

Universidad de La Habana  
Facultad de Matemática y Computación



# Sistema de Control de Personal e Inventarios

Autor:

**José Alejandro Solís Fernández**

Tutores:

**Roberto Marti Cedeño**

**Alexis Massó Muñoz**

Trabajo de Diploma  
presentado en opción al título de  
Licenciado en Ciencia de la Computación

Fecha

[github.com/username/repo](https://github.com/username/repo)

A mis padres por guiarme tantos años. Cualquier logro mío es también suyo.

# Agradecimientos

A toda mi familia, en especial a mis padres por dedicarse más a nosotros que a ellos mismos. A mis hermanas por darme ejemplo siempre y cuidarme tanto. A mis abuelos Papo, Yeyi y Cuca por el inmenso cariño y apoyo. A mi abuelo Papi por transmitirme su amor por la matemática y la pelota, por lo orgulloso que se sentiría si hoy pudiera verme. A Adam y Lucía, que aún no son conscientes de cuánta luz le dan a la familia.

A los tantos amigos que me ha regalado la vida. A los siete que se han mantenido por tantos años y se han vuelto mis hermanos. A los mejores compañeros de carrera que pudiera pedir y toda la amistad que nos queda por delante.

A los muchos profesores que me han inspirado a lo largo de todos estos años. A mi tutor Roberto por su atención y ayuda.

A todos los que no están en este texto pero han moldeado mi vida y me han hecho quien soy.

# Opinión del tutor

Opiniones de los tutores

# Resumen

La falta de automatización en procesos triviales llevados a cabo en la mayor parte de los distintos departamentos de la Universidad de La Habana, motiva un plan de acción que haga uso de la digitalización y las capacidades computacionales en interés de una administración más sencilla, fácil y sostenible.

En consecuencia, se lleva a cabo un estudio del estado del arte de algunos de estos procesos, manteniendo el enfoque en el control de acceso de personal a áreas restringidas y el control de inventario. Además, se hace una propuesta de solución general, reutilizable y extensible a fin de alcanzar más resultados de los que pueda lograr el presente trabajo. También, se hace una implementación y una muestra del flujo de funcionamiento de las mismas.

# Abstract

The lack of automation in trivial processes carried out in most of the different departments of the University of Havana, motivates an action plan that makes use of digitization and computational capacities in interest of a simpler, easier and more sustainable administration.

Consequently, a study of the state of the art of some of these processes is carried out, maintaining the focus on personnel access control to restricted areas and inventory control. In addition, a general, reusable and extensible solution proposal is made in order to achieve more results than the present work can do. Also, an implementation and a sample of the flow of operation of the same is made.

# Índice general

Introducción	1
1. Estado del Arte	3
2. Propuesta	12
3. Detalles de Implementación y Experimentos	17
Conclusiones	28
Recomendaciones	29
Bibliografía	30

# Índice de figuras

2.1. Esquema General . . . . .	13
2.2. Interacción Usuario-API . . . . .	14
2.3. Esquema en 3 etapas . . . . .	15
2.4. Flujo de generación de código QR . . . . .	16
3.1. Vista de CRUD del personal . . . . .	23
3.2. Agregar Persona . . . . .	24
3.3. Inventario . . . . .	25
3.4. Añadir Objeto al inventario . . . . .	25
3.5. Editar Objeto . . . . .	26
3.6. Eliminar Objeto . . . . .	26
3.7. Endpoint QR . . . . .	27
3.8. Código QR del objeto . . . . .	27



# Ejemplos de código

3.1. Engine . . . . .	19
3.2. Declarative . . . . .	19
3.3. Personal . . . . .	19
3.4. Inventario . . . . .	20
3.5. CRUD para el personal autorizado . . . . .	21
3.6. Códigos QR en Personal . . . . .	21
3.7. CRUD para el inventario . . . . .	22
3.8. Códigos QR en Inventario . . . . .	23

# Introducción

Actualmente, todo departamento de la Universidad de La Habana está plagado de operaciones de uso diario, monótonas y carentes de un sistema digitalizado que las maneje de forma automática y eficiente.

Por ejemplo, por cuestiones de seguridad se debe controlar el acceso al campus universitario. Diariamente, por cada individuo que desee ingresar al área, se debe revisar en extensas listas de estudiantes, profesores y demás trabajadores de la universidad que su nombre aparezca. Esta práctica depende totalmente del factor humano e implica la ejecución de una tarea constante y repetitiva, lo cual probablemente resulte en negligencias laborales hasta cierto grado entendibles.

Por otro lado, el mantenimiento y el control de los inventarios puede realzar su rendimiento considerablemente si fuese llevado a cabo a través de un procedimiento maquinal. Actualmente, el sostenimiento de esta tarea recae en montones de hojas de papel que pierden su validez tan rápido como un elemento cambie dentro del recinto de depósito.

## Motivación

El correcto control de estos registros peligra diariamente, sometiéndose a la posibilidad de verse afectado por el habitual y comprensible error humano. Resulta motivante participar en la creación de una estructura que soporte los cimientos de futuras proyecciones que continúen el proceso de Transformación Digital que está en vigor en la Universidad de La Habana [1].

## Antecedentes

El Departamento del Nodo Central de la Universidad de La Habana tiene actualmente, entre sus múltiples tareas, la labor de unificar bajo un mismo régimen los sistemas de autenticación central, de incidencias y reportes, de gestión de dominio, etc. De este modo, explotando la necesidad de desarrollar tal sistema unificador,

se organizan proyecciones de ampliación en sus funcionalidades, como es el caso del control de personal e inventarios.

## **Objetivos Generales**

- Automatización de los sistemas de organización y gestiones del departamento.

## **Objetivos Específicos**

- Automatización de control de acceso del personal.
- Lectura de código QR del documento de identidad en la entrada y salida.
- Implementación de un sistema de gestión de inventarios.
- Nomenclar, identificar y documentar cada elemento activo.

# Capítulo 1

## Estado del Arte

En este capítulo se hace un análisis del estado del arte de los procedimientos similares al cual se le hace frente. Se estudian sus particularidades y, teniendo en cuenta sus ventajas y desventajas, se ofrecen razones para incluir su utilización en la propuesta de solución.

Se recorren los métodos más popularmente adoptados en los últimos años para la gestión de inventario y el control de acceso y seguridad. Además, se sondea el estado de las APIs (application programming interface o interfaz para programas de aplicación en español) para considerar su empleo en el campo bajo investigación, de mano con una observación de los lenguajes de programación, *frameworks* y sistemas de gestión de bases de datos (SGBD) que puedan favorecer su desarrollo.

## Gestión de inventario

Generalmente, son los almacenes las áreas que más requieren un sistema de gestión de inventarios automatizado ya que buscar manualmente en todos los espacios de depósito disponibles requiere mucho esfuerzo. Para evitarlo se mantiene la información detallada del producto digitalmente, y para los recintos de almacenamiento de mayor extensión se puede ubicar en qué zona está presente cada producto deseado. En la globalización de las industrias, el sistema de gestión de inventario de almacenes tiene su propia importancia debido a las ganancias que genera.

Los grandes almacenes, a pesar de la variedad de tecnologías de comunicación remota existentes, se benefician más con la Identificación por Radiofrecuencia o RFID (por *Radio Frequency Identification*) [2]. Este es un sistema de almacenamiento de datos en etiquetas, tarjetas o transpondedores. El objetivo principal es transmitir datos mediante el *tag* (dispositivo portátil) que es leído por el receptor RFID. Los datos transmitidos dependen de cada aplicación, pero pueden pasar información sobre la localización del producto u otros datos más específicos.

En los sistemas inteligentes, hay un crecimiento lineal en el concepto de localización, porque esta está jugando un papel crucial en la vida contemporánea [3]. Los mecanismos de identificación se basan en la tecnología AIDC (identificación automática y captura de datos o en inglés *automatic identification and data capture*). La tecnología AIDC tradicional es la de código de barras, que es operada por *scanners* ópticos para leer etiquetas. Los sistemas AIDC que utilizan tecnologías portadoras de datos se han implementado con éxito en muchas grandes organizaciones multinacionales [4].

