TRABAJO FINAL DE CARRERA

Río Negro
Universidad Nacional

# Sistema de Presentismo con QR

Licenciatura en Sistemas

Autor: José Luis Orias

Director: Enrique Molinari

Codirector: Horacio Muñoz


Viedma, Río Negro, Argentina - 2025

## Índice

## Introducción

Los registros en papel o sistemas antiguos generan errores, pérdida de datos y dificultad en la gestión. En los sistemas tradicionales, algunos empleados fichan por compañeros ausentes. Existen dificultades al momento de acceder al historial de entrada/salida o justificar ausencias.

Por otro lado, los costes de implementar, mantener y escalar un sistema de asistencia con biometría o tarjetas magnéticas o con chip es mayor, ya que se requiere hardware especializado.

A esto también se le suma el uso innecesario de tarjetas plásticas, papeles o fichas que generan más costes y residuos.

Por dichos motivos, el presente trabajo final de carrera consiste en el desarrollo de un sistema de presentismo basado en códigos QR para el registro de asistencia de los empleados de una empresa. Este sistema incluye una interfaz web para la visualización de los códigos QR (Biblioteca de la CEPAL [CEPAL], 2022) , una aplicación móvil desarrollada en android (Wambua, 2023) destinada a escanear dichos códigos y registrar la asistencia, un panel web para la administración de usuarios y QRs, y una API REST (IBM, 2025) que expone los servicios del backend. De esta manera, se aplicarán habilidades de desarrollo de software full-stack, que abarcan el diseño, la construcción, el despliegue y la operación de los componentes de frontend, backend y base de datos.

Con la utilización de este sistema, se evitarán suplantaciones y errores en el registro de asistencia, se mejorará la experiencia de los empleados, ofreciendo mayor comodidad y rapidez, como así también se podrá reducir costos de implementación, mantenimiento y escalabilidad.

También cabe destacar que se hará una gran contribución a la sostenibilidad apoyando la digitalización verde (UNEP, 2024), evitando la utilización de papel y/o tarjetas plásticas que generan costes y residuos innecesarios.

## Una Solución Moderna para el Registro de Asistencia

El control de asistencia constituye un proceso esencial dentro de las organizaciones, dado que permite garantizar la puntualidad, permanencia y cumplimiento de las jornadas laborales por parte de los empleados. Tradicionalmente, este registro se ha llevado a cabo mediante planillas en papel, relojes biométricos o tarjetas magnéticas, métodos que, si bien cumplen su función, presentan limitaciones en términos de eficiencia, seguridad y capacidad de gestión de la información.

En este contexto, la incorporación de un sistema de presentismo basado en códigos QR se plantea como una alternativa innovadora y eficaz. A través de esta tecnología, cada empleado puede registrar su asistencia escaneando un código QR generado dinámicamente, el cual es

validado en tiempo real y almacenado en una base de datos central. Este mecanismo garantiza mayor precisión en los registros y posibilita la obtención de reportes automáticos que contribuyen a la toma de decisiones estratégicas en el ámbito de la gestión de recursos humanos.

Su adopción favorece la confiabilidad, la seguridad y la transparencia en el control de asistencia, contribuyendo a una gestión más ágil y efectiva de los recursos humanos.

## Una Alternativa más Eficiente

Los sistemas tradicionales de control de asistencia, como el fichaje mediante tarjetas magnéticas o dispositivos biométricos de huella digital, han sido ampliamente utilizados en organizaciones de distintos tamaños. No obstante, estos métodos presentan limitaciones importantes: requieren de hardware especializado que implica costos de adquisición y mantenimiento, pueden generar demoras en el registro y, en ocasiones, se ven afectados por fallas técnicas o desgaste de los dispositivos.

Frente a estas dificultades, surge una alternativa más eficiente basada en códigos QR, la cual elimina la necesidad de depender de equipos físicos específicos. A través de este sistema, los empleados registran su asistencia escaneando un código dinámico generado en tiempo real, validado de forma automática y almacenado en una base de datos centralizada. Este procedimiento no solo agiliza el control de presentismo, sino que también aporta mayor flexibilidad, dado que puede integrarse fácilmente en aplicaciones móviles o plataformas web accesibles desde cualquier dispositivo con cámara.

A continuación se muestra una comparación entre los sistemas de fichaje tradicional con tarjeta magnética y con biometría, y los sistemas modernos con QR.

| Criterio | Tarjeta Magnética | Biometría | Código QR |
|---|---|---|---|
| Costo de Implementación | Medio: requiere lectores y tarjetas para los empleados | Alto: equipos biométricos costosos. | Bajo: puede usarse con cámaras y software. |
| Facilidad de Uso | Sencillo: basta pasar la tarjeta. | Muy fácil: solo colocar dedo o rostro. | Fácil: escanear con el móvil |
| Seguridad / Suplantación | Media: se puede perder o prestar la tarjeta. | Alta: difícil de falsificar | Media-Alta: QR dinámico. |
| Mantenimiento | Bajo: solo reemplazo de tarjetas o lector. | Medio-Alto: limpieza, calibración del lector biométrico. | Bajo: mantenimiento de software. |
| Dependencia de | Alta: requiere lector y | Alta: requiere | Baja: funciona con |

| | | | |
|---|---|---|---|
| Hardware | tarjetas físicas. | dispositivos especializados. | celulares y una pantalla |
| Privacidad / Datos Personales | Alta: no almacena datos sensibles. | Sensible: maneja datos biométricos (requiere protección legal). | Media: maneja datos de usuario. |
| Integración con Sistemas Digitales | Media: necesita adaptadores o software específico. | Alta: se integra fácilmente con sistemas modernos de RRHH. | Alta: se conecta bien con apps móviles y plataformas web. |

## Optimización de Costos en el Registro de Asistencia

Uno de los principales desafíos en la implementación de sistemas de control de asistencia tradicionales radica en los elevados costos de inversión inicial y de mantenimiento. Tecnologías como relojes biométricos o lectores de tarjetas magnéticas requieren hardware especializado, cuya instalación y actualización implican un gasto significativo para la organización.

En contraste, un sistema de presentismo basado en códigos QR se presenta como una alternativa económicamente más accesible. Su funcionamiento requiere únicamente de una pantalla para la visualización de los códigos dinámicos, lo cual reduce considerablemente la inversión en infraestructura. El registro de asistencia puede realizarse mediante dispositivos móviles convencionales con cámara, evitando la dependencia de equipamiento adicional o costoso.

De este modo, la utilización de códigos QR no solo garantiza un control de asistencia confiable y seguro, sino que también contribuye a la optimización de los recursos financieros de la empresa, ofreciendo una solución moderna, práctica y alineada con los principios de eficiencia organizacional.

