Control horario del personal de una empresa: Propuesta de solución no biométrica, bajo coste y bajo impacto.

Proyecto fin de carrera de Ingeniería Informática - 2016
Universidad de Las Palmas de Gran Canaria

Tutor: Alexis Quesada Arencibia
Alumno: Néstor Angulo de Ugarte

Proyecto fin de carrera de Ingeniería Informática de la Universidad de Las Palmas de Gran Canaria presentado por el alumno:

Néstor Angulo de Ugarte

Título del proyecto:

Control horario del personal de una empresa: Propuesta de solución no biométrica, bajo coste y bajo impacto.

Tutor:

Alexis Quesada Arencibia.

Fecha: Mayo de 2016

Agradecimientos

Me gustaría reconocer aquí el trabajo de mi tutor Alexis Quesada Arencibia, así como la de todos los profesores de la escuela de Ingeniería Informática que han hecho que me interese realmente por lo que estaba aprendiendo, y me han ayudado a cimentar los conocimientos prácticos que luego se desarrollan en el mundo laboral. El trabajo de un docente es, en mi opinión, una labor de vital importancia y determinante en el futuro de los alumnos, y en este caso ha sido excelente.

Además, quisiera agradecer la colaboración de Aday Talavera Hierro, ex-compañero de la carrera, que, durante el desarrollo de este producto en el que se basa esta memoria ayudó en ciertas partes del desarrollo y los procesos de brainstorming.

En la misma línea, extiendo este agradecimiento a los diseñadores María Moreno Córdoba y Javier González Vega, que aportaron su gran maestría y creatividad en el proceso de la imagen corporativa y la carcasa física del sensor.

Por otro lado, es menester agradecer también el gran apoyo del Hermano Juan, director de la Casa Hogar San Miguel durante el periodo de relacionado con esta memoria. Miembro de la congregación católica de los Hermanos de la Cruz Blanca, la cual desarrolla una labor humanitaria que no tiene precio, y que creyó en este proyecto. Es difícil encontrar personas del mundo eclesiástico tan entusiastas pro-tecnología, y que además supo tener la paciencia adicional que el desarrollo de este tipo de proyectos requiere.

Por supuesto, sin mis padres y mi hermano esto no habría sido posible, y han estado siempre para apoyarme, en lo bueno y en lo malo, así como todas aquellas personas que me han demostrado repetidamente su cariño y amor, aderezado con una alta dosis de paciencia, durante este largo camino. Y seguro que seguirán ahí para continuar haciéndolo a posteriori de este hito, el cual les dedico a ellos. No os merezco.

Por último, también quisiera dedicar este momento de mi vida al que posiblemente es una de las más importantes figuras que me han inspirado durante los últimos años académicos, el Dr. Manuel Galán Moreno, cuya labor formadora fue más allá de lo estrictamente docente, enseñándome constantemente lecciones de vida y para la vida. Profesor de la Universidad de Las Palmas de Gran Canaria, y secretario del departamento de investigación del CICEI, sin ser mi profesor directamente, me enseñó con su empeño y su tenacidad habitual, diría incluso cabezonería, a pesar de mis constantes tropiezos. Si soy algo en un futuro próximo, en gran parte será debido a él.

Gracias.

Descanse en paz.

A Manuel Galán Moreno,

profesor de la Universidad de Las Palmas de Gran Canaria,

doctor por el departamento de Matemáticas,

pero sobre todo, un Maestro.

Descanse en Paz. RIP 1 de Enero de 2014

Índice de imágenes y diagramas

Índice de Tablas

Índice de Código

Índice del documento

1. Introducción

La presente memoria ha de tomarse como un fotografía del estado de un proyecto iterativo e incremental, detallando el análisis, diseño e implementación de un sistema de control de presencia de trabajadores aplicado a una necesidad real de una empresa del sur de Tenerife, la cual está dedicada a la formación, control y cuidado de internos con alta dependencia por alta discapacidad psíquica.

Dicha empresa, la Casa Familiar San Miguel[1], sita en el polígono industrial de Las Chafiras, en San Miguel de Abona, Tenerife, es regentada por la congregación religiosa de los Hermanos Franciscanos de La Cruz Blanca[2], dependiente de la Iglesia Católica. Como casi todas las organizaciones de este tipo, se financian prácticamente a base de donativos, obras sociales de empresas y subvenciones gubernamentales.

Debido a que son responsables de la salud y formación de personas con alta dependencia, surge la necesidad de controlar mejor al personal de cara a un correcto funcionamiento del centro, así como una gestión de informes de cumplimiento, horas extras y días libres más efectiva. En el momento del desarrollo del presente proyecto, la metodología aplicada era a base de firma manuscrita en hojas impresas de turnos, por lo que requería un proceso manual de recopilación, cálculo y digitalización de dicha información de manera mensual. Por tanto, un proceso automático digitalizado aportaría indudablemente un ahorro considerable en tiempo y otras ventajas.

Además, requerían actualizar sus sistemas de control de sus trabajadores para adaptarse a las normativas vigentes, siendo el Real Decreto Legislativo 2/2015, del 23 de Octubre,[3] en su sección 5º, apartados 7 y 9, la última actualización de la norma por esas fechas, que obliga a las empresas a registrar adecuadamente la jornada laboral diaria de cada trabajador, así como el control de horas extraordinarias y días libres. Sin embargo, cabe destacar que dicho decreto no especifica la forma en la que dicho control debe llevarse a cabo, solo que la información debe almacenarse por 4 años. A la fecha de entrega de esta memoria de de Proyecto Fín de Carrera, se acaba de aprobar la última revisión de estas normativas en formato Real Decreto-ley 8/2019, de 8 de marzo, “medidas urgentes de protección social y de lucha contra la precariedad laboral en la jornada de trabajo”, donde se recoge, de manera resumida estos puntos:

- El Art. 12, establece: “la jornada de los trabajadores a tiempo parcial se registrará día a día y se totalizará mensualmente, entregando copia al trabajador, junto con el recibo de salarios, del resumen de todas las horas realizadas en cada mes, tanto las ordinarias como las complementarias”.
- El Art. 34, “Será de obligación de la empresa registrar la jornada de cada trabajador de forma fehaciente. Este registro de jornada será diario y deberá incluir el horario concreto de entrada y salida”.

[1] http://cruzblanca.org/casas.htm
[2] http://cruzblanca.org/
[3] https://www.boe.es/buscar/act.php?id=BOE-A-2015-11430

- El Art. 35, “la obligatoriedad de que cada trabajador reciba mensualmente la relación de horas ordinarias y extraordinarias que ha realizado en dicho mes, también los representantes sindicales y comités de empresa recibirán esta información de toda la plantilla”.

La normativa obliga también a mantener los registros 4 años. En cualquier caso, los sprints programados al final vienen a cubrir las necesidades nuevas de esta norma.

Los sistemas de este tipo presentes en el mercado en el momento del desarrollo del producto responden, en todos los casos, a sistemas con arquitectura cliente - servidor, donde el cliente consiste en un sensor desde el que el trabajador genera lo que denominaremos durante esta memoria como *check-in*, *check-out* y *break*, o simplemente *checks*, es decir, deja constancia en el sistema de su entrada en la empresa o sede, su salida y sus descansos. El servidor a su vez, es el centro de procesamiento y almacenamiento de dichos datos. En algún caso, se ha encontrado soluciones donde dichas vertientes son realizadas en el mismo dispositivo físico pero en procesos separados, aunque lo que predomina son soluciones donde estas dos partes se separan físicamente. En nuestra aproximación se ha optado por un sistema completamente separado en dispositivos físicos diferentes y no interdependientes, para facilitar la modularidad del sistema y su robustez.

Se ha de tomar en cuenta que, por la naturaleza práctica del proyecto, el desarrollo se han seguido metodologías de desarrollo AGILE, y esta memoria es el reflejo de una de esos lanzamientos congelado en el momento concreto que se comenzó el desarrollo esta.

2. Objetivos

Según se ha introducido anteriormente, el sistema desarrollado se hubo de realizar sobre los siguientes **requisitos por parte del cliente**:

- **Bajo coste**
  Se requería que los costes del sistema fueran contenidos y por debajo del precio de la oferta de paquetes comerciales actuales debido a la forma en la que la organización en cuestión se financia.
- **Bajo impacto**
  Parte del interés de este proyecto era generar una solución cuya implantación implicara el mínimo impacto en la rutina general, facilitando al máximo tanto el registro como su administración y optimizando el tiempo invertido, además de los conocimientos necesarios para el uso de la plataforma. La instalación física del sistema debía ser llevada a cabo también empleando la infraestructura de base de la empresa, tratando de utilizarla sin generar nuevas necesidades.
- **No biométrica**
  En contra de la dirección que están tomando estos sistemas actualmente, en las que la veracidad y coherencia en sus datos pasa por la incorporación de sensores biométricos, se propone una solución cuyo nivel de coherencia es similar y no implica un análisis biométrico automatizado, aunque, como veremos más adelante, abre la puerta a la incorporación de dicha característica en su proceso.
- **Robustez y seguridad**
  Era de interés de la propia empresa que la solución a instalar poseyera un cierto nivel de robustez en su información, intentando que fuera lo menos vulnerable posible a la “picaresca” durante el proceso de registro, almacenamiento y gestión. Además se adoptarán medidas de seguridad para garantizar en la medida de lo posible el correcto almacenamiento de la información así como prevenir posibles acciones, intencionadas o no, que puedan poner en peligro el sistema o desvirtuar dicha información.

Por otro lado, se establecen estos **otros requerimientos** autoimpuestos, relacionados mayoritariamente con la **calidad del software**:

- **Mantenibilidad y Software Libre**
  Se procurará aplicar el máximo nivel de modularidad al que se pueda comprometer el diseño con el fin de facilitar la mantenibilidad del sistema y sus actualizaciones. Se priorizará las soluciones de Software abiertas, así como la garantía de libre disponibilidad del software y los datos, en contraposición a la estrategia predominante en las empresas de este sector donde intentan siempre trabajar con un software cerrado con los datos inaccesibles, de forma que cualquier servicio adicional pase por un servicio de consultoría. A tenor de esto, se tratará de usar las librerías de terceros que mejor se adapten a las necesidades que estén más ampliamente mantenidas por sus creadores y colaboradores, garantizando sus

actualizaciones y delegando parte de la responsabilidad de mantenibilidad, agilizando el proceso de *release*.

- **Portabilidad y compatibilidad**
  El sistema debe ser instalable y ejecutable intentando minimizar los requerimientos y costes en licencias, ampliando el rango de plataformas compatibles en las que pueda ejecutarse. El uso de estándares en el proceso garantizará también la mayor compatibilidad con la mayoría de estos sistemas. Todo esto redundará claramente a favor de la libertad del usuario.
- **Eficiencia**
  Se tratará que la solución sea lo más eficiente posible dadas las actuales tecnologías disponibles de electrónica embebida, tanto en consumo de recursos como en espacio.
- **Diseño, usabilidad y utilidad**
  La solución deberá ceñirse a la utilidad para la que ha sido diseñado, pero también está contemplado que la solución final represente no sólo eso y que, en definitiva, durante el tiempo que no se está utilizando para su cometido principal pueda desempeñar otras funciones. Durante el proceso de desarrollo se tratará, en la medida de lo posible, que sea lo más usable y accesible posible sin comprometer con ello la complejidad del sistema.

Además, se planifican medidas de monitorización para registrar cualquier mal funcionamiento e identificación de áreas en las que sea posible ser optimizado.

Por tanto, el objetivo es crear un sistema “cliente - servidor” de fácil uso, robusto e integrado en el propio entorno del cliente donde:

- El *cliente* será un sistema “Software + Hardware” que poseerá un sensor para que el trabajador pueda registrar sus entradas y salidas.

- El *servidor* será un sistema Software, a través del cual el responsable de Recursos Humanos del centro podrá gestionar:
  - Altas y bajas de trabajadores
  - Turnos
  - Consulta y generación manual de Checks
  - Informes de checks
- Se contempla además que el trabajador pueda entrar en el servidor también para consultar en tiempo real sus checks.

Aparte, como objetivo adicional, se diseñará dicho sistema para que pueda ser implantado fácilmente en cualquier PyME cuyas necesidades sean modestas. Por tanto, deberá ser un sistema portable e independiente pero sin perder por ello robustez en sus datos ni posibilidades de interconexión con otros sistemas, dando como resultado un sistema adaptable a escenarios variados.

3. Estructura del documento

Con el objetivo de facilitar la lectura del presente documento, se presenta en este apartado la estructura general del mismo.

Tras haber realizado la *introducción* del proyecto y después de haber definido los objetivos del mismo, se explicarán cada una de las fases por las que ha pasado el proyecto.

En primer lugar se presenta el estudio sobre el *Estado del Arte* y las herramientas existentes antes de comenzar el proyecto. A continuación, se explica la metodología utilizada durante la elaboración de la aplicación así como los recursos utilizados.

En los siguientes apartados se detallan las fases de *análisis, diseño y desarrollo del proyecto*.

Seguidamente, se puede observar los resultados obtenidos, las conclusiones y el trabajo futuro que podría realizarse para ampliar este proyecto.

Por último, se muestra la bibliografía utilizada y se presentan los anexos del documento, donde puede encontrarse el manual de usuario y la guía de instalación.

En cada una de las fases presentadas en este documento se podrán encontrar diferentes apartados. El objetivo es facilitar la lectura y comprensión. A lo largo del documento también se citarán las referencias bibliográficas consultadas.

El “Anexo A – Manual de usuario Check-IN” contiene toda la información sobre las funcionalidades de la aplicación. Además, se acompaña cada explicación con imágenes para facilitar la comprensión.

El “Anexo B – Manual de usuario del sensor Check-IN” contiene la información necesaria para hacer funcionar y entender el funcionamiento del sensor de lectura de los fichajes.

El “Anexo C – Presupuesto del sensor Check-IN” detalla los precios y modelos de los componentes del sensor para hacer un cálculo bruto de su coste.

4. Estado del Arte

El primer problema importante a solucionar cuando se trata de generar un sistema de control de presencia o control horario es cómo se identifica a un individuo de manera inequívoca, generando una relación uno-a-uno (1:1) entre la persona y el proceso que se utilice para identificarla. Esto implica 4 elementos:

- **Identidad** del individuo: Se debe primero recabar la información real del individuo necesaria para el sistema (*Identity Set up*).
- **Patrón** de autenticación: Recabado, Validación y configuración del método de autenticación acordado (*Pattern Set up*).
- **Validador**: Responsable de validación de que la identidad corresponde con el patrón de autenticación utilizado.
- **Autenticador**: Elemento que recoge el patrón y procede a la verificación de autenticación

Tradicionalmente, la manera más fiable de llevar a cabo este proceso ha sido la firma manuscrita en una hoja de presencia, que debía ser controlada, analizada y archivada en registros. En el momento de la firma, la identidad de la persona queda acreditada por un validador o testigo, que vincula la identidad con la firma (patrón), utilizando como método autenticador la comparación de las firmas.

Evidentemente, este método puede ser calificado de fiable debido a que una firma manuscrita es muy difícil de falsificar, y a la presencia de otros indicadores (como por ejemplo, la presión y color del bolígrafo utilizado) que pueden garantizar la identidad del individuo, y por ello se sigue utilizando como método principal en contratos. Sin embargo, no es menos evidente que es un método engorroso, falible (si el controlador es un humano, ya que para identificar una firma falsa se requiere una investigación especializada denominada Peritaje Caligráfico[4]) y que consume bastante tiempo y espacio.

Las diferentes maneras de identificación o autenticación de un individuo o una fuente fiable han sido siempre un área de estudio y constante evolución a lo largo de la Historia del Ser Humano. Desde el simple conocimiento de un lenguaje escrito frente al analfabetismo general, pasando por las claves criptográficas (como el bastón del cifrado de Julio César[5] o la máquina Enigma[6]) y los sellos de cera, hasta

Img. 1: Personal de una empresa fichando.

[4] https://es.wikipedia.org/wiki/Peritaje_caligr%C3%A1fico
[5] https://es.wikipedia.org/wiki/Cifrado_C%C3%A9sar
[6] https://es.wikipedia.org/wiki/Enigma_(m%C3%A1quina)

nuestros actuales certificados digitales o tecnologías biométricas. Por tanto, podemos afirmar que la criptología, biometría y otras ciencias forenses han estado muy ligadas al desarrollo de metodologías de identificación.

Sin embargo, es importante destacar que no es viable iniciar un proceso de investigación o utilizar complejas tecnologías cada vez que un trabajador entra o sale de un espacio de trabajo. Por tanto, en este campo particular del control de presencia se deben buscar metodologías en equilibrio entre robustez, tiempo y recursos económicos, dependiendo del caso concreto (es evidente que no es lo mismo controlar las entradas y salidas del personal de una empresa de mensajería, que las de los laboratorios del Departamento de Defensa de un país).

Hoy por hoy, de manera formal, para autenticar a un individuo se utiliza alguno de estos elementos o combinación de ellos:

- Algo que sólo el individuo *conoce* (ej.: un PIN, una contraseña)
- Algo que sólo el individuo *posee* (ej.: una tarjeta, un certificado digital)
- Algo que sólo el individuo *es* (ej.: huella dactilar, mapa de retina/iris)
- Algo que sólo el individuo *genera* (ej.: voz, firma, patrón de latidos)

En este orden de cosas, a principios del siglo pasado, eran comunes los sistemas de fichaje "desatendidos" como los relojes de fichaje, en los que se usaban tarjetas personales de cartón que se perforaban o imprimían. Este sistema, uno de los más ampliamente utilizados hasta nuestros días, ha evolucionado utilizando *smart cards* o tarjetas inteligentes. El gran problema de este método de tarjetas y reloj de fichaje es que la identidad de la persona que usaba la tarjeta no se podía garantizar de manera inequívoca, es decir, un compañero podría pedirle a otro que pasara su tarjeta en su nombre. Como se verá más adelante, este punto es vital en el desarrollo del presente proyecto.

Img. 2: Reloj de fichaje antiguo

Actualmente, existen muchas empresas exclusivamente dedicadas a la comercialización de productos para este fin directamente, o dentro de una *suite* de servicios integrales. Las diferentes tecnologías de interés que utilizan dichas soluciones encontradas durante el proceso de estudio del Estado del Arte se pueden compendiar en esta tabla:

Tecnología	Pro	Contra
Login (PIN o Usuario y contraseña)	+ Sencillo + Barato + Fácil implantación + Compatibilidad con todos los procesos	- Poco robusto (fácilmente eludible) - Lenta validación - Requiere aprenderlo de memoria
Tarjetas (Bandas magnéticas o	+ Sencillo + Barato	- Poco robusto (se puede prestar o clonar)

chip)	+ Fácil implantación + Rápida validación	- Objeto externo que se puede perder - La banda magnética es especialmente sensible - Elementos mecánicos
Tecnologías RFID (de aproximación)	+ Sencillo + Barato + Fácil implantación + Rápida validación	- Poco robusto (se puede prestar o clonar) - Objeto externo que se puede perder
Cámaras	+ Rápida validación + Robusto + Almacena pruebas gráficas	- Caro - Complejo (mucha tecnología anexa necesaria) - Difícil implantación
Biométrica	+ Rápida validación + Robusto + Fácil implantación	- Caro - Complejo - Información biométrica personal - Rechazo

Tabla 1: Comparativa de diferentes metodologías de identificación

A modo de resumen, en base al estudio se podría representar de esta manera:

Tecnología
Login (numérico o con usuario y contraseña)
Tarjetas de acceso de banda magnética o chip
Tecnología RFID
Cámaras con algoritmo de detección
Validación Biométrica

Tabla 2: Resumen de factores de las metodologías de identificación

Estos datos fueron recopilados de una investigación de empresas que ofrecen servicios relacionados con la temática de este proyecto, entre las que me gustaría destacar a modo resumen las siguientes seis:

Empresa	País	Tipos	Precios de partida	Web
Biosentinel	España	Huella digital, RFID, Video vigilancia, kits autoinstalables, software.	1.155€ (kit básico 20 usuarios y 1 año soporte)	biosentinel.es

Grupo Spec	España	Huella digital, RFID, Banda magnética, Lector de matrículas, Lector de DNIs , kits autoinstalables, software.	1.730€ (kit básico 50 usuarios)	grupospec.com
Suprema INC	Corea	Huella digital, RFID, lector DNI y pasaportes, software	650€ (kit básico 20 usuarios)	supremainc.com
Securtek	Canadá	Huella digital, RFID, software	545€ (kit básico)	securtek.com
SCI-Spain	España (Canarias)	Banda magnética, SmartCards, RFID, huella digital, software.	1.600€ (lector)	sci-spain.com
MHP	España (Canarias)	Huella digital, RFID, Banda magnética, PIN, lector DNI	Adaptado al cliente, ~1000€/lector y 100€/mes como licencia.	mhp.es

Tabla 3: Comparativa de diferentes empresas especializadas en control horario e identificación

En el momento del desarrollo de esta memoria, la tecnología emergente dominante a nivel comercial para la identificación personal es la relacionada con la identificación biométrica, especialmente la de huella dactilar, también llamado dermatoglifo[7]. Esta tendencia recibe su nombre de la estrategia de identificación inequívoca de un individuo basada en mediciones (métrica, de *Metron* en griego, que significa "medida") de elementos relacionados con su cuerpo orgánico (*Bio*, "vida" en griego) basándose en la premisa de que el entorno, el ADN y las experiencias modelan de manera única el crecimiento de los diferentes órganos y elementos de nuestro cuerpo, pudiéndose crear con ellos una suerte de firma o huella única. En el caso de la huella dactilar, está basada en medidas de las crestas papilares de los dedos (principalmente los dedos índice y pulgar de las manos).

He aquí una tabla resumen de las tecnologías biométricas actuales[8]:

	Ojo (Iris)	**Ojo (Retina)**	**Huellas dactilares**	**Vascular dedo**	**Vascular mano**	**Geometría de la mano**	**Escritura y firma**	**Voz**	**Cara 2D**	**Cara 3D**
Fiabilidad	Muy alta	Muy Alta	Muy Alta	Muy Alta	Muy Alta	Alta	Media	Alta	Media	Alta
Facilidad de uso	Media	Baja	Alta	Muy Alta	Muy Alta	Alta	Alta	Alta	Alta	Alta

[7] https://es.wikipedia.org/wiki/Huella_dactilar
[8] Extraída del artículo https://es.wikipedia.org/wiki/Biometr%C3%ADa

Prevención de ataques	Muy alta	Muy Alta	Alta	Muy Alta	Muy Alta	Alta	Media	Media	Media	Alta
Aceptación	Media	Baja	Alta	Alta	Alta	Alta	Muy Alta	Alta	Muy alta	Muy alta
Estabilidad	Alta	Alta	Alta	Alta	Alta	Media	Baja	Media	Media	Alta

Tabla 4: Comparativa de factores de los diferentes procesos de identificación biométrica
Tabla extraída del artículo de Wikipedia en Español: Biometría https://es.wikipedia.org/wiki/Biometr%C3%ADa

Sin embargo, y a despecho de lo que refleja esta última tabla, durante la investigación que se llevó a cabo a la hora de elegir un método eficaz en el caso práctico objeto de esta memoria, en una encuesta informal llevada a cabo con empresas que ya tenían instalado un sistema de fichaje por huella digital, se encontraron ciertos factores interesantes en contra de esta que causaban cierto rechazo. De manera esquemática, estos fueron los argumentos en contra del uso de esta tecnología que se recopilaron entre empresas que las utilizaban:

- Fiabilidad del sensor:
  Causaba errores de *no identificación* con cierta recurrencia, por lo que era obligatorio añadir un elemento externo alternativo de autenticación, normalmente un teclado numérico.
  Riesgo: Retrasos, gestión de claves

- Limpieza del sensor:
  Causaba rechazo entre cierto sector del personal ya que la obligación de posar el dedo encima de una superficie donde otras personas lo han hecho antes constantemente obligaba a evaluar parámetros de higiene. Por tanto, se requería incorporar elementos de higiene cerca del sensor y otro de limpieza del propio sensor.
  Adicionalmente, la propia limpieza del sensor afectaba a su fiabilidad.
  Riesgo: Rechazo, retrasos, gasto adicional

Al margen de esto, observamos que la inmensa mayoría de las soluciones comerciales constituyen paquetes cerrados con un software propietario, con un precio superior a las expectativas del cliente objetivo de este proyecto, y basado en licencias. Si bien es cierto que se ofrecen servicios de consultoría por parte de las empresas para adaptar los datos al sistema del cliente, pero es un precio adicional a considerar. Además, esta metodología de licencias y software propietario cerrado genera una dependencia que se también se ha de considerar a largo plazo.

5. Metodología y recursos utilizados

Por su naturaleza práctica, este proyecto fue planteado utilizando una metodología ágil[9] de desarrollo, en un marco de desarrollo de software iterativo e incremental[10]. Esto implica que se parte de una planificación, análisis de requisitos, diseño e implementación de un modelo base simple (llamado MVP[11] o mínimo producto viable) que resuelve una necesidad concreta simple, y se itera utilizando el feedback del cliente final y el monitoreo de la actividad del producto, creando un bucle en espiral creciente donde el producto crece en base a lanzamientos rápidos y evaluaciones constantes, añadiendo funcionalidades a medida que van surgiendo las necesidades, así como desarrollando su documentación a la par.

5.1. Metodologías AGILE

En 2001 se acuñó la expresión “metodologías ágiles” tras una reunión de expertos en desarrollo de software, en el que se analizó la situación y el futuro de este campo. En esta, además se firmaron los 12 principios del Manifiesto Ágil[12]:

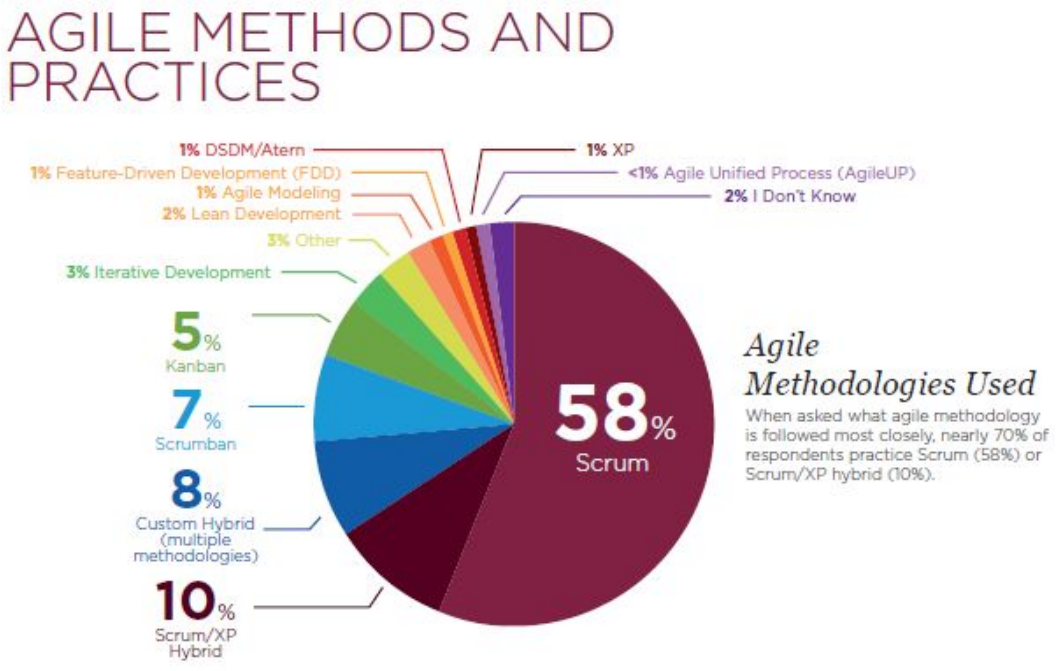
1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.

Img. 3: Bucle de la metodología Ágil

[9] https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software
[10] https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente
[11] https://es.wikipedia.org/wiki/Producto_viable_m%C3%ADnimo
[12] http://agilemanifesto.org

6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Si bien no se siguió ninguna de las metodologías ágiles más conocidas de manera formal y estricta, el tipo de desarrollo utilizó parte de las características de *Extreme Programming*[13] (XP) y *SCRUM*[14] como base. Se debe tener en cuenta que ambas metodologías plantean equipos de desarrollo, los cuales en este proyecto se reducen a la figura del desarrollador y del cliente. Además, estas metodologías están pensadas especialmente para el desarrollo de software, y en este caso está orientado a un producto hardware-software. Es bastante habitual encontrar metodologías híbridas, aplicando unas u otras de manera adaptativa según los casos concretos y las personas involucradas en los proyectos. En una encuesta a gestores de proyectos, se reflejó que la metodología más seguida fue la SCRUM, seguida de una híbrida basada en la combinación de SCRUM y XP, como el caso de estudio de este proyecto.



Img.4: Encuesta de uso de metodologías ágiles a directores de proyectos

[13] https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema
[14] https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

Estas metodologías surgieron en contraposición al tradicional método en cascada, el cual es calificado como "burocrático, lento, degradante e inconsistente con las formas de desarrollo de software que realmente realizaban un trabajo eficiente".

Las llamadas "ágiles" hacen hincapié especialmente en la comunicación constante cara a cara y el lanzamiento rápido de las diversas versiones, involucrando al cliente final en el proceso al ser el tester y aportar *feedback* en cada iteración.

Img.5: Comparación de procesos tradicional y Ágiles

5.2. ¿Qué es Scrum?

