Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Análisis de Viabilidad para Contadores de Aforo

Feasibility Analysis for People Counters

Daniel Nuez Wehbe

La Laguna, 10 de junio de 2021

D. **Francisco Javier Rodríguez González**, con N.I.F. 43.618.712-V profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Alenjandro Pérez Nava**, con N.I.F. 43.821.179-S profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Análisis de Viabilidad para Contadores de Aforo*”

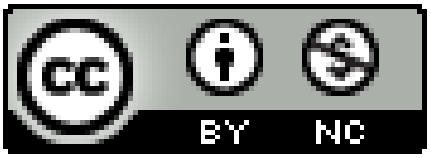
ha sido realizada bajo su dirección por D. **Daniel Nuez Wehbe**, con N.I.F. 51.147.242-A.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de junio de 2021

Agradecimientos

En primer lugar, me gustaría agradecer a mi familia y amigos, por apoyarme a lo largo de estos años. Pero especialmente me gustaría agradecer a Elena por su ayuda y apoyo incondicional, y a mi tutor Francisco. Gracias por haberme ofrecido la libertad y el apoyo necesarios para poder llevar a cabo este proyecto. Sin ninguno de ellos no habría conseguido llegar hasta aquí.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial 4.0 Internacional.

Resumen

Debido a la situación actual a la que nos hemos enfrentado en el año 2020, cada vez es más necesario mantener un control en los aforos de los establecimientos. Pero consideramos que este control debe llevarse a cabo de la forma más óptima posible. Por ello se ha buscado llevar a cabo este estudio de viabilidad para comprobar si es factible crear un producto que satisfaga estas necesidades, buscando seguir las principales tendencias actuales del mercado: el Internet de las Cosas y el Big Data.

Partiendo de esta situación inicial, se ha realizado un estudio de diferentes opciones tanto de hardware, como de tecnologías y herramientas líderes en el mercado, el cual ha dado lugar a la implementación e interconexión de dos arquitecturas. Por un lado, encontramos un contador de aforo, basado en ***Python*** *y* ***OpenCV****, ejecutándose en una Raspberry Pi junto a una cámara USB. Mientras que por el otro, se encuentra una aplicación web que utiliza las siguientes tecnologías:* ***React****,* ***NestJS****,* ***TypeScript****,* ***GraphQL****; la cual permite acceder a la información que genere el contador de una manera ágil y sencilla, permitiendo también la administración de los diferentes usuarios que tengan acceso a la misma.*

En Como conclusión, hemos demostrado la total viabilidad de un proyecto de estas características. La duración estimada del mismo es de unos ocho meses, y el retorno de inversión se obtendría a los diez meses de comercialización. Además, la solución ofertada permite una instalación sencilla, un precio competitivo y una gran escalabilidad.

Palabras clave: Contador, Aforo, Internet de las Cosas, Big Data, Arquitectura, Interconexión.

Abstract

Due to the current situation that we faced in 2020, it is increasingly necessary to maintain control over the capacity of the establishments. Nevertheless, we consider that this control must be carried out in the most optimal way possible. For this reason, this feasibility study has been carried out to check whether it is viable to create a product that meets these needs, seeking to follow the primary market trends: the Internet of Things and Big Data.

Starting from this initial situation, we have studied different hardware options, leading technologies and tools in the market, which has led to the implementation and interconnection of two architectures. On the one hand, we find the people counter, based on ***Python*** *and* ***OpenCV****, running on a* ***Raspberry Pi*** *together with a* ***USB camera****. While on the other, there is a web application that uses the following technologies:* ***React****,* ***NestJS****,* ***TypeScript****,* ***GraphQL****; which allows access to the information generated by the meter in an agile and simple way, also allowing the administration of the different users who have access to it.*

In conclusion, we have demonstrated the total viability of a project of these characteristics. Its estimated duration is about eight months, and the return on investment would be obtained after ten months of commercialization. In addition, the offered solution allows a simple installation, a competitive price and significant scalability.

Keywords: Counter, Capacity, Internet of Things, Big Data, Architecture, Interconnection.

Índice general

1. **Introducción** **1**
   - 1.1. Definición del problema . . . 1
   - 1.2. Justificación . . . 1
   - 1.3. Tendencias del mercado . . . 2
     - 1.3.1. Internet de las Cosas . . . 2
     - 1.3.2. Big Data . . . 2
   - 1.4. Objetivos . . . 4
2. **Estudio previo** **6**
   - 2.1. Contador de aforo . . . 6
   - 2.2. Aplicación Web . . . 9
3. **Desarrollo de arquitecturas** **13**
   - 3.1. Contador de Aforo . . . 13
     - 3.1.1. Algoritmo de conteo . . . 13
     - 3.1.2. Configuración del hardware . . . 19
   - 3.2. Aplicación Web . . . 22
     - 3.2.1. Servidor Backend . . . 22
     - 3.2.2. Frontend . . . 26
     - 3.2.3. Base de Datos . . . 32
   - 3.3. Interconectar arquitecturas . . . 32
4. **Estudio de viabilidad económica** **34**
   - 4.1. Desarrollo del proyecto . . . 34
   - 4.2. Modelo de Comercialización . . . 37
5. **Conclusiones y líneas futuras** **42**
6. **Summary and Conclusions** **44**
7. **Presupuesto** **46**
A. **Código del contador de aforo** **47**
   - A.1. centroidTracker.py . . . 47
   - A.2. trackableObject.py . . . 49
   - A.3. peopleCounter.py . . . 49

Índice de Figuras

1.1. Dispositivos IoT instalados mundialmente 3
1.2. Estadísticas sobre el Big Data 4

2.1. Frames por segundo utilizando los datos de prueba de COCO 7
2.2. Precisión utilizando el conjunto de pruebas de Pascal VOC 8
2.3. Frames por segundo utilizando el conjunto de pruebas de Pascal VOC 8
2.4. Popularidad de los frameworks client-side 10
2.5. Ejemplo de un gráfico de datos de GraphQL 12

3.1. Ejemplo de ejecución del algoritmo 14
3.2. Diagrama de flujo de la lógica del método Update 16
3.3. Diagrama de flujo del bucle principal del programa 18
3.4. Menú de configuración de la Raspberry Pi 19
3.5. Menú de configuración de la Raspberry Pi, aumentar el espacio 20
3.6. Menú de configuración de la Raspberry Pi, mensaje de confirmación 20
3.7. Página de Login versión Web 27
3.8. Dashboard versión Web 27
3.9. Tabla de usuarios versión Web 28
3.10. Formulario para añadir un usuario 29
3.11. Formulario para editar un usuario 30
3.12. Mensaje de confirmación para eliminar un usuario 30
3.13. Dashboard versión Web 31
3.14. Mensaje de confirmación al crear un usuario 31
3.15. Mensaje de error al iniciar sesión 31
3.16. Tablas generadas 32
3.17. Tabla de Usuarios 32

4.1. Diagrama de Gantt 35
4.2. Costo de desarrollo a lo largo del tiempo 37
4.3. Paquetes de suscripción 38
4.4. Retorno de Inversión 40

5.1. Porcentaje de acierto del algoritmo implementado 42
5.2. Rendimiento del algoritmo implementado 43

Índice de Tablas

4.1. Supuesto de ventas a lo largo del tiempo 40

7.1. Presupuesto del proyecto 46

Capítulo 1

Introducción

1.1. Definición del problema

Durante la última década no se ha prestado especial interés al control de aforo en los establecimientos o recintos. Principalmente la realización de este tipo de controles ha estado ligado a bienes digitales, como podrían ser entradas a eventos como ferias o conciertos. Aunque, en ciertas comunidades autónomas, si que es obligatorio[1] contar con dispositivos de conteo, ya que es una forma de garantizar la seguridad a fin de evitar aglomeraciones y colas en caso de una emergencia.

A lo largo del año 2020 se ha incrementado la necesidad de limitar los aforos de los locales y establecimientos, además de realizar un control de acceso más exhaustivo para no sobrepasar este límite de capacidad de personas por metro cuadrado. Para llevarlo a cabo, muchas empresas optan por la solución más sencilla, que es el uso de personas para realizar un conteo de los clientes que entran y salen de los establecimientos.

Esto supone algunos problemas, sobre todo para grandes superficies, donde realizar este control de esta manera supone la contratación de más personal dedicado exclusivamente a esta tarea, y, si además se cuenta con diferentes accesos al recinto, esto requiere la coordinación de los mismos.

1.2. Justificación

Para realizar el control de aforo de la manera más óptima posible, lo mejor es contar con dispositivos que realicen un conteo automático, y permitan a uno o varios usuarios la visualización de las personas que entran y salen del establecimiento o recinto.

Sin embargo las principales soluciones que ofrece actualmente el mercado son, o dispositivos que realizan un conteo muy básico, o sistemas con un precio demasiado elevado.

Por tanto, el presente trabajo de investigación académica pretende realizar un estudio de herramientas y tecnologías de última generación y a futuro, con las que desarrollar un prototipo de aplicativo que permita cubrir las necesidades actuales del mercado, pivotando sobre los nichos de mercado que consideran más importantes las consultoras, que son: el **Internet de las Cosas (IoT)** y el **Big Data**[2]. Algunas de las principales necesidades identificadas son:

[1] Véase BOE: `https://www.boe.es/diario_boe/txt.php?id=BOE-A-2006-21359`

[2] Macrodatos

- Precios competitivos.
- Fácil acceso a los datos generados.
- Elaboración de estadísticas.
- Fácil instalación y escalabilidad.

1.3. Tendencias del mercado

Como se ha comentado, tras realizar un estudio sobre el estado actual del mercado, se han identificado dos tendencias que están cobrando mucha importancia estos últimos años: el Internet de las Cosas y el Big Data.

1.3.1. Internet de las Cosas

El IoT se refiere a un sistema de objetos físicos conectados a través de Internet, que pueden transmitir datos a otros dispositivos y sistemas sin ningún tipo de intervención manual [32, 26].

Los principales beneficios que aporta el IoT a los negocios son:

- Mejoras de productividad
- Ahorro de costes
- Análisis predictivo
- Reducción de errores humanos
- Supervisión mejorada
- Experiencia de cliente mejorada
- Nuevas oportunidades de negocio

Además, está ganando mucha importancia, ya que ofrece a los negocios la posibilidad de obtener información en tiempo real sobre su estado actual y permite la automatización de numerosas actividades. Esto ha dado lugar a un incremento en la fabricación industrial impulsada por el IoT, que se conoce como Industria 4.0.

Con esta nueva industria, también conocida como la Cuarta Revolución Industrial, los productos se pueden construir con la capacidad de conectarse a Internet para compartir datos de un lado a otro. Esto podría respaldar nuevos modelos comerciales digitales que permitan a los fabricantes cambiar rápidamente qué y cómo construyen para satisfacer mejor las demandas del mercado [41].

1.3.2. Big Data

El término Big Data describe a un gran volumen de datos que pueden ser estructurados o desestructurados, y al análisis de estos para obtener información y conocimiento [32].

Para poder comprender este concepto tan amplio, el analista Doug Laney publica un artículo titulado: "3D Data Management: Controlling Data Volume, Velocity, and Variety"[17], en el que define la idea que tenemos hoy de grandes datos como las tres Vs:

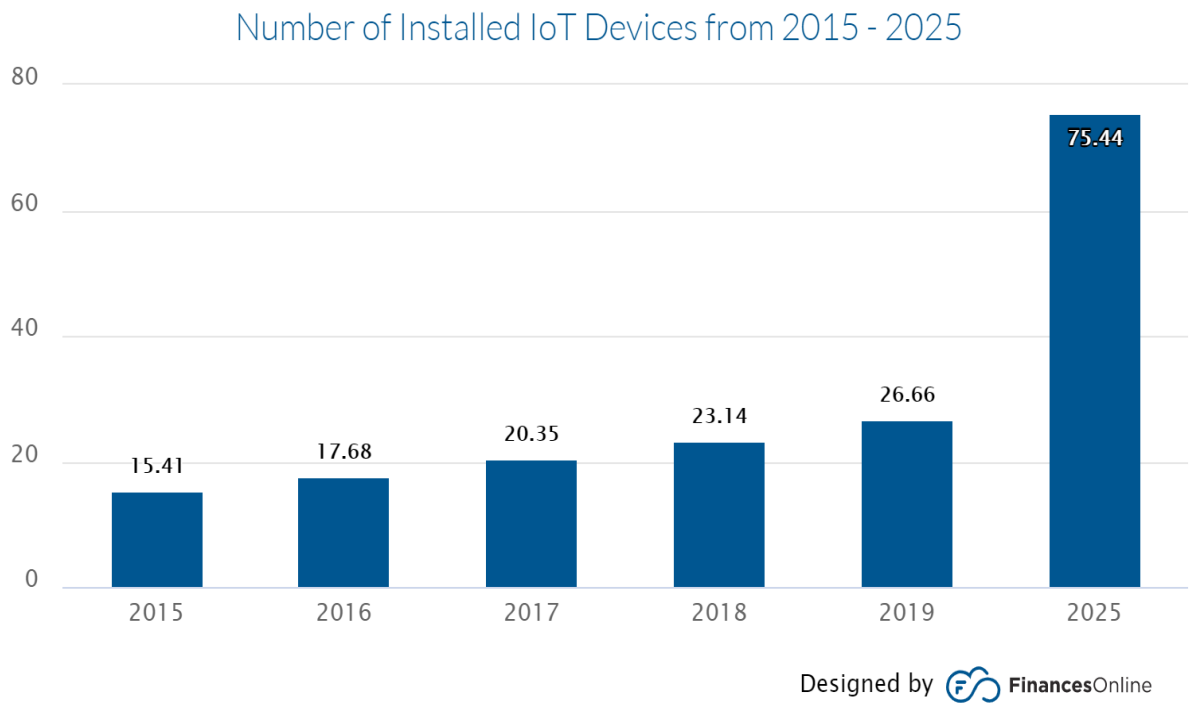


Figura 1.1: Dispositivos IoT instalados mundialmente

- Volumen: actualmente los datos pueden ser recopilados de diversas fuentes, como sensores, transacciones online o redes sociales entre muchas otras. Estos datos, que no pueden ser procesados o almacenados utilizando métodos tradicionales, suponen un problema que se ha conseguido solucionar gracias al almacenamiento distribuido. También define algunas pautas sobre qué datos deben ser recogidos y cómo hacerlo de una manera más eficiente.

- Velocidad: cada vez se incrementa más las fuentes de información de las que obtener datos. Uno de los principales causantes de ello es el Internet de las Cosas, y en consecuencia, también aumenta la velocidad necesaria para procesar y almacenar la información. Por tanto, no basta solo con disponer de un gran ancho de banda, sino que se necesitan implementar soluciones a nivel de arquitectura que permitan extraer, integrar y reorganizar los datos o utilizar **Caches**[46] que ofrezcan acceso directo a los datos de cada transacción.

- Variedad: la principal barrera para realizar una gestión de datos efectiva es que existe una enorme variedad de formatos de datos incompatibles, estructurados, no estructurados, y un largo etcétera.

Como hemos visto, el IoT juega un papel cada vez más importante en la captación de datos, ya que nos permite contar con una gran cantidad de dispositivos autónomos conectados a internet que vuelcan datos en nuestras bases de datos donde serán procesados para generar información.

"El IoT es una serie de arroyos y ríos que desembocan en el océano del Big Data"(Lirette, 2019) [18].

Pero la verdadera importancia del Big Data no reside en cuánta información somos capaces de generar, sino en cómo podemos aprovecharla para obtener ventajas competitivas en el mercado.

Figura 1.2: Estadísticas sobre el Big Data

1.4. Objetivos

El estudio realizado pretende dar respuesta a la siguiente pregunta: **¿Es viable comercializar un contador de aforo de manera competitiva?**

Una parte muy importante del proyecto será un estudio de herramientas y tecnologías disponibles en el mercado que puedan ser aprovechadas para el desarrollo, ya que la tendencia actual del mercado es desarrollar dispositivos autónomos con la capacidad de generar datos, así como obtener de manera sencilla y rápida estos datos para obtener información de ellos.

Para la elaboración del trabajo se han definido los siguientes objetivos:

- Analizar las herramientas actuales en el mercado para identificar a la competencia.
- Estudio de hardware, tecnologías y herramientas.
- Desarrollo de arquitecturas.
    - Contador de Aforo.
        - Desarrollar el algoritmo de conteo.
        - Configurar el hardware.

    - Aplicación Web para consultar los datos.
        - Implementar el **Frontend**[48].
        - Desarrollar el servidor **Backend**[48].
        - Establecer la Base de Datos.
    - Interconectar arquitecturas.
    - Testeo del correcto funcionamiento.

- Desarrollar un plan de negocio para la comercialización del producto:
    - Crear un **Diagrama de Gantt**[47] con las tareas a realizar, su duración y los recursos asignados a cada una.
    - Prever el costo total del proyecto.
    - Diseñar un modelo de comercialización del producto.
    - Calcular el **ROI**[3].

[3] Retorno de Inversión (Return Of Investment)

Capítulo 2

Estudio previo

Una parte crucial para el desarrollo del proyecto consiste en realizar un estudio previo para poder diseñar las diferentes arquitecturas implicadas para obtener el producto deseado. Por ello, se ha realizado una investigación acerca de qué hardware y qué tecnologías son las mejores opciones para implementar estas arquitecturas.

2.1. Contador de aforo

A la hora de diseñar el contador se buscó dar respuesta a las siguientes preguntas:

- **¿Qué se va a ejecutar?**
- **¿Dónde se va a ejecutar?**
- **¿Cómo va a obtener la entrada de vídeo necesaria para realizar el conteo?**

Para responder a la primera pregunta, se buscaron diferentes tecnologías y lenguajes de programación que permitieran el desarrollo de un algoritmo de conteo, recibiendo como entrada un vídeo sobre el que obtener información.

Se terminó por decidir que se utilizaría **OpenCV**[24], ya que esta es una de las mayores librerías **Open-Source**[1] de **visión por computador**, **machine learning**[2] y procesamiento de imagen que existe actualmente, además de permitir operaciones en **tiempo real**, lo cual es crucial para el correcto funcionamiento del prototipo.

