3 partes: vistas, controladores y rutas. A continuación se detallan las funciones que cumplen cada una de ellas y como interactúan entre sí.

- **Vistas:** también llamadas views son las encargadas de presentar los datos al usuario de una forma amigable. Están escritas en Dart, y hacen usos de distintas clases definidas en Flutter para su correcta visualización en diferentes dispositivos y resoluciones.
- **Controladores:** también conocidos como controllers son los intermediarios entre los servicios y las vistas, es decir son los encargados de proveer datos a las vistas. Por un lado, obtienen los datos a través de los servicios, hacen las transformaciones necesarias sobre estos, y los envían a las vistas para que estas los “pinten”. También son el intermediario en la otra dirección, en este caso a través de socket.io. Cabe destacar que en muchas ocasiones son las vistas las encargadas de ser el medio a través del cual un usuario puede modificar los mismos o crear nuevos datos. Se realiza el mismo procedimiento antes explicado pero en sentido contrario.
- **Rutas:** en este caso es el archivo *main.dart* definen los puntos de acceso del usuario a la aplicación. Cada ruta tiene asociada una vista y un controlador, que hace que estos se relacionen entre sí.

En la figura 3.2 se muestra un esquema de como interactúan entre sí las partes descritas anteriormente.

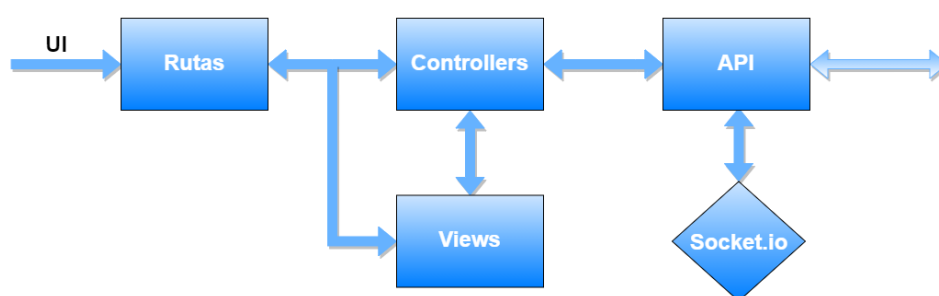


Figura 3.2: Arquitectura Cliente



## Capítulo 4

# Diseño

En el presente capítulo se describe todo el proceso de diseño de la aplicación FIND YOUR STAR, incluyendo diagramas e imágenes de la interfaz de usuario para una mejor comprensión de la misma.

### 4.1. Diagrama de interacción

A continuación se muestra un diagrama de interacción 4.1 de lo que es el conjunto de vistas que forma la interfaz de la aplicación.

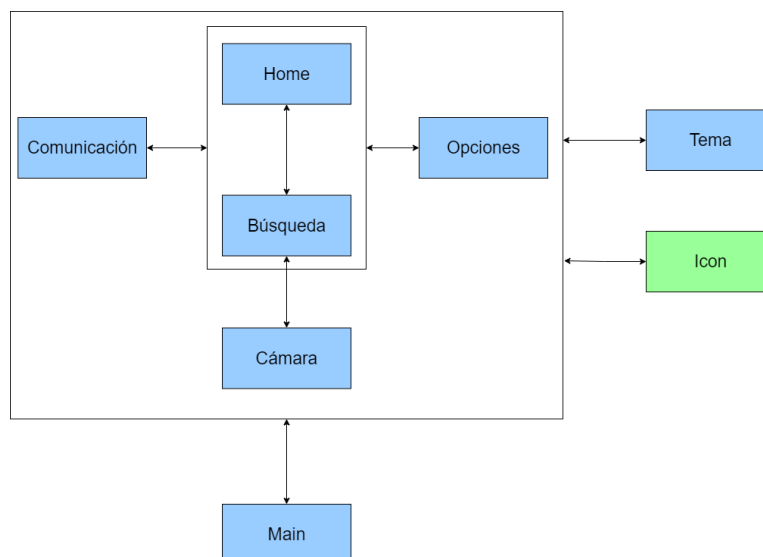


Figura 4.1: Diagrama interacción vistas

## 4.2. Interfaz de usuario

En esta sección se describirá como está diseñada la interfaz de usuario. Se trata de un diseño intuitivo, cuyo manejo es sencillo para un usuario promedio.

Se definirán cada una de las visuales que componen la aplicación. También se expondrán algunos detalles relevantes. Además en cada parte se explicarán y mostrarán ambas visuales en ambos dispositivos y las diferencias entre ellos.

### 4.2.1. Home

El home es la página principal de la aplicación y es la que se muestra cuando iniciamos esta 4.2. En ella se muestran unos campos a rellenar:

- **Center(RA,Dec):** son los campos de las coordenadas ecuatoriales dadas en números decimales.
- **Center(RA,hms):** ascensión recta dada en horas, minutos y segundos
- **Center(Dec,dms):** declinación dada en grados, minutos y segundos
- **Orientación:** la orientación la cual se debe definir si se está situado arriba (búsqueda norte) o abajo (búsqueda sur), y los grados y orientación respecto a la cual se está orientado valga la redundancia.

También se compone de un botón de buscar en la esquina inferior derecha, el cual nos llevará a la página de búsqueda, cuando los datos estén rellenados correctamente. Esto es que sean valores numéricos y que estén dentro de los parámetros correctos, por ejemplo, no campo de minutos no puede ser un valor superior a 60. En el caso de que algún valor sea incorrecto o se encuentre vacío, no nos dejará pasar a la página de búsqueda y saldrán unos mensajes de error como se muestra en la figura 4.3.

Por último, aunque no menos importante, nos fijamos en la barra de cabecera, en ella se muestra el nombre de la aplicación en la esquina superior izquierda y dos iconos (comunicación y configuración) en la esquina superior derecha, estos últimos se explicarán más adelante.



