Universidad de Concepción
Departamento de Ingeniería Informática
y Ciencias de la Computación

SISTEMA DE GESTIÓN DE ESTACIONAMIENTOS MEDIANTE EL USO DE HERRAMIENTAS IoT Y VISIÓN COMPUTACIONAL EN TIEMPO REAL

Antonio Iván San Martín Mora

Memoria presentada para la obtención del título de Ingeniero Civil Informático

Patrocinante: Marcela Varas Contreras
Comisión Evaluadora: Pedro Pinacho D. Julio Godoy D.

Mes de Abril, 2024

Índice

Índice de Figuras

Índice de Tablas

1. Introducción

1.1 Antecedentes Generales del Problema

Diariamente los usuarios de los estacionamientos de la Universidad de Concepción, se ven afectados por la aglomeración vehicular; en su mayoría causada por el constante aumento del parque vehicular en Chile. Este fenómeno impacta a todos los estacionamientos de la Universidad, pero es aún más evidente en aquellos con poca visibilidad, como es el caso del estacionamiento de la Facultad de Ingeniería, en donde se provocan constantes confusiones respecto a si hay o no estacionamientos disponibles.

Éste estacionamiento consta de una sola entrada y salida de vehículos con una barrera de acceso, por lo que el ingreso está restringido a las personas que porten tarjeta de acceso.

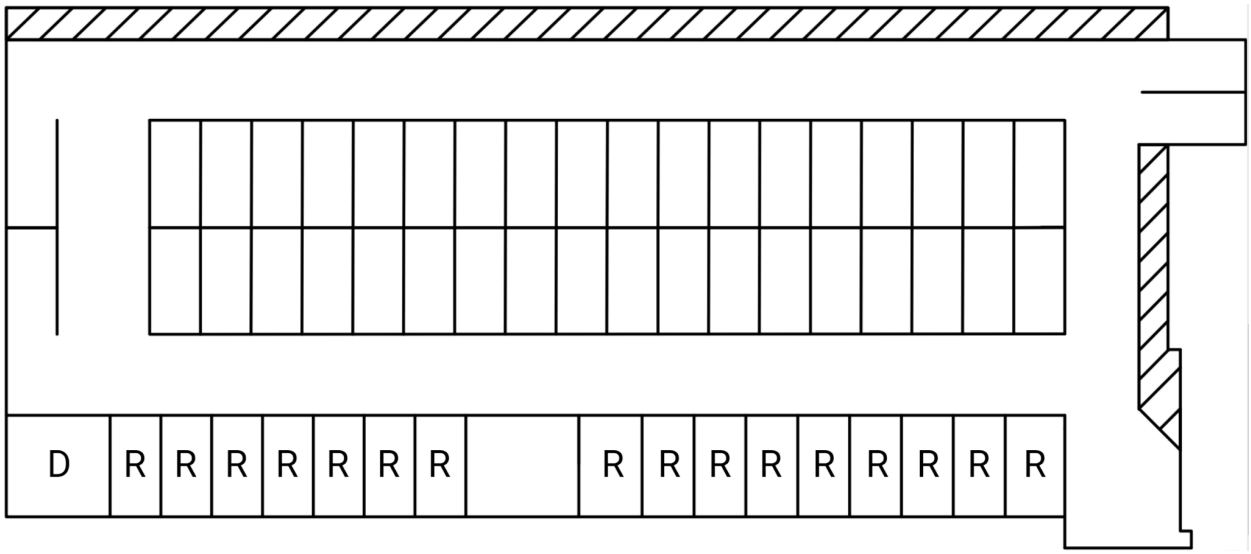

Figura 1. Diagrama a escala del estacionamiento utilizado. D: Reservado para personas en situación de discapacidad. R: Reservado para cargos específicos de la facultad. Los espacios que no están marcados se consideran disponibles para todo público.

Este estacionamiento consta de 57 espacios disponibles distribuidos en 2 filas principales de 18 espacios cada una, con acceso abierto a todo público, 2 filas laterales a la entrada del edificio con 8 y 9 espacios respectivamente, todos reservados para cargos específicos de funcionarios de la facultad, a excepción del espacio del extremo izquierdo que está reservado para personas en situación de discapacidad. Además existen 4 espacios distribuidos en lugares que no afecten al tránsito del espacio, 2 en el extremo izquierdo y 2 en la esquina superior derecha.

1.2 Solución Propuesta y Alcances

En base al análisis de la situación descrita anteriormente, se propone desarrollar una solución que le posibilite a los usuarios de los estacionamientos saber por anticipado si hay o no disponibilidad. Uno de los costos asociados a la no visibilidad de la disponibilidad de estacionamientos, es el tiempo que se ocupa en ingresar, abrir la barrera, revisar el estacionamiento y tener que retirarse para realizar el mismo proceso en un nuevo recinto.

Una solución innovadora es clave, ya que las alternativas más convencionales, como agrandar los espacios o distribuirlos de otra manera, muchas veces requieren de una gran inversión o son de difícil implementación. Y si se trata de innovación, a menudo se habla de Internet of Things [1].

La solución planteada a esta problemática consta de un sistema de IoT (Internet of Things) con visión computacional que sea capaz de reconocer los espacios disponibles en los estacionamientos y enviar esta información hacia una aplicación móvil, en la que se pueda visualizar con anterioridad, si existen espacios en donde poder estacionar o no, de manera que se evite el proceso de dar vueltas buscando un espacio e interrumpir o entorpecer el recorrido de los demás vehículos con esta misma problemática. El proyecto considera un MVP (Minimum Viable Product) como entregable, en el que se implementen estas características para obtener un resultado preliminar, considerando que:

- Los vehículos están correctamente estacionados
- Se respetan los estacionamientos reservados de todo tipo
- Las imágenes tienen la calidad suficiente para generar resultados
- Los ángulos de las cámaras generan una descripción fidedigna del estacionamiento
- El sistema se puede escalar simple y constantemente para ajustarse a las necesidades del proyecto.

1.3 Objetivo General

Desarrollar un sistema de software que, mediante el uso de tecnologías innovadoras, sea capaz de detectar y contabilizar la cantidad de espacios de estacionamiento disponibles en tiempo real. Esta información será entregada a los usuarios de la manera más simple y directa posible.

1.4 Objetivos Específicos

1. Familiarizar a un equipo de desarrollo con la metodología de trabajo y el stack tecnológico.
2. Desarrollo de un sistema de visión computacional.
3. Desarrollo de una Aplicación móvil en Flutter.
4. Desarrollo de una API que conecte el sistema de visión computacional con la app.
5. Instalación de cámara IP en el estacionamiento de la Facultad de Ingeniería UdeC.
6. Gestión de dos MVPs durante el proyecto.

7. Aplicar períodos de testing durante al menos cada entrega de MVP.

1.5 Metodología

Este proyecto será parte de la Fábrica de Software UdeC [2] y por tanto será desarrollado por un equipo de estudiantes de Ingeniería Civil Informática conformado por:

Nombre	Cargo	Área de desempeño
Michael Ignacio Villanueva Torres	Desarrollador	Frontend
Mauricio Francisco Furniel Campos	Desarrollador	Frontend
Nicolás Eduardo Soto Soto	Desarrollador	Backend
Diego Emilio Rebollo García	Desarrollador Jr	Backend

Tabla 1. Equipo de trabajo Fábrica de Software.

Siendo estos liderados por el alumno, Antonio San Martín, quién tendrá el rol de líder técnico y Scrum Master.

Para este proyecto se utilizará la metodología ágil SCRUM [3] y por consiguiente se realizarán reuniones diarias, planificaciones, revisiones y retrospectivas de sprint.
Las plataformas a utilizar serán digitales, principalmente Jira, Google Meet y un repositorio en Git desde el que se podrá acceder siempre al proyecto.

1.6 Estructura del informe

El presente informe se estructura de la siguiente manera:
Capítulo 2 "Marco Teórico": repasa conceptos claves para el entendimiento del proyecto y distintas aproximaciones del estado del arte para situar contextualmente la solución presentada.
Capítulo 3 "Descripción de la propuesta": como su nombre lo indica, describe la propuesta y su arquitectura,de manera preliminar y el proceso para llegar a ella.
Capítulo 4 "Detalle de la propuesta": describe el proceso iterativo mediante el cual fue desarrollado el proyecto y sus avances incrementales.
Capítulo 5 "Evaluación de la propuesta": demuestra la solución en uso y cuál es la propuesta de solución para el seguimiento del proyecto.
Capítulos 6, 7 y 8: Concluye el informe con un análisis de todo el proyecto, sus referencias bibliográficas y un Anexo con información relacionada.

2. Marco Teórico

2.1 Internet of Things (IoT)

El concepto de Internet of Things es un paradigma que contempla una revolución tecnológica hacia un mundo inteligente e interconectado (Smart World [4]) mediante objetos (o grupos de objetos) con capacidad de conexión e intercambio de datos con otros dispositivos o sistemas a través de Internet u otras redes de comunicación mediante el uso de sensores, sistemas software y capacidad de procesamiento. Este tipo de tecnología puede aplicarse a casi cualquier ámbito de la vida cotidiana y por tanto es cada vez más accesible, no sólo para los usuarios, también para los desarrolladores. Es cada vez más sencillo integrar este tipo de tecnología a los distintos sistemas, debido a su gran conectividad. Ejemplos concretos de esta tecnología pueden ser los sistemas de SmartHome, que aprovechan una red de dispositivos interconectados, cuyo funcionamiento es lo más automático y/o manejable posible para el usuario. Dentro de estos sistemas pueden haber cafeteras inteligentes que preparan ciertos tipos de café en cuestión de segundos con solo presionar un botón, refrigeradores inteligentes que llevan control de qué hay en su interior e incluso pueden comprar lo que falte inmediatamente, luces que puedan ser regulables desde una aplicación de smartphone, etc.