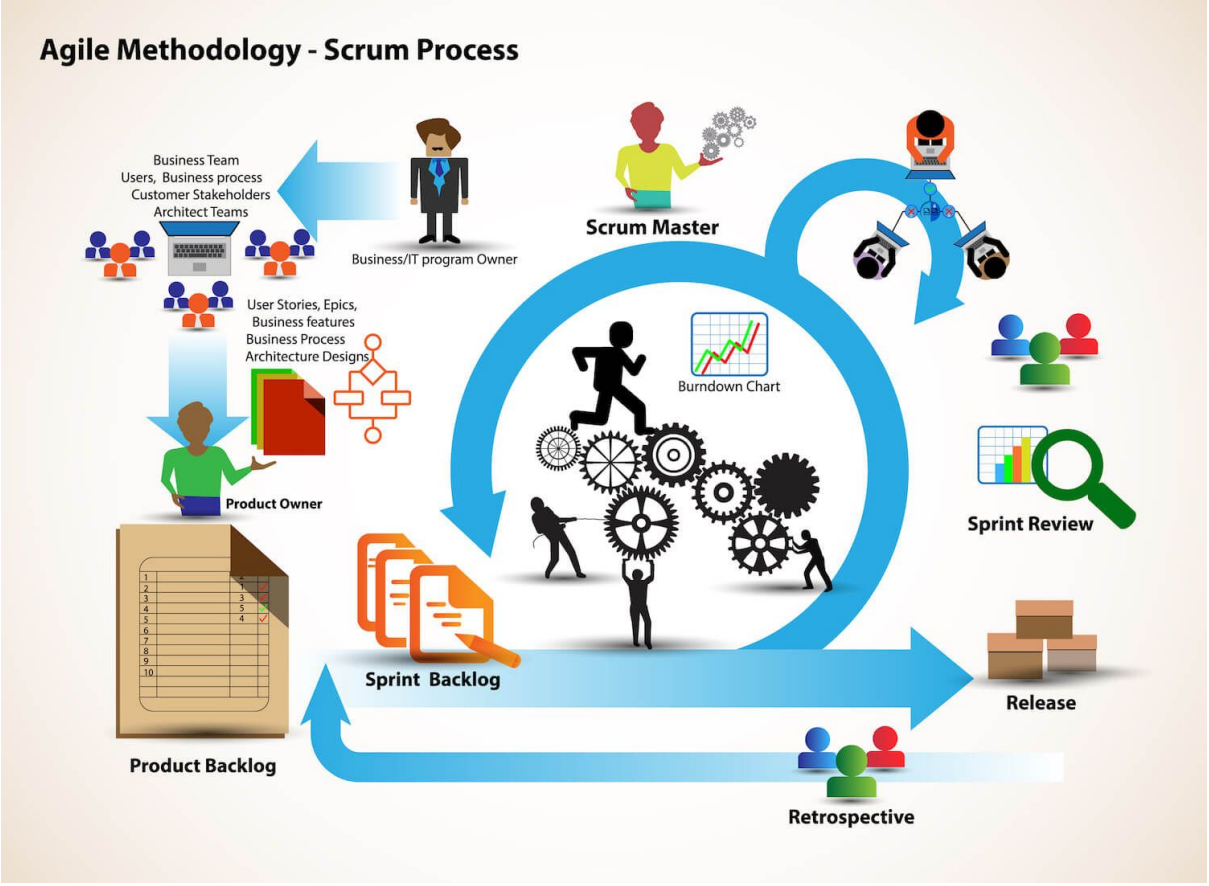
Scrum es una metodología ágil utilizada para el desarrollo de nuevos productos, popular entre los desarrolladores de software, y que ha sido adaptada para otros tipos de procesos.

Esta metodología sigue una serie de rituales y tiene roles rígidamente definidos. Su objetivo es desarrollar productos y servicios a través de pequeños avances incrementales en sus características y atributos, siempre a través de la validación del valor alcanzado con los cambios.

Entre sus principales rituales y agentes, podemos destacar:

- **Backlog**: Es la lista de tareas que deben realizar los equipos para desarrollar el producto o servicio deseado.
- **Sprint**: Es un período de tiempo para completar algunos conjuntos de tareas del backlog con el fin de consolidar un avance gradual en el producto o servicio.
- **Propietario del producto**: Es la persona responsable de la defensa de los intereses de los clientes o usuarios finales, y de definir las tareas que se asignarán en el Backlog.
- **Scrum Master**: Es el encargado de ser un guardián de la metodología Scrum y asegurarse de que se está siguiendo correctamente.

- **Daily Scrum**: Son reuniones diarias, por lo general en la mañana, donde los miembros del equipo hablan sobre su progreso del día anterior y lo que quieren lograr ese día.
- **Retrospectiva**: Al final de cada Sprint el equipo se reúne para evaluar sus resultados, las dificultades que fueron superadas y planificar el siguiente Sprint, siempre basado en las enseñanzas del Sprint anterior.



Img. 6: Concepto del ciclo de vida de la metodología ágil SCRUM.

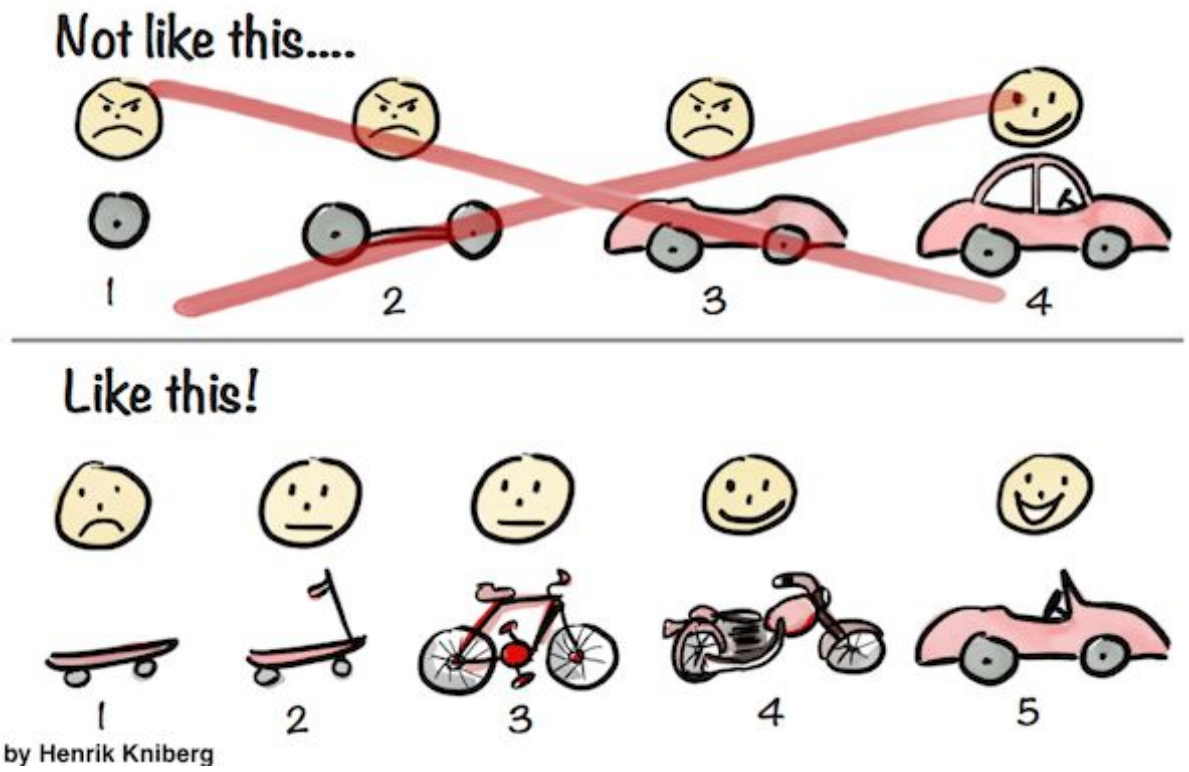
5.3. ¿Qué es MVP? *Minimum Viable Product*

Contrariamente a lo que muchos piensan, el MVP no es necesariamente un producto listo para ser lanzado en el mercado, con unos requisitos mínimos para que los consumidores puedan probarlo en situaciones reales de consumo. En el concepto original, la selección de conceptos y pruebas de prototipos con la ayuda de los posibles usuarios finales, ya era una forma de probar el producto mínimo viable.

El MVP es la forma más fácil de probar un producto con el menor uso posible de los recursos, antes de ponerlo en el mercado. Es una metodología de mejora continua y gradual, con el fin de atender mejor las necesidades y deseos del usuario final.

Una gran ventaja de este enfoque es que si hay un error, o algún atributo o funcionalidad desarrollada, que verdaderamente no cumple con las demandas del mercado, esto se puede corregir incluso en las primeras etapas del proyecto. Y de vuelta a la mesa de dibujo para crear un nuevo MVP – en base a lo que se ha aprendido de este error o desvío

de rumbo – para seguir probando sus mejoras en nuevos ciclos de desarrollo y pruebas incrementales.



Img. 7: Ejemplo la relación de lanzamientos de MVP y la satisfacción del cliente
Source: Making sense of MVP

Un ejemplo clásico de prueba de MVP sin la existencia de un producto real en el mercado, es el uso de una plataforma de crowdfunding para validar la idea de un producto. En la plataforma se describe lo que hace el producto, cuánto va a costar, cómo va a ser comercializado, sus atributos y beneficios e incluso se puede utilizar una animación en 3D de gran realismo simulando su funcionalidad, o incluso la filmación de un prototipo muy bien acabado, utilizado por los usuarios finales.

Dependiendo de la respuesta de los inversores, se puede saber, de manera ágil y menos costosa, si la idea es realmente viable y puede funcionar, o si se necesita un ajuste.

5.4.. Uniendo a Scrum y MVP

Es evidente que la utilización de ambos conceptos es viable y conveniente, uniendo estas dos formas de desarrollo de un nuevo producto.

Básicamente, los dos enfoques se pueden resumir en cinco frases:

1. Analice la situación actual
2. Avance hacia su objetivo
3. Analice y valide los resultados
4. Haga ajustes y mejoras basadas en lo que ha aprendido
5. Vuelva al principio e inicie un nuevo ciclo de mejoras

5.5. Extreme Programming (XP)

eXtreme Programming (o también llamado XP) es una metodología de desarrollo de la ingeniería de software formulada por Kent Beck[15], autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999). Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de la XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

Tal y como lo define Kent Beck, eXtreme Programming (XP) es un método ágil para el desarrollo de software muy útil a la hora de abordar proyectos con requisitos vagos o cambiantes. Es especialmente útil si se aplica a equipos de desarrollo pequeños o medianos. Se ajusta muy bien a los cambios pues es un método adaptativo.

Propone desarrollar el código de forma que su diseño, arquitectura y codificación permitan incorporar modificaciones y añadir una funcionalidad nueva sin demasiado impacto en la calidad del mismo.

Por otro lado está muy orientado hacia las personas, tanto a las que están creando el proyecto como a los usuarios finales. Desarrollando de la manera que propone XP se obtienen resultados rápidamente. Al trabajar con pequeñas iteraciones, se pueden conseguir con frecuencia comentarios del cliente, que tiene como resultado que el producto final cubra ampliamente sus expectativas y necesidades. Muchas de las prácticas que propone este método de programación no son en absoluto nuevas; lo novedoso de XP reside en su propuesta de aplicar las prácticas de forma simultánea y que se realimenten entre ellas. Algunas son técnicas que se han aplicado con éxito anteriormente y han resultado valiosas. El catálogo de prácticas de XP sigue evolucionando a lo largo de los años, poco a poco, se han ido incorporando más prácticas.

5.5.1. Ciclo de vida

En numerosas ocasiones se atribuye el fracaso de los proyectos a la poca definición de requisitos, o a un mal diseño de la arquitectura de un sistema al inicio del proyecto. El método de XP propone potenciar estas actividades, la forma de trabajar es definiendo los requisitos, arquitectura y diseño cada día, y no en un periodo de tiempo determinado y acotado. XP apuesta por simultanear las fases y llevarlas a cabo en paralelo para que, de este modo se vaya adaptando el producto y el sistema a las necesidades según van surgiendo. Cada semana se realiza un ciclo completo o iteración en la que se aborda algo de cada una de las fases tradicionales de un desarrollo software, trabajando con las

[15] https://es.wikipedia.org/wiki/Kent_Beck

historias de usuario, que son requisitos que dan valor al cliente. Cada semana se planifica en qué historias trabajar y se llevan a cabo de forma completa con su análisis, diseño, pruebas y desarrollo. Al final de la semana se podrá mostrar al cliente el resultado de la funcionalidad obtenida para que pueda formarse una opinión sobre ella.

5.5.2. Planificación

Las historias más prioritarias se implementarán en primer lugar. Por otra parte, el equipo planificará al inicio de cada semana la manera de abordar la siguiente iteración de forma detallada. De igual manera, el equipo organizará cada día el trabajo para esa jornada. En resumen, se planifica al inicio del proyecto, de cada iteración y aún más detalladamente, todos los días.

5.5.3. Análisis

Para que el análisis se mantenga actualizado durante todo el proyecto, los clientes y las personas que están construyendo el proyecto debe estar en comunicación constante; pues en el momento en que un desarrollador tenga dudas sobre cómo implementar un requisito, debe poder preguntar al cliente. A su vez, se debe trabajar en estrecha colaboración con los responsables de pruebas y de diseño gráfico para que no quepa ambigüedad en los requisitos.

5.5.2. Diseño y codificación

XP propone trabajar de manera que, tanto el diseño como la arquitectura se creen de forma incremental. De esta manera, se mejora el diseño y la arquitectura poco a poco y de forma constante.

5.5.4. Los Valores

Estos son los 5 valores que guían el desarrollo en XP:

- **Comunicación.** La comunicación entre los desarrolladores y a su vez con el cliente, es la mejor forma de evitar la mayoría de problemas y errores. Hay muchas formas de comunicarse, como un código simple y bien escrito que permita que cualquier desarrollador pueda entender el código desarrollado por otro miembro del equipo. Por otra parte a la hora de resolver un problema, siempre es mejor pensar la solución entre muchos que individualmente.
- **Simplicidad.** No es sencillo construir un sistema sencillo y que cumpla exactamente con los requisitos establecidos. Muchas veces la forma de simplificar un sistema es mediante ensayo y error. Construir de forma sencilla disminuye la cantidad de código y aumenta su calidad.
- ***Feedback.*** Se entiende como una combinación de comentarios, reacciones, sugerencias, opiniones, etc. Es necesario poder conocer siempre cómo de cerca está el producto que se está construyendo de lo que realmente se necesita. Además de la imprescindible comunicación con el cliente, otra herramienta para obtener información sobre el sistema, son los test automáticos que ayudan a revisar el estado del código y facilitan la detección de posibles fallos producidos por cambios.

- **Coraje.** Se necesita valentía para comunicarse clara y eficazmente con el cliente y con los compañeros y afrontar sin miedo los cambios en el proyecto. Y por supuesto es necesario ser valiente para aplicar algunas de las prácticas que XP propone ya que, en ocasiones, puede haber quien dude de su utilidad.
- **Respeto.** Los cuatro valores anteriores llevan implícito el respeto. Es indispensable el respeto mutuo, valorando el trabajo de los demás y sus aportaciones.

5.6. Uniendo todas las metodologías:

Para el presente proyecto, de manera resumida se extrajeron los siguientes puntos claves de la metodología SCRUM y MVP:

- Definición del MVP
- Trabajo por *sprints* y lanzamientos
- Flujo de tareas desde *sprint backlog* hasta el entregable
- Definición de *Backlog Freeze* para el *sprint planning*

De la metodología XP se utilizó:

- La preeminencia del diseño previo a la implementación
- Definición a base de tareas/funcionalidades sencillas
- Corrección de errores completa antes de la iteración de la siguiente funcionalidad
- Integración del cliente durante el proceso como tester y *feedback* continuo
- Desarrollos frecuentes con pequeñas mejoras de manera iterativa e incremental.

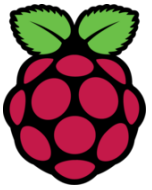
5.7. Recursos

A continuación, se hará un breve resumen de cada uno de los recursos hardware y software destacables utilizados durante el desarrollo del proyecto. La lista completa es muy larga ya que para un desarrollo tan ambicioso se tocan muchas librerías y aplicaciones para las diferentes facetas.

5.7.1. Recursos Hardware

Se hará una breve mención a los diferentes dispositivos y tecnologías hardwares utilizadas.

5.7.1.1. Raspberry PI



Raspberry Pi (también abreviado como RPI) es un ordenador de placa reducida o simple (SBC) de bajo coste desarrollado en el Reino Unido por la Fundación Raspberry Pi[16], con el objetivo de estimular la enseñanza de informática en las escuelas.

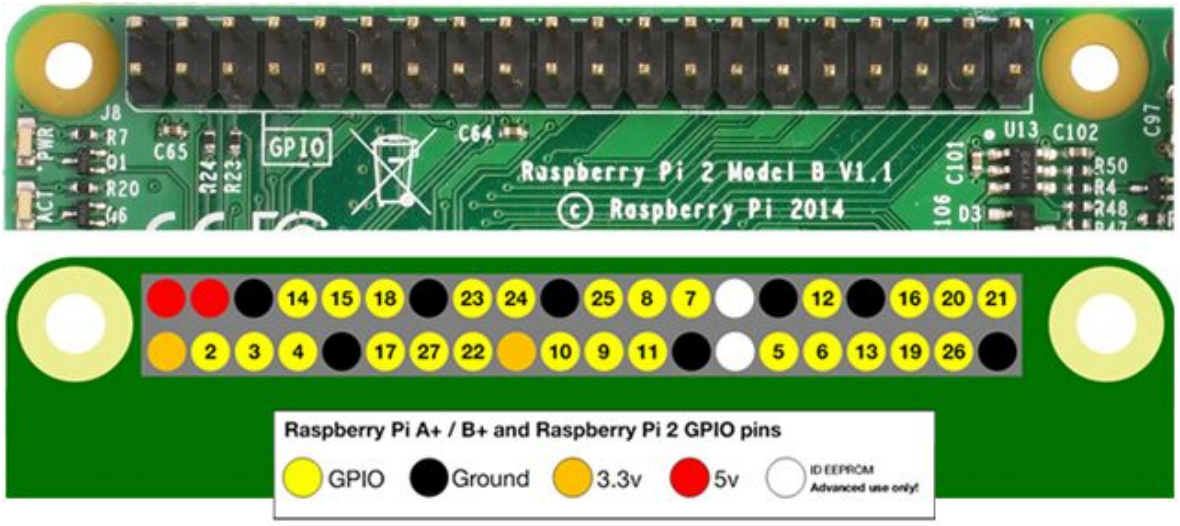
[16] https://www.raspberrypi.org/

Aunque no se indica expresamente si es hardware libre o con derechos de marca, en su web oficial explican que disponen de contratos de distribución y venta con dos empresas, pero al mismo tiempo cualquiera puede convertirse en revendedor o redistribuidor de las tarjetas RaspBerry Pi, por lo que da a entender que es un producto con propiedad registrada, manteniendo el control de la plataforma, pero permitiendo su uso libre tanto a nivel educativo como particular.

En cambio, el software sí es de código abierto, siendo su sistema operativo oficial una versión adaptada de Debian, denominada Raspbian, aunque permite usar otros sistemas operativos, incluido una versión de Windows 10.

En todas sus versiones incluye un procesador Broadcom, una memoria RAM, una GPU, puertos USB, HDMI, Ethernet (El primer modelo no lo tenía), 40 pines GPIO y un conector para cámara. Ninguna de sus ediciones incluye memoria, siendo esta en su primera versión una tarjeta SD y en ediciones posteriores una tarjeta MicroSD.

Estas placas de electrónica embebida son ideales para procesos de prototipado, ya que permiten conectar dispositivos digitales, y ampliar su potencialidad, usando cualquiera de las diferentes interfaces que tiene (DSI o puerto display para pantallas, CSI o puerto de la cámara, GPIO o puerto de entrada-salida de propósito general (digital).



Img. 8: Detalle de la organización de los pines GPIO de una Raspberry Pi

Las versiones que se utilizan en esta memoria son dos:

- Raspberry Pi v1 model B
  - 100 Base Ethernet
  - 4 USB ports
  - 40 GPIO pins
  - Full HDMI port
  - Combined 3.5mm audio jack and composite video
  - Camera interface (CSI)
  - Display interface (DSI)
  - SD card slot

- VideoCore IV 3D graphics core

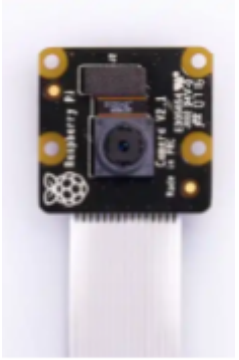
- Raspberry Pi v2 model B:
  - Lo mismo que la versión 1 model B, cambiando el procesador y la memoria:
    - A 900MHz quad-core ARM Cortex-A7 CPU
    - 1GB RAM
    - MicroSD card Slot

Img. 9: Módulo Raspberry Pi v2

Website: https://raspberrypi.org

La compañía además aporta una variedad de accesorios, algunos de los cuales usamos en este proyecto y trataremos a continuación.

5.7.1.2. Cámara PiCam NOIR



Img. 10: Módulo PiCAM NOIR

En noviembre del 2012, se presentó el prototipo final del módulo de la cámara en la feria Electrónica 2012 en Munich, y se dio a conocer que el sensor sería de 5 megapíxeles y que podría grabar vídeo a 1080p H.264 a 30 fotogramas por segundo. Se conecta a la RPI Finalmente el módulo se puso a la venta el 14 de mayo de 2013 en los principales proveedores. Las dimensiones del módulo son 25 x 20 x 9 mm. Para poder hacer uso de él, se tiene que activar en el menú raspi-config de Raspbian Más tarde, a finales de octubre de 2013 se puso a la venta un módulo de cámara de infrarrojos (sin filtro infrarrojos, o NOIR).

5.7.1.3. Tecnologías RFID

RFID o identificación por radiofrecuencia (del inglés Radio Frequency Identification) es un sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados etiquetas, tarjetas o transpondedores RFID. El propósito fundamental de la

tecnología RFID es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio. Las tecnologías RFID se agrupan dentro de las denominadas Auto ID (automatic identification, o identificación automática).

Img. 11: Tarjetas RFID 125Khz

Las etiquetas RFID (RFID tag en inglés) son unos dispositivos pequeños, similares a una pegatina, que pueden ser adheridas o incorporadas a un producto, un animal o una persona. Contienen antenas para permitirles recibir y responder a peticiones por radiofrecuencia desde un emisor-receptor RFID. Las etiquetas pasivas no necesitan alimentación eléctrica interna, mientras que las activas sí lo requieren. Una de las ventajas del uso de radiofrecuencia (en lugar, por ejemplo, de infrarrojos) es que no se requiere visión directa entre emisor y receptor.

Existen varios estándares de protocolos de transmisión RFID, pero en particular haremos uso de las llamada de “baja frecuencia”, a 125Khz en concreto. Esta modalidad permite una distancia de lectura de unos 10-12 cm de distancia, solo una etiqueta. Si detecta colisión no lee.

Se utilizarán tarjetas del tamaño de tarjetas de crédito blancas que poseen una antena pasiva interna con un número identificativo de 12 caracteres.

Además, se utilizarán dos tipos de lectores:

- PhidgetRFID[17], via USB, con Led de lectura (requiere driver, pero lo hay para usar con una gran variedad de lenguajes).
- UART EM4100, de tipo serial, estándar. Muy simple, pero puede conectarse a los conectores de la GPIO.

Img. 12: Módulo PhidgetRFID

5.7.2. Recursos Software

A continuación se listan los recursos software más destacados.

5.7.2.1. Linux Raspbian Lite (Debian Stretch)

Raspbian es una distribución del sistema operativo GNU/Linux[18] y por lo tanto libre basado en Debian[19] para la placa computadora (SBC) Raspberry Pi, orientado a la enseñanza de informática. El lanzamiento inicial fue en junio de 2012.

[17] https://www.phidgets.com/?tier=3&prodid=22
[18] https://es.wikipedia.org/wiki/GNU/Linux
[19] https://es.wikipedia.org/wiki/Debian

Técnicamente el sistema operativo es un port no oficial de Debian armhf para el procesador (CPU) de Raspberry Pi, con soporte optimizado para cálculos en coma flotante por hardware, lo que permite dar más rendimiento en según qué casos. El port fue necesario al no haber versión Debian armhf para la CPU ARMv6 que contiene el Raspberry PI.

Aunque su orientación inicial era hacia la enseñanza, el sistema Debian en el que está basado es ideal para desarrollo de aplicaciones servidores.

5.7.2.2. IDE elegido: Visual Studio Code

Visual Studio Code es un editor de código fuente desarrollado por Microsoft[20] para Windows , Linux y macOS. Incluye soporte para la depuración, control integrado de Git[21], resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto[22], aunque la descarga oficial está bajo software propietario[23].

Visual Studio Code se basa en Electron[24], un framework que se utiliza para implementar aplicaciones NodeJS[25] para el escritorio, que se ejecuta en el motor de diseño Blink.

Es el entorno de desarrollo, o IDE, más utilizado durante el proceso de desarrollo de este proyecto junto con Sublime Text[26], que es más ligero.

Website: https://code.visualstudio.com/

Website Sublime Text: https://www.sublimetext.com/

5.7.2.3. Cacoo diagrams

Aplicación en la nube orientada a la creación de diagramas colaborativos. Posee plantillas para muchos escenarios, desde diagramas de redes de comunicaciones, a brainstormings y mindmaps, así como mockups o diagramas UML. Tiene una versión gratuita que te permite tener hasta 6 diagramas y un espacio limitado. La versión premium pasa a costar unos $5 al mes.

Website: https://cacoo.com

5.7.2.4. Google Docs y Google Drive

Google proporciona una suite de ofimática, con editor de textos (Google Docs), hojas de cálculo (Google Sheets), de presentaciones (Google Slides), etc., así como un entorno de gestión documental en la nube (Google Drive) bastante decente. Dada la naturaleza del proyecto,

[20] https://es.wikipedia.org/wiki/Microsoft
[21] https://es.wikipedia.org/wiki/Git
[22] https://es.wikipedia.org/wiki/C%C3%B3digo_abierto
[23] https://es.wikipedia.org/wiki/Software_propietario
[24] https://es.wikipedia.org/wiki/Electron_(software)
[25] https://es.wikipedia.org/wiki/Node.js
[26] https://es.wikipedia.org/wiki/Sublime_Text

que implica utilizar varios entornos operativos diferentes se eligió probar a desarrollar esta memoria utilizando el editor de textos de Google Drive (Google Docs).

El funcionamiento es bastante parecido a los editores tradicionales, pero con la salvedad de que no se necesita estar en un ordenador con un determinado sofware instalado, sino tan solo abrir un navegador, y abrir el documento en Drive usando Docs, listo, ya puedes continuar trabajando. En comparación también es bastante limitado en opciones, pero la sencillez también fomenta un entorno más limpio.

Website: https://drive.google.com

Tecnologías utilizadas

A continuación, se tratarán las tecnologías software utilizadas durante el proceso de implementación.

5.7.3.1. Tecnologías web: HTML / CSS / JS

En el desarrollo web, existen tres lenguajes de programación orientados hacia el renderizado de páginas web en el lado del cliente, que determinan el aspecto, organización y comportamiento de las interfaces web, comúnmente llamado FrontEnd:

- **HTML** o HyperText Markup Language:

  Lenguaje de etiquetas, es el encargado de estructurar una página web como si se tratara de un documento, especificando los entornos, las secciones y las jerarquías.

- **CSS** o Cascading Style Sheets:

  Lenguaje de reglas, que especifican de qué manera se muestra la información, aportando las características de diseño a la estructura del documento (colores, posiciones, tamaños, grosores, fondos, etc.). Una característica a destacar es que hereda las reglas en cascada, es decir, varios ficheros pueden reescribir una misma regla, dando prioridad al último de la lista.

- **Javascript**:

  Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript[27]. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

  Se usa especialmente para determinar el comportamiento de los diferentes elementos del DOM[28], o capa de objetos.

[27] https://es.wikipedia.org/wiki/ECMAScript
[28] https://es.wikipedia.org/wiki/Document_Object_Model

Estas tecnologías definen pues la estructura, la apariencia y el comportamiento de una interfaz web. Las dos primeras son dependientes del estándar de la World Wide Web consortium (https://www.w3.org/), cuya versión más reciente es la HTML5 y la CSS3.

Por otro lado, según la ECMAScript, la versión de JS más actual es la 2018, aunque la que más ampliamente se utiliza en todo el mundo es la ES5 y la ES6.

5.7.3.2. Bootstrap de Twitter

Bootstrap es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales. A diferencia de muchos frameworks web, solo se ocupa del desarrollo front-end.

5.7.3.3. Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation[29]. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Python es el lenguaje base en que se desarrollará el software principal del proyecto, tanto a nivel de la app web, como de la interfaz gráfica del sensor. Se utilizará principalmente la versión 2.7, ya en proceso de extinción en favor de la versión 3, las cuales no son compatibles entre sí.

Website: https://www.python.org/

5.7.3.4. Django

Django es un framework de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como Modelo–vista–template, similar en esencia con el conocido Modelo-Vista-Controlador. Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005; el framework fue nombrado en alusión al guitarrista de jazz gitano Django Reinhardt. En junio de 2008 fue anunciado que la recién formada Django Software Foundation se haría cargo de Django en el futuro.

La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio No te repitas (DRY, del inglés Don't Repeat Yourself). Python

[29] https://www.python.org/psf/

es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos.

Website: http://djangoproject.com/

5.7.3.5. PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto, publicado bajo la licencia PostgreSQL, similar a la BSD[30] o la MIT[31].

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre o apoyados por organizaciones comerciales. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

PostgreSQL no tiene un gestor de defectos, haciendo muy difícil conocer el estado de sus defectos, lo que le ha valido tanto detractores como defensores.