Como el algoritmo ha de contar personas, debemos encontrar la forma de detectarlas en una imagen o vídeo. Existen múltiples aproximaciones para realizar esta tarea, según las necesidades y los recursos que dispongamos, nos decantaremos por una opción u otra. A continuación se exponen algunos de los modelos más utilizados para realizar esta tarea:

- **R-CNN** [10]: Es un sistema dividido en tres módulos principales. El primero de ellos, llamado **selective search**[3], divide la imagen en un gran número de regiones (alrededor de 2000), e identifica cuales de ellas son más propensas a contener un objeto. El segundo utiliza una red neuronal que hace uso de cada región generada anteriormente para realizar la toma de decisiones sobre qué puede detectarse en la imagen. Finalmente, se clasifican los resultados de cada región, respondiendo a qué objeto u objetos se han encontrado.

[1] Código abierto
[2] Aprendizaje automatizado o aprendizaje de máquinas
[3] Búsqueda selectiva

- **Fast R-CNN** [9] y **Faster R-CNN** [35]: ambas soluciones mejoran el rendimiento del sistema original. Fast R-CNN abandona la fase inicial del sistema para aumentar la velocidad, mientras que Faster R-CNN utiliza dos módulos: el primero llamado **Region Proposal Network**[4], y la salida que genera este módulo es inyectada en el otro que contiene al sistema Fast R-CNN. Este último sistema es el que mejores resultados da de estos, pero sigue siendo lento y computacionalmente muy caro.

- **YOLO (You Only Look Once**[5]**)** [34]. El enfoque dado en esta aproximación consiste en utilizar una red neuronal que predice donde se pueden encontrar los objetos en un **frame**[6], y lo hace directamente sobre la imagen realizando una única evaluación. La principal ventaja que ofrece es su gran velocidad, permitiendo el procesamiento de imagen en tiempo real. Pero su principal desventaja es que la precisión no es tan buena como en otras aproximaciones.

- **SSD (Single Shot MultiBox Detector**[7]**)** [19]. Su implementación es muy similar a la de YOLO, utilizando **bounding boxes**[8] para indicar donde pueden encontrarse los objetos de la imagen. Pero utiliza una técnica llamada **hard negative mining**[9], según la cual, aquellos cuadros delimitadores que no se encuentren en contacto con aquellos que contengan un objeto, son descartados automáticamente.

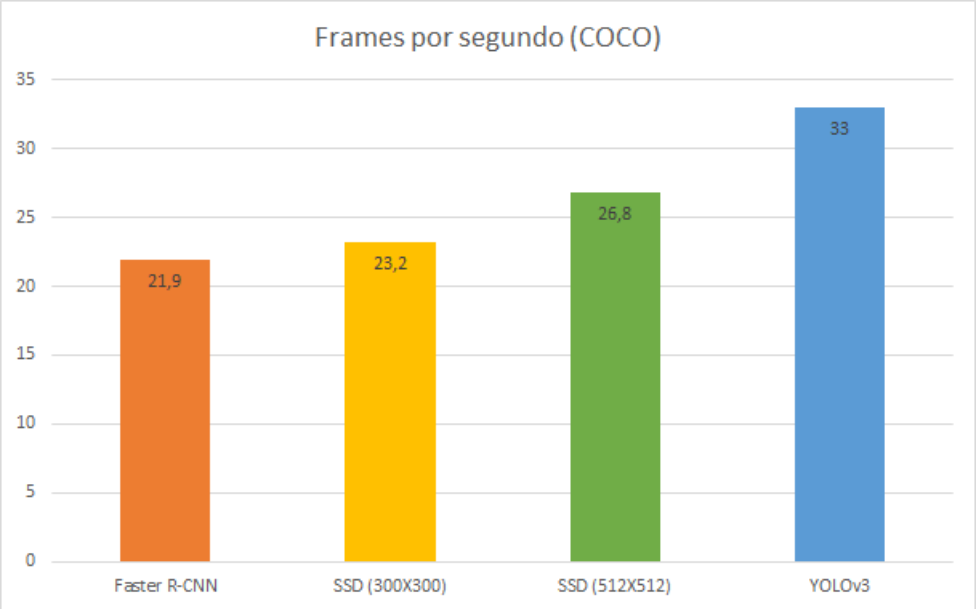


Figura 2.1: Frames por segundo utilizando los datos de prueba de COCO

[4]Red de propuestas regionales
[5]Solo miras una vez
[6]Fotograma
[7]Detector Multicaja de un solo disparo
[8]Cuadros delimitadores
[9]Minado de negativos difícil

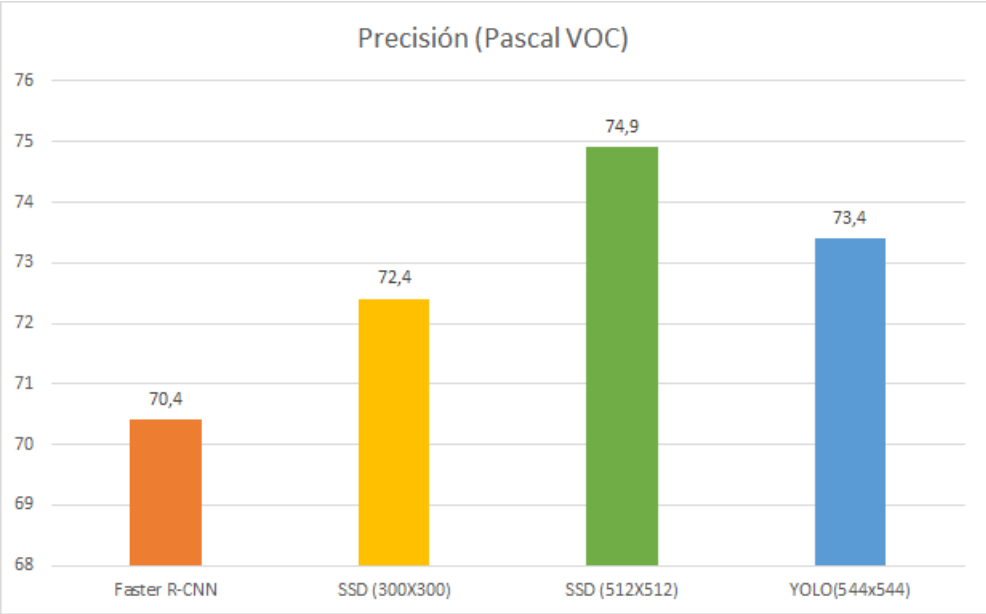


Figura 2.2: Precisión utilizando el conjunto de pruebas de Pascal VOC

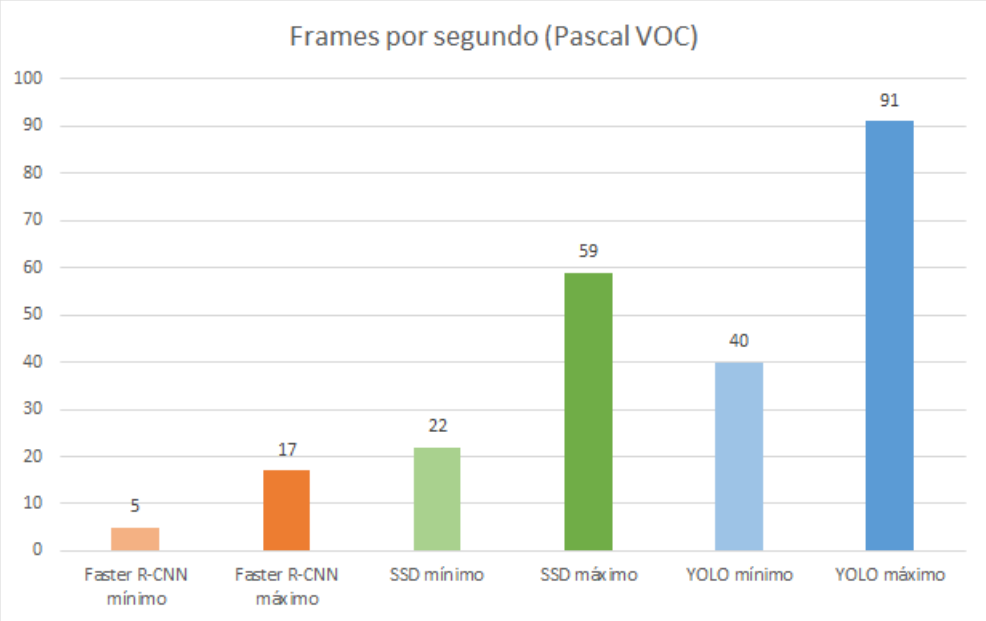


Figura 2.3: Frames por segundo utilizando el conjunto de pruebas de Pascal VOC

Todas estas opciones podrían ser perfectamente válidas para implementar nuestro algoritmo de conteo, pero todas ellas son computacionalmente muy costosas, y no disponemos de un hardware lo suficientemente potente para ejecutarlas en tiempo real. Por suerte existe una solución: el modelo **MobileNet** [12]. Ligero, de baja latencia y con un bajo consumo de energía. Ha sido diseñado para maximizar la precisión de manera efectiva sin dejar de tener en cuenta los recursos restringidos del dispositivo en el que se ejecute. Además, una de las características más interesantes para nuestro caso de uso, es que puede utilizarse junto con los modelos listados anteriormente. En nuestro caso, utilizaremos un modelo basado en MobileNet junto con SSD para maximizar la eficiencia del mismo.

Pero además, OpenCV cuenta con un módulo denominado **DNN**[10] [25], que permite cargar un modelo previamente entrenado. Por tanto, solo necesitamos uno para poder ejecutarlo en nuestro algoritmo.

En cuanto al lenguaje de programación, existían dos opciones a tener en cuenta: **Python** [28] y **C++** [45]. Existen numerosas comparativas [33] entre ambos lenguajes y su desempeño con la librería OpenCV. Finalmente se decidió utilizar Python, ya que es un lenguaje que permite una codificación sencilla y porque a día de hoy ofrece librerías que ejecutan código hecho en **C** [44] o C++, lo cual mejora la eficiencia del programa.

Por otro lado, es muy importante realizar una correcta elección del hardware donde se va a ejecutar el algoritmo de conteo, ya que se busca la eficiencia y simplicidad de instalación del producto. Algunas de las opciones barajadas han sido:

- Rock Pi 4 B.
- Raspberry Pi 4B.
- Asus Tinker board.
- HiKey960.
- Minnowboard Turbot.
- NVIDIA Jetson Nano Developer Kit.

Finalmente se optó por utilizar la **Raspberry Pi 4B**[11] con 4gb de memoria ram, que utiliza el sistema operativo **Raspberry Pi OS**[12], porque nos ofrece un hardware muy adecuado a un precio reducido, y además cuenta con un gran soporte de tecnologías y una gran comunidad. El principal problema al que podríamos enfrentarnos con esta elección es las altas temperaturas que puede llegar a sufrir el procesador, pero existe una gran variedad de carcasas con ventilación para suplirlo.

Por último, para que el algoritmo pueda contar con un input de vídeo en directo, se intentó utilizar una cámara Wifi, pero por motivos de incompatibilidad se tuvo que descartar su uso. Para evitar este posible problema, se ha optado por el uso de una cámara USB estándar, aunque existe una cámara Wifi compatible desarrollada por los propios creadores de la Raspberry que podría ser utilizada.

2.2. Aplicación Web

Del mismo modo que en el apartado anterior, podemos dividir el estudio de la aplicación web en las tres partes que conforman una aplicación web:

- Frontend.
- Backend.
- Base de Datos.

[10] Nueva Red Neuronal Profunda
[11] `https://www.raspberrypi.org/products/raspberry-pi-4-model-b/`
[12] `https://www.raspberrypi.org/software/`

Actualmente, existen numerosos **frameworks**[13] que nos ofrecen colecciones de librerías y módulos para el desarrollo de aplicaciones.

Existen dos tipos de frameworks de desarrollo:

- **Client-side**[14]: constituyen la parte visual de la aplicación.
- **Server-side**[15]: se ejecutan sin ser vistos para asegurar el correcto funcionamiento de la aplicación web.

Los principales frameworks client-side son: **Angular[2]**, **React[29]** y **Vue[42]**.

Se ha decidido trabajar con React debido a que, aparte de ser el más popular de ellos, es un framework muy flexible, tiene una curva de aprendizaje mucho más sencilla que Angular, es Open-Source, facilita una creación de componentes reutilizables, permite el renderizado en el lado del servidor si fuera necesario, utiliza **HTML** [49] junto con **JSX** [15] para renderizar los componentes, etc. Pero la característica más interesante es que tiene un 100 % de retrocompatibilidad, por tanto una aplicación siempre podrá funcionar aunque se actualice la versión.

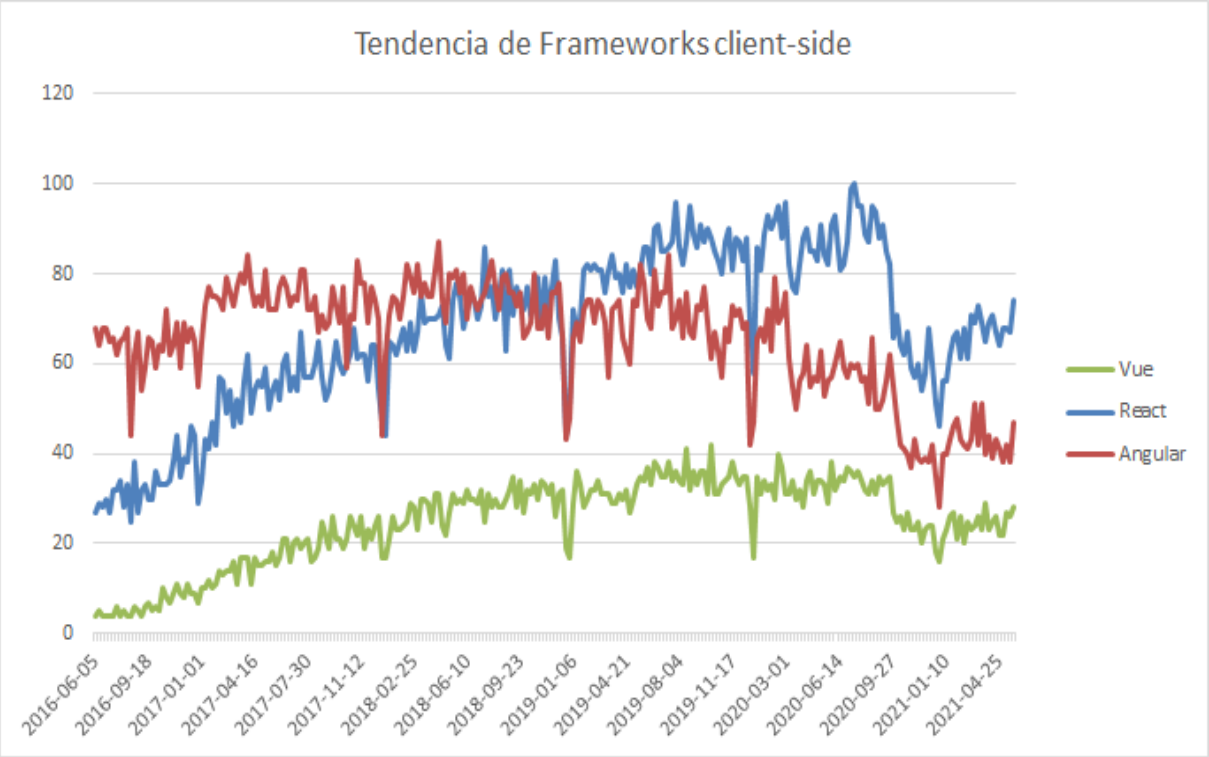


Figura 2.4: Popularidad de los frameworks client-side

En cuanto a la elección del framework que se ejecutará en el backend de la aplicación, se ha optado por utilizar **NestJS** [22], el cual permite crear servidores fácilmente escalables ya que tiene una estructura basada en módulos independientes, utiliza **TypeScript** [40], cuenta con una documentación muy extensa, es Open-Source, tiene soporte con múltiples tecnologías de manera nativa y permite una orientación monolítica o de microservicios y está basado en **NodeJS** [23].

[13] Entorno de trabajo
[14] Lado del cliente
[15] Lado del servidor

La base de datos implementada ha sido **PostgreSQL** [27]. Es una **base de datos relacional** [43], Open-Source, que permite el manejo de una gran cantidad de datos y operaciones simultáneas, cuenta con un sistema de base de datos objeto-relacional que le proporciona características de la **POO**[16] y las versiones son a cinco años, es decir, garantizan la actualización de una versión durante ese periodo de tiempo.

Como alternativas se han estudiado tanto **MySQL** [21] como **MariaDB** [20]. La primera se descartó ya que cuenta con una versión Community gratuita limitada, mientras que el resto de versiones dependen de una suscripción anual; y la segunda también fue descartada, debido a que PostgreSQL ofrece una mayor escalabilidad y mejor rendimiento cuando se trabaja con un gran volumen de datos y de consultas.

Para comunicar el frontend de la aplicación con el backend se ha utilizado **GraphQL** [11] en lugar de la típica **API Rest** [36]. Esta decisión viene dada porque GraphQL nos ofrece algunas mejoras respecto a las APIs Rest:

- No trata con recursos dedicados. Los datos forman un conjunto de grafos conectados entre sí, lo cual nos permite adaptar las consultas a nuestras necesidades, indicando qué datos queremos obtener y combinar diferentes entidades o tipos y, por tanto, no tiene problemas de **Over Fetching**[17] y **Under Fetching**[18].
- Mientras que una API Rest expone múltiples **endpoints**[19], GraphQL solo publica uno, que por defecto se llama */graphql*.
- Utiliza dos métodos denominados **Query** y **Mutation** basados en el método **POST**. El primero solo permite acceder a los datos en modo lectura, mientras que el segundo permite la edición de los mismos.
- etc.

GraphQL utiliza un esquema para describir la forma de su **gráfico de datos** (2.5), con el que representa la información y define una jerarquía de tipos.