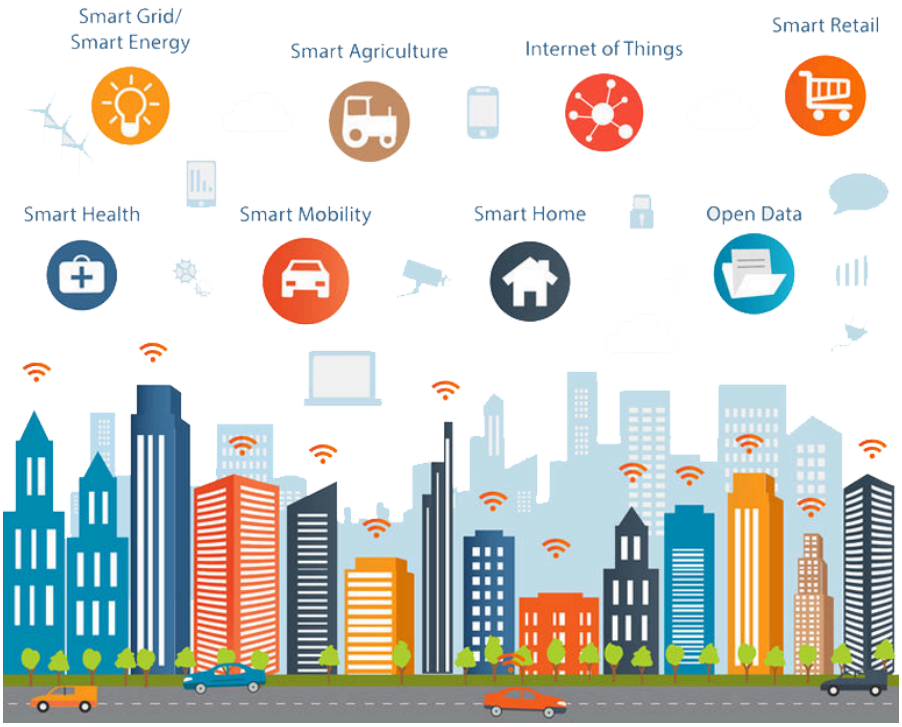

Figura 2. Ejemplos de uso de IoT

2.2 Frameworks de Desarrollo de Aplicaciones móviles

Actualmente, el desarrollo de aplicaciones es una herramienta fundamental para las organizaciones que buscan conectar con sus usuarios, debido a la globalización del uso de los smartphones y la demanda de los mismos por la conexión en tiempo real con la internet. El desarrollo de estas aplicaciones consiste en la creación de software para dispositivos móviles [5] y, por consiguiente, el desarrollo para plataformas de distintos sistemas operativos. Ésto nos lleva a la disyuntiva de qué plataformas elegir para el desarrollo, o si se debe realizar un desarrollo transversal a todos estos sistemas, lo que puede resultar abrumador para el equipo de desarrollo. Los sistemas operativos de dispositivos móviles más populares son Android e IOS, con el 70,19% y 29,12% del mercado de smartphones respectivamente para el Q4 de 2023 [6]. por lo que el desarrollo se puede limitar a estos dos sistemas operativos sin perder usuarios.

Ya con los sistemas operativos objetivo definidos, cabe considerar que existen distintas formas de abordar este desarrollo, el desarrollo de manera nativa y el desarrollo multiplataforma.

2.2.1 Desarrollo Nativo:

Los framework de desarrollo nativo están bien definidos y son estrictos en cuanto a herramientas para desarrollarlas. Una ruta usual de desarrollo nativo contempla la creación de 2 aplicaciones, la primera en Android hasta el desarrollo de un MVP para su posterior conversión al sistema IOS. Para el desarrollo nativo de aplicaciones Android, se utiliza el framework Android Studio y para IOS se utiliza Xcode, cuya principal diferencia es que Android Studio se encuentra disponible para todo sistema operativo, mientras que Xcode solo para MacOS.

2.2.2 Desarrollo Multiplataforma:

Para el desarrollo multiplataforma existen varios frameworks y usualmente no tienen mayores restricciones respecto a las herramientas capaces de interactuar con ellos.

Flutter

Es un framework de desarrollo de aplicaciones multiplataforma creado por Google, que utiliza el lenguaje de programación Dart para funcionar.
Entre sus principales características se encuentran:

- Es un proyecto open-source, por lo que es gratuito y se puede colaborar en su desarrollo.
- Capacidad de desarrollar aplicaciones móviles para IOS y Android simultáneamente.
- Su velocidad, ya que, al compilar a código nativo entrega el mejor rendimiento posible.
- Hot reload, que permite visualizar los cambios en la app sin necesidad de reiniciarla.
- Uso del lenguaje de programación Dart, debido a su simpleza y efectividad.

Debido a estas características [7], Flutter se utiliza en empresas pequeñas, con recursos limitados ya sea de tiempo o dinero e incluso en startups.

React Native

Es la competencia directa a Flutter como framework de desarrollo multiplataforma, existe desde antes que Flutter, está concebida como una librería de JavaScript para la construcción de interfaces de usuario y es actualmente mantenida por Meta.

Algunas de sus características son:

- Se puede utilizar sobre proyectos preexistentes o crear nuevos proyectos desde cero.
- Al utilizar JavaScript, es altamente compatible con desarrollos previos en plataformas web que utilicen React.
- Su código es fácilmente reutilizable y utilizable en forma de bloques, debido a su estructura de componentes React.
- Fast Refresh, al igual que en Flutter, se visualizan los cambios inmediatamente al momento de realizar un guardado.

Ya que no existe realmente la necesidad de utilizar APIs o lenguajes específicos al desarrollo nativo debido a que la aplicación solo se utilizará para desplegar la información obtenida desde la API, descartamos el desarrollo nativo por su complejidad de desarrollo. Nos inclinamos además al framework de desarrollo Flutter principalmente por su facilidad de uso y experiencia previa en el equipo, además de su buen rendimiento en todos los dispositivos debido a que compila a código nativo.

Figura 3. Flutter enfatizando su utilización de ventanas.

2.3 Visión Computacional

En el ámbito de la visión computacional se adquieren, procesan, analizan y comprenden imágenes con el objetivo de obtener información numérica o simbólica para su procesamiento en computación. Uno de los usos más claros de este tipo de métodos es el reconocimiento, seguimiento y clasificación de objetos. Al ser una forma de aplicación de Inteligencia Artificial, estos métodos se centran en realizar y automatizar tareas que replican el comportamiento humano, en este caso, la propia visión y comprensión intrínseca de esta.
Un algoritmo de visión computacional es, por consecuencia, una serie de instrucciones que una computadora utiliza para interpretar y comprender datos visuales de su entorno. Esta definición puede resultar poco concreta por lo que históricamente existen ciertos hitos que marcan de manera más precisa el uso práctico de esta tecnología.

- Inicios (1960 - 1980): Se desarrollan los primeros algoritmos de procesamiento de imágenes y reconocimiento de patrones. Su efectividad era limitada al momento de reconocer imágenes complejas.
- Machine Learning (1980 - 2000): Se aplican técnicas de Machine Learning al reconocimiento y clasificación de imágenes, obteniendo buenos resultados.
- Deep Learning (2000 - 2010): A partir del 2000 surgen las técnicas de Deep Learning que utilizan redes neuronales para reconocer patrones en imágenes y videos. Con la adición de estas técnicas se logra apreciar una mejora considerable en la precisión de este tipo de algoritmos.
- Sistemas en tiempo real (2010 - Presente): En la actualidad, el nivel de complejidad de software y hardware permiten a los sistemas de visión computacional ser ejecutados en tiempo real, lo que amplía fuertemente sus casos de uso.

Este tipo de tecnología está cada día más presente en nuestro día a día, desde los sistemas de seguridad en las tiendas de retail, hasta los sistemas software de las nuevas cámaras web que, por ejemplo, nos permiten manejar la imagen de nuestro entorno a nuestro antojo. El uso de ciertos algoritmos respecto a otros, puede definir que tipo de resultados se buscan y que tan cercanos a los objetivos son. Algunos de los algoritmos útiles para el contexto de este proyecto son:

2.3.1 SURF

Speeded Up Robust Features [8], Es un detector de características locales utilizado para reconocimiento de objetos y clasificación de imágenes. Está parcialmente inspirado en otro algoritmo detector llamado SIFT (Scale-Invariant Feature Transform).
Cuenta con 2 fases principales:

- Extracción de características: Se extrae información útil (características) de un input de tipo imagen. Estas características deben ser lo suficientemente representativas en términos de importancia y particularidad para ser consideradas.
- Descripción de características: La información extraída en la fase previa es descrita de manera lo suficientemente robusta para que existan puntos comunes entre diferentes versiones de un objeto o escenario entre imágenes, todo esto entregado en forma de un vector de 64 dimensiones.

El algoritmo SURF es varias veces más rápido y más robusto ante transformaciones de imágenes que su predecesor SIFT.