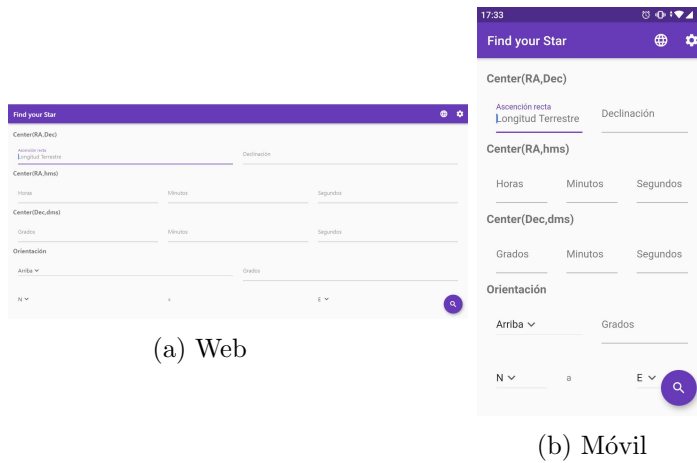


Figura 4.2: Página principal

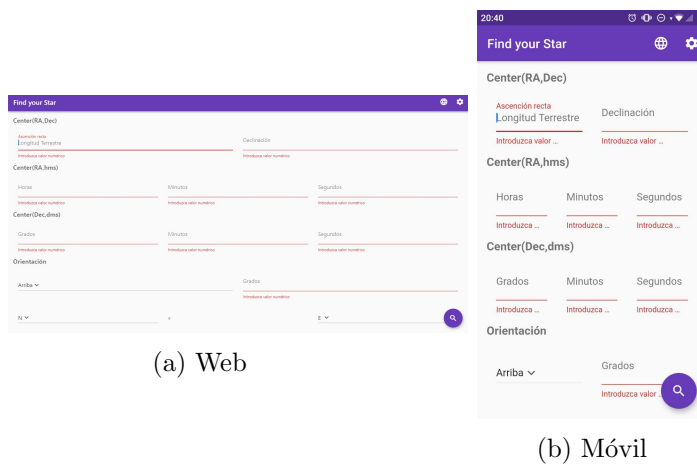


Figura 4.3: Página principal fallo de búsqueda

#### 4.2.2. Búsqueda

La página de búsqueda comparte con la página principal la barra de cabecera y es aquella en la que el usuario sube la imagen que se desea cotejar.

##### Web

En el caso de la aplicación web saldrá un mensaje que ponga “esperando cambios en galería” 4.4.

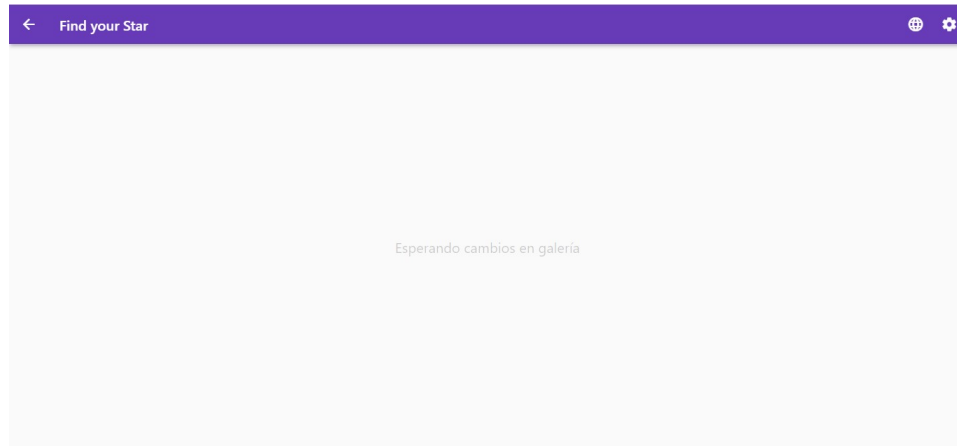


Figura 4.4: Página de búsqueda

Una vez que el usuario suba una foto a la carpeta correspondiente, la página cambiará a una en la que se muestre en la mitad izquierda la foto que se ha subido y en la derecha, varios mensajes 4.5:

- Título de la imagen.
- Coordenadas de la estrella buscada, que coinciden con las introducidas anteriormente en la página principal.
- Mensaje de espera mientras se coteja la imagen.
- Icono de “cargando”.

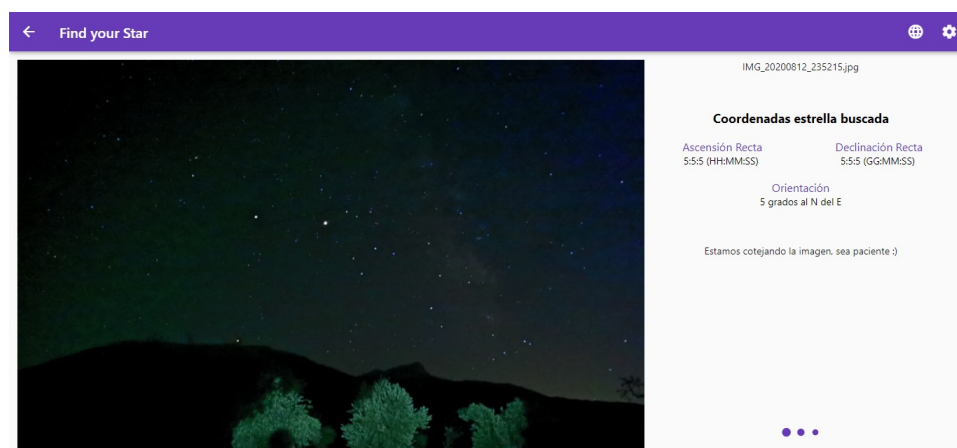


Figura 4.5: Página de búsqueda con imagen

Si se han encontrado estrellas en la imagen 4.6, volverá a cambiar la página mostrando ahora la imagen con todas las estrellas encontradas en ella y en la otra parte se habrán producido dos cambios:

- El mensaje de espera se habrá cambiado por las coordenadas de la imagen que se ha cotejado.
- El icono de “cargando”, se habrá sustituido por el tiempo que ha tardado en cotejar la imagen.