Además se utilizará la plataforma **Apollo** [3], que implementa GraphQL para conectar nuestro servidor con el cliente y está diseñado principalmente para trabajar con React. Apollo nos ofrece un ecosistema muy completo que permite, en el lado del cliente, extender funcionalidades con un gran número de extensiones y la posibilidad de crearlas, el uso de **Hooks** [30] para encapsular la obtención datos, el control de la carga de los mismos y el manejo de errores, permitiendo una integración muy sencilla. Cuenta con una caché funcional que no requiere de ningún tipo de configuración para poder ser usada, aunque permite personalizarla. Otro de los aspectos destacables es el uso de diferentes políticas para obtener datos, de las cuales se pueden resaltar:

- Primero caché, y si no existen datos entonces los obtendrá de la red.
- Solo red, aunque escribe el resultado en la caché.
- Caché y red, lee los datos de la caché y si la respuesta de la red es diferente, la actualiza.

[16] Programación Orientada a Objetos
[17] Sobre búsqueda. Ocurre cuando se envían más datos de los necesarios
[18] Infra búsqueda. Ocurre cuando se envían menos datos de los necesarios
[19] Punto final de comunicación

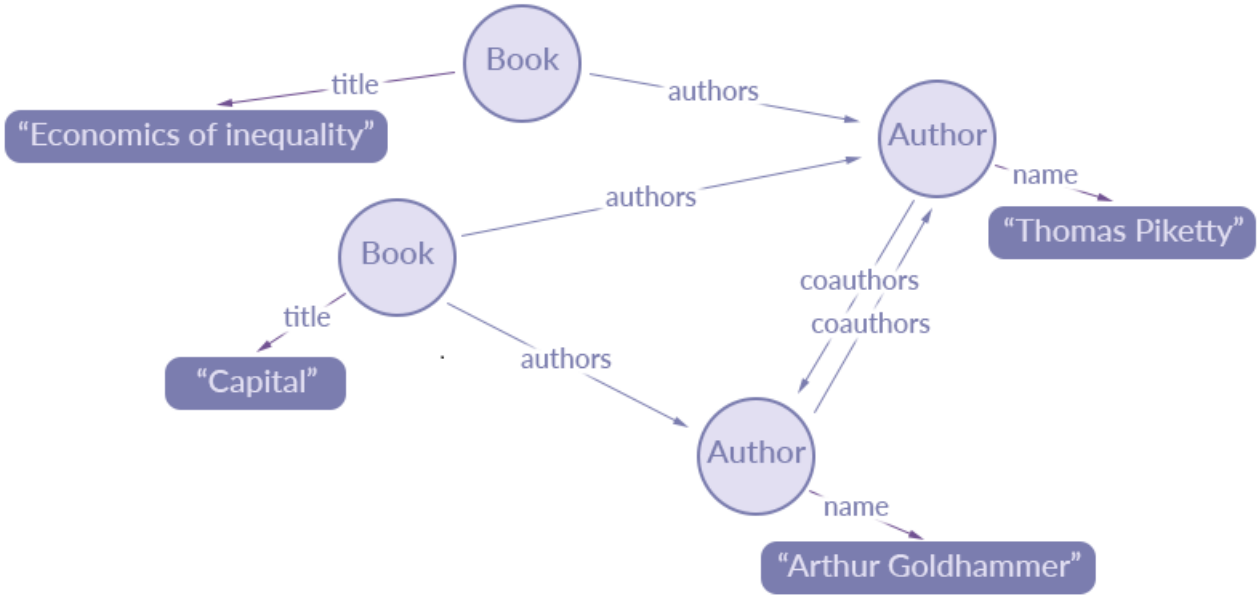


Figura 2.5: Ejemplo de un gráfico de datos de GraphQL

Por parte del servidor, Apollo nos ofrece compatibilidad universal. En caso de contar con un servidor antiguo que utiliza Rest, **SOAP** [52] u otro protocolo, permite ejecutar GraphQL sobre ellos. Además de que funciona con prácticamente cualquier framework de backend e implementar una caché similar a la del frontend.

La **UI**[20] también juega un papel muy importante en el desarrollo de cualquier software, pero debido a que se busca que esta aplicación sea lo más accesible posible, incluso desde dispositivos móviles, se ha tomado la decisión de utilizar el framework **Ionic[13]**, el cual permite desarrollar UIs responsivas para adaptarse a cualquier tipo de pantalla, además de personalizarse si el usuario utiliza un dispositivo **IOS** [14] o **Android** [1]. Se consideró utilizar **React Native** [31], pero se descartó ya que después de realizar una comparativa, aparentemente Ionic permitía generar la aplicación para IOS sin la necesidad de utilizar un computador con **macOS** [50]. Después de un tiempo de desarrollo se descubrió que esto no era cierto, y ya cambiar de framework no era una posibilidad.

Para finalizar, cabe destacar otras tecnologías que también se han utilizado:

- **SASS**[39], una extensión para **CSS** [7].
- **RxJS** [38], una librería que permite utilizar eventos asíncronos para comunicar componentes entre ellos.
- **ChartJS** [6], permite la creación de gráficos de manera sencilla.
- **JWT** [16], un estándar Open-Source que permite implementar una capa de autenticación en nuestras aplicaciones para realizar conexiones seguras.
- **Axios** [4], un cliente **HTTP** [51] que utiliza **promesas** [8].
- **Bcryptjs** [5], una pequeña librería que permite encriptar y desencriptar las contraseñas de los usuarios para almacenarlas en la BDD.

[20] Interfaz de Usuario

Capítulo 3

Desarrollo de arquitecturas

3.1. Contador de Aforo

Como se ha comentado anteriormente, se va a utilizar el módulo DNN de OpenCV, con un modelo previamente entrenado que combina MobileNet junto con SSD. Pero además es necesario mencionar que se ha utilizado el siguiente artículo de la página **pyimagesearch** titulado **OpenCV People Counter** [37].

3.1.1. Algoritmo de conteo

El comportamiento general del algoritmo es el siguiente:

1. Obtenemos el siguiente frame disponible, lo reescalamos y, debido a que la captura de imagen utiliza el formato de colores BGR[1], debemos convertirlo a RGB[2] para que pueda ser procesado de manera correcta.

2. Debido a que debemos optimizar al máximo el algoritmo, se deberá elegir entre la ejecución de dos fases:

   - Detección: durante esta fase, se identificarán a las nuevas personas que hayan aparecido en la imagen o frame, y mantendremos a quiénes ya habían sido detectados sin identificarlas como nuevas personas. Durante esta fase se ejecutará el modelo pre-entrenado para detectar a las personas y, al ser la parte del algoritmo más costosa, solo se ejecutará cada N-frames.
   - Seguimiento: cuando no estamos en la fase de detección, estaremos realizando un seguimiento de las personas que hemos detectado. Para cada una de ellas, generaremos un **tracker**[3] que seguirá a la persona mientras esta se mueve por la imagen. Siempre que no se esté en fase de Detección, se ejecutará el seguimiento.

3. Una vez se termine la fase elegida, se trazará una línea horizontal en la mitad de la imagen para determinar si las personas detectadas se están desplazando hacia arriba o hacia abajo.

[1] Blue Green Red
[2] Red Green Blue
[3] Rastreador o seguimiento

4. A continuación, se comprobará si se ha detectado un nuevo objeto, si se ha perdido o si ha vuelto a aparecer para continuar realizando el seguimiento, o si se ha de eliminar algún objeto que se haya perdido definitivamente.

5. Por último, se dibujará un punto junto con un identificador en la zona de la imagen donde se encuentre la persona, y se actualizará el contador.

6. Repetimos todo el proceso.

Figura 3.1: Ejemplo de ejecución del algoritmo

Las clases utilizadas para implementar este comportamiento son: **CentroidTacker (A.1)** y **TrackableObject (A.2)**; y el fichero **PeopleCounter (A.3)**.

CentroidTracker

Esta clase describe como ha de calcularse cada **Centroid**[4], que, como su nombre indica, apunta al centro de un objeto y mantendrá un registro de aquellos que estén siendo **rastreados**.

Inicialmente tiene definidos los siguientes atributos:

- nextObjectId: Identificador único que será asignado al siguiente objeto detectado.
- objects: Un diccionario que almacenará los IDs de los objetos que tengamos registrados.
- disppeared: Un diccionario que almacenará los IDs de los objetos que hayan sido marcados como perdidos.
- maxDisappeared: Número máximo de frames consecutivos que un objeto puede estar marcado como perdido antes de ser descartado.
- maxDistance: Cuando recibamos nuevos objetos, debemos revisar que no sea alguno de los que ya tenemos registrado, pero que su posición se ha modificado de un frame a otro. Por ello debemos establecer una distancia máxima para considerar cuando un objeto es el mismo, pero desplazado.

[4] Centroide

Además, cuenta con tres métodos que describen el comportamiento de la clase. Los dos primeros tienen un comportamiento muy sencillo y estrechamente relacionado:

- **Register**. Recibirá un nuevo centroide que no está siendo rastreado para asignarle un nuevo identificador.
- **Deregister**. Cuando un centroide haya estado desaparecido durante más tiempo del permitido se procederá a eliminarlo de los diccionarios.

Por último, encontramos el método **Update**, el más importante de los tres, ya que este engloba toda la lógica de esta clase. Recibe como entrada los objetos que hayan sido detectados en forma de bounding boxes[5]. La lógica del método está representada en la figura *Diagrama de flujo de la lógica del método Update (3.2)*.

TrackableObject

Es una pequeña clase que representará un objeto al que se le puede realizar seguimiento. Almacenará el Id que lo identifica, una lista que almacene los diferentes centroides que vaya generando dicho objeto mientras se desplaza, y un booleano que indica si el objeto ya fue contado o no.

PeopleCounter

Por último, nos encontramos con el fichero principal del algoritmo. Este hará uso de los otros dos para realizar el conteo de personas. Debido a que más adelante se hablará de la conexión entre la arquitectura del contador de aforos con la de la aplicación web en la sección **Interconectar arquitecturas**, se obviarán aquí las diferentes partes en las que se realiza dicha conexión.

Inicialmente se han definido los argumentos que recibirá por línea de comandos, algunos de ellos son opcionales y tienen valores por defecto:

- "*–prototxt*": contiene la ruta del fichero prototxt que incluye la estructura de la red neuronal.
- "*–model*": contiene la ruta del modelo pre-entrenado.
- "*–input*": si se especifica, contendrá la ruta de un vídeo que será utilizado como entrada para el algoritmo en lugar de utilizar una cámara web.
- "*–output*": si se especifica, se realizará una captura de vídeo de la ejecución del algoritmo, y se almacenará en la ruta especificada junto a esta opción.
- "*–confidence*": es opcional, e indica cual es el porcentaje mínimo de confianza para poder filtrar detecciones que no sean del todo claras. Por defecto es 0.4.
- "*–skip-frames*": número de frames que deben pasar antes de ejecutar la fase de detección. Si no se especifica, son 30 por defecto.
- "*–test*": sirve para indicar si el algoritmo debe comunicarse con la aplicación web para enviar los datos o no. Por defecto los enviará.

[5] Cuadros delimitadores

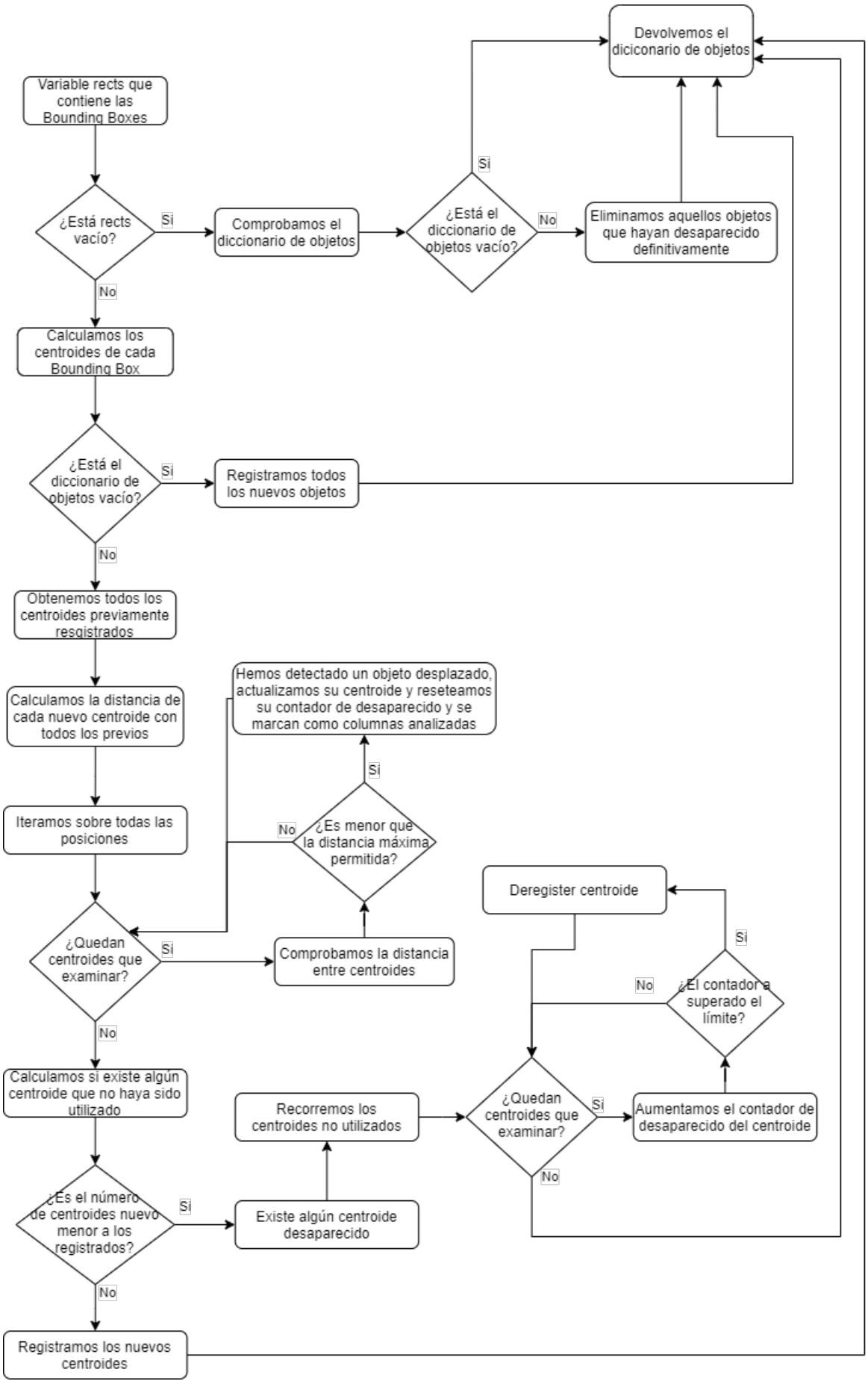


Figura 3.2: Diagrama de flujo de la lógica del método Update

Una vez ejecutado el algoritmo, cargará los datos desde las rutas especificadas, y abrirá el fichero de entrada, o encenderá la cámara web para realizar la captura de vídeo. Se declaran las variables a utilizar durante la ejecución:

- Se inicializa el CentroidTracker indicando la distancia máxima entre centroides y el número máximo de frames que un centroide puede estar desaparecido.
- Un array que almacenará los trackers[6] que se vayan creando.
- Un diccionario para mapear los identificadores únicos de cada TrackableObject.
- Una lista llamada *rects*, que almacenará las Bounding Boxes que se generen.
- Diversos contadores para saber el número de personas que pasa por la imagen hacia arriba o hacia abajo, y el número de frames totales procesados.

La ejecución del bucle principal del programa tiene el siguiente flujo:

[6] Rastreadores o seguimientos

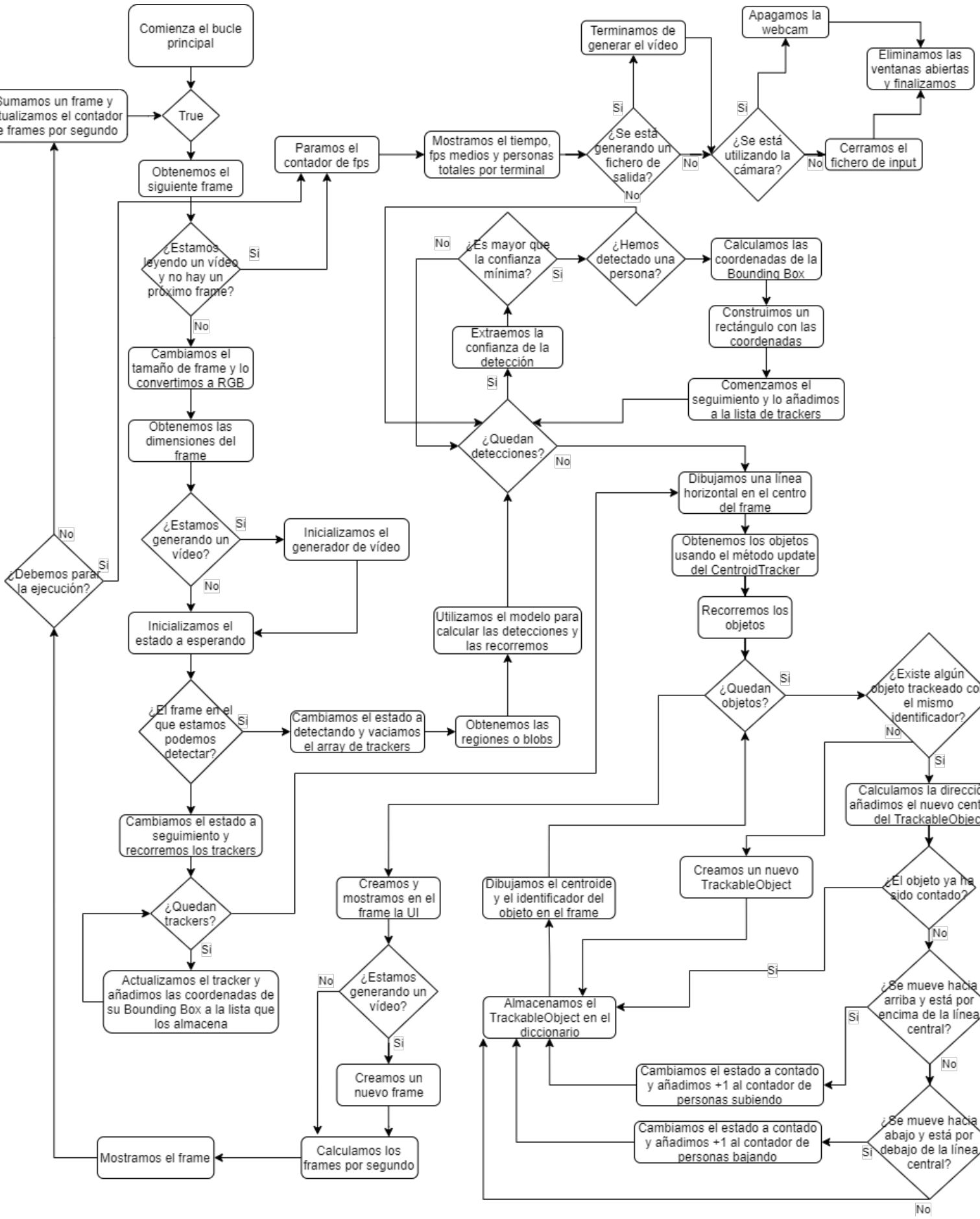


Figura 3.3: Diagrama de flujo del bucle principal del programa

3.1.2. Configuración del hardware

Para comenzar, se instalará el sistema operativo de nuestra Raspberry Pi. Para ello, nos dirigiremos a la página oficial[7] para descargar el *Raspberry Pi Imager* y, una vez se haya ejecutado, seleccionaremos como sistema operativo Raspberry Pi OS (32bits), o su versión Lite. La diferencia entre ambas versiones es que la reducida no tiene escritorio. La Raspberry Pi utiliza una tarjeta microSD (en nuestro caso de 32GB) como almacenamiento secundario, por lo que instalaremos el sistema operativo en ella. Cuando termine de ejecutarse el Raspberry Pi Imager, ya podremos introducir la tarjeta microSD en nuestra Raspberry y encenderla.