2.3.2 RESNET

Introducida por primera vez en 2015 [9], esta es una red neuronal convolucional diseñada para resolver el problema del gradiente desvaneciente/explosivo. Este problema, común en el entrenamiento de redes neuronales, se produce cuando se entrenan redes demasiado profundas y puede hacer que dicho proceso sea difícil y lento. Además, puede resultar en que la red produzca resultados pobres o inconsistentes.
La solución implementada en este algoritmo consiste en el uso de conexiones de “atajo” para la información que se entrega a las capas anteriores en el proceso de retropropagación. Debido a la complejidad (y, por ende, su profundidad) del tipo de redes utilizadas para las tareas de visión computacional, esta arquitectura de red es particularmente exitosa.

2.3.3 YOLO (You Only Look Once) V5

YOLO, es una serie de modelos de visión computacional diseñados para realizar tareas de detección de objetos en tiempo real. La principal ventaja de YOLO es su velocidad y precisión, ya que procesa imágenes en una sola evaluación de la red neuronal (de ahí su nombre “You Only Look Once”), en lugar de las múltiples revisiones que requieren otros sistemas. Esto lo hace ideal para aplicaciones que necesitan reconocimiento de objetos en tiempo real como la vigilancia de seguridad.

La arquitectura de las redes YOLO consta de tres partes fundamentales:

- Columna (CSP Backbone): Una red neuronal convolucional que agrega y forma características de imágenes en diferentes granularidades.
- Cuello (PA-Net Neck): Una serie de capas que mezclan características de imágenes para enviarlas hacia un “predictor”.
- Cabeza: Consume las características del cuello y genera predicciones de clase y caja.

Los modelos YOLO fueron los primeros en conectar la predicción de cajas con la de etiquetas de clase en una red diferenciable de extremo a extremo. YOLO V5 [10] tiene cuatro versiones principales: small (s), medium (m), large (l) y extra large (x), cada una con mejores índices de precisión que la anterior y, a su vez, con mayores tiempos de entrenamiento.

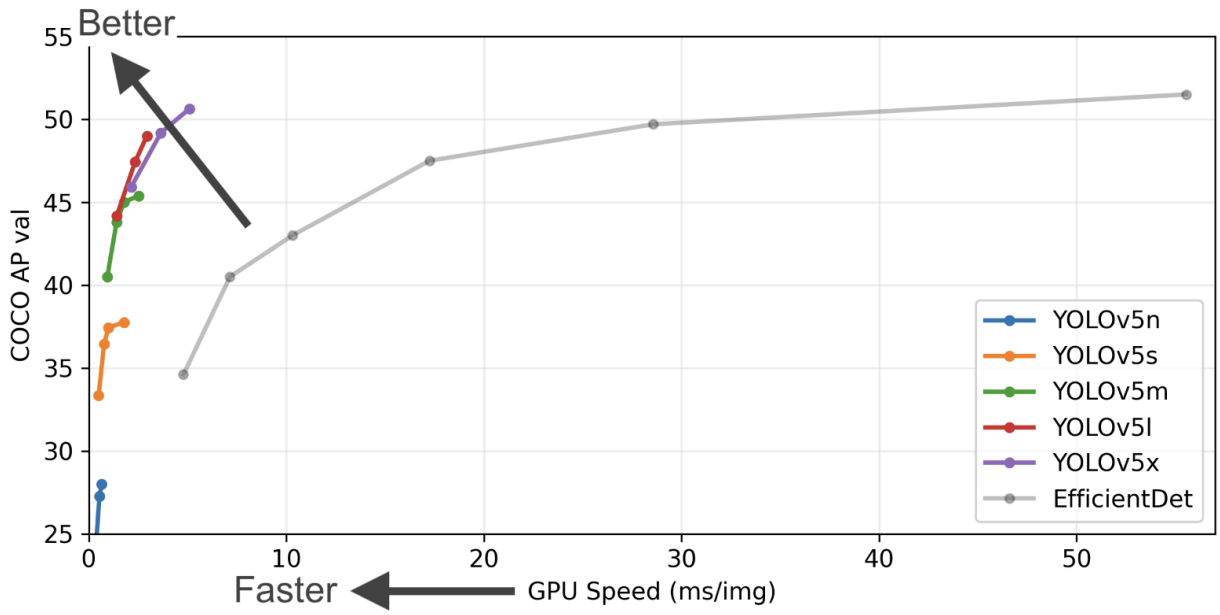

Figura 4. Relación Velocidad x COCO Score (Mejor rendimiento en dataset COCO) [10]

El procedimiento de entrenamiento principal de YOLOv5 es el Data Augmentation, que consiste en la transformación de los datos originales para añadir mayor variedad al entrenamiento. En este framework se utilizan principalmente los métodos de reescalado de imágenes, ajuste de espacio de color y aumento de mosaico, que incluyen:

- Reescalado de imágenes: Cambiar las dimensiones de las imágenes originales.
- Ajuste de espacio de color: Cambio de ciertos colores.
- Aumento de mosaico: Combinación de cuatro imágenes en una sola mediante cortes aleatorios.

Finalmente se seleccionó el modelo YOLO V5 por su eficiencia y rapidez de implementación, así como por la amplia documentación y ejemplos disponibles que facilitan su uso. La elección de este algoritmo también se basó en los resultados positivos obtenidos en pruebas previas, lo que demuestra su capacidad para adaptarse y funcionar eficazmente en distintos entornos y aplicaciones.

3. Descripción de la Propuesta

En este capítulo se explicará la solución planteada anteriormente, junto con el proceso de desarrollo de la misma, su diseño e implementación.

3.1 Soluciones preliminares planteadas

Al inicio del proyecto se realizaron reuniones para definir cómo abordar la problemática considerando soluciones previas [11]. En estas reuniones se aplicaron distintas metodologías de creatividad, como la lluvia de ideas o pensar fuera de la caja, todo ésto para ofrecer la mayor cantidad de soluciones posibles y evitar estancamientos en alguna solución que pudiese no ser la más óptima.

Algunas de las soluciones que se consideraron son:

- Persona contador: Una de las soluciones más simples que se plantearon. Consistía directamente en una persona que llevara contabilizada la cantidad de estacionamientos disponibles. La forma de llevar esta contabilidad podía ser simplemente un bloc de notas físico o desde un celular, o generar una app con interfaz propia que se ajustara a esta necesidad. Luego esta información sería procesada por un sistema backend que finalmente traslada la información a una app para usuarios del estacionamiento.
  Las dos principales desventajas de esta solución consisten en el costo asociado a la persona encargada de introducir la información y el carácter poco innovador de la aplicación generada.

- Cartel en la entrada: Otra solución más simple aún, un cartel (LED) en la entrada del estacionamiento que indique con una interfaz tipo semáforo, si existen espacios disponibles o no. Esta solución sólo contemplaba la idea de la vista inmediata que genera al llegar al edificio, por lo que puede ser considerada como complemento de otras soluciones.

- Cámaras: Sistema de reconocimiento de imágenes que utiliza cámaras para captar imágenes y mediante el uso de algoritmos de visión computacional, identificar los elementos presentes en la imagen, en este caso los autos estacionados.

- Sensores por cada espacio disponible: Instalación de sensores en cada espacio del estacionamiento conectados a una app. Probablemente una de las soluciones más precisas pero logísticamente complicada debido a su instalación y costo asociado.

- Contador en la entrada: Sistema automático de conteo de vehículos, consta de un sensor en la entrada del estacionamiento que se comunica con la app y entrega el resultado de los espacios disponibles, restando cada ingreso y sumando cada salida.

3.2 Solución propuesta

Luego de analizar las ideas previas se acordó que una solución que cumple con los objetivos de la Fábrica de Software, como lo son, la innovación y buena adaptación a los recursos con los que se cuenta, es la utilización de cámaras en conjunto con visión computacional para la contabilización de espacios disponibles en el estacionamiento. Además se llegó al consenso de que incluir una aplicación móvil es transversal a prácticamente todas las soluciones propuestas, por lo que se hizo indispensable generar una aplicación que funcione como visualizador de la cantidad de espacios disponibles generada por el sistema debido a su gran utilidad al poder acceder a ella desde cualquier parte.
A grandes rasgos, el sistema completo consistiría de cámaras para detectar los espacios disponibles, posteriormente la imagen pasa por un sistema de VC que entrega la cantidad de autos detectados que son restados de la cantidad de espacios totales en el estacionamiento, resultando en los espacios disponibles. Esta información es entregada a la aplicación que funciona sólo como un visualizador de la misma.

