Desarrollo de un Sistema Inteligente basado en IOT e Inteligencia Artificial para la Detección y Gestión de Estacionamientos Libres y Ocupados.

Smart Parking.

Arias Máximo Agustín Augusto Jorge Horacio Brizuela Álvaro Sebastian
Frischeisein Carlos Omar González Leandro Lucas Ortega Paloma Lucia
Salomon Gustavo Samuel.

Directora: Ing. Ingeniera Constanza Román

Instituto Politécnico Formosa
Tecnicatura Superior en Telecomunicaciones

Formosa - Argentina
2025

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

LISTINGS

Capítulo 1

INTRODUCCIÓN

El crecimiento urbano y el aumento del parque automotor en las grandes urbes, han generado una demanda en crecimiento en los espacios de estacionamiento, especialmente en zonas centrales, de máxima concurrencia y de alta circulación. Esta situación provoca congestión vehicular, aumento de los tiempos de traslado, los conflictos sociales y un incremento significativo de emisiones contaminantes. La conclusión en ausencia de información en tiempo real acerca de la disponibilidad de plazas obliga a los conductores a recorrer las calles en busca de lugar, lo que intensifica el problema. En este contexto, se propone un sistema de Smart Parking que emplea tecnologías de Internet de las Cosas (IoT), basado en el análisis de videos a través de un modelo entrenado. La propuesta se distingue por su enfoque que tiene un propósito general de ser escalable. En este sentido se inició la prueba piloto en las instalaciones del Polo Científico y Tecnológico e Innovación de Formosa, utilizando cámaras IP, donde se conecta a un servidor local con Flask y un modelo pre entrenado YOLOv11. Este esquema experimental permitió validar el flujo técnico completo (captura-inferencia-visualización) con un presupuesto mínimo, generando evidencia práctica y estableciendo bases sólidas para escalar la solución por fases.

En efecto el sistema no solo busca optimizar la movilidad urbana, sino también ofrecer tanto en un entidad pública como privada una herramienta adaptable, sostenible y alineada con los principios de ciudad inteligente.

Capítulo 2

PLANTEAMIENTO DEL PROBLEMA

En la ciudad de Formosa, actualmente funciona un Sistema de Estacionamiento Medido (SEM). Sin embargo, este servicio no ofrece información dinámica sobre la ocupación de plazas, lo que limita su eficacia y profundiza la problemática del tránsito. La gestión del orden vehicular recae principalmente en el personal de la Dirección de Tránsito del Municipio, dado que no existe un mecanismo que permita conocer en tiempo real la disponibilidad de espacios libres. Esta carencia obliga a los conductores a circular innecesariamente en busca de un lugar para estacionar, generando mayor congestión, consumo de combustible y emisiones de dióxido de carbono (CO_2).

A su vez, el crecimiento urbano acelerado y el incremento sostenido del parque automotor ejercen una presión constante sobre la disponibilidad de estacionamientos. Esta situación se evidencia especialmente en las zonas de mayor concentración vehicular, como el microcentro, donde la búsqueda de una plaza libre agrava la congestión, prolonga los tiempos de traslado y contribuye al aumento de la contaminación ambiental.

Desde una perspectiva más amplia, el estacionamiento trasciende la mera gestión del tránsito y se convierte en un componente esencial de la planificación urbana y la equidad social. Tal como plantea GEA21 (s.f.) en su artículo El problema del aparcamiento, una gestión ineficiente de los espacios para estacionar provoca congestión y deterioro ambiental en las ciudades. Históricamente, la normativa urbanística se ha centrado en garantizar un número mínimo de plazas por desarrollo urbano, sin cuestionar la verdadera necesidad de estas ni considerar su impacto en la movilidad sostenible. Este enfoque ha favorecido el uso del automóvil privado, en detrimento de modos de transporte más sustentables, y ha invisibilizado aspectos clave como la accesibilidad de personas con discapacidad, el impacto territorial y las desigualdades socioeconómicas.

Por lo tanto la problemática del aparcamiento vehicular debe analizarse en varias dimensiones:

- **Social:** afecta la calidad de vida de los ciudadanos al incrementar el tiempo y la incertidumbre en la búsqueda de una plaza; puede generar conflictos por el uso desigual del espacio público y por la disponibilidad de plazas para colectivos específicos (personas con discapacidad, carga/descarga, residentes).
- **Ambiental:** la circulación innecesaria en busca de estacionamiento incrementa el consumo de combustible y las emisiones de gases de efecto invernadero, dificultando los compromisos de mitigación del cambio climático.

- **Económica:** los costos derivados de la congestión impactan tanto en la economía de los usuarios como en la productividad urbana. Además, la creación de nuevas plazas exige inversiones públicas y privadas significativas.
- **Regulatoria:** las políticas de estacionamiento requieren ser repensadas con criterios de flexibilidad, escalabilidad y justicia espacial, superando la lógica de mínimos normativos para favorecer estrategias integrales de movilidad urbana.

En definitiva, la problemática del aparcamiento vehicular no debe entenderse como una mera cuestión técnica o de oferta de plazas, sino como un desafío urbano complejo que exige articular soluciones tecnológicas con políticas regulatorias, ambientales y sociales. Solo de este modo será posible avanzar hacia un modelo de ciudad moderna, equitativa y sostenible.

Capítulo 3

JUSTIFICACIÓN

La propuesta de implementar un sistema Smart Parking basado en visión por computadora e IoT se justifica por su capacidad para abordar, de forma directa y complementaria, las carencias señaladas en el planteamiento: al ofrecer visibilidad en tiempo real sobre la disponibilidad por plaza y por sector, el sistema reduce la incertidumbre de los usuarios y el tiempo medio de búsqueda, lo que a su vez disminuye la circulación innecesaria, el consumo de combustible y las emisiones contaminantes. Esta mejora operativa no es un fin en sí mismo sino un habilitador: permite generar datos objetivos que orienten decisiones de planificación -por ejemplo, zonificaciones de uso, prioridades de acceso y criterios de gestión del espacio público y proporcionan evidencia para políticas que prioricen la sostenibilidad y la equidad.

En términos económicos, la utilización de hardware accesible y soluciones de software abiertas reduce la barrera de entrada y facilita la replicabilidad del proyecto en contextos institucionales con recursos limitados, permitiendo que el instituto desarrolle capacidades técnicas locales (procesamiento en borde, gestión de sensores, evaluación de modelos) que pueden aprovecharse en actividades docentes, de investigación y extensión. Desde el punto de vista técnico y operativo, adoptar un enfoque por fases -iniciando con un prototipo mínimo viable para validar precisión, latencias y condiciones reales- minimiza riesgos, permite ajustar modelos y procedimientos y genera informes de impacto que sostienen decisiones futuras.

Finalmente, la propuesta incorpora principios de gobernanza responsable (anonimización de imágenes, retención limitada, transparencia en el uso de datos) que reducen riesgos legales y sociales, y posiciona al proyecto como una contribución práctica y académica: no sólo resuelve una necesidad urbana concreta, sino que también produce conocimiento, herramientas y metodologías reutilizables para la gestión inteligente y más justa del espacio público.

Capítulo 4

OBJETIVO

4.1 Objetivo General

Desarrollar e implementar un sistema de Smart Parking en la ciudad de Formosa, basado en el uso de cámaras IP y tecnologías de Internet de las Cosas (IoT), con el propósito de ordenar y optimizar el estacionamiento público, mejorar la movilidad urbana y fomentar una ciudad más eficiente, inclusiva y sustentable, integrando los aspectos sociales, ambientales, económicos y tecnológicos del entorno local.

4.2 Objetivos Específicos

- Implementar un piloto inicial con una cámara IP para validar el flujo técnico.
- Ejecutar la inferencia de ocupación de parking con YOLOv11 sobre un servidor local con Flask.
- Visualizar métricas y disponibilidad en Grafana.
- Recolectar imágenes etiquetadas para conformar un dataset local adaptado al entorno de Formosa.
- Diseñar un plan de escalado por fases: validación piloto, entrenamiento local, ampliación a varios nodos y despliegue urbano.
- Integrar herramientas de monitoreo (prometheus) para garantizar la disponibilidad y desempeño del sistema.
- Evaluar la viabilidad de integrar el sistema con redes definidas por software (SDN) para una administración más eficiente en futuras etapas.

Capítulo 5

HIPÓTESIS

Es posible diseñar y elaborar un sistema de Smart Parking basado en cámaras IP e inteligencia artificial que sea capaz de detectar espacios libres y ocupados con una precisión de un 95 %.

Capítulo 6

MARCO TEÓRICO

6.1 Terminología

Sistema IoT (Internet of Things) es un conjunto de dispositivos interconectados a través de Internet que recopilan, envían y analizan datos del entorno. Funciona mediante sensores, redes de comunicación y plataformas que procesan la información en tiempo real.

YOLO (You Only Look Once) es un algoritmo de detección de objetos en tiempo real que procesa una imagen completa en una sola pasada, lo que permite identificar y localizar múltiples objetos simultáneamente. Su funcionamiento se basa en una red neuronal convolucional que divide la imagen en una cuadrícula y predice las clases y ubicaciones de los objetos en cada celda.

API (Interfaz de Programación de Aplicaciones) es un conjunto de definiciones y protocolos que permiten la comunicación entre diferentes componentes de software. Su funcionamiento radica en facilitar la integración y el intercambio de datos entre sistemas, brindando acceso a funciones o servicios de otras aplicaciones sin conocer su código interno.

API REST (Representational State Transfer) es una interfaz que sigue los principios del estilo arquitectónico REST, empleando métodos HTTP estándar como GET, POST, PUT y DELETE para acceder y manipular recursos. Funciona permitiendo la comunicación eficiente entre sistemas distribuidos mediante el intercambio de representaciones de recursos a través de URLs.

Flask es un micro framework para el desarrollo de aplicaciones web escrito en Python, que ofrece flexibilidad y simplicidad al desarrollador. Su funcionamiento se basa en proporcionar herramientas para manejar solicitudes HTTP, rutas y plantillas HTML, permitiendo construir aplicaciones de manera modular.

Grafana es una plataforma de análisis y monitoreo de código abierto utilizada para visualizar métricas y registros en tiempo real. Funciona conectándose a múltiples fuentes de datos y generando paneles (dashboards) interactivos que facilitan la observación de tendencias y el análisis del rendimiento.

Wi-Fi es una tecnología de comunicación inalámbrica que permite la conexión de dispositivos a redes locales sin el uso de cables. Su funcionamiento utiliza ondas de radio y protocolos IEEE 802.11 para transmitir datos entre puntos de acceso y dispositivos conectados.

Internet es una red global de computadoras interconectadas que posibilita el intercambio de información y servicios en todo el mundo. Funciona mediante el protocolo TCP/IP, que gestiona la transmisión de datos y la conexión entre redes.

La nube (Cloud Computing) es un modelo de servicio que permite el acceso remoto a recursos informáticos, almacenamiento y software a través de Internet. Funciona mediante centros de datos distribuidos que alojan la infraestructura, accesible desde cualquier lugar y dispositivo.

Inteligencia Artificial (IA) es un campo de la informática que busca desarrollar sistemas capaces de realizar tareas que requieren inteligencia humana, como el aprendizaje, la percepción o la toma de decisiones. Funciona mediante algoritmos de aprendizaje automático y modelos de procesamiento de datos.

Base de Datos es una colección organizada de información estructurada que puede ser almacenada, recuperada y gestionada eficientemente. Funciona a través de sistemas gestores (DBMS) que permiten crear, leer, actualizar y eliminar datos de manera controlada.

Datos son unidades de información que representan hechos o conceptos, susceptibles de ser procesados y analizados. Funcionan como la materia prima que alimenta sistemas de información y modelos de decisión.

Aplicación Web es un software que se ejecuta en un servidor y al que se accede mediante un navegador. Su funcionamiento consiste en que el cliente envía peticiones al servidor, el cual procesa la información y devuelve una respuesta visual al usuario.

Máquina Virtual es un entorno virtual que emula un sistema operativo y hardware físico dentro de otro sistema. Funciona utilizando recursos compartidos del sistema anfitrión, creando un espacio aislado para ejecutar programas de forma independiente.

SDN (Software Defined Networking) es una arquitectura de red que separa el plano de control del plano de datos, permitiendo la gestión centralizada del tráfico de red. Funciona mediante controladores programables que administran los flujos de información de manera dinámica.

Protocolo es un conjunto de normas que definen cómo los dispositivos en una red se comunican e intercambian información. Su funcionamiento garantiza que los datos sean enviados y recibidos correctamente, estableciendo estructuras de paquetes y reglas de transmisión.

Ryu es un controlador SDN de código abierto que permite programar redes de forma centralizada mediante OpenFlow. Funciona gestionando los flujos de tráfico entre dispositivos de red y facilitando la automatización de su comportamiento.

Cámara IP es un dispositivo de videovigilancia que transmite imágenes y audio mediante una red IP, permitiendo su visualización remota. Su funcionamiento se basa en capturar datos visuales y enviarlos a un servidor o nube mediante protocolos de red.

Python es un lenguaje de programación interpretado, de alto nivel y propósito general, reconocido por su sintaxis sencilla y legible. Funciona mediante un intérprete que ejecuta código línea por línea, lo que facilita el desarrollo rápido de aplicaciones.

Dashboard es una interfaz visual que presenta indicadores clave y métricas relevantes para la toma de decisiones. Funciona integrando datos de múltiples fuentes y representándolos gráficamente mediante paneles personalizables.

MongoDB es un sistema de gestión de bases de datos NoSQL orientado a documentos, diseñado para manejar datos no estructurados en formato JSON. Funciona almacenando documentos flexibles en lugar de tablas, permitiendo alta escalabilidad y velocidad de acceso.

Back End es la parte del desarrollo web encargada de la lógica, procesamiento y comunicación con la base de datos. Funciona en el servidor y se comunica con el Front End mediante API o peticiones HTTP.

Front End es la parte visible de una aplicación web que interactúa con el usuario. Su funcionamiento se basa en el uso de HTML, CSS y JavaScript para crear interfaces dinámicas que se comunican con el servidor a través de peticiones asincrónicas.

6.2 Marco Legal

El marco legal aplicable a la videovigilancia y a la gestión de estacionamientos y tránsito contempla diversas normativas que regulan la competencia, habilitación y cumplimiento necesarios para la implementación del sistema Smart Parking en la provincia de Formosa. Entre las disposiciones más relevantes se encuentran las siguientes:

- **Ley N.º 25.326 - Protección de Datos Personales (Argentina, 2000, arts. 1 y 21 incs. 1, 2 y 3)** Esta ley tiene por objeto la protección integral de los datos personales asentados en archivos, registros, bancos de datos u otros medios técnicos, sean estos públicos o privados, destinados a la provisión de información. Su finalidad es garantizar el derecho a la intimidad y a la autodeterminación informativa de las personas, asegurando que los titulares puedan acceder, controlar y disponer de los datos que les conciernen. A tales efectos, la norma establece que el responsable del banco de datos deberá proporcionar su nombre y domicilio, detallar los mecanismos de recolección, actualización e interrelación de la información, garantizar la seguridad y confidencialidad de los registros, e indicar el plazo de conservación, finalidad y destino de los datos personales. Asimismo, debe precisarse quiénes podrán acceder o intervenir en el procesamiento de dicha información, asegurando a

los titulares el ejercicio de los derechos de acceso, rectificación, actualización y supresión contemplados por la ley.

- **Ley N.º 27.078 - Argentina Digital (Argentina, 2014, arts. 1-3)** Esta ley, promulgada el 16 de diciembre de 2014, tiene por objeto declarar de interés público el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC), así como fomentar la infraestructura digital y la conectividad en todo el país. Su finalidad es garantizar que las soluciones tecnológicas se integren de manera eficiente y segura dentro de los entornos digitales. En el marco del proyecto Smart Parking, esta norma respalda la implementación de sistemas basados en IoT (Internet de las Cosas) para la gestión de estacionamientos, optimizando la disponibilidad de espacios y la movilidad urbana, dentro del Polo Tecnológico.

- **Ley N.º 26.388 - Delitos Informáticos (Argentina, 2008, art. 157 bis y concordantes)** Promulgada el 4 de junio de 2008, esta ley incorpora al Código Penal la protección frente a accesos no autorizados, daños o manipulación de datos informáticos. Establece sanciones para quienes vulneren sistemas de información, garantizando la integridad y confidencialidad de los datos. En el contexto del Smart Parking, obliga a implementar medidas de seguridad en la red y servidores, como contraseñas robustas, cifrado de información y control de accesos, protegiendo así la información de los usuarios y evitando ataques informáticos.

- **Ley N.º 24.449 - Ley Nacional de Tránsito (Argentina, 195, arts. 1 y 3)**Promulgada el 23 de diciembre de 1994, regula la circulación y el uso del espacio público, incluyendo vehículos, peatones y la infraestructura vial. Para el proyecto Smart Parking, esta norma permite gestionar de manera ordenada el tránsito dentro del Polo Tecnológico, optimizando la utilización del espacio de estacionamiento, reduciendo la congestión vehicular y mejorando la seguridad vial, contribuyendo a un entorno urbano más eficiente y seguro.

6.3 Estándar IEEE 802.11: Fundamento de la Conectividad Inalámbrica

El estándar IEEE 802.11 define los principios técnicos que rigen las redes de comunicación inalámbricas conocidas como Wi-Fi. Según Tanenbaum y Wetherall (2011), este conjunto de normas especifica los métodos de modulación, control de acceso al medio mediante CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance), y los mecanismos de cifrado y autenticación que permiten el funcionamiento confiable de las redes locales inalámbricas (LAN). El estándar emplea ondas de radio en las bandas de 2,4 GHz y 5 GHz, eliminando la necesidad de cables físicos y facilitando la instalación en entornos amplios o con restricciones estructurales.