Mientras los códigos de barras son un gran avance sobre las etiquetas de texto normales porque el personal ya no es esencial para ingresar datos manualmente en el sistema, es reemplazada por la tecnología RFID porque los lectores de códigos de barras tienen un alto costo y la seguridad es menor. Si la etiqueta se daña, el *scanner* no puede leer los datos [5]. Las etiquetas RFID tienen más capacidad de almacenamiento de datos que el código de barras. Basado en muchas investigaciones, la tecnología RFID se mantiene significativa y eficiente en el sistema de identificación. La tecnología RFID creó un impacto en el dominio de los almacenes porque elimina los riesgos en las industrias, se mejora la eficiencia y los productos se pueden rastrear fácilmente [6].

En espacios de menor extensión, donde no sea necesaria la implementación de un sistema de localización, se ha generalizado el uso de códigos QR. Estos tienen un tamaño pequeño, tanto como 1 cm por 1 cm y se pueden escanear para recuperar grandes cantidades de datos (más de 100 veces la información que puede contener un código de barras). Los códigos QR dinámicos ayudan a acelerar el proceso de escaneo, ya que contienen URLs breves y también rastrean datos de ubicación en tiempo real, acelerando así el proceso de inventario para maximizar la eficiencia. Ayudan en el proceso de mantener un inventario preciso y también reducen el tiempo dedicado a la entrada manual de datos y mejoran la precisión de los registros en cuestión [7].

Con la creciente popularidad de los códigos QR, se pueden leer fácilmente con la mayoría de las cámaras de los teléfonos inteligentes de forma nativa, sin necesidad de ninguna aplicación de terceros. Los empleados de almacenes e instalaciones de almacenamiento prefieren usar teléfonos inteligentes para escanear y mantener el inventario, ya que son más convenientes y fáciles de usar [8].

Los códigos QR son fáciles de escanear, incluso si están ligeramente desalineados, ya que las capacidades actuales de escaneo son lo suficientemente potentes como para escanearlos desde diferentes ángulos. Pueden cambiar su color, agregar una imagen de fondo o un logotipo para identificar el producto, entre otras opciones de personalización. Para las empresas que ofrecen diferentes tipos de productos a diario, como grandes cadenas minoristas, tiendas de ropa y cadenas de supermercados, tener códigos QR que se puedan identificar fácilmente ayuda a distinguir los productos según sus categorías. Esto ayuda al personal del almacén a etiquetar y organizar sus productos

incluso antes de comenzar a escanearlos [9].

Un código QR puede funcionar incluso si está dañado al 30 %, una necesidad básica para los productos que se envían y procesan a gran escala lo que los convierte en una opción más confiable para el seguimiento de inventario. Su uso ayuda a rastrear el inventario en tiempo real en función de la ubicación GPS del código, la hora del último escaneo e incluso el sistema operativo utilizado para recuperar datos granulares sobre los productos. También se pueden compartir con los clientes para rastrear su envío y brindar más transparencia con respecto a los procesos de transportación y logística [10].

## Control de acceso

Un sistema de control de acceso tiene tres funciones principales: la autenticación que permite la identificación de las personas o vehículos que solicitan acceder a una zona concreta; la autorización que gracias al *software* del sistema realiza las comprobaciones y envía la orden de abrir o no un acceso; y la trazabilidad que facilita la obtención de listados de las personas presentes en un recinto [11].

En base a esto, se pueden distinguir los tipos de control de acceso según la conexión que necesiten para cumplir con sus funciones.

Pueden ser sistemas de acceso autónomos, que no requieren ningún tipo de conectividad ya que los propios terminales disponen de memoria para la gestión de los usuarios. Se trata de sistemas de una seguridad baja y de una capacidad muy limitada. Sin embargo, sistemas autónomos que se apoyan en el uso de inteligencia artificial van incrementando su integración en la vida cotidiana a través de las más recientes innovaciones técnicas. La mayor distinción e influencia de estos sistemas es su capacidad para tomar decisiones sin necesidad de ayuda humana en el proceso [12].

Además, pueden ser sistemas de acceso en red, los cuales utilizan herramientas como los softwares de control de acceso y ofrecen un alto nivel de seguridad. Se pueden controlar a la vez tantas zonas diferentes del recinto como se desee, acotar los accesos por horarios y permisos, y además cuentan con la ventaja de que pueden englobar otras soluciones importantes como el control horario, de visitas, o incluso planes de evacuación y emergencias [13].

También se pueden diferenciar según el sistema de identificación que utilicen, pueden ser de proximidad, biométricos o de reconocimiento de matrícula.

## Sistemas Biométricos

La biometría es un término general utilizado para describir una característica o un proceso. Como característica, es un rasgo biológico (anatómico y fisiológico) medible que se puede utilizar para el reconocimiento automatizado. Como proceso,

es un método automatizado para reconocer a un individuo basado en características biológicas y de comportamiento medibles. Incluye el estudio de datos biológicos y la autenticación basada en biometría, que es una aplicación que utiliza rasgos personales específicos para el control de acceso [14]. Este método identificación es preferible a los métodos tradicionales que involucran contraseñas y números PIN por dos razones principales. Primeramente, la persona a identificar debe estar físicamente presente en el punto de identificación, y en segundo lugar evita la necesidad de recordar una contraseña o llevar un *token* consigo.

Aunque la tecnología biométrica ha existido por un tiempo, los avances modernos en miniaturización, junto con grandes reducciones en el costo, han hecho que la biometría esté fácilmente disponible y sea asequible para propietarios de pequeñas empresas, corporaciones más grandes y agencias del sector público por igual. Muchos de sus defensores creen que incrementará la adopción empresarial y gubernamental de métodos sistemas biométricos, no solo debido a las muchas nuevas tecnologías prometedoras, sino también porque las organizaciones buscan métodos innovadores de gestión de identidad y control de acceso [15].

Sin embargo pueden presentar algunos problemas. Algunos usuarios se pueden mostrar escépticos ante la idea de que sus datos anatómicos sea procesados por un algoritmo. Una toma de datos parcialmente incorrecta puede ocasionar inconvenientes a la hora de validar la identidad de los estudiantes y trabajadores. Además, requiere un arduo trabajo de implementación y demanda tecnologías muy específicas, y a integración con sistemas de infraestructura puede suponer retos para garantizar la disponibilidad y accesibilidad de los datos. También puede incurrir en experiencias negativas con la autenticación multifactor en caso de que un usuario sea víctima de un *hackeo* [16].

## Sistemas de proximidad

Estos sistemas permiten la utilización de tarjetas u otros objetos que al acercarlos al terminal inicia la autenticación. En este tipo de control se destaca la tecnología RFID, que además de ofrecer una alta seguridad, es precisa, fiable y cuenta con una gran capacidad de almacenamiento de datos.

Hoy en día, las tarjetas inteligentes sin necesidad de contacto se utilizan para proporcionar autenticación y control de acceso físico en una amplia variedad de aplicaciones. Investigaciones anteriores han demostrado la vulnerabilidad de las tarjetas inteligentes sin contacto para los ataques de retransmisión [17].

Por ejemplo, un atacante puede transmitir la comunicación entre el lector de tarjetas y la tarjeta inteligente para obtener permiso a un recinto restringido. Para resolver este problema, las tarjetas inteligentes se han ido mejorando con verificación de proximidad segura, es decir, límite de distancia, lo que permite que el lector de

tarjetas y la tarjeta verifiquen su distancia mutua.

Sin embargo, las tecnologías existentes no admiten la implementación de límites de distancia en dichos sistemas: NFC(*Near Field Communication* o comunicación de campo cercano) no puede proporcionar una resolución de distancia suficiente, y la complejidad del *hardware* de los radios de límite de distancia propuestas, por ejemplo, basados en UWB(banda ultra ancha o en inglés *ultra-wideband*) impide su uso en tarjetas inteligentes sin contacto [18].

## Sistemas de reconocimiento de matrícula o *tag*

Controlan el acceso mediante la identificación de la persona, del vehículo o la combinación de ambas.

Se han propuesto sistemas con circuitos de control que chequean las entradas de una puerta y un *software* que monitorea, muestra y registra la información del usuario y el estado del sistema. Existen *softwares* capaces de leer y mostrar el número de tarjeta de usuario, el nombre de usuario, la hora de llegada y la cantidad de veces que se usó la tarjeta y guardar todos estos datos [19].

También se ha puesto en práctica el uso de modelos arquitectónicos del dispositivo de acceso que permita la lectura de una imagen de código QR utilizando los datos de usuario necesarios para poder otorgarle los permisos requeridos[20].