Durante el primer arranque es recomendable conectar la Raspberry por cable ethernet, ya que realiza algunas configuraciones iniciales automáticas, y otras en las que el usuario debe intervenir. Una vez terminadas, comenzaremos a configurar el dispositivo[8]. Lo primero que haremos será aumentar el espacio del sistema de archivos, ya que por defecto no se habrá ocupado todo el tamaño de la tarjeta. Para ello ejecutaremos el comando:

```
1  $ sudo raspi-config
```

Nos aparecerá un menú de opciones como el siguiente, y elegiremos la opción *6 Advanced Options*:

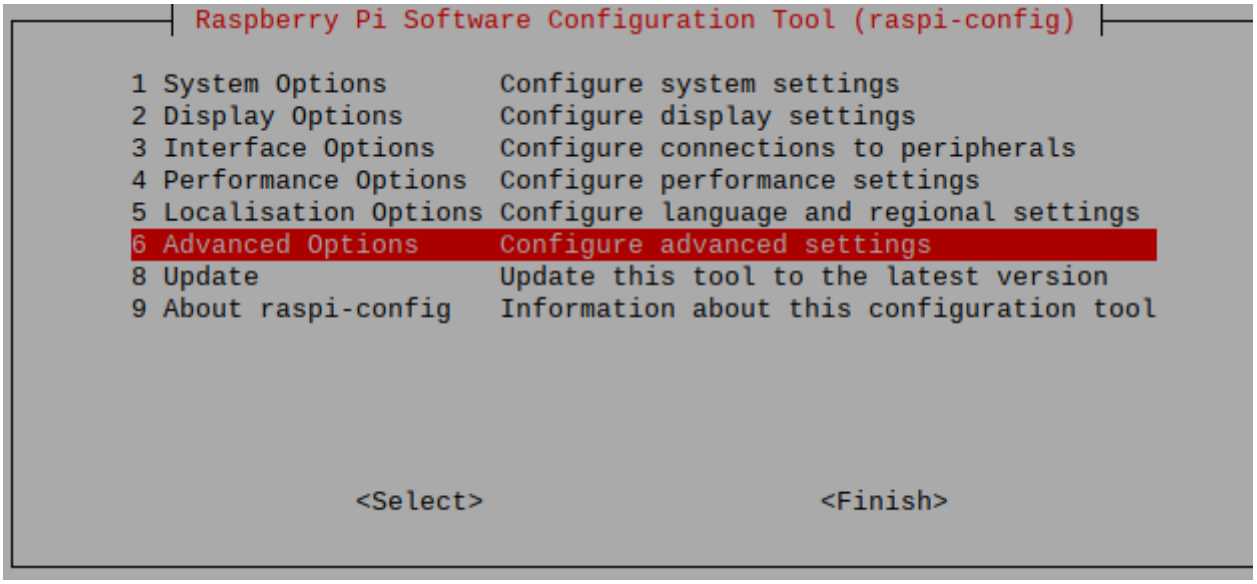


Figura 3.4: Menú de configuración de la Raspberry Pi

Marcaremos ahora la primera opción *A1 Expand Filesystem*:

[7] https://www.raspberrypi.org/software/

[8] https://www.pyimagesearch.com/2019/09/16/install-opencv-4-on-raspberry-pi-4-and-raspbian-buster/

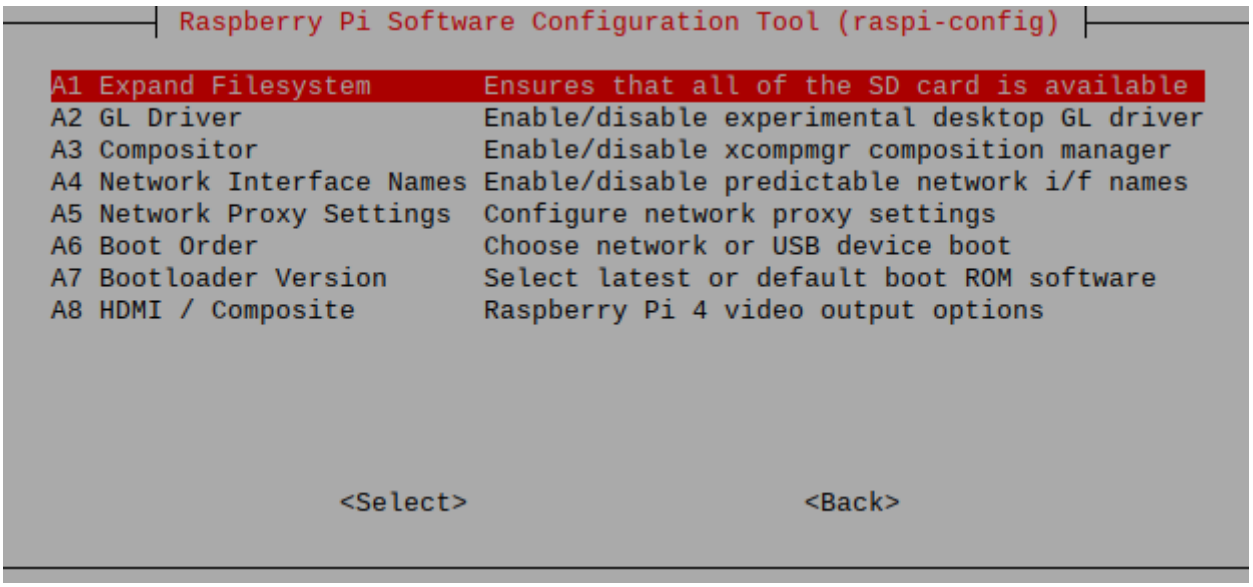


Figura 3.5: Menú de configuración de la Raspberry Pi, aumentar el espacio

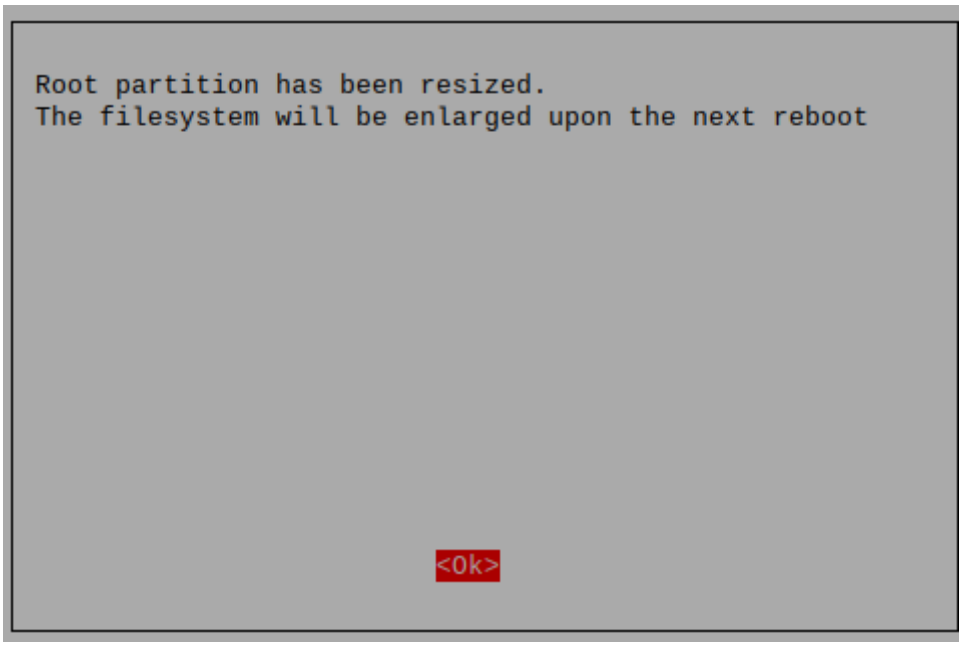


Figura 3.6: Menú de configuración de la Raspberry Pi, mensaje de confirmación

Una vez hayamos terminado, debemos reiniciar la Raspberry, ya sea desde la interfaz de usuario, o utilizando el comando:

```
$ sudo reboot
```

A partir de este momento comenzaremos con la instalación de los paquetes necesarios para el correcto funcionamiento del algoritmo de conteo. Primero debemos actualizar las dependencias que ya estén instaladas en nuestra Raspberry. Para ello ejecutaremos el siguiente comando:

```
$ sudo apt-get update && sudo apt-get upgrade
```

Continuaremos instalando algunos paquetes básicos:

```
$ sudo apt-get install build-essential cmake pkg-config
```

Para poder trabajar con imágenes y vídeo, debemos instalar algunos paquetes de **I/O**[9]:

```
$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
$ sudo apt-get install libxvidcore-dev libx264-dev
```

OpenCV utiliza algunas dependencias que deberemos instalar. Pero también añadiremos algunos paquetes extra que mejorarán un poco el rendimiento:

```
$ sudo apt-get install libfontconfig1-dev libcairo2-dev
$ sudo apt-get install libgdk-pixbuf2.0-dev libpango1.0-dev
$ sudo apt-get install libgtk2.0-dev libgtk-3-dev
$ sudo apt-get install libatlas-base-dev gfortran
$ sudo apt-get install libhdf5-dev libhdf5-serial-dev libhdf5-103
$ sudo apt-get install libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5
```

Instalamos python3:

```
$ sudo apt-get install python3-dev
```

A continuación instalaremos los paquetes de python3 necesarios para la ejecución del algoritmo:

NumPy

Basta con utilizar el siguiente comando:

```
$ sudo apt-get install python3-dev
```

dlib

Para instalar este paquete, primero debemos ejecutar los siguientes comandos:

```
$ sudo apt-get install libopenblas-dev liblapack-dev
$ sudo apt-get install libx11-dev
```

Y para instalar una versión de dlib optimizada para nuestro dispositivo ejecutamos:

```
$ git clone https://github.com/davisking/dlib.git
$ cd dlib
$ python setup.py install --yes USE_NEON_INSTRUCTIONS
```

imutils

Ejecutaremos:

```
$ pip3 install imutils
```

En este momento ya podemos instalar OpenCV. Existen dos métodos: Directamente desde *pip3*; o descargar el código fuente de OpenCV y compilarlo para nuestro dispositivo. Como buscamos que el algoritmo sea lo más eficiente posible, nos decantaremos por la segunda opción. Para ello, nos descargaremos el código de OpenCV:

```
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/4.5.2.zip
$ wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.5.2.zip
$ unzip opencv.zip
$ unzip opencv_contrib.zip
$ mv opencv-4.5.2 opencv
$ mv opencv_contrib-4.5.2 opencv_contrib
```

[9] Input Output. Entrada y Salida

El principal inconveniente de tomar esta decisión es que para compilar e instalar OpenCV podemos tardar entre 2 y 4 horas. Como es un proceso muy demandante, incrementaremos la memoria de **Swap**[10] de nuestra Raspberry para que pueda aguantarlo. Comenzaremos editando el siguiente archivo:

```
$ sudo nano /etc/dphys-swapfile
```

Dentro del fichero, editaremos la línea en la que se inicializa la variable *CONF_SWAPSIZE*, cambiando el valor que tiene por 2048:

```
$ CONF_SWAPSIZE=2048
```

Guardamos y cerramos el archivo, y reiniciamos el servicio de swap:

```
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo /etc/init.d/dphys-swapfile start
```

A continuación ejecutaremos la **build**[11] de OpenCV:

```
$ cd opencv
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
  -D CMAKE_INSTALL_PREFIX=/usr/local \
  -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
  -D ENABLE_NEON=ON \
  -D ENABLE_VFPV3=ON \
  -D BUILD_TESTS=OFF \
  -D INSTALL_PYTHON_EXAMPLES=OFF \
  -D OPENCV_ENABLE_NONFREE=ON \
  -D CMAKE_SHARED_LINKER_FLAGS=-latomic \
  -D BUILD_EXAMPLES=OFF ..
```

Una vez finalice, estamos preparados para comenzar con la compilación. Para ello ejecutaremos el siguiente comando, indicando que queremos que se utilicen los 4 núcleos del procesador:

```
$ make -j4
```

Cuando haya terminado, instalaremos la versión optimizada para nuestra Raspberry de OpenCV:

```
$ sudo make install
$ sudo ldconfig
```

Una vez finalice, debemos resetear el tamaño de la memoria de swap utilizando los comandos anteriores (el tamaño original es 100).

Llegados a este punto, ya tendremos nuestra Raspberry Pi completamente preparada para ejecutar el algoritmo de conteo.

3.2. Aplicación Web

3.2.1. Servidor Backend

Para poder desarrollar una aplicación web que sea robusta y escalable, debe construirse sobre unos cimientos firmes. Por ello, se ha decidido contar con NestJS como framework

[10] https://es.wikipedia.org/wiki/Espacio_de_intercambio
[11] construcción

principal para implementar el Backend[12] de nuestra aplicación, precisamente porque permite estructurar el proyecto en diferentes módulos independientes unos de otros. La principal función de nuestro servidor será interconectar la arquitectura del contador de aforo con nuestra aplicación web, comprobando que las peticiones que reciba estén debidamente autenticadas y se disponga de la autorización adecuada.

Lo primero que debemos hacer para lograr esto, es configurar las tecnologías principales que vamos a utilizar, que son GraphQL, Apollo y TypeORM. Gracias a que NestJS ofrece soporte a muchas otras tecnologías de manera nativa, realizar esta tarea es muy sencilla ya que solo ha de llevarse a cabo en el **módulo principal**[13] del servidor. Es interesante observar que no se requiere ningún tipo de configuración de Apollo, ya que el **GraphQLModule** actúa como una envoltura alrededor de Apollo Server. Por tanto, los parámetros que definamos aquí serán utilizados por una instancia subyacente de este framework.

Para crear el esquema de GraphQL, existen dos aproximaciones distintas:

- **Code First**[14]: se utilizarán **decoradores**[15] de NestJS y clases utilizando TypseScript para generar el esquema de GraphQL.
- **Schema First**[16]: si nos decantamos por esta opción, deberemos definir manualmente el esquema utilizando la sintaxis de GraphQL.

En nuestro caso, hemos elegido la aproximación Code First, por lo que definiremos nuestras entidades de base de datos como clases.

Por otro lado, los módulos implementados están organizados siguiendo una estructura de directorios específica, agrupando archivos con lógicas similares, aunque no todos ellos son necesarios. Dicha estructura es la siguiente:

- **Models**[17]: En esta carpeta se definirán las entidades que sean necesarias para el módulo, es decir, se indica cuál es la estructura que van a tener los datos almacenados en la BDD. Pero además, se define cuáles son los campos que va a utilizar GraphQL a la hora de responder a las consultas que reciba. Esto se logra utilizando los decoradores que ofrece NestJS: **@Entity()**, **@PrimaryColumn()** y **@Column()** en el caso de las entidades, y **@ObjectType()** y **@Field()** para los campos que va a utilizar GraphQL. Para que GraphQL pueda interactuar con las instacias de un modelo, es necesario especificar al menos un Resolver y un Service.
- **Resolvers**[18]: GraphQL necesita que se utilice un **resolver**[19] por cada módulo para su correcto funcionamiento. Este especifica las diferentes consultas que puede recibir, su tipo y los parámetros que acepta. Se declararán como mínimo las peticiones de un servicio **CRUD**[20].

[12] https://github.com/alu0100881165/tfg_web_back/tree/master
[13] https://github.com/alu0100881165/tfg_web_back/blob/master/src/app.module.ts
[14] https://docs.nestjs.com/graphql/quick-start#code-first
[15] https://medium.com/google-developers/exploring-es7-decorators-76ecb65fb841
[16] https://docs.nestjs.com/graphql/quick-start#schema-first
[17] https://github.com/alu0100881165/tfg_web_back/blob/master/src/modules/auth/models/user.model.ts
[18] https://github.com/alu0100881165/tfg_web_back/blob/master/src/modules/auth/resolvers/auth.resolver.ts
[19] Resolutor
[20] Crear, Leer, Actualizar y Eliminar

- **Services**[21]: Los resolvers hacen uso de los servicios para poder interactuar con la base de datos. Como se puede apreciar en el ejemplo de resolver anterior, este hace uso de un *authService* que ejecuta la parte de la lógica detrás de esta consulta que involucra a la BDD.

- **Responses**: En el caso de querer personalizar la respuesta que vamos a dar a una consulta, esta estará definida dentro de esta carpeta.

- **Controllers**: Los resolvers se utilizan para las consultas de GraphQL, pero ¿qué ocurre si nos llega una petición REST? En esta carpeta se declararán los **controllers**[22], los cuales funcionarán de la misma manera que los resolvers, utilizando los servicios para acceder a la BDD, solo que el tráfico de entrada será REST.