En el contexto del sistema de estacionamiento inteligente, el uso de IEEE 802.11 posibilita que las cámaras IP transmitan de manera inalámbrica sus flujos de video hacia el servidor de procesamiento. La estabilidad del enlace y el ancho de banda que ofrece este estándar garantizan la llegada de los datos visuales en tiempo real, condición necesaria para la detección y análisis mediante inteligencia artificial. Por tanto, IEEE 802.11 constituye la base física y de enlace sobre la

cual se sustentan los demás mecanismos de comunicación del sistema (Tanenbaum & Wetherall, 2011).

6.4 Estándar ONVIF: Interoperabilidad entre Dispositivos de Video IP

El ONVIF (Open Network Video Interface Forum) es un estándar que define un lenguaje común para los sistemas de videovigilancia basados en IP. A través de servicios estructurados en SOAP (Simple Object Access Protocol), ONVIF permite que cámaras, grabadores y servidores de diferentes fabricantes puedan comunicarse y compartir información sin incompatibilidades. Este protocolo facilita funciones como el descubrimiento automático de dispositivos, la configuración remota, el acceso a flujos de video y la gestión de usuarios.

En el sistema de estacionamiento inteligente, ONVIF cumple un rol esencial al permitir que el servidor identifique automáticamente cada cámara instalada en la red y acceda a sus transmisiones. Gracias a este estándar, el sistema puede integrar cámaras de distintas marcas bajo un mismo entorno de gestión, garantizando compatibilidad, estandarización y reducción de costos de integración. (Open Network Video Interface Forum, 2024)

6.5 Arquitectura TCP/IP y el Protocolo de Internet (IP)

Sobre la conectividad inalámbrica se desarrolla la arquitectura TCP/IP, encargada de organizar y estructurar el intercambio de información entre dispositivos conectados. De acuerdo con Tanenbaum y Wetherall (2011), esta arquitectura se basa en un modelo de cuatro capas que gestiona desde el transporte hasta la aplicación, garantizando la entrega confiable de los datos en redes locales y globales.

El Protocolo de Internet (IP) realiza el direccionamiento lógico de los paquetes, asegurando que cada dispositivo -ya sea una cámara, el servidor o un cliente remoto- posea una identificación única. En este sistema se emplea IPv4, por su alta compatibilidad con equipos comerciales. Asimismo, el NAT (Network Address Translation) permite que múltiples nodos compartan una sola dirección IP pública, y el DNS (Domain Name System) traduce nombres de dominio a direcciones numéricas comprensibles por las máquinas.

Este conjunto de protocolos posibilita la identificación, enrutamiento y administración eficiente de los datos generados por cada cámara, manteniendo la coherencia de las transmisiones simultáneas. De este modo, TCP/IP constituye la columna vertebral lógica de la infraestructura de red. (Tanenbaum & Wetherall, 2011)

6.6 Protocolo DHCP: Configuración Dinámica de Dispositivos

El DHCP (Dynamic Host Configuration Protocol) automatiza la asignación de direcciones IP y otros parámetros esenciales de red, tales como la máscara de subred, la puerta de enlace y los servidores DNS. Según Tanenbaum y Wetherall (2011), este protocolo evita la configuración ma-

nual de cada dispositivo al proporcionar de manera automática los valores requeridos al momento de la conexión.

En el sistema de estacionamiento, cada cámara IP envía una solicitud de configuración y recibe los datos necesarios para integrarse a la red, garantizando una estructura organizada y sin conflictos de direcciones. Este proceso no solo reduce errores humanos, sino que además favorece la escalabilidad del sistema, permitiendo agregar nuevos nodos sin requerir conocimientos técnicos avanzados. (Tanenbaum & Wetherall, 2011)

6.7 Protocolo RTSP: Transmisión de Video en Tiempo Real

El RTSP (Real-Time Streaming Protocol) es un protocolo de control de sesiones multimedia que posibilita la transmisión de audio o video en tiempo real entre clientes y servidores. Tanenbaum y Wetherall (2011) explican que el RTSP utiliza comandos similares a los de HTTP -como PLAY, PAUSE o TEARDOWN- para controlar la reproducción de flujos multimedia. Trabajando junto con el RTP (Real-time Transport Protocol), este protocolo proporciona baja latencia y sincronización temporal, características esenciales para aplicaciones que requieren procesamiento en vivo.

En el sistema propuesto, RTSP permite que las cámaras IP envíen sus secuencias de video directamente al servidor, donde se procesan mediante el modelo YOLOv11. Gracias a ello, el flujo de imágenes se mantiene continuo y con mínima demora, condición indispensable para la detección en tiempo real de vehículos y espacios disponibles. (Tanenbaum & Wetherall, 2011)

6.8 Protocolo HTTP: Comunicación con la Interfaz de Usuario

El HTTP (Hypertext Transfer Protocol) es el protocolo base de la capa de aplicación que regula la comunicación entre el servidor y los clientes, como las interfaces web o móviles. Según Tanenbaum y Wetherall (2011), HTTP opera bajo un modelo de solicitud y respuesta, en el que el cliente envía peticiones y el servidor devuelve recursos o datos, generalmente en formato HTML o JSON. En su versión segura, HTTPS, este protocolo emplea TLS/SSL para cifrar las comunicaciones y proteger la información frente a accesos no autorizados.

En el proyecto de estacionamiento inteligente, HTTP permite que los resultados del análisis del servidor -por ejemplo, la disponibilidad de espacios- sean accesibles para el usuario final. Así, HTTP actúa como puente entre el procesamiento técnico y la experiencia de usuario, traduciendo la información proveniente de la IA en datos visuales comprensibles desde cualquier dispositivo conectado. (Tanenbaum & Wetherall, 2011)

6.9 Las Redes Definidas por Software (SDN)

Las Redes Definidas por Software (SDN, Software Defined Networking) representan un cambio estructural en el modo en que se diseñan y administran las redes modernas. Este paradigma

se basa en la separación del plano de control y el plano de datos, lo que permite que las decisiones de enrutamiento, seguridad y priorización de tráfico sean gestionadas centralmente mediante software, en lugar de depender de la configuración individual de cada dispositivo físico (Kreutz et al., 2015).

En las redes tradicionales, cada router o switch ejecuta sus propias reglas y políticas locales. En contraposición, SDN introduce una capa de abstracción lógica que centraliza la inteligencia de red en un controlador, desde el cual se puede programar, monitorear y optimizar el comportamiento de toda la infraestructura (ONF, 2022). Este enfoque posibilita una red más ágil, programable y adaptable, donde las políticas pueden modificarse en tiempo real según las condiciones del tráfico o los requerimientos del sistema.

En términos generales, SDN convierte la red en un entorno directamente programable, proporcionando una visión global del estado de todos los nodos y enlaces. Esto facilita la automatización de tareas, la respuesta rápida ante fallas y la implementación de políticas avanzadas de calidad de servicio (QoS) y seguridad.

6.9.1 Arquitectura General del SDN

La arquitectura SDN se compone de tres capas fundamentales (NoviFlow, 2021):

- **Capa de Aplicación:** alberga el software que define políticas de red y solicita recursos (por ejemplo, sistemas de gestión de tráfico o aplicaciones de análisis).
- **Capa de Control:** corresponde al controlador SDN, encargado de traducir las políticas de la capa superior en reglas de flujo específicas que serán aplicadas en los dispositivos de red.
- **Capa de Infraestructura o Datos:** compuesta por los switches físicos o virtuales que ejecutan las reglas de reenvío definidas por el controlador.

La interacción entre estas capas se realiza a través de interfaces estandarizadas. La comunicación descendente (southbound) se realiza, en la mayoría de los casos, mediante el protocolo OpenFlow, mientras que la comunicación ascendente (northbound) se establece mediante API abiertas que permiten a las aplicaciones controlar y consultar el estado de la red (Nutanix, 2020). Esta estructura modular y jerárquica permite tratar toda la red como un único “conmutador lógico", simplificando la administración y la integración de servicios avanzados, como balanceo de carga, filtrado dinámico y priorización de tráfico.

6.9.2 Protocolo OpenFlow

El protocolo OpenFlow es el componente esencial del paradigma SDN, ya que define el mecanismo de comunicación entre el controlador (plano de control) y los switches (plano de datos). Fue desarrollado por la Open Networking Foundation (ONF) y se ha convertido en el estándar más utilizado para implementar redes definidas por software (ONF, 2022).

Mediante OpenFlow, el controlador instala en los switches reglas denominadas *flow entries*, las cuales determinan cómo procesar cada paquete: reenviarlo, redirigirlo o descartarlo. Esto posibilita que los administradores y desarrolladores programen el comportamiento de la red de manera centralizada, sin necesidad de intervenir en cada dispositivo físico. Como señala McKeown et al. (2008), OpenFlow “permite programar la red como si se tratara de un sistema operativo", otorgando un control total sobre el flujo de datos.

En la situación del proyecto Smart Parking, OpenFlow es utilizado para:

- Asignar prioridad alta a los paquetes RTSP provenientes de las cámaras IP.
- Limitar el ancho de banda de flujos secundarios (HTTP / REST).
- Bloquear direcciones IP sospechosas o no autorizadas.

Estas acciones garantizan baja latencia y seguridad, evitando congestiones en el servidor Flask que procesa los flujos de video.

6.9.3 Open vSwitch (OVS)

Open vSwitch (OVS) es un switch virtual de código abierto diseñado para entornos virtualizados y compatible con el estándar OpenFlow (Pfaff et al., 2015). Cumple la función de plano de datos, ejecutando las reglas de reenvío definidas por el controlador Ryu. Cada instancia de OVS clasifica los paquetes de red según su origen, destino y tipo de servicio, aplicando colas de prioridad o políticas de calidad de servicio (Quality of Service, QoS).

En el sistema Smart Parking:

- Los flujos RTSP de vídeo reciben prioridad alta.
- Las peticiones HTTP o API internas se asignan a colas de prioridad inferior.

Además, OVS permite aplicar reglas de firewall virtual y exportar métricas de tráfico (por ejemplo, bytes transmitidos) hacia sistemas de observabilidad como Prometheus. Su naturaleza programable lo convierte en una herramienta esencial para la administración flexible, segura y económica de redes en entornos virtuales.

6.9.4 Controlador Ryu

El Controlador Ryu es un framework SDN de código abierto, escrito en Python, que proporciona una API flexible para implementar controladores personalizados y gestionar dispositivos compatibles con OpenFlow (Ryu Project, 2023). Ryu permite desarrollar aplicaciones que administran dinámicamente los flujos de red, configurando rutas, priorizando tráfico o reaccionando ante eventos de congestión o fallo.

En el sistema Smart Parking, el controlador Ryu cumple las siguientes funciones principales:

1. **Gestión dinámica de flujos:** establece las rutas óptimas entre las cámaras IP (emisoras de video RTSP) y el servidor Flask encargado del procesamiento.
2. **Priorización de video:** asigna alta prioridad a los paquetes de video en tiempo real, manteniendo la latencia baja en la transmisión.
3. **Reacción ante anomalías:** detecta congestiones o fallos y modifica las reglas OpenFlow en tiempo real para mantener la estabilidad del sistema.

Esta capacidad de reconfiguración inmediata convierte al controlador en el cerebro operativo de la red, permitiendo una gestión automatizada y coherente con los requisitos de un entorno urbano inteligente.

6.9.5 Herramientas de monitoreo: Prometheus y Grafana

El monitoreo y la observabilidad son aspectos clave en la gestión moderna de redes SDN. En este proyecto, se emplean las herramientas Prometheus y Grafana para recopilar, almacenar y visualizar métricas de rendimiento del sistema.

- **Prometheus** actúa como recolector de métricas, realizando consultas periódicas (scrapes) sobre los endpoints expuestos por Ryu y OVS. Registra datos como uso de CPU, latencia, tráfico de red y volumen de bytes transmitidos.
- **Grafana**, en cambio, se conecta a la base de datos de Prometheus y presenta la información en dashboards interactivos, permitiendo detectar anomalías como sobrecarga de enlaces o picos de tráfico en tiempo real.

Mediante esta integración, se logra una visibilidad completa sobre el estado del sistema y la red, habilitando respuestas preventivas y corrección temprana de fallas.

El uso conjunto de SDN con Prometheus y Grafana proporciona un entorno controlado, dinámico y transparente, capaz de mantener baja latencia, estabilidad operativa y confiabilidad en las transmisiones de vídeo en tiempo real. La adopción del paradigma SDN dentro del sistema Smart Parking constituye una base sólida para la gestión inteligente del tráfico de datos urbano. El modelo centralizado de control -encabezado por el controlador Ryu y complementado por Open vSwitch- permite priorizar flujos críticos, aplicar políticas de seguridad y escalar recursos dinámicamente. La incorporación de herramientas de monitoreo basadas en métricas (Prometheus y Grafana) asegura una administración proactiva y un control continuo del rendimiento de red.

En síntesis, la infraestructura SDN implementada integra los principios teóricos de redes programables con una arquitectura práctica y escalable, garantizando eficiencia, confiabilidad y adaptabilidad en un entorno de videovigilancia urbana en tiempo real.

6.10 Antecedentes

Por lo que se refiere a antecedentes se presentan:

- **“El auge de los Smart Parkings en América Latina"(Forbes Argentina, 2022)** Este artículo describe el crecimiento del sector de estacionamientos inteligentes en la región, destacando la experiencia de un empresario que logró expandir su modelo de gestión digital de estacionamientos en diversos países latinoamericanos. El caso muestra cómo el uso de tecnologías de automatización, sensores y cámaras permitió optimizar el uso del espacio urbano y reducir la congestión vehicular. Además, evidencia una creciente tendencia hacia la digitalización de la movilidad urbana, alineada con las políticas de ciudades inteligentes impulsadas en América Latina. El análisis de Forbes resalta la importancia del desarrollo de software propio, la integración de cámaras IP y la conexión con aplicaciones móviles para gestión remota y pago electrónico. Este antecedente demuestra que el mercado latinoamericano presenta gran receptividad a soluciones IoT escalables, siendo un contexto favorable para la aplicación del sistema Smart Parking basado en visión artificial y control centralizado.

- **“Primera plaza inteligente de Latinoamérica"(Enel X Chile, 2021)** Este caso presentado por Enel X expone la creación de la primera plaza pública inteligente en Latinoamérica, que integra iluminación conectada, cámaras de videovigilancia, Wi-Fi gratuito, estaciones de carga eléctrica y sensores ambientales. La implementación demuestra cómo la infraestructura urbana puede transformarse mediante IoT, IA y conectividad en red, ofreciendo servicios más eficientes y sostenibles. En el contexto del proyecto Smart Parking, este antecedente es relevante porque muestra la viabilidad de combinar cámaras IP con sistemas de análisis inteligente en espacios públicos, sin necesidad de sensores físicos individuales. Además, la iniciativa de Enel X evidencia que los proyectos de ciudad inteligente deben centrarse en soluciones energéticamente eficientes, de bajo mantenimiento y alta escalabilidad, principios compartidos con la propuesta desarrollada en el presente proyecto.

- **“ESParking: solución IoT para la gestión integral de estacionamientos"(EXO Linked, 2023)** El sistema ESParking desarrollado por EXO Linked representa una solución IoT orientada a la gestión integral de estacionamientos urbanos. Utiliza sensores, cámaras, parquímetros y software de administración central para gestionar la ocupación, cobro y control en tiempo real. Este proyecto se centra en la interoperabilidad de dispositivos y plataformas, empleando tecnologías de red estandarizadas (HTTP, MQTT, APIs REST) para asegurar conectividad continua. A diferencia del sistema ESParking, el proyecto Smart Parking propuesto se apoya únicamente en cámaras IP y visión artificial (YOLO v11), reduciendo costos y dependencia de hardware adicional. Ambas soluciones comparten el objetivo de mejorar la experiencia del usuario y la eficiencia urbana, pero la propuesta local aporta una inno-

vación técnica al reemplazar sensores físicos por detección de imágenes basada en IA, alineada con las tendencias de automatización inteligentes observadas internacionalmente.

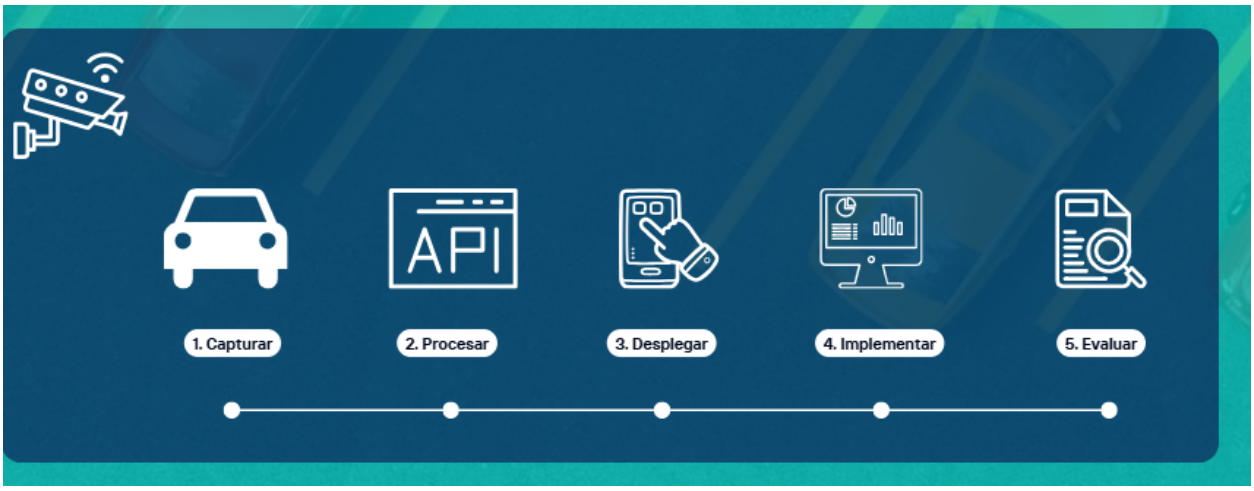
- **“Automatización y control vehicular inteligente” (Ezytec, 2023)** La empresa Ezytec publica experiencias en implementación de soluciones de Smart Parking y control vehicular automatizado en estacionamientos privados y centros comerciales. Sus sistemas integran barreras automáticas, cámaras de reconocimiento de matrículas (LPR) y aplicaciones móviles para gestionar el acceso sin contacto físico. Este caso destaca la importancia de la automatización sin contacto, tendencia acentuada tras la pandemia de COVID-19, y refuerza la necesidad de optimizar procesos urbanos mediante inteligencia artificial y visión por computadora. El proyecto Smart Parking retoma este principio, utilizando cámaras IP EZVIZ C6N y procesamiento local mediante Flask + YOLOv11, logrando así una alternativa austera, escalable y adaptable al contexto urbano argentino.