## API

Históricamente, las API han existido desde la llegada de las computadoras personales. Las API existían principalmente para el intercambio entre dos o más programas. Sin embargo, a partir del surgimiento de las API en la web, alrededor del año 2000, estas han recibido un interés considerable por parte de profesionales e investigadores hasta el punto de que algunos expertos argumentan que ahora vivimos en la economía API. Esta posición está respaldada por el hecho de que el ser humano se ha interconectado, unos con otros, como nunca antes; y las API impulsan principalmente esta interconexión de personas, aplicaciones y sistemas.

Como tal, las API se están convirtiendo en la fibra del ecosistema digital que busca interconectar negocios y economías para crear valor y desarrollar más capacidades. El mercado de las aplicaciones móviles, una de las áreas de mayor crecimiento en tecnología de la información, hace mucho uso de las API ya que los desarrolladores de aplicaciones móviles confían principalmente en ellas para proporcionar aplicaciones fiables e interoperables.

Una API puede garantizar extensibilidad para objetivos que se escapen del alcance del presente trabajo, y suplir de manera efectiva las nuevas necesidades que puedan



surgir. A la hora de poner en práctica las tareas que se persiguen, una API puede ayudar a mejorar la revisión de código permitiendo separar la comprobación de las componentes del Frontend y la comprobación de las componentes de su contraparte, el Backend.

Hay muchos tipos de API, y varias maneras de categorizarlas. Una de estas maneras es el estilo de su arquitectura.

Entre los estilos arquitectónicos fundamentales se encuentra REST. Sus siglas son por transferencia de estado representacional( REpresentational State Transfer). Las API REST se basan en una estructura cliente-servidor manejando las solicitudes a través de HTTP, y mantienen una comunicación sin estados, lo que significa que no se almacena información del cliente entre las solicitudes de obtención o *request* y cada una está separada y desconectada. Tienen una interfaz uniforme entre las componentes para que la información se transfiera en una forma estándar: los recursos solicitados son identificables y separados de las representaciones enviadas al cliente y pueden ser manipulados por el cliente a través de la representación que reciben porque esta contiene suficiente información para hacerlo y para que el cliente pueda describir cómo la debe procesar [21].

También se puede destacar el uso de *webhooks*. Los *webhooks* se basan en eventos y son, en pocas palabras, una solicitud HTTP desencadenada por un evento en un sistema de origen y enviada a un sistema de destino. Tienen un mensaje, o carga útil con información, y se envían a una URL única. Ofrecen una forma sencilla de hacer posible el intercambio de información en tiempo real entre plataformas en línea. Además se pueden integrar a otro tipo de arquitecturas como REST [22].

En la actualidad se utiliza bastante SOAP, acrónimo de Protocolo Simple de Acceso a Objetos o Simple Object Access Protocol. Estas APIs están más estructuradas y formalizadas que otras, son fiables y confiables, pero pueden ser más lentas. Utilizan un protocolo de mensajería basado en XML que incluye las etiquetas de **Envelope**, **Header** y **Body** según lo requiera el *endpoint*. Uno de sus puntos fuertes es que también puede aprovechar el protocolo de transporte de correo simple (SMTP), el protocolo de control de transmisión (TCP) y el protocolo de datos de usuario (UDP) para pasar mensajes de un lado a otro. Esto permite una mayor flexibilidad en lo que respecta a mudanza de datos [23].

También se encuentra GraphQL, nombre acuñado por Graph Query Language. Graph Query Language define cómo una API solicita información a otra y, en lugar de depender de cómo el servidor define el punto final, una consulta GraphQL puede solicitar una información específica. Fue creado originalmente por Facebook como una herramienta interna en 2012, pero lo lanzaron públicamente en 2015 como un lenguaje de código abierto para las APIs [24].

Otro tipo de API conocido son los WebSockets. Estas APIs se basan en el protocolo de comunicaciones informáticas WebSocket, que es un canal de comunicación *full-*

*duplex* o sea que se puede transmitir en ambas direcciones en un portador de señal al mismo tiempo, a través de una única conexión TCP. Proporcionan una forma estándar para que los servidores envíen información y datos a los clientes, incluso cuando el cliente no solicita datos. También permiten que los datos se comuniquen entre clientes y servidores, manteniendo las conexiones abiertas [25].

## Base de datos

En la actualidad hay un gran número de SGBDs, y cada uno viene con sus propias especialidades y funcionalidades únicas, además de las principales y fundamentales para administrar datos (obtener, guardar, actualizar, optimizar). Debido al amplio espectro de posibilidades, existe confusión en cuál sería la opción más popularmente aceptada para cada problema en término de capacidad de sus prestaciones o cuál tiene las mejores herramientas de modelado.

Entre los SGBD más populares se encuentra MySQL que Es un SGBD relacional de código libre y multi-modelo. Una herramienta confiable y diseñada para mejorar la seguridad y la escalabilidad de las bases de datos. Sus funcionalidades garantizan un procesamiento de datos a alta velocidad, herramientas para la recuperación de datos, soporta arquitecturas cliente-servidor, y una amplia y activa comunidad que, a la par del apoyo técnico, provee a los usuarios y desarrolladores de toda la ayuda necesaria [26].

Otro SGBD objeto-relacional de gran importancia es Oracle. Tiene bases de datos grandes, consume poca memoria y procesa los datos rápidamente. Usa técnicas de encriptación, data masking y actividades de monitoreo para salvaguardar la base de datos. Es mayormente usada para procesar transacciones online [27].

Microsoft SQL Server, es también un sistema relacional efectivo y comercial. Provee de una eficiente administración de la carga de trabajo y una misma base de datos puede ser utilizada por varios usuarios. Es seguro y consistente. Además, configurar una nueva base de datos es fácil y se pueden crear tablas sin sintaxis [28].

Otro SGBD avanzado y de código libre es PostgreSQL. Su funcionalidad clave es la importación y exportación de datos. Cuenta con una amplia gama de plug-ins para incrementar sus capacidades funcionales [29].

SQLite es un sistema integrado y de código abierto, diseñado alrededor del año 2000. Provee bases de datos livianas, sin configuración, sin requisitos de un servidor o instalación. Se puede utilizar con todos los lenguajes de programación sin ningún problema de compatibilidad y a pesar de su simplicidad, es uno de los más populares en el ámbito internacional actual [30].

También se tiene presente MongoDB, un sistema NoSQL multiplataforma y de código libre. Es mayormente preferido por desarrolladores que trabajan con grandes volúmenes de datos almacenados. Incluye entre sus prestaciones la capacidad de

distribuir un solo conjunto de datos en múltiples bases de datos [31].

Entre otros muchos sistemas de gestión de bases de datos que también presentan diversas prestaciones interesantes.

## Lenguaje y Plataforma

Cualquier lenguaje de programación se puede usar para desarrollar una API. Sin embargo, algunos pueden ser mejores y más eficientes de usar que otros.

Entre los criterios que se pueden destacar para la elección de uno u otro lenguaje está la comodidad del desarrollador con el lenguaje. Mientras más se conozca un lenguaje más se limita la cantidad de cosas desconocidas y nuevos conceptos para poder concentrarse en escribir buen código. También se puede considerar la variedad de opciones diferentes que tenga para bibliotecas y plataformas que puedan ayudar en el desarrollo de la API. Además, puede ser muy importante cuando se intenta ganar tiempo el soporte que puedan o no tener las plataformas del lenguaje, a la par del tamaño y grado de actividad de su comunidad.

Dicho esto, existen varios lenguajes que se destacan por su uso en el desarrollo de APIs.

Entre los principales se pueden destacar Python. Este lenguaje tiene una sintaxis simple similar al idioma inglés que hace que el código sea comprensible sin sacrificar productividad. Con necesidad de menos líneas de código, este lenguaje de programación se hace ideal para producir rápidamente productos mínimos viables. Cuenta con una voluminosa extensión de bibliotecas y paquetes que pueden ser de utilidad. Además, tiene diversas herramientas para garantizar la escalabilidad del producto [32].

Otro útil lenguaje es Java. El código escrito en Java se puede ejecutar en cualquier plataforma, independientemente de dónde se haya creado originalmente. Tal independencia de plataforma la proporciona el entorno donde se ejecuta el código Java: Java Virtual Machine. En este sentido, Java se diferencia de otros lenguajes que requieren compiladores. Se dice que es más seguro que cualquier otro lenguaje de programación. Su seguridad se obtiene de un gran conjunto de APIs, herramientas e implementaciones de mecanismos, protocolos y algoritmos de seguridad de uso común [33].