3.2.1 Arquitectura de la solución propuesta

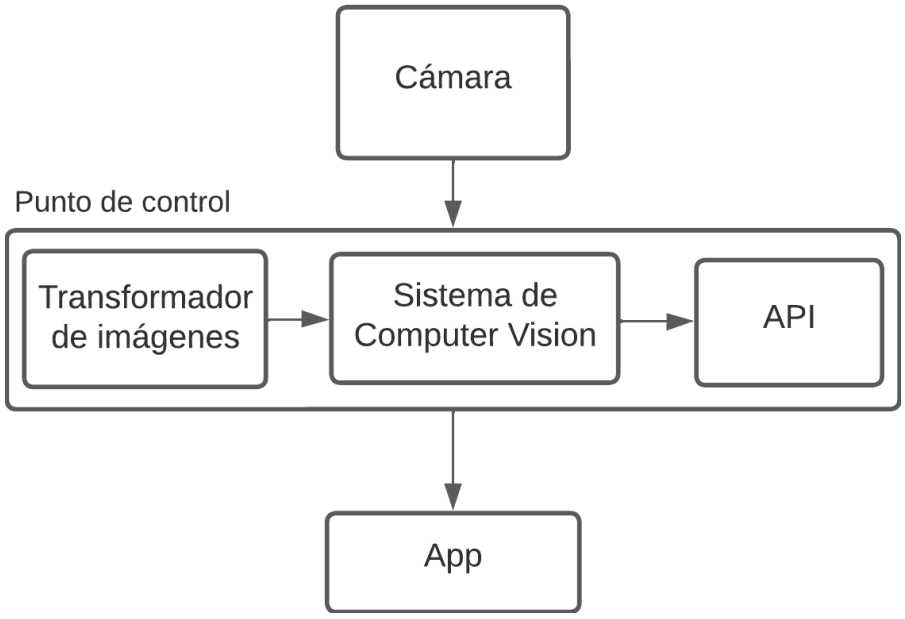

Figura 5. Arquitectura del sistema.

Elementos en la arquitectura:

Cámara: Instalada preliminarmente en el frontis del edificio de sistemas, capta imágenes en tiempo real del estacionamiento de la facultad.
Punto de Control: Lugar físico en el que se desarrolla el sistema de captación, transformación y procesamiento de las imágenes y su posterior envío de resultados hacia la API.
Transformador de imágenes: Recibe periódicamente las imágenes desde la cámara hacia un sistema de almacenamiento y transforma el formato para ser compatible con el proceso de Computer Vision.
Sistema de Computer Vision: Recibe las imágenes ya transformadas y las procesa mediante un algoritmo de redes neuronales para ver qué objetos están presentes en ellas.
API: Recibe el resultado de los objetos presentes en las imágenes y lo almacena para la app.
App: Consulta el resultado desde la API y la despliega en su interfaz gráfica.

La instalación de la cámara para este prototipo será una emulación en la que se usará una cámara IP de vigilancia para adquirir imágenes del estacionamiento, de la misma forma el punto de control utilizado será preliminarmente un computador personal de tipo notebook en el que se ejecutarán los sistemas de transformación de imágenes, computer vision y API.

4. Detalle de la propuesta

4.1 **Herramientas utilizadas**

Para el correcto funcionamiento del proyecto, el equipo utilizó las siguientes herramientas de desarrollo:

Herramientas Software:

Para el desarrollo de la aplicación móvil:

- Flutter
- Dart
- Android Studio/Xcode

Flutter y Dart se utilizaron para la creación de la aplicación móvil en ambientes multiplataforma y los entornos de desarrollo Android Studio/Xcode para realizar pruebas.

Para el desarrollo del sistema de visión computacional:

- Python
- Pytorch
- scikit-learn
- matplotlib
- numPy

Estos frameworks de desarrollo se eligieron por su facilidad de uso y vasta cantidad de documentación y ejemplos disponibles en la red.

Para el desarrollo de la API:

- FastAPI
- uvicorn
- websockets

Se utilizó FastAPI por su facilidad de implementación en Python, uvicorn por su alta compatibilidad con FastAPI y websockets para la comunicación en tiempo real entre el servidor y la aplicación móvil.

** Las versiones específicas de estas herramientas se encuentran en el documento requirements.txt del correspondiente repositorio de github*

Herramientas Hardware:

PC

- Macbook Pro M1 2020

** El equipo de desarrollo trabajó en diversos sistemas de características similares*

Cámaras

- Cámara IP
- Webcam Logitech Streamcam 1080p
- Webcam LinkOn 1080p

4.2 Diseño de componentes

En esta sección se explicará el diseño de cada componente del proyecto de manera individual, contemplando la lógica detrás de cada resultado obtenido, a través de las distintas iteraciones y cambios realizados.

4.2.1 Aplicación Móvil

Como se mencionó anteriormente, una de las ideas con mayor acogida por parte del equipo consistía en una aplicación móvil capaz de desplegar la cantidad de espacios disponibles en el estacionamiento sin necesidad de estar cerca de este, haciendo que el conductor pueda planificar su viaje con antelación o chequear el estado del estacionamiento antes de entrar.
El diseño de la aplicación se realiza desde el sprint 2 y su desarrollo contempla desde el sprint 3 hasta el final.

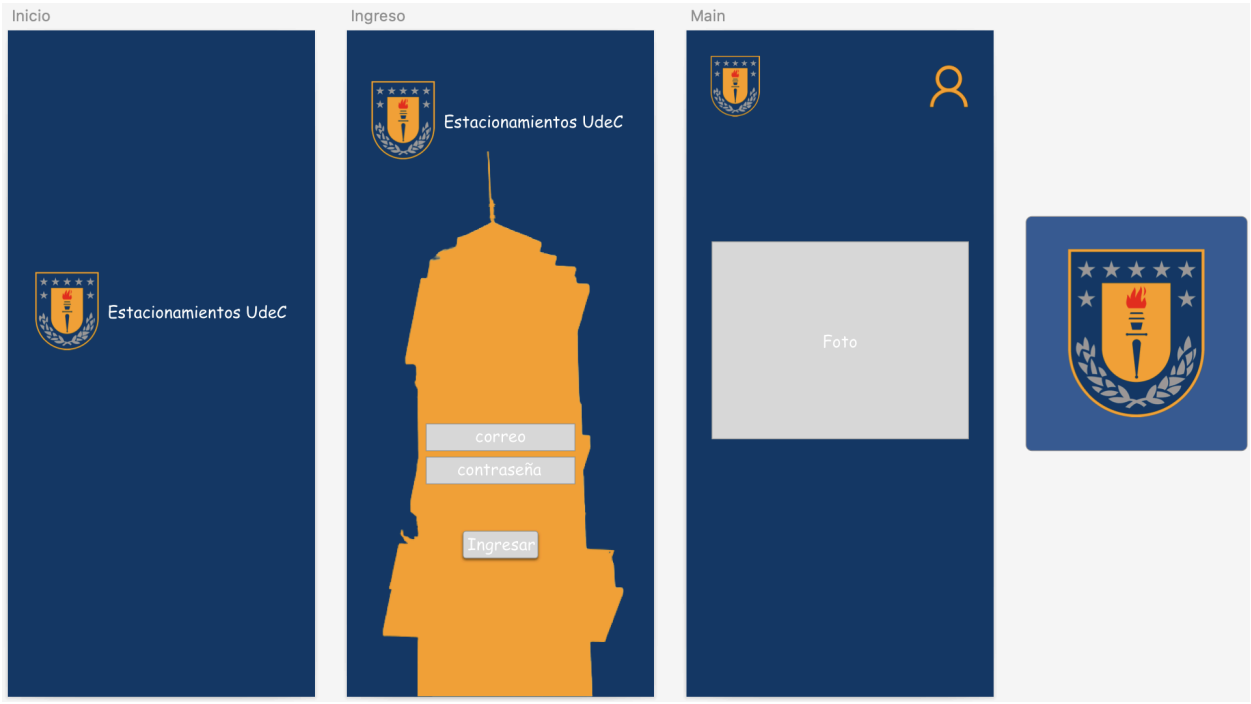

Figura 6. Diseño Inicial del sistema.

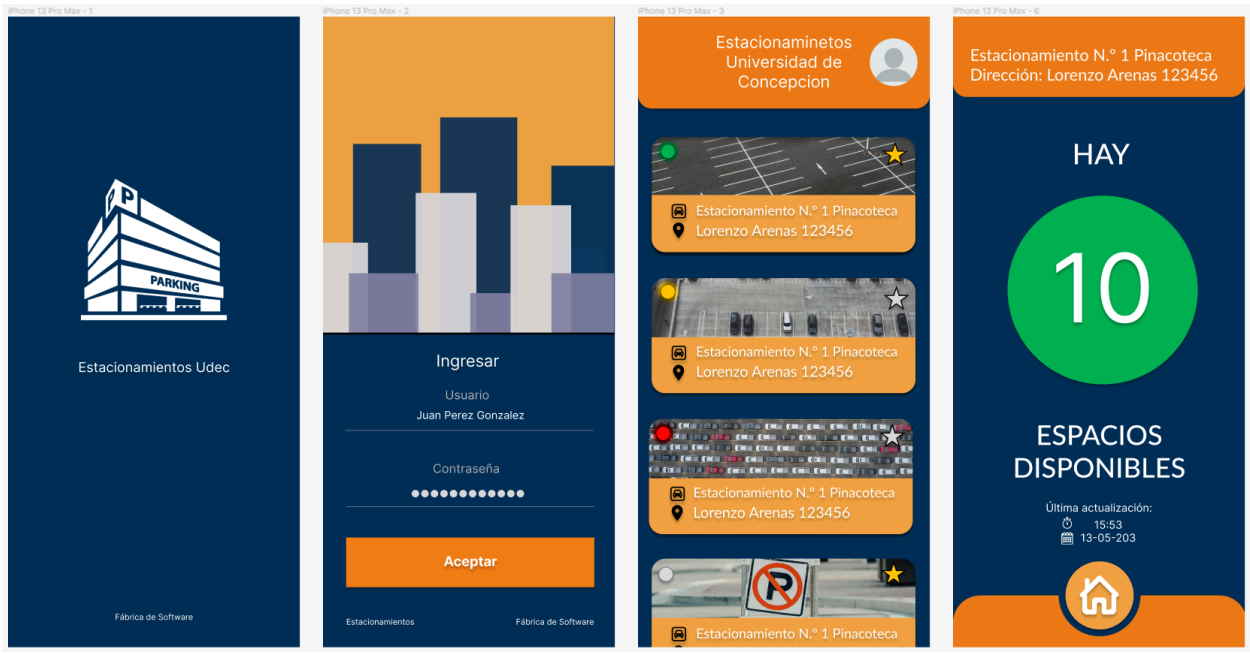

Figura 7. Diseño actual del sistema. Pantallas de inicio y de perfil.