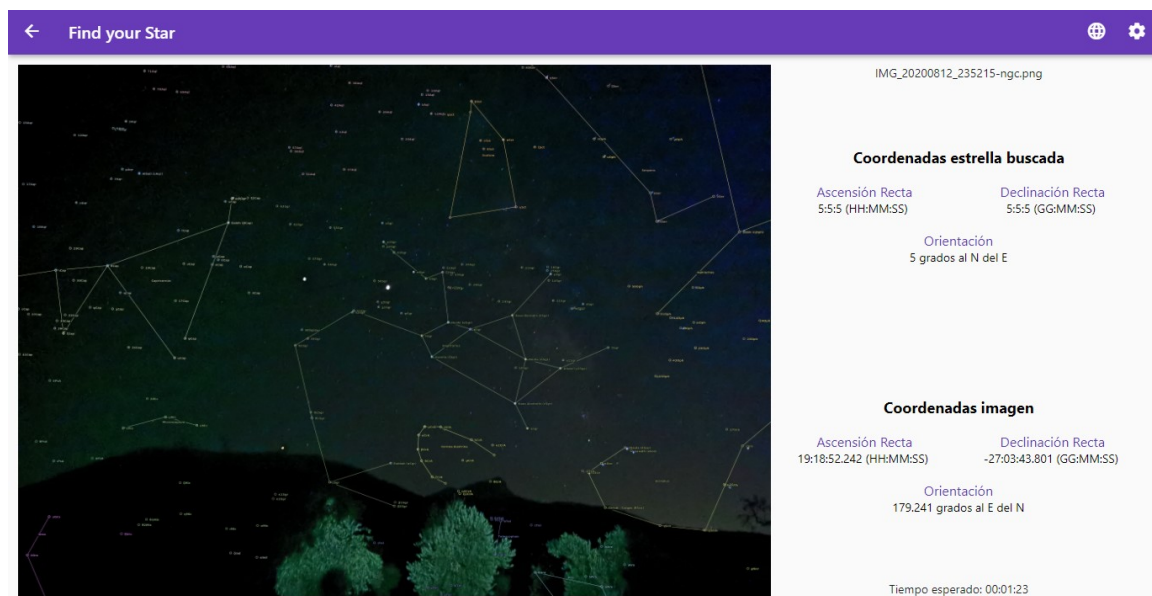


Figura 4.6: Imagen cotejada con éxito

También cabe la posibilidad de que no se encuentre ninguna estrella en la imagen, en este caso la página de búsqueda se mostrará como en la figura 4.7. La imagen mostrará los puntos que ha tomado de referencia y además el mensaje se habrá cambiado por: No se ha encontrado ninguna estrella :( . Inténtelo de nuevo.

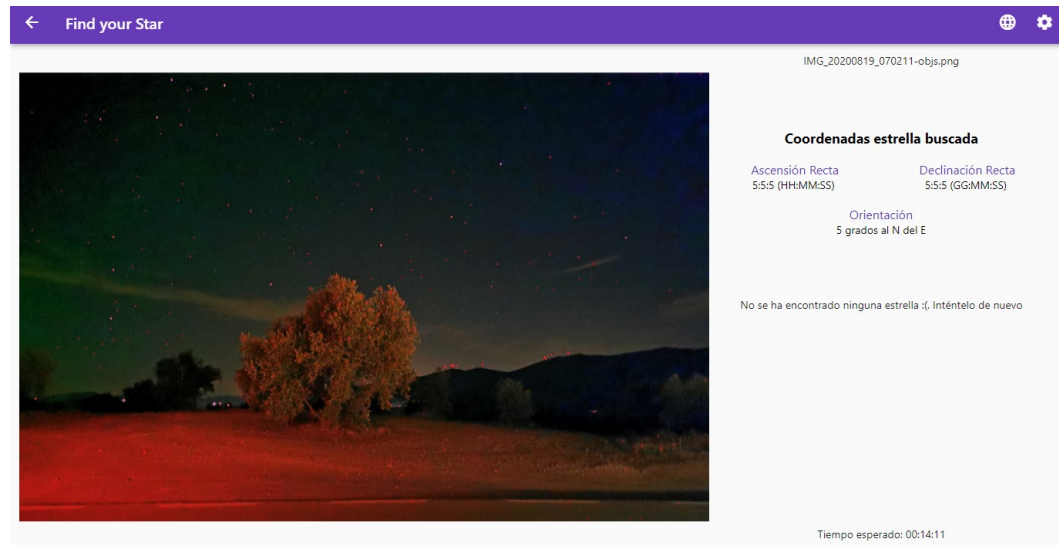
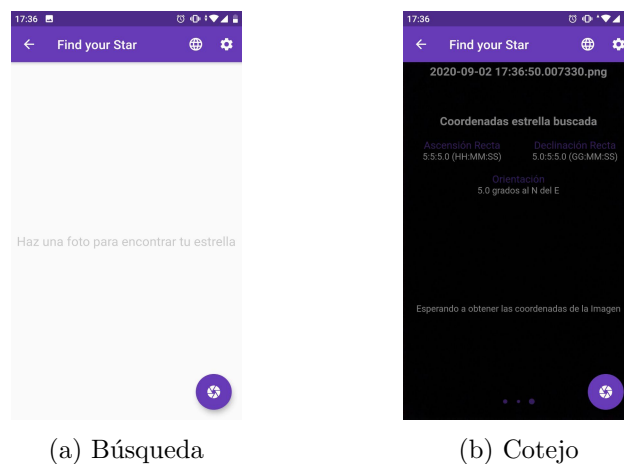


Figura 4.7: Error al cotejar la imagen

## Móvil

En el caso de la aplicación móvil saldrá un mensaje que ponga “Haz una foto para encontrar tu estrella” 4.8a y un botón para acceder a la cámara del móvil (vista **Cámara**) .

Una vez que hagamos una foto con la cámara, se mostrará lo mismo que en la aplicación web, salvo por un formato visual diferente, más conveniente para la pantalla del móvil, como se puede apreciar en la figura 4.8b



(a) Búsqueda

(b) Cotejo

Figura 4.8: Página búsqueda móvil

Teniendo en cuenta este formato visual, comparte las mismas características que las de web, explicadas anteriormente.

### 4.2.3. Comunicación

Se puede dar el caso en el que una vez introducidas las coordenadas de la estrella que se quiere buscar correctamente, al pasar a la página de búsqueda nos salga el mensaje “Intentando conectar... Compruebe IP y Puerto”, como se muestra en la figura