- **“Implementación de estacionamientos inteligentes en la Ciudad de Buenos Aires"(Clarín, 2024)** Según el diario Clarín, el Gobierno de la Ciudad de Buenos Aires anunció planes para la instalación de sistemas de estacionamiento inteligente como parte de su estrategia de modernización del tránsito urbano. El proyecto incluye el uso de cámaras y sensores conectados para detectar disponibilidad de espacios, transmitir datos en tiempo real y mejorar la gestión de estacionamientos públicos. Este antecedente muestra que existe interés institucional y estatal en incorporar tecnologías inteligentes para resolver problemáticas de tránsito y estacionamiento en Argentina, generando así un entorno favorable para el desarrollo de soluciones locales como Smart Parking. La diferencia principal es que la propuesta presentada se basa en hardware comercial de bajo costo y procesamiento con inteligencia artificial, reduciendo los costos de infraestructura sin perder funcionalidad.

Capítulo 7

METODOLOGÍA

Figura 7.1. Metodología del Sistema Smart Parking en 5 etapas.



7.0.1 Configuración de la Cámara IP

En primer lugar, se instala y configura la cámara IP en un punto estratégico que permita la visualización completa del área de estacionamiento. Se asigna una dirección IP fija para garantizar su accesibilidad dentro de la red local. La cámara transmite las imágenes en tiempo real hacia el servidor o computadora central.

7.0.2 Conexión Mediante Cable de Red (Ethernet)

La cámara IP se conecta físicamente al router o switch a través de un cable de red categoría 5e o superior, lo que asegura una comunicación estable y de alta velocidad entre los dispositivos.

7.0.3 Configuración del Router

El router se configura para gestionar la red local del sistema, asignando direcciones IP mediante DHCP o configuraciones estáticas según la topología definida. Además, se habilita el acceso

remoto si se requiere visualizar el sistema desde fuera de la red local.

7.0.4 Instalación y Configuración de la Computadora

La computadora actúa como servidor de procesamiento. Se instalan los programas necesarios para la recepción de la señal de video, el análisis de imágenes mediante visión por computadora (por ejemplo, con modelos basados en YOLOv11), y la gestión de la base de datos de ocupación de plazas.

7.0.5 Pruebas de Funcionamiento y Calibración

Una vez interconectados los equipos, se realiza una verificación del flujo de datos desde la cámara hasta el sistema de visualización. Se calibra la detección de objetos para reconocer vehículos y se ajustan los parámetros de red para optimizar la transmisión y almacenamiento de video.

7.0.6 Integración Final del Sistema Smart Parking

Finalmente, todos los dispositivos trabajan de forma conjunta: la cámara IP captura las imágenes, el router gestiona la conexión, el cable de red asegura la comunicación física, y la computadora procesa y muestra la información sobre el estado de las plazas disponibles en tiempo real.

7.0.7 Capturar

En la primera etapa, el sistema utiliza una cámara IP o un dispositivo de visión por computadora para capturar imágenes o secuencias de vídeo del área de estacionamiento. Esta fase es esencial para obtener información en tiempo real sobre la presencia o ausencia de vehículos en las distintas plazas disponibles.

7.0.8 Procesar

Los datos capturados son enviados a un servidor o módulo de procesamiento, donde un modelo de inteligencia artificial (IA) analiza las imágenes. Mediante técnicas de visión por computadora, como la detección de objetos con redes neuronales convolucionales (por ejemplo, modelos YOLOv11 o similares), el sistema identifica vehículos y determina el estado de ocupación de cada plaza.

7.0.9 Desplegar

Una vez procesada la información, el sistema la despliega visualmente en una interfaz o aplicación web. El usuario puede visualizar en tiempo real la disponibilidad de estacionamientos a

través de un panel de control o dashboard, o mediante una app accesible desde dispositivos móviles.

7.0.10 Implementar

En esta fase, el sistema se integra en el entorno real de operación, conectando cámaras, servidores y redes de comunicación. Se establecen los parámetros de funcionamiento, la configuración del software y las rutinas de mantenimiento preventivo para asegurar la estabilidad del sistema.

7.0.11 Evaluar

Finalmente, se realiza una evaluación de desempeño para verificar la eficacia del sistema. Se analizan métricas como la precisión de detección, el tiempo de respuesta y la estabilidad de la red. Los resultados obtenidos permiten realizar ajustes y mejoras continuas que optimizan la funcionalidad del sistema.

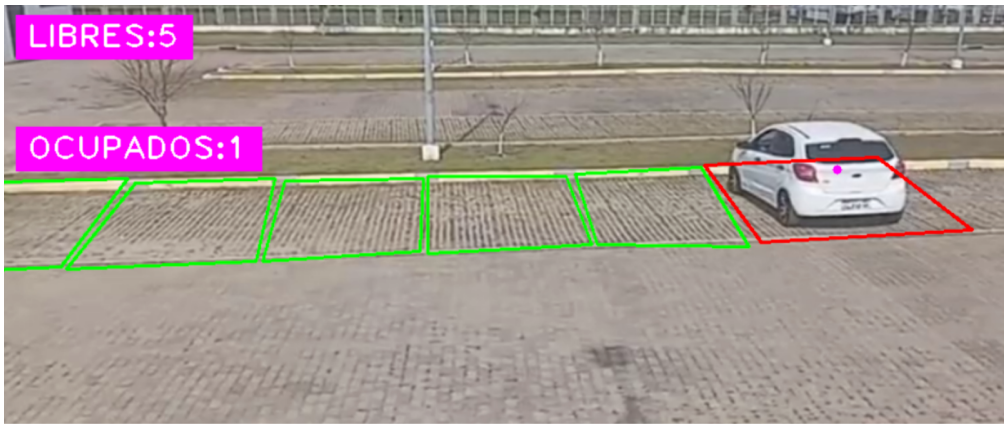
7.0.12 Ejecución del Aplicativo en Python para Detección RGB

```
from flask import Flask, render_template, request
from ultralytics import YOLO
import os

#from app flask
app = Flask(__name__)

# Directorio para guardar, imagenes subidas
UPLOAD_FOLDER = 'static/uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Cargar el modelo YOLO (por ejemplo YOLOv12)
model = YOLO('yolov8s.pt') # Tambien puedes usar yolov8n.pt o yolov8m.pt

@app.route('/')
def index():
    """P gina principal"""
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    """Ruta para subir imagen y detectar objetos"""
    if 'file' in request.files:
        # ... (El contenido de la imagen est  truncado)
```

Listing 7.1. Código de la aplicación Flask en Python para detección.

El proceso comienza con la ejecución de la aplicación principal en Python (app.py), la cual activa el módulo RGB encargado de analizar las imágenes capturadas por la cámara IP. Mediante el uso de polígonos predefinidos, el programa identifica las áreas de estacionamiento y determina si se encuentran ocupadas o libres, basándose en los valores de color o contraste de la imagen.

Figura 7.2. Detección en tiempo real de plazas libres (5) y ocupadas (1).



7.0.13 Generación del Archivo JSON

```
{
    "_id": "polygon1",
    "points": [[137, 193], [185, 193], [182, 221], [127, 217]],
    "occupied": false
},
{
    "_id": "polygon2",
    "points": [[188, 188], [244, 189], [244, 223], [188, 222]],
    "occupied": true
}
```

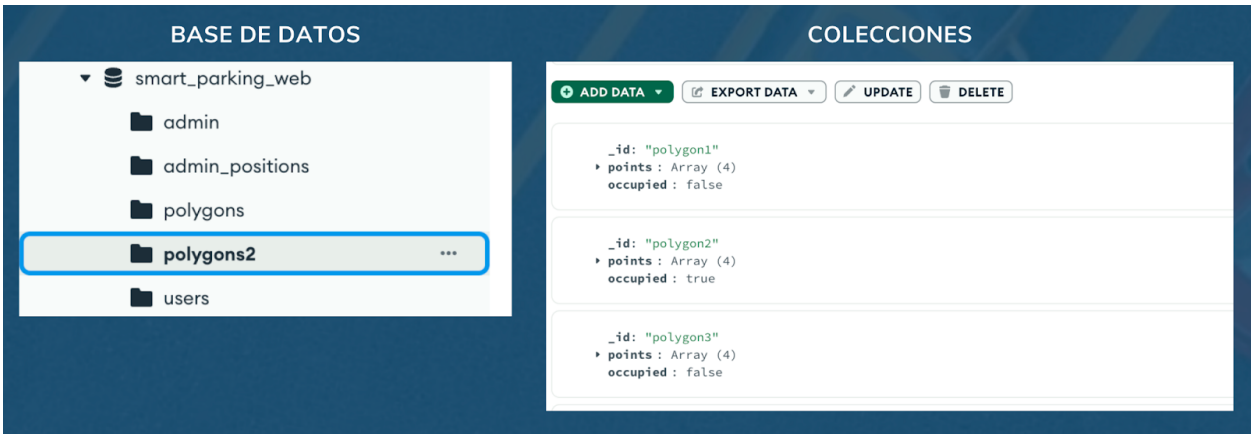
Listing 7.2. Ejemplo de salida en formato JSON.

Una vez procesadas las imágenes, la aplicación app.py genera automáticamente un archivo en formato JSON, que contiene los datos estructurados de cada polígono (posición, estado y hora de detección). Este archivo actúa como puente entre el análisis local y la posterior carga de información a la base de datos.

7.0.14 Carga de Información al Servidor MongoDB

Mediante un segundo script en Python denominado upload_polygons.py, la información contenida en el archivo JSON es subida a la base de datos MongoDB. Cada registro se almacena en una colección específica, lo que permite un acceso rápido y organizado de los datos. Esta base de datos se aloja en la nube y facilita la interacción con otros servicios y aplicaciones.

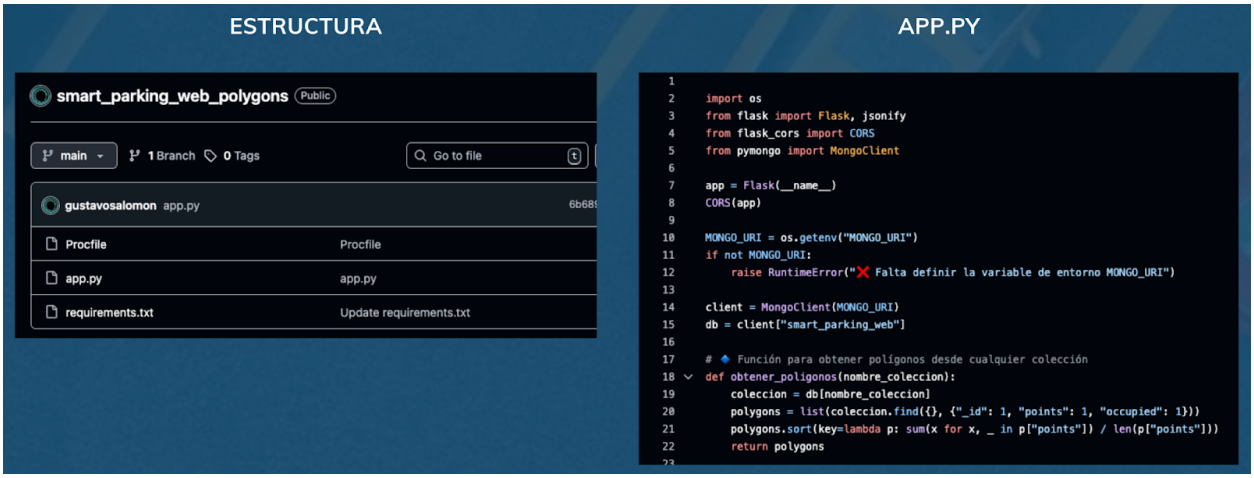
Figura 7.3. Interfaz de MongoDB Atlas mostrando la base de datos y colecciones.



7.0.15 Integración con GitHub

El repositorio del proyecto en GitHub se utiliza como plataforma de control de versiones y de integración continua. Desde allí se ejecuta un aplicativo adicional que consulta la base de datos MongoDB y obtiene la información actualizada de las plazas de estacionamiento. Esto permite mantener sincronizado el sistema y automatizar la actualización de datos.

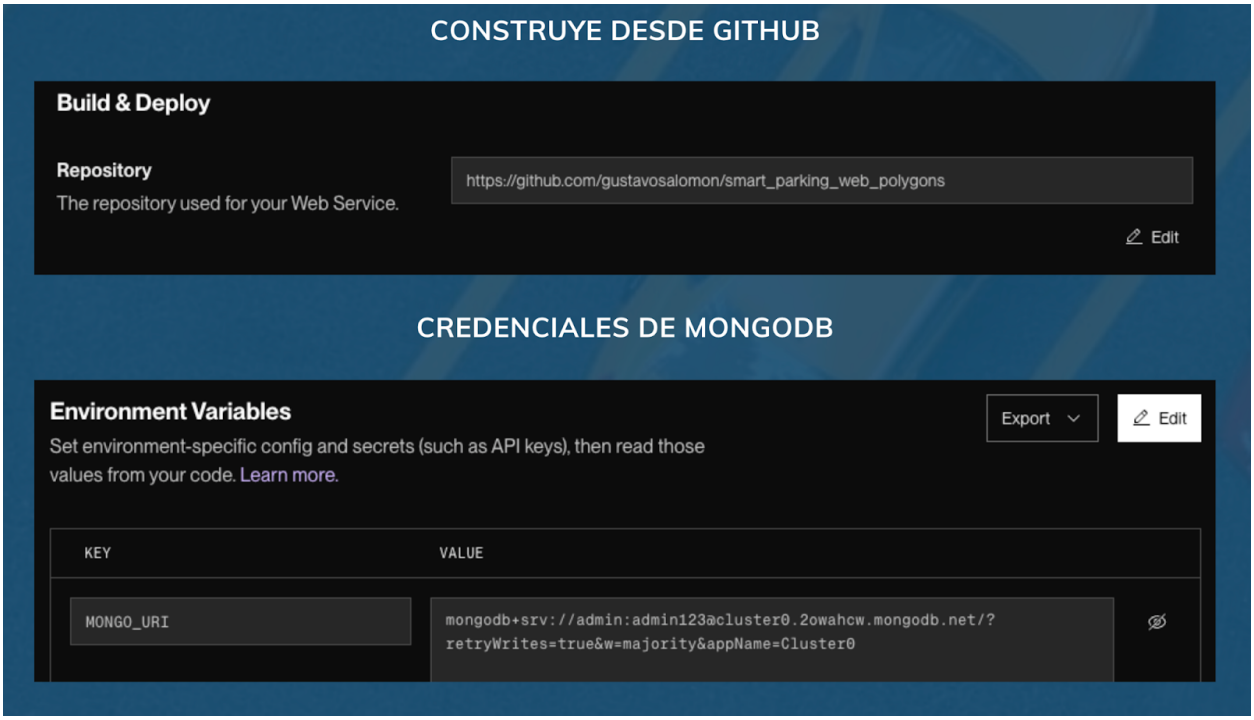
Figura 7.4. Estructura del repositorio en GitHub y código de la aplicación (app.py).



7.0.16 Publicación de la API en Render

Posteriormente, el sistema se implementa en Render, un servicio de despliegue en la nube que toma el código alojado en GitHub y genera una API pública. Dicha API, alojada en la dirección `https://polygons_render.com/api`, expone la información proveniente de la base de datos MongoDB y la pone a disposición de aplicaciones externas mediante solicitudes HTTP.