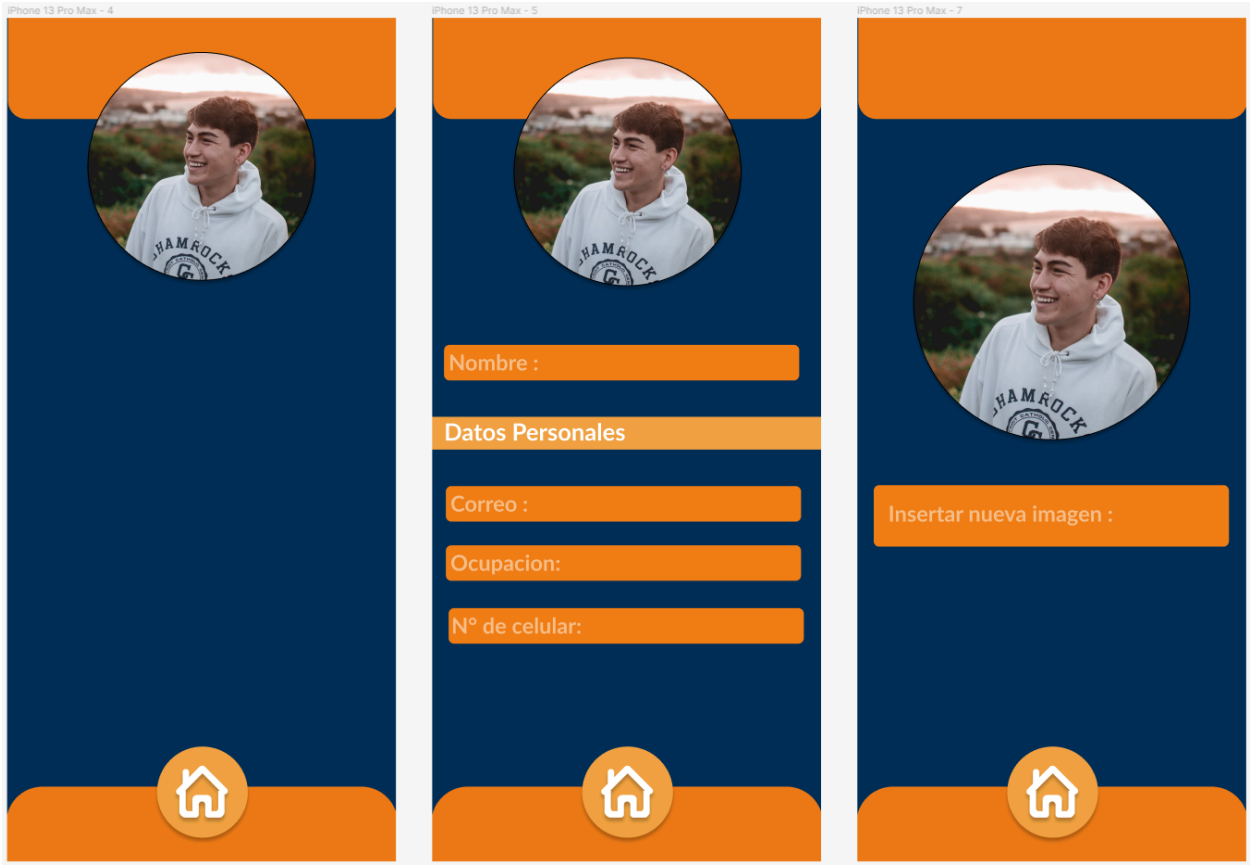

Figura 8. Diseño actual del sistema. Pantalla de perfil

Como se puede apreciar en las figuras 6,7 y 8, el diseño fue evolucionando para ajustarse a las necesidades del desarrollo.
La aplicación móvil se realizó utilizando el framework Flutter a partir del diseño generado inicialmente. Este diseño contempla una pantalla de inicio con estructura de Login en la que se debe ingresar un usuario y una contraseña, con el objetivo de diferenciar usuarios dentro de la aplicación, una pantalla principal de selección de estacionamiento (considerando que en un futuro se pueda aplicar esta tecnología a otros estacionamientos de la universidad), el visualizador del contador del estacionamiento (información consultada a la API) y una pantalla de perfil donde realizar ajustes y/o consultar información personal.

4.2.2 API

El funcionamiento de una API en el contexto del proyecto es probablemente lo que fue más claro desde el inicio del proyecto, se necesitaba generar una conexión entre dos componentes de software, por lo que el proyecto siempre la contempló.
El proceso de creación de la API fue expedito, durante el quinto sprint se refactorizó código proveniente de un proyecto anterior, manteniendo la idea original y solo ajustándose a nuevos componentes. Esta idea consiste en una API generada por FastAPI, que constantemente realiza llamados al sistema de Visión computacional para tener actualizado el valor de espacios disponibles en el estacionamiento. Detalle en Anexo 1.2

4.2.3 Visión Computacional

Para reconocer la cantidad de espacios disponibles en el estacionamiento se utilizó un sistema de visión computacional que implementa el algoritmo YOLOv5. Esta solución fue desarrollada con el objetivo de disminuir la complejidad que hubiese implicado realizar un sistema propio que cumpliera esta función. Detalle en Anexo 1.1

Aplicando la metodología indicada en el capítulo 1.5, se desarrollaron 6 sprints en los que se realizaron los siguientes avances:

	Fecha de inicio	Fecha de término	Detalle
Sprint 0	04 de Abril	07 de Abril	Se presenta al grupo, define el stack tecnológico, metodología y las plataformas a utilizar.
Sprint 1	07 de Abril	24 de Abril	Se hacen pruebas del proyecto anterior y debido a confusiones respecto a su código, se desarrolla un nuevo repositorio de github con el objetivo de refactorizar este código existente para mejorar su legibilidad, diseño y estructura.
Sprint 2	24 de Abril	08 de Mayo	Rediseño de la app en herramientas de diseño UX/UI y primera fase de refactorización de Backend en que se documenta el código de la versión anterior.
Sprint 3	08 de Mayo	22 de Mayo	Comienzo del desarrollo de la app y corrección de errores del sistema de Visión Computacional
Sprint 4	22 de Mayo	05 de Junio	Prototipo de la app funcional independientemente y sistema de Visión Computacional también funciona independientemente con pruebas de imágenes estáticas.
Sprint 5	05 de Junio	19 de Junio	Desarrollo y conexión de la API con los sistemas de la app y el elemento de Visión Computacional.
Sprint 6	19 de Junio	03 de Julio	Pruebas físicas del sistema completo funcionando.

Tabla 2. Planificación del desarrollo

La distribución del trabajo en equipo fue directamente relacionada a los roles asignados inicialmente, el equipo de Frontend se dedicó a realizar el rediseño de la app y el ajuste e incorporación de nuevos elementos en la aplicación móvil, por ejemplo, la creación de un selector de estacionamientos. Mientras que el equipo de Backend implementó el sistema de detección mediante YOLO V5, se realizaron diferenciaciones de etiquetas y correcciones en el sistema, además de la implementación de las conexiones entre sistemas como la API y el sistema de visión computacional. El líder técnico del equipo realizó labores de soporte y gestión como Scrum Master además de la creación de funciones de utilidad como la obtención de imágenes con resultados y la versión con soporte a dos cámaras. Para más detalle referirse al Anexo 1.

5. Evaluación de la propuesta

5.1 Resultados

Para la captación de resultados se utilizaron las herramientas de hardware previamente descritas (sección 4.1), utilizando imágenes previamente captadas en el estacionamiento de la Facultad y utilizando las mismas para pruebas en tiempo real con 2 cámaras web a modo de prueba. Al posicionar estas cámaras de manera correcta, se hacía un llamado a la API a través de la aplicación, lo que desencadenaba el sistema de reconocimiento con una imagen de ese instante, la imagen pasa por el sistema y se entrega el resultado correspondiente a la API, que es llevada finalmente a la aplicación para ser desplegada.

Figura 9. Resultados de la cámara 1. 29 aciertos de 32 vehículos en la imagen.

En este análisis de resultados, se rescata un estado del estacionamiento que demuestra abiertamente los posibles escenarios adversos a los que se puede enfrentar la solución. En este caso, se indica que hay 4 espacios disponibles, cuando en realidad hay solo 2, esto debido a que:

- Los ángulos captados consideran una totalidad de 69 espacios disponibles debido a la superposición de espacios, cuando sólo son 57.
- No se captan ciertos vehículos, ya sea porque otros vehículos o el ambiente los cubren casi totalmente o debido a su lejanía con las cámaras.
- Se captan los vehículos que transitan en el estacionamiento, sin necesidad de estar utilizando un espacio (Camión blanco en Figura 9).