## Fortaleciendo la Seguridad en el Registro de Asistencia

Los sistemas tradicionales de control de presentismo suelen presentar vulnerabilidades que pueden derivar en acciones fraudulentas, como el fichaje en nombre de otro empleado, la suplantación de identidad o errores derivados de registros manuales. Estas situaciones no solo afectan la confiabilidad de los datos, sino que también impactan negativamente en la gestión de recursos humanos y en la transparencia organizacional.

La implementación de un sistema de asistencia basado en códigos QR dinámicos permite reducir de manera significativa estas prácticas indebidas. Al generar códigos únicos y

temporales, se evita la posibilidad de reutilizarlos o compartirlos entre empleados, garantizando que cada registro corresponda de manera legítima a la persona que realiza la marcación. Además, la validación en tiempo real y el almacenamiento automático de la información contribuyen a minimizar errores humanos y asegurar la integridad de los datos.

En este sentido, el uso de códigos QR fortalece los mecanismos de seguridad del sistema de presentismo, previniendo fraudes, suplantaciones y registros incorrectos, lo cual se traduce en una mayor confiabilidad y equidad dentro de la organización.

## Una Mejor Experiencia para los Empleados

La satisfacción y comodidad de los empleados son factores clave en la adopción de cualquier sistema organizacional. Los métodos tradicionales de control de asistencia, como las planillas manuales o los dispositivos biométricos, pueden resultar engorrosos, generar demoras en el ingreso y provocar una percepción negativa respecto al proceso de registro.

Un sistema de presentismo basado en códigos QR ofrece una alternativa ágil, cómoda y sencilla de utilizar. Mediante el escaneo del código desde un dispositivo móvil, el empleado puede registrar su asistencia en cuestión de segundos, sin necesidad de contacto físico ni de portar tarjetas u otros elementos adicionales. Este procedimiento no solo optimiza los tiempos de marcación, sino que también se adapta con facilidad a diferentes entornos laborales y modalidades de trabajo.

En consecuencia, la incorporación de esta tecnología contribuye a mejorar la experiencia del empleado, promoviendo una interacción más fluida con el sistema, reduciendo posibles frustraciones y fortaleciendo el compromiso con las políticas organizacionales.

## Más Transparencia, Más Trazabilidad

La gestión eficiente de la asistencia requiere no solo de precisión en el registro, sino también de mecanismos que garanticen la transparencia y la trazabilidad de la información generada. En los sistemas tradicionales, los errores de registro, la manipulación de datos o la dificultad para acceder a históricos confiables pueden comprometer la integridad del proceso y afectar la confianza organizacional.

La implementación de un sistema de presentismo basado en códigos QR permite incrementar de manera significativa la transparencia y la trazabilidad. Cada marcación queda registrada en tiempo real dentro de una base de datos centralizada, acompañada de información verificable como la identidad del empleado, la fecha y la hora exacta del evento. Esto asegura la generación de un historial completo y accesible que no puede ser alterado de manera indebida.

De este modo, la utilización de códigos QR contribuye a fortalecer la confianza y la equidad en el control de asistencia, al tiempo que ofrece a las organizaciones herramientas precisas para la auditoría, el análisis y la toma de decisiones estratégicas en la gestión de recursos humanos.

## Un aporte a la Sostenibilidad Empresarial

La sostenibilidad se ha consolidado en los últimos años como un eje estratégico en el desarrollo de las organizaciones. Las empresas son evaluadas en relación con su impacto social y ambiental. En este contexto, la digitalización verde surge como una alternativa clave para reducir la dependencia de materiales físicos y, a la vez, promover prácticas responsables que favorezcan tanto a la organización como al entorno en el que se inserta.

El presente proyecto propone la implementación como una solución tecnológica que contribuye a la sostenibilidad empresarial. A diferencia de los métodos tradicionales que utilizan planillas en papel, tarjetas plásticas u otros soportes físicos, este sistema se apoya en dispositivos móviles y plataformas digitales para registrar la asistencia de los empleados. De esta forma, se logra una disminución significativa en el uso de recursos materiales, lo cual repercute de manera directa en la reducción de la huella ecológica de la empresa.

Otro aspecto relevante es el fomento de una cultura digital sostenible dentro de la organización. La implementación de este tipo de herramientas tecnológicas sensibiliza a los empleados acerca de la importancia de adoptar prácticas responsables y respetuosas con el medioambiente, al mismo tiempo que promueve la adaptación a nuevas dinámicas laborales asociadas a la transformación digital.

De esta manera, se busca contribuir a la modernización empresarial mediante la incorporación de herramientas digitales que no solo optimicen los procesos internos, sino que también reflejan un compromiso concreto con la sostenibilidad.

## Alcance del Trabajo

Este trabajo abarca el desarrollo e implementación de un sistema de presentismo basado en códigos QR, cuyo propósito es optimizar el registro de asistencia de los empleados mediante un proceso ágil, seguro y confiable. El alcance contempla la incorporación de funcionalidades orientadas a mejorar la eficiencia en la gestión de la asistencia, tales como la generación dinámica de códigos QR y su visualización en un frontend web, el acceso de usuarios a través de una aplicación móvil y de un panel de administración, el escaneo y validación conjunta de códigos QR y usuarios, así como el registro de asistencias en la base de datos.

Asimismo, el sistema incluirá herramientas de administración como el alta, baja lógica, consulta y modificación de empleados, y la visualización de registros de asistencia por parte del administrador. Estas funcionalidades permiten cubrir los requerimientos esenciales de control y gestión de asistencia en un entorno digital, favoreciendo la precisión y confiabilidad del proceso.

No obstante, el sistema no incorporará ciertas características que se consideran fuera del alcance de este trabajo, entre ellas la modificación de datos personales por parte de los empleados, la consulta de asistencias personales de los empleados, la generación y exportación de reportes en formatos Excel o PDF, la emisión de alertas automáticas al administrador frente a irregularidades en el registro de asistencias, ni el registro de la geolocalización de los empleados en el momento de su ingreso, ni tampoco la gestión de usuarios (actualizar y restablecer contraseñas).

En este marco, el alcance definido establece los límites funcionales del sistema y permite enfocar los esfuerzos de desarrollo en aquellas características que aportan mayor valor en términos de digitalización verde, sostenibilidad y modernización de la gestión empresarial.

## Metodología de Trabajo Aplicada

El presente trabajo final de carrera se centra en el desarrollo de software, y para gestionar su ejecución se adoptó una metodología híbrida, que combina enfoques tradicionales y ágiles. Esta estrategia permitió estructurar y organizar el proyecto de manera clara, asegurando la definición precisa de objetivos, la planificación de las etapas y la capacidad de adaptación ante cambios durante el desarrollo.