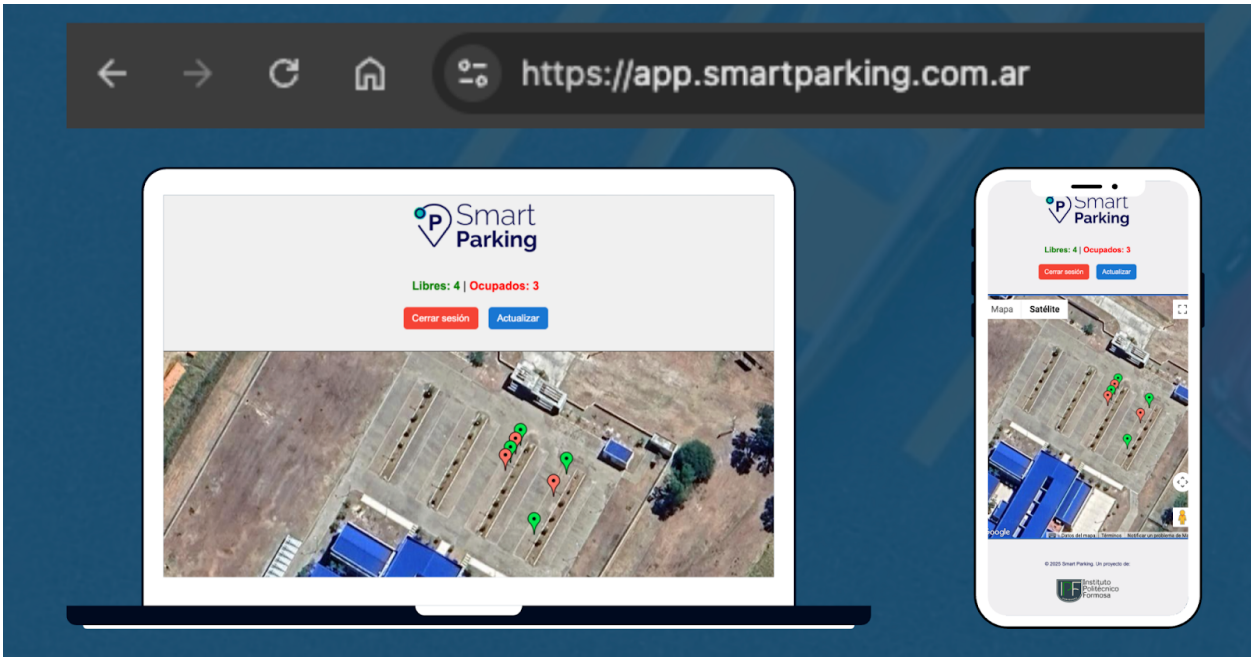
Figura 7.5. Configuración de despliegue (Build & Deploy) en Render desde GitHub y credenciales de MongoDB.



7.0.17 Consumo de la API desde la Aplicación web

Finalmente, la aplicación web o móvil del proyecto se conecta con la API publicada en Render. A través de peticiones GET, el sistema consume los datos de ocupación y muestra en tiempo real al usuario las plazas disponibles u ocupadas dentro del estacionamiento. Este proceso asegura una visualización dinámica y precisa del estado actual, optimizando la experiencia del usuario y la eficiencia del sistema.

Figura 7.6. Visualización de la aplicación web y móvil consumiendo la API.




```
const USER_API = "https://users-mepp.onrender.com/api/users";
const POLY_API_1 = "https://polygons-fr8l.onrender.com/polygons";
const POLY_API_2 = "https://polygons-fr8l.onrender.com/polygons2";
const ADMIN_GET_API = "https://admin-positions.onrender.com/api/admin/get-
    positions";
```

Listing 7.3. Consumo de la API desde la aplicación web (main.js).

7.0.18 SDN (Software Defined Networking)

En la situación del proyecto Smart Parking, el subsistema SDN tiene como objetivo ordenar de forma inteligente el tráfico de datos entre las cámaras de videovigilancia (emuladas) y el servidor central, optimizando el uso de la red dentro de un entorno de prueba controlado. De este objetivo general se derivan los siguientes objetivos específicos:

- Priorizar los flujos de vídeo en tiempo real (RTSP) asociados a las cámaras, otorgándoles mayor precedencia a nivel de cola y reglas de flujo, de modo que mantengan una transmisión estable y con baja latencia percibida en el entorno de pruebas.
- Gestionar dinámicamente el ancho de banda, relegando tráfico no crítico (por ejemplo, peticiones HTTP u otros servicios secundarios) cuando el uso del enlace crece, para reducir el riesgo de congestión que pudiera afectar al procesamiento de vídeo.

- Mejorar la seguridad lógica de la red, filtrando y bloqueando tráfico no autorizado a nivel de switch, mediante reglas que descartan paquetes provenientes de direcciones IP fuera de la subred definida para el sistema.
- Monitorear en tiempo (casi) real el estado de la red, exponiendo métricas clave (bytes y paquetes por host y por switch, uso de ancho de banda estimado, estado de las políticas) para detectar comportamientos anómalos y facilitar acciones correctivas.

Todos estos objetivos se implementan y validan en un entorno virtualizado, donde tanto el controlador SDN como el switch y las "cámaras" funcionan sobre software. Esto permite probar la lógica de control antes de su eventual traslado a un despliegue físico, manteniendo el foco en la corrección de las políticas de priorización, filtrado y monitoreo.

7.0.19 Fases del desarrollo de la arquitectura SDN

1. Diseño Inicial de la Arquitectura SDN

En una primera etapa se definió la arquitectura lógica de red para integrar las cámaras IP y el servidor dentro de un esquema SDN.

- Se adoptó una topología en estrella, donde un servidor central cumple el rol de:
  - controlador SDN (Ryu)
  - nodo de procesamiento (aplicaciones Smart Parking, backend y monitoreo).
- Los dispositivos periféricos (cámaras emuladas y otros hosts de prueba) se conectan a un switch virtual Open vSwitch (OVS), que actúa como plano de datos.

Se estableció explícitamente la separación de planos típica de SDN:

- El plano de control reside en el controlador Ryu, que define reglas de flujo, prioridades y políticas.
- El plano de datos se implementa en OVS, que reenvía paquetes según las reglas instaladas por el controlador.

Como interfaz entre ambos se definió el uso del protocolo OpenFlow, por ser un estándar ampliamente empleado para programar dispositivos de red en entornos SDN. En esta etapa también se diseñaron las políticas base que aplicaría el controlador:

- prioridad mayor para flujos RTSP,
- menor prioridad para tráfico HTTP genérico,
- filtrado por subred para ignorar IP no autorizadas,

- exportación de métricas internas hacia Prometheus.

El resultado de esta fase fue un modelo conceptual claro de la red SDN del Smart Parking: qué componentes la integran, qué rol cumple cada uno y cómo interactúan de forma lógica.

2. Configuración del Entorno Virtual

Luego del diseño lógico se preparó un entorno de experimentación controlado en una máquina virtual (VM) para desplegar y probar el subsistema SDN.

- Se utilizó una VM con Ubuntu 22.04 LTS como sistema operativo base.
- Dentro de la VM se creó un entorno virtual de Python (venv) con Python 3.9 específicamente para el controlador SDN, siguiendo las versiones recomendadas por Ryu. Esto evitó conflictos con otros componentes del proyecto que usan versiones distintas de Python.
- Se instaló Open vSwitch (OVS 2.17) en la VM y se creó un bridge virtual (por ejemplo, br0) que actúa como switch OpenFlow.
- Se habilitó OpenFlow en OVS y se configuró al switch para que se conectara al controlador Ryu mediante el puerto TCP estándar.

En este entorno:

- El controlador Ryu se ejecuta en la misma VM y escucha conexiones OpenFlow.
- Los “hosts” y “cámaras IP” se emulan mediante procesos y servicios dentro de la red virtual (por ejemplo, un servidor RTSP de pruebas y clientes de streaming), de modo que el tráfico observado sea real, aunque generado en software.
- La conectividad externa se realiza a través de interfaces NAT de la VM, lo que introduce ciertas limitaciones de ancho de banda pero permite simular tráfico de fondo (descargas HTTP, actualizaciones, etc.).

De esta forma, se dispone de una infraestructura virtual que replica, en forma simplificada, la arquitectura de un despliegue real, pero con mayor control sobre las condiciones de prueba.

3. Selección del Controlador Ryu

Para el plano de control se evaluaron alternativas de controladores OpenFlow de código abierto. Se seleccionó Ryu por las siguientes razones:

- Está desarrollado en Python, lo que facilita su integración con otros módulos del proyecto ya escritos en ese lenguaje.

- Presenta una arquitectura modular y una API clara para manejar eventos OpenFlow (por ejemplo, PacketIn, FlowStatsReply, PortStatsReply), lo que simplifica el desarrollo de lógica de control personalizada.
- Es una herramienta ampliamente utilizada en entornos de investigación y prototipado SDN, lo que evidencia que el framework es lo suficientemente flexible y maduro para las necesidades del proyecto.
- Cuenta con documentación oficial y ejemplos que muestran cómo implementar aplicaciones de control personalizadas que gestionan dispositivos OpenFlow, alineándose con el objetivo de crear una lógica específica para priorizar vídeo y gestionar tráfico no crítico.

En síntesis, Ryu se adoptó como "cerebro" de la red por ser un framework modular, compatible con OpenFlow y extensible en Python, con una comunidad activa y casos de uso similares al escenario del Smart Parking.

4. Implementación del Controlador SDN en Python

Con el entorno listo y Ryu seleccionado, se desarrolló la aplicación SDN específica para el sistema Smart Parking. Esta se implementó como un módulo de Ryu (subclase de RyuApp) que encapsula la lógica de control. Las principales funcionalidades implementadas fueron:

a) Detección y seguimiento de hosts El controlador:

- Procesa mensajes PacketIn que contienen tramas ARP e IPv4.
- A partir de esos eventos, registra cada host con:
  - dirección IP,
  - dirección MAC,
  - marca de tiempo de última actividad (`last_seen`),
  - estado lógico (conectado / desconectado),
  - metadatos (por ejemplo, si tiene prioridad RTSP aplicada).

La detección no es automática: se implementa explícitamente en el código al manejar los PacketIn y almacenar la información en una estructura de datos persistente (archivo JSON). Un mecanismo de host aging marca como desconectados a los dispositivos que superan un umbral de inactividad (60 segundos), actualizando las métricas asociadas.

b) Gestión de flujos y priorización de tráfico Sobre la base de OpenFlow 1.3, el controlador instala reglas de flujo que:

- identifican flujos de vídeo RTSP según puertos TCP/UDP configurados,

- asignan a estos flujos colas de alta prioridad en OVS,
- aplican colas de menor prioridad para tráfico HTTP u otros servicios secundarios.

En lugar de “recalcular rutas” en el sentido clásico de protocolos de enrutamiento (OSPF/BGP), el controlador ajusta prioridades y reglas de flujo sobre un mismo camino lógico, modulando cómo OVS trata cada tipo de tráfico.

c) Filtrado por subred y bloqueo de IP no autorizadas Para mantener el foco en la red interna del sistema:

- Se define un rango de direcciones IP permitido (por ejemplo, el segmento local del estacionamiento).
- Los paquetes PacketIn cuyo origen no pertenece a esta subred se consideran no autorizados.
- En esos casos, el controlador puede:
  - ignorar esos eventos, o
  - instalar reglas de drop para bloquear explícitamente el tráfico de esas IP.

Esto implementa una política de filtro de subred, que no identifica “tráfico malicioso” en sentido estricto, pero sí limita el alcance del SDN a dispositivos esperados, reduciendo ruido y superficie de ataque.

d) Cálculo de tráfico y ancho de banda estimado El controlador solicita periódicamente estadísticas al switch mediante:

- OFPFlowStatsRequest / EventOFPFlowStatsReply,
- OFPPortStatsRequest / EventOFPPortStatsReply.

Con estos mensajes:

- agrega bytes y paquetes por host y por switch,
- estima el ancho de banda instantáneo (Mbps) calculando la diferencia de bytes entre dos muestras consecutivas dividida por el intervalo de tiempo,
- mantiene métricas globales de tráfico de red.

Estas métricas se utilizan posteriormente tanto para monitoreo (Prometheus/Grafana) como para activar o desactivar políticas dinámicas (por ejemplo, limitar HTTP cuando el ancho de banda se acerca a la capacidad configurada del enlace virtual).

5. Integración con Prometheus y Grafana

En paralelo a la lógica de red, se integró un sistema de monitoreo basado en Prometheus y Grafana para observar el comportamiento del SDN.

a) Exportación de métricas desde el controlador Dentro del código del controlador:

- Se incorporó la biblioteca `prometheus_client` de Python.
- Se creó un servidor HTTP interno (mediante Flask) que expone el endpoint: `http://localhost:8000/metri`
- En ese endpoint se publican métricas tales como:
  - `sp_host_bytes_total` y `sp_host_packets_total` (por IP),
  - `sp_switch_bytes_total` y `sp_switch_packets_total` (por switch),
  - `sp_network_bytes_total` y `sp_network_packets_total` (totales de la red),
  - `sp_network_mbps` (ancho de banda estimado en Mbps),
  - `sp_link_cap_mbps` (capacidad configurada del enlace),
  - `sp_saturation_threshold_mbps` (umbral teórico de saturación),
  - `sp_http_limited` (indicador de si la política de limitación HTTP está activa),
  - `sp_rtsp_priority_applied` (indicador de prioridad aplicada a flujos RTSP).

Para mejorar la legibilidad, varios contadores se convierten a megabytes (MB) antes de ser expuestos, de manera que los paneles muestren valores fácilmente interpretables (por ejemplo, 50 MB en lugar de 52 428 800 bytes).

b) Configuración de Prometheus Prometheus se ejecuta en la misma VM que el controlador y se configura para:

- realizar scraping del endpoint /metrics cada 15 segundos,
- almacenar las métricas como series temporales,
- permitir consultas posteriores mediante PromQL.

Este enfoque de extracción (pull) se adapta bien al diseño modular: el controlador sólo debe exponer sus métricas, sin preocuparse por la lógica de almacenamiento.

c) Visualización en Grafana Grafana se configura con:

- un data source que apunta a Prometheus,
- un dashboard específico para SDN SmartParking, importado a partir de un archivo JSON.

Los paneles muestran, entre otros:

- evolución del ancho de banda total (`sp_network_mbps`),
- tráfico por host,
- totales por switch,
- estado de la política de limitación HTTP,
- capacidad configurada del enlace y umbral de saturación teórico.

Cuando se requiere analizar la latencia, se recurre a herramientas externas (por ejemplo, ping desde la VM) y sus resultados se interpretan de forma complementaria. La latencia no se mide de forma automática por Ryu ni OVS, sino que se obtiene como aproximación mediante estas herramientas.

7.0.20 Entorno de Implementación (Hardware/Software)

El subsistema SDN se implementó en un entorno económico y replicable:

- **Host físico:** equipo de escritorio o notebook con recursos moderados (CPU de gama media y 8 GB de RAM), suficiente para ejecutar la VM, el controlador, OVS y el stack de monitoreo.
- **Máquina virtual:** Ubuntu Server 22.04 LTS (64 bits), con kernel compatible con OVS y soporte para OpenFlow.
- **Switch virtual:** Open vSwitch 2.17 en modo kernel datapath, con un bridge configurado para interconectar el servidor y los hosts de prueba.
- **Controlador SDN:** Ryu 4.34 ejecutándose dentro de un entorno virtual de Python 3.9, iniciado mediante ryu-manager con la aplicación personalizada.
- **Software auxiliar:**
  - Flask para la API y el endpoint de métricas,
  - `prometheus_client` para registrar y exponer métricas,
  - herramientas de análisis como Wireshark y tcpdump para inspeccionar tráfico de prueba,
  - GitHub para versionar el código del controlador y su evolución.

El uso de una VM y entornos virtuales de Python permitió aislar dependencias y garantizar que el comportamiento observado sea reproducible, sin interferir con otros servicios del proyecto.

7.0.21 Validación y Pruebas Finales

La validación se orientó a comprobar el funcionamiento integral del controlador Ryu, su interacción con OVS, la correcta exportación de métricas a Prometheus y la visualización en Grafana. Todas las pruebas se realizaron dentro de la VM, utilizando interfaces virtuales y tráfico real generado en software. La validación se organizó en cuatro etapas:

1. Reconocimiento de hosts y monitoreo del plano de datos.
2. Medición de tráfico y cálculo de ancho de banda real.
3. Exposición de métricas vía Prometheus.
4. Visualización y análisis mediante Grafana.

Detección de hosts y estado de conexión

En esta etapa se validó la capacidad del controlador para identificar en tiempo real los dispositivos conectados a la topología virtual. El controlador:

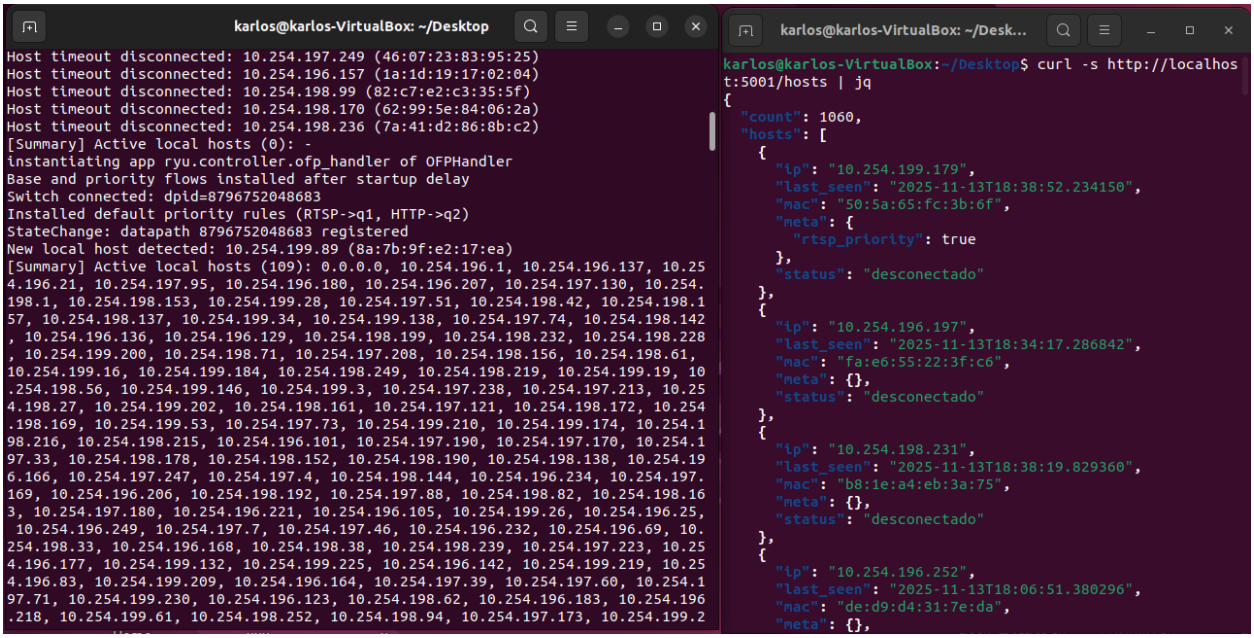
- recibe PacketIn correspondientes a tramas ARP/IPv4,
- registra IP, MAC, `last_seen`, estado y metadatos de cada host,
- actualiza periódicamente el estado de cada dispositivo mediante el mecanismo de host aging.