- **Guards**: Los guardias nos permiten implementar lógica que va a interceptar una petición (ya sea por GraphQL o REST), antes de que se ejecute el resolver o el controller correspondiente. Gracias a ellos, podemos filtrar las peticiones que se reciben.

- **Decorators**: Como se ha comentado anteriormente, y se ha podido observar en los ejemplos expuestos, se utilizan muchos decoradores para dotar de funcionalidades al código. Por ello, NestJS nos permite la opción de crear los nuestros personalizados, los cuales se encontrarán definidos en esta carpeta.

Por último, todos los módulos tiene un archivo **Module** definido, en el que se especifica cuáles son los modelos, resolvers, controllers y servicios que utiliza. Si fuera necesario el uso de algún modelo o servicio de otro módulo, este sería importado desde aquí.

Los diferentes módulos que han sido implementados para la aplicación web, son los siguientes:

AuthModule

Es probablemente el módulo más importante y complejo de todo el servidor. En él se implementa el modelo del usuario junto a su resolver y servicios, con los que poder obtener por ejemplo un usuario determinado, o un listado de los usuarios registrados. También contiene toda la lógica que utilizaremos para autenticar a los usuarios y contadores, y para autorizar el uso de consultas restringidas. Además, junto con el módulo del contador, son los únicos que utilizarán un controlador. Más adelante veremos el por qué.

Para que un usuario pueda utilizar nuestra aplicación, debe estar registrado y haber iniciado sesión previamente. Durante el proceso de Login, se otorgará al usuario dos **Tokens**[23], uno de acceso y otro de refresco. El primero de ellos sirve para autenticar al usuario sin necesidad de que tenga que volver a introducir sus credenciales. Por tanto, podemos hacer que la consulta de login sea de dominio público, pero para poder ejecutar cualquier otra se debe haber iniciado sesión previamente, ya que será necesario un token de acceso para poder utilizarlas. Por otro lado, el token de refresco nos permite mantener la sesión de un usuario iniciada por un tiempo determinado. Se almacena como una

[21] `https://github.com/alu0100881165/tfg_web_back/blob/master/src/modules/auth/services/auth.service.ts`
[22] Controladores
[23] Símbolos

cookie[24] en el navegador, y cuando un usuario dispone de uno válido, automáticamente inicia sesión y se le concede un token de acceso con el que podrá navegar por la aplicación.

También es necesario destacar la definición de dos guards, uno de **autenticación**[25] y otro de **autorización**[26] basado en roles, y un decorador que combina ambos guardias. NestJS utiliza una clase denominada **ExecutionContext**[27], la cual contiene información acerca del proceso de ejecución actual. ¿Pero esto en qué se traduce en nuestro código? Muy sencillo, nos permite obtener información sobre quién realizó la petición, o a qué método va dirigida, además de incluir el objeto **Request**[28], en el que se encuentra la información que envía un usuario al servidor.

Por tanto, para poder implementar un guardia, es necesario hacer uso de este ExecutionContext, además de implementar la clase **CanActivate**, y definir un método con su mismo nombre, el cual contendrá la lógica necesaria para decidir si se permite que continúe esa petición, o si debe ser rechazada. En nuestro caso, el guardia de autenticación será utilizado para asegurarnos de que el usuario detrás de la petición ha iniciado sesión previamente, por lo que dentro de los **Headers**[29] del objeto Request, deberá existir una autorización que incluya un token de acceso.

Los usuarios registrados tendrán definido uno o varios roles a los que pertenecen. Gracias a esto podemos implementar nuestro guardia de autorización, el cual comprobará que el usuario que realice una petición pertenezca a un rol determinado para poder ejecutarla.

Al combinar estos dos guards en un **decorador**[30], podemos proteger una ruta o consulta de manera muy sencilla, añadiendo antes de su definición el siguiente código: **@Auth()**. Especificando entre los paréntesis del mismo el rol o roles a los que se permitirá el acceso.

Es necesario el uso de un controller[31] en este módulo, ya que cuando un usuario dispone de un token de refresco y entra en la aplicación, automáticamente se manda una petición REST al servidor, para que compruebe si su token es válido. Si lo es, le responderá con un token de acceso, permitiendo su acceso automático.

Por último, todos estos archivos serán incluidos en el fichero principal del **módulo de autenticación**[32]. En este caso, debido a que existe una relación entre los usuarios y las compañías o empresas, se debe importar su modulo para poder hacer uso de su modelo. Además, se exportará tanto el servicio de usuario como el de autenticación, para que puedan ser utilizados por otros módulos.

---

[24] https://es.wikipedia.org/wiki/Cookie_(inform%C3%A1tica)

[25] https://github.com/alu0100881165/tfg_web_back/blob/master/src/modules/auth/guards/auth.guard.ts

[26] https://github.com/alu0100881165/tfg_web_back/blob/master/src/modules/auth/guards/roles.guard.ts

[27] https://docs.nestjs.com/fundamentals/execution-context

[28] Petición

[29] Cabeceras

[30] https://github.com/alu0100881165/tfg_web_back/blob/master/src/modules/auth/decorators/auth.decorator.ts

[31] https://github.com/alu0100881165/tfg_web_back/blob/master/src/modules/auth/controllers/auth.controller.ts

[32] https://github.com/alu0100881165/tfg_web_back/blob/master/src/modules/auth/auth.module.ts

CompanyModule

Al contrario que el anterior, este es el módulo más sencillo de todos. Esto se debe a que su principal función es agrupar a los usuarios y contadores de una misma empresa. Para asociarlos, se establecen una relación **One to Many**[33] entre una empresa y varios contadores, así como entre una empresa y varios usuarios. Esto se traduce en que una empresa puede tener muchos usuarios y contadores relacionados, pero estos solo pueden pertenecer a una única empresa asociada.

CounterModule

El contador es un caso especial ya que es un dispositivo pasivo, no requiere de interacción con una persona para comenzar a funcionar, por lo que se ha tenido que implementar una autenticación algo diferente, pero esto será tratado más adelante en la sección **Interconectar arquitecturas**.

Por otro lado, se ha implementado un guardia cuyo funcionamiento es idéntico al del módulo de autenticación, pero los tipos de datos con los que trabaja son diferentes, al tratarse de un contador y no de un usuario.

También cuenta con un decorador[34], el cual nos permite acceder al ExecutionContext, para extraer la información de un objeto contador que nos sea necesaria de manera automática, como por ejemplo:

```
1 @Counter() { username }: AccessTokenPayloadCounter
```

StatisticsModule

Finalmente, encontramos el módulo de estadísticas. Los contadores lo utilizarán para enviar los datos que vayan generando en intervalos de cinco segundos.

Una vez recibidos, se invocará al servicio encargado de crearlas[35], el cual comprobará si el tiempo transcurrido entre la última inserción en la base de datos y la nueva estadística, supera los 10 minutos. Si es así, se creará una nueva entrada en la base de datos con la nueva información, pero si no, se actualizarán los valores de la última entrada. Se ha tomado esta decisión para que el tamaño de la base de datos aumente demasiado rápido.

3.2.2. Frontend

Como hemos comentado anteriormente, el desarrollo del frontend de la aplicación se ha llevado a cabo utilizando principalmente **React**, **SCSS**, **TypeScript** y **Apollo Client**, además del uso de **Ionic** como framework de UI. Utilizando estas tecnologías se han desarrollado las siguientes pantallas para la aplicación:

Login

Visualmente es un pequeño formulario en el que introducir nuestro usuario y contraseña. Si el login es satisfactorio, se recibirá por parte del backend un token de acceso y otro

[33] Uno a Muchos

[34] https://github.com/alu0100881165/tfg_web_back/blob/master/src/modules/counter/decorators/counter.decorator.ts

[35] https://github.com/alu0100881165/tfg_web_back/blob/master/src/modules/statistics/services/statistics.service.ts

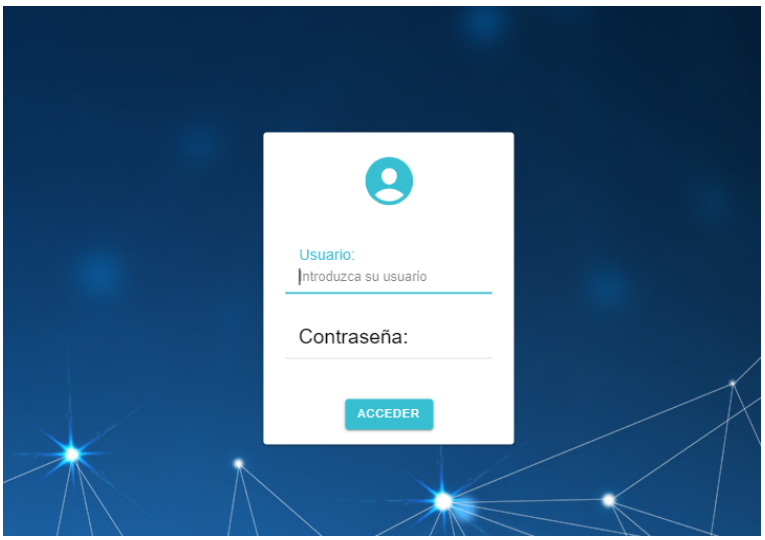


Figura 3.7: Página de Login versión Web

de refresco, y es entonces cuando la lógica detrás de esta pantalla comienza a funcionar.

El token de acceso se añadirá a Apollo[36], para que a partir de ahora todas las peticiones que se hagan incluyan este token en el **Header**[37] de **Authorization**[38].

Por otro lado, el token de refresco será almacenado en el navegador en forma de cookie. Si se vuelve a acceder a esta pantalla disponiendo uno, se realizará una petición REST al backend enviándolo, y si es válido, se iniciará la sesión automáticamente recibiendo además un nuevo token de acceso.

Dashboard

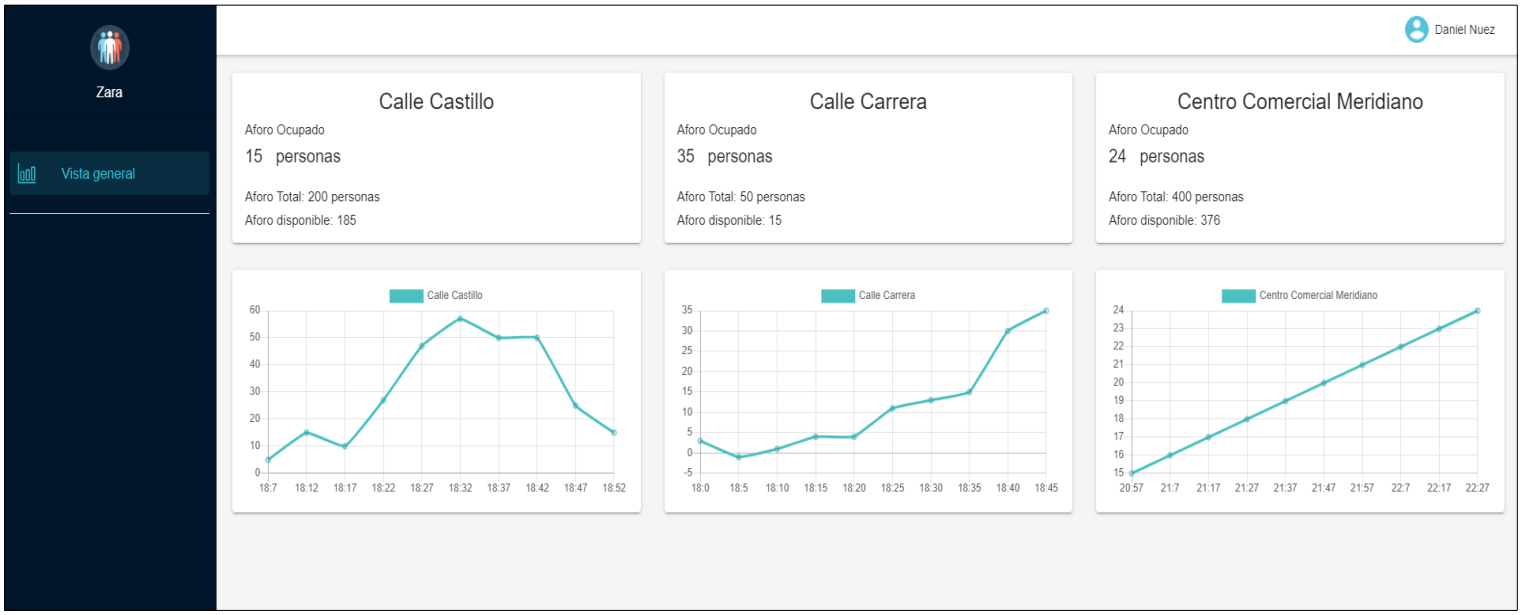


Figura 3.8: Dashboard versión Web

[36] https://github.com/alu0100881165/tfg_web/blob/master/src/App.tsx
[37] Cabecera
[38] Autorización

Una vez hayamos iniciado sesión, seremos redirigidos al dashboard. En el lateral izquierdo podremos ver el menú de la aplicación donde aparecerá un icono y el nombre de la empresa a la que pertenezca el usuario actual junto con un listado de la pantallas disponibles. En la figura **3.8** solo se puede ver la de *Vista general*, ya que el resto están protegidas y solo pueden ser utilizadas por un usuario administrador.

Esto se ha logrado haciendo que este menú construya la lista utilizando un fichero de **páginas**[39] y otro de **rutas**[40].

En el primero se definen todas las vistas de la aplicación, su ruta, el icono, el nombre de la pantalla y el rol mínimo que se ha de tener para poder acceder a ella. Por tanto, si el usuario actual no cumple con este requisito, estas pantallas no aparecerán.

Pero esto no basta para proteger esas vistas, ya que si un usuario introdujera una ruta a la que no tiene acceso en la **URL**[41] del navegador, podría acceder sin ningún problema. Para ello utilizamos el archivo de rutas que configura la navegación de la aplicación. Este permite acceder a la pantalla de login sin ningún problema, pero si accedemos a cualquier otra y no hemos iniciado sesión o nuestro rol no es válido, nos redirigirá a la pantalla de login automáticamente.

En la parte superior derecha de la pantalla, podemos ver un pequeño icono junto con el nombre del usuario que ha iniciado sesión. Si pulsamos sobre él, se desplegará un menú desde el cual podremos cerrar la sesión.

En cuanto al **contenido de la página**[42], se puede consultar en tiempo real el estado del aforo de los contadores que tengamos asociados, además de unas gráficas con las estadísticas de los mismos a lo largo del tiempo. Cada 5 segundos se actualizan los valores de los contadores, mientras que las gráficas cada 10 minutos.

Administrar usuarios

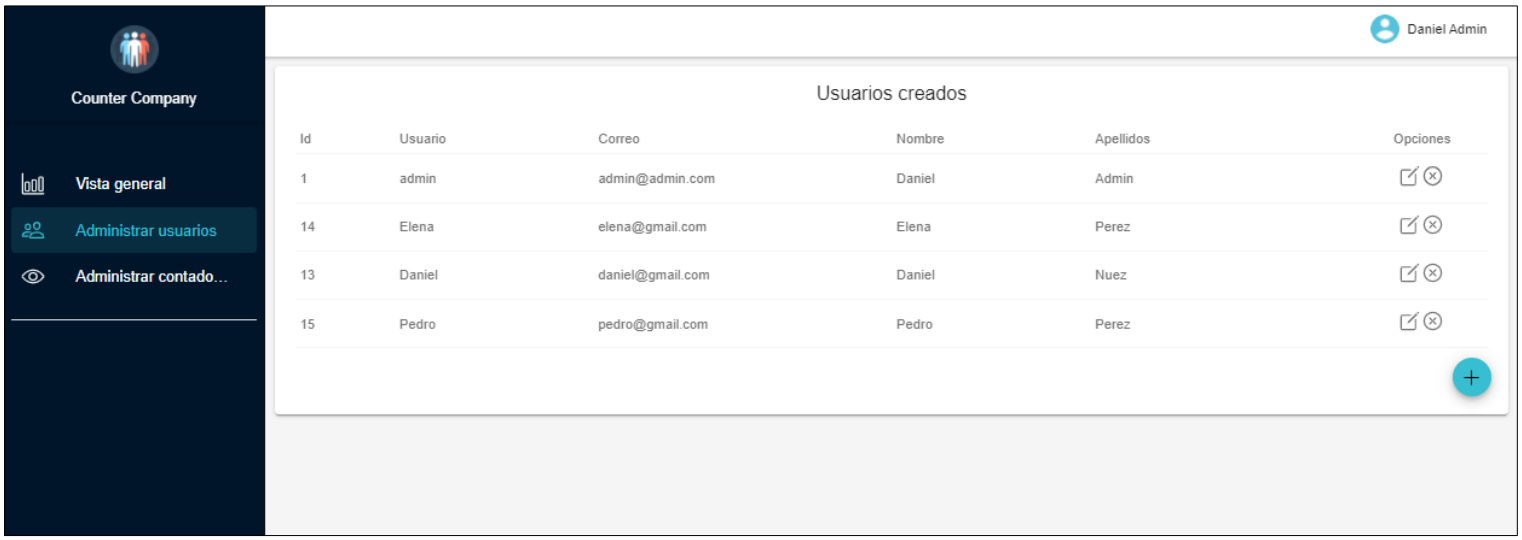


Figura 3.9: Tabla de usuarios versión Web

[39] https://github.com/alu0100881165/tfg_web/blob/master/src/routes/Pages.ts
[40] https://github.com/alu0100881165/tfg_web/blob/master/src/routes/Routes.tsx
[41] https://es.wikipedia.org/wiki/Localizador_de_recursos_uniforme
[42] https://github.com/alu0100881165/tfg_web/blob/master/src/components/graphicsContainer/GraphicsContainer.tsx

Desde esta pantalla[43] se pueden visualizar todos los usuarios que estén registrados en nuestra empresa organizados en una tabla, cuyas columnas corresponden a algunos de sus datos.

Existen tres botones dentro de nuestra tabla. El botón de añadir un nuevo usuario, el cual desplegará un **formulario**[44] en que añadir los datos correspondientes.

Figura 3.10: Formulario para añadir un usuario

En la columna de opciones encontramos un botón que nos permitirá editar al usuario que se encuentre en esa fila. Se reutilizará el mismo formulario usado para la creación, solo que el campo contraseña no estará disponible, el título se adaptará a la acción que realicemos y el tamaño del mismo será algo mayor, en el caso de que algún campo sea demasiado largo.