### Planteamiento del Problema y Objetivos

En esta fase se identificaron las necesidades a resolver, se definieron los objetivos generales y específicos del sistema, se determinaron los requerimientos funcionales y no funcionales, y se evaluaron los riesgos asociados al desarrollo del proyecto.

### Diseño de la Solución y Metodología

Se definió la arquitectura del sistema, el modelo de datos y las tecnologías a utilizar. Además, se estableció la metodología híbrida, combinando la planificación estructurada de los métodos tradicionales con la flexibilidad de Scrum (Alaimo, 2017) para la gestión de las tareas durante el desarrollo.

**Desarrollo del Sistema**

En esta etapa se implementaron las funcionalidades del sistema, aplicando la metodología ágil Scrum para organizar el trabajo en ciclos iterativos, definir objetivos a corto plazo y adaptarse a cambios en los requerimientos conforme avanzaba el proyecto.

**Cierre del Desarrollo y Ajustes**

Se realizaron pruebas, correcciones y ajustes necesarios para garantizar la funcionalidad, seguridad y confiabilidad del sistema, asegurando que cumpliera con los objetivos planteados.

**Remisión y Corrección del Documento Final**

Se revisó el contenido del trabajo, ajustando redacción, formato y coherencia, así como la integración de los resultados del desarrollo en el documento final.

**Presentación y Defensa**

Finalmente, se preparó la presentación del proyecto y la defensa ante el tribunal, destacando los objetivos alcanzados, las soluciones implementadas y la contribución del sistema a la digitalización y sostenibilidad empresarial.

Cabe destacar que la escritura del trabajo final de carrera fue una actividad continua que se realizó a lo largo de todo el desarrollo del proyecto, integrando la documentación de cada fase y reflejando los avances, decisiones y resultados obtenidos en cada etapa.

## Etapa del Desarrollo del Sistema

En el siguiente apartado de Desarrollo, se detalla el proceso completo de construcción del sistema, describiendo las distintas etapas y tareas que fueron realizadas. Para organizar el trabajo se elaboró un backlog con las historias de usuario correspondientes y se planificaron varios sprints que permitieron avanzar de manera iterativa e incremental. Además, se presenta la arquitectura del software que sustenta la solución y se explica cómo se implementa cada uno de los componentes que conforman el sistema, abarcando tanto los aspectos de backend, frontend y base de datos, como la integración entre ellos.

**Backlog y Planificación de los Sprint**

Para la planificación y organización del desarrollo del sistema se elaboró un backlog (Alaimo, 2017) que incluyó las siguientes historias de usuario (HU) (Alaimo, 2017), distribuidas en distintos sprints según su prioridad y complejidad.

| | |
|---|---|
| Sprint 1 | Alta de Empleados.<br>Generación de QR.<br>Lectura de QR. |
| Sprint 2 | Inicio de Sesión.<br>Cierre de Sesión.<br>Validación de QR.<br>Manejo de QR inválido.<br>Registro de Asistencias. |
| Sprint 3 | Consulta de Empleados.<br>Modificación de Empleados.<br>Baja lógica de Empleados<br>Consulta de Asistencias. |

**Puesta a Punto del Entorno de Desarrollo**

Previo al inicio del desarrollo, se realizó la puesta a punto del entorno de trabajo, configurando todas las herramientas necesarias para garantizar un flujo de desarrollo ágil y ordenado. Esto incluyó la instalación y configuración del entorno de desarrollo integrado (IDE), la creación de los proyectos base para los distintos componentes del sistema (frontend, backend y base de datos), la definición del control de versiones con Git (Chacon & Straub, 2014), y la integración con un repositorio remoto. Además, se prepararon los entornos locales con las dependencias necesarias, como el servidor de aplicaciones y base de datos requeridas, asegurando así que todos los módulos pudieran ser desarrollados, probados y desplegados de manera coherente y reproducible.

Para el desarrollo del código se utilizaron distintos entornos de desarrollo integrado (IDE) según la naturaleza de cada componente del sistema: Visual Studio Code (Microsoft, 2024) para el desarrollo del frontend, IntelliJ IDEA Community Edition (JetBrains, 2025) para los microservicios backend, y Android Studio para la aplicación móvil. Asimismo, se configuró el control de versiones con Git, estableciendo un repositorio remoto (en GitLab) para facilitar la colaboración y el seguimiento de cambios.

Para la gestión de la base de datos del sistema, se instaló PostgreSQL 17 (The PostgreSQL Global Development Group, 2024) como motor principal de almacenamiento, elegido por su robustez, rendimiento y compatibilidad con entornos empresariales. La administración de las bases de datos se realizó mediante la herramienta pgAdmin 4 (pgAdmin Development Team, 2024), que permitió crear, gestionar y monitorear las estructuras, tablas y relaciones de forma visual e intuitiva. Esta configuración facilitó las tareas de mantenimiento,

ejecución de consultas y verificación del correcto funcionamiento durante las distintas etapas del desarrollo.

Los proyectos base correspondientes a los microservicios del sistema fueron creados utilizando el framework Spring Boot (VMware, 2024), el cual permitió establecer una estructura inicial sólida y modular para cada servicio. Gracias a su enfoque de configuración simplificada y su amplia integración con diversas tecnologías del ecosistema Spring, fue posible acelerar el desarrollo, estandarizar la arquitectura y facilitar el despliegue independiente de cada componente. Esta base sirvió como punto de partida para implementar las funcionalidades específicas de cada microservicio, garantizando coherencia y mantenibilidad en toda la solución.

Los proyectos base para los frontends web fueron creados utilizando React (Meta, 2024) junto con Vite (Vite Team, 2024) como herramienta de construcción y desarrollo. Esta combinación permitió disponer de un entorno moderno, liviano y altamente eficiente, optimizando los tiempos de compilación y la recarga en caliente durante el desarrollo. React facilitó la creación de interfaces dinámicas y reutilizables mediante componentes, mientras que Vite aportó una configuración simple y un rendimiento superior, permitiendo mantener un flujo de trabajo ágil y productivo en la implementación de las vistas y funcionalidades del sistema.

El proyecto base para la aplicación móvil fue creado utilizando Kotlin (JetBrains 2025. Kotlin), un lenguaje moderno, seguro y totalmente interoperable con Java, recomendado para el desarrollo en Android. A partir de esta base, se construyeron las distintas pantallas y funcionalidades de la aplicación, asegurando una integración fluida con los servicios backend y una experiencia de usuario consistente y eficiente.

**Arquitectura de la Solución**

En la ***Figura 1*** se muestra una vista general de la arquitectura del sistema, en la cual se representan los principales componentes y contenedores que interactúan entre sí para dar forma a la solución integral.

**Flujo Principal del Sistema - Registro de Asistencia**

En la ***Figura 2*** se muestra el flujo principal del sistema, el cual va desde el logueo en la app móvil de un empleado hasta el registro de su asistencia en el sistema seguido de la actualización del código QR.