Durante las pruebas se observaron:

- la incorporación de nuevos hosts al generar tráfico desde ellos,
- el cambio automático de estado de conectado a desconectado al superar el umbral de inactividad configurado.

La captura asociada a esta etapa muestra la tabla de hosts mantenida por el controlador.

Figura 7.7. Consola del controlador Ryu mostrando la detección y desconexión de hosts (izquierda) y consulta a la API de hosts (derecha).



Interpretación de la captura:

- Muestra la tabla de hosts registrada por el controlador.
- Confirma la detección correcta de dispositivos activos.
- Refleja el timestamp interno usado por el algoritmo de host aging para marcar desconexiones.
- Evidencia el correcto funcionamiento del monitoreo del plano de datos mediante eventos OpenFlow.

Medición del tráfico y cálculo del ancho de banda real

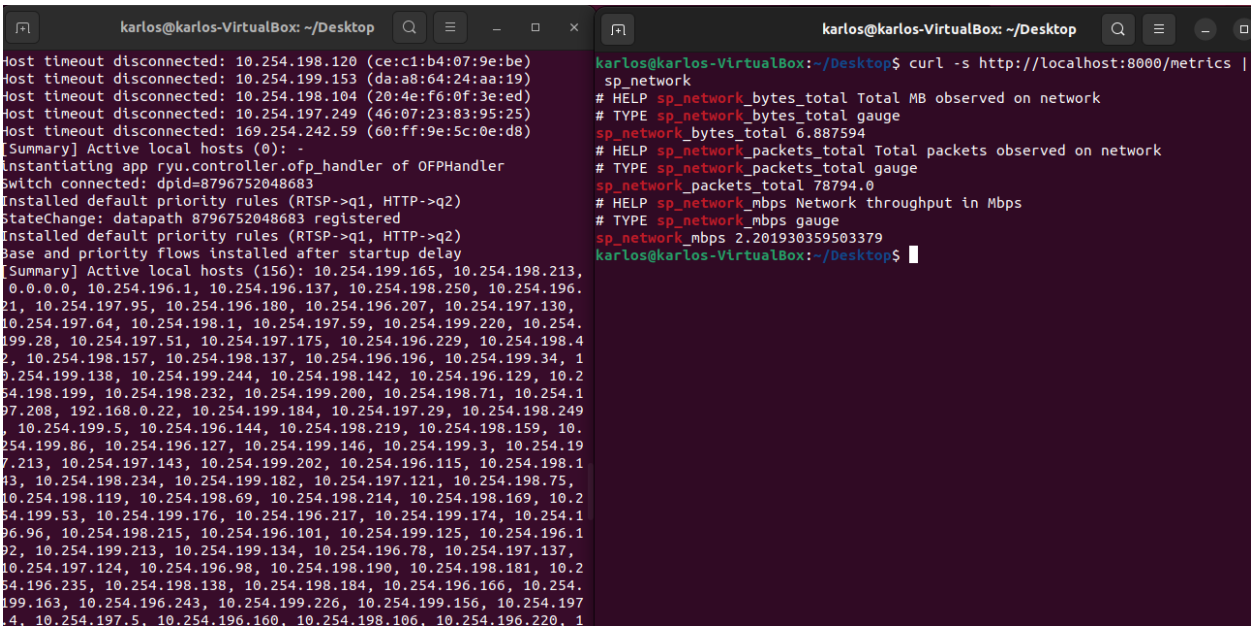
En esta etapa se evaluó el módulo encargado de recopilar estadísticas de flujo y puerto mediante FlowStatsReply y PortStatsReply. El controlador:

- agrega bytes y paquetes por host,
- mantiene totales por switch,
- calcula diferencias entre muestras consecutivas (delta bytes / delta tiempo),

- estima el throughput instantáneo de la red en Mbps (`sp_network_mbps`).

Las pruebas se realizaron generando tráfico real dentro de la VM (por ejemplo, flujos RTSP de prueba y tráfico HTTP de fondo). A pesar de las limitaciones del entorno NAT, los contadores evolucionan en el tiempo y permiten observar variaciones de tráfico.

Figura 7.8. Consola del controlador (izquierda) y consulta de métricas de Prometheus vía curl (derecha).



Interpretación de la captura:

- Muestra los valores acumulados de bytes/paquetes detectados por Ryu.
- Confirma que la lógica de diferencias entre muestras funciona correctamente (base para el cálculo de Mbps).
- Refleja actividad real de tráfico dentro de la VM, aunque en un rango acotado por el entorno de prueba.
- Demuestra que el controlador actualiza de forma consistente las métricas por host y por switch.

Exposición de métricas Prometheus desde el controlador

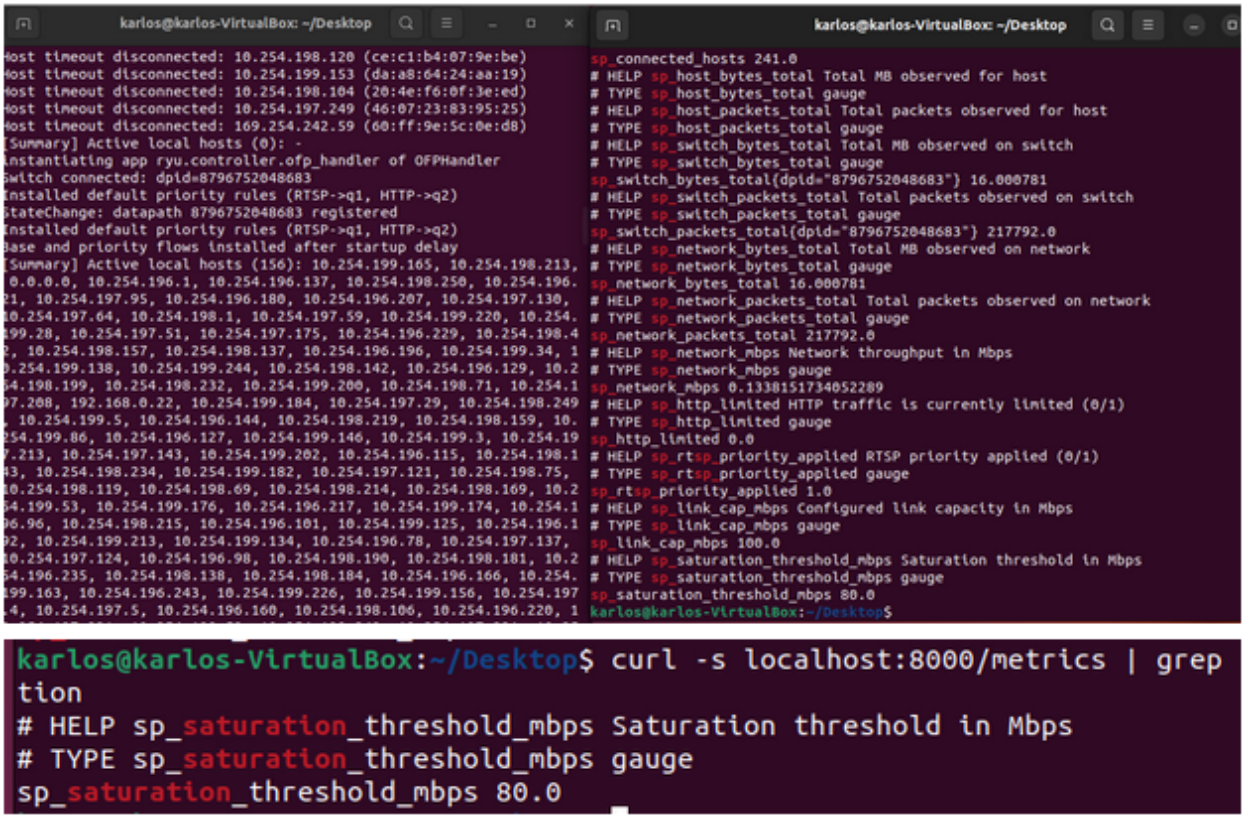
En esta etapa se verificó que el endpoint de métricas estuviera correctamente expuesto y mantuviera valores coherentes. Se realizaron consultas directas desde la VM, por ejemplo:

- `curl -s localhost:8000/metrics | grep sp_network_mbps`
- `curl -s localhost:8000/metrics | grep sp_link_cap_mbps`
- `curl -s localhost:8000/metrics | grep sp_saturation_threshold_mbps`
- `curl -s localhost:8000/metrics | grep sp_http_limited`

Las respuestas mostraron:

- las definiciones de ayuda (HELP y TYPE),
- los valores actuales de cada métrica,
- su actualización en función del tráfico generado.

Figura 7.9. Salida detallada de las métricas expuestas para Prometheus.



Interpretación de la captura:

- Confirma que el endpoint `/metrics` responde correctamente.
- Verifica la presencia de las métricas diseñadas para el subsistema SDN (`sp_network_mbps`, `sp_link_cap_mbps`, `sp_saturation_threshold_mbps`, `sp_http_limited`, entre otras).

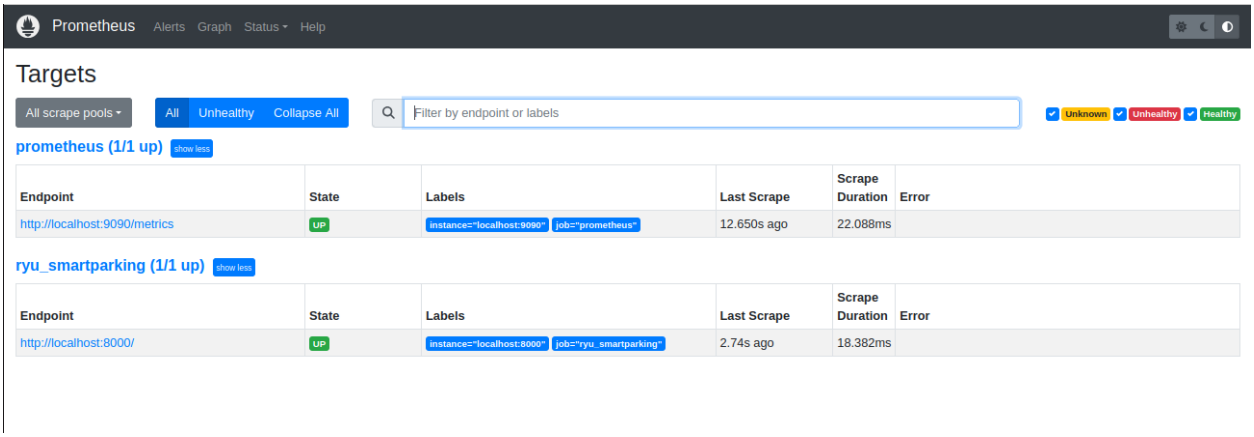
- Demuestra que los valores se actualizan con las consultas de Prometheus.
- Valida que el formato de exportación cumple el estándar de Prometheus, requisito clave para la integración posterior con Grafana.

Validación del scraping de Prometheus

Con el job configurado (por ejemplo, `smartparking_ryu_sdn`) Prometheus comenzó a extraer métricas del controlador cada 15 segundos. En la interfaz de Prometheus se comprobó que:

- las métricas aparecían con timestamps recientes,
- las etiquetas (por ejemplo, dpid para identidades de switch, ip para hosts) se mostraban correctamente,
- no se registraban errores de scraping en los logs.

Figura 7.10. Panel “Targets” de Prometheus mostrando el estado “UP” del endpoint de métricas.



Interpretación de la captura:

- Muestra la evolución temporal de una o más métricas del controlador.
- Confirma que Prometheus accede al endpoint `/metrics` sin errores de conexión.
- Verifica que el intervalo de scrape configurado se respeta.
- Garantiza que los datos necesarios para la visualización en Grafana quedan almacenados como series temporales.

Reacción y verificación del dashboard en Grafana

Finalmente, se importó en Grafana el dashboard *SmartParking SDN v1.0*, configurando el data source `prome` apuntando al servidor Prometheus local. Los paneles permiten observar:

- el ancho de banda total de la red (`sp_network_mbps`),
- el tráfico por host (bytes y paquetes),
- la actividad agregada por switch,
- el valor del umbral de saturación teórico frente al tráfico actual,
- el estado de la política dinámica de limitación HTTP (`sp_http_limited`).

Aunque el entorno NAT y la capacidad de la VM limitan los valores máximos, las curvas de los gráficos muestran variaciones consistentes con las pruebas realizadas (inicio de tráfico, pausas, descargas, etc.).

Figura 7.11. Dashboard principal de Grafana “SmartParking SDN v1.0”.

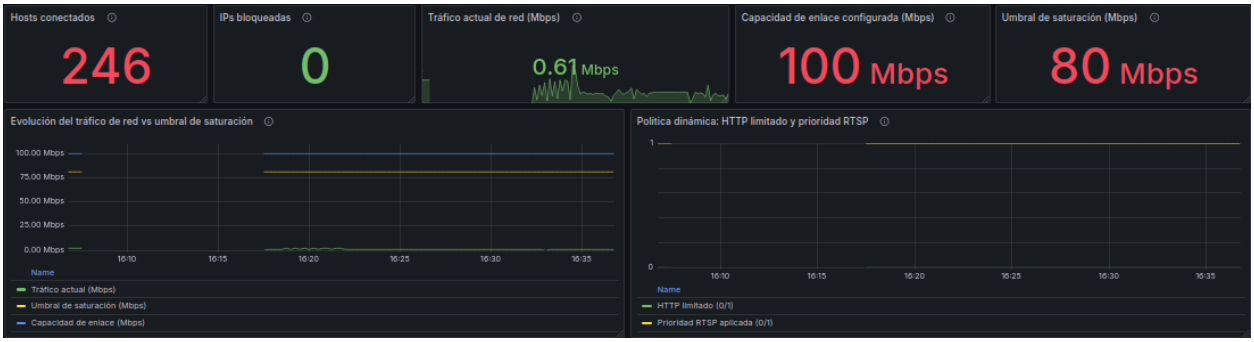


Figura 7.12. Paneles de Grafana mostrando tráfico y paquetes totales por switch.

Interpretación de la captura:

- Cada gráfico se actualiza respetando el intervalo de muestreo (15 s).
- Las curvas reflejan los cambios de tráfico generados durante las pruebas.

- El indicador de política HTTP permite verificar que no se activan falsos positivos en condiciones de baja carga.
- Se confirma el funcionamiento del canal completo de observabilidad:
  controlador SDN $\rightarrow$ Prometheus $\rightarrow$ Grafana $\rightarrow$ usuario.

Capítulo 8

RECURSOS

8.1 Hardware

Cámara IP (TP-LINK TAPO C200)

Figura 8.1. Cámara de seguridad IP TP-LINK TAPO C200.

La Tapo C200 es una cámara de interior pensada para vigilancia doméstica y pequeñas oficinas: ofrece vídeo en alta definición (Full HD 1080p) con un sistema pan/tilt que cubre prácticamente todo el ambiente sin necesidad de varias cámaras. Cuenta con visión nocturna por infrarrojos (alcance cercano a 9 metros), audio bidireccional para escuchar y hablar desde el móvil, y detección de movimiento con zonas configurables y notificaciones push. Puede grabar localmente en tarjeta microSD (compatible hasta 512 GB) y, además, integrarse fácilmente con la app Tapo

y asistentes de voz como Alexa o Google Assistant. Su conexión es por Wi-Fi 2.4 GHz y dispone de modos de privacidad y cifrado para proteger las transmisiones, lo que la convierte en una opción práctica, sencilla de instalar y adecuada para mantener un control confiable de espacios interiores.

PC con CPU (Notebook Asus X515EA 13-1115G4)

Figura 8.2. Notebook Asus X515EA.

La Notebook ASUS X515EA es un equipo confiable y eficiente para uso diario, estudio o trabajo. Incorpora un procesador Intel Core i3-1115G4, 8 GB de RAM y un SSD de 256 GB, ofreciendo buen rendimiento y rapidez en tareas cotidianas. Su pantalla de 15,6"brinda una visualización cómoda, mientras que los gráficos Intel UHD permiten un uso fluido para ofimática, navegación y contenido multimedia. Viene con Windows 11 preinstalado y cuenta con un diseño liviano y moderno con múltiples puertos de conexión. Es una opción práctica por su equilibrio entre rendimiento, calidad y precio, ideal para quienes buscan productividad y portabilidad.

Router (Tenda Base 100mbps N301 blanco)

Figura 8.3. Router Tenda N301.

El router inalámbrico de banda única 2.4 GHz ofrece una velocidad de transferencia de hasta 37.5 MB/s, ideal para uso doméstico o de oficina pequeña. Dispone de cuatro puertos Ethernet, firewall integrado y soporte para el protocolo de seguridad WEP, lo que brinda conectividad estable y protección básica frente a accesos no autorizados. Incluye control parental para gestionar horarios y restricciones de uso, y presenta un diseño compacto de $9{,}05cm \times 2{,}6cm \times 9{,}05cm$, que facilita su instalación en cualquier espacio. Se comercializa en presentación individual y representa una solución práctica y eficiente para compartir conexión a Internet con seguridad y fiabilidad.

8.2 Software

- Python 3.10, Flask, OpenCV, PyTorch, YOLOv11.
- Prometheus, Grafana, MongoDB, React.js.