Un factor fundamental al momento de realizar este análisis, es el ángulo cubierto por las cámaras. Como se puede apreciar, los distintos ángulos pueden influenciar en los resultados obtenidos, de manera que, si se utilizaran ángulos que superponen espacios en el estacionamiento, por la manera en que funciona la solución, se altera el resultado al obtener vehículos que aparezcan de manera duplicada. Además, el uso de dos cámaras simultáneas debiese implicar la inclusión de un sistema que diferencie los espacios individuales o sectores específicos para el cálculo dentro del estacionamiento. El sistema actual recibe imágenes independientes y calcula los espacios disponibles en base a la resta entre los vehículos detectados y los espacios totales del estacionamiento, por lo que los sectores de visión compartida entre las imágenes implican la repetición de espacios. La solución a este problema puede ser la manipulación de la imagen al momento de la instalación de las cámaras, a modo de seccionar el estacionamiento artificialmente o la sofisticación del sistema de reconocimiento para detectar cada espacio disponible independientemente. Ambas soluciones escapan del alcance de esta investigación, por lo que se considerará como una recomendación de trabajo futuro.

Otra posible causa de resultados imprecisos en el proyecto es el conjunto de imágenes de entrenamiento utilizados para el sistema de visión computacional. Este sistema fue entrenado con las imágenes propias de la librería original de YOLO v5, por lo que también se recomienda que, en el momento de la instalación oficial del sistema, se realice un método constante de entrenamiento con imágenes del estacionamiento, aunque debe realizarse de manera proporcional, ya que, si se entrena con demasiadas imágenes del estacionamiento, puede provocarse un sobreajuste del entrenamiento, haciendo que el sistema no reconozca bien vehículos ajenos a los que reconoce con mayor frecuencia.

El constante movimiento de los vehículos en el estacionamiento puede suponer un problema al momento de captar los espacios disponibles, ya que, la solución capta estados del estacionamiento, sin considerar los espacios como un lugar físico en el que se deben contabilizar los vehículos, sino más bien, como un gran espacio en el que se cuentan vehículos. Este error puede ser considerado como aceptable, dado el origen de la problemática, puesto que, al momento de necesitar un espacio y utilizar la aplicación, si existe una cantidad de vehículos dentro del estacionamiento que completen la totalidad de espacios disponibles, muy probablemente en un período de un par de segundos, el estacionamiento se llenará de igual manera. Además, como se mencionó anteriormente, la solución a este problema implica una mejora del sistema que escapa del alcance del proyecto.

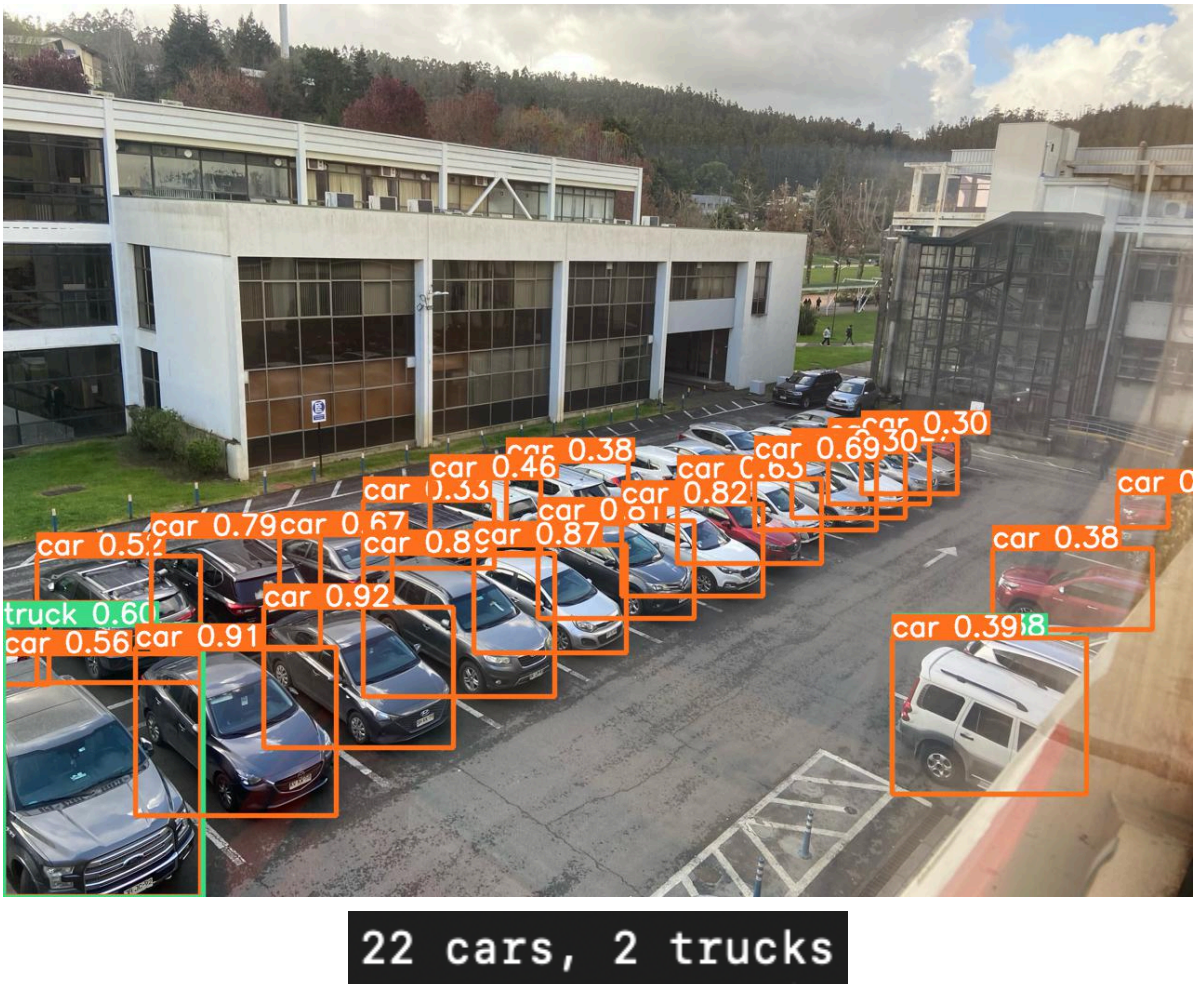

Figura 10. Resultados de la cámara 2. 24 aciertos de 37 vehículos en la imagen.

Al considerar las imágenes en un análisis individual, como se hizo al principio del proyecto, se puede llegar a otras conclusiones, en el primer caso se consigue un 90.63% de eficacia y en el segundo un 64.86% lo que también obedece a la cercanía de las cámaras con los vehículos, esto también es representado por la considerable disminución de la certeza de la predicción a medida que se alejan. Por este motivo, el componente que más impacto tiene en los resultados obtenidos es el sistema de cámaras.

5.2 Evaluación de Usabilidad

Con el objetivo de garantizar que el software generado cumpla con los requerimientos iniciales y asegurar su facilidad de uso, se generó una evaluación de usabilidad mediante pruebas de tareas (Tasks Tests [12]) para el sistema que consiste en una serie de reuniones con 24 potenciales usuarios de la aplicación para verificar su nivel de funcionalidad. Estos usuarios potenciales cumplen con la característica de ser estudiantes, personal de la universidad y/o visitantes ocasionales que sean conductores o participen de la búsqueda de estacionamientos para un vehículo común.

Para este análisis se dividió el contenido a evaluar en la aplicación, en sus 3 pantallas principales, fase inicial (Pantalla de inicio de sesión), selección de estacionamiento y consulta de espacios disponibles. Se pidió a los usuarios llevar a cabo la tarea correspondiente a cada pantalla y se obtuvieron los siguientes resultados:

Fase inicial (Pantalla de inicio de sesión): Esta pantalla estaba pensada para la inclusión de un sistema de ingreso con usuario a la aplicación. Originalmente se pensaba incluir los sistemas de ingreso propios de los otros sistemas software pertenecientes a la UdeC (INFODA, Aplicación Móvil UdeC, etc), pero no fue posible agregarlo a esta versión por lo que se muestra una pantalla de login estacionaria, en la que si se presiona el botón de “Aceptar” se ingresa automáticamente.
La recepción por parte de los usuarios fue mixta, lo que ocurría con mayor frecuencia era el instinto por agregar un mail y contraseña, algunos pusieron sus credenciales Universitarias pensando que podrían ingresar. Esta pantalla está agregada pensando en el futuro y la reacción de los usuarios es la esperada, por lo que, a pesar de no cumplir con su funcionalidad en este momento, está correctamente pensada para un usuario del estacionamiento.
La nota promedio otorgada por el grupo de usuarios fue de 6.6/10.

Selección de estacionamiento: Esta pantalla también considera una escalabilidad del sistema, agregando un carrusel de estacionamientos donde consultar los espacios disponibles. Este carrusel incluye información de los estacionamientos, como, su nombre, dirección y estado (indicado con un color que lo representa, verde si hay espacios disponibles, amarillo si quedan menos de 10 espacios y rojo si está completamente lleno).
Los usuarios revisaban distintos estacionamientos del carrusel, los que, al no estar disponibles, arrojaban resultados del mismo estacionamiento. El comentario de lo interactivo que resulta el semáforo pequeño como parte del carrusel se repitió y parece ser una buena decisión de diseño.
La nota promedio otorgada por el grupo de usuarios fue de 7.8/10.