**Desarrollo de los Componentes del Frontend**

El sistema de presentismo con QR comprende el desarrollo de tres componentes frontend, los cuales serán descritos a continuación.

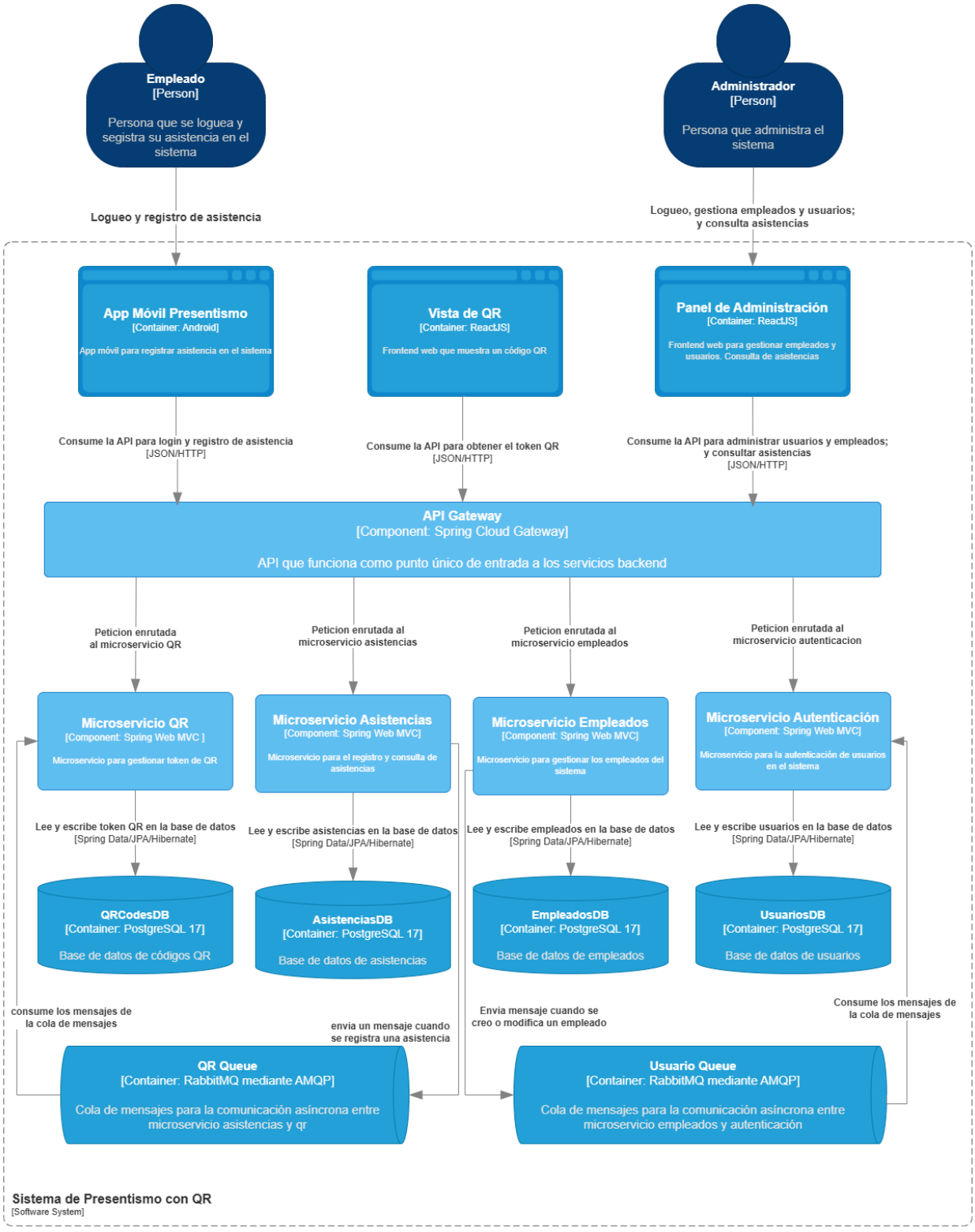


*Figura 1: Arquitectura del Sistema*

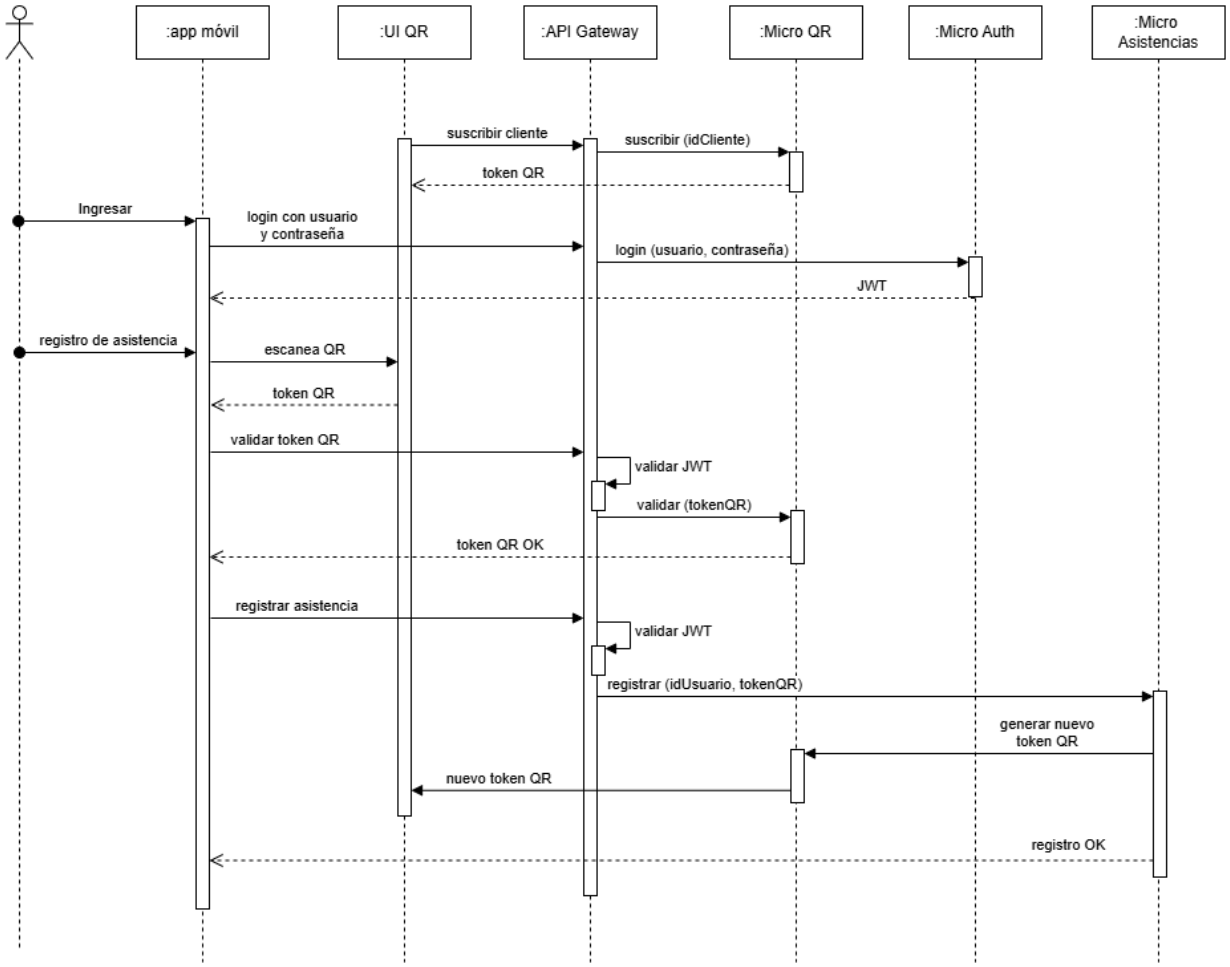


*Figura 2: Diagrama de Secuencia del Registro de Asistencia*

**Componente Vista de QR**

Este componente tiene como finalidad mostrar un código QR que puede ser escaneado por los empleados para registrar su asistencia en el sistema. Para satisfacer este requerimiento, se desarrolló una aplicación web utilizando React, una de las tecnologías más utilizadas actualmente para la construcción de interfaces de usuario interactivas, dinámicas y reutilizables. Además, React facilita la comunicación con el backend mediante el consumo de datos a través de HTTP/JSON, lo cual resulta especialmente conveniente cuando el backend está implementado con Spring Boot.

La generación del código QR en el navegador se realizó mediante la librería qrcode.react (O'Shannessy, 2024), una herramienta práctica y eficiente para crear códigos QR directamente en el frontend. Esta librería permite generar la imagen del código a partir de un token que contiene la información necesaria, sin requerir procesamiento adicional.

Si bien se evaluó la posibilidad de generar la imagen del código QR en el backend y enviarla al frontend para su visualización, este enfoque resultaba menos eficiente en términos de rendimiento y uso de recursos. Por este motivo, se optó por enviar únicamente el token y generar el código QR en el cliente.

Una vez que el código QR es escaneado por un empleado y se completa el registro en el sistema, este debe actualizarse con un nuevo valor. Para lograrlo, se implementó una comunicación basada en eventos mediante el uso de Server-Sent Events (SSE)(WHATWG, 2024), lo que permite que el cliente web permanezca suscrito y reciba automáticamente el nuevo token generado por el backend, garantizando así la actualización dinámica del código QR mostrado en pantalla.

**Componente Panel de Administración**

Se desarrolló una aplicación web utilizando React, con el objetivo de proporcionar al usuario administrador una herramienta moderna, ágil e intuitiva para la gestión integral del sistema. Desde esta aplicación, el administrador puede realizar operaciones de gestión de empleados, gestión de usuarios y consulta de asistencias, centralizando las principales tareas administrativas en una única interfaz unificada.

El sistema cuenta con un módulo de autenticación segura, que requiere el inicio de sesión mediante credenciales válidas. Solo los usuarios con el rol de administrador pueden acceder a esta aplicación, lo que garantiza la protección de los recursos sensibles. Una vez validado el inicio de sesión, el backend genera un token de autenticación (JWT)(Jones, Bradley, & Sakimura, 2015), el cual se almacena temporalmente en el cliente y se envía en cada solicitud posterior. Este mecanismo permite mantener la sesión activa de forma segura, autenticando cada petición sin necesidad de reingresar las credenciales.