Es popular también la combinación de Javascript con Node.js. Junto con HTML y CSS, JavaScript es una de las tecnologías centrales de la World Wide Web. Se usaba principalmente para el desarrollo en Frontend hasta el lanzamiento del sistema de ejecución Node.js. Elegir Node.js es una buena decisión para desarrollar juegos, chats, servicios de transmisión o productos de Internet de las cosas. Se adapta bien a los productos basados en microservicios, lográndose mediante características tales

como el soporte de una entrada-salida sin bloqueo y una cantidad significativa de conexiones [34].

El pre-procesador de hipertexto PHP es un lenguaje de secuencias de comandos de código abierto ampliamente utilizado. PHP es capaz de generar contenido de página dinámico. Puede recopilar datos de formulario y también puede enviar y recibir cookies. Puede agregar, eliminar y modificar datos en la base de datos directamente.. Además, se puede utilizar para controlar el acceso de los usuarios y cifrar los datos [35].

Ruby es un lenguaje de programación interpretado y orientado a objetos. Se enfoca en objetos y datos, en lugar de acciones individuales y una lógica específica detrás de ellas. Estos objetos luego se tratan como unidades dentro de un programa que interactúan entre sí. Aunque el lenguaje está perdiendo popularidad recientemente, su *framework* principal, Rails, está funcionando bien. Las esferas en las que Ruby brilla al máximo incluyen las redes sociales y el comercio electrónico, dado que tiene la capacidad de admitir aplicaciones con mucho tráfico [36].

También se puede señalar a Golang, un lenguaje de programación de código abierto joven y ambicioso proporcionado por Google. El diseño del mismo se inició en 2007, mientras que la primera versión estable se lanzó en 2011 y desde el principio, el lenguaje ha mantenido la atención de los desarrolladores como un proyecto prometedor. Mientras que otros lenguajes de un intérprete para ejecutarse, Golang se traduce directamente a formatos que un procesador entiende [37].

## Capítulo 2

### Propuesta

A pesar de que los sistemas de identificación por radiofrecuencia sean los más generalizados a utilizar en el control de inventario de los grandes almacenes alrededor del mundo, y que complejos sistemas de autenticación y autorización con mecanismos biométricos y diversas e innovadoras tecnologías sean los aplicados en los más rigurosos controles de acceso, son ideas que eluden no solo las capacidades de la Universidad de La Habana, sino también las necesidades.

Los almacenes de inventario de los distintos departamentos son pequeños locales, donde si bien es cierta la falta de organización brindada por un procedimiento automatizado, las labores de localización de los objetos no necesitan un sistema tan complejo. Por otro lado, la mayor parte de transeúntes no pertenecientes al ámbito universitario que procuran ingresar en la zona, lo hacen con fines honestos: turismo, interés cultural, actividades socio-políticas, etc. por lo que, aun siendo necesaria una manera digital y sostenible de controlar el tránsito de personal, no es imperante la aplicación del método más rígido e inexorable en estado del arte.

En cambio, estas metas pueden ser alcanzadas con un módulo de software que se comunique o interactúe con otros (posteriormente implementados) para cumplir estas funciones. Una especie de *kit* de herramientas para garantizar la resolución de cada interrogante planteada, apoyado de un sistema de almacenamiento de datos que contenga la información del personal autorizado a adentrarse al campus universitario, y los datos de inventario para los departamentos que lo requieran.



Figura 2.1: Esquema General

La idea central es tratar todas las exigencias dentro de una misma función que se encargue de encontrarles una solución. El usuario establece algún tipo de conexión con esta “caja negra”, la cual manteniendo un intercambio continuo con la colección de datos puede garantizarle un resultado a sus preguntas.

Tal *software* y relación con el conjunto de datos se propone en forma de API REST. Con una API REST, se obtiene una representación de los datos solicitados almacenados en una base de datos, los cuales se procesan y utilizan en métodos o *endpoints* para dar respuesta a las demandas recibidas. Además, requerimientos especiales como la generación de códigos QR que contengan información de interés, pueden ser una de las muchas herramientas que contenga la API.

Es una propuesta viable, y la justificación de tal afirmación va entrelazada a la concepción del presente trabajo como columna vertebral de un sistema mayor, aún inexistente. Los sistemas que implementan API REST pueden escalar de forma eficiente porque se optimizan las interacciones entre el cliente y el servidor. La tecnología sin estado elimina la carga del servidor porque este no debe retener la información de solicitudes pasadas del cliente. El almacenamiento en caché bien administrado elimina de forma parcial o total algunas interacciones entre el cliente y el servidor. Todas estas características admiten la escalabilidad, sin provocar cuellos de botella en la comunicación que reduzcan el rendimiento [38].

Además, los servicios web RESTful admiten una separación total entre el cliente y el servidor, simplificando y desacoplando varios componentes de forma que cada parte pueda evolucionar de manera independiente. Los cambios de la plataforma o la tecnología en la aplicación del servidor no afectan la aplicación del cliente. La capacidad de ordenar en capas las funciones de la aplicación aumenta la flexibilidad aún más [39].

También es importante que sean independientes de la tecnología que se utiliza. Se pueden escribir aplicaciones del lado del cliente y del servidor en diversos lenguajes de programación, sin afectar el diseño de la API, o cambiar la tecnología subyacente en cualquiera de los lados sin que se vea afectada la comunicación [40].

Para el control de acceso, se considera como usuario al trabajador encargado de llevar a cabo esta labor. Este recopila la identificación de la persona en la puerta de entrada, y la somete al escrutinio de la API a través de la conexión a Internet para confirmar la validez de la información.



Figura 2.2: Interacción Usuario-API

Cuando se lleva a cabo esta operación no se tiene ningún tipo de restricción tecnológica en el dispositivo o sistema operativo que utilice el usuario, siempre que garantice establecer una conexión satisfactoria a la Web. Tampoco interesa con qué tipo de aplicación de *Front-end* interaccione gracias a la separación de preocupaciones que avala la API REST [41]. De esta manera se aíslan las problemáticas y se afianza la flexibilidad del producto.

También existe otro tipo de usuario encargado de conservar al día el estado de la base de datos con respecto al personal autorizado. Los nuevos ingresos, los ex-estudiantes, los trabajadores recién contratados, etc. representan parte de las modificaciones a mantener al tanto. Para completar las maniobras involucradas no intervienen limitantes en el tipo de base de datos y gestor de la misma, ni en los sistemas que conformen las capas *Front* y *Back* en el momento.

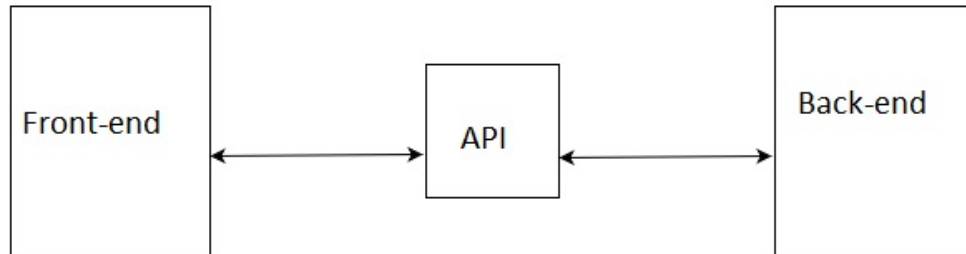


Figura 2.3: Esquema en 3 etapas

De igual forma, para mantener el correcto control del estado del inventario, se pueden gestionar los nuevos objetos a almacenar, la cantidad de objetos de un tipo en el almacén, etc. a través de una interfaz que se conecte a la API, y esta a su vez a la base de datos.

También desde la propia API se pueden procesar los datos necesarios para generar los códigos QR tanto del personal autorizado como de los elementos almacenados en los inventarios. Una vez se tenga un correcto control del contenido informacional de estos entes, obtener los códigos puede formar parte de este conjunto de instrumentos que ofrece una API fácilmente.

De esta forma se aísla el problema de generación de los códigos QR. Su contenido es obtenido de datos de los cuales se tiene constancia por la interacción con la base de datos, y su transmisión al usuario final se realiza a través de HTTP (Hypertext transfer protocol) como protocolo de transporte, sin necesidad de conocer qué tipo de aplicación la recibe y muestra.