Consulta de espacios disponibles: En esta pantalla se despliegan los resultados del llamado a la API, junto con la información de la última actualización. Sigue la misma lógica de colores presente en la pantalla de selección de estacionamiento, pero en el medio de la pantalla con el número de estacionamientos disponibles para ese estacionamiento.

Esta tarea fue la más correcta, los usuarios sólo debían acceder y hacer un update de la pantalla (deslizar hacia abajo), lo que generaba un nuevo llamado a la API, que luego desplegaba los resultados, por lo que se considera intuitiva y fácil de usar.
La nota promedio otorgada por el grupo de usuarios fue de 8.8/10.
Además de estas pruebas, se generaron pruebas unitarias para cada sistema perteneciente al software, como el funcionamiento de la API y el sistema de visión computacional. De manera equivalente, se realizaron para cada función de la aplicación al final de su desarrollo y pruebas de funcionalidad en sistemas operativos. Esto con el fin de verificar el correcto funcionamiento de la aplicación en distintos entornos.

La nota de usabilidad otorgada para la aplicación fue de 7.7/10, lo que demuestra que hay grandes posibilidades de mejora para el sistema. Estos resultados están dentro de lo esperado, sobre todo considerando la escalabilidad considerada dentro de la aplicación, esto sumado a los resultados obtenidos, pueden generar una idea clara respecto a las acciones a considerar a futuro para el sistema. La evaluación completa se puede encontrar en el Anexo 4.

5.3 Análisis de alternativas futuras

Para considerar la creación de un sistema funcional para el estacionamiento de la FIUdeC se deben contemplar diversas opciones al momento de considerar el ingreso y procesamiento de los datos. Como se demostró con anterioridad estas tareas serán resueltas por un sistema de cámaras y un sistema de punto de control respectivamente.

Alternativas de sistemas de cámaras:
El estacionamiento se encuentra rodeado de edificios donde ubicar cámaras, por lo que se pueden instalar de diversas maneras dependiendo del tipo de cámara seleccionado. De manera general se pueden encontrar los siguientes tipos de cámara:

Cámara tipo bala: Llevan ese nombre por su forma cilíndrica, son cámaras de posición fija (aunque ajustable) que tienen un largo alcance de visión. Se utilizan mayoritariamente para vigilancia en exteriores, por lo que suelen llevar protectores de sol.

Cámara PTZ: Son cámaras con un propio sistema de movimiento motorizado, son de mayor tamaño y usualmente de mayor resolución también, se utilizan en exteriores y su capacidad de moverse en casi cualquier dirección e incluso hacer zoom, permite hacer recorridos por áreas más extensas o más aún, preestablecer sectores de mayor interés para sus rutinas. Por todas sus funcionalidades, suelen ser también las más caras.

Cámara tipo domo: Llevan también su nombre por su forma de semicírculo, son muy discretas debido a la misma, además por su forma están diseñadas para poder cambiar el ángulo de visión fácilmente. Son particularmente seguras ya que suelen llevar una capa protectora que cubre al lente.

Cámaras de visión nocturna: Su principal característica es que pueden grabar imágenes en condiciones de poca luz o en la oscuridad gracias a su tecnología infrarroja. Pueden ser de

distintas formas (incluso como las mencionadas anteriormente) y sus sensores infrarrojos se encuentran rodeando la cámara principal.

Además, se utilizarán los siguientes parámetros para evaluar la factibilidad de estas alternativas:

- Resolución de la imagen captada: Esencial para un correcto reconocimiento de los elementos presentes en cada imagen. Actualmente se tiene un estándar de calidad de imagen adecuado, pero mientras mejor la calidad de origen, mejores detalles se pueden captar para diferenciar los vehículos.
- Campo de visión: Se refiere a la capacidad de la cámara de cubrir (de una u otra forma) la mayor parte del estacionamiento posible.
- Resistencia (a la intemperie): Contempla la capacidad de la cámara de resistir condiciones adversas.
- Facilidad de instalación: Qué tan fácil es instalar, configurar y mantener el sistema.
- Conectividad: La capacidad de interconectarse como elemento de un sistema, ya sea a Internet o a otro tipo de red.
- Costo: Costo monetario promedio de las cámaras de este tipo.

	Resolución	Campo de Visión	Resistencia	Facilidad de instalación	Conectividad	Costo
Bala	Alta	Amplio	Alta	Alta/Media	Varía	Variable
Domo	Alta	Depende	Muy alta	Media	Varía	Alto
PTZ	Alta	Amplio y Móvil	Alta	Profesional	Varía	Muy Alto
De visión nocturna	Media/Alta	Medio pero nocturno	Alta	Media	Varía	Bajo/Medio

Tabla 3. Análisis cualitativo de los parámetros en los posibles sistemas de cámaras.

Debido a las conclusiones a las que se llegaron con los resultados, este sistema de cámaras debe contener al menos 2 cámaras de buena calidad para su funcionamiento óptimo, de este modo, la recomendación ideal sería utilizar 2 o 3 cámaras de tipo **PTZ** en ángulos de instalación profesional en edificios aledaños al estacionamiento, si esta alternativa escapa al presupuesto se pueden reemplazar por cámaras de tipo Bala de buena calidad, sobre todo debido a su amplio campo de visión.

Alternativas de sistemas de punto de control:

Se consideraron varias alternativas para su análisis, destacan:

Raspberry Pi: El sistema sería conectado y ejecutado por una placa Raspberry Pi, lo que implica la necesidad de una instalación física del tipo mini escritorio o cápsula con acceso a internet, fuente de poder y almacenamiento. El aporte de esta placa es fundamentalmente el empaquetado del sistema completo en un dispositivo independiente, capaz de funcionar en cualquier parte.

Máquina Virtual local: Esta solución contempla la ejecución del sistema en una máquina virtual instalada en algún equipo de la universidad, conectado al sistema de cámaras y funcionando ininterrumpidamente. Esta alternativa fue consultada en la universidad y su desarrollo es factible ya que se pueden utilizar MVL que ya se utilizan internamente.

Sistemas Cloud: Esta alternativa es la más innovadora de las mencionadas, ya que este tipo de tecnología está actualmente muy a la vanguardia en empresas tecnológicas de desarrollo. Se puede considerar alguna solución en uno de los sistemas cloud del mercado (GCP, AWS, Azure, etc), ya sea con una máquina virtual de manera similar a la solución anterior, o utilizando otras herramientas disponibles en estos sistemas, esto a su vez, contempla un análisis a mayor escala.

Contenedores Docker/Kubernetes: Estos sistemas son, como su nombre lo indica, contenedores de entornos de trabajo particulares, se pueden instalar ciertos componentes en un contenedor y trabajarlos independientemente del equipo en que se ejecuten. Son dependientes de un equipo que los contenga, por lo que se tienen que ejecutar localmente o en algún servicio de nube, por lo que esta alternativa puede utilizar otras de las ya mencionadas, pero en menor medida.

Dentro de estas opciones existen parámetros clave para definir cuál opción calza de mejor manera dentro del proyecto, estos son:

- Capacidad de procesamiento: La capacidad de procesamiento necesaria para ejecutar el software de visión por computadora.
- Almacenamiento: La cantidad de almacenamiento disponible para guardar distintos tipos de datos.
- Facilidad de Uso: Qué tan fácil es instalar, configurar y mantener el sistema.
- Escalabilidad: La capacidad de mejorar alguna característica del sistema a medida que crecen las necesidades del mismo.
- Conectividad: La capacidad de interconectarse como sistema, ya sea a Internet o a otro tipo de red.
- Seguridad: La fiabilidad de las características de seguridad disponibles para proteger los datos y el sistema de amenazas.

- Costo: Los costos asociados al proyecto como la inversión inicial y los costos de mantenimiento a largo plazo.

	Capacidad de procesamiento	Almacenamiento	Facilidad de uso	Escalabilidad	Conectividad	Seguridad	Costo
Raspberry Pi	Baja	Baja	Alta	Baja	Alta	Media	Bajo
MVL	Media	Media	Baja/ Media	Media	Alta	Alta	Nulo/Bajo
Cloud	Escalable	Escalable	Alta	Alta	Alta	Alta	Escalable
Contenedores	Escalable	Escalable	Media/ Alta	Alta	Alta	Alta	Nulo/Bajo

Tabla 4. Análisis cualitativo de los parámetros en los posibles puntos de control.

Dado que el uso del sistema está pensado para contemplar períodos de gran tráfico y por ende gran cantidad de solicitudes, la alternativa **Cloud**, al poder escalar las capacidades del sistema y tener servicios dedicados exclusivamente a cada parámetro estudiado, parece la mejor opción para aplicar al proyecto a futuro.

6. Conclusiones

Se ha generado una versión preliminar de la solución que cumple con los objetivos inicialmente planteados. Ya existe un sistema capaz de reconocer los vehículos en el espacio definido como objetivo, y se ha creado una aplicación móvil que despliega la información adquirida para verificar la disponibilidad de espacios en el estacionamiento de la Facultad de Ingeniería de la Universidad de Concepción.

Para desarrollar esta solución, se utilizaron diversas herramientas de software, como Flutter para disponibilizar la aplicación en múltiples plataformas, Python y varias librerías para crear el sistema de API, y la librería YOLO para la red de visión computacional.