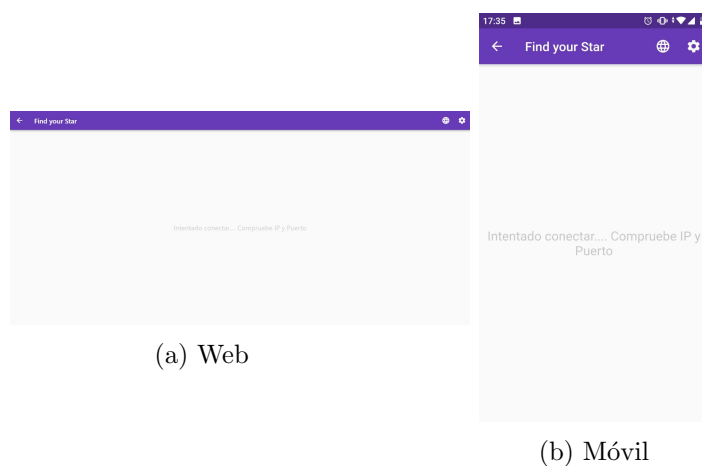
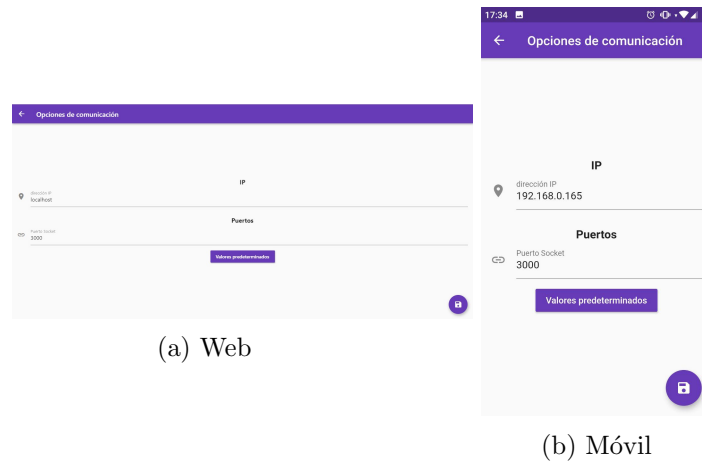


Figura 4.9: Fallo de conexión

Esto es porque o bien la dirección IP a la que está conectada el dispositivo, no es la misma que la del servidor en la que está corriendo el servicio o el puerto de escucha, por el que se envía la información entre cliente y servidor no es el correcto. Para poder solucionar este problema de forma sencilla para el usuario y que la aplicación funcione sin ningún problema, se ha desarrollado la visual de comunicación, a la cual se puede acceder desde el *Home* o desde la página de *búsqueda* a través del icono del “mundo”, y consta de dos apartados; uno para poner la IP y otro para poner el puerto, como se muestra en la figura 4.10

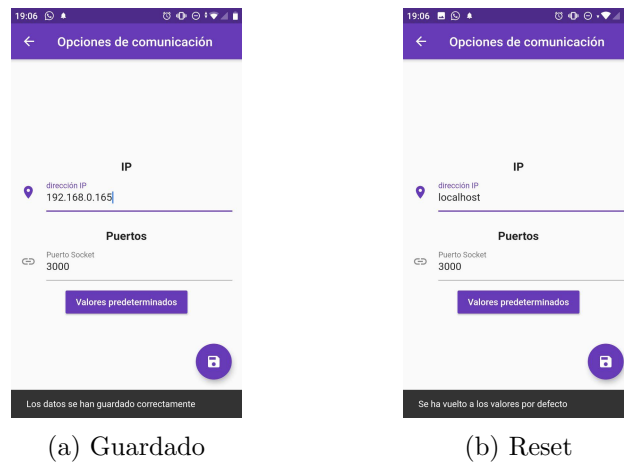


(a) Web

(b) Móvil

Figura 4.10: Página de comunicación

Esta página consta de un botón de guardado, para guardar la configuración si se ha cambiado y un botón “reset” para volver a los valores predeterminados. Cuando son pulsados, se muestra un mensaje emergente de guardado o de vuelta a los valores por defecto, respectivamente. 4.11



(a) Guardado

(b) Reset

Figura 4.11: Mensajes emergentes

#### 4.2.4. Configuración

Por último, la página de configuración a la que se puede acceder a través del icono de la “tuerca”. La finalidad de esta vista es poder poner las dimensiones, en grados o en arcominutos, de la cámara que usemos para realizar las fotos en el caso de que las sepamos. Si no las sabemos, una vez que se coteje la primera imagen, se guardarán las dimensiones y saldrán en esta vista automáticamente.

Consta de un desplegable para elegir si la unidad de las dimensiones y dos apartados; uno para el ancho y otro para el alto, como se puede apreciar en la figura 4.12

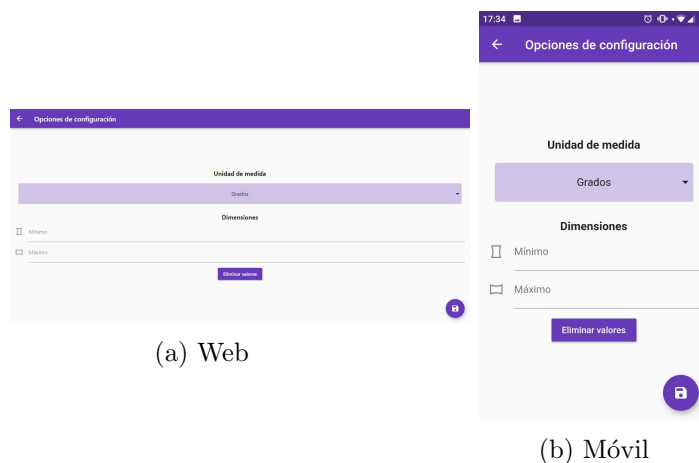


Figura 4.12: Página de comunicación

Consta de un botón de guardar cuya finalidad y mensaje emergente es igual al de la página de *comunicación* y un botón para eliminar los valores. Cuando este último es pulsado, se muestra un mensaje emergente confirmando la acción 4.13

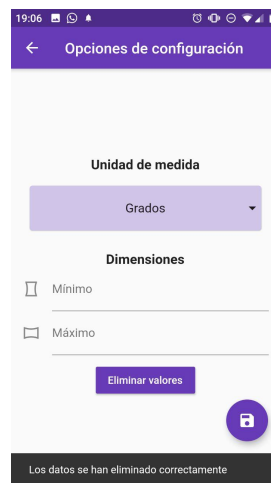


Figura 4.13: Mensaje de delete



## Capítulo 5

# Implementación

En este capítulo se mostrará la implementación de la aplicación mostrando detalles que pueden ser interesantes y que pueden ayudar a la comprensión de ciertos detalles claves dentro de la aplicación.

Se dividirá en 2 secciones, correspondientes a las 2 partes principales del sistema, con el fin de mejorar su claridad y ayudar a la comprensión.