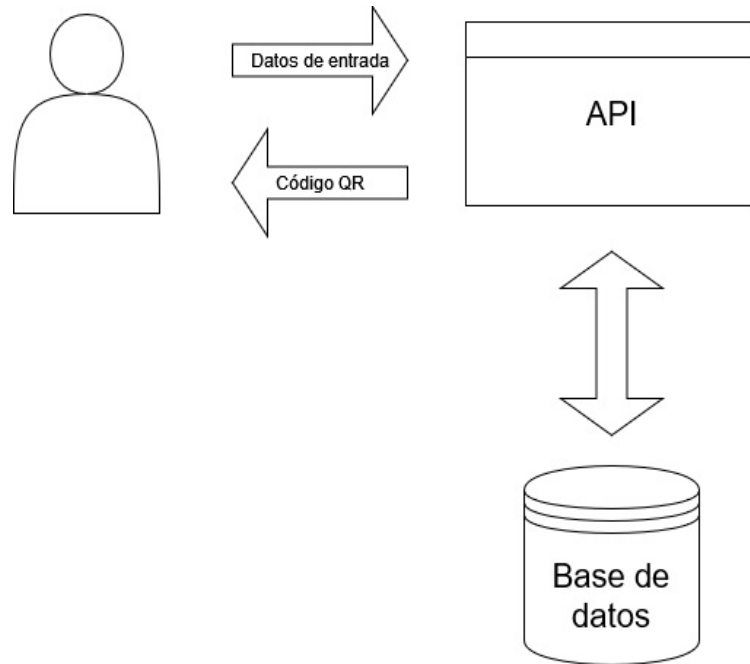


Figura 2.4: Flujo de generación de código QR

El usuario envía los datos necesarios para identificar el QR específico requerido. La API maneja esta información obtenida interactuando, primeramente, con la base de datos para obtener el contenido necesario. Las respuestas de la base de datos a las peticiones de la API son utilizadas en el código a generar y este, posteriormente, es enviado al usuario.

## Capítulo 3

# Detalles de Implementación y Experimentos

En este capítulo se tratan detalles específicos sobre la producción de la propuesta de solución brindada. Se analizan y explican las decisiones tomadas en el desarrollo del producto. Se revisan los detalles de implementación de manera general y también cubriendo las distintas especificaciones que conforman el resultado como un todo.

Teniendo en cuenta el estudio realizado en el estado del arte, se hace una elección de las herramientas a utilizar para el desarrollo del *software*, acompañado de una breve explicación. Se recorren interesantes aspectos en la realización a tener en cuenta en el entendimiento y posterior ampliación del sistema. Se analiza cómo se realiza la conexión con el sistema de almacenamiento de datos, cómo se modelan las distintas necesidades presentadas, y cómo se transfiere la información de una capa a la próxima.

### Elección de herramientas y plataformas

Durante la ejecución del propósito trazado se obtan por distintas vías de trabajo. Se elige un lenguaje de programación, un marco de trabajo (*framework*), un sistema de gestión de base de datos junto con un *ORM*(Object-Relational Mapping o mapeo objeto-relacional en español), entre otros elementos.

### Lenguaje de programación y *Framework*

La elección de estos dos componentes está fuertemente relacionada, ya que la comodidad del programador con el lenguaje es clave para la elaboración de código limpio y eficiente, pero un buen *framework* puede afectar positivamente la productividad en el desarrollo.

Python es un lenguaje completamente adecuado para la problemática actual. Cuenta con distintas bibliotecas para facilitar el trabajo y estas resultan de gran utilidad. Un ejemplo inmediato es la biblioteca *qrcode* [42], a través de la cual se maneja la generación de los códigos QR necesarios. Python es, además, un lenguaje bastante más intuitivamente entendible que otros, ofreciendo una mayor claridad a futuros programadores que se propongan extender las funcionalidades actuales.

Además, Python cuenta con distintas herramientas favorables para el desarrollo de APIs REST como Django y FastAPI. Mientras que Django tiene una estructura de código muy efectiva y hace hincapié en la seguridad al proporcionar defensa contra las inyecciones de SQL y otros ciber-ataques comunes, el *software* de Django puede limitar la velocidad de desarrollo debido a los numerosos módulos reutilizables [43]. El formato de FastApi puede facilitar más la interacción con los distintos *endpoints* de la API, garantizando mayor claridad para futuras expansiones. FastApi propone una capacidad de producción acelerada poco igualable por otras plataformas de la industria y un esquema intuitivo [44].

## Base de datos y *ORM*

Tras analizar las diferencias, los pros y los contras de cada uno, se selecciona como sistema gestor de bases de datos a SQLite. Un sistema pequeño y autocontenido, completamente equipado y de gran fiabilidad que trabaja con el motor de bases de datos SQL [45].

SQLAlchemy es el kit de herramientas SQL de Python y el *ORM* que brinda todo el poder y la flexibilidad de SQL para el desarrollo de una aplicación. Proporciona un conjunto completo de patrones de persistencia de nivel empresarial bien conocidos, diseñados para un acceso a las bases de datos de forma eficiente y de alto rendimiento, adaptados a un lenguaje de dominio simple [46]. Permite el mapeo de las clases a la base de datos de múltiples maneras, lo cual favorece a que el modelo de objeto y el esquema de la base de datos se desarrollen con una estructura limpia y desacoplada desde el principio [47].

## Aspectos de realización

La implementación del plan diseñado se ve conformada de múltiples detalles y especificaciones cuya documentación es válida, no solo para las aspiraciones actuales, sino para el futuro crecimiento del sistema.

## Conexión con la base de datos

Para incorporar la base de datos de SQLite, primeramente, se crea una instancia de **Engine**, lo cual es el punto de partida para cualquier aplicación de SQLAlchemy. Esta instancia es la base de operaciones entregada a la aplicación a través de un conjunto de conexiones que describen cómo hablar con la base de datos [48].

```
1 from sqlalchemy import create_engine
2 .
3 .
4 .
5 #Se crea la instancia engine
6 engine = create_engine("sqlite:///data.db")
```

Ejemplo de código 3.1: Engine

Se crea un sistema **declarative** que interactúe con el ORM de SQLAlchemy. La configuración objeto-relacional de SQLAlchemy implica la combinación de objetos **Table**, **mapper()** y objetos de clase para definir una clase mapeada. **declarative** permite que los tres se expresen a la vez dentro de la declaración de clase de un modelo [49].

```
1 from sqlalchemy.ext.declarative import declarative_base
2 .
3 .
4 .
5 #Se crea la instancia declaritive base
6 Base = declarative_base()
```

Ejemplo de código 3.2: Declarative

## Modelos

Para el control de acceso se crea un modelo **Person**, en cuya tabla correspondiente de la base de datos se almacena todo el personal autorizado, junto con una serie de información de relevancia. Como ejemplo se agregan el nombre y los apellidos, pero este modelo puede incrementar sus propiedades en medida de lo necesario.

```
1 class Person(Base):
2     __tablename__ = 'person'
3     id = Column(Integer, primary_key=True)
4     name = Column(String(20))
5     last_name = Column(String(20))
```

---

### Ejemplo de código 3.3: Personal

De igual forma, se utiliza un modelo para representar los objetos de inventario. Estos `Item` también contienen los datos de interés que los caracterizan. Los que en este caso se utilizan son un número de serie, descripción, cantidad en inventario y una categoría.

```
1 class Item(Base):
2     __tablename__ = 'item'
3     id = Column(Integer, primary_key=True)
4     serial_number = Column(String(8))
5     description = Column(String(256))
6     stock_number = Column(Integer)
7     category_id = Column(Integer, ForeignKey('category.id'))
8
9 class ItemCategory(Base):
10     __tablename__ = 'category'
11     id = Column(Integer, primary_key=True)
12     name = Column(String(256))
```

---

### Ejemplo de código 3.4: Inventario

Esas propiedades son las que se utilizan para llenar de contenido los códigos QR que se generan.

Para poder asentar la intención de crear una tabla de cada modelo y que cada vez que se cree una nueva instancia de estos se añada a dicha tabla, los modelos heredan de `Base`. Como se explica en el epígrafe anterior, a través de `declarative` se otorga a `Base` el rol de registro o *ledger* para estas operaciones.

## Endpoints

Para habilitar *endpoints* que lidien con peticiones de escritura y lectura de la base de datos ha de habilitarse, primeramente, una sesión a través del `sessionmaker` de SQLAlchemy, para manejar este flujo de interacción. Esta clase proporciona la interfaz donde se realizan las *queries* que devuelven y modifican los objetos mapeados al ORM. Estos objetos se mantienen dentro de la sesión, en una estructura de datos que mantiene copias únicas de cada uno [50].