[43] https://github.com/alu0100881165/tfg_web/blob/master/src/components/userTable/UserTable.tsx

[44] https://github.com/alu0100881165/tfg_web/blob/master/src/components/userForm/userForm.tsx

Figura 3.11: Formulario para editar un usuario

Existe un último botón que permite la eliminación de un usuario, situado al lado del anterior en la columna de opciones. Una vez seleccionado, aparecerá un mensaje para confirmar que se desea borrar ese usuario.

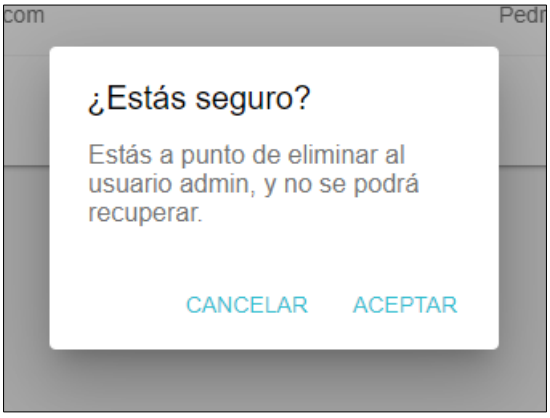


Figura 3.12: Mensaje de confirmación para eliminar un usuario

Administrar contadores

Esta tabla[45] es muy similar a la de la pantalla anterior, teniendo como únicas diferencias que la creación, edición y eliminación de contadores está bloqueada. Esto se debe a que los campos de los contadores son muy delicados, ya que algunos están relacionados con su autenticación y además, como veremos en el siguiente capítulo, se dispondrá de un número limitado de contadores. Por lo que no se podrán crear bajo demanda, sino que se darán de alta los contadores necesarios desde otro portal, o en su defecto desde la propia base de datos.

[45] `https://github.com/alu0100881165/tfg_web/blob/master/src/components/counterTable/CounterTable.tsx`

Figura 3.13: Dashboard versión Web

Por último, se debe mencionar el uso de un componente por parte de la aplicación de manera global, el cual maneja los mensajes tanto de error como de éxito en las diferentes llamadas al backend de la aplicación. Para que esta herramienta sea accesible desde los diferentes puntos de la aplicación, se ha utilizado un **Contexto**[46]. Esto nos proporciona un mecanismo para compartir los datos sin tener que pasarlos entre los componentes directamente. De hecho, llevamos utilizándolos para toda la aplicación. Existen tres contextos que realizan diferentes funciones:

- **Inicio de sesión**[47]: desde él se realizan todas las peticiones que tengan que ver con la autenticación, como son el Login, Registro y el Logout[48].
- **Obtención de datos**[49]: centraliza todas las consultas que tengan que ver con obtener información acerca de los usuarios, contadores y estadísticas.
- **Utilidades**[50]: a parte de poder mostrar mensajes de error y confirmación, también ofrece un mensaje para cuando está cargando un componente, o se está esperando a que alguna consulta finalice.

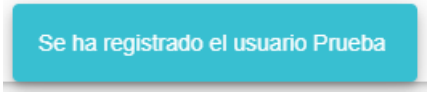


Figura 3.14: Mensaje de confirmación al crear un usuario

Figura 3.15: Mensaje de error al iniciar sesión

[46] https://es.reactjs.org/docs/context.html
[47] https://github.com/alu0100881165/tfg_web/blob/master/src/context/auth/auth.context.tsx
[48] Cerrar sesión
[49] https://github.com/alu0100881165/tfg_web/blob/master/src/context/data/data.context.tsx
[50] https://github.com/alu0100881165/tfg_web/blob/master/src/context/error/utils.context.tsx

3.2.3. Base de Datos

Otra de las principales ventajas del uso de NestJS y TypeORM, es que cuando se arranque el servidor, y se realice la conexión con la base de datos, se comprobará si existen las tablas que representan las diferentes entidades declaradas en los módulos. Si no es así, las creará automáticamente, incluyendo las relaciones entre ellas. Para realizar esto, utiliza una tabla denominada **migrations_typeorm**, en la cual se definen las consultas necesarias para actualizar el esquema de la base de datos, ya sea creando, eliminando o editando tablas.

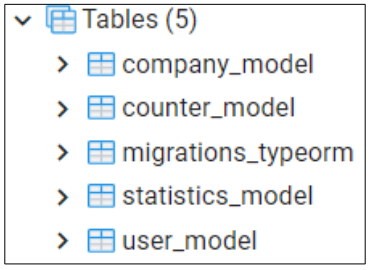


Figura 3.16: Tablas generadas

id [PK] integer	username character varying (255)	email character varying (255)	roles user_model_roles_enum[]	password character varying (255)	firstname character varying (255)	lastname character varying (255)	tokenVersion integer	company integer
1	admin	admin@admin.com	{BaseUser,Admin}	$2b$12$T8pCZabzygY5IAc9ii...	Daniel	Admin	0	1
13	Daniel	daniel@gmail.com	{BaseUser}	$2b$12$iB5e7FkgnBwMTGLS...	Daniel	Nuez	0	1
14	Elena	elena@gmail.com	{BaseUser}	$2b$12$odWc89AYTHeLXQD...	Elena	Perez	0	1
15	Pedro	pedro@gmail.com	{BaseUser}	$2b$12$0OquW5Wz636j9sbh...	Pedro	Perez	0	1
16	Prueba	prueba@prueba.com	{BaseUser}	$2b$12$87/xDUHKz/MuuPz5...	Prueba	Testing	0	1

Figura 3.17: Tabla de Usuarios

3.3. Interconectar arquitecturas

Hasta ahora hemos hablado de cómo funcionan las arquitecturas del contador de aforo y de la aplicación web de manera aislada, sin especificar como se comunican entre ellas. Como sabemos, para autenticar a un usuario en el frontend de la aplicación este debe rellenar un formulario con sus credenciales y, si son válidas, se le otorgará un token de acceso y otro de refresco. Pero replicar este comportamiento no es posible en el contador, ya que no existe un usuario que rellene estos datos cada vez que se encienda.

Como solución, se ha decidido utilizar una estrategia basada en **APIKey**[51]. Para ello se ha configurado un campo de contraseña en la entidad de los contadores y, por tanto, en la tabla de la base de datos. Cuando se registra un nuevo contador, se deberá introducir una contraseña, que se encriptará y almacenará. Una vez dado de alta, se procederá a configurar el fichero **.env** del contador, en el cual se establecerán las variables de entorno que serán utilizadas. Estas variables contienen información sensible, y se encuentran en un fichero a parte para protegerlas, ya que para acceder a ellas desde el código es necesario el uso de la **librería dontenv**[52]. Las variables definidas para la correcta autenticación del contador son: el usuario que lo representa, y la contraseña del mismo

[51] https://en.wikipedia.org/wiki/Application_programming_interface_key
[52] https://pypi.org/project/python-dotenv/

encriptada, que hará de APIKey. Estos campos se envían al backend donde son validados, y si son correctos se envía un token de acceso como respuesta, para que el contador pueda comenzar a enviar los datos. Aunque por simplicidad y para permitir una mayor comodidad a la hora de hacer pruebas, actualmente en el prototipo se está utilizando la contraseña en texto plano. Pero si existen los mecanismos necesarios para **encriptarla**[53].

[53] https://github.com/alu0100881165/tfg_web_back/blob/master/src/modules/counter/services/counter.service.ts

Capítulo 4

Estudio de viabilidad económica

Como ya sabemos, actualmente es fundamental mantener un control sobre el aforo de los establecimientos, pero las soluciones que ofrece el mercado no son las más óptimas. Se ha realizado un análisis de los principales productos que cubren esta necesidad, pero nos hemos encontrado con dos situaciones opuestas: por un lado, los productos ofertados son buenos, pero los precios son desorbitados; mientras que por el otro, existen opciones más económicas, pero que realizan un conteo demasiado básico. En algunos casos, el coste de solo una cámara es de 600€, pero en otros casos la suma asciende a los 2800€.

Por ello, se ha buscado brindar nuestro servicio desmarcándonos de la competencia, ya que ofrecemos un contador de aforo junto con una plataforma online desde la cual poder visualizar la información y gestionar los usuarios de manera muy sencilla.

Pero para conocer realmente la viabilidad del proyecto, debemos dar respuesta a preguntas como: ¿cuánto cuesta desarrollar el proyecto? ¿cuánto tiempo sería necesario para llevarlo a cabo? ¿cuándo se recuperaría la inversión inicial? ¿cómo se pretende comercializarlo?

La mejor manera de responderlas es simular el desarrollo y crear un plan de comercialización para nuestro producto.

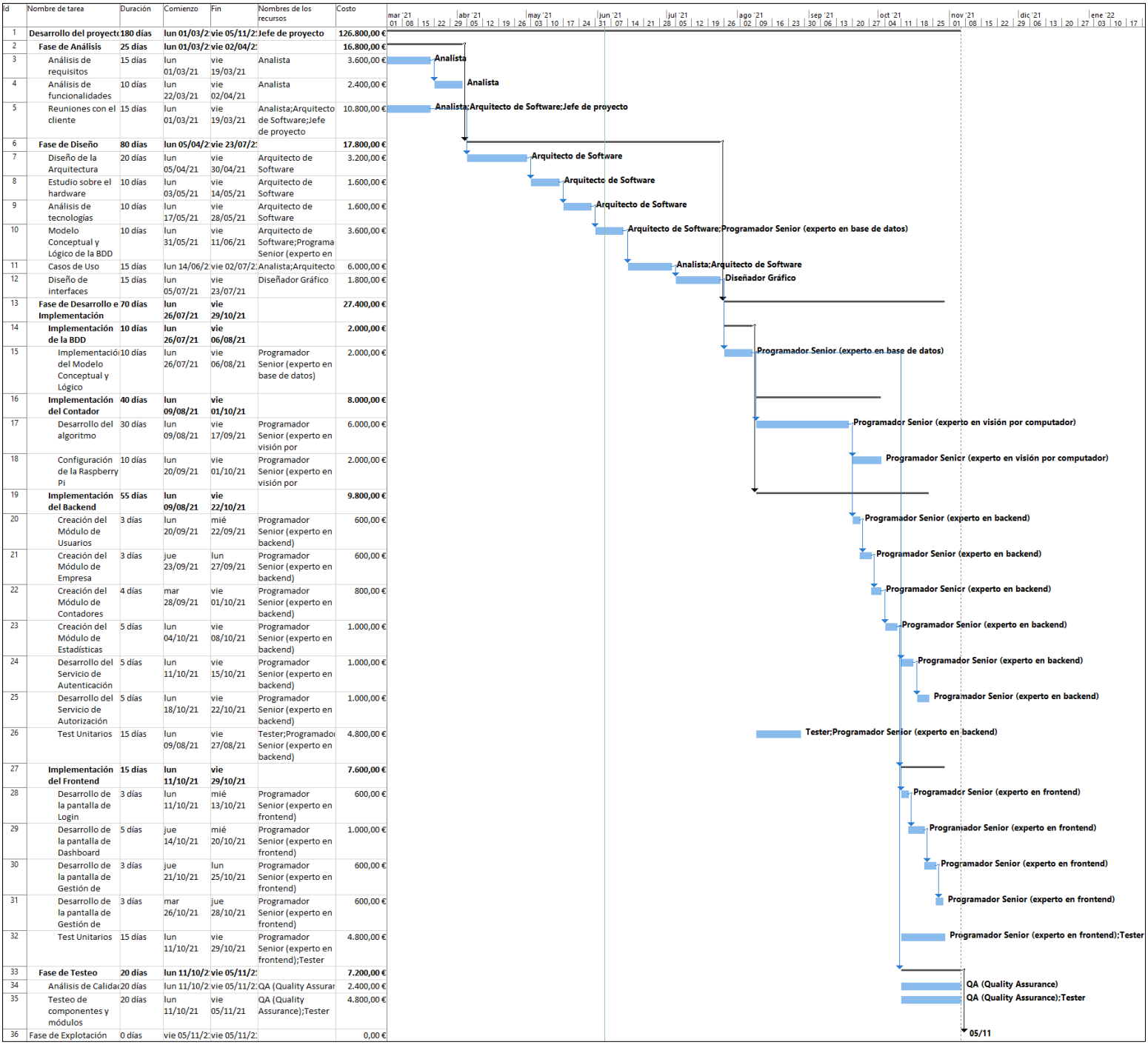
4.1. Desarrollo del proyecto

Se ha utilizado el programa **Microsoft Project**[1], con el cual hemos planificado y presupuestado el desarrollo de nuestro proyecto, dividiéndolo en diferentes fases(**4.1**).

Durante la primera Fase de Análisis, se pretende comprender cuál es el problema que queremos resolver. Para ello se constituye un conjunto de requisitos que debe cumplir el producto, indicando las funcionalidades que debe tener y las necesidades que debe satisfacer. Una parte fundamental de esta etapa son las reuniones con las partes implicadas. A medida que se van obteniendo avances en el análisis, se deben exponer en estas reuniones para asegurar que cumplen los requerimientos de los clientes.

Una vez haya finalizado la primera etapa, comenzaremos con la Fase de Diseño, en la que se establecerán las arquitecturas que compondrán el producto, realizando tanto un análisis de hardware como de las tecnologías a utilizar. Una vez estas cuestiones más teóricas hayan sido completadas, se realizará una primera aproximación a la estructura que seguirá la base de datos, junto con la interfaz de usuario y los casos de uso. Estos

[1] https://www.microsoft.com/eses/microsoft365/project/projectmanagementsoftware



Id	Nombre de tarea	Duración	Comienzo	Fin	Nombres de los recursos	Costo
1	**Desarrollo del proyecto**	**180 días**	**lun 01/03/21**	**vie 05/11/21**	**Jefe de proyecto**	**126.800,00 €**
2	**Fase de Análisis**	**25 días**	**lun 01/03/21**	**vie 02/04/21**		**16.800,00 €**
3	Análisis de requisitos	15 días	lun 01/03/21	vie 19/03/21	Analista	3.600,00 €
4	Análisis de funcionalidades	10 días	lun 22/03/21	vie 02/04/21	Analista	2.400,00 €
5	Reuniones con el cliente	15 días	lun 01/03/21	vie 19/03/21	Analista;Arquitecto de Software;Jefe de proyecto	10.800,00 €
6	**Fase de Diseño**	**80 días**	**lun 05/04/21**	**vie 23/07/21**		**17.800,00 €**
7	Diseño de la Arquitectura	20 días	lun 05/04/21	vie 30/04/21	Arquitecto de Software	3.200,00 €
8	Estudio sobre el hardware	10 días	lun 03/05/21	vie 14/05/21	Arquitecto de Software	1.600,00 €
9	Análisis de tecnologías	10 días	lun 17/05/21	vie 28/05/21	Arquitecto de Software	1.600,00 €
10	Modelo Conceptual y Lógico de la BDD	10 días	lun 31/05/21	vie 11/06/21	Arquitecto de Software;Programa Senior (experto en	3.600,00 €
11	Casos de Uso	15 días	lun 14/06/21	vie 02/07/21	Analista;Arquitecto	6.000,00 €
12	Diseño de interfaces	15 días	lun 05/07/21	vie 23/07/21	Diseñador Gráfico	1.800,00 €
13	**Fase de Desarrollo e Implementación**	**70 días**	**lun 26/07/21**	**vie 29/10/21**		**27.400,00 €**
14	**Implementación de la BDD**	**10 días**	**lun 26/07/21**	**vie 06/08/21**		**2.000,00 €**
15	Implementación del Modelo Conceptual y Lógico	10 días	lun 26/07/21	vie 06/08/21	Programador Senior (experto en base de datos)	2.000,00 €
16	**Implementación del Contador**	**40 días**	**lun 09/08/21**	**vie 01/10/21**		**8.000,00 €**
17	Desarrollo del algoritmo	30 días	lun 09/08/21	vie 17/09/21	Programador Senior (experto en visión por	6.000,00 €
18	Configuración de la Raspberry Pi	10 días	lun 20/09/21	vie 01/10/21	Programador Senior (experto en visión por	2.000,00 €
19	**Implementación del Backend**	**55 días**	**lun 09/08/21**	**vie 22/10/21**		**9.800,00 €**
20	Creación del Módulo de Usuarios	3 días	lun 20/09/21	mié 22/09/21	Programador Senior (experto en backend)	600,00 €
21	Creación del Módulo de Empresa	3 días	jue 23/09/21	lun 27/09/21	Programador Senior (experto en backend)	600,00 €
22	Creación del Módulo de Contadores	4 días	mar 28/09/21	vie 01/10/21	Programador Senior (experto en backend)	800,00 €
23	Creación del Módulo de Estadísticas	5 días	lun 04/10/21	vie 08/10/21	Programador Senior (experto en backend)	1.000,00 €
24	Desarrollo del Servicio de Autenticación	5 días	lun 11/10/21	vie 15/10/21	Programador Senior (experto en backend)	1.000,00 €
25	Desarrollo del Servicio de Autorización	5 días	lun 18/10/21	vie 22/10/21	Programador Senior (experto en backend)	1.000,00 €
26	Test Unitarios	15 días	lun 09/08/21	vie 27/08/21	Tester;Programador Senior (experto en backend)	4.800,00 €
27	**Implementación del Frontend**	**15 días**	**lun 11/10/21**	**vie 29/10/21**		**7.600,00 €**
28	Desarrollo de la pantalla de Login	3 días	lun 11/10/21	mié 13/10/21	Programador Senior (experto en frontend)	600,00 €
29	Desarrollo de la pantalla de Dashboard	5 días	jue 14/10/21	mié 20/10/21	Programador Senior (experto en frontend)	1.000,00 €
30	Desarrollo de la pantalla de Gestión de	3 días	jue 21/10/21	lun 25/10/21	Programador Senior (experto en frontend)	600,00 €
31	Desarrollo de la pantalla de Gestión de	3 días	mar 26/10/21	jue 28/10/21	Programador Senior (experto en frontend)	600,00 €
32	Test Unitarios	15 días	lun 11/10/21	vie 29/10/21	Programador Senior (experto en frontend);Tester	4.800,00 €
33	**Fase de Testeo**	**20 días**	**lun 11/10/21**	**vie 05/11/21**		**7.200,00 €**
34	Análisis de Calida	20 días	lun 11/10/21	vie 05/11/21	QA (Quality Assurar	2.400,00 €
35	Testeo de componentes y módulos	20 días	lun 11/10/21	vie 05/11/21	QA (Quality Assurance);Tester	4.800,00 €
36	Fase de Explotación	0 días	vie 05/11/21	vie 05/11/21		0,00 €

Figura 4.1: Diagrama de Gantt

últimos consisten en describir las diferentes interacciones que pueden llevarse a cabo mientras se utiliza la aplicación para especificar cuál debe ser el comportamiento esperado por parte del software.