En cuanto al diseño de la interfaz, se adoptó una estructura basada en componentes reutilizables, aprovechando las ventajas del enfoque modular de React. Para el estilo y la maquetación visual, se utilizó el framework Bootstrap (The Bootstrap Authors, 2024), que permitió aplicar un diseño moderno, adaptable y coherente en todos los elementos de la aplicación, facilitando además la implementación de un diseño responsive para distintos dispositivos. La aplicación incluye una barra lateral de navegación (Sidebar), que agrupa las distintas secciones del sistema —empleados, usuarios y asistencias—, y un componente principal (Main) donde se muestra el contenido correspondiente. Asimismo, se incorporaron los componentes Header, Footer e Inicio, los cuales aportan estructura y coherencia visual a la interfaz. El Header muestra información general del sistema y las opciones de usuario; el Footer contiene información institucional y enlaces secundarios; mientras que el componente Inicio cumple un rol central en la lógica de navegación, ya que determina dinámicamente qué contenido se muestra en la sección principal según la opción seleccionada en la barra lateral.

El uso de React permitió implementar un enfoque modular, escalable y mantenible, favoreciendo el desarrollo ágil y la incorporación futura de nuevas funcionalidades. De esta manera, se obtiene una aplicación eficiente, de fácil mantenimiento y con una experiencia de usuario fluida y coherente con los estándares modernos de desarrollo web.

**Componente App Móvil Presentismo**

Se desarrolló una aplicación móvil utilizando Kotlin, con el propósito de permitir que los empleados puedan registrar su asistencia de manera rápida, segura y eficiente. Esta aplicación complementa el sistema web administrativo, facilitando la interacción directa del personal mediante un dispositivo móvil.

La aplicación cuenta con un módulo de autenticación, que requiere que el empleado inicie sesión con sus credenciales personales antes de acceder a las funcionalidades principales. Una vez validado el inicio de sesión, el servidor genera un token de autenticación (JWT), que se almacena de forma segura en el dispositivo. Este token es reutilizado en todas las peticiones posteriores hacia el backend, garantizando la autenticación persistente del usuario sin necesidad de iniciar sesión nuevamente, mientras la sesión permanezca activa.

La pantalla principal de la aplicación presenta una interfaz simple y funcional, compuesta por dos botones principales:

- Registrar Asistencia, que permite al empleado iniciar el proceso de marcado de ingreso o salida.
- Cerrar Sesión, que finaliza la sesión actual y elimina el token almacenado para proteger la seguridad del usuario.

Cuando el usuario selecciona la opción Registrar Asistencia, la aplicación activa la cámara del dispositivo para escanear un código QR generado por el sistema web. Una vez obtenido el código, la aplicación valida su contenido y envía los datos al backend para registrar la asistencia del empleado. El sistema responde indicando si el registro fue exitoso o si ocurrió algún error, mostrando un mensaje informativo al usuario. Esta retroalimentación inmediata mejora la experiencia y brinda transparencia en el proceso de registro.