Según algunos estudios, se diferencian especialmente en el rendimiento cuando se trabaja de manera intensiva y en aspectos relacionados con la seguridad. Por regla general se recomienda que para proyectos pequeños se utilice MySQL y en proyecto medianos o grandes, o que prevean escalar, PostgreSQL. Aunque también hay una facción que lo recomiendan para cualquier proyecto serio, sin importar su tamaño.

Website: https://www.postgresql.org/
Website comparativa de PostgreSQL vs MySQL:
https://www.2ndquadrant.com/en/postgresql/postgresql-vs-mysql/

5.7.3.6. Virtualizaciones: Docker y Vagrant

Aunque en el presente proyecto no tienen una importancia capital, considero interesante nombrar ambas tecnologías de virtualización y su importancia de cara a un desarrollo y a una coordinación de equipo de desarrollo. Estas tecnologías nos permitirán emular entornos de hardware y software reales, haciendo más fácil la medición y el control de errores típicos al desplegar un desarrollo en una máquina diferente (el típico problema de “en mi casa funcionaba”). Si bien ambas lo hacen de manera diferente.

- **Docker**

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Docker utiliza características de aislamiento de recursos del kernel

[30] https://es.wikipedia.org/wiki/Licencia_BSD
[31] https://es.wikipedia.org/wiki/Licencia_MIT

Linux, tales como cgroups y espacios de nombres (namespaces) para permitir que "contenedores" independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales[32].

El soporte del kernel Linux para los espacios de nombres aísla la vista que tiene una aplicación de su entorno operativo, incluyendo árboles de proceso, red, ID de usuario y sistemas de archivos montados, mientras que los cgroups del kernel proporcionan aislamiento de recursos, incluyendo la CPU, la memoria, el bloque de E/S y de la red.

Website: https://www.docker.com/

- **Vagrant**

Vagrant es una herramienta para la creación y configuración de entornos de desarrollo virtualizados. Originalmente se desarrolló para VirtualBox[33] y sistemas de configuración tales como Chef, Salt y Puppet.

Website: https://www.vagrantup.com/

La diferencia principal es que Vagrant levanta una máquina virtual completa con todos sus recursos y su sistema operativo, mientras que docker tan solo las capas específicas de aplicación y librerías necesarias, ahorrando mucho en recursos. Mientras que Vagrant nos permite imitar el entorno real de la máquina al completo, permitiendo modificar el sistema operativo como si fuera un entorno real, Docker hace hincapié en una tecnología concreta, limitando el acceso al backend donde se sustenta esas tecnologías.

Se recomienda SIEMPRE generar una máquina Vagrant o un sistema Docker que emule en la medida de lo posible el entorno real de funcionamiento, antes de comenzar con el desarrollo y que todos los desarrolladores utilicen las mismas máquinas. Aquí se hace una recopilación de las principales ventajas de hacer esto:

- Separación del entorno de desarrollo de la configuración de la máquina anfitrión.
- Rápida puesta en marcha de entornos con diferentes configuraciones y aplicaciones de manera rápida y segura.
- La aplicación en desarrollo incorpora información sobre el “ecosistema” en que se ejecuta.
- Sería posible desarrollar con garantías en un escenario con diferentes desarrolladores con diferentes sistemas operativos.
- Facilita testear el test de nuevas versiones de librerías o tecnologías nuevas sin comprometer la máquina anfitrión.
- Facilita trabajar con proyectos con distintas dependencias.
- Reduce o puede eliminar las inconsistencias entre máquinas de desarrollo y producción.

[32] https://es.wikipedia.org/wiki/M%C3%A1quina_virtual
[33] https://es.wikipedia.org/wiki/VirtualBox

5.7.3.7. API REST y JSON

La interfaz de programación de aplicaciones, conocida también por la sigla **API**, en inglés, *application programming interface*, o interfaz de programación de aplicaciones, es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. A efectos prácticos, es la capa de comunicación entre aplicaciones diferentes.

En nuestro proyecto, el sensor se comunica con el servidor mediante esta capa API.

La transferencia de estado representacional (en inglés *representational state transfer*) o **REST** es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. Describen cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML[34], JSON[35], etc) sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes.

Como características clave se nombran las siguientes:

- Un **protocolo cliente/servidor sin estado**: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST)
- Un conjunto de **operaciones bien definidas** que se aplican a todos los *recursos* de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son **POST**, **GET**, **PUT** y **DELETE**. Con frecuencia estas operaciones se equiparan a las operaciones CRUD[36] en bases de datos (CLAB en castellano: crear, leer, actualizar, borrar) que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.
- Una **sintaxis universal** para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI[37].
- El **uso de hipermedios**, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son **típicamente HTML o XML**. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

JSON (acrónimo de **JavaScript Object Notation**, «notación de objeto de JavaScript») es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.

[34] https://es.wikipedia.org/wiki/XML
[35] https://es.wikipedia.org/wiki/JSON
[36] https://es.wikipedia.org/wiki/CRUD
[37] https://es.wikipedia.org/wiki/Uniform_Resource_Identifier

Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos es que resulta mucho más sencillo escribir un analizador sintáctico (parser) para él. En JavaScript, un texto JSON se puede analizar fácilmente usando la función `eval()`, algo que (debido a la ubicuidad de JavaScript en casi cualquier navegador web) ha sido fundamental para que haya sido aceptado por parte de la comunidad de desarrolladores AJAX[38].

De manera resumida, nuestra API será REST y usará como formato de intercambio de mensajes JSON.

[38] https://es.wikipedia.org/wiki/AJAX

6. Análisis

6.1. El centro objetivo

En el caso concreto de estudio de esta memoria, el centro consta de 37 trabajadores:

- 4 técnicos (fisioterapeuta, trabajador social, psicólogo y de recursos humanos)
- 30 auxiliares y personal de limpieza
- 3 eclesiásticos (no sujetos al convenio colectivo ya que no poseen contrato ni perciben salario, pero poseen acceso administrador al sistema)

Img. 13: Imagen del centro, objetivo del proyecto

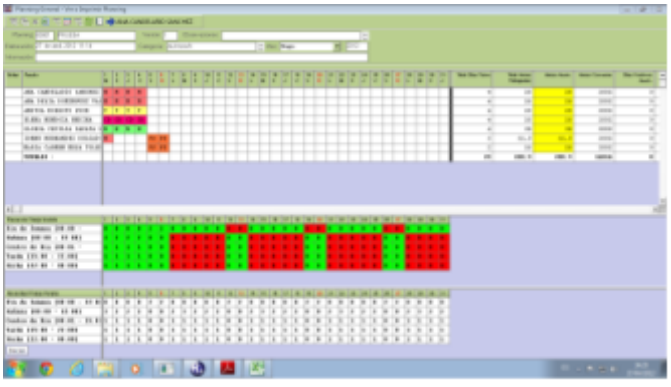
El centro ofrece los siguientes servicios orientados a personas con una alta dependencia por discapacidad psíquica severa:

- Centro de día (equivalente a un colegio, con un horario similar hasta las 18.00 habitualmente).
- Centro de atención residencial (servicio de residencia y cuidados durante periodos que varían entre unos pocos días e indefinidos).

Los turnos en los que se divide el trabajo del personal son los siguientes:

- Turno de mañana (8.30 a 16.00)
- Turno de tarde (15.00 a 22.30)
- Turno de noche o guardia (22.00 - 9.00)

Según la normativa vigente, los trabajadores deben fichar al principio de sus turnos y al acabar, y actualmente este proceso se lleva a cabo a través de hojas de firmas. Esto implica:



Img. 14: Detalle del planificador de turnos de Resiplus

- Generación e impresión de la hoja semanal de firmas
- Habilitación de la hoja en la zona común de los trabajadores (sala al lado del despacho del técnico de Recursos Humanos)
- El técnico de recursos humanos o alguno de los eclesiásticos debe estar presente en el momento de la firma.
- Cada semana se recopilan los registros de la anterior y se introducen a mano en una hoja de excel por el técnico de Recursos Humanos.

- Se validan y se archivan las hojas de firmas en sus carpetas correspondientes.
- Se resuelven las incidencias.
- Al final del mes se emite un informe por persona con las horas trabajadas.

Este proceso da lugar a las siguientes problemáticas según conversaciones con los empleados y responsables del centro:

- Sólo el turno de mañana y el inicio del de tarde pueden cubrirse con alguien revisando el correcto relleno de las hojas de firmas, pero la realidad es que no se hace de manera efectiva.
- Muchas veces, los procesos de fichaje se quedan sin supervisión, creándose incoherencias entre las horas de entradas y salidas reales y las que aparecen en los registros
- Los empleados intercambian favores de fichaje para encubrir ausencias.
- No se controlan ausencias reducidas, especialmente durante los turnos de noche, tales como conversaciones privadas, comidas, recados, descansos, etc. Esto ha provocado en alguna ocasión incidentes al no haber un empleado cerca cuando alguno de los internos ha sufrido una crisis.
- No existe una mecánica fiable de control de horas trabajadas.

6.2. La necesidad a cubrir

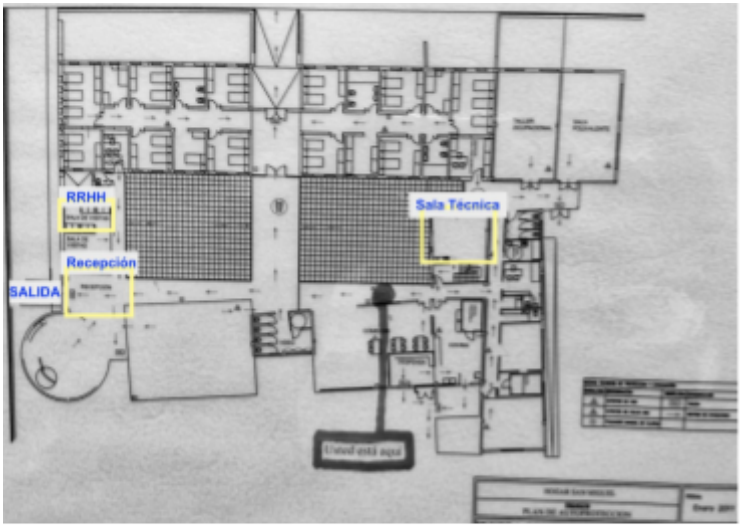
El proyecto se basa en desarrollar un sistema barato, eficiente, robusto, de bajo impacto y no biométrico para controlar el cumplimiento de asistencia y control horario del personal del centro.

6.3. Circunstancias y variables de entorno

En la instalación local, se ha determinado el mejor espacio para el sensor, cerca de la puerta de salida del centro, en la zona de “Recepción”.

La legislación vigente, el convenio colectivo de empresa de Instituto de Atención Social y Sociosanitaria de 2001, artículo 42 (Uniformidad), obliga a llevar una tarjeta identificativa visible proporcionada por la empresa, la que en este caso constaría de:

- nombre y apellidos
- DNI
- puesto que desempeña
- logo de la asociación y del centro



Img. 15: Fotografía y diagrama de un plano del centro

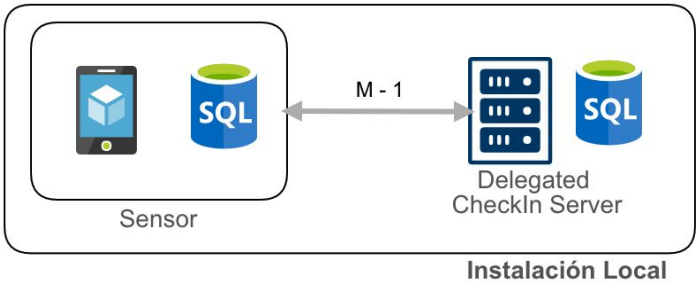
6.3.1. Parque tecnológico existente

Existen las siguientes instalaciones tecnológicas a disposición de los trabajadores y de este proyecto:

- Conectividad inalámbrica Wifi disponible.
- Área de "Oficina Técnica", con un armario rack de servidores, donde se encuentra centralizada la conectividad de todas las instalaciones y el servidor de gestión del centro denominado *Resiplus*[39]. Dicho servidor, funciona bajo *Windows 7 Pro*.
- Todos los miembros técnicos del centro y los administradores utilizan ordenadores personales con instalaciones de *Windows 7 Home* y *Windows Vista Home*. Hay un terminal también disponible para los auxiliares y profesionales de limpieza.

6.4. Requisitos técnicos

En base a las circunstancias del cliente de este sistema, y a sus requerimientos, así como aquellos autoimpuestos, que han sido expuestos en la sección de "Objetivos", definimos una solución con una arquitectura cliente-servidor según el siguiente diagrama:



Img. 16: Diagrama cliente-servidor del Check-In

En él se observan dos objetos diferenciados conectados:

- Sensor:
  - Objeto Hardware + Software
  - Captador de checks
  - Relación Muchos (puede haber varios sensores)

- Servidor:
  - Objeto Hardware + Software
  - Herramientas de gestión vía web server
  - Relación 1 (sólo un servidor)

[39] https://www.addinformatica.com/

Además se establecen las siguientes características en base a los requerimientos técnicos:

- **Solución basada en Web**

Se ha optado por una solución de tipo web para disminuir el gasto necesario en parque tecnológico ya que para un navegador no se necesitan requisitos concretos ni licencias de software. Por otro lado es altamente portable.

- **Diseño Web adaptable** (Responsive Web design)

Se opta por este paradigma de diseño para aumentar la compatibilidad de uso del interfaz de administración a cualquier tipo de pantalla, facilitando así que pueda ser consultado por un dispositivo móvil, por ejemplo. Esto permite ahorrar en un principio en desarrollos a medida para plataformas móviles.

- **Conectividad inalámbrica**

Dada la disponibilidad de esta vía de comunicación digital presente, se utilizará una solución que permita aprovecharla.

- **Metodología de identificación**

Aprovechando la circunstancia de que todos ya llevan una tarjeta identificativa, la cual es obligatoria, podemos utilizar dichas tarjetas como material identificativo para el proceso de fichaje. Para ello, debemos sustituir dichas tarjetas por tarjetas “inteligentes” que nos permitan identificar eficientemente a los trabajadores.

6.5. Requisitos funcionales

Según la ingeniería del software[40], se define como una función del sistema de software o sus componentes. Una función es descrita como un conjunto de entradas, comportamientos y salidas. Los requisitos funcionales pueden ser: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que se supone, un sistema debe cumplir. Los requisitos de comportamiento para cada requisito funcional se muestran en los casos de uso[41].

Son complementados por los requisitos no funcionales[42], que se enfocan en cambio en el diseño o la implementación. Como requerimientos funcionales se considerarán principalmente aquellas funciones que son esenciales para considerar un éxito el proyecto, y se basan principalmente en los que establece el cliente como mínimos.

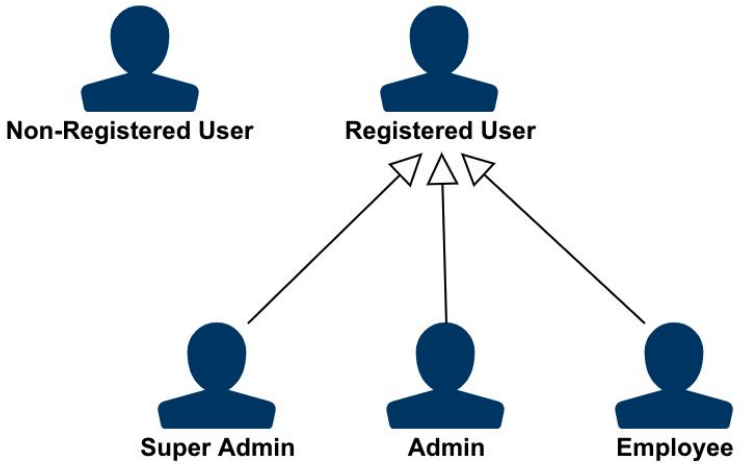
Para ello, lo primero que se identifican son los actores del sistema.

[40] https://es.wikipedia.org/wiki/Requisito_funcional
[41] https://es.wikipedia.org/wiki/Casos_de_uso
[42] https://es.wikipedia.org/wiki/Requisito_no_funcional

6.5.1. Actores

Un actor en nuestro sistema es una entidad con un rol determinado, y por tanto una serie de acciones que pueden llevar a cabo.

El siguiente diagrama refleja la jerarquía de actores.



Img. 17: Diagrama de actores del sistema

En la siguiente tabla se definen de manera formal:

Actor (rol)	Tipo	Definición
Usuario no registrado	Principal	Usuario que accede al sistema sin haberse autenticado. Se le solicita credenciales y se le da la posibilidad de recordar la contraseña.
Usuario registrado	Principal	Usuario autenticado en el sistema. Puede interactuar con las diferentes funcionalidades del sistema.
Super Admin	Principal	Usuario registrado con acceso total a todo el sistema. Está pensado como usuario de mantenimiento y soporte.
Admin	Principal	Usuario registrado con acceso restringido a las funciones de configuración del sistema (backoffice). Pensado para los responsables de Recursos Humanos del centro.
Employee	Principal	Usuario registrado sin acceso al backoffice. Pensado para que pueda modificar cierta información sobre sí mismo y consultar

		información de sus propios informes.

Tabla 5: Definición de los actores

6.5.2. Acciones

Aquí se refleja de manera reducida la tabla de acciones posibles por parte de los actores, que ayudará a tener una visión general de la potencialidad del PSI. Se reflejan de menor a mayor permisos, indicando que cada rol siguiente tiene las mismas funciones que el anterior más los reflejados adicionalmente.

Actor (rol)	Función	Resumen
Usuario no registrado	Iniciar sesión	Solo puede utilizar el formulario de autenticación si posee las credenciales.
	Recordar contraseña	Puede solicitar que le recuerden la contraseña en el caso de que la haya perdido.
Usuario registrado	Editar perfil	Al estar autenticado, puede hacer pequeños cambios de información personal en el sistema, como cambiar la contraseña, el correo, teléfono, etc.
	Cerrar sesión	Puede salir del sistema como usuario autenticado.
Employee	Visualizar informe personal	Como empleado puede visualizar sus checks y sus informes de cumplimiento horario
Admin	Acceder al backoffice	Puede acceder al entorno de administración
	Gestión del personal	Puede añadir, quitar y modificar fichas de personal (admin o employee)
	Gestión de tarjetas	Puede añadir, quitar y asignar tarjetas al personal
	Gestión de Checks	Puede añadir (manualmente), quitar o modificar checks del sistema.
	Gestión de turnos	Puede añadir, quitar y modificar turnos de trabajo
	Gestión de informes	Puede generar informes
Super Admin	Gestión de usuarios y permisos	Puede modificar los permisos de usuarios y crear otros Super Admin.

Tabla 6: Acciones de los actores

6.6. Requisitos no funcionales

Se refieren a todos los requisitos que no describen información a guardar, ni funciones a realizar, sino características de funcionamiento, por eso suelen denominarse Atributos de calidad de un sistema. Introducimos, por tanto, algunas decisiones que han quedado patentes como necesarias a través de las diferentes iteraciones del desarrollo del proyecto en sí.

6.6.1. Check. Definición y tipos

Un *check* es la medida mínima de fichaje, y representa una entrada en los registros del sistema. Contiene la siguiente información:

- ¿Quién?
- ¿Dónde?
- ¿Cuándo?
- ¿Por qué?

Definiendo cada campo y sus implicaciones:

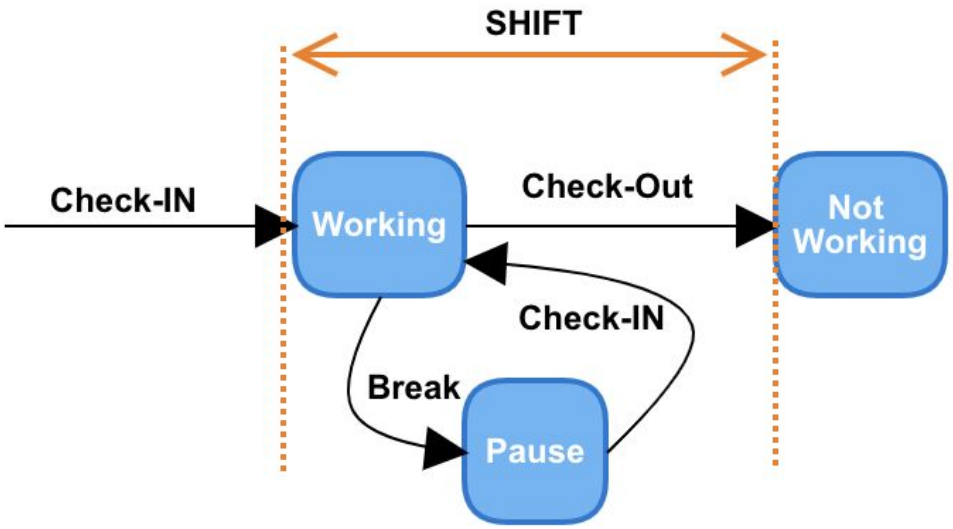
- **Quién**:
  Define la persona con la está relacionado el *check*. La forma que por ahora se utilizará para identificar a la persona es a través de la tarjeta identificativa obligatoria, modificándose para que sea una "tarjeta inteligente[43]". Esta modificación será transparente para la propia persona, es decir, recibirá una tarjeta exactamente igual a la que tenía, solo que incluyendo un elemento digital que no será visible.

- **Dónde**:
  Define desde dónde se generó el check. Normalmente será el sensor especialmente diseñado para ello, pero es posible que exista en el futuro varios sensores, además de la posibilidad de haber sido generado por un usuario humano, como por ejemplo, un check manual realizado por el técnico de Recursos Humanos.

- **Cuándo**:
  Esta información es quizás la que da más sentido a este proyecto, ya que se trata de fijar la fecha y la hora de cuando una persona entra o sale del centro. Atendiendo al estándar ISO 8601[44], que regula el formato de muestra de la fecha y hora, con su representación de zona horaria, se utilizará la metodología UNIX o POSIX[45] para almacenar la marca temporal. Esta consiste en un número entero que representa la cantidad de segundos transcurridos desde la medianoche UTC del 1 de Enero de 1970 si el valor es positivo, y los segundos restantes para esa fecha si la cantidad es negativa.

[43] https://es.wikipedia.org/wiki/Tarjeta_inteligente
[44] https://es.wikipedia.org/wiki/ISO_8601
[45] https://es.wikipedia.org/wiki/Tiempo_Unix

- **Por qué**:
  La idea es reflejar la razón del *check*. Se definen **tres tipos**:
  Check-in: Entrada en el centro, normalmente relacionada con el inicio del turno de trabajo, pero también con el regreso de un *break*.
  Check-out: Salida del centro, normalmente relacionada con el fin del turno de trabajo.
  Break: Salida del centro breve, relacionada con una parada corta en el turno de trabajo, como puede ser la hora de la comida o un descanso, por ejemplo.
  El siguiente diagrama de estados aclara el funcionamiento de estas señales:



Img. 18: Diagrama de estados de un trabajador y la relación las señales de *check*

Donde SHIFT refleja el tiempo del turno, y la contabilización de horas trabajadas ocurre **sólo** durante el tiempo que se permanezca en el estado de "Working" o "trabajando".

6.6.2. Tecnología hackeable y Open Source

Durante el proceso de análisis, se establecieron como requisitos las siguientes condiciones con respecto al desarrollo del software y el uso del hardware:

- El objetivo del proyecto es que sea de tipo Llave en Mano[46].
- El resultado no debe provocar ninguna situación de esclavitud u obligación de ser continuado por el desarrollador que lo comenzó.

[46] https://es.wikipedia.org/wiki/Llave_en_mano

- Se debe utilizar software libre[47] y código abierto[48] en la medida de lo posible, así como estándares y librerías de terceros mantenidos actualmente para asegurar disponibilidad en el tiempo. La licencia que se utilizará será la GPLv3[49].
- El centro recibirá la información y las claves de toda la infraestructura para que pueda libremente contratar a otra empresa para que continúe el desarrollo sin cortapisa ni licencias.
- El hardware generado deberá permitir ser hackeable, sin bloquear arranque y sin contraseñas ocultas, priorizando licencias de Open Hardware[50] y Creative Commons [51], permitiendo que pueda ampliar sus funciones si se deseara en la medida de lo posible, más allá del objeto inicial con el que se concibió en un inicio.

Es por ello, que hemos apostado por el hardware de Raspberry Pi, cuya versión 2 model B [52] es la última lanzada al mercado en el momento del inicio del desarrollo de esta memoria:

Img. 19: Detalle de la placa Raspberry Pi v2 model B

Lanzada en 2014 es el primer modelo que no incluye el mismo procesador usado en los tres anteriores: se sustituye por uno de la misma marca, pero de modelo BCM2836. Pasa de ser de un núcleo a cuatro, y de 700MHz a 900MHz. No obstante emplea la misma gráfica, la VideoCore IV. Dobla la cantidad de memoria RAM, pasando de 512MB a 1GB (Algo menos en realidad) esta memoria está compartida con la gráfica. También incluye 40 pines GPIO, y mantiene los cuatro puertos USB. Suprime la conexión RCA.

6.6.3. Sencillez = Velocidad

Uno de los preceptos de los diseños para este proyecto es intentar maximizar la velocidad de procesamiento de fichaje. Haciendo un cálculo rápido:

- Cada firma en una hoja lleva aproximadamente unos 15sg de media.
- El turno con mayor cantidad de entradas es el de la mañana con unas 15 entradas.
- Poniendo que el proceso es fluido, 15sg * 15 * 2(entrada y salida) = ~450 sg = ~8 minutos
- El turno de la tarde: 10 trabajadores = ~5 minutos
- El de noche: 5 trabajadores = ~3 minutos
- Total de tiempo invertido en proceso de firma diarios = de 15 a 20 minutos diarios.
- Los fines de semanas son turnos de 5 personas, por lo que se contabilizan 10 minutos al día.
- Por semana, 5 * 20min + 2 * 10min = ~ 120min/semana

[47] https://es.wikipedia.org/wiki/Software_libre
[48] https://es.wikipedia.org/wiki/C%C3%B3digo_abierto
[49] https://es.wikipedia.org/wiki/GNU_General_Public_License#Versi%C3%B3n_3
[50] https://es.wikipedia.org/wiki/Hardware_libre
[51] https://es.wikipedia.org/wiki/Creative_Commons
[52] https://es.wikipedia.org/wiki/Raspberry_Pi#Raspberry_Pi_2_modelo_B[52]%E2%80%8B

- Cada semana hay que recopilar e imputar las horas de los trabajadores en su hoja. Calculando solo el tiempo que se tarda una vez el técnico tiene las hojas delante y apunta a cada trabajador su valor es de 15sg por trabajador.
- Siendo, de media, unos 30 trabajadores por jornada, con una sola entrada y una sola salida cada uno.
- 60 registros diarios * 5 días laborables = 300 registros por semana laborable * 15sg por registro = ~1,25h.
- En los fines de semana son 15 trabajadores diarios: 30 registros * 2 días = 60 * 15sg = 0,25h
- Por tanto, el proceso de registro lleva más de 1 hora y media todas las semanas
- El tiempo total invertido en procesos de fichaje y registro a la semana supera las 3h y media (>210 min), teniendo en cuenta que no se aplican otros factores que también influyen en los tiempos más, no existen pausas y partiendo de escenarios ideales de fluidez en los procesos.

Nuestro sistema aplicará los siguientes objetivos:

- Reducir a la mitad el proceso de fichaje: ~8sg
- Reducir el proceso de registro de los horarios a 0 (en un escenario ideal).
- El cálculo por tanto sería: ~60min/semana
- El *speed up* = 210min/60min = 3,5

Para que el proceso sea tan rápido, se aplicarán las siguientes decisiones:

- No se establecerá un sistema auxiliar si no se posee la tarjeta (por ejemplo, un teclado numérico, o sistema auxiliar de usuario y contraseña).
- No existirá por tanto ningún elemento de interacción con el sensor diferente de la propia tarjeta.
- Si no se tiene la tarjeta encima, situación que no debe darse ya que es obligatoria, se solicitará mediante petición formal al técnico de Recursos Humanos que se introduzca el *check* manualmente, por lo que el proceso será supervisado y asíncrono.
- Se calcula entre 5-6sg de proceso del *check* y 2sg de colocación.

6.6.4. Aumentando la Robustez: Una fotografía

Un sistema basado en fichaje **solo con las tarjetas identificativas no es un sistema robusto**. En los espacios donde esto funciona así es bien conocida la costumbre de fichar por compañeros, prestado y clonado de tarjetas, etc. Para mejorar el nivel de robustez y fiabilidad de la información, hemos añadido como característica en el *Backlog Freeze* de la iteración actual la toma de una fotografía en el momento del *check*. Por tanto, adicionalmente al valor de la tarjeta en la sección "*Quién"* de un *check*, se proporciona una prueba gráfica.

Esto abre, a su vez, una serie de implicaciones y de decisiones adicionales tanto sobre el propio proyecto como sobre la metodología de funcionamiento de este tipo de sistemas.

6.6.4.1. ¿Es legal tomar una fotografía de un empleado?

Fuentes legales consultadas, que nos han pedido que no se refleje en el presente documento, nos arroja tres interesantes implicaciones:

1. La legalidad vigente no permite la toma de material gráfico del empleado sin su consentimiento expreso.
2. La legalidad vigente permite al empresario definir la forma y la metodología de control de absentismo laboral, pero debe estar reflejado en sus estatutos, en los convenios o en el contrato para que pueda ser utilizado a nivel penal y laboral. Así mismo, el sistema a utilizar, si este es digital, debe poseer un certificado de auditoría para apoyar que los datos sean válidos y coherentes en un posible juicio.
3. Se debe especificar la metodología para la "correcta toma de la fotografía" durante el *check* para poder ser "exigido" a los empleados y evitar que utilicen diferentes artimañas para que la imagen tomada no sea clara o concluyente.