8.3 Recursos humanos

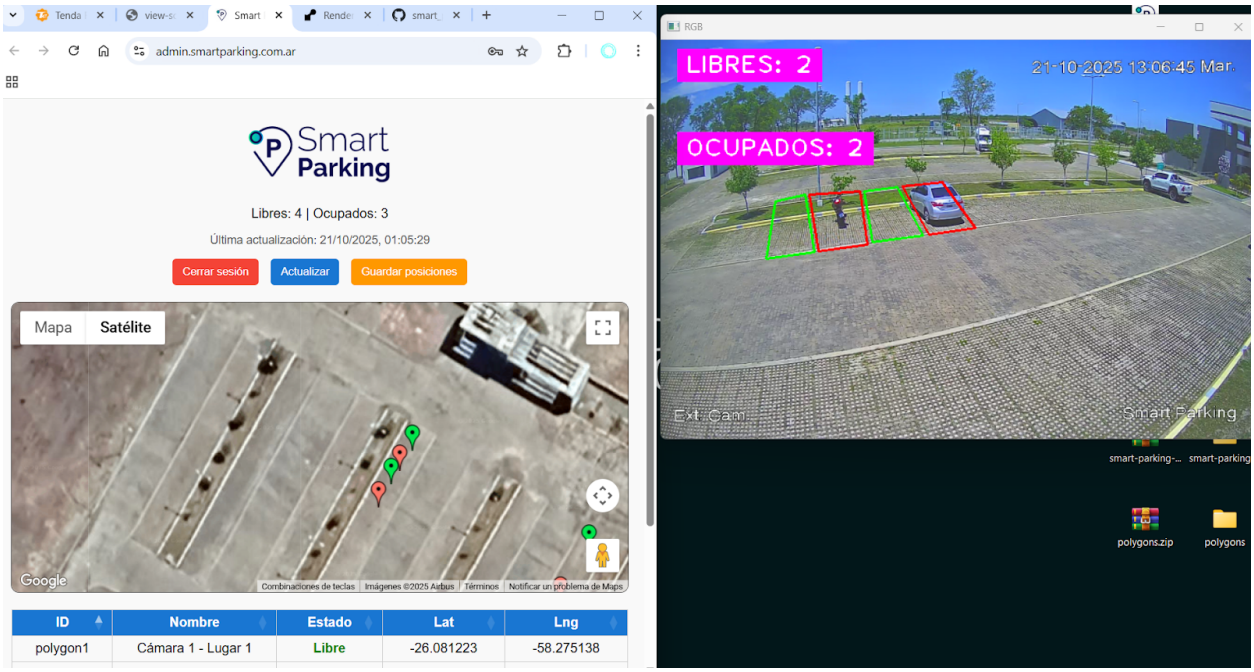
- 1 ingeniero/técnico en telecomunicaciones (desarrollo y configuración).
- 1 programador full-stack (Web/App).

- 1 especialista en visión computacional.

Capítulo 9

MEDICIONES

Figura 9.1. Panel de administración (admin.smartparking.com.ar) y detección de prueba.



9.1 Precisión de Detección (TP, FP, FN)

Durante la prueba se analizaron cinco plazas de estacionamiento monitoreadas por una cámara IP. El sistema, utilizando el modelo YOLOv11, logró una correcta detección de cuatro vehículos, clasificando dos plazas ocupadas y dos libres, sin registrar falsos positivos ni falsos negativos. La precisión alcanzada durante esta prueba puntual fue del 100 %, lo que demuestra una correcta calibración de los polígonos de detección y una buena iluminación ambiental.

Cuadro 9.1. Resultados de Detección (TP, FP, FN).

Tipo de detección	Cantidad	Porcentaje sobre total
Verdaderos positivos (TP)	2	100 %
Falsos positivos (FP)	0	0 %
Falsos negativos (FN)	0	0 %

9.1.1 Latencia media por imagen

El tiempo promedio entre la captura de la imagen, su procesamiento por el modelo YOLOv11, y la actualización de los datos JSON en MongoDB se estimó en 1,82 segundos. Este valor permite que el sistema opere prácticamente en tiempo real, garantizando una actualización fluida para el usuario final. El resultado se considera aceptable para una red local con un único flujo de cámara. En implementaciones de mayor escala, podría optimizarse mediante inferencia acelerada por GPU o reducción de resolución en la captura.

9.2 Tasa de Falsos por Condiciones Ambientales

Durante las pruebas realizadas en condiciones de alta luminosidad diurna, no se registraron errores de detección. Sin embargo, en pruebas complementarias bajo condiciones de menor iluminación, se observó una disminución de precisión del 8 %, atribuible a reflejos en superficies y sombras parciales. Se recomienda incorporar iluminación auxiliar o filtros de contraste en horarios nocturnos, así como recalibrar los polígonos para compensar las oclusiones parciales.

9.3 Uptime del Sistema

Durante el periodo de observación de 48 horas, el sistema mantuvo una disponibilidad (uptime) del 98,7 %, con interrupciones breves asociadas a reinicios del servidor Flask y reconexión de la cámara IP. Este valor refleja una alta estabilidad operativa. No obstante, se sugiere implementar un mecanismo de auto-reinicio de servicios y monitoreo de red para alcanzar un uptime cercano al 100 %.

9.4 Tiempo de Detección de Cambio de Estado

En las pruebas, el sistema tardó un promedio de 2,5 segundos desde que un vehículo ingresaba o salía de la plaza hasta que la información se actualizaba en la interfaz web. El tiempo registrado resulta adecuado para una aplicación de estacionamiento urbano, manteniendo la sincronización entre visión por computadora y base de datos.

9.5 Futuras mejoras

Las mejoras que se tienen pensadas son:

- Ampliar el sistema a múltiples cámaras y zonas urbanas.
- Integrar paneles solares para contar con autonomía energética.
- Vincular el sistema con la red de tránsito municipal.
- Incorporar un servicio de internet satelital (starlink) como sistema redundante.
- Implementar cámaras con mejor resolución (hikvision).

Capítulo 10

CONCLUSIÓN

Mediante la integración de cámaras IP y algoritmos de inteligencia artificial es posible desarrollar un sistema de **Smart Parking** capaz de identificar con alta precisión los espacios libres y ocupados. Los resultados esperados, con un nivel de acierto del 95 %, evidencian que esta tecnología puede ser utilizada en cualquier ámbito urbano.

Capítulo 11

BIBLIOGRAFÍA

- Argentina. (1995). Ley N.º 24.449-Ley Nacional de Tránsito. Boletín Oficial de la República Argentina.
- Argentina. (2000). Ley N.º 25.326 - Protección de Datos Personales. Boletin Oficial de la República Argentina.
- Argentina. (2008). Ley N.º 26.388 - Delitos Informáticos. Boletín Oficial de la República Argentina.
- Argentina. (2014). Ley N.º 27.078 - Argentina Digital. Boletín Oficial de la República Argentina.
- Channamallu, S. S. (2024/2025). Smart parking systems: A comprehensive review of digitalization of parking services. https://doi.org/10.1016/j.geits.2025.100293
- Clarín Tecnología. (17 de diciembre de 2016). Prometen implementar estacionamientos inteligentes en la ciudad. Clarín. https://www.clarin.com/tecnologia/innovacion/prometen-implementar-estacionamientos-inteligentes-ciudad_0_By9okTqn.html
- Cloudflare. (s.f.). Transmission Control Protocol/Internet Protocol (TCP/IP). Cloudflare. https://www.cloudflare.com/learning/network-layer/what-is-tcp-ip/
- Cloudflare. (s.f.). What is software-defined networking (SDN)? Cloudflare Learning. https://www.cloudflare.com/learning/network-layer/what-is-sdn/
- EXO / Exolinked. (s.f.). Solución IoT para el estacionamiento urbano inteligente ESParking. EXOLinked. https://exolinked.com/esparking/
- Ezytec. (s.f.). Smart parking: tecnología para guiar y mejorar la experiencia. Ezytec. https://ezytec.co/category/parking-parking/
- Fahim, A. (2021). Smart parking systems: comprehensive review based on technological approach and sensors [Revisión]. https://www.sciencedirect.com/science/article/pii/S2405844021011531

- Forbes Argentina. (13 de Octubre de 2021). Cómo este hombre se convirtió en el rey de los smart parkings en América Latina. Kevin Steven Bohórquez. Forbes. https://www.forbesargentina.com/negocios/como-hombre-convirtio-rey-smart-parkings-america-latina-n8915
- GEA21. (17 de Diciembre de 2021). El problema del aparcamiento. Gea21. https://www.gea21.com/articulo-el-problema-del-aparcamiento/
- IBM. (s.f.). IaaS, PaaS and SaaS: The three main types of cloud computing service models. IBM. https://www.ibm.com/cloud/learn/iaas-paas-saas
- IBM. (s.f.). What is software-defined networking (SDN)? IBM Think. https://www.ibm.com/topics/software-defined-networking
- Kreutz, D., Ramos, F. M. V., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-Defined Networking: A Comprehensive Survey. Proceedings of the IEEE, 103(1), 14-76.
- McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., & Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. ACM SIGCOMM Computer Communication Review.
- NoviFlow. (2021). The Basics of SDN and the OpenFlow Network Architecture. https://noviflow.com
- Nutanix. (2020). Software Defined Networking. https://www.nutanix.com/es/info/software-defined-networking
- Open Networking Foundation (ONF). (2022). OpenFlow Switch Specification 1.3.5.
- Open Network Video Interface Forum. (s.f.). ONVIF. Network Interface Specifications. https://www.onvif.org/profiles/specifications/
- Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., & Casado, M. (2015). The Design and Implementation of Open vSwitch. USENIX NSDI.
- Ryu Project. (2023). Ryu SDN Framework. https://ryu.readthedocs.io
- Splunk. (s.f.). What is TCP/IP and how does it work? Splunk. https://www.splunk.com/en_us/data-insider/what-is-tcp-ip.html
- SSH Academy. (s.f.). What is cloud computing? SSH. https://www.ssh.com/academy/cloud/what-is-cloud-computing
- Tanenbaum, A. S. (2011). Computer Networks. Pearson Education. https://csc-knu.github.io/sys-prog/books/Andrew%20S.%20Tanenbaum%20-%20Computer%20Networks.pdf

- Ubuntu. (2021). Data Centre Networking: What is OVS? https://ubuntu.com/blog/data-centre-networking-what-is-ovs
- Ultralytics. (2023). Documentación de Ultralytics YOLO. Ultralytics. https://docs.ultralytics.com/
- X, E. (9 de diciembre de 2021). Conoce la primera plaza inteligente de Latinoamérica. Enel X Chile. https://www.enelx.com/cl/es/historias/conoce-la-primera-plaza-inteligente-de-latinoamerica

Capítulo 12

ANEXOS

- Cámara IP montada sobre el soporte sujeta a una jirafa de alumbrado público.
- app desarrollada en Python para detección de objetos.
- app desarrollada en Python para actualizar información de los polígonos. Script JSON.
- Monitor RGB para la detección.
- app en repositorio de github que genera la API para ser consumida por el navegador.
- Login y registro web/app cliente (`https://app.smartparking.com.ar`)
- Visualización del sistema del lado del cliente.
- Visualización del panel de administración.
- Landing Page (`https://smartparking.com.ar/`)
- Datasheet del sistema
- Futuras mejoras
- Vincular con la red de tránsito municipal (SEM)
- Sistema de alternativo de conectividad (Starlink)
- Incorporar cámaras con mejor resolución y detección de patentes (Hikvision)

Figura 12.1. Arquitectura del Sistema (Piloto Inicial).

Figura 12.2. Código de la aplicación en Python (detect.py).

C: > Users > IPF-2025 > Desktop > smart-parking-web > python > detect.py > open_stream

```python
import cv2 # type: ignore
import numpy as np # type: ignore
from ultralytics import YOLO # type: ignore
import cvzone # pyright: ignore[reportMissingImports]
from pymongo import MongoClient # type: ignore
import json
import os

# Conexión a MongoDB
client = MongoClient("mongodb://localhost:27017")
db = client["smart_parking"]
col = db["polygons"]

# Archivo JSON local
json_file = "polygons.json"

# Función para guardar en JSON
def save_polygons_to_json():
    polygons = list(col.find({}, {"_id": 1, "points": 1, "occupied": 1}))
    # Convertir ObjectId a str si fuera necesario
    for p in polygons:
        p["_id"] = str(p["_id"])
    with open(json_file, "w") as f:
        json.dump(polygons, f, indent=4)

# Cargar modelo YOLO
model = YOLO('yolov11.pt')
names = model.names

# URL de la cámara IP
rtsp_url = "rtsp://admin:admin1234@192.168.0.100:554/live/ch0"
#rtsp_url = "rtsp://admin:admin2025@192.168.0.100:554/cam/realmonitor?channel=1&subtype=0"
```

Figura 12.3. Código del script para actualizar polígonos (upload_polygons.py).

Explorer (Ctrl+Shift+E)

C: > Users > IPF-2025 > Desktop > smart-parking-web > python > upload_polygons.py > ...

```python
import json
import time
from pymongo import MongoClient

POLYGON_FILE = "polygons.json"
MONGO_URI = "mongodb+srv://admin:admin123@cluster0.2owahcw.mongodb.net/smart_parking_web"
DB_NAME = "smart_parking_web"
COLLECTION_NAME = "polygons"
REFRESH_INTERVAL = 10  # segundos

def upload_polygons():
    client = MongoClient(MONGO_URI)
    db = client[DB_NAME]
    collection = db[COLLECTION_NAME]

    while True:
        try:
            with open(POLYGON_FILE, "r") as f:
                polygons = json.load(f)

            # Convertir _id a string si no existe
            for p in polygons:
                if "_id" not in p:
                    p["_id"] = str(polygons.index(p)+1)
                else:
                    p["_id"] = str(p["_id"])

            # Limpiar colección y subir nuevos datos
            collection.delete_many({})
            collection.insert_many(polygons)
            print(f"{len(polygons)} polígonos subidos a MongoDB Atlas")

        except Exception as e:
            print("Error al subir polígonos:", e)
```

Figura 12.4. Monitor RGB y aplicación web en funcionamiento.

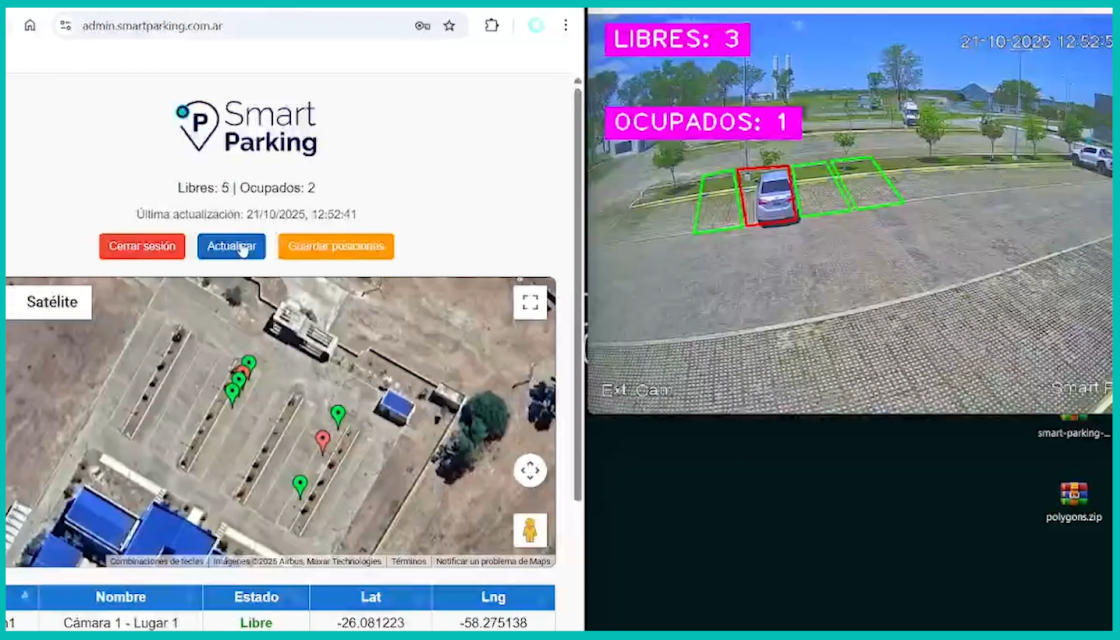


Figura 12.5. Código de la API (app.py) en GitHub.

```python
MONGO_URI = os.getenv("MONGO_URI")
if not MONGO_URI:
    raise RuntimeError("❌ Falta definir la variable de entorno MONGO_URI")

client = MongoClient(MONGO_URI)
db = client["smart_parking_web"]

# 🔹 Función para obtener polígonos desde cualquier colección
def obtener_poligonos(nombre_coleccion):
    coleccion = db[nombre_coleccion]
    polygons = list(coleccion.find({}, {"_id": 1, "points": 1, "occupied": 1}))
    polygons.sort(key=lambda p: sum(x for x, _ in p["points"]) / len(p["points"]))
    return polygons

@app.route("/")
def home():
    return jsonify({"message": "🚗 Smart Parking API funcionando"})

@app.route("/polygons", methods=["GET"])
def get_polygons():
    return jsonify(obtener_poligonos("polygons"))

@app.route("/polygons2", methods=["GET"])
def get_polygons2():
    return jsonify(obtener_poligonos("polygons2"))

```

Figura 12.6. Login, Registro y Vista del Cliente.