### 5.1. Cliente

En esta sección se explicarán brevemente las partes involucradas en la parte de cliente de la aplicación. Como se ha comentado en capítulos anteriores, esta parte está desarrollada con el SDK Flutter. A continuación, se darán algunos detalles de las rutas, controladores y vistas, así como de los mecanismos utilizados para la comunicación en tiempo real con el servidor.

#### 5.1.1. Main

Es el archivo principal de la aplicación, en el se define el título de la aplicación, el tema de esta, la ruta inicial, que sería el *Home* de la aplicación y las rutas a las otras vistas.

A continuación se muestra una imagen 5.1 de lo descrito anteriormente:

```

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Find your Star',
      theme: getMiTema(MisTemasKeys.CLARO),
      home: MyHomePage(title: 'Find your Star'),
      initialRoute: '/',
      routes: {
        '/config': (context) {
          Map argumento = ModalRoute.of(context).settings.arguments;
          return ViewOptions(name: argumento['name']);
        },
        '/busqueda': (context) => ViewBusqueda(),
        '/camara': (context) => ViewCamera(),
        '/comunicacion': (context) {
          Map argumento = ModalRoute.of(context).settings.arguments;
          return ViewComunicacion(name: argumento['name']);
        },
      },
    ); // MaterialApp
  }
}

```

Figura 5.1: Main.dart

Cuando se inicia la aplicación este se encarga de llevarnos a la vista de la página *Home*, esta vista tiene asociado el *controller\_home*.

### 5.1.2. Controller\_home

Como se ha explicado anteriormente, en la vista *Home* se introducen las coordenadas de la estrella que se quiere buscar y se validan al pulsar el botón de búsqueda. Si todo ha ido correctamente pasa a la vista de *Búsqueda* a través del *controller\_busqueda* y los datos serán guardados por el controlador. Esto se hace a través de la dependencia **shared\_preference** [9].

**Shared\_preference** se usa para guardar la configuración de la aplicación Flutter y las preferencias del usuario para su posterior reutilización.

### 5.1.3. Controller\_busqueda

Este controlador es el asociado a la vista *Búsqueda* y se caracteriza principalmente por tener la clase **TiempoReal**. En esta clase se hace uso de la dependencia *socket\_io\_client*, la cual se utiliza para la sincronización en tiempo real del cliente con el servidor. De esta manera se mantiene en un estado de escucha, para cuando el servidor envíe la imagen cotejada, las coordenadas y las dimensiones de la misma. De igual forma, se encarga de emitir la imagen pasada por el usuario y sus dimensiones, en el caso de que la tenga, es decir, que estén guardadas en el **shared\_preference** en 3 variables del *controller\_opciones*. También es el encargado de comprobar que la IP y el puerto, que están guardados en el **shared\_preference** en dos variables

del *controller\_comunicación*, coincidan con los definidos en el servidor.

Hay que tener en cuenta un punto fundamental, y es que la vista cambia dependiendo del dispositivo que se esté usando. Para poder diferenciarlos el servidor manda un *tipo*; web o mobile y un *uuid*, este último es un id único que se le asocia a cada uno de los dispositivos móviles que se conecten. El controlador recoge esto y envía los valores necesarios a la vista para que sea mostrada correctamente.

#### 5.1.4. Controller\_opciones y controller\_comunicación

Su función es guardar, eliminar o resetear los valores que corresponden a cada una de ellas (dimensiones, unidad, IP y puerto) en el *shared\_preference*.

#### 5.1.5. Controller\_camara

Este controlador está asociado a la vista *Cámara*, que son exclusivos para los dispositivos móviles ya que hacen uso de su propia cámara para realizar la foto. En este controlador se utiliza la dependencia **path**, para construir la ruta donde se guarda la imagen capturada. Una vez se ha hecho la foto vuelve a la vista de *Búsqueda* para el cotejo de la imagen. 5.2

```
Future<Map> loadCamera(var cameras, var firstCamera,
    CameraController controller, initializeControllerFuture) async {
  // Obtén una lista de las cámaras disponibles en el dispositivo.
  cameras = await availableCameras();

  // Obtén una cámara específica de la lista de cámaras disponibles
  firstCamera = cameras.first;

  controller = CameraController(firstCamera, ResolutionPreset.ultraHigh);

  // A continuación, debes inicializar el controlador. Esto devuelve un Future!
  initializeControllerFuture = controller.initialize();
  Map result = new Map();
  result['camara'] = firstCamera;
  result['controller'] = controller;
  result['iniciar'] = initializeControllerFuture;
  return result;
}

Future<String> hacerFoto(iniciar, CameraController controller) async {
  await iniciar;

  // Construye la ruta donde la imagen debe ser guardada usando
  // el paquete path.
  final path = join(
    // (await getApplicationDocumentsDirectory()).path,
    (await getTemporaryDirectory()).path,
    '${DateTime.now()}.png',
  );

  //Esperamos a que haga una foto y la guardamos en la ruta creada anteriormente
  await controller.takePicture(path);

  return path;
}
```

Figura 5.2: controller\_camera.dart

## 5.2. Servidor

Ahora se pasará a explicar la parte de la implementación del servidor detalladamente. Por lo que se explicará cada parte importante de forma exhaustiva y de destacarán los elementos más importantes.

### 5.2.1. main.js

El archivo **main.js** es el encargado de preparar todo lo necesario para que el servidor funcione sin problemas.

Se requieren los siguiente *middleware de terceros*:

- **morgan:** Muestra la información de las peticiones (method, status, etc). [10]
- **chalk:** Asigna un color a un string por consola. [11]
- **helmet:** Ayuda a proteger la aplicación estableciendo varias cabeceras HTTP. [12]
- **body-parser:** Accede a las variables en peticiones POST. [13]
- **compression:** Para la compresión. [14]

### Helmet

Helmet se usa para la protección del servidor y por tanto debe ser lo primero que se use antes de todo el código

```
app.use(helmet()); // Siempre primero, protección de la aplicación.
```

Figura 5.3: helmet

### Morgan y Chalk

Como se ha dicho antes son usado para mostrar información de la petición y asignar un color a un string por consola. Se ha creado una función para dar un formato, dependiendo del tipo de método. Ejemplo:

```
//Función para mostrar por consola información de las peticiones con coloritos :)
function formatoMorgan(tokens, req, res) {
  var method = tokens.method(req, res);
  switch (method) {
    case 'GET':
      method = chalk.blue(method);
      break;
    case 'POST':
      method = chalk.green(method);
      break;
    case 'PUT':
      method = chalk.orange(method);
      break;
    case 'DELETE':
      method = chalk.red(method);
      break;
  }
}
```