El aspecto de control de acceso cuenta con dos responsabilidades principales. En primer lugar se debe poblar la base de datos con el personal y se debe lidiar con los que dejen el centro y ya no pertenezcan a este conjunto de personas, lo que implica las tareas de agregar y eliminar personas.

```

1 @app.get("/{id}", tags=['Personnel'])
2 def get_person(id:int, session: Session = Depends(get_session)):
3     person = session.query(models.Person).get(id)
4     return person
5
6 @app.post("/", tags=['Personnel'])
7 def add_person(person:schemas.Person, session = Depends(get_session)):
8     person = models.Person(name = person.name, last_name = person.
9         last_name)
10    session.add(person)
11    session.commit()
12    session.refresh(person)
13    return person
14
15 @app.put("/{id}", tags=['Personnel'])
16 def update_person(id:int, person:schemas.Person, session = Depends(
17     get_session)):
18     person_object = session.query(models.Person).get(id)
19     person_object.name = person.name
20     person_object.last_name = person.last_name
21     session.commit()
22     return person_object
23
24 @app.delete("/{id}", tags=['Personnel'])
25 def delete_person(id:int, session = Depends(get_session)):
26     person_object = session.query(models.Person).get(id)
27     session.delete(person_object)
28     session.commit()
29     session.close()
30     return 'person was deleted'

```

Ejemplo de código 3.5: CRUD para el personal autorizado

Por otro lado, el encargado de dejar continuar o no a los transeuntes debe poder obtener la información almacenada de un individuo que intente adentrarse al centro. Esto se hace recibiendo el código QR con la información de esa persona cifrada dentro de si.

```

1 @app.get("/{id}", tags=['Personnel'])
2 def get_person(id:int, session: Session = Depends(get_session)):
3     person = session.query(models.Person).get(id)
4     img = qrcode.make(person.name + " " + person.last_name)
5     img.save("person_qr.png")
6     return "QR Code saved"

```

Ejemplo de código 3.6: Códigos QR en Personal

De igual manera sucede con la gestión del inventario. Es necesario suplir de métodos que controlen la creación de los nuevos objetos a ser almacenados, editen la

cantidad de objetos de un tipo que se tiene en inventario y obtener todos elementos en registro.

```

1 @app.get("/inventory/", tags=['Inventory'])
2 def get_items(session: Session = Depends(get_session)):
3     items = session.query(models.Item).all()
4     return items
5
6 @app.post("/inventory/", tags=['Inventory'])
7 def add_item(item:schemas.Item, session = Depends(get_session)):
8     item = models.Item(
9         serial_number = item.serial_number,
10        description = item.description,
11        stock_number = item.stock_number,
12        category_id = item.category_id
13    )
14    session.add(item)
15    session.commit()
16    session.refresh(item)
17    return item
18
19 @app.get("/inventory/{id}", tags=['Inventory'])
20 def get_item(id:int, session: Session = Depends(get_session)):
21     item = session.query(models.Item).get(id)
22     img = qrcode.make(item.description + " " + item.serial_number)
23     img.save("item_qr.png")
24     return "QR Code Saved"
25
26 @app.put("/inventory/{id}", tags=['Inventory'])
27 def update_item(id:int, item:schemas.Item, session = Depends(get_session)):
28     item_object = session.query(models.Item).get(id)
29     item_object.serial_number = item.serial_number
30     item_object.description = item.description
31     item_object.stock_number = item.stock_number
32     session.commit()
33     return item_object
34
35 @app.delete("/inventory/{id}", tags=['Inventory'])
36 def delete_item(id:int, session = Depends(get_session)):
37     item_object = session.query(models.Item).get(id)
38     session.delete(item_object)
39     session.commit()
40     session.close()
41     return 'item was deleted'

```

Ejemplo de código 3.7: CRUD para el inventario

Para etiquetar de manera eficiente los objetos dentro del inventario, se genera un

código QR con la información necesaria para ello. El usuario puede acceder a estos códigos vía un *endpoint* específica

```
1 @app.get("/inventory/QR/{id}", tags=['Inventory'])
2 def get_item(id:int, session: Session = Depends(get_session)):
3     item = session.query(models.Item).get(id)
4     img = qrcode.make(item.description + " " + item.serial_number)
5     img.save("item_qr.png")
6     return "QR Code Saved"
```

#### Ejemplo de código 3.8: Códigos QR en Inventario

Como fragmento de las características REST de la API que se construye, los CRUDs se implementan a través de métodos `get`, `post`, `put`, `delete`. Cada uno de estos métodos constituye un *endpoint* de la aplicación que se está implementando.

Para la obtención de los códigos QR se utiliza la biblioteca de Python `qrcode`. Esta otorga gran cantidad de funcionalidades para útiles que facilitan la generación y almacenamiento del código.

## Interfaz de la API

Actualmente, la estructura de la API, la organización dispuesta a sus *endpoints* y las respuestas obtenidas de la misma son mostradas a través de la interfaz de usuario de Swagger. Esta permite que cualquier persona, ya sea el equipo de desarrollo o los consumidores finales, visualice e interactúe con los recursos de la API. No requiere tener implementada ninguna lógica en la capa de presentación ya que se genera automáticamente a partir de la especificación que facilita la implementación del *back-end* y el consumo del lado del cliente.

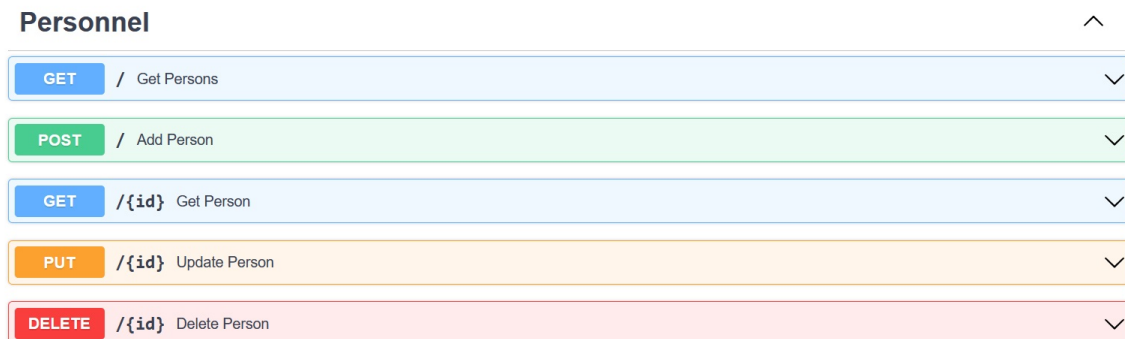


Figura 3.1: Vista de CRUD del personal



Para realizar las distintas operaciones que dispone la API, basta con ejecutar el método deseado que muestra Swagger. La siguiente ilustración evidencia como rellenando las propiedades de una nueva persona, esta puede agregarse al conjunto de personas autorizadas a acceder al campus universitario.

The image shows a Swagger UI interface for the 'Add Person' endpoint. The interface is divided into several sections:

- Method and Path:** A green bar at the top left shows 'POST' and '/ Add Person'.
- Parameters:** A section below the method bar with a 'Parameters' tab. It shows 'No parameters' and has 'Cancel' and 'Reset' buttons.
- Request body:** A section below the parameters with a 'Request body' tab. It shows 'application/json' as the content type and a required JSON body: 

```
{  "name": "Juan",  "last_name": "García"}
```

.
- Execute:** A large blue button at the bottom labeled 'Execute'.

Figura 3.2: Agregar Persona

De la misma manera, a través de los restantes *endpoints* que conforman la API en su totalidad, se controla el flujo de información entre el usuario final y la base de datos.

## Experimentación

Para llevar a cabo la experimentación del producto implementado se hace uso del paquete de Python llamado `sqladmin`, para simular la interacción del cliente con la API de una manera cómoda al usuario. En el presente epígrafe se hace muestra de algunos de los intercambios realizables por el usuario con el sistema implementado.

Inventory

Export+ New Item

Actions ▾

<input type="checkbox"/>	id	Serial Number	Description	Stock Number
<input type="checkbox"/>	1	HAB123AS	AC Machine	1
<input type="checkbox"/>	2	AJO167DS	Pencil	25
<input type="checkbox"/>	3	UI8IO90A	RAM Memory DDR4 8GB	4

Showing 1 to 3 of 3 items< prev1next > Show 10 / Page ▾

Figura 3.3: Inventario

Se obtiene un listado de todo el inventario presente en el sistema el cual puede disfrutar acceso al área restringida. A partir de este punto el usuario puede ver, editar o eliminar cualquier elemento de la tabla correspondiente o agregar nuevas entradas con el botón en la esquina superior derecha, en correspondiente con las funcionalidades CRUD implementadas en la API.