Esta versión de la aplicación está lejos de ser el sistema definitivo para resolver la problemática inicial, principalmente debido al sistema de cámaras utilizado. Al no ser un sistema fijo en el lugar objetivo, no permite el uso del sistema en tiempo real. Además, la ejecución de la red de visión computacional depende de un equipo local, lo que impide su funcionamiento independiente. Ante estas dificultades, se han planteado posibles soluciones y sugerencias para un desarrollo futuro.

El desarrollo de la aplicación es la parte del proyecto que requiere menos cambios posteriores, principalmente la corrección de errores que puedan surgir y la integración con otros sistemas que se puedan implementar.

Como se mencionó en el análisis de alternativas, los puntos clave para la mejora se centran en el sistema de cámaras y la consolidación del punto de control. Para crear una solución duradera y escalable a otros estacionamientos del campus o incluso para su comercialización, se requiere de la solidez que proporcionan estas alternativas exploradas.

A pesar de la interrupción del apoyo al proyecto a fines de agosto, este continúa siendo parte de la Fábrica de Software y se espera su implementación completa en el futuro. Personalmente, espero que se consideren al menos parcialmente mis recomendaciones y ver el proyecto funcionar correctamente, no solo en este estacionamiento, sino también en otros lugares del campus.

7. Bibliografía

[1] Rose, K., Eldridge, S., & Chapin, L. (2015). The internet of things: An overview. *The internet society (ISOC)*, *80*, 1-50. (IOT)

[2] Facultad de Ingeniería UdeC. (Agosto 2022). CISA UdeC busca potenciar el desarrollo de software. https://fi.udec.cl/cisa-udec-busca-potenciar-el-desarrollo-de-software/

[3] Atlassian. (s.f.). A guide to Scrum. https://www.atlassian.com/agile/scrum

[4] Investopedia. (2024). Smart Home. https://www.investopedia.com/terms/s/smart-home.asp

[5] IBM. (s.f.). Mobile Application Development. IBM. https://www.ibm.com/topics/mobile-application-development

[6] StatCounter. (Noviembre 2023). OS Market Share Worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-202311-202311-bar

[7] Tashildar, A., Shah, N., Gala, R., Giri, T., & Chavhan, P. (2020). Application development using flutter. *International Research Journal of Modernization in Engineering Technology and Science*, *2*(8), 1262-1266.

[8] Bay, H., Tuytelaars, T., & Van Gool, L. (2006). Surf: Speeded up robust features. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9* (pp. 404-417). Springer Berlin Heidelberg.

[9] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[10] Ultralytics. (2020). YOLOv5. PyTorch. https://pytorch.org/hub/ultralytics_yolov5/

[11] Baran, J.; Miklis, A.; Żabińska, I. Research towards Sustainable Parking Solutions. Multidiscip. Asp. Prod. Eng. 2021, 4, 376–386.

[12] Yesmin, S. y Atikuzzaman, M. (2023). Usability testing of a website through different devices: a task-based approach in a public university setting in Bangladesh. Information Discovery and Delivery

8. Anexos

8.1 Código utilizado

8.1.1 Backend de Visión Computacional

Función principal:

```
def count_vehicles(img_path):
    # Cargar el modelo de yolo
    model = attempt_load('yolov5/Backend/yolov5s.pt')
    #Se definen las id de los vehiculos a detectar
    car_id = 2
    truck_id = 7

    dataset = LoadImages(img_path, img_size=640)
    path, img, im0s, _ , s= next(iter(dataset))
    img = torch.from_numpy(img).to(torch.device('cpu'))
    img = img.float()
    img /= 255.0
    if img.ndimension() == 3:
        img = img.unsqueeze(0)
    pred = model(img)[0]
    pred = non_max_suppression(pred, 0.25, 0.45, classes=[car_id, truck_id], agnostic=None)
```

```
#Se cuentan la cantidad de autos detectados al igual que las camionetas
car_count = 0
truck_count = 0

for det in pred:
    if det is not None and len(det):
        for *xyxy, conf, cls in det:
            if cls.item() == car_id:
                car_count += 1
            elif cls.item() == truck_id:
                truck_count += 1
#Se retorna la suma de los autos y camionetas detectados
return car_count + truck_count
```

Función de extracción de imágenes de resultado:

```python
def count_vehicles_2(img_path):
    #Función de generación de resultados
    model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)
    imgs = [img_path]
    results = model(imgs)
    results.print()
```

8.1.2 API

```python
class ConnectionManager:
    def __init__(self):
        self.active_connections: List[WebSocket] = []

    async def connect(self, websocket: WebSocket):
        await websocket.accept()
        self.active_connections.append(websocket)

    def disconnect(self, websocket: WebSocket):
        self.active_connections.remove(websocket)

    async def send_personal_message(self, message: str, websocket: WebSocket):
        await websocket.send_text(message)

    async def broadcast(self, message: str):
        for connection in self.active_connections:
            await connection.send_text(message)

manager = ConnectionManager()
```

```python
@app.websocket(Router.WEBOCKET)
async def websocket_endpoint(websocket: WebSocket):
    global TOTAL_PARKING_SPOTS
    global USED_SPOTS
    await manager.connect(websocket)

    try:
        while True:
            await asyncio.sleep(2)
            await websocket.send_text(f"Estacionamientos disponibles: {TOTAL_PARKING_SPOTS - USED_SPOTS}")
    except WebSocketDisconnect:
        manager.disconnect(websocket)
        await manager.broadcast(f"Cliente ha cerrado la conexion")


@app.get(Router.GET_MAIN_API_LANDING)
def ping():
    return {"Ping": "Pong"}
```

8.1.3 Llamado a sistema de cámaras

```python
@app.get(Router.GET_AVAILABLE_SPACES)
async def available_spaces():
    global USED_SPOTS
    global img_counter
    USED_SPOTS_1= 0
    USED_SPOTS_2 = 0

    camara_0 = cv2.VideoCapture(0)
    print("2 cam open")

    ret0,frame0 = camara_0.read()

    if ret0 == True:
        #primera cam
        cv2.imwrite('detectar/IMG_src0_'+str(img_counter)+'.jpeg',frame0)
        count='detectar/IMG_src0_'+str(img_counter)+'.jpeg'
        USED_SPOTS_1 = count_vehicles(count)
        print(USED_SPOTS_1)
    camara_0.release()
```

```python
    camara_1 = cv2.VideoCapture(1)
    ret1,frame1 = camara_1.read()
    if ret1 == True:
        #segunda cam
        cv2.imwrite('detectar/IMG_src1_'+str(img_counter)+'.jpeg',frame1)
        count='detectar/IMG_src1_'+str(img_counter)+'.jpeg'
        USED_SPOTS_2 = count_vehicles(count)
        print(USED_SPOTS_2)
    camara_1.release()
    img_counter=img_counter+1
    spots = USED_SPOTS_1 + USED_SPOTS_2
    USED_SPOTS = spots


    return {"cars_parked": USED_SPOTS, "available_spaces": TOTAL_PARKING_SPOTS - USED_SPOTS}
```

8.2 Especificaciones de Hardware

PC

- Macbook Pro 2020
  Apple M1 chip
  8gb ram
  256GB SSD

Cámaras

- Cámara IP de exteriores
  5 MP
  1920*1080p @30fps
  Wifi
  IP66
- Webcam Logitech Streamcam 1080p
  1920*1080p @60fps
  USB C
- Webcam LinkOn 1080p
  1920*1080p @30fps
  USB A

8.3 Demostración de uso del sistema

A continuación se presenta un vídeo demostrativo del sistema en el que al actualizar la aplicación luego de mostrar un vehículo, cambia el número desplegado por la app.

https://drive.google.com/file/d/1uBL_9rGq8VODAMMObIuLmDcnlF747yOO/view?usp=sharing

8.4 Detalle de Evaluación de Usabilidad

Usuario	Nota task 1	Nota task 2	Nota task 3	Promedio
Elsa Aqueveque N.	8.8	9.2	9.5	9.17
Laura San Martin M.	8	9	10	9
Amaro San Martín M.	7.5	8	9.5	8.34
Karina Mora S.	7.8	9	8.5	8.44
Jorge Navarrete V.	7.4	7.5	8	7.63
Sergio Silvestre D.	6.6	7.8	8.8	7.73
Matias Poblete F.	5.3	7.4	9.2	7.3
Camila Manriquez C.	7.2	8.4	9.1	8.23
Camilo Ruiz B.	5	7.4	8.1	6.83
Jonathan Venegas C.	5.6	7.1	8.4	7.03
Omar Hernández M.	6.1	7.2	8.7	7.33
Pablo Correa V.	7.5	6.5	8.2	7.4
Jaime San Martín A.	6.5	7.5	9.5	7.83
Vicente Herrera J.	5.2	8.1	8.7	7.33
Rubén Smith H.	5.3	7.8	8.8	7.3
Maximiliano Vergara R.	6	8.2	9	7.73
Valentina Castillo M.	7.2	8.7	8.8	8.23
Pedro Díaz G.	7	8.5	9	8.17
Juan Manríquez M.	5.1	7.5	8.5	7.03
Sandra Lema G.	7.2	7.4	8.2	7.6
Nicolás Garrido G.	5.6	6.4	7.8	6.6
Jorge Villegas I.	6	7.8	8.4	7.4
Esteban Araya V.	7.3	8.2	9.3	8.27
David Leon A.	6.8	7.3	8.5	7.53
Promedios	**6.58**	**7.8**	**8.77**	**7.7**

8.5 Repositorio del proyecto

https://github.com/antsanmora/EstacionamientosUdeC