En este punto ya se dispone de toda la información necesaria para comenzar la Fase de Desarrollo. Como su propio nombre indica, a partir de este momento se iniciará la implementación de las diferentes arquitecturas ya sea configurando hardware, implementando algoritmos o desplegando las bases de datos, teniendo en cuenta los requerimientos definidos en las fases anteriores.

Aunque a medida que se realiza el desarrollo de la aplicación se deben realizar pruebas para comprobar el correcto funcionamiento de los avances obtenidos, se llevará a cabo una Fase de Testeo. En ella, se deberá comprobar el desempeño del hardware y del software, se debe asegurar que el producto cumple con unos estándares de calidad mínimos, serán ejecutadas pruebas en entornos simulados, se comprobarán las especificaciones del producto, etc.

Al terminar esta última etapa, el producto estará listo para ser comercializado, arrancando así la Fase de Explotación.

Para llevar a cabo las diferentes tareas que componen cada fase del desarrollo, se ha supuesto que se cuenta con un equipo formado por:

- Un **Jefe de proyecto**: se encargará de supervisar el correcto desarrollo del proyecto. Participará activamente durante las primeras fases de análisis.
- Un **Analista**: es el principal responsable de establecer las bases sobre las que posteriormente comenzará el desarrollo.
- Un **Arquitecto de Software**: realizará el estudio de las posibles tecnologías, hardware y el diseño las arquitecturas que serán utilizadas.
- Un **Diseñador Gráfico**: esbozará la Interfaz de Usuario de la aplicación.
- Cuatro **Programadores Senior**: cada uno estará dedicado exclusivamente a una parte de la aplicación, es decir, Frontend, Backend, Base de Datos y Visión por Computador.
- Un **Tester**: a medida que se desarrollan de las diferentes partes que conforman la arquitectura de la aplicación web, se realizará un proceso de testeo para asegurar el correcto funcionamiento de las mismas.
- Un **Quality Assurance**[2]: su misión será la de asegurar que nuestra aplicación cumple los estándares de calidad establecidos.

Además se han establecido los siguientes salarios:

- Jefe de proyecto: 40€/hora.
- Analista: 30€/hora.
- Arquitecto de Software: 20€/hora.

[2] Seguro de calidad

- Diseñador Gráfico: 15€/hora.
- Programadores Senior: 25€/hora cada uno.
- Tester: 15€/hora.
- Quality Assurance: 15€/hora.

Tras haber añadido toda esta información a Microsoft Project, y establecer la duración estimada de las tareas, obtenemos que el tiempo de desarrollo es de **180 días laborales** o **250 días naturales**, comenzando el *01 de Marzo de 2021* y finalizando el *05 de noviembre de 2021*, siendo la jornada laboral de 8 horas diarias de lunes a viernes. Por último, el costo total del desarrollo del proyecto es de **126.800€**.

Figura 4.2: Costo de desarrollo a lo largo del tiempo

4.2. Modelo de Comercialización

Para poder establecer los precios con los que se pretende comercializar el producto, necesitamos saber cuánto cuesta producir cada contador de aforos, instalarlo y se deben contemplar otro tipo de gastos como son el **Hosting**[3] **Online** de nuestra aplicación web, y la posibilidad de tener que reparar algún contador defectuoso.

El precio total de cada contador asciende a las 100€, ya que están compuestos por:

- Raspberry Pi 4b de 4gb: 60€.
- Tarjeta microSD de 32gb: 10€.
- Cámara USB: 30€.

Se ha decidido hospedar nuestra plataforma web utilizando el servicio de **Amazon Web Service**, ya que ofrece una plataforma que se adapta a los requerimientos de la aplicación, ya sea con un mejor hardware o ancho de banda, o con el aumento dinámico de

[3]Hospedaje

servidores para repartir la carga en caso de tener mucho tráfico entrante. El precio final es de 138,02 euros mensualmente. En un inicio se ha configurado que cada instancia[4] tenga las siguiente características:

- Procesador de 8 núcleos
- 32GB de memoria Ram
- 100GB de memoria secundaria (SSD[5])
- Conexión a internet de hasta 5 Gigabit.

En caso de que se produzca alguna mal función o exista algún defecto irreparable en los contadores de aforo, se procederá al cambio del mismo. Esto supondría un costo de 200 euros, ya que en el peor de los casos habría que cambiar todas las partes que lo componen, y además habría que realizar una nueva instalación.

Una vez aclarados todos estos costes adicionales, se ha decidido comercializar el producto ofreciendo diferentes planes de suscripción mensual:

Figura 4.3: Paquetes de suscripción

En función del plan que se elija, los costes de instalación de cada uno son 100, 150 y 200€ respectivamente.

Cabe destacar que en estos precios no se tienen en cuenta ninguno de los costes adicionales. Por tanto, cada vez que se contrata un nuevo plan, la empresa sufre unas pérdidas que se esperan recuperar a futuro gracias a la suscripción.

Para finalizar, se ha simulado un plan de ventas en el que hemos establecido una proyección de las ventas estimadas durante las primeras 70 semanas. En la siguiente tabla se puede observar las contrataciones de los diferentes planes a lo largo de este tiempo, y los costos de cada nueva venta, del hosting y el número de contadores reparados. Además, se ha decidido aumentar cada mes un 5 % el precio del hospedaje, simulando la necesidad de disponer de mejores o un mayor número de instancias, y a partir del sexto mes de comercialización se ha establecido que comienzan a detectarse las posibles mal funciones o defectos de los contadores.

[4]Cada nuevo servidor cuyo uso sea necesario es llamado instancia.
[5]Unidad de estado sólido

Semana	Plan Basic	Plan Pro	Plan Premium	Costo packs	Hosting	Reparaciones
1	2	0	0	600	138,02	0
2	1	0	0	300	0,00	0
3	0	0	0	0	0,00	0
4	0	0	0	0	0,00	0
5	3	3	0	2550	144,92	0
6	0	0	1	1000	0,00	0
7	4	4	0	3400	0,00	0
8	1	0	0	300	0,00	0
9	1	0	2	2300	152,17	0
10	3	2	0	2000	0,00	0
11	1	0	0	300	0,00	0
12	4	3	2	4850	0,00	0
13	3	0	0	900	167,38	0
14	1	4	0	2500	0,00	0
15	1	3	2	3950	0,00	0
16	0	0	0	0	0,00	0
17	2	0	0	600	184,12	0
18	3	0	0	900	0,00	0
19	4	0	0	1200	0,00	0
20	1	2	0	1400	0,00	0
21	1	0	0	300	211,74	2
22	3	0	0	900	0,00	5
23	1	4	3	5500	0,00	0
24	2	5	1	4350	0,00	4
25	3	0	0	900	243,50	3
26	0	0	0	0	0,00	1
27	0	0	0	0	0,00	2
28	2	1	0	1150	0,00	5
29	0	0	3	3000	255,68	1
30	1	3	0	1950	0,00	0
31	0	0	0	0	0,00	4
32	0	0	0	0	0,00	3
33	0	0	0	0	268,46	0
34	0	0	0	0	0,00	0
35	0	0	0	0	0,00	0
36	4	0	0	1200	0,00	3
37	3	0	0	900	281,88	1
38	0	0	0	0	0,00	4
39	1	2	0	1400	0,00	5
40	0	0	0	0	0,00	0
41	2	0	3	3600	295,98	10
42	0	0	0	0	0,00	3
43	0	3	0	1650	0,00	1
44	0	0	1	1000	0,00	0
45	0	0	0	0	310,78	0
46	0	0	0	0	0,00	0
47	0	0	0	0	0,00	0

48	0	0	0	0	0,00	0
49	0	0	0	0	326,32	4
50	0	0	0	0	0,00	5
51	0	0	0	0	0,00	1
52	0	0	0	0	0,00	2
53	0	0	0	0	342,63	1
54	0	0	0	0	0,00	4
55	0	0	0	0	0,00	1
56	0	0	0	0	0,00	1
57	0	0	0	0	359,76	2
58	0	0	0	0	0,00	6
59	0	0	0	0	0,00	5
60	0	0	0	0	0,00	3
61	0	0	0	0	377,75	1
62	0	0	0	0	0,00	1
63	0	0	0	0	0,00	1
64	0	0	0	0	0,00	0
65	0	0	0	0	396,64	2
66	0	0	0	0	0,00	0
67	0	0	0	0	0,00	2
68	0	0	0	0	0,00	2
69	0	0	0	0	416,47	0

Tabla 4.1: Supuesto de ventas a lo largo del tiempo

Utilizando todos estos datos, se ha calculado el **ROI**[6]:

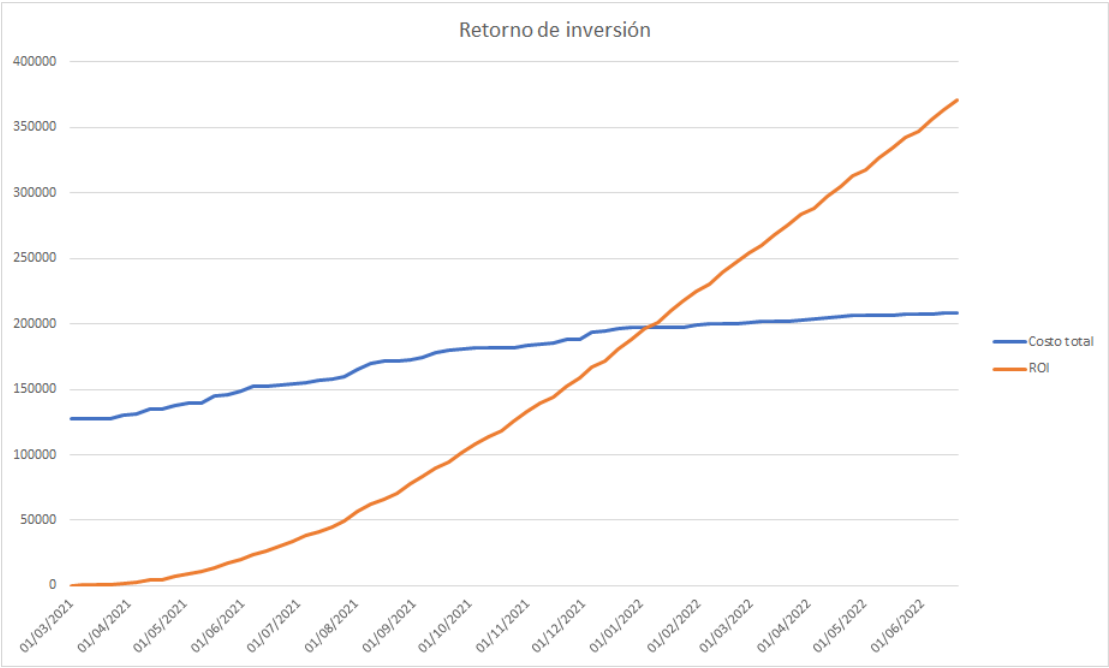


Figura 4.4: Retorno de Inversión

Se puede comprobar que a pesar de contar con 35 semanas de las 70 sin nuevas ventas

[6]Retorno de Inversión

en el supuesto, se consigue recuperar la inversión en tan solo 46 semanas, es decir, en 10 meses y medio.

Capítulo 5

Conclusiones y líneas futuras

Cada vez es más evidente la necesidad contar con dispositivos de conteo de aforo para conocer en tiempo real el número de personas que se encuentran en nuestro establecimiento

No obstante, también podemos obtener mucha información a partir de los datos que genera este conteo, como las horas con mayor afluencia de clientes, o hacer una planificación más a largo plazo sabiendo qué días de la semana acude más gente.

Se ha buscado llevar a cabo este proyecto de la forma más completa posible. Hemos realizado un estudio de mercado en el que se han analizado las principales necesidades y las principales tendencias del mercado para dar forma al prototipo que se ha implementado.

Todo ello ha ido acompañado de un análisis de viabilidad, que concluye que es posible llevar a cabo un proyecto de estas características, con un tiempo de desarrollo y de recuperación de la inversión inicial razonablemente cortos.

Como resultado, se ha obtenido un prototipo completamente funcional, que ofrece una instalación rápida y sencilla, y precios competitivos en comparación con la competencia. Pero además de ofrecer el contador de aforos, contamos una plataforma web que brinda acceso a los datos desde cualquier dispositivo y una gestión sencilla de usuarios y contadores. En cuanto a la escalabilidad del producto, se pueden instalar tantos contadores como sea necesario.

A continuación se pueden observar los resultados medidos utilizando algunos vídeos como prueba:

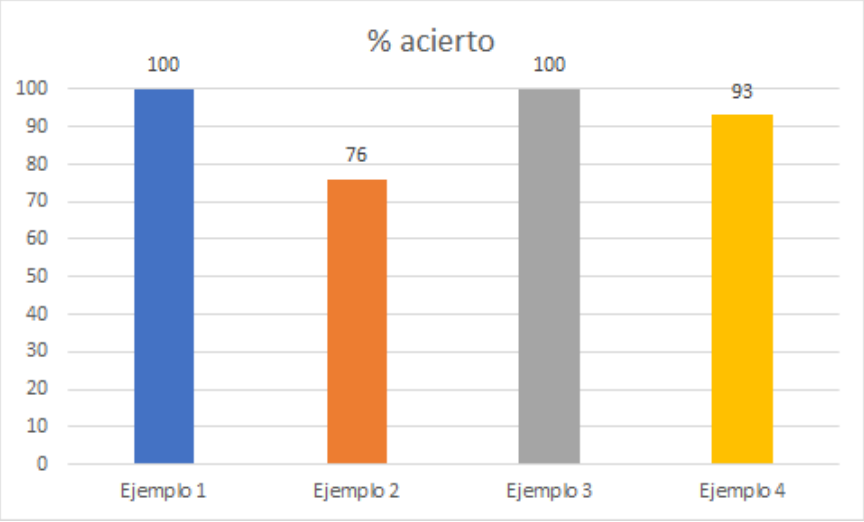


Figura 5.1: Porcentaje de acierto del algoritmo implementado

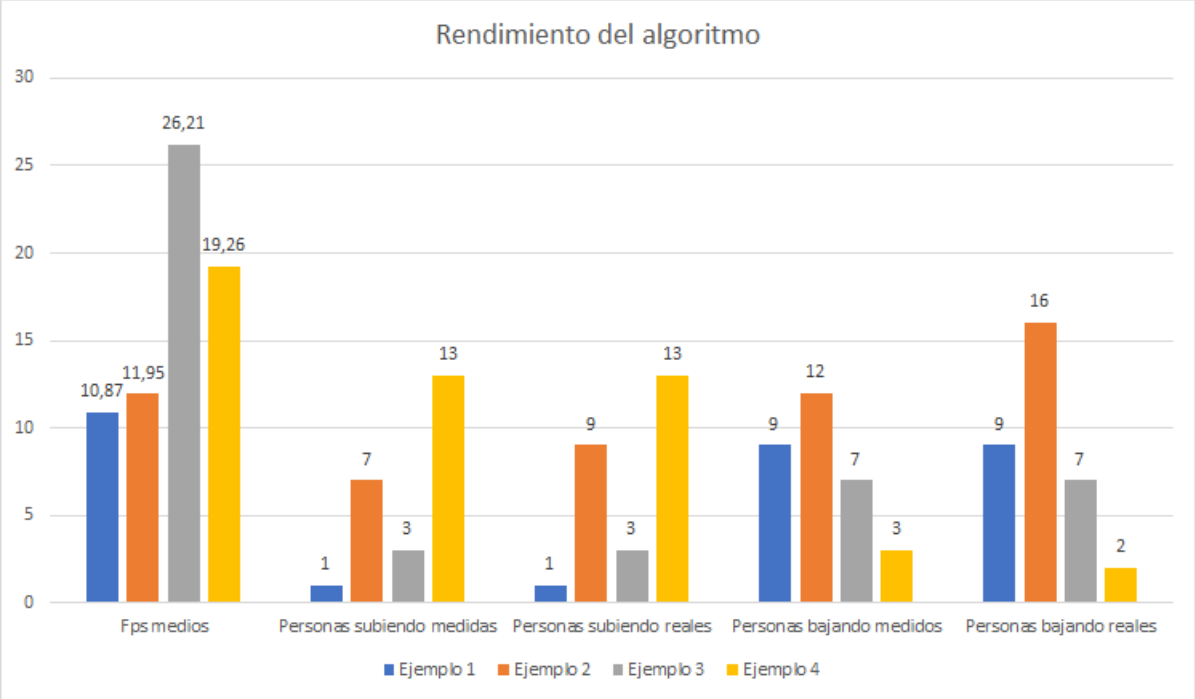


Figura 5.2: Rendimiento del algoritmo implementado

Como se discutió en el **Modelo de Comercialización 4.2**, el hosting online también es dinámico y su capacidad se puede aumentar si es necesario. Además, se podría considerar el uso del producto para fines distintos al conteo, como la videovigilancia, ya que puede detectar a una persona en las imágenes.

Respecto a las líneas futuras, cabe destacar que se ha realizado un estudio conservador ya que no utilizamos ningún software personalizado. Por lo tanto, se podría considerar el desarrollo de un algoritmo de conteo aún más ajustado al hardware utilizado, que incluso podría diseñarse para nuestro caso de uso en lugar de utilizar la Raspberry Pi.

Si se dispusiera de hardware más potente, el algoritmo también podría complicarse para ofrecer más funcionalidades, como identificar el sexo o la edad de las personas que está detectando, procesar estos datos y ofrecer información más detallada a los usuarios de la aplicación web.