Para la comunicación con el backend, se implementó una arquitectura basada en servicios utilizando Retrofit (Square, Retrofit), una librería ampliamente utilizada en el desarrollo Android para el consumo de APIs REST. El proyecto define tres componentes principales:

- ApiClient, encargado de configurar la conexión y crear instancias de Retrofit.
- ApiService, que define las interfaces con los endpoints del backend (por ejemplo, autenticación y registro de asistencia).
- RetrofitClient, que gestiona la creación del cliente HTTP y la serialización de datos JSON.

Esta estructura modular facilita la mantenibilidad, reutilización y escalabilidad del código, además de garantizar una comunicación segura y eficiente con el servidor a través de la API Gateway, que actúa como punto de acceso central a los distintos microservicios del sistema.

La aplicación móvil, desarrollada en Kotlin con un enfoque orientado a componentes y buenas prácticas de arquitectura, permite una integración fluida con el backend y ofrece una experiencia de usuario optimizada para el registro diario de asistencias. Su diseño simple, combinado con la validación mediante código QR, reduce los tiempos de interacción y minimiza los errores humanos en el proceso.

Los tres frontends desarrollados se comunican con el sistema a través de una API Gateway. Esta actúa como un punto de entrada único para todas las solicitudes provenientes del entorno cliente, gestionando la autenticación, el enrutamiento y la seguridad de las comunicaciones hacia los distintos microservicios que conforman el backend. De esta manera, se garantiza una interfaz unificada y segura entre los frontends y los servicios internos, simplificando la arquitectura y mejorando la escalabilidad del sistema.

**Componente API Gateway**

El componente API Gateway constituye el punto de acceso central del sistema y cumple un rol fundamental dentro de la arquitectura de microservicios. Su principal función es recibir, procesar y redirigir todas las solicitudes provenientes de los distintos frontends hacia los microservicios correspondientes del backend.

Para su implementación se utilizó Spring Cloud Gateway (VMware. 2024), una herramienta moderna y flexible del ecosistema Spring, diseñada específicamente para el enrutamiento de peticiones y la gestión de políticas de seguridad, autenticación y control de tráfico. Este componente actúa como un proxy inteligente, permitiendo definir rutas dinámicas y aplicar filtros antes y después del procesamiento de cada solicitud.

En cuanto a la seguridad, el gateway integra Spring Security (VMware. 2024) junto con el protocolo OAuth2 (Hardt, 2012), lo que garantiza un acceso controlado y seguro a los recursos del sistema. Mediante este mecanismo, cada solicitud debe incluir un token de acceso (JWT) emitido por el servidor de autenticación, el cual es validado por el gateway antes de permitir el acceso a los servicios internos. Este proceso asegura que solo los usuarios y aplicaciones debidamente autenticados y autorizados puedan interactuar con los microservicios.

Además, el uso de Spring Cloud Gateway permite aplicar filtros globales y filtros específicos por ruta, que posibilitan tareas como el registro de logs, la auditoría de solicitudes, la verificación de cabeceras, la transformación de respuestas o el control de tasas de acceso (rate limiting). Estas características mejoran la observabilidad, trazabilidad y resiliencia del sistema frente a cargas variables o intentos de acceso indebido.

Gracias a esta arquitectura, la API Gateway se convierte en el componente intermediario esencial entre los clientes y los microservicios, brindando una capa adicional de seguridad, abstracción y escalabilidad. Su implementación con Spring Cloud Gateway y OAuth2 contribuye a mantener un diseño limpio, centralizado y alineado con las buenas prácticas de la arquitectura de microservicios moderna.

**Desarrollo de los Microservicios del Backend**

**Microservicio QR**

El microservicio QR tiene como responsabilidad principal la generación, gestión y actualización de los códigos QR utilizados por los empleados para registrar su asistencia. Este componente constituye una parte clave del ecosistema de microservicios, ya que permite vincular el proceso de registro presencial con el sistema de control de asistencias de forma automática y segura.

Para su desarrollo se empleó Spring Boot, integrando diversos módulos del ecosistema Spring, entre ellos Spring Web MVC, Spring Data JPA y Spring Security. (VMware. 2024)

Spring Web MVC se utilizó para la exposición de endpoints REST, permitiendo que los distintos clientes (frontend web y aplicación móvil) puedan interactuar con el microservicio de forma estandarizada mediante solicitudes HTTP.

Spring Data JPA, junto con Hibernate (Bauer & King, 2015), se implementó para la persistencia de datos, facilitando la interacción con la base de datos y el manejo de las entidades relacionadas con los tokens QR generados.

En cuanto a la seguridad, se configuró Spring Security para restringir el acceso únicamente a usuarios o servicios autenticados, permitiendo así que solo los componentes autorizados (por ejemplo, el microservicio de asistencias o la API Gateway) puedan interactuar con sus recursos.

El microservicio también incorpora RabbitMQ (RabbitMQ, 2024) como sistema de mensajería, lo que posibilita la comunicación asincrónica entre los distintos componentes del sistema. En particular, el microservicio QR se suscribe a una cola donde recibe notificaciones del microservicio de asistencias cada vez que se registra un nuevo evento de asistencia. Al recibir este mensaje, el microservicio QR procesa la información, invalida el token anterior y genera un nuevo token, el cual será representado mediante un nuevo código QR. Este diseño desacopla los servicios y mejora la escalabilidad y la tolerancia a fallos del sistema.

Para mantener sincronizado el código QR mostrado en el frontend, el microservicio QR implementa un mecanismo de comunicación en tiempo real mediante Server-Sent Events (SSE). A través de este canal, el frontend permanece suscrito a los eventos emitidos por el microservicio, lo que permite actualizar dinámicamente el código QR cada vez que se genera un

nuevo token. Este enfoque ofrece una experiencia fluida y sin necesidad de recargar la página, garantizando que el código QR visible sea siempre válido y actualizado.

En conjunto, la combinación de Spring Boot, Spring Web MVC, Spring Data JPA, Spring Security, RabbitMQ y SSE permite que el microservicio QR cumpla eficazmente su función dentro de la arquitectura general, proporcionando una solución robusta, segura y en tiempo real para la gestión y renovación de los códigos QR utilizados en el proceso de registro de asistencias.

**Microservicio Asistencias**

El microservicio de asistencias tiene como objetivo principal registrar, consultar y administrar las asistencias de los empleados dentro del sistema. Este componente constituye una parte esencial de la arquitectura, ya que centraliza toda la información relacionada con los registros de presencia y se integra con otros microservicios para mantener la coherencia de los datos en tiempo real.