Figura 5.4: Morgan y Chalk

## Control de página errónea

Se ha controlado, que en el caso de que se busque cualquier página que no sea una de las de la aplicación envíe un estado 404 y un mensaje.

```
//The 404 Route (ALWAYS Keep this as the last route)
app.get('*', function (req, res, next) {
  res.status(404);
  next("Esta página no existe");
});
```

Figura 5.5: Error 404

## Redirección a página web

Se ha especificado un redireccionamiento a la página web, donde se encuentra la compilación en producción de la aplicación web en Flutter 5.6.

```
app.get('/', function (req, res) {
  res.redirect('/vistas/web');
});
```

Figura 5.6: Redirect

### Variables de entorno

Se han usado variables de entorno para cambiar fácilmente dichas variables, sin modificar el código. En estas se pueden encontrar tanto el modo de entorno, el puerto web y la IP del servidor Redis [14]. En el caso de que alguna de las variables no esté especificada, se le asignará un valor por defecto como podemos ver en la imagen 5.7

```
// Variables
var port = process.env.PORT || 3000;
var environment = process.env.NODE_ENV || 'development';
```

Figura 5.7: Variables

#### 5.2.2. api.js

En el archivo **api.js** se realiza tanto la sincronización en tiempo real con *socket.io*, como la sincronización entre los clusters mediante el servidor Redis. Ahora se pasará a explicar cada una de sus partes:

#### Creación del entorno Socket.io y enlace con Redis Server

Se creará el entorno socket.io escuchándose por el puerto 3000, esto quiere decir que el cliente socket.io debe poner la IP del servidor y el puerto 3000.

Para que los clusters estén sincronizados con el entorno socket.io se le añadirá un adaptador de Redis server, donde se especifica tanto la IP como el puerto de dicho servidor.

```
//Creación del servidor socket.io escuchando por el puerto 3000
var io = require('socket.io')(3000, {
  serveClient: false,
  // below are engine.IO options
  pingInterval: 10000,
  pingTimeout: 5000,
  cookie: false
});

//Uso de redis comunicación adecuada entre dos nodos
const redis = require('socket.io-redis');
io.adapter(redis({
  host: redisServer,
  port: 6379
}));
```

Figura 5.8: Entorno del tiempo real

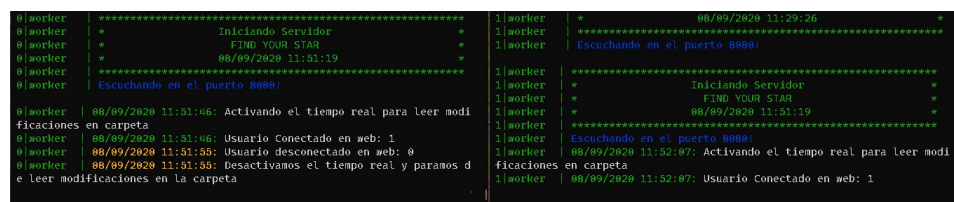
## Comunicación cliente-servidor

Una vez se ha creado el entorno de socket.io y se ha añadido el servidor Redis, se escuchará por la función “connection” a todos los clientes que se conecten a nuestra aplicación, dependiendo del cluster al que se conecten. Al conectarse un cliente este debe pasar por parámetros un identificador único (*uuid*) y el tipo del dispositivo (*web* y *mobile*). Al identificar cada cliente por su *uuid*, se puede mandar la respuesta a la petición hecha anteriormente, aunque el cliente se haya desconectado y vuelto a conectar.

### Cliente tipo Web esperando nueva imagen

Cuando se conecte un cliente tipo *Web*, se usará la dependencia **chokidar** [15] haciendo que cada vez que se añada una imagen a la ruta “/usr/local/etc/tiempo\_real”, el servidor lea la imagen la codifique en *base64* y la emita a todos los usuarios de tipo *Web*. En este caso, como los usuarios web están esperando las modificaciones de la carpeta que se encuentra en el servidor, estos no son únicos y por tanto en este caso no se usaría el *uuid*, indicando en este caso un identificador web, llamado “star”.

En el caso de que no hubiese ningún cliente de tipo *Web*, el proceso por el que se observan los cambios en la ruta mencionada anteriormente se terminará, volviéndose a iniciar una vez que se conecte un cliente web 5.9.



```

0|worker | *****
0|worker | *           Iniciando Servidor           *
0|worker | *           FIND YOUR STAR                *
0|worker | *           08/09/2020 11:51:19            *
0|worker | *****
0|worker | Escuchando en el puerto 8080.
0|worker | 08/09/2020 11:51:46: Activando el tiempo real para leer modi
0|worker | ficaciones en carpeta
0|worker | 08/09/2020 11:51:46: Usuario Conectado en web: 1
0|worker | 08/09/2020 11:51:55: Usuario desconectado en web: 0
0|worker | 08/09/2020 11:51:55: Desactivamos el tiempo real y paramos d
0|worker | e leer modificaciones en la carpeta
1|worker | ***** 08/09/2020 11:29:26 *****
1|worker | *****
1|worker | *           08/09/2020 11:29:26            *
1|worker | *           Iniciando Servidor           *
1|worker | *           FIND YOUR STAR                *
1|worker | *           08/09/2020 11:51:19            *
1|worker | *****
1|worker | Escuchando en el puerto 8080.
1|worker | 08/09/2020 11:52:07: Activando el tiempo real para leer modi
1|worker | ficaciones en carpeta
1|worker | 08/09/2020 11:52:07: Usuario Conectado en web: 1

```

Figura 5.9: Conexión de un cliente web con el servidor

Una vez se emite la imagen al cliente, se empezará a calcular las coordenadas de esta.

### Cliente tipo Mobile enviado nueva imagen

En el caso de clientes de tipo *mobile* una vez están conectados al servidor, le enviarán la foto, codificada en *base64*, que se haya capturado con la cámara. El servidor decodificará dicha imagen y empezará a calcular las coordenadas de esta.

### Cálculo de coordenadas

El cálculo de coordenadas es el mismo tanto para web (cuando se añade una nueva imagen a la ruta), como para móvil (cuando se envía una foto). Para realizar dicho cálculo, se siguen una serie de pasos.