New Item

Serial Number

BX88B88B

Description

Chair

Stock Number

Many

Please enter a number.

Cancel

Save

Save and continue editing

Save and add another

Figura 3.4: Añadir Objeto al inventario

Al intentar añadir un nuevo objeto este debe superar las distintas validaciones dispuestas en el esquema del modelado. En caso negativo debe editar el campo problemático y de otra forma, si todo está corregido y en orden, se incluye el nuevo elemento en la tabla de la base de datos.

Edit Item

---

Serial Number

HAB123AS

Description

AC Machine

Stock Number

2

Cancel

Save

Save and continue editing

Save and add another

Figura 3.5: Editar Objeto

Un usuario quisiera editar un objeto para aumentar, por ejemplo, la cantidad de objetos de ese tipo que hay en inventario. Esto se procesa con naturalidad a través de la interfaz de usuario accediendo al *endpoint* correspondiente de FastApi.

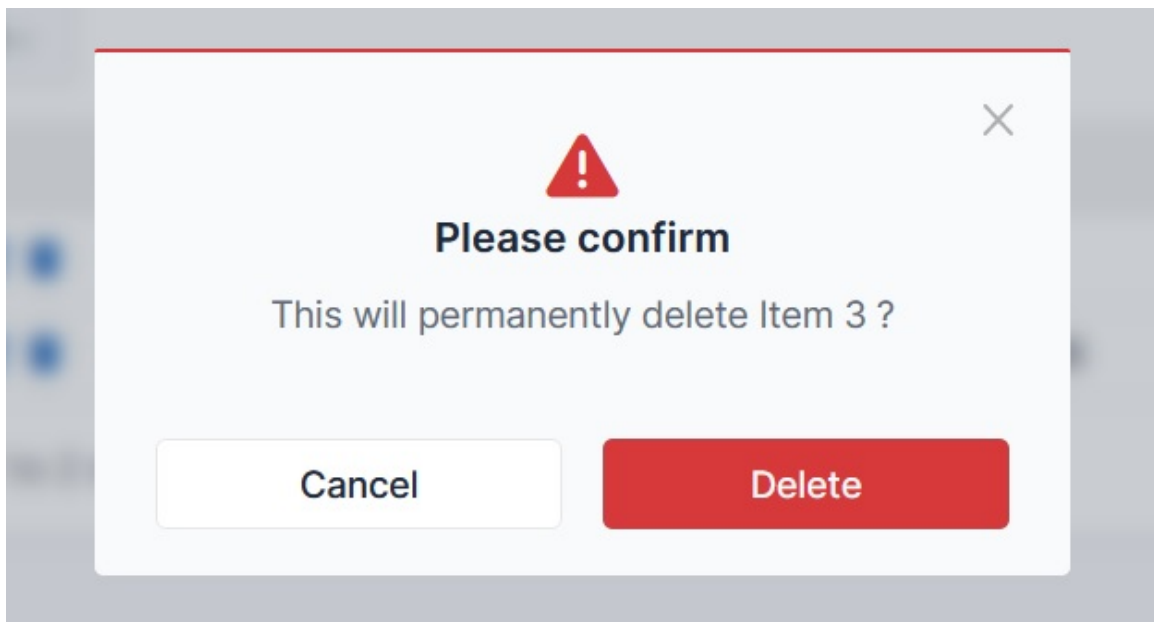
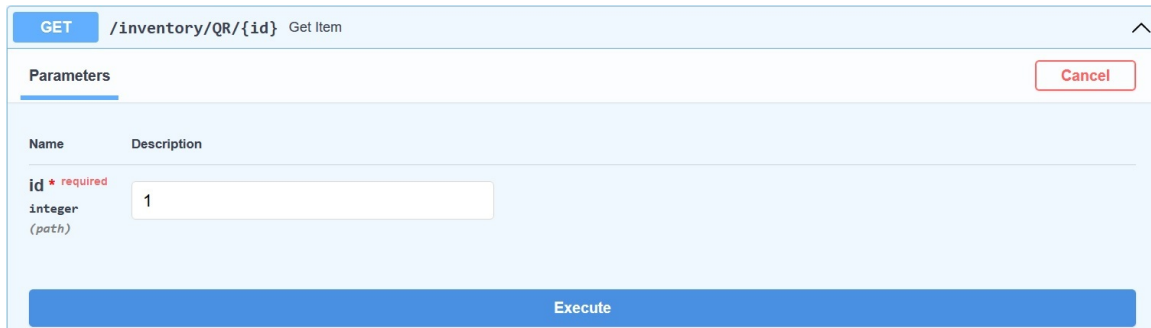


Figura 3.6: Eliminar Objeto

Para eliminar un objeto de la tabla, luego de interactuar con el botón correspondiente, el usuario recibe un mensaje *pop-up* de confirmación y, en caso de aceptar, se

procede con éxito y se elimina tal registro de la base de datos.

Como modo de comprobación en la generación de códigos QR, se puede acceder a la API directamente a través del Swagger ofrecido y ejecutar el método correspondiente.



The image shows a Swagger UI interface for a REST API endpoint. At the top, it displays the HTTP method 'GET' and the endpoint path '/inventory/QR/{id}' with the description 'Get Item'. Below this, there is a 'Parameters' section. It contains a table with two columns: 'Name' and 'Description'. The first parameter is 'id', which is marked as 'required' with a red asterisk. Its type is 'integer' and its location is '(path)'. A text input field next to 'id' contains the value '1'. At the bottom of the interface, there is a large blue button labeled 'Execute'. A 'Cancel' button is also visible in the top right corner of the parameters section.

Figura 3.7: Endpoint QR

Se aprecia la petición enviada a la API de generar un QR para un objeto de inventario en específico. A continuación se muestra la imagen obtenida.



Figura 3.8: Código QR del objeto

De manera análoga se maneja el tema de control de acceso, utilizando el mismo estilo de procedimiento y logrando un flujo de trabajo igual de cómodo e intuitivo.

# Conclusiones

En el presente trabajo se aborda como punto principal la creación de un sistema digitalizado que, comenzando por solo algunas funcionalidades, funda las bases para gestiones actualmente dependientes de un obsoleto sistema manual. Se pone en evidencia como el desarrollo y mantenimiento de una API garantiza la solución no solo a los problemas de control de inventario y punto de acceso, sino a múltiples dificultades que puedan ser presentadas en el futuro.

Se hace un estudio de como son combatidos problemas del mismo campo al día de hoy y, analizando los convenientes e inconvenientes de cada estrategia, se plantea una propuesta exenta de requerimientos tecnológicos, sostenible y reutilizable. Además, se muestra una vía de implementación del producto a través de medios apropiados, argumentando la elección de los mismos.

# Recomendaciones

Se recomienda la continuidad y el mantenimiento de la API para disminuir la carga de trabajo en distintos sectores de la Universidad. Las capacidades de un sistema así pueden potenciar considerablemente la productividad en un centro de trabajo, utilizada apropiadamente y fortaleciendo sus utilidades.

Como parte del estudio del estado del arte, se muestra como algunos de los sistemas de control de acceso en la vanguardia tecnológica actual utilizan métodos de control biométricos. Estos resultan muy interesantes y su puesta en práctica puede ser de gran valor. Su freno principal es la cantidad de requerimientos técnicos que presenta. Plantear un plan de inversión en tales sistemas y trabajar en cursos que enseñen a futuros estudiantes de la Facultad de Ciencias de la Computación las nociones principales de esa rama puede traer beneficios notables a la institución.