Para su desarrollo se utilizó Spring Boot, aprovechando las capacidades de Spring Web MVC, Spring Data JPA y Spring Security, junto con herramientas de mensajería y comunicación como RabbitMQ y WebClient. (VMware. 2024)

- Spring Web MVC se empleó para exponer endpoints REST, a través de los cuales los diferentes clientes y microservicios pueden interactuar con el sistema de asistencias, ya sea para registrar un nuevo evento o consultar los existentes.
- Spring Data JPA, junto con Hibernate, se utilizó para la persistencia de datos, facilitando el acceso, manipulación y almacenamiento de las asistencias registradas en la base de datos.
- En materia de seguridad, se implementó Spring Security, configurado para permitir el acceso únicamente a usuarios y servicios autenticados, garantizando así la integridad y confidencialidad de la información gestionada.

En cuanto a la comunicación entre microservicios, este componente se integra con el microservicio QR mediante RabbitMQ, adoptando un esquema de mensajería asincrónica. Cada vez que se registra una nueva asistencia, el microservicio de asistencias envía un mensaje a una cola en RabbitMQ. Dicho mensaje es consumido por el microservicio QR, que a partir de esa notificación genera un nuevo token y actualiza el código QR mostrado en el frontend. Esta arquitectura basada en eventos permite un bajo acoplamiento entre servicios y una mayor escalabilidad y tolerancia a fallos, ya que los procesos no dependen de la disponibilidad inmediata del otro microservicio.

Por otro lado, para enriquecer los datos de las asistencias, el microservicio necesita obtener información básica sobre los empleados (por ejemplo, nombre y apellido). En lugar de replicar una tabla de empleados dentro de su propia base de datos —lo cual generaría

duplicación de datos y mayores costos de mantenimiento—, se optó por consumir estos datos directamente desde el microservicio de empleados

Para ello, se implementó el cliente reactivo WebClient, que permite realizar peticiones HTTP al microservicio de empleados de manera eficiente y segura. Este enfoque garantiza que la información de los empleados siempre esté actualizada y centralizada, respetando los principios de independencia y responsabilidad única de cada microservicio.

Gracias a esta combinación de tecnologías —Spring Boot, Spring Web MVC, Spring Data JPA, Spring Security, RabbitMQ y WebClient—, el microservicio de asistencias ofrece una solución robusta, modular y escalable. Su diseño asegura la integridad de los datos, mejora la interoperabilidad entre componentes y mantiene una comunicación fluida dentro del ecosistema de microservicios.

**Microservicio Empleados**

El microservicio de empleados tiene como función principal la gestión completa del ciclo de vida de los empleados dentro del sistema. A través de este componente se pueden realizar las operaciones de alta, baja, modificación y consulta (ABMC) de los registros de empleados, siendo un pilar fundamental para la administración y el control de los datos del personal.

Para su implementación se utilizó Spring Boot, integrando los módulos Spring Web MVC, Spring Data JPA, Spring Security y RabbitMQ, siguiendo el mismo estándar tecnológico que el resto de los microservicios del sistema.

- Spring Web MVC se empleó para exponer endpoints REST que permiten a la aplicación web y a otros microservicios interactuar con el sistema de empleados de forma segura y estructurada.
- Spring Data JPA, junto con Hibernate, se utilizó para la persistencia de datos y la administración de las entidades de empleados. Se implementó una baja lógica, de manera que al desactivar un empleado este no se elimina físicamente de la base de datos, sino que se marca como inactivo. Este enfoque garantiza la integridad referencial y permite conservar el historial de registros.
- Spring Security se configuró para controlar el acceso a los recursos expuestos, asegurando que solo los usuarios y servicios autenticados puedan realizar operaciones sobre los datos de los empleados.

En cuanto a la comunicación asincrónica, el microservicio de empleados se integra con el microservicio de autenticación mediante RabbitMQ. Cada vez que se registra un nuevo empleado, este microservicio envía un mensaje a una cola conteniendo información básica del nuevo registro —como el correo electrónico y el número de documento (DNI)—. El microservicio de autenticación consume este mensaje y, con esos datos, procede a crear automáticamente las credenciales de acceso del nuevo usuario en el sistema, utilizando el correo electrónico como nombre de usuario y el DNI como contraseña inicial.

Este mecanismo de mensajería desacoplada permite automatizar el proceso de creación de usuarios en el sistema de autenticación, garantizando la coherencia entre los módulos y reduciendo la intervención manual. Además, el uso de RabbitMQ mejora la escalabilidad y la resiliencia, ya que las operaciones no dependen de la disponibilidad inmediata del servicio consumidor.

Gracias a esta combinación de tecnologías —Spring Boot, Spring Web MVC, Spring Data JPA, Spring Security y RabbitMQ—, el microservicio de empleados ofrece una solución robusta, modular y segura para la gestión de datos del personal. Su diseño promueve la integridad de la información, la automatización de procesos y la interoperabilidad con otros componentes del sistema, manteniendo la coherencia general dentro de la arquitectura de microservicios.

**Microservicio Autenticación**

El microservicio de autenticación es el componente encargado de gestionar el acceso y la seguridad de los usuarios dentro del sistema. Su función principal es verificar las credenciales de los usuarios, generar tokens de acceso y validar la autenticación en cada solicitud proveniente de los distintos frontends y microservicios. Este módulo constituye un elemento clave en la arquitectura, ya que garantiza la protección de los recursos y la correcta aplicación de las políticas de seguridad.

Para su desarrollo se utilizó Spring Boot, integrando los módulos Spring Web MVC, Spring Security, Spring Data JPA, Hibernate, RabbitMQ y JSON Web Tokens (JWT) para el manejo de autenticaciones sin estado (stateless).

- Spring Web MVC se empleó para exponer endpoints REST que permiten realizar operaciones como el inicio de sesión, validación de tokens y gestión básica de usuarios autenticados.
- Spring Data JPA junto con Hibernate se utilizaron para la persistencia de usuarios y roles en una base de datos propia, lo que permite un control total sobre la administración de credenciales y permisos sin depender de servicios externos.
- Spring Security se configuró para proteger los endpoints del sistema, validando las credenciales en el proceso de autenticación y verificando la validez de los tokens JWT en cada solicitud posterior.

El proceso de autenticación se basa en el uso de JSON Web Tokens (JWT), que permiten mantener una comunicación segura entre el cliente y el servidor sin necesidad de mantener sesiones persistentes. Una vez que un usuario ingresa sus credenciales válidas, el microservicio genera un token JWT que incluye la información del usuario autenticado y su rol dentro del sistema. Este token se devuelve al cliente y debe ser incluido en el encabezado de cada petición futura, permitiendo así el control de acceso distribuido a través de la API Gateway.