Primero se comprueba si están guardadas las dimensiones de la imagen, las cuales son enviadas por el cliente, y luego se procede a ejecutar el comando “solve-field” de Astrometry con la ruta donde está la imagen que se quiere cotejar e indicando que guarde los resultados en “/usr/local/etc/coordenadas” para comenzar con el cotejamiento de la imagen recibida, esto es posible gracias a la dependencia *shell* [16], la cual permite ejecutar comandos de shell de Unix sobre el servidor de Node.js.

Lo normal cuando se sube la primera imagen que se quiere cotejar, es que el cliente no sepa de antemano cuales son las dimensiones de la imagen que ha enviado al servidor, a no ser que el usuario las haya introducido manualmente en la vista *configuración*. Por lo que cuando el servidor recibe la imagen y comprueba que no se han enviado también las dimensiones, llama a la función *conversionToScale*, la cual se explicará con más detalle en el siguiente apartado, y se ejecutará el comando “solve-field” con los datos que retorne dicha función, para el cálculo de coordenadas.

Si la imagen se ha cotejado con **éxito**, el servidor enviará al cliente, el título de esta, la imagen con las estrellas en *base64*, las coordenadas, las dimensiones de dicha imagen, mostrando por consola el mensaje en azul que se puede ver en la figura 5.10 y el tiempo que ha tardado en cotejarla.

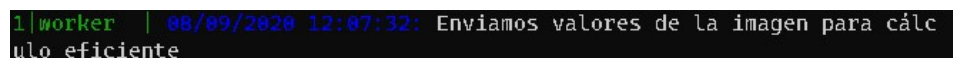
A screenshot of a terminal window with a black background. The text is displayed in a monospaced font. The first line is green and reads '1|worker'. The second line is blue and reads '| 08/09/2020 12:07:32: Enviamos valores de la imagen para cálculo eficiente'.

Figura 5.10: Envío de valores de la imagen cotejada

El cliente recibirá los datos y los guardará en el **shared\_preference**, de esta manera cuando se quieran cotejar las siguientes imágenes realizadas con la misma cámara o móvil, el cliente enviará al servidor las dimensiones que ha guardado con anterioridad. El servidor las recibirá (cuando eso ocurre se muestra un mensaje en azul por consola 5.11) y ejecutará el comando “solve-field” con ellas, consiguiendo así una mejora sustancial en el tiempo que Astrometry tarda en una imagen, como se puede observar en las figuras 5.12 y 5.13.



```
0|worker | 08/09/2020 12:29:50: Se han recibido los parámetros { scale:  
'degwidth', max: 65.5621, min: 51.7027 }
```

Figura 5.11: Recibo de dimensiones

```
1|worker | 08/09/2020 12:04:15: Calculando imagen....  
1|worker | 08/09/2020 12:07:32: Tiempo total: 00:03:17  
1|worker | 08/09/2020 12:07:32: Enviamos valores de la imagen para cálculo eficiente
```

Figura 5.12: Calculo imagen sin dimensiones

```
1|worker | 08/09/2020 12:09:22: Calculando imagen....  
1|worker | 08/09/2020 12:09:39: Tiempo total: 00:00:16
```

Figura 5.13: Calculo imagen con dimensiones

En el caso de que la imagen no haya podido ser cotejada, se enviará al cliente la imagen de error que devuelve “solve-field” en *base64*, un mensaje de error, que se mostrará en lugar de las coordenadas como se ha explicado antes en el diseño de la interfaz (este mismo mensaje será mostrado por consola en color rojo) y el tiempo que ha tardado 5.14.

Para los casos de error, que suelen tardar mucho tiempo, se ha puesto un parámetro límite de 5’ en el “solve-field”, aunque este no siempre funciona correctamente.

```
0|worker | 08/09/2020 12:50:53: No se ha encontrado ninguna estrella  
0|worker | 08/09/2020 12:50:53: Tiempo total: 00:10:55
```

Figura 5.14: Calculo imagen con dimensiones

## Función `conversionToScale`

Esta función se ha creado con el objetivo de reducir el tiempo que tarda Astrometry en cotejar una imagen sin tener sus dimensiones previamente, ya que usando el comando por defecto, podía llegar a tardar más de una hora en cotejar una imagen que tuviese estrellas.

La función consta de una fórmula simple para la conversión de píxeles (unidad en la que vienen las dimensiones de la imagen) a escala (unidad que usa solve-field para el cotejo de las imágenes) como se puede observar en la figura 5.15.

```
//Funcion para convertir Los pixeles (dimensiones imagen)
//en escala (dimension soportada por el solve-field)
function conversionToScale(anchoP, altoP) {
    var ancho = anchoP / 3780;
    var alto = altoP / 3780;

    if (ancho > alto) {
        return ancho;
    } else {
        return alto;
    }
}
```

Figura 5.15: Función para la conversión de píxeles a escala

## Capítulo 6

# Conclusiones y Trabajos Futuros

### 6.1. Conclusiones

Este capítulo está dedicado a la conclusiones finales y personales del proyecto. Los objetivos de este proyecto eran crear una aplicación multiplataforma, con sincronización en tiempo real entre cliente y servidor y que nos mostrase en la UI una indicación de hacia donde se encuentra la estrella que se está buscando, partiendo de la foto capturada previamente con la cámara o el móvil. Los dos primeros objetivos se han podido realizar completos con éxito, sin embargo el último y principal objetivo no se ha podido llevar a cabo. Esto ha sido; por una parte por falta de conocimientos más amplios sobre como se determinan con exactitud, las coordenadas y orientaciones en el campo estelar, para la realización de un algoritmo capaz de determinar hacia donde se encuentra una estrella de otra y, por otra parte, la situación del **COVID19** que ha impedido que se pueda realizar **pruebas de campo reales** que habrían facilitado en cierta medida, una mejor comprensión de lo especificado anteriormente.

Para lograr los dos objetivos primeros, la parte de análisis ha sido fundamental, para una correcta comprensión de como se debían usar las tecnologías y una correcta estructuración de la relación entre el cliente y el servidor.

Desarrollar esta aplicación me ha proporcionado a nivel personal, el tener la oportunidad de trabajar con tecnologías nuevas que no había usado nunca y tampoco se estudian en la carrera. Además de aprender a desarrollar un proyecto de principio a fin y ampliar mis conocimientos sobre la materia.

## 6.2. Trabajos futuros

En el futuro esta aplicación incluiría el objetivo principal de indicar hacia donde se encuentra la estrella que se está buscando, partiendo de la foto capturada previamente con la cámara o el móvil, la cual como se ha dicho antes no se ha podido realizar principalmente por la situación del **COVID19**. Además, Flutter pronto permitirá también crear aplicaciones de escritorio nativas, por lo que se podría implementar fácilmente en la aplicación debido a como se ha estructurado el código.