Es de interés de la organización del centro poder utilizar el incumplimiento de los horarios y ausencias durante los turnos como argumento para despidos procedentes. Por tanto, se determinó que si bien el punto 1 y 3 pueden ser especificados en un adendo al contrato del trabajador, el punto 2 requería de una auditoría que en esta fase del proyecto no era viable. Este punto se ha añadido a líneas futuras al ser una mejora de los procesos a medio o largo plazo, pero no a corto como una característica de los próximos sprints.

Es importante reflejar además que una auditoría no se pasa de manera inmediata, ya que posiblemente implicaría modificaciones para adaptarse a normativas y a mecánicas formales definidas por la entidad certificadora, y que además implica una inversión económica adicional en el producto, que, debido a que no es una certificación sobre el producto adaptado a este centro, sino sobre el propio producto y su lógica de negocio, deberíamos ser los creadores los que corramos con los gastos, lo cual no es viable en este momento.

Por tanto, por parte de la organización, y como tarea del técnico de Recursos Humanos, se preparó un adendo al contrato donde:

- se informaba al empleado del nuevo método de control de presencia y absentismo, así como del funcionamiento, los actores y la metodología.
- se solicitaba autorización al empleado para la toma de las imagénes y para qué ámbito.
- se solicitaba la aceptación de los términos en los que se explicaba que se incurriría en una falta grave, y posible motivo de despido procedente, el hecho de no atenerse a este sistema, no cumplir con los horarios pactados según el propio sistema y no facilitar la "correcta toma de la fotografía" en el momento del fichaje.

6.6.4.2. La correcta toma de la fotografía

Aunque pueda parecer banal, esta parte requería de un cierto estudio ingenieril para responder a las siguientes cuestiones:

1. ¿Qué es una correcta toma? Sobre encuadre e iluminación

2. El problema de las alturas
3. Metodología para un fichaje eficiente.

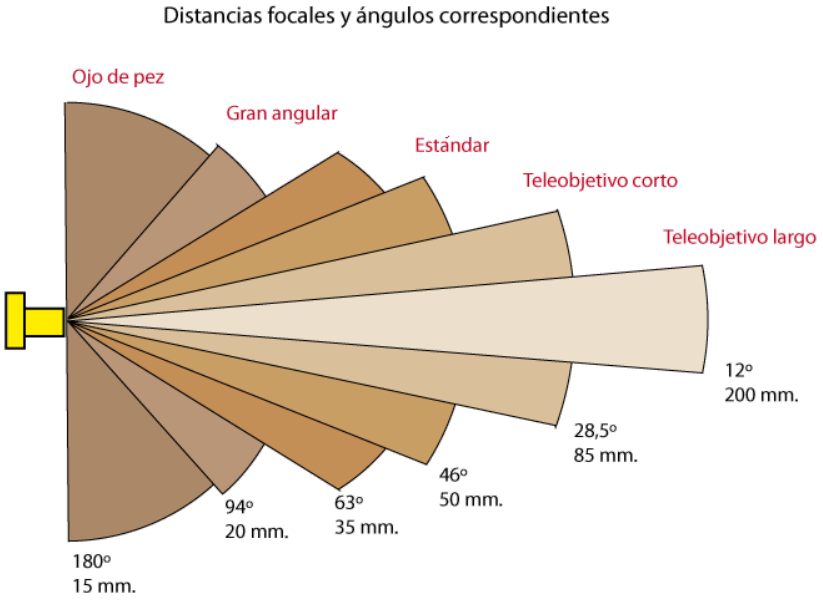
6.6.4.2.1. ¿Qué es una correcta toma? Sobre encuadre e iluminación

No es interés de esta memoria hacer un estudio sobre técnicas de fotografía, sin embargo, sí que es cierto que existen una serie de factores a tener en cuenta cuando hablamos de tomar una imagen para mejorar la robustez de nuestro sistema. Lo primero que a uno le viene a la cabeza es que si uno pone una cámara fija, debe calcular correctamente para que la toma de la imagen sea lo más precisa posible, ya que el objetivo no es tomar una buena foto, sino identificar sin lugar a dudas a la persona que hace el *check*.

Evidentemente, el factor encuadre, o qué existe en el rango de captación del sensor fotográfico, es importante ya que el sistema no podrá "adaptarse" como haría un fotógrafo al ser un sensor fijo. Esto trae a colación la pregunta de cuál es el ángulo adecuado, si es necesario ampliar el rango con un objetivo tipo gran angular, qué es necesario que se vea y por qué. Por ello, en esta iteración, la primera en la que aparece este elemento, se han tomado las siguientes decisiones con respecto a este factor:

Img. 20: Módulo PiCAM NOIR

- Se utilizará el módulo Pi Camera[53] de la Raspberry PI, que posee un enfoque fijo de 1mm hasta el infinito y un ángulo de visión similar al de un objetivo estándar - gran angular[54] de 35mm, que viene a ser la distancia focal más típica que se utiliza en el cine y en los retratos generales.



Img. 21: Diagrama de distancias focales y ángulos correspondientes

[53] https://www.raspberrypi.org/documentation/hardware/camera/
[54] https://es.wikipedia.org/wiki/Objetivo_gran_angular

- No se aplicará un objetivo adicional al sensor para ampliar el rango de visión. De hacerlo se ampliaría el rango por encima de los 50˚- 60˚ y deformaría la imagen, en lo que llamamos comúnmente efecto "ojo de pez[55]". Si las fotografías las va a revisar un humano, le costará más reconocer las personas representadas si están deformadas. Si el sistema va a ser validado por un algoritmo, ya no es tan importante esta deformación. Por ello, se decide dejar el rango por defecto del módulo por ahora.

Img. 22: Ejemplo de distorsión Ojo de Pez

- Según estudios[56], el factor más importante de cara al reconocimiento humano de un rostro son los ojos, y de manera secundaria, la nariz y la boca. Dado que el pelo puede variar, es un factor que se suele descartar. A nivel computacional, también son parte de los factores utilizados, junto con la forma de la cara. Por tanto, lo importante que debe intentar mostrarse en cada imagen es el rostro, desde la barbilla hasta las cejas, descartando el resto por ser información ambiental no esencial.

Por otro lado, el factor de la iluminación, o cómo afecta las diferentes condiciones de iluminación al sensor y a la propia imagen a la hora de facilitar la identificación de las personas. Es decir, podemos colocar un sensor al aire libre que durante el día proporcione fotografías con una exposición adecuada de luz, pero el problema viene al anochecer. También se ha de tener en cuenta los elementos ambientales que puedan afectar en algún momento a la imagen, como zonas reflectantes, puntos de luz, etc. Por tanto se decide lo siguiente:

- No se recomienda instalar un flash luminoso, ya que implicaría más lógica electrónica con una fotoresistencia, aparte de la molestia que implica recibir un impacto luminoso de flash en la cara cada vez que entras o sales del centro, pudiendo incluso llegar a molestar de manera ambiental por la noche.
- Utilizar una cámara sin filtro infrarrojo (NOIR), o cámara de rango de luz combinado (luz visible + rango infrarrojo). Esto amplía la información de la imagen y facilita un posible algoritmo de detección facial en un futuro ya que existen estudios que demuestran que las imagen de rango combinado aumentan las probabilidades de generar aciertos y disminuir las falsas detecciones. Se propone como línea futura más adelante en la memoria.
- Al utilizar una cámara NOIR, se añade como mejora en próximos sprints un flash de tipo infrarrojo, mejorando las condiciones lumínicas de la imagen sin molestar al ojo humano ya que nuestro rango de visión no incluye la franja infrarroja y por tanto no detectaremos dicho impacto de luz. De hecho, se puede lanzar siempre, sin

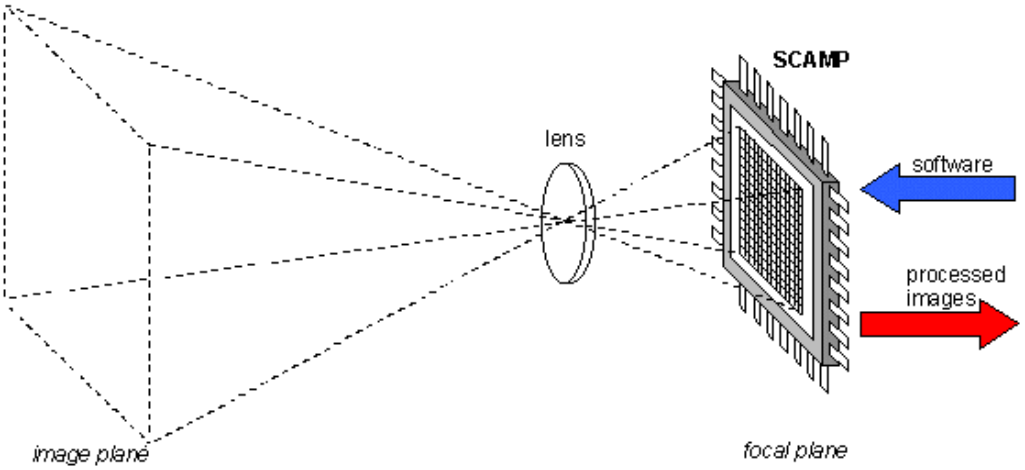
[55] https://es.wikipedia.org/wiki/Objetivo_ojo_de_pez

[56] Mathias S. Keil. "I Look in Your Eyes, Honey: Internal Face Features Induce Spatial Frequency Preference for Human Face Processing". *PLOS Computational Biology.* número 5(3), marzo de 2009.

necesidad de un fotoresistor que nos ayude a decidir cuándo lanzar un flash o no, simplificando el diagrama electrónico.

6.6.4.2.2. El problema de las alturas

Dado el rango de visión[57] de la cámara, con una distancia focal de 35mm, y ángulo horizontal de 53˚ y vertical de 41˚ debemos calcular cuál es la altura óptima del sensor para que abarque el mayor rango de altura de rostros, priorizando desde el mentón a las cejas.



Img. 23: Diagrama de relación geométrica de imagen, lente y sensor

Tras diversas pruebas, se decide optar por las medidas recomendadas para un dispositivo que se enfrenta a una problemática similar: el videoportero[58]. Según la normativa para este tipo de dispositivos y según el Colegio Oficial de Ingenieros Industriales, las medidas de altura de instalación deben rondar entre 1,30m y 1,60m desde el suelo al objetivo de la cámara. En nuestro caso, optamos por una altura de 1.50m, calculando para la correcta exposición de rostro completo de personas con un rango de altura entre los 1,90m y los 1,50m, correspondiente a la inmensa mayoría de la población, siendo la estatura media en España de 1,766m para los hombres y 1,634m para las mujeres, según la Wikipedia[59].

6.6.4.2.3. Metodología para un fichaje eficiente.

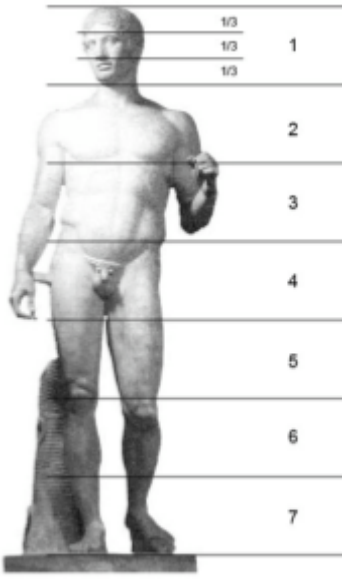
Por último, hablando en términos de eficiencia, definimos una metodología de fichaje óptima basándonos en la simplicidad de las instrucciones y el tiempo para llevarlo a cabo, por lo que se ha decidido que se proceda de la siguiente manera:

- Colocarse de frente al objetivo de la cámara

[57] https://es.wikipedia.org/wiki/%C3%81ngulo_de_visi%C3%B3n
[58] https://blog.fermax.com/esp/dudas-frecuentes-al-instalar-portero-automatico
[59] https://es.wikipedia.org/wiki/Estatura

- La distancia del cuerpo al sensor debe ser igual a la distancia del brazo estirado completamente
- Pase la tarjeta por delante del sensor con el brazo completamente estirado
- Mirar hacia la pantalla del sensor.

Este método es sencillo de recordar, garantiza un correcto encuadre en la inmensa mayoría de los casos (más del 90% de los casos) y propone un ángulo correcto del rostro. Además, es dinámico, al ser dependiente de la altura del sujeto debido a que la proporción de distancia del objetivo de la cámara estará marcada por la longitud del brazo, que, tomando el canon de Policleto[60] de proporcionalidad del cuerpo humano como base, corresponde alrededor del 3/7 de la altura. Se ha demostrado mejor que la mecánica de la medida fija (habitualmente 50cm), marcando una línea en el suelo.



Img. 24: Representación del canon de Policleto

[60] https://es.wikipedia.org/wiki/Canon_(arte)

7. Diseño

Como consecuencia del análisis, se plantean ahora las decisiones de diseño. Teniendo en cuenta que hemos elegido un modelo cliente-servidor, trataremos ambos por separado.

7.1. Modelo cliente - servidor

Repasando conceptos, este modelo de desarrollo del software plantea que existen dos agentes separados:

- Cliente: Quién inicia la transacción
- Servidor: Quién procesa la petición y le da respuesta

Estrictamente hablando, el propio sensor compondría en sí mismo un modelo cliente-servidor, siendo el cliente en este caso el trabajador, que ejecuta la acción de pasar la tarjeta, y el sensor la reconoce, la procesa y manda un mensaje por pantalla al trabajador. Además, el propio sensor luego se comporta como el cliente del sistema, enviando la información al servidor para que sea procesada.

A efectos macro del proyecto y por simplificar este concepto, trataremos al aparato sensor como un cliente del modelo y al software servidor como la parte servidora del modelo.

7.2. Cliente: Sensor Check-IN

Siendo el objetivo visible más novedoso de este PSI, se han aplicado una serie de decisiones valientes durante su diseño y manufactura. El cliente es por tanto un dispositivo hardware+software. Sin embargo, es importante destacar, que el modelo que nos ocupa en esta iteración es el modelo v4b, existiendo 4 versiones previas totalmente diferentes, con un proceso cada vez más sofisticado. Cabe remarcar que la release de cada modelo de sensor ha marcado cada *milestone* de versiones. No es tarea de esta memoria analizarlos todos, pero a continuación se hará una breve mención de las características concretas y alguna imagen del proceso de diseño y montaje, y el producto final en aras de entender el trabajo previo.

7.2.1. Release v0.1b

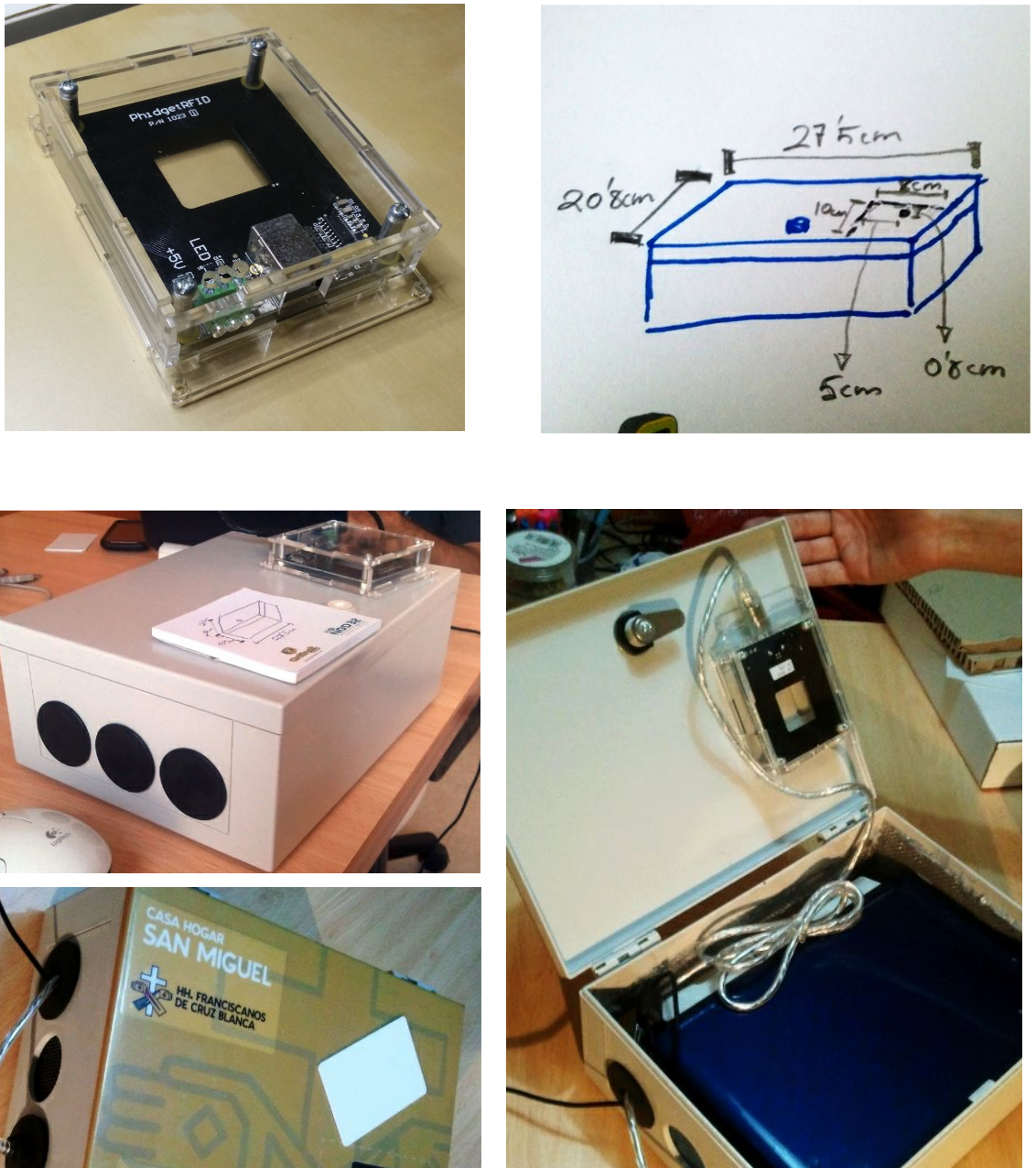
El primer sensor constaba de una caja de plástico que originariamente se utilizaba para las instalaciones eléctricas como caja de conexiones, adaptada y conectada a una toma de corriente.

Contenía:

- Un Acer Aspire One ZG5 de 9”, como hardware computacional
- Un lector USB de tarjetas RFID Phidgets 125Khzs

Características a remarcar:

- El sensor y el servidor eran un mismo dispositivo físico
- Solo almacenaba Check-IN y Check-OUT según la tarjeta que pasara cerca, encendiendo un led para indicar la lectura.



Img. 25: Imágenes de diferentes fases de construcción del sensor v0.1b

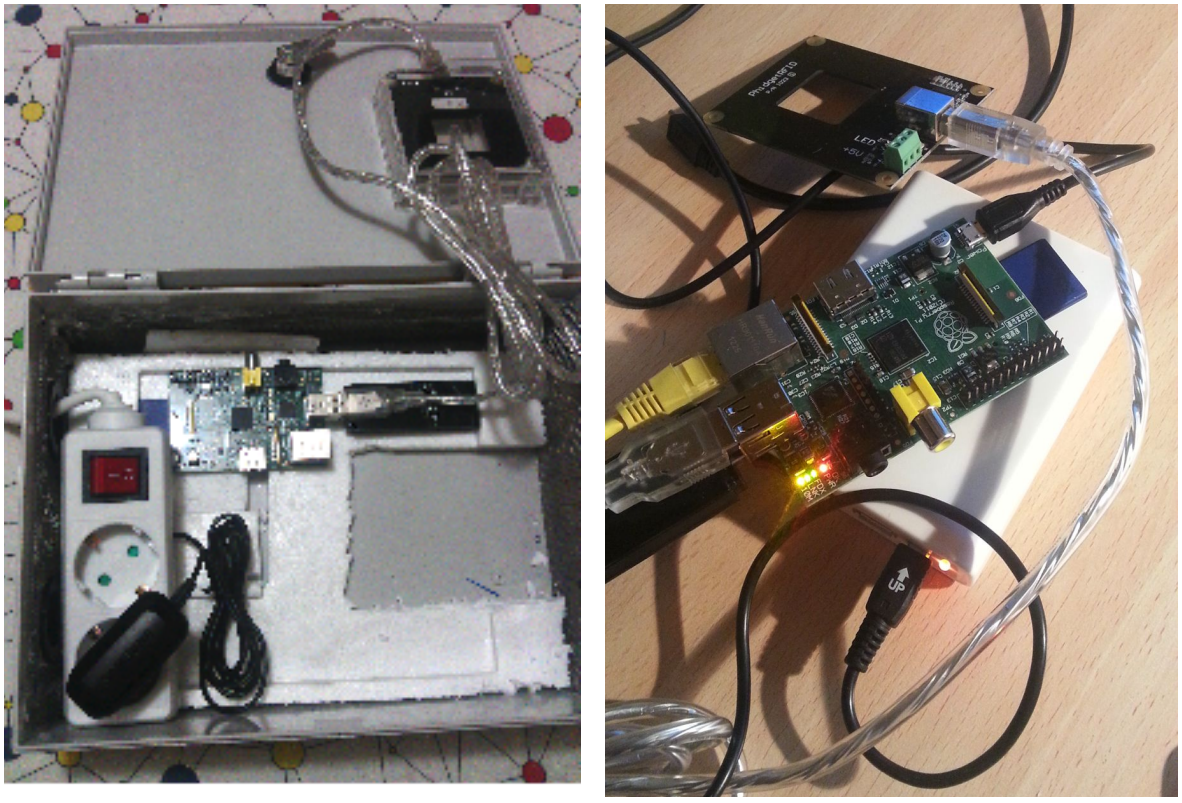
Img. 26: Imágenes de diferentes fases de construcción del sensor v0.1b

7.2.2. Release v1b

En la siguiente release se decide separar el sensor y el server en dos dispositivos diferentes. La caja continúa siendo la misma, pero el netbook Acer Aspire One pasa a ser el servidor y la Raspberry PI v1 model B, pasa ser el elemento computacional del sensor. Sin embargo, esta versión apenas duró instalada en favor de la v2b.

Características:

- Dispositivo sensor:
  - Raspberry PI v1 model B
  - Un lector USB de tarjetas RFID Phidgets 125Khzs
  - Led de lectura de tarjeta
  - Se genera el primer software independiente del software del servidor, adaptándolo al sistema Raspbian.
  - Conectado via Wifi con un dongle Conceptronic que cumplía la 802.11b, enviando datos por API al server.
  - Se le añade una batería para aumentar la robustez ante apagones
- Dispositivo servidor:
  - Un Acer Aspire One ZG5 de 9”, como hardware computacional
  - Metido en el Rack de la oficina técnica, bajo doble llave (oficina y rack)



Img. 27: Imágenes de diferentes fases de construcción del sensor v1b

7.2.3. Release v2b

La caja disminuye a un tamaño más manejable al adaptarse al tamaño de las Raspberry Pi v1 model B, utilizada en la release anterior. De hecho, la carcasa fue fabricada utilizando la caja de cartón de un Nexus One, vaciada y adaptada para que permitiera el flujo de aire.

Características:

- Dispositivo sensor:
  - Raspberry PI v1 model B
  - Un lector USB de tarjetas RFID Phidgets 125Khzs
  - Caja de cartón de un Nexus One, reutilizada y adaptada. 15,6 x 12,2 x 5,6 cm
  - Led de lectura de tarjeta
  - Conectado via Wifi con un nuevo adaptador más pequeño y con la norma 802.11g, mejorando la conectividad con la instalación inalámbrica existente, de la misma empresa de la Raspberry PI
  - Se elimina la batería por problemas de fiabilidad
- Dispositivo servidor:
  - Un Acer Aspire One ZG5 de 9”, como hardware computacional
  - Metido en el Rack de la oficina técnica, bajo doble llave (oficina y rack)

Img. 28: Imágenes de diferentes fases de construcción del sensor v2b

7.2.4. Release v3b

Esta release nunca llegó a producción, pero fue el primer intento de diseñar una caja específica para el sensor y aumentar sus características con una pantalla simple LCD de 1.8” y una cámara. Se detectaron algunos problemas de diseño y se decidió pasar directamente a la release v4b. Además, consistió en el primer esfuerzo por convertir el sensor en un elemento que aportara algún valor decorativo.

Características:

- Dispositivo sensor:
  - Raspberry PI v1 model B
  - Un lector USB de tarjetas RFID Phidgets 125Khzs
  - Caja de madera y metacrilato hecha a medida
  - Pantalla de 1.8” a color
  - Cámara PiCam
  - Led de lectura de tarjeta
- Dispositivo servidor:
  - Un Acer Aspire One ZG5 de 9”, como hardware computacional
  - Metido en el Rack de la oficina técnica, bajo doble llave (oficina y rack)

Img. 29: Imágenes de diferentes fases de construcción del sensor v3b

7.2.5. Release v4b: Un elemento decorativo

Una de las características más relevantes de la presente release y que marca muchas de las decisiones, es la de orientar el sensor a ser algo más que un sensor de fichaje. Puestos a tenerlo a la vista, la intención es que además fuera útil, aportando información de temperatura, noticias o algún hecho curioso. Si bien esto no se consiguió finalizar para esta release, si se sentaron las bases para que fuera posible.

Por ello, y orientado también a convertirlo en producto vendible a otros centros, se le desarrolló una imagen corporativa y un concepto adicionalmente, aparte de una carcasa a medida.

Es, además, en esta release que se introducen varios cambios de concepto encaminados en convertir este proyecto en un producto comercial viable:

- Pantalla táctil más grande. No se pretende por ser táctil que se permita interacción aún con el usuario, pero se abre esta posibilidad.
- La cámara sin filtro infrarrojo (NOIR), como se ha especificado en la fase de análisis.
- Se cambia el sensor de tarjetas por uno más simple, a través del puerto serial, eliminando el USB, disminuyendo drásticamente el precio (de 50€ a 6€), disminuyendo el consumo y simplificando el proceso de captación de los datos al no tener que pasar a través de un driver USB.
- Se sustituye el servidor por una Raspberry Pi v2 model B, simplificando el parque tecnológico y abaratando costes del lado del servidor. Esto permite además la posibilidad de no tener que depender del parque tecnológico local y que el software no tenga por qué ser instalable, sino en una imagen ISO que se instala en la tarjeta.

  Características:
  - Dispositivo Sensor:
    - Raspberry PI v1 model B
    - Lector serial RFID 125KHz
    - Caja de madera y metacrilato hecha a medida
    - Pantalla 2.8” LCD táctil a color
    - Cámara PiCam NOIR
    - Se introduce el tercer mensaje de un *check*: *Break*, para reflejar una pausa.
  - Dispositivo servidor:
    - Raspberry PI v2 model B
    - Metido en el Rack de la oficina técnica, bajo doble llave (oficina y rack)

7.2.5.1. Imagen corporativa de Check-IN y concepto

Se desarrolló los siguientes logos en su versión ampliada y simple, haciendo juego con el logo simple de la empresa que montaremos para esto (el círculo partido), y la unión de la “K” con el “-” y la “I” de “Check-IN”:

Img. 30: Logo corporativo de la versión v4b

No es el objetivo de esta memoria hacer un estudio y manual de imagen corporativa, baste decir, que se realizó éste generando las paletas de colores, las diferentes versiones del logo, etc.

Img. 31: Imagen concepto del sensor v4b

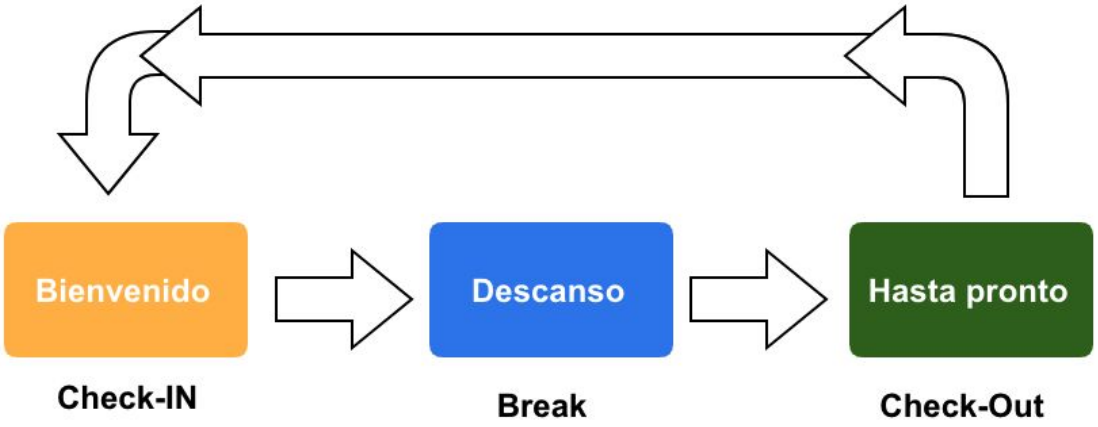
7.2.5.2. Simplificando: El carrusel de mensajes

Durante la fase de análisis, se ha especificado que existen ahora 3 señales: *Check-IN*, *Check-Out* y *Break*, que resuelven el valor de *¿Por qué?* del *check*. Al tener una pantalla en nuestro sensor, se hizo un estudio de la mejor manera de mostrar esa información de manera simple y haciendo el proceso sencillo para el trabajador, ya que el propio sensor no tiene ningún botón para interactuar por decisión durante el análisis.

Analizando el resto de dispositivos de la competencia, se ha observado que la práctica totalidad de las soluciones poseen algún tipo de botón para indicar la razón del

check, ya sea físico o mediante una pantalla táctil. Sin embargo, en aras de simplificar el proceso en nuestro caso, se ha diseñado un proceso tipo carrusel para especificar este parámetro necesario del *check* .