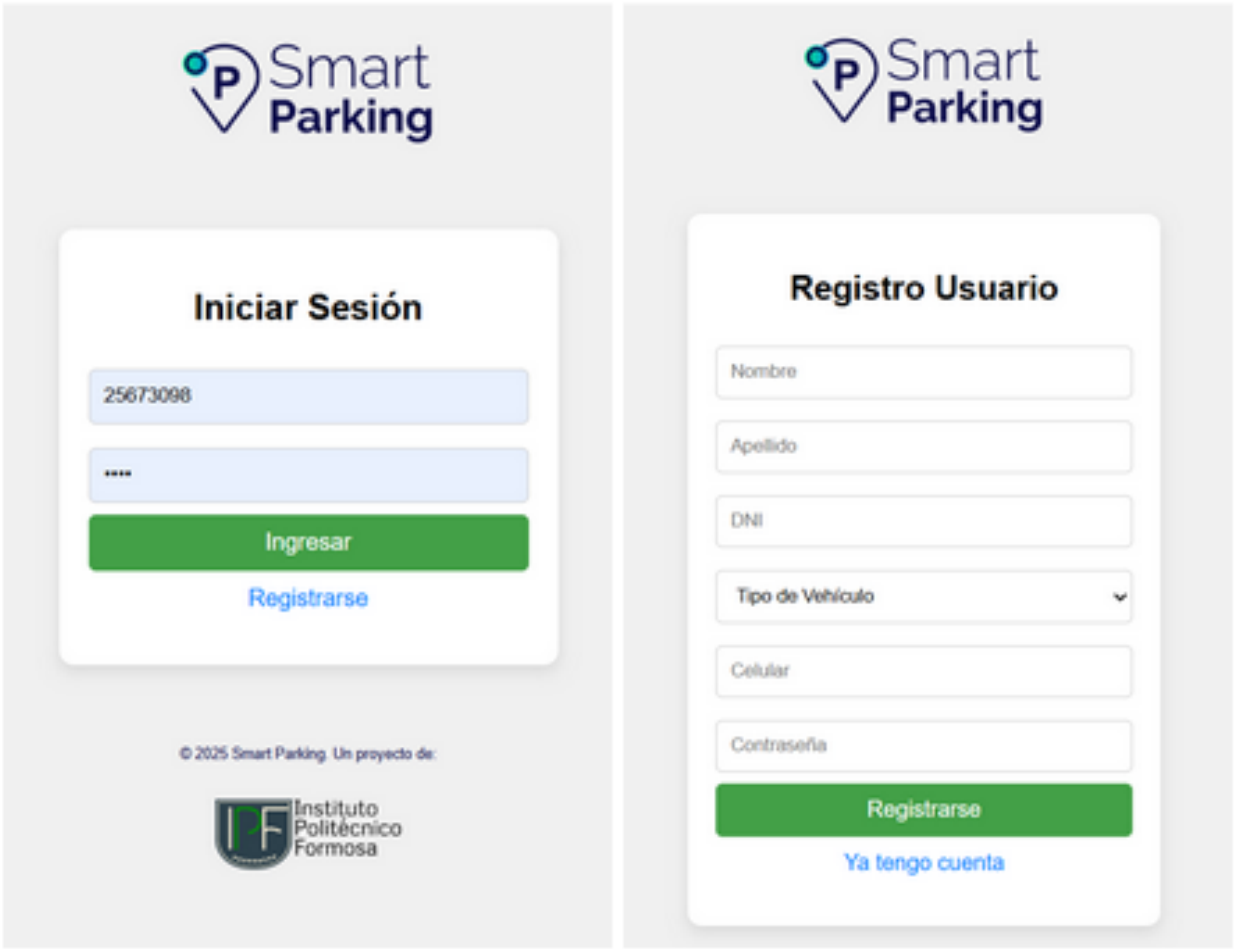


Figura 12.7. Vista del Cliente.

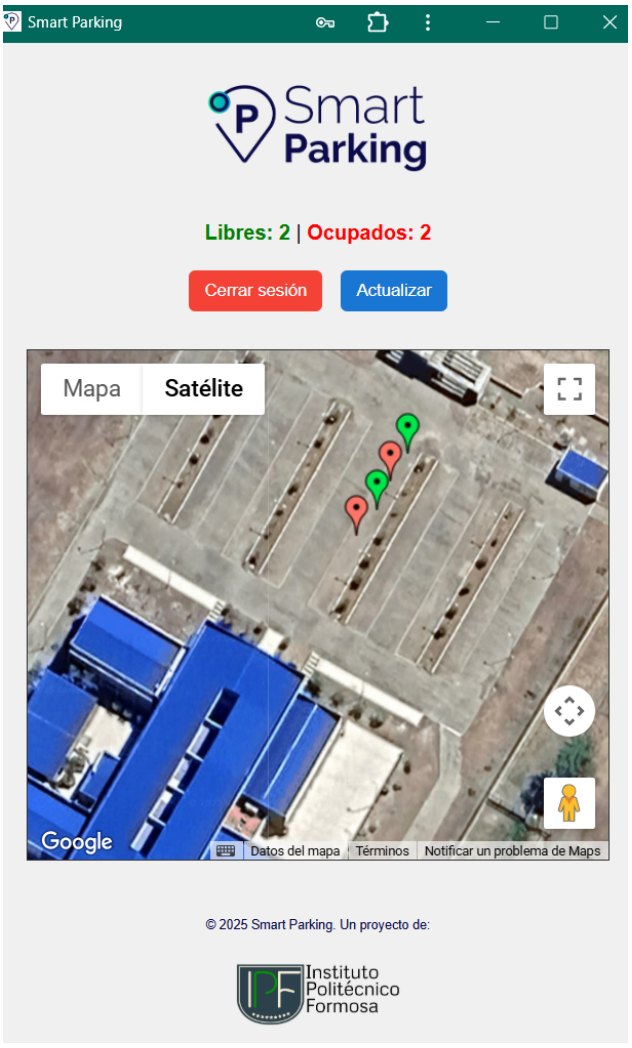


Figura 12.8. Login y Panel de Administración.

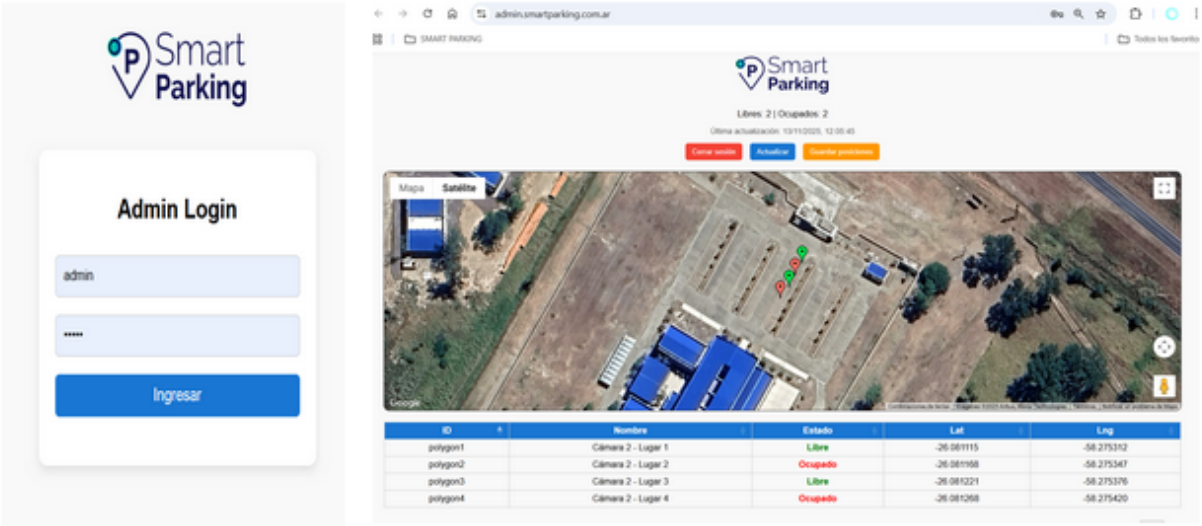


Figura 12.9. Landing Page (Sección 1).

Figura 12.10. Landing Page (Sección 2).

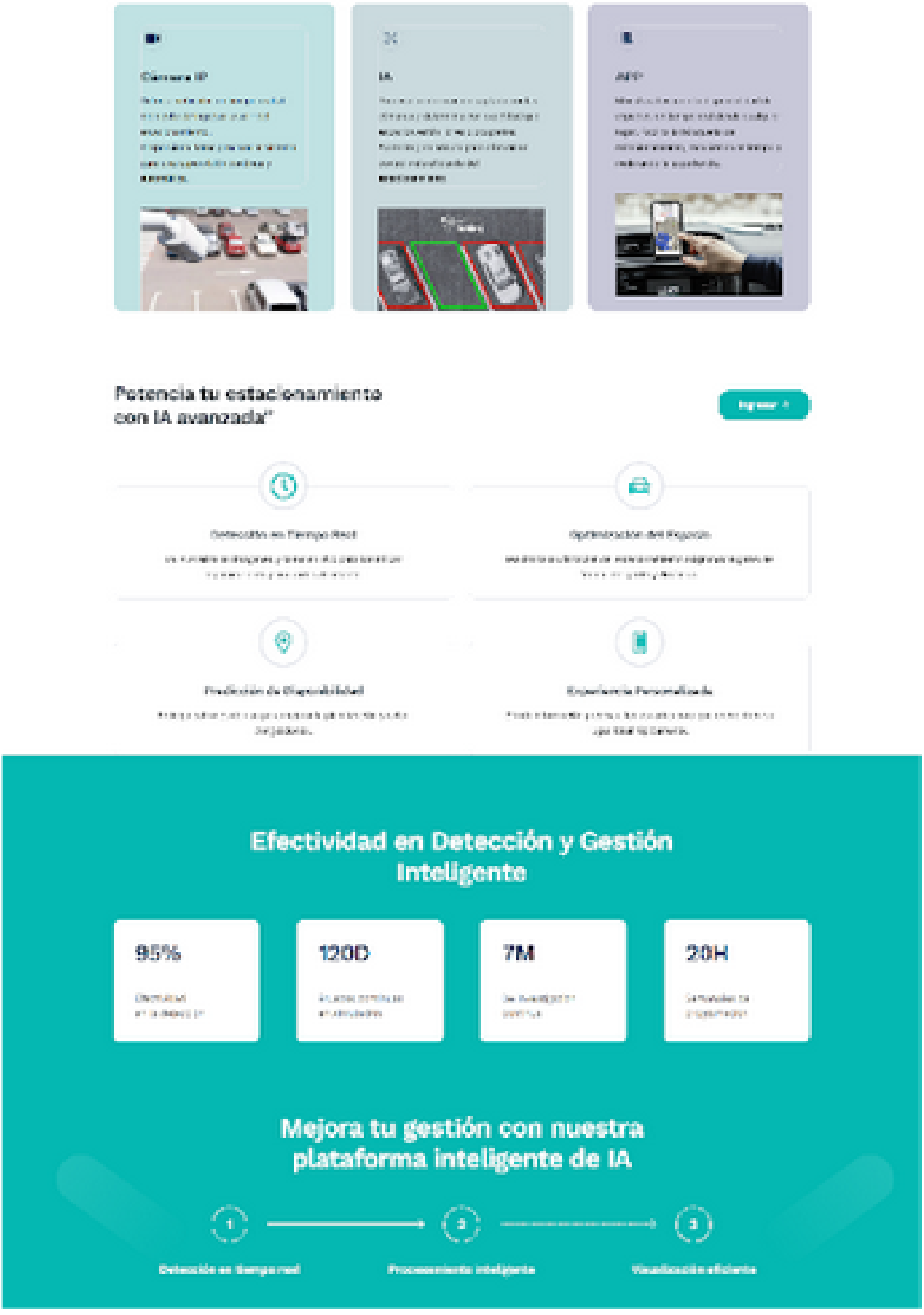
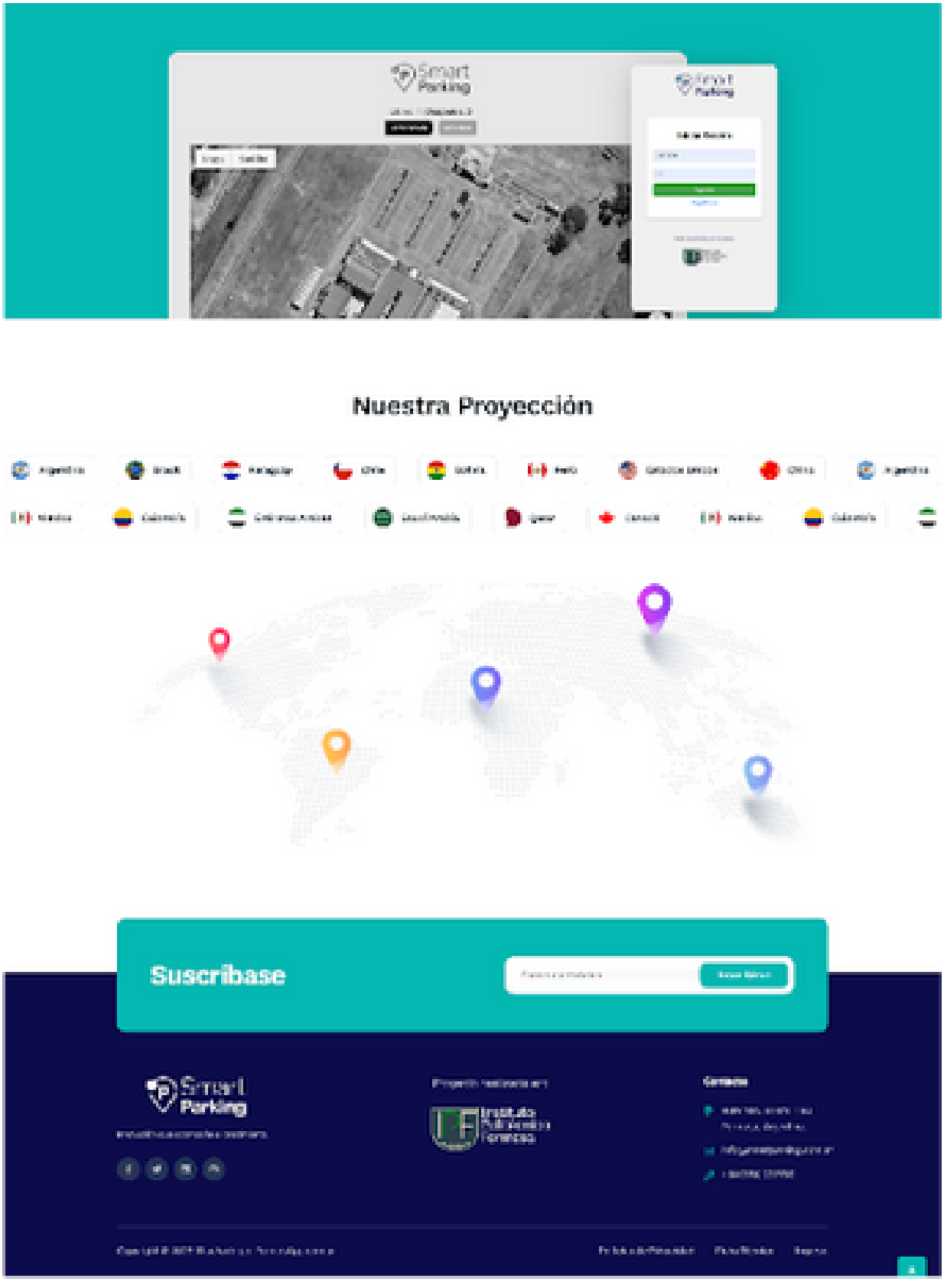


Figura 12.11. Landing Page (Sección 3).



Datasheet del sistema

Descripción General El sistema de Smart-Parking permite la detección y registro de espacios vacíos y ocupados de un estacionamiento mediante una cámara IP, conectada vía cable UTP a un router, el cual a su vez está vinculado a una notebook HP 17 que actua como servidor que a la vez procesa la información y gestiona el modelo de IA que determina los espacios vacíos y ocupados.

Cuadro 12.1. Especificaciones Cámara IP Esvin 2MP Interior.

Característica	**Especificación**
Marca / Modelo	Esvin IP Camera 2MP Interior
Tipo	Cámara IP de vigilancia
Resolución	1920×1080 (Full HD)
Sensor	CMOS 2 Megapíxeles
Conectividad	RJ45 - Cable UTP (Ethernet)
Compresión de Video	H.264/H.265
Alimentación	PoE (Power over Ethernet) / DC 12V
Ubicación Recomendada	Interior
Funciones Extras	Visión nocturna, detección de movimiento

Conectividad y Red

Cuadro 12.2. Especificaciones Conectividad UTP.

Característica	**Especificación**
Tipo	UTP Cat5e/Cat6
Longitud típica	Hasta 100 metros
Velocidad soportada	100 Mbps-1 Gbps
Uso principal	Transmisión de datos IP

Cuadro 12.3. Especificaciones Router.

Característica	**Especificación**
Función principal	Distribución de red LAN/WAN
Puertos Ethernet	4 LAN + 1 WAN (RJ45)
Protocolos soportados	IPv4/IPv6, DHCP, NAT
Seguridad	WPA2/WPA3
Velocidad soportada	100 Mbps/1 Gbps

Servidor - Notebook HP i7 16GB

Cuadro 12.4. Especificaciones Servidor.

Característica	Especificación
Marca/Modelo	HP Notebook
Procesador	Intel Core i7 (8/10/11ª gen aprox.)
Núcleos / Hilos	4-8 núcleos/8-16 hilos
Memoria RAM	16 GB DDR4
Almacenamiento	SSD 512 GB (o superior)
SO recomendado	Windows 10/11 Pro o Linux Ubuntu 22+
Función en el sistema	Servidor central de gestión y análisis
Conectividad	Ethernet RJ45 y WiFi

Especificaciones Generales del Sistema

Cuadro 12.5. Resumen de Componentes y Funciones.

Componente	Conexión	Función Principal
Cámara IP Esvin 2MP	UTP Router	Captura de video y detección de vehículos
Router	LAN/WAN	Gestión de red y distribución de datos
Notebook HP i7 16GB	LAN (Ethernet)	Servidor de procesamiento y almacenamiento

Futuras mejoras (Gráficos)

Figura 12.12. Arquitectura escalable: plano 1 (40 Lugares).