Además, el microservicio se integra con RabbitMQ para comunicarse con el microservicio de empleados, permitiendo la creación automática de usuarios. Cuando el microservicio de empleados registra un nuevo empleado, envía un mensaje a una cola de RabbitMQ con los datos básicos del mismo —correo electrónico y número de documento (DNI)—. El microservicio de autenticación consume ese mensaje, crea el registro correspondiente en su base de datos y genera las credenciales de acceso iniciales, estableciendo el correo electrónico como nombre de usuario y el DNI como contraseña predeterminada. Este proceso se realiza de manera automática, asegurando la sincronización entre ambos servicios y reduciendo la intervención manual.

El uso conjunto de Spring Security, JWT y RabbitMQ garantiza una arquitectura segura, eficiente y desacoplada, en la que la autenticación y la autorización se gestionan de forma centralizada pero sin comprometer la independencia de los microservicios. Gracias a esta implementación, el microservicio de autenticación actúa como autoridad de seguridad del sistema, proporcionando una base sólida para el control de accesos, la emisión de tokens y la validación de identidades dentro de toda la infraestructura distribuida.

**Bases de Datos Independientes**

Cada microservicio cuenta con su propia base de datos independiente, lo que permite mantener un aislamiento total de los datos y evitar dependencias directas entre los distintos componentes del sistema. Esta decisión se alinea con las buenas prácticas de diseño en arquitecturas de microservicios, promoviendo la autonomía y escalabilidad de cada módulo. PostgreSQL fue seleccionada por su robustez, cumplimiento con estándares SQL y su excelente integración con Spring Boot, a través de los módulos Spring Data JPA e Hibernate, que simplifican el mapeo objeto-relacional y el manejo transaccional. Gracias a esta combinación, cada microservicio puede gestionar de manera eficiente sus propios datos, garantizando consistencia, rendimiento y facilidad de mantenimiento en toda la solución.

## Conclusiones

El desarrollo del Sistema de Presentismo con QR representó una experiencia integral de aplicación práctica de los conocimientos adquiridos a lo largo de la carrera, abarcando todas las etapas del desarrollo de software full-stack, desde el diseño de la arquitectura hasta la implementación y validación final del sistema. La elección de una arquitectura basada en microservicios permitió alcanzar una solución modular, escalable y fácilmente mantenible, donde cada componente cumple una función específica y se comunica de forma segura y eficiente a través de una API Gateway centralizada.

El uso de tecnologías modernas —Spring Boot para el backend, React con Vite y Bootstrap para los frontends web, Kotlin para la aplicación móvil y PostgreSQL 17 como base de datos— garantizó un sistema seguro, confiable y de alto rendimiento, alineado con las buenas prácticas de la ingeniería de software actual. La incorporación de mecanismos de autenticación mediante JWT, la comunicación asincrónica con RabbitMQ y la adopción de un modelo de bases de datos independientes fortalecieron la seguridad, integridad y consistencia de la información.

Asimismo, se priorizó la usabilidad y la experiencia del usuario, logrando interfaces modernas, intuitivas y adaptables a diferentes dispositivos. El empleo de códigos QR dinámicos permitió eliminar las limitaciones de los sistemas tradicionales, reduciendo costos operativos, errores humanos y riesgos de suplantación, al tiempo que se aportó una solución tecnológica moderna, eficiente y sostenible, alineada con la digitalización verde y la reducción del uso de materiales físicos.

En definitiva, el sistema desarrollado demuestra la viabilidad y efectividad de integrar tecnologías contemporáneas bajo un enfoque distribuido y orientado a servicios, ofreciendo una herramienta innovadora para la gestión de asistencia en entornos empresariales. Este trabajo no solo evidencia el dominio técnico alcanzado, sino también la capacidad para diseñar, implementar y desplegar soluciones robustas, seguras y adaptadas a las necesidades actuales de transformación digital.

## Referencias


- Biblioteca de la CEPAL. Comisión Económica para América Latina y el Caribe (CEPAL). (2022, 3 noviembre). Qué es Código QR – Qué son los Códigos QR. En Biblioguías (Biblioteca de la CEPAL, Naciones Unidas). https://biblioguias.cepal.org/QR
- Wambua, M. S. (2023). Modern Android 13 development cookbook: Over 70 recipes to solve Android development issues and create better apps with Kotlin and Jetpack Compose (1.ª ed.). Packt Publishing.
- IBM. (2025, 24 de abril). ¿Qué es una API REST (API RESTful)? Recuperado de IBM website. IBM
- Ford, N., Parsons, R., & Kua, P. (2021). Software architecture: The hard parts: Modern trade-off analyses for distributed architectures. O'Reilly Media.
- UNEP. (2024, 12 de noviembre). Digitalization for sustainability. Recuperado de UN Environment Programme website. UNEP - UN Environment Programme
- Alaimo, D. M. (2017). Proyectos ágiles con Scrum: flexibilidad, aprendizaje, innovación y colaboración en contextos complejos (1.ª ed.). Buenos Aires: Granica.
- Chacon, S., & Straub, B. (2014). Pro Git (2nd ed.). Apress.

- Microsoft. (2024). Visual Studio Code (Versión 1.106.3) code.visualstudio.com
- JetBrains. (2025). IntelliJ IDEA Community Edition (2024.2.1) www.jetbrains.com/idea/
- The PostgreSQL Global Development Group. (2024). PostgreSQL (Versión 17) www.postgresql.org/
- pgAdmin Development Team. (2024). pgAdmin 4 (Versión 9.6) www.pgadmin.org/
- VMware. (2024). Spring Boot (Versión 3.x) spring.io/projects/spring-boot
- Meta. (2025). React (Versión 19.1) https://react.dev/
- Vite Team. (2025). Vite (Versión 7) https://vitejs.dev/
- JetBrains. (2024). Kotlin (Versión 2.0.21) kotlinlang.org/
- O'Shannessy, P. (2024). qrcode.react (Versión 4.2) www.npmjs.com/package/qrcode.react
- WHATWG. (2024). Server-Sent Events — HTML Living Standard. html.spec.whatwg.org/multipage/server-sent-events.html
- Jones, M., Bradley, J., & Sakimura, N. (2015). JSON Web Token (JWT) (RFC 7519). Internet Engineering Task Force. datatracker.ietf.org/doc/html/rfc7519
- The Bootstrap Authors. (2025). Bootstrap (Versión 5.3.8) getbootstrap.com/
- Square, Retrofit. A type-safe HTTP client for Android and Java. square.github.io/retrofit/
- Hardt, D. (2012). The OAuth 2.0 Authorization Framework. IETF RFC 6749. datatracker.ietf.org/doc/rfc6749/
- RabbitMQ. (2024). RabbitMQ Documentation. VMware, Inc. www.rabbitmq.com/
- Bauer, C., & King, G. (2015). Java Persistence with Hibernate (2nd ed.). Manning Publications.