Se ha resuelto mediante mensajes en la pantalla siguiendo este diagrama de tipo carrusel:



Img. 32: Diagrama del carrusel de mensajes del sensor

De esta manera, en el momento que el empleado pase la tarjeta por la pantalla, el mensaje que se mostrará en pantalla será el que le corresponde si se acerca su hora de entrada o de salida, pero dispondrá de 5sg antes de que la señal se grabe en el sistema, tiempo durante el cual, si pasa de nuevo la tarjeta, se mostrará el siguiente mensaje según el diagrama de carrusel de arriba y se reinicia la cuenta de 5sg. Así, el proceso de establecimiento de la razón del *check* es tan sencillo como pasar la tarjeta por el sensor tantas veces como se quiera hasta que el mensaje sea el deseado, luego se espera unos 5sg y el mensaje queda especificado y grabado.

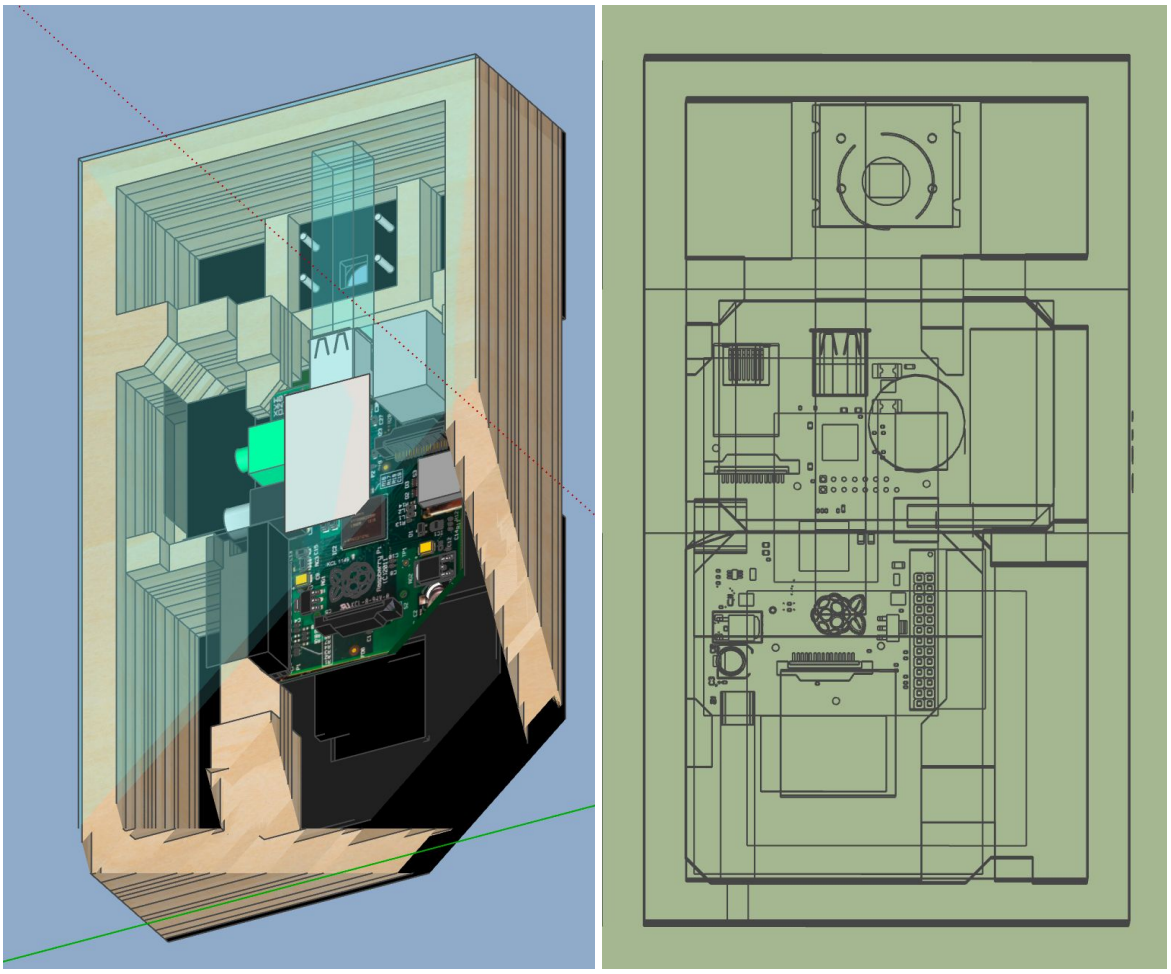
7.2.5.3. La Carcasa

En base al concepto diseñado, y al interés de convertir este modelo de sensor en un elemento decorativo, se eligió como materiales la madera y el metacrilato, teniendo especial cuidado en proponer un diseño que permitiera:

- La correcta ventilación de la electrónica interna
- Cierta protección a los elementos sensibles del sensor como la pantalla y la cámara
- La integración de todos los elementos en un bloque sin botones ni hendiduras a la vista
- El único cable que sale de la carcasa es el de corriente, que a su vez está fijado dentro para evitar la desconexión accidental
- El mínimo número de elementos móviles o de enganche.

Basándonos en modelos parecidos observados en Internet, y dado que el coste de producción era muy bajo y daba mucha libertad, se determinó generar una carcasa por capas, de manera que se monte colocando los elementos electrónicos a la vez, permitiendo

que las capas formen un pequeño laberinto interno vertical afianzando cada pieza con las diferentes configuraciones de las capas, de tal modo que al finalizar el montaje, se fijaran todas firmemente usando 4 tornillos pasadores en los cuatro extremos de la caja, permitiendo mantener una tensión de presión entre las capas y a su vez estas sobre los diferentes elementos electrónicos colocados dentro.



Img. 33: Detalle del modelado en 3D de las capas de la carcasa

La cuestión del material la marcó el coste y rapidez de producción, siendo la madera de balsa la elegida, ya que pueden comprarse en planchas y el corte es muy rápido usando una CNC (máquina de corte por láser[61]), por encima del plástico, el metal o las impresiones 3D.

7.2.5.4. El Hardware de la v4b

Parte	Producto	Características
Placa base	Raspberry Pi	v1 model B
Almacenamiento	Tarjeta SD	> 4GB

[61] https://es.wikipedia.org/wiki/Corte_con_l%C3%A1ser

		>= categoría 10
Cámara	Pi NOIR camera module	v1
Pantalla	2.8" TFT display + touchscreen for RPI by Lexi	320x480 resolution + RTC module
Lector tarjetas	Lector UART EM4100 RFID 125KHz	-
Adaptador Wifi	Wi-Pi dongle	802.11g WPA/WPA2
Alimentador	Power supply	2.5A

Tabla 7: Hardware del sensor v4b

Se debe remarcar que la Raspberry Pi no incorpora una pila para almacenar la fecha y la hora del dispositivo una vez se apaga este, por lo que cada vez que se enciende tras un apagón comienza el 1 de Enero de 1970 (fecha POSIX) y uno de los primeros procesos que hace cuando se conecta a Internet es consultar el NTPD (Network Time Protocol Daemon) configurado para establecer la fecha y la hora. Es recomendable instalar un módulo RTC (Real Time Clock module), el cual consiste en un pequeño dispositivo con una pila CR2032 de Litio, almacena y mantiene la fecha y la hora por varios años. Hay que configurar la Raspberry Pi para que cargue la fecha de los registros de ese módulo a través de los pins GPIO.

7.2.5.5. Software de la v4b

Concepto	Producto	Versión
Sistema Operativo	Linux Raspbian	-
Lenguaje de Programación	Python	3
	Bash	-
Framework web	Django	2
API	Django-Rest Framework	>= 3.9

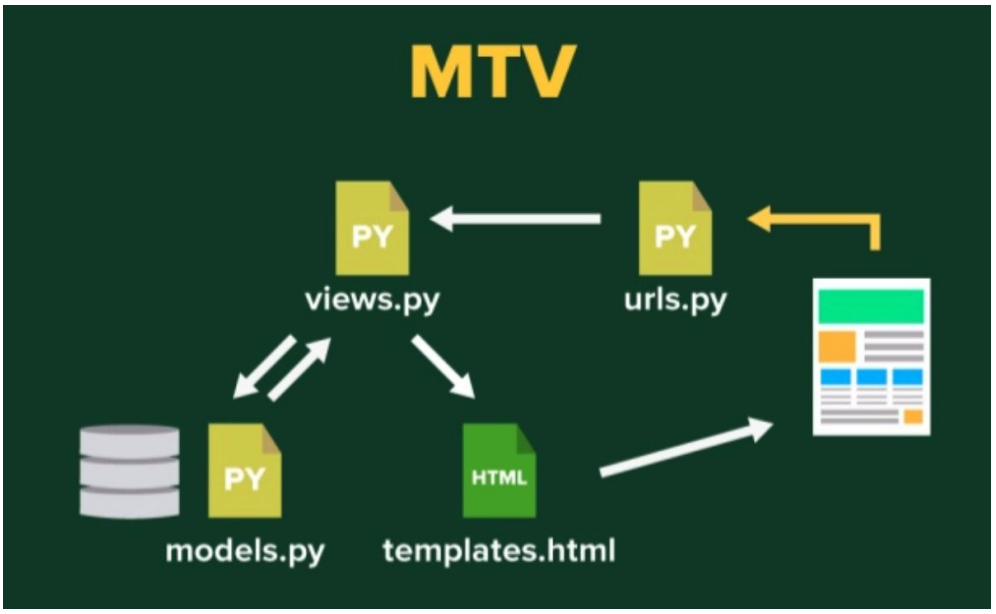
Tabla 8: Software del sensor v4b

7.3. Servidor: Check-IN Server

Por contra, el servidor ha sufrido pocas modificaciones importantes durante las diferentes iteraciones ya que el desarrollo del software de este y sus funcionalidades se establecieron y validaron desde el primer MVP, siendo el sensor el que más mejoras requería de cara a la usabilidad y robustez.

7.3.1. Software

A nivel de software, el patrón arquitectónico de desarrollo que subyace en el uso del framework Django es el de modelo-vista-controlador[62], o MVC, aunque en el caso de este framework con alguna diferencia de concepto (técnicamente utiliza el controlador como vista y a la vista la llama *template*), por lo que los expertos lo llaman MTV (Model-Template-View).



Img. 34: Diagrama de funcionamiento del modelo Model-Template-View

Este paradigma pretende separar los datos, la lógica de negocio y su representación, manteniendo procesos separados relacionados con la visualización y el tratamiento de los datos. Este patrón se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

En el caso concreto de Django, los diferentes elementos del flujo son:

1. El Cliente que solicita una URL
2. Django usa su enrutador ("urls.py") para ejecutar la vista ("views.py") más adecuada.
3. La vista seleccionada recoge los datos enviados en la petición (por ejemplo, usando GET o POST), consulta con los modelos ("models.py") de datos y utiliza el ORM[63] (capa que se comunica con la base de datos).
4. Los datos involucrados son tratados, y si es el caso, devueltos a la vista.

[62] https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador
[63] https://es.wikipedia.org/wiki/Mapeo_objeto-relacional

5. La vista renderiza una plantilla (“templates.html”) usando los datos recibidos como contexto y la devuelve al usuario
6. El cliente recibe un código HTML+CSS+JS+media que se muestra en su navegador.

Este esquema de flujo es igual de válido si el cliente es un humano, como si es un sensor a través de la API.

En cualquier caso, los requisitos software son las siguientes:

Lado	Concepto	Producto	Versión
Backend	Sistema Operativo	Linux	-
	Motor de base de datos	PostgreSQL	> 9.4
	Lenguaje de Programación	Python	3
	Framework web	Django	2
	API	Django-Rest Framework	>= 3.9
FrontEnd	Estructura y visualización	HTML + CSS	5 y 3
	Framework web	Bootstrap	4
	Interacción	JS	>= ES5

Tabla 9: Requisitos de Software del servidor

Es importante remarcar que Django posee un Mapeador Objeto-Relacional, lo que significa que no tocaremos prácticamente nada de SQL o PostgreSQL, siendo el propio Django el encargado de servir como capa intermedia ORM de comunicación con la base de datos.

7.3.2. Hardware

A nivel hardware, ha pasado en esta última versión v4b de un proceso web en un servidor Linux de escritorio tipo Ubuntu Server, a correr en una plataforma Raspberry Pi (abreviada como RPI), en una distribución llamada Raspbian, basada en Debian[64]. En realidad, el trasvase en cuanto a código de una plataforma a otra es transparente, ya que se basa en el uso de tecnología web programada en Python[65] usando el framework de desarrollo Django[66], siendo estas tecnologías multiplataforma. Como curiosidad, a nivel máquina sí que existe un cambio, ya que pasa de ser un lenguaje interpretado traducido a procesadores tipo CISC[67] (en el caso concreto de este proyecto de Intel x86[68]) a tipo RISC[69] de ARM[70]. Este cambio de plataforma, además, puede permitir que el servidor se distribuya

[64] https://es.wikipedia.org/wiki/Debian
[65] https://es.wikipedia.org/wiki/Python
[66] https://es.wikipedia.org/wiki/Django_(framework)
[67] https://es.wikipedia.org/wiki/Complex_instruction_set_computing
[68] https://es.wikipedia.org/wiki/Intel_Corporation
[69] https://es.wikipedia.org/wiki/Reduced_instruction_set_computing
[70] https://es.wikipedia.org/wiki/Arquitectura_ARM

en imágenes ISO para tarjetas microSD de las Raspberry Pi que se utilicen para estos proyectos.

De esta manera, también se ahorra costes a las empresas, ya que no deben tener un servidor específico para este proceso en particular, ahorra complejidad, ya que la arquitectura está controlada, y ahorra espacio, ya que el factor forma de la RPI es muy pequeño en comparación con un ordenador de escritorio, servidor o portátil. La carga de trabajo necesaria para atender a estos procesos se adapta perfectamente a este hardware, así que se optimizan recursos en definitiva.

A modo resumen, esto son los requisitos Hardware mínimos:

Parte	Producto	Características
Placa base	Raspberry Pi	>= v2 model B
Almacenamiento	Tarjeta microSD	> 8GB >= categoría 10
Carcasa	Carcasa de Raspberry Pi	-
Alimentador	Power supply	2.5A

Tabla 10: Requisitos de Hardware del servidor

7.3.3. Base de Datos

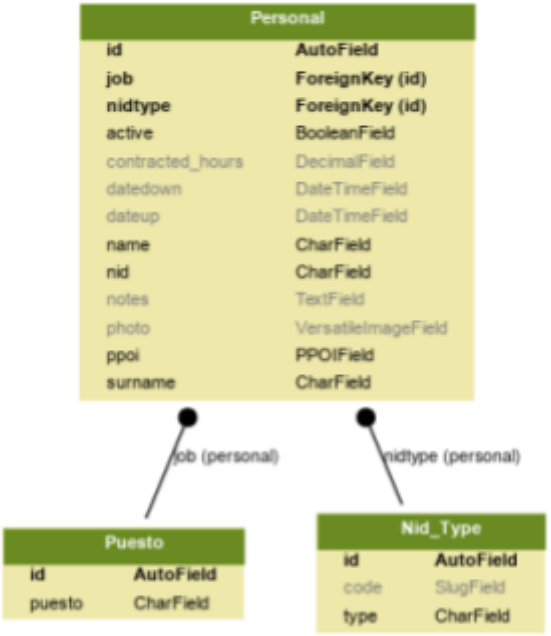
El modelo de datos del sistema se lleva acabo usando SQL como lenguaje base, y por tanto es de tipo relacional, y se ejecuta sobre un motor PostgreSQL[71]. Se decidió por este motor sobre MySQL[72] por tener una robustez mayor y un nivel de seguridad más alto. Es cierto que al nivel que se mueve esta aplicación en este punto no aporta grandes ventajas, y simplifica mucho más el proceso usar MySQL, pero era un requerimiento a medio plazo que se decidió aplicar desde ya para evitar incompatibilidades futuras. En cualquier caso, como se ha comentado antes, se trabajará mayoritariamente con el ORM.

7.3.3.1. Estructura de datos

Nuestra aproximación para este sistema consta de 3 entidades importantes:

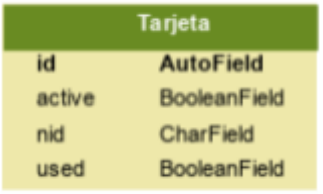
- **Personal**:
  Define la entidad del empleado a controlar, almacenando los datos básicos como nombre y apellidos, documentación de identidad, puesto, foto, etc.

[71] https://es.wikipedia.org/wiki/PostgreSQL
[72] https://es.wikipedia.org/wiki/MySQL



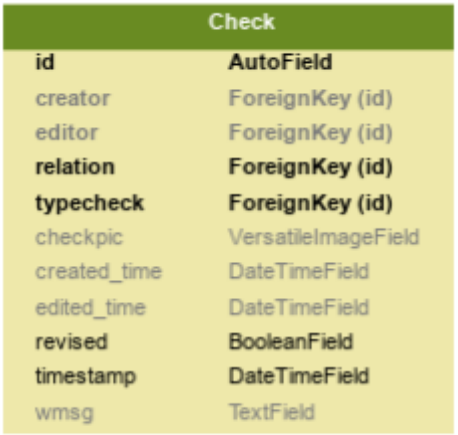
Img. 35: Diagrama de la entidad Personal

- **Tarjeta**:
  Es la entidad que almacena las identificaciones de las tarjetas y que luego serán vinculadas a los trabajadores.
  Posee un campo para saber si está en "uso" (ayuda en las consultas inversas, al no tener que hace consultas de si esta tarjeta está asignada a alguien), y otro para saber si está activa, para que sea sencillo desactivarla cuando se requiera controlar esa eventualidad (por ejemplo, que la tarjeta se pierda).



Img. 36: Diagrama entidad Tarjeta

- **Check**:
  Es el modelo que gestiona los *check*, almacenando la info que se determinó en la fase de análisis:
  - "relation": Es la relación tarjeta_empleado que ha generado el *check*.
  - "creator": Indica el "Dónde", siendo el creador del check el usuario asignado al sensor desde el que se ha generado este, o el usuario del administrador si se ha hecho manualmente.
  - "timestamp": Almacena la marca de tiempo en formato tiempo POSIX.
  - "typecheck": Refleja el "Por qué", pudiendo ser inicialmente de tipo Check-In (entra en el centro), Break (pausa



Img. 37: Diagrama entidad Check

corta), Check-Out (sale del centro). Este campo es una clave de otra tabla para poder indicar más tipos de checks en el futuro si fuera necesario.

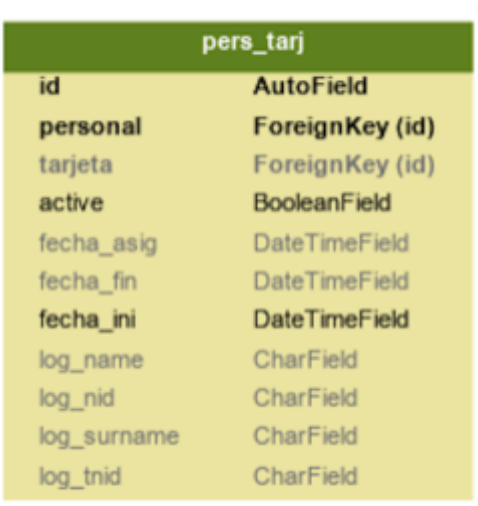
- "checkpic": Es la imagen tomada del check.

Existen otros campos de control como cuando fue la última vez que se editó o cuando se creó, o el mensaje de warning si se generara una notificación por el *check*.

- **Relación Personal - Tarjeta**

Según hemos diseñado nuestro sistema, la relación Personal - Tarjeta puede ser M:M, es decir, una tarjeta puede estar asignada a varias personas (por ejemplo, tarjeta maestra o grupal) y una persona puede tener varias tarjetas (por ejemplo, técnico de Recursos Humanos). Sin embargo, a través de interfaz no se permitirá asignar una tarjeta ya en uso (de ahí la importancia del campo "used"), por lo que hablamos de una "teórica" relación "Personal - 1:M - Tarjeta".

Por tanto la relación debe estar en una tabla aparte que denominamos "pers_tarj", donde se relaciona el Personal, la Tarjeta, las fechas de validez y si está activo. Este enfoque además prepara el terreno para que en un futura pueda definirse incluso permisos a las tarjetas, tales como puertas permitidas y demás.



Img. 38: Diagrama relación pers_tarj

Los campo "log_" son una solución al problema de incoherencia si un personal es eliminado, pero la relación ya había generado *checks*. De esta manera, la información se guarda en la relación aunque el usuario haya sido eliminado. El ejemplo más claro de esta funcionalidad radica en el hecho de que un empleado deja de trabajar en el centro, pero ya había usado su tarjeta para acceder a ciertos sitios (si se hubiera implementado ya el acceso de cerraduras inteligentes), o se mantiene el histórico de *check*s durante x años por ley. De esta manera, se tiene una medida de seguridad. Evidentemente, esto se podría solventar también definiendo una correcta estrategia de *contraints* y acciones "on_delete", "on_update" y demás señales que implementa MySQL y PostgreSQL para mantener la coherencia de la información relacionada. Se planifica para sprints futuros.

En el siguiente diagrama se puede apreciar un mapa completo de todo este modelo de datos, al margen de los modelos internos de gestión. Se aprecian modelos adicionales, como por ejemplo el de "horarios", que permitirá calcular el desvío de tiempo de retraso con respecto a la hora la que se ha establecido el turno, además de cuanto es el margen permisible para que se considere una falta.

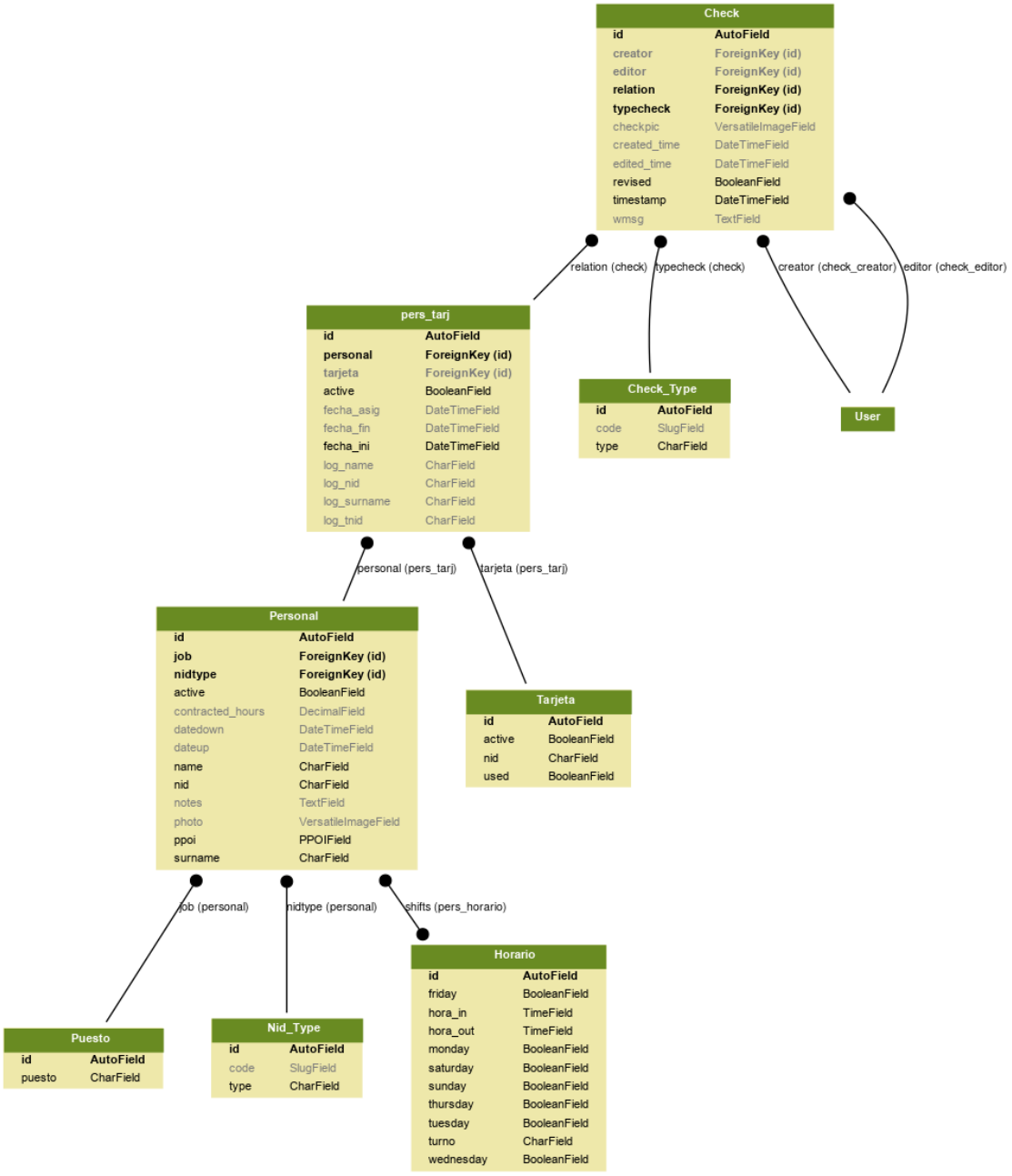
El sistema tiene un proceso de notificaciones, en las que se diferencian, por ahora, 3 tipos de Warnings:

- Azul: Retraso de menos del tiempo de margen establecido en el turno. Típicamente de 5 minutos, lo que implica que cualquier persona que haya

llegado entre las 8.01 y las 8.06, será notificado como un warning menor al grupo administrador.

- Rojo: Si el retraso supera el margen o ha pasado media hora tras la hora normal de fichaje.
- Amarillo: Si un personal supera las horas contratadas en el mes en curso.

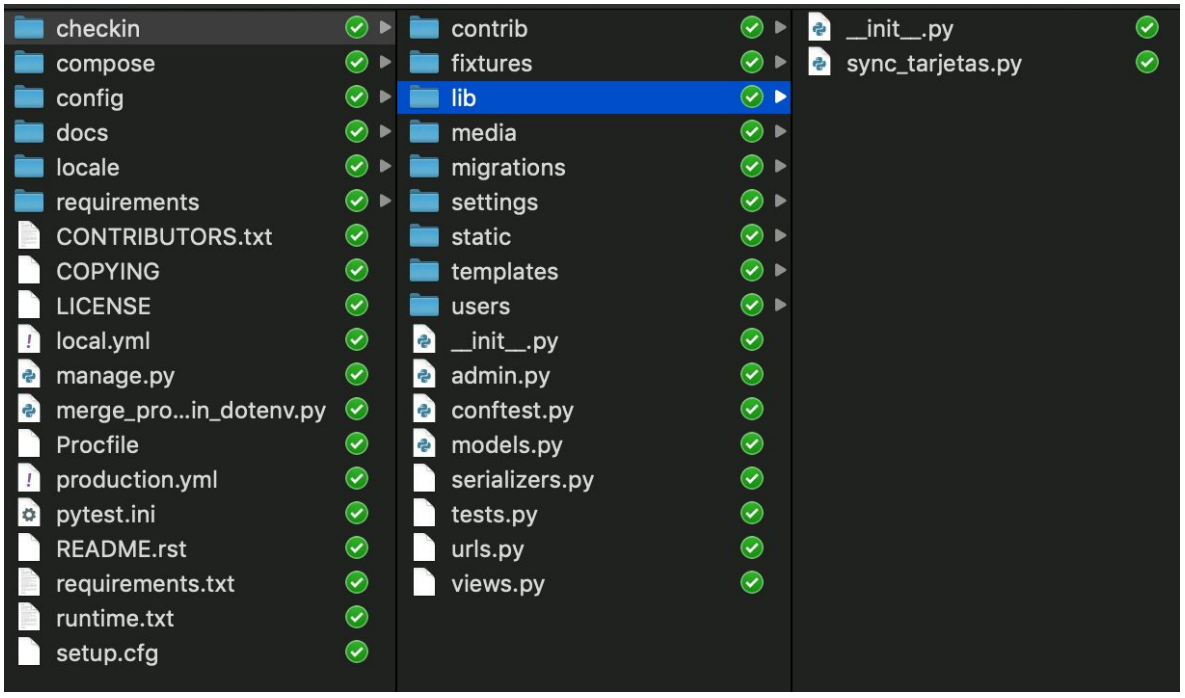
Estas notificaciones, por ahora solo se avisan al técnico de recursos humanos (admin), que puede descartarlas (marcarlas como vistas), filtrarlas, etc. Las dos primeras se almacenan en los propios *checks*. La última es un proceso de sincronía programada para ejecutarse cada día.



Img. 39: Detalle de la organización de las tablas de la base de datos

7.3.4. Estructura de la aplicación

La estructura final de ficheros del proyecto corresponde en su gran mayoría a la organización necesaria de las aplicaciones basadas en el framework Django, por lo que me limitaré a reflejar aquellas que son de interés.



Img. 40: Detalle de la organización de ficheros del servidor

Según el desarrollo de un proyecto en Django, las siguientes carpetas son las importantes:

- "checkin": es la "app" de Django, y dentro se encuentran los ficheros que la definen, como:
    - "models.py": Definición de los modelos de datos.
    - "urls.py": URLs de enrutamiento propias internas de esta app.
    - "views.py": Vistas a ejecutar desde el enrutador
    - "serializers.py": Vistas de la API
    - "admin.py": Definición de aspecto del administrador
    - Carpeta "templates": Donde se almacenan las plantillas del proyecto, con cada página maquetada diferente de las estándar.
    - Carpeta "media": Donde se almacenan las imágenes de los trabajadores y las de los checks.
    - Carpeta "static": Donde se almacena material tipo assets de las diferentes apps o secciones que utilice la app "checkin". Sino, las toma de la carpeta

donde se instalen, dentro de la propia carpeta de Python, como apps instalables (habitualmente, `/usr/lib/python2.7/dist-packages`).
    - Carpeta “lib”: Funciones adicionales.