# Bibliografía

- [1] R. Tamayo, *Transformacion digital, proceso estrategico y urgente para Cuba*, 2022 (vid. pág. 1).
- [2] B. S. S. Tejesh y S. Neeraja, «Warehouse inventory management system using IoT and open source framework,» *Alexandria engineering journal*, vol. 57, n.º 4, págs. 3817-3823, 2018 (vid. pág. 3).
- [3] S. M. Huynh, D. Parry, A. C. M. Fong y J. Tang, «Novel RFID and ontology based home localization system for misplaced objects,» *IEEE Transactions on Consumer Electronics*, vol. 60, n.º 3, págs. 402-410, 2014 (vid. pág. 4).
- [4] S. Hodgson, F. Nabhani y S. Zarei, «AIDC feasibility within a manufacturing SME,» *Assembly Automation*, 2010 (vid. pág. 4).
- [5] N. Wartha y V. Londhe, «Context-Aware Approach for enhancing security and privacy of RFID,» *International Journal of Engineering And Computer Science*, vol. 4, n.º 1, pág. 10 781 088, 2015 (vid. pág. 4).
- [6] S. S. Saab y Z. S. Nakad, «A standalone RFID indoor positioning system using passive tags,» *IEEE Transactions on Industrial Electronics*, vol. 58, n.º 5, págs. 1961-1970, 2010 (vid. pág. 4).
- [7] P. Kieseberg et al., «QR code security,» en *Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia*, 2010, págs. 430-435 (vid. pág. 4).
- [8] Y. Liu, J. Yang y M. Liu, «Recognition of QR Code with mobile phones,» en *2008 Chinese control and decision conference*, IEEE, 2008, págs. 203-206 (vid. pág. 4).
- [9] S. Tiwari, «An introduction to QR code technology,» en *2016 international conference on information technology (ICIT)*, IEEE, 2016, págs. 39-44 (vid. pág. 5).
- [10] S. Singh, «QR code analysis,» *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 6, n.º 5, 2016 (vid. pág. 5).

- [11] F. Kerschbaum, «An access control model for mobile physical objects,» en *Proceedings of the 15th ACM symposium on Access control models and technologies*, 2010, págs. 193-202 (vid. pág. 5).
- [12] K. Shahriari y M. Shahriari, «IEEE standard review—Ethically aligned design: A vision for prioritizing human wellbeing with artificial intelligence and autonomous systems,» en *2017 IEEE Canada International Humanitarian Technology Conference (IHTC)*, IEEE, 2017, págs. 197-201 (vid. pág. 5).
- [13] S. Hadim y N. Mohamed, «Middleware for wireless sensor networks: A survey,» en *2006 1st International Conference on Communication Systems Software & Middleware*, IEEE, 2006, págs. 1-7 (vid. pág. 5).
- [14] R. Newman, *Security and access control using biometric technologies*. Cengage Learning, 2009 (vid. pág. 6).
- [15] D. Bhattacharyya, R. Ranjan, F. Alisherov, M. Choi et al., «Biometric authentication: A review,» *International Journal of u-and e-Service, Science and Technology*, vol. 2, n.º 3, págs. 13-28, 2009 (vid. pág. 6).
- [16] A. C. Weaver, «Biometric authentication,» *Computer*, vol. 39, n.º 2, págs. 96-97, 2006 (vid. pág. 6).
- [17] E. Longo y J. Stapleton, «PKI Note: Smart Cards,» en *PKI Note Series, PKI Forum*, 2002 (vid. pág. 6).
- [18] A. Ranganathan, B. Danev y S. Capkun, «Proximity verification for contactless access control and authentication systems,» en *Proceedings of the 31st Annual Computer Security Applications Conference*, 2015, págs. 271-280 (vid. pág. 7).
- [19] O. Abd Allah, S. Abdalla, M. Mekki y A. Awadallah, «RFID based access control and registration system,» en *2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, IEEE, 2018, págs. 1-4 (vid. pág. 7).
- [20] L. A. P. Neves, G. A. Giraldi, K. S. Martins y W. R. S. Lima, «QRCode DOOR Project: Access Control Application using QR Code Image,» (vid. pág. 7).
- [21] S. Sohan, F. Maurer, C. Anslow y M. P. Robillard, «A study of the effectiveness of usage examples in REST API documentation,» en *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, 2017, págs. 53-61 (vid. pág. 8).
- [22] M. Biehl, *Webhooks—Events for RESTful APIs*. API-University Press, 2017, vol. 4 (vid. pág. 8).
- [23] F. Belqasmi, J. Singh, S. Y. B. Melhem y R. H. Glitho, «SOAP-based vs. RESTful web services: a case study for multimedia conferencing,» *IEEE internet computing*, vol. 16, n.º 4, págs. 54-63, 2012 (vid. pág. 8).



- [24] M. Biehl, *GraphQL API Design*. API-University Press, 2018, vol. 5 (vid. pág. 8).
- [25] I. Fette y A. Melnikov, «The websocket protocol,» inf. téc., 2011 (vid. pág. 9).
- [26] A. MySQL, *MySQL*, 2001 (vid. pág. 9).
- [27] B. Raza, A. Mateen, M. Sher, M. M. Awais y T. Hussain, «Autonomicity in Oracle database management system,» en *2010 International Conference on Data Storage and Data Engineering*, IEEE, 2010, págs. 296-300 (vid. pág. 9).
- [28] R. Mistry y S. Misner, *Introducing Microsoft SQL Server 2014*. Microsoft Press, 2014 (vid. pág. 9).
- [29] B. PostgreSQL, «PostgreSQL,» *Web resource: <http://www.PostgreSQL.org/about>*, 1996 (vid. pág. 9).
- [30] M. Owens y G. Allen, *SQLite*. Springer, 2010 (vid. pág. 9).
- [31] D. Mongo, *Mongodb*, 2015 (vid. pág. 10).
- [32] G. Van Rossum y F. L. Drake Jr, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995, vol. 620 (vid. pág. 10).
- [33] M. Campione, K. Walrath y A. Huml, *The Java tutorial: a short course on the basics*. Addison-Wesley Professional, 2001 (vid. pág. 10).
- [34] S. Tilkov y S. Vinoski, «Node.js: Using JavaScript to build high-performance network programs,» *IEEE Internet Computing*, vol. 14, n.º 6, págs. 80-83, 2010 (vid. pág. 11).
- [35] R. Lerdorf, K. Tatroe, B. Kaehms y R. McGredy, *Programming Php*. O'Reilly Media, Inc.", 2002 (vid. pág. 11).
- [36] D. H. Hansson y R. C. Team, «Ruby on rails,» *Development*, vol. 4, pág. 0, 2009 (vid. pág. 11).
- [37] J. Newmarch y R. Petty, «Overview of the Go language,» en *Network Programming with Go Language*, Springer, 2022, págs. 25-34 (vid. pág. 11).
- [38] M. Masse, *REST API design rulebook: designing consistent RESTful web service interfaces*. O'Reilly Media, Inc.", 2011 (vid. pág. 13).
- [39] A. Neumann, N. Laranjeiro y J. Bernardino, «An analysis of public REST web service APIs,» *IEEE Transactions on Services Computing*, vol. 14, n.º 4, págs. 957-970, 2018 (vid. pág. 13).
- [40] H. Wenhui, H. Yu, L. Xueyang y X. Chen, «Study on REST API test model supporting web service integration,» en *2017 IEEE 3rd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HSPC), and IEEE International Conference on Intelligent Data and Security (IDS)*, IEEE, 2017, págs. 133-138 (vid. pág. 13).

- [41] I. Zuzak y S. Schreier, «Arrested development: Guidelines for designing rest frameworks,» *IEEE Internet Computing*, vol. 16, n.º 4, págs. 26-35, 2012 (vid. pág. 14).
- [42] <https://pypi.org/project/qrcode/>, «Python QR Code Library,» (vid. pág. 18).
- [43] V. Gagliardi, «Modern Django and the Django REST Framework,» en *Decoupled Django*, Springer, 2021, págs. 31-40 (vid. pág. 18).
- [44] L. O’Sullivan, «Secure communication platform design manual,» Tesis doct., Institute of Technology Carlow, 2020 (vid. pág. 18).
- [45] <https://www.sqlite.org/index.html>, «SQLite Documentation,» (vid. pág. 18).
- [46] R. Copeland, *Essential sqlalchemy*. .ºReilly Media, Inc.", 2008 (vid. pág. 18).
- [47] J. Myers y R. Copeland, *Essential SQLAlchemy: Mapping Python to Databases*. .ºReilly Media, Inc.", 2015 (vid. pág. 18).
- [48] <https://docs.sqlalchemy.org/en/14/core/engines.html>, «Engine Configuration SQLAlchemy,» (vid. pág. 19).
- [49] <https://docs.sqlalchemy.org/en/13/orm/extensions/declarative.html>, «SQLAlchemy Declarative Basic Use,» (vid. pág. 19).
- [50] <https://docs.sqlalchemy.org/en/14/orm/>, «SQLAlchemy ORM Sessions,» (vid. pág. 20).