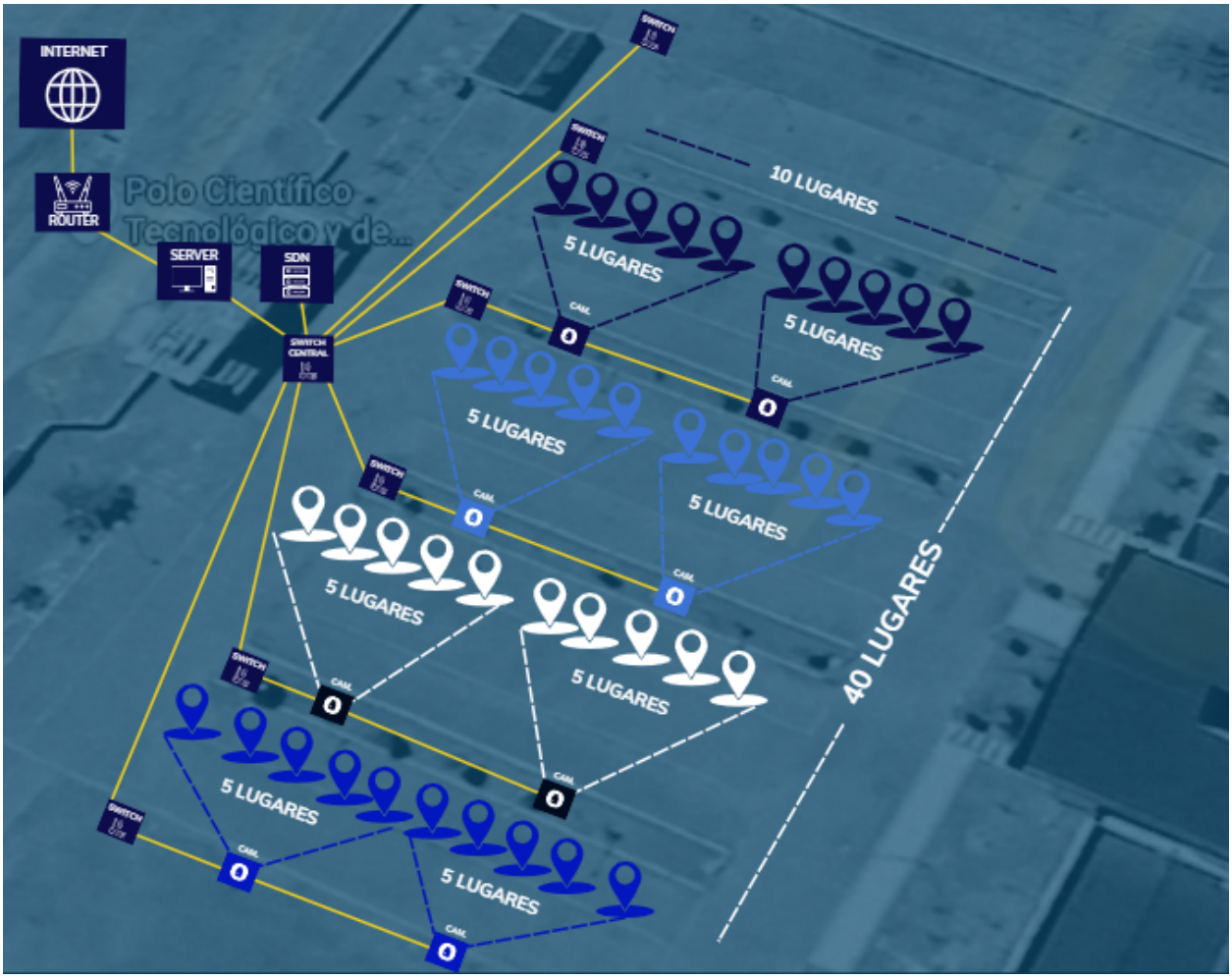


Figura 12.13. Arquitectura escalable: plano 2 (40 Lugares).

Figura 12.14. Sistema de alimentación fotovoltaico autónomo.

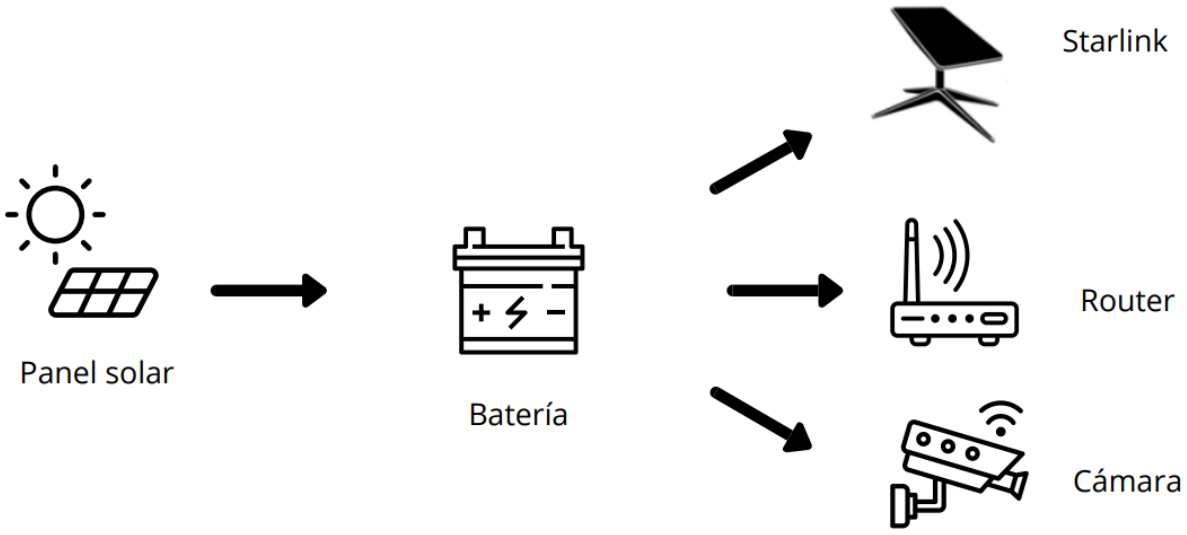


Figura 12.15. Integración con el Sistema de Estacionamiento Medido (SEM) municipal.

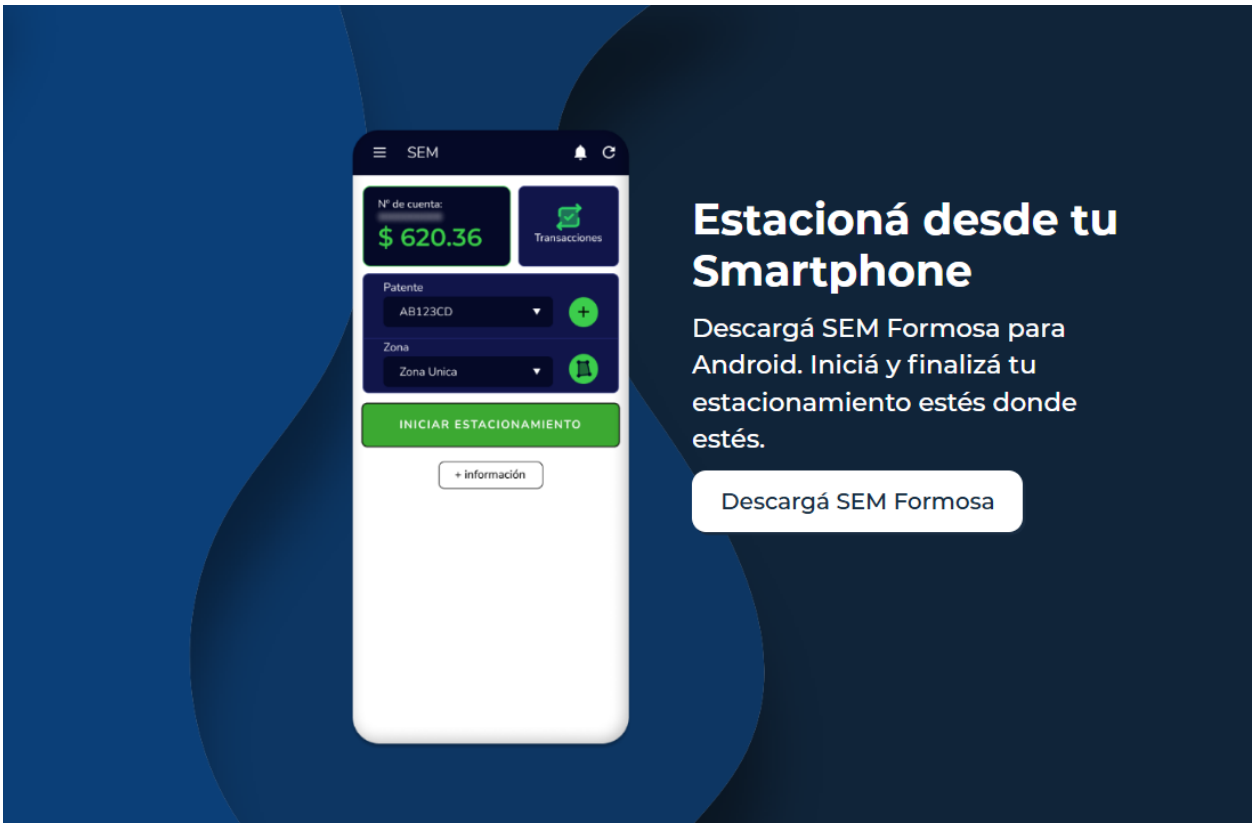


Figura 12.16. Sistema alternativo de conectividad (Starlink).

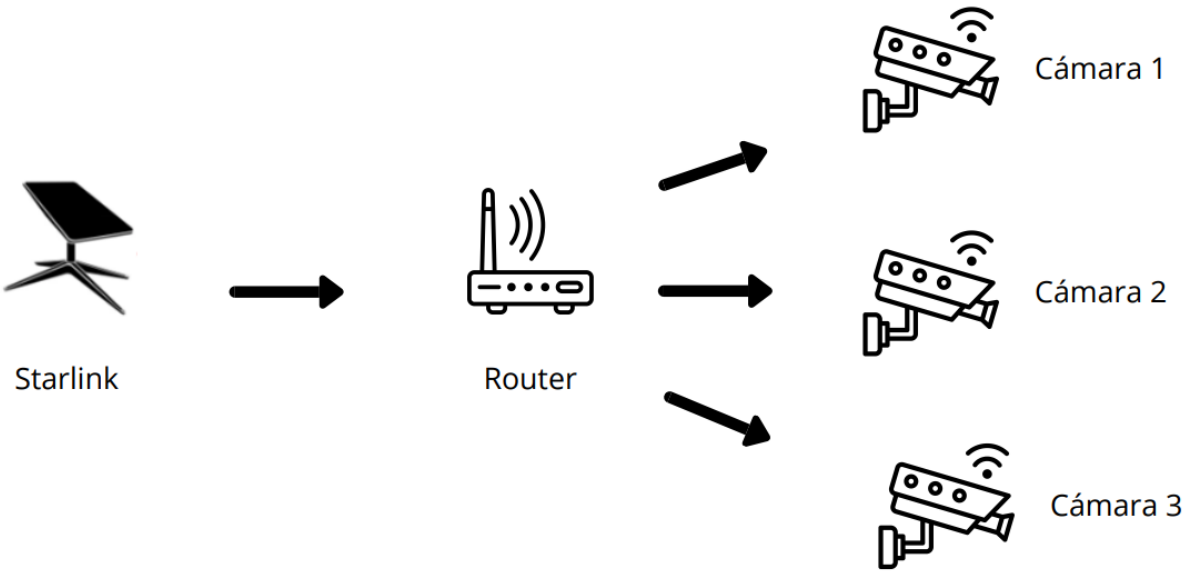


Figura 12.17. Incorporación de cámaras de alta resolución (Hikvision) para detección de patentes.

• Especificación

Cámara	
Tipo de sensor	CMOS de escaneo progresivo de 1/3"
Resolución máx.	2688 × 1520
Iluminación mín.	Color: 0.023Lux@(F1.9, AGC ON) B/N: 0.011Lux@(F1.9, AGC ON) 0 Lux con IR activado
Contraventana	De 1/30 s a 1/100.000 s
Interruptor día/noche	Soportado
Filtro de corte IR	Filtro de corte IR Módulo de vidrio azul para reducir el fenómeno fantasma
Lente	
Tipo de lente	Lente varifocal, lente motorizada de 3,1 a 6 mm
Apertura	De 3,1 a 6 mm: F1,9
Distancia focal y campo de visión	De 3,1 a 6 mm, campo de visión horizontal: de 105,5° a 51,4°, campo de visión vertical: de 55,9° a 28,9°, campo de visión diagonal: de 126,9° a 58,2°
Focus	Automático, lente varifocal, lente accionada por motor
Suplemento Ligero	
Tipo de luz complementaria	Luz IR, luz blanca cálida
Gama de luz complementaria	Video: 3,1 mm a 6 mm: 10 m Matrícula del vehículo: 3,1 mm a 6 mm: 5 m
Longitud de onda IR	Luz blanca cálida 3000 K, luz IR 850 nm
Imagen	
WDR	140 dB
Flujo principal	50 Hz: 25 fps (2688 × 1520, 1920 × 1080, 1280 × 720) 60 Hz: 30 fps (2688 × 1520, 1920 × 1080, 1280 × 720)
Sub-Flujo	50 Hz: 25 fps (1920 × 1080, 1280 × 720, 704 × 576, 640 × 480) 60 Hz: 30 fps (1920 × 1080, 1280 × 720, 704 × 480, 640 × 480)
Configuración de imagen	Saturación: 0 a 100 Brillo: 0 a 100 Contraste: 0 a 100 Ganancia: 0 a 100 3D DNR: Se pueden seleccionar los modos Desactivado, Normal y Experto. Balance de blancos: Se pueden seleccionar el Balance de blancos 1 y el Balance de blancos 2.
Interruptor de parámetros de imagen	Sí
Conmutador WDR	Encendido, Tiempo, Brillo
Velocidad de fotogramas	25 fps (P)/30 fps (N)
Video	
Retorno de la inversión	8 regiones respectivamente para flujo principal y subflujo, 6 niveles para cada región.
Compresión de video	Flujo principal: H.265/H.264/MJPEG, Subflujo: H.265/H.264/MJPEG,
Tipo H.264	Perfil bajo Perfil medio Perfil alto
Velocidad de bits de video	De 32 Kbps a 16 Mbps
Tipo de velocidad de bits	Constante, Variable
Codificación de video escalable (SVC)	Codificación H.265 y H.264

Modo de captura	
Tipo de disparador	Detección de video, bobina de E/S, tráfico mixto de radar
Tipo de imagen	Imagen de escena, Imagen de escena + Imagen de primer plano
Modo de escena	Entrada y salida, puerta de peaje, entrada y salida de estacionamiento subterráneo
Tipo de captura	Modo de luz estroboscópica
Parámetros de captura	
Formato de imagen	JPEG
Resolución de captura	2688 × 1520
Reconocimiento de matrículas	Adelante, Atrás, Bidireccional
Filtro de placa falsa	Sí
LED Control	Admite el control automático de la luz / control de tiempo de los LED
Reconocimiento inteligente	ANPR, función de vehículo motorizado, reconocimiento de color del vehículo, permitir el reconocimiento del fabricante del vehículo
Superposición de imagen de captura	Ubicación, hora de captura, número de placa, número de captura, color del vehículo Tipo de vehículo
Entrada y Salida	
Modo de control de puerta de barrera	Por Cámara/Plataforma/Mixto
Mantenga el poste de la pluma abierto para seguir al vehículo	Habilitar/Deshabilitar
Puerta de barrera de bloqueo para vehículos de gran tamaño	Sí
Salida de relé	Salida de relé de 2 canales, que admite la apertura y el cierre de puertas de barrera
Estado de la puerta de barrera	Ninguno, puerta de barrera cerrada en su lugar, puerta de barrera abierta en su lugar, control remoto abierto en su lugar
Lista de permitidos y lista de bloqueo	Hasta 500,000 listas de permitidos y listas de bloqueo en total Nota: Solo cuando la tarjeta TF está disponible, la función puede surtir efecto.
Red	
Almacenamiento en red	Compatible con NAS (NFS, SMB/CIFS), reposición automática de red (ANR) junto con la tarjeta de memoria Hikvision de gama alta, el cifrado de tarjetas de memoria y la detección de estado.
Protocolo	TCP/IP, ICMP, HTTP, HTTPS, FTP Upload Picture, SRTP, DHCP, DNS, DDNS, RTP, RTSP, RTCP, NTP, SNMP, IGMP, QoS, IPv6, UDP, SSL/TLS, WebSocket
API	Interfaz de video en red abierta ONVIF (PROFILE S, PROFILE G, PROFILE T), ISAPI, SDK, ISUP, Otap, Guarding Vision Platform
Cliente	Hik-Central, SDKdemo, PMS
Navegador	Plug-in requerido visualización en vivo: IE10+ Plug-in free live view: Chrome 57.0+, Firefox 52.0+, Safari 11+ Servicio local: Chrome 41.0+, Firefox 30.0+
Seguridad	Protección con contraseña, contraseña complicada, cifrado HTTPS, marca de agua, filtro de direcciones IP, autenticación básica y implícita para HTTP/HTTPS, WSSE y autenticación implícita para Open Network Video Interface, RTP/RTSP A TRAVÉS DE HTTPS, Configuración de tiempo de espera de control, registro de auditoría de seguridad, TLS 1.2
Interfaz	
Audio	1 entrada (entrada de línea), 1 salida (salida de línea)
Modo de comunicación	1 puerto Ethernet RJ45 10M/100M