Por otro lado, asumiendo que en lugar de un prototipo, ya estamos hablando de un producto final, se debe mejorar el esquema de la base de datos ya que los contadores y usuarios actualmente pertenecen a una empresa. Debería haber una tabla más, llamada *Establecimiento*, con la que organizar mejor estas relaciones.

La aplicación web debe completarse agregando diferentes pantallas de configuración para cada usuario de forma individual. Si se agrega una nueva tabla intermedia, se debe modificar cómo se muestra la información para que sea completa e intuitiva.

Capítulo 6

Summary and Conclusions

The need to have people counting devices is increasingly evident to know the number of people in our establishment in real-time.

However, we can also obtain much information from the data generated by this content, such as the hours with the highest influx of customers, or make more long-term planning knowing which days of the week more people attend.

This project has been undertaken in the most thorough way possible. We have carried out a market study wherein the principal needs and the principal trends of the market have been analyzed to shape the prototype that has been implemented.

All this has been accompanied by a feasibility analysis, which concludes that carrying out a project of these characteristics is possible, with a reasonably short development time and recovery of the initial investment.

As a result, a fully functional prototype has been obtained, offering quick and easy installation, and competitive prices compared to the competition. Besides offering the people counter, we have developed a web platform that provides access to data from any device and simple management of users and meters. Regarding the scalability of the product, it can be installed as many counters as necessary.

We can see the measured results using some videos as proof in the following figures: **Porcentaje de acierto del algoritmo implementado** and **Rendimiento del algoritmo implementado**.

As discussed in the **Marketing Model 4.2**, online hosting is also dynamic, and its capacity can be increased if necessary. In addition, the use of the product could be considered for purposes other than counting, such as video surveillance, since it can detect a person in the images.

Regarding future lines, it should be noted that a conservative study has been carried out since we do not use any custom software. Therefore, the development of a counting algorithm adjusted to the hardware used could be considered, which could even be designed for our use case instead of using the Raspberry Pi.

If more powerful hardware was available, the algorithm could also be sophisticated to offer more functionalities, such as identifying the sex or the age of the people detected, processing this data, and offering more detailed information to users of the web application.

On the other hand, assuming that instead of a prototype, we are already talking about a final product, the database schema must be improved since the counters and users currently belong to a company. There should be one more table, called *Establishment*, with which to better organize these relationships better.

The web application must be refined by adding different configuration screens for each

user individually. To conclude, it would be compelling to modify how the information is displayed if a new intermedite table is added, so that the final outlook is enhanced.

Capítulo 7

Presupuesto

A parte de la inversión necesaria para llevar a cabo el desarrollo para el contador de aforos, se ha realizado un presupuesto sobre el trabajo realizado.

Tareas	Horas	Precio
Desarrollo de un plan de negocio	30 horas	15€/hora
Estudio del hardware, tecnologías, herramientas y diseño de la aplicación Web	50 horas	15€/hora
Implementación del algoritmo de conteo	35 horas	15€/hora
Desarrollo de la aplicación Web	125 horas	15€/hora
Documentación	60 horas	15€/hora
Documentación	300 horas	4500€

Tabla 7.1: Presupuesto del proyecto

Por tanto, para realizar este proyecto hicieron falta 300 horas, con un precio final de 4500€.

Apéndice A

Código del contador de aforo

A.1. centroidTracker.py

```
from scipy.spatial import distance as dist
from collections import OrderedDict
import numpy as np

class CentroidTracker:
    def __init__(self, maxDisappeared=50, maxDistance=50):
        self.nextObjectID = 0
        self.objects = OrderedDict()
        self.disappeared = OrderedDict()

        self.maxDisappeared = maxDisappeared

        self.maxDistance = maxDistance

    def register(self, centroid):
        self.objects[self.nextObjectID] = centroid
        self.disappeared[self.nextObjectID] = 0
        self.nextObjectID += 1

    def deregister(self, objectID):
        del self.objects[objectID]
        del self.disappeared[objectID]

    def update(self, rects):
        if len(rects) == 0:
            for objectID in list(self.disappeared.keys()):
                self.disappeared[objectID] += 1
                if self.disappeared[objectID] > self.maxDisappeared:
                    self.deregister(objectID)
            return self.objects

        inputCentroids = np.zeros((len(rects), 2), dtype="int")

        for (i, (startX, startY, endX, endY)) in enumerate(rects):
```

```
            cX = int((startX + endX) / 2.0)
            cY = int((startY + endY) / 2.0)
            inputCentroids[i] = (cX, cY)

        if len(self.objects) == 0:
            for i in range(0, len(inputCentroids)):
                self.register(inputCentroids[i])

        else:
            objectIDs = list(self.objects.keys())
            objectCentroids = list(self.objects.values())

            D = dist.cdist(np.array(objectCentroids), inputCentroids)

            rows = D.min(axis=1).argsort()
            cols = D.argmin(axis=1)[rows]

            usedRows = set()
            usedCols = set()

            for (row, col) in zip(rows, cols):
                if row in usedRows or col in usedCols:
                        continue

                if D[row, col] > self.maxDistance:
                    continue

                objectID = objectIDs[row]
                self.objects[objectID] = inputCentroids[col]
                self.disappeared[objectID] = 0

                usedRows.add(row)
                usedCols.add(col)

            unusedRows = set(range(0, D.shape[0])).difference(usedRows)
            unusedCols = set(range(0, D.shape[1])).difference(usedCols)

            if D.shape[0] >= D.shape[1]:
                for row in unusedRows:
                    objectID = objectIDs[row]
                    self.disappeared[objectID] += 1

                    if self.disappeared[objectID] > self.maxDisappeared:
                        self.deregister(objectID)
            else:
                for col in unusedCols:
                    self.register(inputCentroids[col])

        return self.objects
```

A.2. trackableObject.py

```
class TrackableObject:
    def __init__(self, objectID, centroid):
        self.objectID = objectID
        self.centroids = [centroid]
        self.counted = False
```

A.3. peopleCounter.py

```
from centroidTrackable.centroidtracker import CentroidTracker
from centroidTrackable.trackableobject import TrackableObject
from imutils.video import VideoStream
from imutils.video import FPS
from dotenv import load_dotenv
import requests
import datetime
import os
import numpy as np
import argparse
import imutils
import time
import dlib
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True, help="path to Caffe 'deploy' prototxt
file")
ap.add_argument("-m", "--model", required=True, help="path to Caffe pre-trained model")
ap.add_argument("-i", "--input", type=str, help="path to optional input video file")
ap.add_argument("-o", "--output", type=str, help="path to optional output video file")
ap.add_argument("-c", "--confidence", type=float, default=0.4, help="minimum probability
to filter weak detections")
ap.add_argument("-s", "--skip-frames", type=int, default=30, help="# of skip frames
between detections")
ap.add_argument("-t", "--test", type=int, default=0,
                help="1 or 0 (default). If 0, it will connect with the database. If 1,
                it wont connect to the database, its for testing purposes.")
args = vars(ap.parse_args())

CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car",
"cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person",
"pottedplant", "sheep", "sofa", "train", "tvmonitor"]

print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
```

```
37
38  newframetime=0
39  prevframetime=0
40
41  if not args.get("input", False):
42      print("[INFO] starting video stream...")
43      vs = VideoStream(src=0).start()
44      time.sleep(2.0)
45  else:
46      print("[INFO] opening video file...")
47      vs = cv2.VideoCapture(args["input"])
48
49  writer = None
50
51  W = None
52  H = None
53
54  ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
55  trackers = []
56  trackableObjects = {}
57
58  totalFrames = 0
59  totalDown = 0
60  totalUp = 0
61  header = {}
62
63  startTime = datetime.datetime.now()
64
65  mode = 0
66
67  if args["test"] == 0:
68      load_dotenv()
69      print("[INFO] estas en modo conexion")
70      response = requests.post('http://192.168.1.57:3000/counter/login', data={'username':
71      os.getenv('USERNAME'), 'password': os.getenv('PASS')})
72      headers = {'Authorization': 'Bearer {}'.format(response.json()['accessToken'])}
73  else:
74      mode = 1
75      print("[INFO] estas en modo test")
76
77  fps = FPS().start()
78
79  while True:
80      frame = vs.read()
81      frame = frame[1] if args.get("input", False) else frame
82
83      if args["input"] is not None and frame is None:
84          break
85
86      frame = imutils.resize(frame, width=500)
```

```
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    if W is None or H is None:
        (H, W) = frame.shape[:2]

    if args["output"] is not None and writer is None:
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args["output"], fourcc, 30,
            (W, H), True)

    status = "Waiting"
    rects = []

    if totalFrames % args["skip_frames"] == 0:
        status = "Detecting"
        trackers = []

        blob = cv2.dnn.blobFromImage(frame, 0.007843, (W, H), 127.5)
        net.setInput(blob)
        detections = net.forward()

        for i in np.arange(0, detections.shape[2]):
            confidence = detections[0, 0, i, 2]

            if confidence > args["confidence"]:
                idx = int(detections[0, 0, i, 1])

                if CLASSES[idx] != "person":
                    continue

                box = detections[0, 0, i, 3:7] * np.array([W, H, W, H])
                (startX, startY, endX, endY) = box.astype("int")

                tracker = dlib.correlation_tracker()
                rect = dlib.rectangle(startX, startY, endX, endY)
                tracker.start_track(rgb, rect)

                trackers.append(tracker)

    else:
        for tracker in trackers:
            status = "Tracking"

            tracker.update(rgb)
            pos = tracker.get_position()

            startX = int(pos.left())
            startY = int(pos.top())
            endX = int(pos.right())
            endY = int(pos.bottom())
```

```
            rects.append((startX, startY, endX, endY))

    cv2.line(frame, (0, H // 2), (W, H // 2), (0, 255, 255), 2)
    objects = ct.update(rects)

    for (objectID, centroid) in objects.items():
        to = trackableObjects.get(objectID, None)

        if to is None:
            to = TrackableObject(objectID, centroid)

        else:
            y = [c[1] for c in to.centroids]
            direction = centroid[1] - np.mean(y)
            to.centroids.append(centroid)

            if not to.counted:
                if direction < 0 and centroid[1] < H // 2:
                    totalUp += 1
                    to.counted = True

                elif direction > 0 and centroid[1] > H // 2:
                    totalDown += 1
                    to.counted = True

        trackableObjects[objectID] = to

        text = "ID {}".format(objectID)
        cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0), -1)

    info = [
        ("Up", totalUp),
        ("Down", totalDown),
        ("Status", status),
    ]

    for (i, (k, v)) in enumerate(info):
        text = "{}: {}".format(k, v)
        cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

    if writer is not None:
        writer.write(frame)

    newframetime = time.time()
    framespersecond = 1/(newframetime-prevframetime)
    prevframetime = newframetime
```

```
187
188     framespersecond = int(framespersecond)
189     framespersecond = str(framespersecond)
190
191     cv2.putText(frame, framespersecond, (7, 70), cv2.FONT_HERSHEY_SIMPLEX, 1, (100, 255,
192     0), 3, cv2.LINE_AA)
193
194     cv2.imshow("Frame", frame)
195     key = cv2.waitKey(1) & 0xFF
196
197     if key == ord("q"):
198         break
199
200     totalFrames += 1
201     fps.update()
202
203     if mode == 0 and int((datetime.datetime.now() - startTime).total_seconds()) > 5:
204         startTime = datetime.datetime.now()
205         print("[INFO] sending data to backend")
206         x = requests.post('http://192.168.1.57:3000/counter/update-values',
207         data={'entering': totalUp, 'exiting': totalDown}, headers=headers)
208         print(x)
209
210
211 fps.stop()
212 print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
213 print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
214
215 print("[INFO] personas hacia arriba: " + str(totalUp))
216 print("[INFO] personas hacia abajo: " + str(totalDown))
217 print("[INFO] personas en total: " + str(totalUp + totalDown))
218
219 if writer is not None:
220     writer.release()
221
222 if not args.get("input", False):
223     vs.stop()
224
225 else:
226     vs.release()
227
228 cv2.destroyAllWindows()
```

Bibliografía

[1] *Android*. url: https://www.android.com/intl/es%5C_es/what-is-android/.

[2] *Angular*. url: https://angular.io/.

[3] *Apollo*. url: https://www.apollographql.com/.

[4] *Axios*. url: https://github.com/axios/axios.

[5] *Bcrypt*. url: https://github.com/dcodeIO/bcrypt.js/blob/master/README.md.

[6] *Chartjs*. url: https://www.chartjs.org/.

[7] MDN contributors. *CSS*. 2021. url: https://developer.mozilla.org/es/docs/Web/CSS.

[8] MDN contributors. *Promise*. 2021. url: https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Promise.

[9] Ross Girshick. *Fast R-CNN*. 2015. url: https://arxiv.org/abs/1504.08083.

[10] Ross Girshick y col. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. url: https://arxiv.org/abs/1311.2524.

[11] *GraphQL*. url: https://graphql.org/.

[12] Andrew G. Howard y col. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. url: https://arxiv.org/abs/1704.04861.

[13] *Ionic Frmework*. url: https://ionicframework.com/.

[14] *IOS*. url: https://www.apple.com/es/ios/ios-14/.

[15] *JSX*. url: https://es.reactjs.org/docs/introducing-jsx.html.

[16] *JWT*. url: https://jwt.io/.

[17] Doug Laney. *3D Data Management: Controlling Data Volume, Velocity, and Variety*. [The original article was published on this web: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf, but it has been deleted]. 2001. url: https://pdfcoffee.com/ad949-3d-data-management-controlling-data-volume-velocity-and-varietypdf-pdf-free.html.

[18] Cody Lirette. *What is the Relationship Between IoT and Big Data?* 2019. url: https://www.soracom.io/blog/what-is-the-relationship-between-iot-and-big-data/.

[19] Wei Liu y col. *SSD: Single Shot MultiBox Detector*. 2016. url: https://arxiv.org/abs/1512.02325.

[20] *MariaDB*. url: https://mariadb.org/.

[21] *MySQL*. url: https://www.mysql.com/.

[22] *NestJS*. url: https://nestjs.com/.

[23] *NodeJS*. url: https://nodejs.org/es/.

[24] *OpenCV*. url: https://opencv.org/.

[25] *OpenCV DNN*. url: https://docs.opencv.org/master/d2/d58/tutorial_table_of_content_dnn.html.

[26] Oracle. *¿Qué es Internet of Things (IoT)?* url: https://www.oracle.com/es/internet-of-things/what-is-iot/.

[27] *PostgreSQL*. url: https://www.postgresql.org/.

[28] *Python*. url: https://www.python.org/.

[29] *React*. url: https://es.reactjs.org/.

[30] *React hooks*. url: https://es.reactjs.org/docs/hooks-intro.html.

[31] *React Native*. url: https://reactnative.dev/.

[32] REALIZATOR. *Internet of Things and Big Data – Better Together*. 2018. url: https://www.whizlabs.com/blog/iot-and-big-data/.

[33] REALIZATOR. *OPENCV: COMPARING THE SPEED OF C++ AND PYTHON CODE ON THE RASPBERRY PI FOR STEREO VISION*. 2020. url: https://developer.mozilla.org/es/docs/Web/CSS.

[34] Joseph Redmon y col. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. url: https://arxiv.org/abs/1506.02640.

[35] Shaoqing Ren y col. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. url: https://arxiv.org/abs/1506.01497.

[36] *REST*. url: https://restfulapi.net/.

[37] Adrian Rosebrock. *OpenCV People Counter*. 2018. url: https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/.

[38] *RxJS*. url: https://rxjs.dev/.

[39] *SASS*. url: https://sass-lang.com/.

[40] *TypeScript*. url: https://www.typescriptlang.org/.

[41] Amit Vernam. *AI, automation and the economy: understanding industry 4.0*. url: https://enterprise.verizon.com/resources/articles/s/understanding-industry-4-0-and-how-it-will-help/.

[42] *Vue*. url: https://vuejs.org/.

[43] Wikipedia. *Base de datos relacional — Wikipedia, La enciclopedia libre*. [Internet; descargado 6-junio-2021]. 2021. url: https://es.wikipedia.org/w/index.php?title=Base_de_datos_relacional&oldid=135968432.

[44] Wikipedia. *C (lenguaje de programación) — Wikipedia, La enciclopedia libre*. [Internet; descargado 6-junio-2021]. 2021. url: https://es.wikipedia.org/w/index.php?title=C_(lenguaje_de_programaci%C3%B3n)&oldid=135092282.

[45] Wikipedia. *C++ — Wikipedia, La enciclopedia libre*. [Internet; descargado 6-junio-2021]. 2021. url: https://es.wikipedia.org/w/index.php?title=C%2B%2B&oldid=135578484.

[46] Wikipedia. *Caché (informática) — Wikipedia, La enciclopedia libre*. [Internet; descargado 6-junio-2021]. 2021. url: https://es.wikipedia.org/w/index.php?title=Cach%C3%A9_(inform%C3%A1tica)&oldid=135686291.

[47] Wikipedia. *Diagrama de Gantt — Wikipedia, La enciclopedia libre*. [Internet; descargado 6-junio-2021]. 2021. url: https://es.wikipedia.org/w/index.php?title=Diagrama_de_Gantt&oldid=136024078.

[48] Wikipedia. *Front end y back end — Wikipedia, La enciclopedia libre*. [Internet; descargado 6-junio-2021]. 2020. url: https://es.wikipedia.org/w/index.php?title=Front_end_y_back_end&oldid=131027640.

[49] Wikipedia. *HTML — Wikipedia, La enciclopedia libre*. [Internet; descargado 6-junio-2021]. 2021. url: https://es.wikipedia.org/w/index.php?title=HTML&oldid=136084861.

[50] Wikipedia. *MacOS — Wikipedia, La enciclopedia libre*. [Internet; descargado 6-junio-2021]. 2021. url: https://es.wikipedia.org/w/index.php?title=MacOS&oldid=135918894.

[51] Wikipedia. *Protocolo de transferencia de hipertexto — Wikipedia, La enciclopedia libre*. [Internet; descargado 6-junio-2021]. 2021. url: https://es.wikipedia.org/w/index.php?title=Protocolo_de_transferencia_de_hipertexto&oldid=135472492.

[52] Wikipedia. *Simple Object Access Protocol — Wikipedia, La enciclopedia libre*. [Internet; descargado 6-junio-2021]. 2021. url: https://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&oldid=133095996.