- Carpeta “config”: Donde se encuentran las configuraciones generales del todo el sistema, por ejemplo, las “urls.py” o enrutadores generales.

El resto de carpetas y ficheros son importantes, pero no relacionados con el desarrollo del proyecto. Por ejemplo, la carpeta “compose” define las reglas de montaje y ejecución de un entorno *dockerizado* con *docker-compose*.

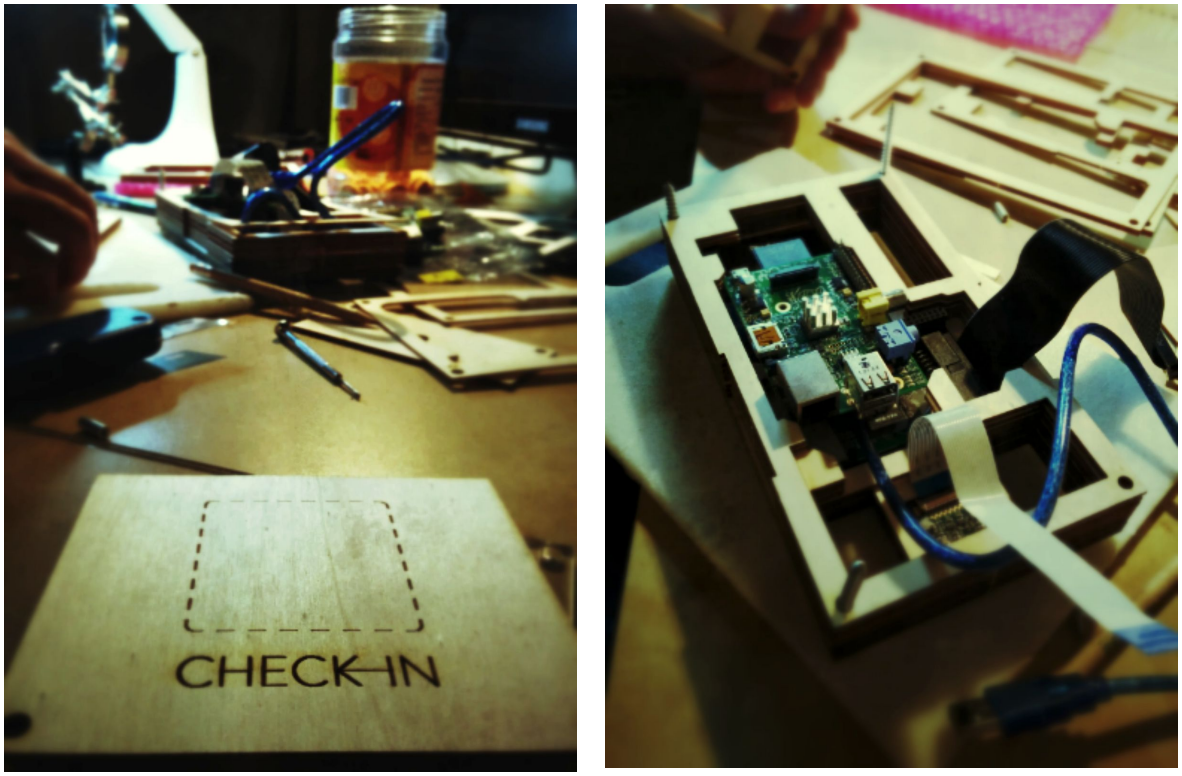
Otro ejemplo, la carpeta “requirements”, define las librerías que se requieren para que el sistema funcione.

8. Implementación

Una vez establecidos los pasos del diseño de este proyecto, se procede a la implementación.

8.1. Sensor Check-In

La carcasa es impresa usando una CNC, y montada capa a capa con los elementos tecnológicos arriba referidos, creando un dispositivo entero, sólido y funcional para leer tarjetas, almacenar imágenes y comunicarse con el servidor:



Img. 41: Detalle del montaje del sensor

El software instalado dentro consta de un sistema operativo Raspbian y un par procesos tipo daemon, uno que gestiona de manera monolítica el proceso de captura de imágenes y muestra de mensajes en el carrusel, y otro que gestiona el envío de los datos por API al server.

El proceso demonio de captura de imágenes y checks, comienza con un set up de algunas opciones, donde se observa las URLs del servidor y de almacenamiento de imágenes, así como carpeta de logs y el margen en sg para considerar el check introducido por si el trabajador prefiere hacer uso del carrusel de mensajes:


```
DEFAULTS = {
```

```
    'loglevel': 0,
    'server_url': 'http://localhost:8000/',
    'spool_directory': '/var/spool/checkin/',
    'picfolder': '/var/spool/checkin/pics',
    'picresolution': (640, 480),
    'fontfolder': os.path.join(os.path.dirname(__file__), 'fonts'),
    'imgfolder': os.path.join(os.path.dirname(__file__), 'imgs'),
    'checking_time': 5,  # Time in seconds,
    'LOGFORMAT': "%(asctime)-15s %(levelname)s: %(message)s",
    'PIDFILE': "/tmp/checkin-daemon.pid",
    'LOGFILE': "/var/log/20h/checkind.log",
    'LOGLEVEL': logging.DEBUG,
    'BLACK':  (  0,   0,   0),
    'WHITE':  (255, 255, 255),
    'RED'  :  (255,   0,   0),
    'GREEN':  (  0, 255,   0),
    'BLUE' :  (  0,   0, 255),
    'CYAN' :  (  0, 255, 255),
    'MAGENTA':(255,   0, 255),
    'YELLOW': (255, 255,   0),
}
```

Cód. 1: Detalle de configuración del demonio capturador de eventos del sensor.

Aquí un detalle del código de captura de la foto, utilizando una librería de threads en python para permitir que pueda ser lanzado este proceso en un hilo diferente y aprovechar capacidad multicore de futuras versiones del sensor:

```
class Photo(threading.Thread):
    """
    Takes a photo from camera and saves it to a file in spool directory
    """
    def __init__(self, timestamp):
        threading.Thread.__init__(self)
        self.timestamp = timestamp

    def run(self):
        with picamera.PiCamera() as camera:
            try:
```

```
                camera.resolution = DEFAULTS['picresolution']
                camera.capture(os.path.join(DEFAULTS['picfolder'],
'%s.jpg' % self.timestamp))
                return True
            except Exception as error:
                logging.warning("Couldn't take the pic (Error: %s" %
error)
                return False
```

Cód.2: Detalle del toma de imágenes

Para la muestra en pantalla de las imágenes se ha utilizado la librería pygame[73], que funciona de interfaz de programación para pintar imágenes en la pantalla y gestionar su flujo, pasándole como parámetros el dispositivo de la pantalla, y algunas opciones, como visibilidad del ratón o fondos de pantalla.

```
# Setup Screen PyGame
        ######################################
        os.environ["SDL_FBDEV"] = "/dev/fb1"
        pygame.init()
        # set up the window
        self.screen = pygame.display.set_mode((320, 240), 0, 32)
        pygame.mouse.set_visible(0)
        pygame.display.set_caption('Drawing')

        # Fill background
        self.background = pygame.Surface(self.screen.get_size())
        self.background = self.background.convert()
        self.background.fill(DEFAULTS['BLACK'])
        # Display some text
        font = os.path.join(DEFAULTS['fontfolder'], "Roboto-Bold.ttf")
        font2 = os.path.join(DEFAULTS['fontfolder'], "Roboto-Thin.ttf")
        font3 = os.path.join(DEFAULTS['fontfolder'],
"RobotoCondensed-Regular.ttf")
        font4 = os.path.join(DEFAULTS['fontfolder'], "Roboto-Light.ttf")
        self.fonthora = pygame.font.Font(font, 90)
        self.fontmin = pygame.font.Font(font2, 90)
```

[73] https://www.pygame.org

```
        self.fontfecha = pygame.font.Font(font3, 34)
        self.fontmsg = pygame.font.Font(font4, 55)
```

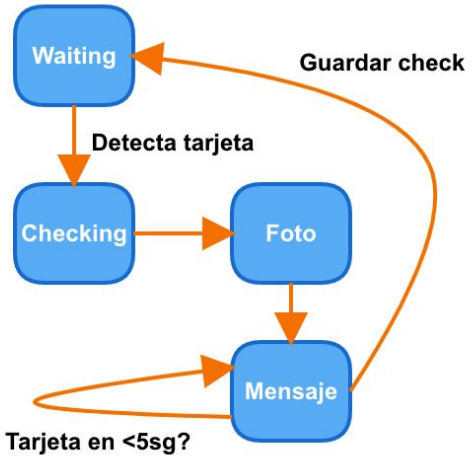
Cód.3: Detalle del código de inicialización de pantalla

Así, el sensor tiene un código principal en que gestiona el proceso de esperar por una tarjeta, y mientras muestra la hora y la fecha, o cuando detecta una tarjeta, mostrar las pantallas de los mensajes:

```
                else:
                    #logging.info("Reading if cards are avaliable...")
                    card = self.read_card()
                if card is not None:
                    # Get timestamp
                    timestamp = self.get_timestamp()
                    # Taking the picture to the spool directory
                    pic = Photo(timestamp)
                    pic.start()
                    # Change status to checking
                    self.status = self.STATUS_CHECKING
                    status = self.set_card_status(card)
                    # Show current card status to user
                    #TODO: Screen control with fade?
                    self.show_screen(status)
                    self.checking_start = datetime.now()
                    time.sleep(0.5)
                else:
                    self.show_screen()
                    time.sleep(0.5)
```

Cód. 4: Detalle del código de cambio de estado de esperando a checking, donde comienza la lógica

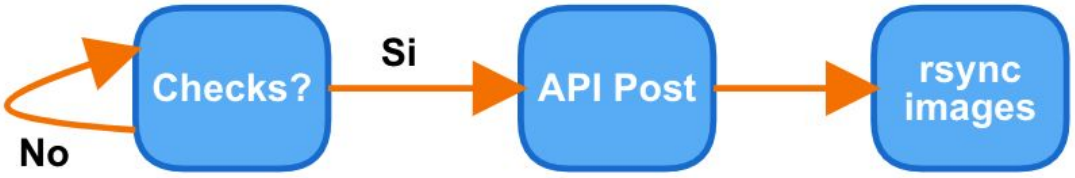
Siguiendo un esquema resumido en el siguiente diagrama:



Img. 42: Diagrama del demonio de captura de la tarjeta

Img. 43: Foto tomada durante un check-In

Para el proceso de envío al servidor, se envía de dos tandas, lanzadas por cron cada minuto, según el siguiente diagrama:



Img. 44: Diagrama del demonio de envío de datos al server e imágenes

Podría simularse utilizando un curl con una estructura similar a esta, siendo el valor creator un ID de usuario correspondiente al usuario del sensor que debe ser configurado antes en el servidor:

```
curl
-X POST
http://127.0.0.1:8000/checkin/checks/ -HGVjdG9yMQ==" -H
"Content-Type: application/json"
-d '{"relation":1, "timestamp":"2015-02-11T02:45:17Z",
"typecheck":2, "creator":3}'
```

Cód. 5: Ejemplo de CURL para actualización de checks

8.2. Servidor Check-In

Se introduce la RPI v2 en su caja y se conecta mediante cable al rack de conexiones del armario de servidor que existe en el Área Técnica, permitiendo estar disponible en la red interna del centro.

El sistema interno es un Raspbian también y la aplicación de servidor es una estructura web definida por el framework de desarrollo de Django y una base de datos tipo PostgreSQL.

Una vez lanzada, el servidor tiene 3 URLs de interés:

- `http://<IP/host>` : te transporta al entorno web del proyecto
- `http://<IP/host>/admin/` : entorno administración (donde se gestiona el check-in)
- `http://<IP/host>/api/` : Punto de acceso para la API.

```
def post(self, request, format=None):
       if request.DATA:
           try:
               relationObj =
pers_tarj.objects.get(tarjeta__nid=request.DATA["relation"], active=True)
           except Exception as e:
```

```
                return Response(
                    u"[E] Inconsistencia en BBDD: Varias relaciones activas con la
misma tarjeta ('%s'). Error: %s" % (request.DATA["relation"], e),
                    status=status.HTTP_400_BAD_REQUEST)
            try:
                obj_aux = Check()
                obj_aux.checkpic = "%s%s.jpg" % (SENSOR_FOLDER_REL,
request.DATA["timestamp"])
                request.DATA["checkpic"] = obj_aux.checkpic
                request.DATA["timestamp"] =
datetime.isoformat(datetime.utcfromtimestamp(float(request.DATA["timestamp"])))
            except Exception as e:
                return Response(
                    u"[E] Timestamp incorrecto: ('%s'). Error: %s" %
(request.DATA["timestamp"], e),
                    status=status.HTTP_400_BAD_REQUEST)
            request.DATA["relation"] = relationObj.id
            serializer = CheckSerializer(data=request.DATA)
            if serializer.is_valid():
                serializer.save()
                return Response(serializer.data, status=status.HTTP_201_CREATED)
            return Response("[E] Not valid: %s" % serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
        else:
            return Response(u"[E] Datos vacíos.",
status=status.HTTP_400_BAD_REQUEST)
```

Cód. 6: Detalle del código recepción de Checks vía POST

El fichero `models.py` de la aplicación Django (situado en `<directorio-del-proyecto>/checkin/models.py`), contiene la definición de los modelos de la base de datos y es el fichero central alrededor del que se mueve el ORM de Django. La nomenclatura de Python es bastante fácil de entender, así que con poner el fichero es bastante explicativo sobre cómo se organizan los modelos y algunas funciones auxiliares:

```
# Create your models here.
# PUESTO MODEL
# --------------------------------
class Puesto(models.Model):
    """
        Model Puesto:
```

```
            Puestos que pueden desempeñar los Trabajadores.
            -Campos:
                * Puesto
    """
    puesto = models.CharField(
        "Puesto",
        max_length=100,
        blank=False, null=False,
        help_text="Puesto que puede desempeñar un trabajador."
    )

    class Meta:
        verbose_name = _('Puesto')
        verbose_name_plural = _('Puestos')

    def __str__(self):
        return self.puesto


# Nid_Type MODEL
# ---------------------------------
class Nid_Type(models.Model):
    """
        Model Nid_Type:
            Tipo de documento de identificación.
            -Campos:
                * Type
    """

    type = models.CharField(
        "Tipo",
        max_length=50,
        blank=False, null=False,
        help_text="Tipo de Identificador"
    )
    code = models.SlugField(
        u"Código",
        max_length=50,
        blank=True, null=True,
        help_text=u"Código del tipo",
        editable=False)
```

```
    class Meta:
        verbose_name = _('Tipo de ID')
        verbose_name_plural = _('Tipos de ID')

    def save(self, *args, **kwargs):
        if not self.code:
            self.code = slugify(self.type)
        # Now, call the "real" save() method
        super(Nid_Type, self).save(*args, **kwargs)

    def __str__(self):
        return self.type


# PERSONAL MODEL
# ----------------------------------
class Personal(models.Model):
    """
        Stores the information of a worker of the company.

        - Dependencies:
            :model: `checkin.Nid_Type`: Type of documentation
            :model: `checkin.Puesto`: Job into the organization
            :model: `checkin.Horario`: Shifts the user is applied to.
    """
    name = models.CharField(
        "Nombre", help_text="Nombre del trabajador",
        max_length=50,
        null=False, blank=False)

    surname = models.CharField(
        "Apellidos",
        max_length=150,
        null=False, blank=False)

    nidtype = models.ForeignKey(
        "Nid_Type",
        verbose_name=u'Tipo de ID',
        on_delete=models.PROTECT)
```

```
    nid = models.CharField(
        "Número", help_text="Número de documento de identificación",
        max_length=12,
        null=False, blank=False)

    job = models.ForeignKey(
        "Puesto",
        verbose_name=u'Puesto',
        on_delete=models.DO_NOTHING
    )
    # TODO: Set on_delete with default Puesto

    photo = VersatileImageField(
        blank=True, null=True, upload_to='workers/',
        placeholder_image=OnStoragePlaceholderImage(
            path='assets/photodefault.png',
        ),
        ppoi_field='ppoi'
    )
    ppoi = PPOIField(
        'Image PPOI'
    )

    active = models.BooleanField("¿Activo?", default=True)
    dateup = models.DateTimeField(
        "Fecha alta", help_text="Fecha de alta en el sistema",
        blank=True, null=True,
        default=timezone.now)
    datedown = models.DateTimeField(
        "Fecha baja", help_text="Fecha de baja en el sistema",
        blank=True, null=True)

    notes = models.TextField(
        "Notas", help_text="Notas sobre el trabajador",
        blank=True, null=True)

    contracted_hours = models.DecimalField(
        "Contrato", help_text='Horas diarias contratadas',
        decimal_places=2, max_digits=6,
        blank=True, null=True)
    shifts = models.ManyToManyField(
```

```
        'Horario', related_name='pers_horario',
        blank=True)

    def complete_Name(self):
        """
        Creates a complete name set using 'Name + Surname' (if they exist)
        """
        s = ''
        if not self.surname is None:
            s = ' %s' % self.surname
        return "%s%s" % (self.name, s)

    class Meta:
        verbose_name = _('Trabajador')
        verbose_name_plural = _('Trabajadores')

    def __str__(self):
        s = ''
        if not self.surname is None:
            s = ' %s' % self.surname
        return "%s%s" % (self.name, s)

    @receiver(models.signals.post_save, sender="checkin19.Personal")
    def warm_Person_gallery_images(sender, instance, **kwargs):
        """Ensures Person head shots are created post-save"""
        person_img_warmer = VersatileImageFieldWarmer(
            instance_or_queryset=instance,
            rendition_key_set='image_gallery',
            image_attr='photo'
        )
        num_created, failed_to_create = person_img_warmer.warm()


# TARJETA MODEL
# ---------------------------------
class Tarjeta(models.Model):
    """
        Modelo Tarjeta:
            Gestión de las tarjetas existentes. La numeración de la tarjeta
            debe estar siempre en mayúsculas.
            - Campos
```

```
                * nid: Identificador de la Tarjeta
                * active: ¿Activa?
                * used: ¿Está en uso?
    """
    nid = models.CharField(
        "Identificador", help_text="Numeración de la tarjeta",
        max_length=12)
    active = models.BooleanField("¿Activa?", default=True)
    used = models.BooleanField("¿En uso?", default=False)

    def save(self, *args, **kwargs):
        self.nid = self.nid.upper()
        # Now, call the "real" save() method
        super(Tarjeta, self).save(*args, **kwargs)

    class Meta:
        verbose_name = _('Tarjeta')
        verbose_name_plural = _('Tarjetas')

    def __str__(self):
        return self.nid.upper()


# PERSONA_TARJETA RELATIONSHIP MODEL
# ----------------------------------
class pers_tarj(models.Model):
    """
        Modelo pers_tarj:
            -Campos:
                * tarjeta (FK): Tarjeta asociada a la relación
                * personal (FK): Personal asociado a la relación
                * fecha_asig: Fecha en la que se realizó la asignación (not
editable)
                * fecha_ini: Fecha en la que comienza a ser válida la tarjeta
                * fecha_fin: Fecha en la que se programa la finalización de la
asignación
                * log_tnid : ID de la tarjeta asociada.
                * log_nombre: Nombre del personal con la tarjeta asociada (not
editable)
                * log_apellidos: Apellidos del personal con la tarjeta asociada
(not editable)
```

```
                * log_nid: documento identificativo del personal con dicha tarjeta
asociada (not editable)
                * active: ¿Activa la relacion?
    """
    tarjeta = models.ForeignKey(
        'Tarjeta', help_text="Tarjeta que se asocia",
        blank=True, null=True,
        db_index=True,
        # limit_choices_to={'active': True, 'used': False})
        limit_choices_to={'active': True, },
        on_delete=models.DO_NOTHING)
    # TODO: Set on_delete with default card number

    personal = models.ForeignKey(
        'Personal', help_text="Personal al que se asocia la tarjeta",
        db_index=True,
        blank=False, null=False,
        limit_choices_to={'active': True},
        on_delete=models.DO_NOTHING)
    # TODO: Set on_delete with default person name

    active = models.BooleanField(
        "¿Activa?", default=True,
        null=False, blank=False)
    fecha_asig = models.DateTimeField(
        'Fecha de asignación',
        auto_now_add=True)
    fecha_ini = models.DateTimeField(
        'Fecha Inicio', help_text="Fecha de inicio de validez",
        blank=False, null=False,
        default=timezone.now)
    fecha_fin = models.DateTimeField(
        'Fecha fin', help_text="Fecha de fin de validez",
        blank=True, null=True)
    log_tnid = models.CharField(
        'ID tarjeta', help_text="Registro: ID de tarjeta asignada",
        max_length=20,
        editable=False, blank=True, null=True)
    log_name = models.CharField(
        'Nombre del trabajador', help_text="Registro: Nombre del trabajador",
        max_length=50, editable=False, blank=True, null=True)
```

```
    log_surname = models.CharField(
        'Apellidos', help_text="Registro: Apellidos del trabajador",
        max_length=150, editable=False, blank=True, null=True)
    log_nid = models.CharField(
        'ID del trabajador', help_text="Registro: NID del trabajador",
        max_length=12, editable=False, blank=True, null=True)

    class Meta:
        verbose_name = _(u'asignación')
        verbose_name_plural = _('asignaciones')
        unique_together = ("tarjeta", "personal", "active", )

    def save(self, *args, **kwargs):
        self.log_name = self.personal.name
        self.log_surname = self.personal.surname
        self.log_nid = self.personal.nid
        self.log_tnid = self.tarjeta.nid
        try:
            rel_anterior = pers_tarj.objects.get(id=self.id)
            if rel_anterior.tarjeta != self.tarjeta:
                rel_anterior.tarjeta.used = False
                rel_anterior.tarjeta.save()
        except self.DoesNotExist:
            pass

        if not self.active and self.tarjeta.used:
            self.tarjeta.used = False
            self.tarjeta.save()
        elif self.active and not self.tarjeta.used:
            self.tarjeta.used = True
            self.tarjeta.save()

        # Now, call the "real" save() method.
        super(pers_tarj, self).save(*args, **kwargs)

    # TODO: If we deactivate the card, we mark it as innused

    def pre_delete(self, *args, **kwargs):
        self.tarjeta.used = False
        self.tarjeta.save()
        # Now, call the "real" pre_delete() method.
```

```
        super(pers_tarj, self).pre_delete(*args, **kwargs)

    def delete(self, *args, **kwargs):
        self.active = False
        self.fecha_fin = timezone.now

    def __str__(self):
        return u"Tarjeta de %s. %s" % (self.log_name[0], self.log_surname)


# Check_Type MODEL
# ----------------------------------
class Check_Type(models.Model):
    """
        Model Check_Type:
            Tipo de check realizado
            -Campos:
                * Tipo

            Ejemplos:
              * IN
              * OUT
              * OTHER
              * unknown [...]
    """
    type = models.CharField(
        "Tipo",
        max_length=50,
        blank=False, null=False,
        help_text="Tipo de Check"
    )
    code = models.SlugField(
        u"Código",
        max_length=50,
        blank=True, null=True,
        help_text=u"Código del tipo")

    def save(self, *args, **kwargs):
        if not self.code:
            self.code = slugify(self.type)
        # Now, call the "real" save() method
```

```
        super(Check_Type, self).save(*args, **kwargs)

    class Meta:
        verbose_name = _('Tipo de Check')
        verbose_name_plural = _('Tipos de Check')

    def __str__(self):
        return self.type


# CHECK MODEL
# ------------------------------------
class Check(models.Model):
    """
         Modelo check:
            Almacena los checks del sistema y los vincula con los demás modelos.
            -Campos:
                * relation (FK): Relacion pers_tarj que generó el check
                * timestamp: Momento del check
                * typecheck (FK): in o out, entrada o salida, etc.
                * revised: Revisado si es una alerta
                * wmsg: Mensaje de alerta
                * creator: Quien creo este check: 0: Sistema automático, ID: id
del usuario
                * editor: Quien editó los detalles del check después de ser creado
                * revisor: Quien revisó el check después de ser creado
    """
    relation = models.ForeignKey(
        'pers_tarj', verbose_name=u'¿Quién?',
        blank=False, null=False,
        on_delete=models.DO_NOTHING
    )
    # TODO: Set on_delete with default person-card relationship

    timestamp = models.DateTimeField(
        'Hora', help_text=u"Fecha",
        blank=False, null=False,
        default=timezone.now)

    typecheck = models.ForeignKey(
        'Check_Type', verbose_name=u'Tipo de Check',
```

```
        on_delete=models.DO_NOTHING)
    # TODO: Set on_delete with default check_type
    revised = models.BooleanField("¿Revisado?", default=False)
    wmsg = models.TextField(
        "Alerta", help_text=u"Mensaje descriptivo de la alerta.",
        null=True, blank=True)
    checkpic = VersatileImageField(
        blank=True, null=True,
        upload_to=SENSOR_FOLDER_REL,
        default=None
    )
    creator = models.ForeignKey(
        auth_model, verbose_name=u'Creador',
        help_text='¿Quién crea el Check?',
        related_name='check_creator',
        # default=get_current_user,
        blank=True, null=True,
        on_delete=models.DO_NOTHING)
    # TODO: Set on_delete with default creator name

    editor = models.ForeignKey(
        auth_model, verbose_name=u'Editor',
        help_text='Último que editó el Check?',
        related_name='check_editor',
        blank=True, null=True,
        on_delete=models.DO_NOTHING)
    # TODO: Set on_delete with default editor name

    # revisor = models.ForeignKey(
    #     'auth.user', verbose_name=u'Revisor',
    #     help_text='Último que revisó el Check?',
    #     related_name='check_revisor',
    #     default=get_current_user,
    #     blank=True, null=True
    #     )
    created_time = models.DateTimeField(
        "Fecha de creación", help_text='¿Cuándo se creó el Check?',
        auto_now_add=True)
    edited_time = models.DateTimeField(
        "Fecha de edición", help_text='Última fecha de edición del Check',
        auto_now=True)
```

```
    class Meta:
        verbose_name = _('check')
        verbose_name_plural = _('checks')

    def save(self, *args, **kwargs):
        """
        Filling the rest of fields.
        """
        if not self.creator:
            self.creator = 0
        super(Check, self).save(*args, **kwargs)

    def __str__(self):
        return str(self.id)


# HORARIO MODEL
# --------------------------------------
class Horario(models.Model):
    """
        Clase Horario:
            Gestiona los diferentes turnos del centro.
            Campos:
            * turno: Nombre sencillo para el turno
            * hora_in: Hora de inicio del turno
            * hora_out: Hora de fin del turno
            * monday ... sunday: Dias de la semana en los que se aplica el turno.
    """
    turno = models.CharField(
        "Turno", help_text=u"Escriba un nombre para el turno",
        null=False, blank=False,
        max_length=100)
    hora_in = models.TimeField(
        "Entrada", help_text='Hora de entrada')
    hora_out = models.TimeField(
        "Salida", help_text='Hora de salida')
    monday = models.BooleanField(
        "Lunes",
        default=True)
    tuesday = models.BooleanField(
```

```
        "Martes",
        default=True)
    wednesday = models.BooleanField(
        u"Miércoles",
        default=True)
    thursday = models.BooleanField(
        "Jueves",
        default=True)
    friday = models.BooleanField(
        "Viernes",
        default=True)
    saturday = models.BooleanField(
        u"Sábado",
        default=True)
    sunday = models.BooleanField(
        "Domingo",
        default=True)

    class Meta:
        verbose_name = _('Horario')
        verbose_name_plural = _('Horarios')

    def __str__(self):
        return self.turno
```

Cód. 7: Detalle de los modelos de datos

Apartando el funcionamiento de una aplicación de Django, hay tres ficheros adicionales que pueden ser de interés:

- `checkin/admin.py`: Donde se define el comportamiento del área de administración
- `checkin/views.py`: Dónde se encuentra la definición de vistas de la aplicación.
- `config/base.py`: Dónde está la configuración básica de la aplicación, aunque se separan entornos, definiendo también `local.py` y `production.py` para añadir configuraciones específicas si se está desarrollando o está en producción.

8.3. Una pequeña aplicación más

Una de las cosas de la que nos dimos cuenta durante la implementación del proyecto fue que si bien las tarjetas podían leerse con el sensor, no habíamos tenido en cuenta que el técnico de Recursos Humanos, no tenía forma de saber que código identificaba una tarjeta si la tenían en la mano, a menos que imprimiera todos los códigos en alguna de las caras de la tarjeta y luego al comprobarlas en el propio sistema o darlas de alta, las introdujera a mano.

Por ello, se decidió incluir un sensor más, este USB, para conectar al equipo del técnico, de manera que pusiera en pantalla el código de la tarjeta cuando la pasar por encima, simplificando enormemente el proceso.

Img. 45: Lector PhidgetRFID para el técnico de RRHH.

Para ello se tuvo que desarrollar un pequeño demonio para Windows, utilizando Java como lenguaje de desarrollo ya que es un lenguaje multiplataforma, e instalando la máquina virtual de Java (https://java.com/), el demonio era fácil de dejar como servicio que se carga al inicio, por lo que hubo de configurarse este ordenador.

Denominado `rfid-reader-robot` y utilizando el sensor Phidget relacionado en la memoria, en cuya página se pueden encontrar fácilmente las librerías para programar su funcionamiento en Windows (librerías dll) y usando Java. El código principal del robot se encuentra comprimido en el fichero `.jar`.

Su única función es imprimir por pantall donde esté el foco de texto en ese momento colocado el código de la tarjeta.

8.4. Siguientes Sprints

Como se ha comentado anteriormente, en el proyecto se ha aplicado una metodología de desarrollo tipo ágil, donde la planificación de las iteraciones se basa en SCRUM. Para el desarrollo de esta memoria, se aplicó una congelación de las características determinadas para la versión v4b (*Backlog Freeze*), que representa un *sprint* en concreto. Existen unas cuantas funcionalidades necesarias que se han planificado para sprints futuros.