Como otros objetivos futuros que no eran los principales, se podría poner la opción de que la aplicación cogiese la ubicación del usuario al activar el GPS, lo que sería una ayuda para conseguir el objetivo principal.

# Bibliografía

- [1] Astrometry recuperado de <http://astrometry.net/>.
- [2] Significado de las coordenadas ecuatoriales recuperado de [https://es.wikipedia.org/wiki/coordenadas\\_ecuatoriales](https://es.wikipedia.org/wiki/coordenadas_ecuatoriales).
- [3] Orientación norte y sur recuperado de <https://www.educacion.com/astronomia-orientacion.htm>.
- [4] Definición de nodejs recuperado de <https://es.wikipedia.org/wiki/node.js>.
- [5] Dart: <https://github.com/rikulo/socket.io-client-dart>.
- [6] Definición de socketio recuperado de <https://socket.io/docs/>.
- [7] Página oficial de flutter recuperado de <https://flutter-es.io/>.
- [8] Qué es docker (software) recuperado de [https://es.wikipedia.org/wiki/docker\\_\(software\)](https://es.wikipedia.org/wiki/docker_(software)).
- [9] Dependencia sharedpreference recuperado de [https://pub.dev/packages/shared\\_preferences](https://pub.dev/packages/shared_preferences).
- [10] Middleware morgan recuperado de <https://www.npmjs.com/package/morgan>.
- [11] Middleware chalk recuperado de <https://www.npmjs.com/package/chalkn>.
- [12] Middleware helmet recuperado de <https://www.npmjs.com/package/helmet>.
- [13] Middleware body-parser recuperado de <https://www.npmjs.com/package/body-parser>.
- [14] Servidor redis recuperado de <https://redis.io/>.
- [15] Middleware chokidar recuperado de <https://www.npmjs.com/package/chokidar>.
- [16] Middleware shelljs recuperado de <https://www.npmjs.com/package/shelljs>.
- [17] Middleware compression recuperado de <https://www.npmjs.com/package/compression>.









# Apéndice A

## Anexo

En este anexo se explicará como instalar el proyecto en los diferentes modos de entorno. Contiene la misma información que el archivo “README” incluido en el archivo *zip*

### A.1. Instalación en local (Desarrollo)

#### A.1.1. Requisitos

- Node 10 o superior
- Redis Server (redis-server)
- Astrometry: <http://astrometry.net/doc/build.html#build>

#### A.1.2. Instalación de los archivos FITS para Astrometry

Para descargarse los archivos necesarios de Astrometry habrá que ejecutar los scripts de la carpeta *fitsAstrometry* indicándole en cada script la ruta donde debe guardarse esos ficheros.

```
./fitsAstrometry/getIndexFits4100.sh /usr/local/astrometry/data
```

#### A.1.3. Instalación de módulos

Una vez están todos los requisitos instalados, se procederá a ejecutar el siguiente comando:

```
1 npm install
```

### A.1.4. Instalación y ejecución de PM2

Para la instalación del módulo

```
1 sudo npm install pm2 -g
```

La configuración del servicio en la ejecución, se puede ver en el archivo **pm2.json**. En este archivo se puede cambiar tanto las instancias a ejecutar como el puerto.

```
1 pm2 start pm2.json
```

## A.2. Instalación en local (Producción)

Para una mayor seguridad y minimizado de archivos, se puede crear un servicio que se puede ejecutar en producción.

### A.2.1. Requisitos

- Node 10 o superior
- Redis Server (redis-server)
- Astrometry: <http://astrometry.net/doc/build.html#build>

### A.2.2. Instalación de módulos

Una vez están todos los requisitos instalados, se procederá a ejecutar el siguiente comando:

```
1 npm install
```

### A.2.3. Creación de los archivos en producción

```
1 npm run-script build
```

Esto genera una carpeta *dist* donde estarán todos los archivos necesarios para ejecutar el servicio.

### A.2.4. Instalación de los archivos FITS para Astrometry

Para descargarse los archivos necesarios de Astrometry habrá que ejecutar los scripts de la carpeta *fitsAstrometry*, dentro de la carpeta *dist*, indicándole en cada script la ruta donde debe guardarse esos ficheros.

```
./fitsAstrometry/getIndexFits4100.sh /usr/local/astrometry/data
```

### A.2.5. Instalación de módulos

Una vez están todos los requisitos instalados, se procederá a ejecutar el siguiente comando, dentro de la carpeta *dist*:

```
1 npm install
```

### A.2.6. Instalación y ejecución de PM2

```
1 sudo npm install pm2 -g
```

La configuración del servicio en la ejecución, se puede ver en el archivo **pm2.json**. En este archivo se puede cambiar tanto las instancias a ejecutar como el puerto.

```
1 pm2 start pm2.json
```

## A.3. Instalación en Docker

Para crearnos el contenedor en Docker, es necesario NodeJS para instalar los módulos de desarrollo y construir la carpeta *dist*.

### A.3.1. Requisitos

- Node 10 o superior
- Docker

### A.3.2. Instalación de módulos

Una vez están todos los requisitos instalados, se procederá a ejecutar el siguiente comando:

```
1 npm install
```

Creación de los archivos en producción

```
1 npm run-script build
```

Esto genera una carpeta **dist** donde estarán todos los archivos necesarios para ejecutar el servicio y los que se usaran en la imagen del Docker.

### A.3.3. Creación de la imagen en Docker

```
1 docker build -t find-your-star .
```

Este comando leerá el archivo *Dockerfile* y ejecutará cada comando dentro de él para construir una imagen de nuestra aplicación.

### A.3.4. Ejecución de Docker con Docker-Compose

Para que la aplicación funcione correctamente es necesario del servicio **server-redis**, por tanto el docker-compose ejecutará tanto el servicio Redis como la aplicación.

En el archivo *docker-compose.yml* se pueden redirigir tanto el puerto de la web (**80**) como el puerto del socketIO (**3000**) y también cambiar la ruta donde se van a guardar las imágenes que se realizarán con la cámara.

```
1 ports:
2     "<puerto-socket>:3000"
3     "<puerto-web>:80"
4 volumes:
5     "<ruta-imagenes>:/usr/local/etc/tiempo\_real"
```

```
1 docker-compose up
```