Es importante diferenciarlas de las Líneas Futuras en cuanto a la temporalidad: éstas últimas están pensadas como “posibles” características futuras a medio o largo plazo, mientras que las características planificadas para las siguientes iteraciones son habilidades del sistema que “deben” estar por requerimiento del cliente y del propio proyecto una vez se ha ido probando los prototipos y se han ido descubriendo su existencia, o simplemente por prioridades tras establecerlas al diseñar el MVP.

Próxima funcionalidad	Prioridad	Resumen
Generación y Exportación informes en otros formatos	Media	Posibilidad de exportar el informe personalizado con un formateo adecuado, con sello de la empresa y porcentajes y comparativas en formatos, tales como PDF, CSV, XLSX, etc.
Impresión etiquetas tarjetas	Media	Facilita la creación de tarjetas a nuevo personal. El proceso es sencillo, con los datos del personal, se genera una plantilla a imprimir en PDF con marcas de impresión, usando papel de pegatinas para tarjeta tipo tarjeta de crédito como el siguiente:  Img. 46: Etiquetas marca APLI para tarjetas tipo Tarjeta de visita
Comunicación segura de la API vía SSL	Alta	Los datos transmitidos desde el sensor al servidor no se transmiten usando una comunicación segura, por lo que podría ser vulnerable a ataques Man in the Middle[74].
Gestión de almacenamiento y backups óptima de las capturas	Media	A medio/largo plazo, las capturas de la cámara en cada check puede llegar a ocupar mucho espacio, y debe diseñarse una estrategia de almacenamiento, recuperación y visualización óptimo.
Flash infrarrojo para mejorar la toma de imágenes	Baja	De cara a mejorar las condiciones lumínicas de la toma de las imágenes en los *checks*, se debe añadir un anillo de leds infrarrojos Ultrabright que se enciendan coordinadamente con la exposición de la

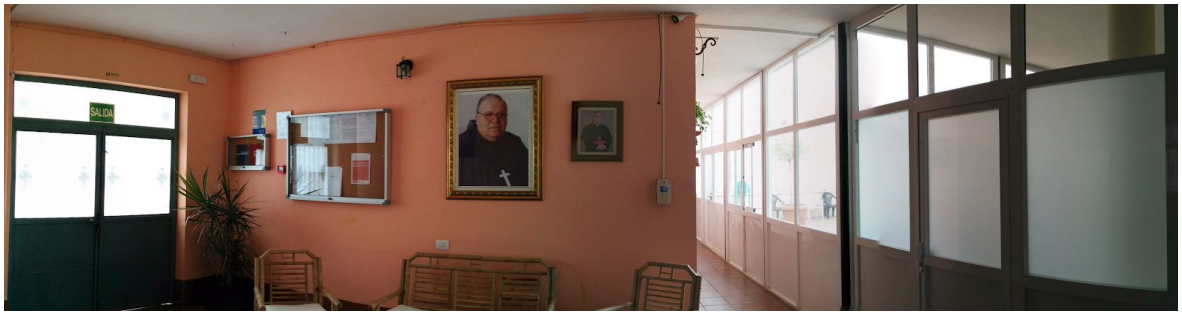
[74] https://es.wikipedia.org/wiki/Ataque_de_intermediario

		imagen.
Página de configuración del sensor	Media	En el servidor, una sección estaría disponible para configurar remotamente valores del propio sensor, así como monitorizar su estado.
Definir una estrategia de CONSTRAINTS y ON_DELETE, ON_UPDATE, etc. adecuada para mantener una coherencia de datos.	Alta	Durante el proceso de diseño de la base de datos, se ha seguido una aproximación conservadora, no permitiendo eliminar elementos, sino “desactivarlos” para evitar problemas de coherencia. Se debe hacer un estudio profundo de las diferentes estrategias y consecuencias, así como estrategias de triggers por acción.
Proporcionar etiquetas QR con la URL del sistema Check-In	Baja	Mediante una foto del código QR, el navegador puede redireccionar a la persona que lo escanee directamente al sistema Check-In si quieren consultar algo en vez de introducir la URL.
Configurar un “manifest.json”	Baja	Esto permitirá añadir que la web del servicio pueda ser añadida como una app al escritorio móvil, facilitando su consulta pulsando en el icono.

Tabla 11: Resumen de características a desarrollar en próximos sprints

9. Resultados

Como resultado final, el paquete entregado y configurado al cliente fue el siguiente.
1 x Sensor Check-IN
1 x Servidor RPI v2 con el servicio Check-IN
1 x Sensor USB en la oficina de RRHH, para consultas de tarjetas.
40x Tarjeta RFID 125Khz blancas.



Img. 47 : Detalle de la instalación final del sensor

Repasando los objetivos marcados en esta memoria, el proyecto ha conseguido crear un marco de desarrollo con un sensor, y un servicio que, si bien algo básico en esta versión inicial, no necesita de demasiado trabajo adicional para añadir las funcionalidades adicionales que se puedan requerir, siendo una base sólida para la creación tanto de una solución ad-hoc como de un producto. El sensor cumple con las exigencias de bajo coste, bajo impacto y no biométrico, y el software se puede usar para dar altas y bajas de trabajadores, gestión de turnos, consulta y generación manual de Checks, así como generar informes de checks.

Lo más complejo desarrollado ha sido todo el proceso de diseño hasta llegar a esta base estable.

10. Conclusiones

Como conclusión de este proyecto cabe destacar claramente una ventaja innegable: Un proceso manual ha sido trasvasado a digital, lo que abre un enorme mundo de posibilidades.

Sin embargo, los verdaderos factores de éxito aquí son:

- Ahorro de tiempo (se ha calculado un SpeedUp de x3,5)
- Ahorro de espacio (no se necesita archivar las firmas de manera física)
- Ahorro de papel (no se necesita imprimir las hojas de fichajes)
- Ahorro económico (no depende de licencias o “contratos esclavos”).
- Ahorro de personal dedicado, al automatizar un proceso manual y mecánico (no es necesario revisar las firmas y validarlas)
- Mejora de productividad, al aumentar el control sobre el personal y la exactitud de cara su cumplimiento horario
- Cumplimiento de normativas para certificaciones, que requieren de control de datos digital, documentado y exhaustivo

Además, al ser una solución *open hardware* y *open source*, el centro gana potencialidad de aumentar las prestaciones de este sistema con propuestas como las que se exponen en “Líneas Futuras”, así como la posible integración sencilla con otro software que pueda requerir esta información, al plantear una implementación utilizando estándares (SQL, JSON, etc.) y el uso de APIs. Esto último facilita claramente retomar este proyecto en un futuro por cualquier desarrollador con estos conocimientos, que, al ser de tecnologías ampliamente utilizadas, bien documentadas y abiertas, no es difícil de encontrar.

10.1. Comentarios personales

Ha sido un proyecto maravilloso, pero extremadamente ambicioso si no se enfoca desde un punto de vista generalista para desarrollar de una vez y vender luego a varios clientes bajo una empresa. Este era el enfoque cuando se inició este proyecto, pero aprendí sobre dos factores importantes durante este:

1) Se necesita un equipo. En solitario es prácticamente imposible abarcar la miríada de tecnologías y fases de un desarrollo así en un tiempo viable. Se empieza con un hilo simple, pero se complica cada vez más. Además, sin un proceso estructurado de desarrollo, con *boiler plates*, repositorios, protocolos, entornos de desarrollo, etc. pronto todo acaba siendo un desastre.

2) El hardware da muchos problemas de soporte y producción. Es decir, funciona muy bien como un proyecto personal, pero cuando hay clientes de por medio, debes tener un sistema de soporte dedicado a esto. Además, el proceso de producción cambia radicalmente el enfoque con el que

> desarrollas un proyecto que involucra hardware. Es evidente que para esto se debe tener un músculo financiero que cubra los gastos iniciales no solo del desarrollo y el I+D+I, sino también del proceso de producción, reparación, parcheado, actualización. En el papel es fácil de ver, pero cuando lo vives es cuando entiendes la verdadera dimensión.

Por tanto, la importancia de la metodología de desarrollo, de las fases, de los roles, de los entornos y de los test, han quedado suficientemente respaldados durante el desarrollo de este proyecto.

11. Líneas futuras

El proyecto en sí da juego a la hora de pensar en posibles mejoras e interacciones de cara a convertirlo en un producto completo, competitivo y hasta disruptivo a medio o largo plazo. He aquí algunas de ellas.

11.1. App móvil

En nuestros días se observa una tendencia muy evidente hacia el uso de dispositivos móviles, más a mano y más manejables y portátiles que ordenadores de mesa, o que incluso tablets, por lo que se propone preparar un app (aplicación nativa) para dispositivo móvil que permita trabajar de la misma manera que con la versión de escritorio.

Técnicamente, esta app podría ser de 3 tipos (nativa de alguno de los dos sistemas operativos imperantes, Android e IOS, o híbrida) o cualquier combinación de ellos:

- Native Android app
- Native IOS app
- Hybrid app

11.1.1. ¿Qué es una aplicación híbrida?

Aplicaciones desarrolladas en HTML, CSS y Javascript que corren sobre una capa de abstracción / framework (Apache Cordova, Ionic, React Native, Capacitor…) que proporciona acceso a los recursos del terminal de forma neutral al tipo de dispositivo. Por ello, mediante este paradigma, se desarrolla una única aplicación común que luego se prepara para cada plataforma (con pequeños ajustes funcionales si fuera necesario).

11.1.2. ¿Qué es una aplicación nativa?

Aplicaciones desarrolladas sobre el lenguaje de programación nativo (Java, Swift, Objetive-C,…) del dispositivo. Estas aplicaciones son 100% dependientes de la plataforma. Por ello, hay que desarrollar y mantener una aplicación completa para cada plataforma destino (iOS, Android u otros).

11.2. Formatos de RFID

La tecnología RFID ha ido evolucionando hacia estándares más potentes e interesantes que el sistema base de 125KHz que se usa en esta memoria, como por ejemplo el sistema MiFare[75], así como los estándares utilizados como métodos de pago como el NFC[76], presentes en las tarjetas de crédito, móviles y smartwatches de últimas generaciones, etc. Todas hacen uso de la misma tecnología de inducción eléctrica, donde existe un elemento activo y uno pasivo, incluso semipasivo, pero los sensores son ligeramente diferentes, por los que no son compatibles entre sí.

[75] https://es.wikipedia.org/wiki/Mifare
[76] https://es.wikipedia.org/wiki/Near_field_communication

Incluir un elemento multisensor en el propio dispositivo sensor, permitiría ampliar el rango de metodologías de identificación, desde llaveros y pegatinas a móviles o *smartwatches*.

11.3. Reconocimiento facial

Existe hoy por hoy investigaciones y desarrollo de librerías bastante interesantes para reconocimiento facial usando biometría, como por ejemplo OpenCV[77], librería en C++ que un estudiante pudo compilar para uso en plataformas embebidas como Raspberry PI, como la que utilizamos en este proyecto. Además, el sistema vendría equipado con otro adicional de aprendizaje (Machine Learning).

En el momento del desarrollo de esta memoria se asegura una precisión de reconocimiento en buenas condiciones de toma de la imagen de un 85%. Se propone un protocolo adicional como el siguiente:

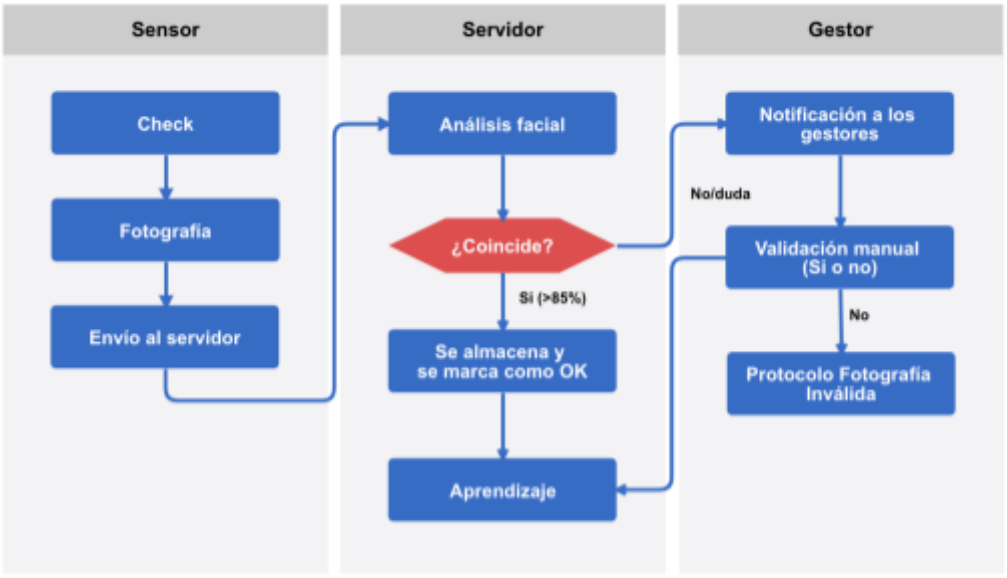
Esto implicaría:

- A corto/medio plazo una reducción de fraudes, al tener un sistema que revisa las fotografías tomadas continuamente (Mayor robustez).
- Aligera la carga de las revisiones manuales de las fotografías, reduciéndose a las que el sistema no reconoce (Mayor sencillez).
- Un ligero aumento de ruido en notificaciones mientras el sistema aprende.

Es en previsión de esta línea por la que se decidió utilizar una cámara NOIR, es decir, sin filtro infrarrojo, ya que permite mejorar la toma de una imagen en un rango más amplio de condiciones lumínicas, es decir, sumando el rango de luz visible el infrarrojo. Esta característica, permite mayor rendimiento en los algoritmos de detección facial[78], a la par que mejor rendimiento en entornos de baja luminosidad (atardeceres o de noche). Además, permite la posibilidad de uso de flash de leds infrarrojos que no afectan a la vista ya que no se encuentran en el espectro de luz sensible al ojo humano.

[77] https://en.wikipedia.org/wiki/OpenCV

[78] Joseph Wilder, P. Jonathon Phillips, Cunhong Jiang, Stephen Wiener, “Comparison of Visible and Infra-Red Imagery for Face Recognition”, National Institute of Standards and Technology of United States of America, descargado en IEEE Xplore el 21 de Julio de 2010. https://www.nist.gov/sites/default/files/documents/2016/12/15/comparison_infrared.pdf



Img. 48. Diagrama de flujo de decisión propuesto

11.4. Machine Learning: Predicción de retrasos

Dentro de la disciplina de inteligencia artificial, el aprendizaje automático[79], o Machine Learning, es el conjunto de estrategias y tecnologías que permiten que el propio sistema aprenda y ajuste sus valores en comparación a resultados supervisados, con el objetivo de prever un comportamiento futuro o ser capaz de identificar elementos con mayor precisión.

Para el caso del presente proyecto, se podría aplicar en dos partes:

- Durante el proceso de aprendizaje de reconocimiento facial, permitiendo que el sistema sea dinámico en su capacidad de reconocimiento de los trabajadores cuando fichan, adaptando sus medidas biométricas con el fin de reducir el número de falsos positivos y generar resultados, en definitiva, más precisos.
- Cómo estudio experimental sobre costumbres humanas, al poder predecir retrasos, en porcentaje de fiabilidad, sobre las horas establecidas de los turnos personales. Si bien, para este caso se requiere:
    - Aceptar que el Ser Humano es un “animal de costumbres”, tomando como axioma el dicho popular que hace referencia a la tendencia natural que nuestra especie tiene hacia la generación y seguimiento de rutinas.
    - Un número suficientemente alto de muestras previas como para permitir que un sistema sea capaz de emitir conclusiones en escala temporal.

[79] https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

11.5. Mejorando la robustez: medidas adicionales

Uno de los requisitos del cliente, es que el sistema posea una alta tolerancia a sabotajes externos que puedan afectar a la coherencia de los registros del sistema. A los ya diseñados, se aportan las siguientes medidas adicionales:

11.5.1. Batería

En previsión de un corte en el suministro eléctrico, el sistema debería incorporar una batería de alimentación. Puesto que el sistema se basa en un sistema electrónico embebido, su consumo no es especialmente alto. Mención aparte requiere la pantalla, la cual representa alrededor de un 30-35% del consumo total del sistema sensor. En concreto, esta es una media a *grosso modo* de consumo de las diferentes partes:

Dispositivo	Descripción	Consumo medio
LCD touchscreen display + RTC device	Pantalla LCD táctil de 5,3” + dispositivo de pila para mantenimiento de reloj interno	500mA
Raspberry PI 2B+	Placa PC embebida	500mA
Wifi dongle	Dispositivo para conectividad Wifi	250mA
RaspiCam NOIR	Cámara	250mA
EM4100 UART 125Khz	Lector tarjetas RFID 125Khz	50mA
Total		**1550mA**

Tabla 12: Consumos aproximados de energía de los diferentes dispositivos del sensor

Estas medidas pueden optimizarse aún más implementando ajustes específicos.

Con esta configuración, una batería de 4000mA podría proporcionar una alimentación ininterrumpida del sistema por encima de una hora en condiciones de carga de trabajo alta (existen varios experimentos que aseveran dar más de 5 horas para una configuración similar debidamente estudiados y justificados en páginas especializadas), tiempo suficiente para el restablecimiento de la alimentación eléctrica.

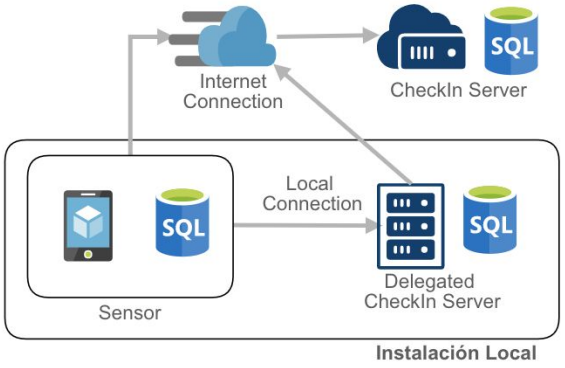
11.5.2. Control de movimiento

En previsión de robo o manipulación indebida, un sensor de movimiento puede resultar útil, disparando un evento de toma de fotografía a través de la cámara y lanzando una alerta al responsable del sistema con dicha imagen. El protocolo puede incluir un bucle con estas acciones si el movimiento se prolonga durante varios segundos. Esto puede ayudar también a hacer seguimiento de casos en los que un sensor aparece movido y por tanto no es capaz de tomar correctamente las imágenes de los trabajadores.

11.5.3. Sistema híbrido Local-Nube

Con el objetivo de facilitar el acceso desde fuera de las instalaciones, el respaldo del sistema y la robustez de la información, se propone transformar el sistema en uno híbrido, donde la arquitectura cliente-servidor, divide la parte servidor en una jerarquía de dispositivos servidores, con redundancia. Así, el esquema propuesto pasaría a ser:

- Instalación Local: Servidor delegado (autónomo) y sensores
- Instalación en la Nube: Servidor central y copias de seguridad.



Img. 49. Diagrama de un modelo híbrido local-nube

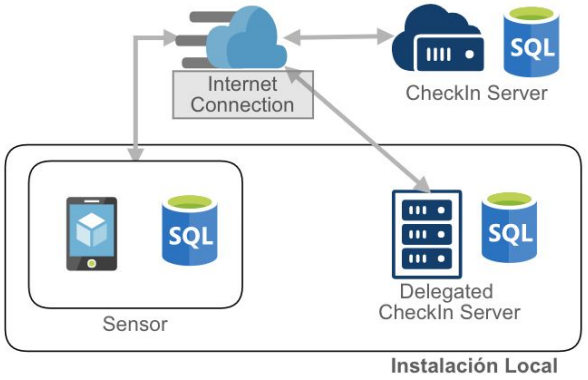
Esto permitiría una mayor tolerancia a fallos de conectividad así como una capacidad de recuperación de la información ante desastre. Nótese que no es una sincronización “servidor local - servidor remoto” solamente, sino que adiciona una doble cola de mensajes al sensor, una local y una remota, el cual es, lógicamente, la fuente mayoritaria de información del sistema. La configuración de tarjetas, checks locales, personal, turnos y demás se lleva a cabo desde el servidor local, pero la frecuencia de generación de información más alta la posee el propio sensor.

Por contra, encarecería el sistema un poco más al tener que mantener una suscripción a un servicio tipo SaaS[80], aunque la cantidad puede ser repercutida dentro de la tasa de mantenimiento siempre y cuando el SaaS consiga una masa mínima de usuarios con este sistema instalado. Esto a su vez, permitiría una diversificación del negocio.

11.5.4. Conectividad GSM

En la línea de lo anteriormente expuesto, y de manera complementaria, se plantea la posibilidad de incorporar un módulo de conectividad GSM[81] para mejorar las posibilidades de conectividad en entornos donde una conectividad Wifi o Ethernet no sea viable o inestable. En tal caso, el sistema podría migrar a uno totalmente SaaS, o una nube híbrida con el siguiente esquema:

[80] https://es.wikipedia.org/wiki/Software_as_a_service
[81] https://es.wikipedia.org/wiki/Sistema_global_para_las_comunicaciones_m%C3%B3viles



Img. 50: Diagrama de un modelo híbrido de sincronización a través de la nube

11.6. Cerraduras inteligentes y Backtracking

Al estar basado en un proyecto *open hardware* y *open software*, es fácil imaginar una extensión de estos sensores y del sistema servidor para implementar cerraduras inteligentes que se abran con las mismas tarjetas, habilitando la posibilidad de almacenar información de recorrido de los empleados, así como la generación de permisos y rutas permitidas. Esto a su vez, permitiría la característica de *backtracking* en el caso de que se necesite estudiar alguna incidencia en zonas sensibles (por ejemplo, la enfermería), ya sea por ruta seguida por una persona, o por controlar la cantidad de accesos a un espacio y las personas involucradas.

11.7. Arcos RFID

Es posible añadir arcos RFID parecidos a los utilizados en las tiendas de ropa para controlar los robos, al pitar cuando un sensor detecta una etiqueta RFID no retirada por los responsables de la tienda cuando sale por la puerta.

En este caso, pueden incorporarse a otros accesos del centro donde controlar que se ha pasado por ahí, como por ejemplo en el caso de que quieran salir a fumar un cigarrillo al patio de atrás. Estos arcos simplifican el proceso de check, al no tener que hacer nada más que pasar a través de éstos, dado que la identificación del empleado debe estar colgada y visible en su cuerpo, típicamente como un colgante o una pinza en el pecho o en la cintura.

Img. 51: Arcos RFID genéricos

También pueden utilizarse para controlar quién y cuándo pasar por algún punto de interés.

11.8. Certificación ISO/AENOR

A la hora de poder utilizar la información almacenada por este sistema en un proceso penal o en algún otro proceso es probable que se requiera que el éste haya pasado por una auditoría externa, consiguiendo una certificación que valide los resultados y la fiabilidad del proceso y coherencia de sus datos. Evidentemente, de cara a la utilización de este sistema como producto, las certificaciones aportan un innegable valor de confianza de cara a futuros clientes.

11.8.1. ISO 27001:2015

ISO/IEC 27001 es un estándar para la seguridad de la información que especifica los requisitos necesarios para establecer, implantar, mantener y mejorar un sistema de gestión de la seguridad de la información (SGSI).

Esta norma se encuentra dividida en dos partes; la primera se compone de 10 puntos entre los cuales se encuentran: Objeto y campo de aplicación, Referencias normativas, Término y definiciones, Contexto de la organización, Liderazgo, Planificación, Soporte, Operación, Evaluación de desempeño y Mejora Contínua.

Beneficios de la ISO 27001:

- Demuestra la garantía independiente de los controles internos y cumple los requisitos de gestión corporativa y de continuidad de la actividad comercial
- Demuestra independientemente que se respetan las leyes y normativas que sean de aplicación
- Demuestra el compromiso de la cúpula directiva de su organización con la seguridad de la información

11.8.2. ISO 25000. Calidad del software

ISO/IEC 25000, conocida como SQuaRE (System and Software Quality Requirements and Evaluation), es una familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software.

La familia ISO/IEC 25000 es el resultado de la evolución de otras normas anteriores, especialmente de las normas ISO/IEC 9126, que describe las particularidades de un modelo de calidad del producto software, e ISO/IEC 14598, que abordaba el proceso de evaluación de productos software. Esta familia de normas ISO/IEC 25000 se encuentra compuesta por cinco divisiones.

Img. 52: División ISO 2500n

En especial, dentro de la división 2501n, las certificaciones 25010, *System and software quality models*: describe el modelo de calidad

para el producto software y para la calidad en uso. Esta Norma presenta las características y subcaracterísticas de calidad frente a las cuales evaluar el producto software.

El modelo de calidad representa la piedra angular en torno a la cual se establece el sistema para la evaluación de la calidad del producto. En este modelo se determinan las características de calidad que se van a tener en cuenta a la hora de evaluar las propiedades de un producto software determinado.

La calidad del producto software se puede interpretar como el grado en que dicho producto satisface los requisitos de sus usuarios aportando de esta manera un valor. Son precisamente estos requisitos (funcionalidad, rendimiento, seguridad, mantenibilidad, etc.) los que se encuentran representados en el modelo de calidad, el cual categoriza la calidad del producto en características y subcaracterísticas.

El modelo de calidad del producto definido por la ISO/IEC 25010 se encuentra compuesto por las ocho características de calidad que se muestran en la siguiente figura:

Img. 53: Árbol de características relacionadas con la Calidad del Software según la ISO

11.8.3. AENOR ISO 9001

La Norma ISO 9001:2015 elaborada por la Organización Internacional para la Normalización (ISO), determina los requisitos para un Sistema de Gestión de la Calidad. La organización demuestra su capacidad para proporcionar de forma coherente productos o servicios que satisfacen los requisitos del cliente y los reglamentarios aplicables.

Esta norma internacional promueve la adopción de un enfoque basado en procesos cuando se desarrolla, implanta y mejora la eficacia de un sistema de gestión de la calidad, basado a su vez en el ciclo de mejora continua PDCA (Planificar, Hacer, Comprobar, Actuar).

Beneficios de AENOR ISO 9001:

- Mejorar la imagen de los productos y/o servicios ofrecidos
- Favorecer su desarrollo y afianzar su posición
- Ganar cuota de mercado y acceder a mercados exteriores gracias a la confianza que genera entre los clientes y consumidores
- Aumento de la satisfacción de los clientes
- Cimentar las bases de la gestión de la calidad y estimular a la empresa para entrar en un proceso de mejora continua
- Aumentar la motivación y participación del personal, así como mejorar la gestión de los recursos

12. Bibliografía

Una breve reseña a aquellos textos y recursos consultados sin los que esta memoria no puede ser desarrollada:

- Documentación:
  - Documentación Python: https://www.python.org/doc/
  - Documentación Django: https://docs.djangoproject.com
  - Documentación de la estructura base para proyectos (o *boilerplate*) desarrollados en Django "Cookiecutter": https://cookiecutter-django.readthedocs.io
  - Documentación Docker: https://docs.docker.com/
  - Documentación Raspberry Pi: https://www.raspberrypi.org/documentation/

- Referencias bibliográficas:
  - "Two scoops of Django 1.11: Best Practices for the Django Web Framework": https://www.twoscoopspress.com/products/two-scoops-of-django-1-11
  - Agile Manifesto: http://agilemanifesto.org/
  - SCRUM guide: https://www.scrum.org/resources/scrum-guide

- Referencias de configuración:
  - Configuración del módulo RTC en Raspberry Pi: http://ingeniapp.com/configura-un-rtc-reloj-tiempo-real-en-tu-raspberry-pi/
  - Configuración de la pantalla de 2.8" desarrollada por Texy: https://www.raspberrypi.org/forums/viewtopic.php?f=93&t=53144
  - Configuración de lector RFID serial para Raspberry Pi: https://behindthesciences.com/electronics/raspberry-pi-rfid-tag-reader/

13. Anexo A - Manual de Usuario de Check-IN

A continuación se refleja un manual de usuario sencillo.

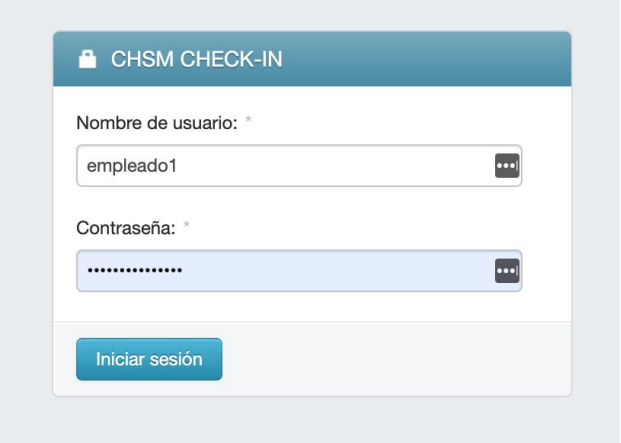
13.1. Consideraciones previas

- Ningún usuario anónimo podrá interactuar con el sistema, ni podrá darse de alta por sí mismo. Se le mostrará una pantalla de *login*.
- El alta de un usuario viene realizado por los usuarios administradores
- Lo usuarios registrados no administradores, no tendrán acceso a la parte de administración, solo a la relación de sus checks y a cambios simples sobre alguno de los datos de ficha.
- Sólo los usuarios administradores tienen acceso a la zona de administración

13.2. Usuario registrado: *Employee* (empleado)

13.2.1. Login

Una vez ingrese en el sistema, utilizando la URL que le proporcione el administrador de su centro de trabajo, le solicitará mediante formulario que se autentique mediante un usuarios (el correo que haya proporcionado al administrador para crear su ficha) y la contraseña que usted especificó cuando se registró.



Img. 54: Captura del formulario de login del sistema

13.2.2. Recuperación de contraseña

Puede solicitar un proceso de recuperación de contraseña mediante correo.

13.2.3. Mis checks

Una vez autenticado, verá una tabla con sus últimos 60 días de *checks* que podrá imprimir si lo desea.

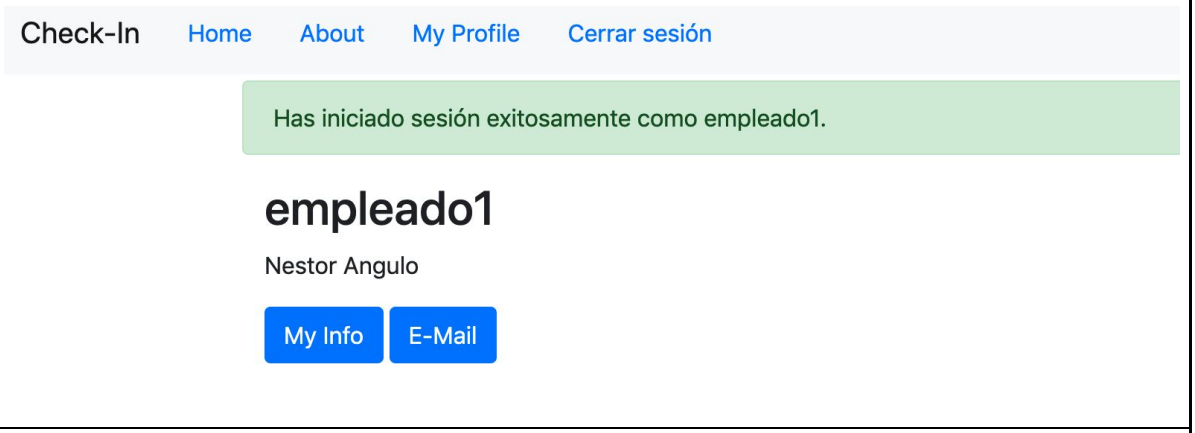
ID	¿Quién?	Hora	Tipo de Check
36	Tarjeta de N. Angulo de Ugarte	3 de Junio de 2015 a las 06:59	Check-Out
35	Tarjeta de N. Angulo de Ugarte	3 de Junio de 2015 a las 06:25	Check-In
34	Tarjeta de N. Angulo de Ugarte	3 de Junio de 2015 a las 06:22	Break
1	Tarjeta de N. Angulo de Ugarte	3 de Junio de 2015 a las 04:04	Check-In

Img. 55: Captura de un ejemplo de tabla de checks.

13.2.4. Mi perfil

Una vez autenticado, puede modificar su perfil añadiendo una cuenta de correo alternativa o cambiando su contraseña.

Para ello, clickee en la opción del menú “My profile” o “Mi perfil”.



Img. 56: Captura una vez iniciada la sesión

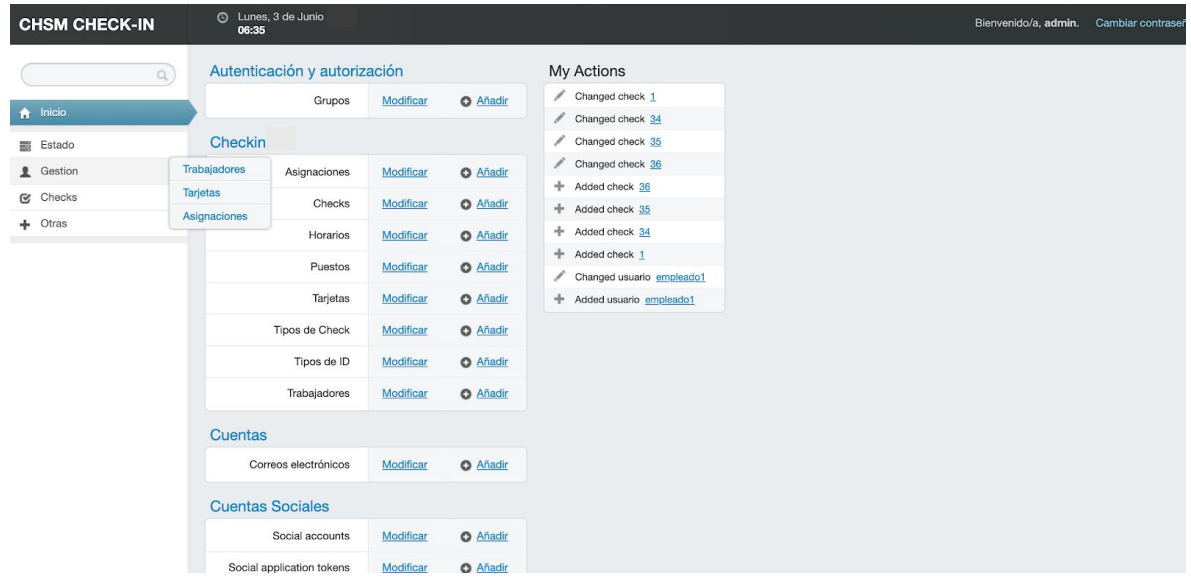
13.2.5. Logout

Si desea cerrar su cuenta, puede hacerlo haciendo en la opción del menú “Cerrar sesión”.

***Nota**: Su sesión se cerrará automáticamente cada vez que cierre la página o pestaña*

13.3. Usuario registrado: *Admin* (administrador)

Si su usuario tiene permisos de administrador, puede acceder a la URL de administración, típicamente `/admin`, donde se muestra el entorno administrativo:



Img. 57: Captura una vez iniciada la sesión

El administrador puede desde aquí ver rápidamente las últimas acciones que se han llevado a cabo con el usuario, ver un listado rápido de las acciones que se pueden realizar, ir a algunas de las opciones del lado izquierdo o buscar un usuario concreto (en el menú a la izquierda arriba, que es equivalente a entrar en la sección *Trabajadores* y usar el buscador que se encuentra en la parte superior de dicha sección, sin filtros.

Se observa que desde el propio listado se pueden añadir elementos de una de esas entidades, o modificar (que significa ir al listado).

Cabe considerar una serie de puntos en general en los formularios de añadir/modificar:

- Los campos *datetime* son renderizados utilizando un campo fecha y un campo hora, ambos con atajos para dar un valor rápido como *hoy* y *ahora*.
- Los campos que son claves relacionadas a otras tablas, se renderizan con un menú desplegable mostrando los datos existentes, o con la posibilidad de añadir un nuevo elemento, borrarlo o modificarlo, lo que permite la creación *inline* de otros objetos necesarios. Cuando se desea modificar o crear, se abre una ventana nueva donde permite trabajar con un formulario de alta o modificación de esa entidad.
- Todos los objetos tienen una opción “Histórico” desde donde puede observarse los cambios que ha sufrido un objeto y por quién. Solo aparece cuando el objeto se modifica.
- En un formulario de alta/modificación, siempre hay 3 opciones de grabado:
    - Grabar simplemente (vuelve al listado)

- Grabar y añadir otro
- Grabar y continuar editando

13.3.1. Trabajadores

Para esto debemos hacer click en la sección *checkin* > *Trabajadores*, o en la de la izquierda en el menú *Gestión* > *Trabajadores*.

Img. 58: Captura una vez iniciada la sesión

En esta sección se puede buscar una palabra, filtrando si se quiere por puesto, si se quieren los activos o no activos o todos o por turno de trabajo.

En listado encontraremos el resultado de esta búsqueda con sus filtros aplicados, donde se muestran algunos datos importantes, así como el último *check* que realizó. El campo *activo* nos permite editarlo directamente ahí para el caso de que se quiera desactivar a un trabajador directamente, pero debemos pulsar en el botón *grabar* que se encuentra al finalizar esta tabla para que sea efectivo. Además, el listado está paginado en caso de que exista mucho personal.

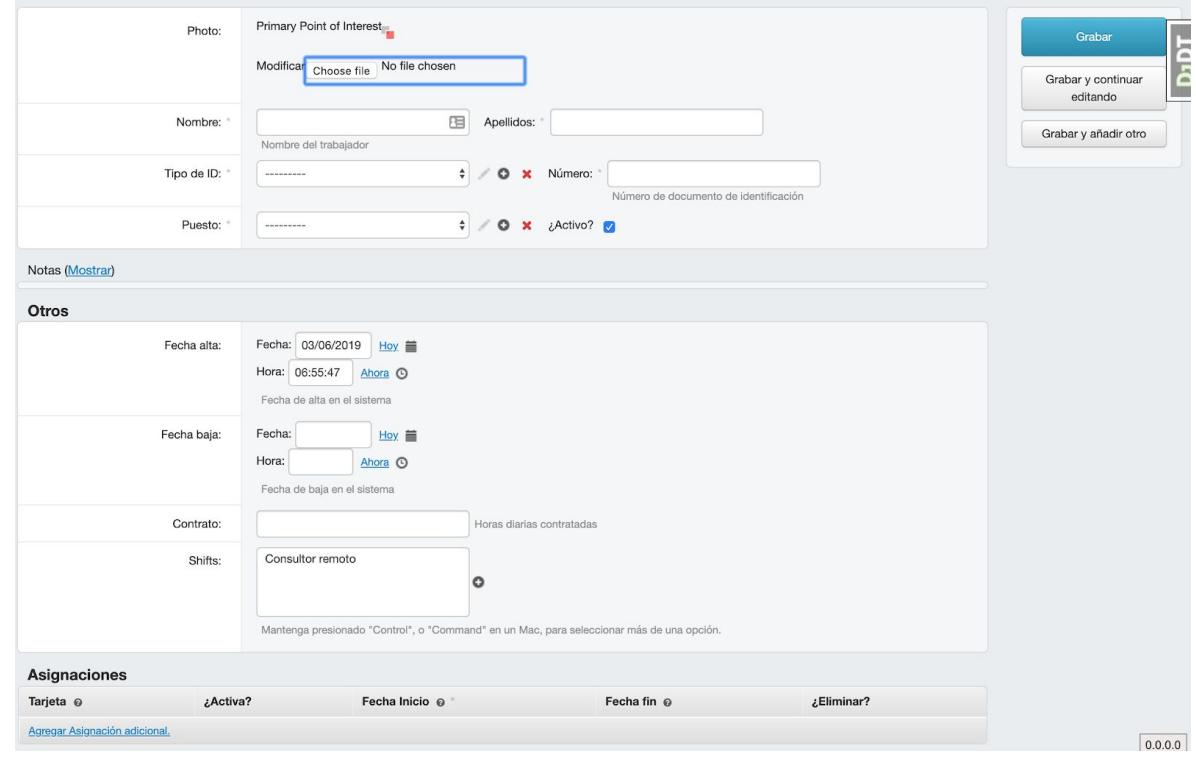
El personal no debe ser eliminado, solo desactivado, añadido o modificado. Si se quiere eliminar debe contactar con su super administrador, o bien seleccionarlo de la lista, y seleccionar la opción “eliminar elementos seleccionados” en el menú desplegable que se encuentra al final de la tabla:

Img. 59: Detalle de la eliminación de un trabajador

Seguidamente pulse en “Ir” y le saldrá un mensaje de advertencia, ya que la eliminación del trabajador puede afectar a otros objetos.

13.3.2. Trabajadores: Dar de alta o modificar.

Por último, tenemos un botón para añadir trabajadores, que nos lleva a un formulario parecido al de la siguiente captura:



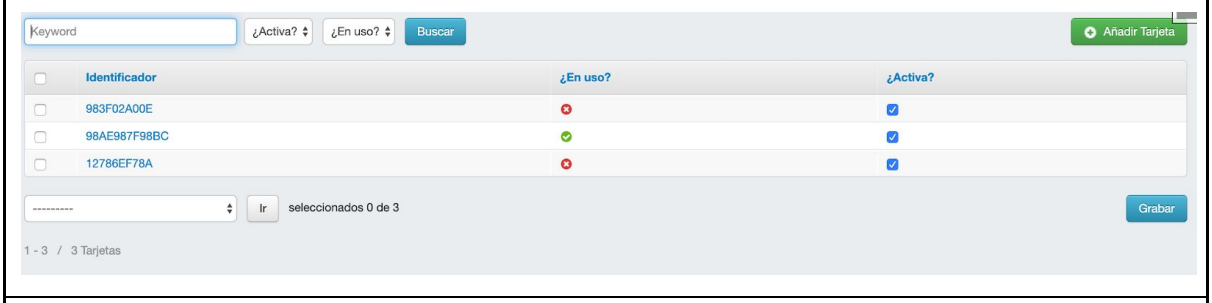
Img. 60: Detalle del formulario de alta de un trabajador

Consideraciones específicas de este formulario:

- Existe un campo notas, donde el administrador puede añadir anotaciones específica de los trabajadores, pero dicho campo está oculto por defecto. Para mostrarlo, debe hacer click en el vínculo “mostrar” que sigue al nombre del campo.
- Un trabajador puede trabajar en diferentes turnos, por lo que en el campo correspondiente puede elegir todos los turnos que estime necesarios, incluso añadir más.
- Al final de la página puede asignar rápidamente una tarjeta a este usuario, eligiendo de las que se muestran o añadiendo una más.

13.3.3. Tarjetas

La sección de Tarjetas, es parecida a la de Personal en cuanto a consideraciones y al proceso de alta/modificación o eliminación:



Img. 61: Detalle de la sección Tarjetas

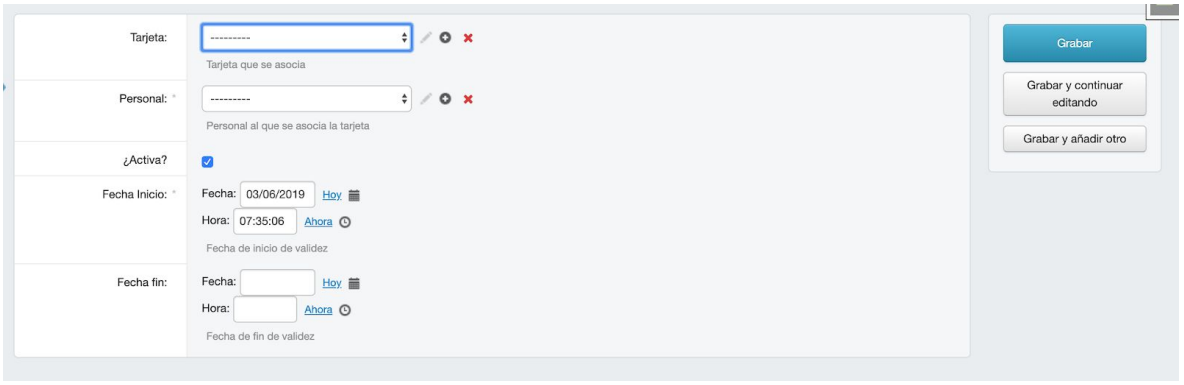
Img. 62: Detalle del formulario de alta/modificación de Tarjetas

13.3.4. Asignaciones Trabajador - Tarjeta

Para que un trabajador pueda hacer un *check* debe tener una tarjeta asignada. Esa asignación se puede realizar durante el proceso de alta o modificación de un trabajador, en la parte inferior, o manualmente en la tabla “Asignaciones”:

Img. 63: Detalle del listado de asignaciones de tarjetas

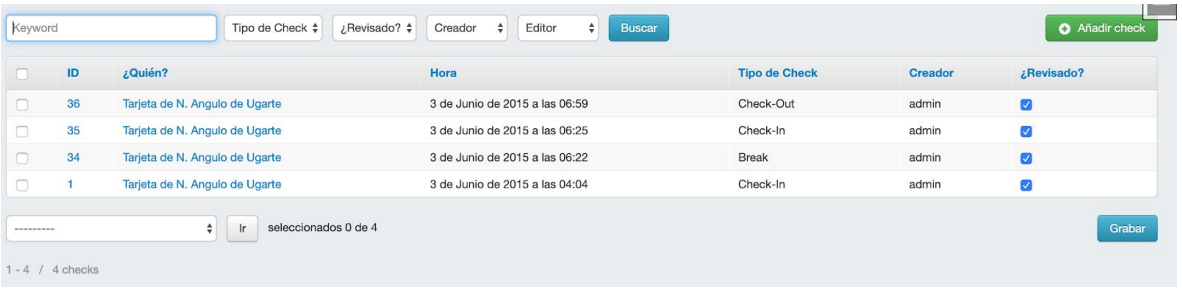
Durante el proceso normal de alta podemos especificar si la relación Trabajador-tarjeta está activa o si tiene fecha de caducidad.



Img. 64: Detalle del listado de asignaciones de tarjetas

13.3.5. Checks

Una vez tenemos un trabajador, una tarjeta y una relación, podemos hacer un *check*, introduciendo este manualmente en el sistema o modificando uno activo.

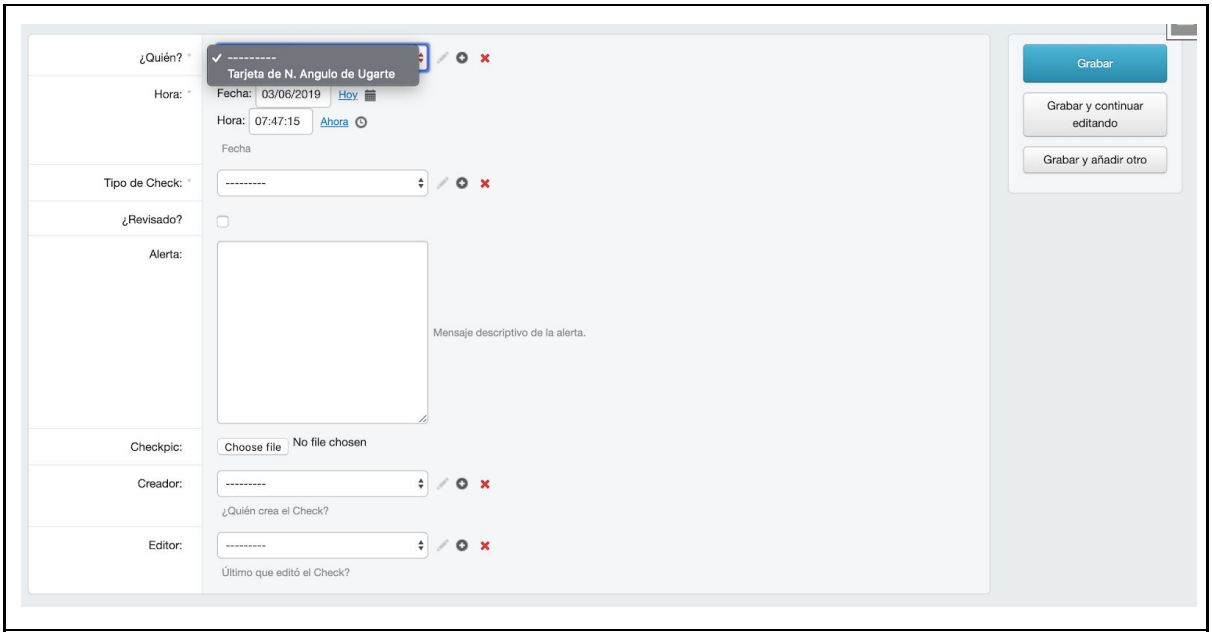


Img. 65: Detalle del listado de checks, introducidos en este caso por el admin y revisados por este

En el listado se puede filtrar por palabras clave (como nombre/apellidos del trabajador, DNI, número de tarjeta, etc.) o por tipo de checks, o solo mostrar los que están o no revisados, o los que creó o editó un usuario en particular.

En la imagen de ejemplo observamos un listado de checks ya revisados y generados manualmente.

Este es el formulario de añadir/modificar un *check*:

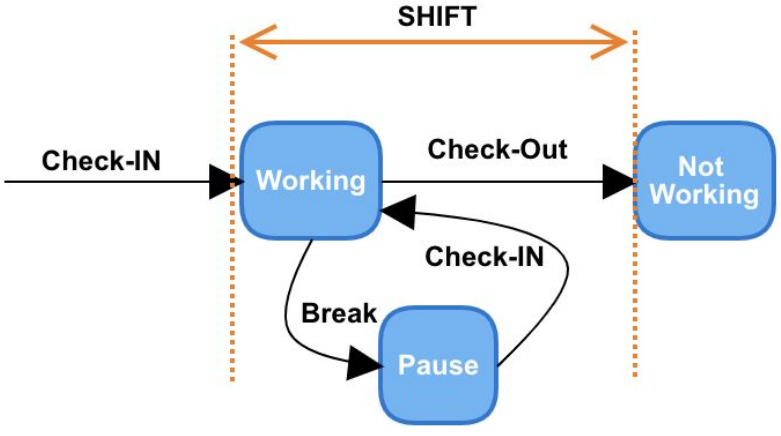


Img. 66: Detalle del alta de un *check*

Consideraciones de este formulario en concreto:

- El tipo de *check* puede variar según las necesidades de cada empresa, pero inicialmente son de tres tipos:
  - Check-In (código IN)
  - Check-Out (código OUT)
  - Break (Código BREAK)

Responde al esquema del siguiente diagrama, donde las horas contabilizadas al trabajador, serán las que se calculen mientras el trabajador esté en el estado “working” o “trabajando”.



Img. 67: Diagrama de estados de un trabajador en el Check-In

- La revisión del *check* corresponde al caso en el que este haya causado una notificación, por lo que saldrá un texto en el campo “Alerta”. Si es así, el

check aparecerá como no revisado hasta que el administrador lo marque manualmente y grabe de nuevo el check.

- El campo creador, relaciona quién creó este check. Si fué automático por el sensor, aparecerá el usuario del sensor correspondiente (puede haber más de uno)
- El campo editor, se fijará al último usuario que editó el *check* (puede haber más de un admin).

14. Anexo B - Manual de Usuario del sensor Check-IN

El sensor Check-IN es un elemento que funciona en conjunto con el servidor Check-in y debe estar configurado por el administrador del sistema. Aquí se pone a su disposición un manual de usuario simple.

14.1. El dispositivo

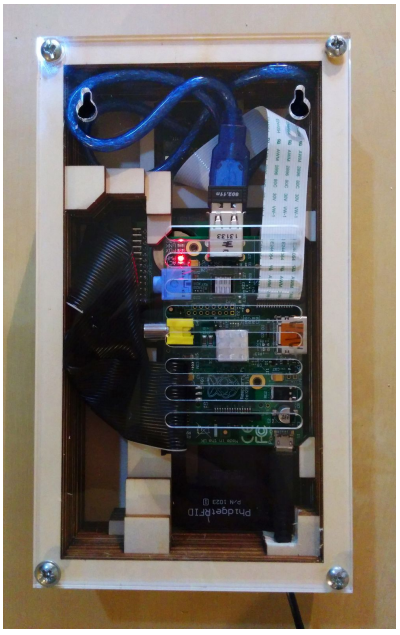
El sensor tiene una forma rectangular y pesa unos 400gr, formado de dos lados de metacrilato transparente que conforman la parte delantera y la trasera.

Img. 68: Detalle de la parte delantera del sensor

Parte delantera

Se observa:

- Orificio de la cámara en la parte superior.
- Pantalla en la parte central
- Zona de paso de la tarjeta marcado con un patrón discontinuo en la parte inferior.
- Cable de alimentación sobresaliendo por debajo.



Img. 69: Detalle de la parte trasera del sensor

Parte trasera

Se observa:

- Orificios de colgado en la parte superior
- Ranuras de ventilación enn la parte media
- Botón de reinicio a la mitad del sensor
- Electrónica interna visible
- Cable de alimentación sobresaliendo por debajo

14.2. Proceso de colgado

1. Seleccionar una zona con iluminación adecuada, sin luz directa del sol o elementos reflectantes en el rango de visión de la cámara. Debe tener una toma de corriente en el rango de 1.50m de distancia.
2. Medir 1.55m de altura y marcar la línea coincidente con la parte superior de la caja que debe estar paralelo a la línea del suelo. La altura de la cámara deberá quedar alrededor de un 1.50m - 1.60m de altura.
3. Colgar usando las perforaciones de la parte superior trasera con dos tornillos a la pared.

14.3. Encendido

1. El dispositivo se encenderá automáticamente una vez se conecte.
2. Espere unos 30sg mientras el dispositivo hace sus comprobaciones
3. El dispositivo estará listo cuando muestre la hora y la fecha en la pantalla.

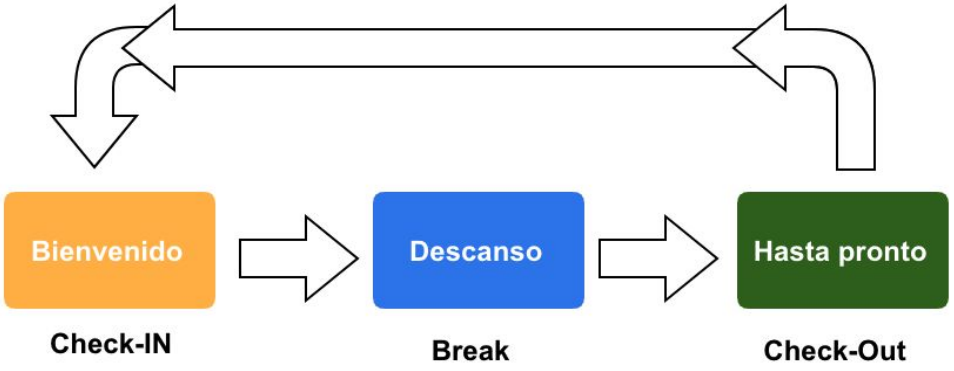
Img. 70: Detalle de la parte delantera del sensor

14.4. Reinicio

Si ocurriera algún error o el sensor tenga un comportamiento errático puede reiniciar el dispositivo pulsando el botón que encontrará a la mitad de la parte trasera del sensor. Es accesible por tacto sin introduce la mano por detrás del sensor, pero también puede descolgarlo para llegar a él conn mayor facilidad.

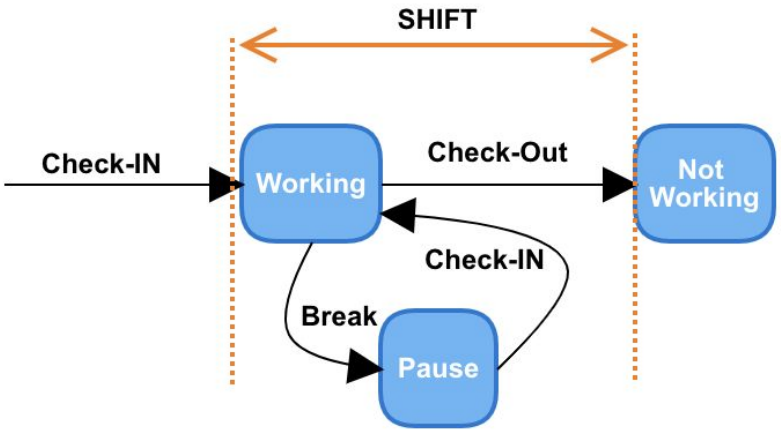
14.5. Proceso de check-In

1. Colóquese delante del sensor a la distancia de éste que marque su brazo totalmente estirado
2. Mire a la pantalla
3. Pase su tarjeta de identificación por la zona habilitada al respecto.
4. Aparecerá un mensaje en pantalla que podrá ser:
    a. Bienvenido: Significa que usted está entrando en el centro
    b. Descanso: Significa que usted va a ausentarse por un periodo corto
    c. Hasta luego: Significa que usted abandona el centro
5. Este mensaje estará presente en pantalla durante unos 5sg. Si es el mensaje que usted desea que se almacene, no haga nada durante esos 5sg.
6. Si el mensaje no es el que usted desea, vuelva a pasar la tarjeta durante estos 5sg, y el mensaje cambiará en un constante carrusel cada vez que usted pase la tarjeta, hasta que el mensaje en pantalla es el que usted desea.



Img. 72: Detalle del proceso de carrusel de mensajes

***Nota**: Tenga en cuenta que después de un "Descanso", viene un "Bienvenido", como se observa en el gráfico, y que sólo se contabilizarán las horas mientras permanezca en el estado "working" o "trabajando".*



Img. 73: Diagrama de estados de un trabajador en el Check-In

15. Anexo C - Presupuesto del sensor Check-IN

Reflejo aquí un resumen general de los precios en bruto de los componentes del sensor Check-IN y el resto de materiales hardware adquiridos para el proyecto.

Sensor Check-IN:

Dispositivo	Descripción	Cant.	Precio	Coste
Display	Pantalla TFT táctil de 2,8" + RTC	1	53€	53€
Raspberry PI v1 model B	Placa PC embebida	1	49€	49€
Sandisk MicroSD 8GB cat10	Tarjeta microSD de 8GB	1	14€	14€
MicroUSB AC adaptador 5V 2.1A	Alimentador de corriente microUSB	1	16€	16€
Wifi dongle	Dispositivo para conectividad Wifi	1	9€	9€
RaspiCam NOIR	Cámara + cable	1	23€	23€
EM4100 UART 125Khz	Lector tarjetas RFID 125Khz + cables	1	10€	10€
Carcasa a medida - Madera y metacrilato	12 capas de madera de balsa y 2 capas de metacrilato, cortadas por láser	1	60€	60€
Total				**~ 235€**

Tabla 13: Costes aproximados de los componentes del sensor

Lectores de tarjetas y tarjetas inteligentes:

Dispositivo	Descripción	Cant.	Precio	Coste
Phidgets RFID USB sensors	Lectores USB de tarjeta RFID + cable USB Type B	2	50€	100€
Tarjetas RFID 125KHz	Tarjetas blancas inteligentes	40	0,75€	40€
Total				**~ 140€**

Tabla 14: Costes aproximados de los sensor externo y las tarjetas

Por tanto, el coste del sensor más los lectores y las tarjetas, nos arroja un presupuesto de material de algo menos de 400€ en total (alrededor de 375€).