**Diseño integral de la arquitectura técnica: especicación de los elementos del backend, tecnologías y relaciones entre ellos**

**Cliente.**

En nuestro sistema, el cliente estará compuesto por varios agentes; la aplicación móvil para el usuario, la app web para empleados y administrador, las pulseras con tecnología NFC y el lector de las mismas pulseras.

Nuestro cliente será creado con las tecnologías HTML, CSS, Angular2.js, el sistema será multiplataforma gracias a que usaremos el framework Ionic2 para ello.

Conectaremos con el servidor gracias al protocolo HTTP mediante peticiones POST, GET, PUT… Trataremos de utilizar en el mayor número de ocasiones, verbos idempotentes (PUT, GET, HEAD, DELETE...), que permiten recuperar cualquier petición que no haya llegado al servidor, simplemente repitiéndola.

**Servidor.**

Este será un servidor Apache y estará desarrollado con PHP, utilizará como hemos dichos anteriormente el protocolo HTTP y será de código abierto. Estará diseñado de modo que cumpla los principios REST, estilo arquitectónico de por excelencia de los servicios web.

**Base de datos.**

El sistema de gestión de bases de datos que utilizaremos será PostgreSQL este ofrece fiabilidad, integridad de datos y características integradas enfocadas al desarrollador. A su vez, permite unir consultas a tablas relativamente grandes de manera eficiente. Emplearemos para el diseño y gestión del sistema una única base de datos relacional o SQL que mostraremos más adelante en el modelo de datos.

**Relaciones entre las distintas entidades**

En primer lugar, para la interpretación de las IP’s de los diferentes recursos a los que se quiere acceder utilizaremos el protocolo DNS.

Para el servicio de mailing con el usuario se implementará un sistema basado en gmail, crearemos un correo corporativo para evitar ser identificados como spam, y crearemos un proceso para el envío automático de correos.

En nuestro sistema se realizarán copias de seguridad periódicas, según lo que quiera cada uno de nuestros clientes, nosotros en principio realizaremos una copia total cada dos semanas y una parcial todos lo días, para esto utilizaremos Bacula, un framework gratuito para la elaboración de copias de seguridad.

**Especicación de elementos y tecnologías de seguridad y autenticación**

**Seguridad**

* No se puede acceder a todos los datos del sistema a no ser que se haga desde la cuenta de Administrador.
* Cifrado del saldo disponible en la cuenta de un usuario.
* El acceso a los distintos servicios estará restringido por los diferentes roles que hay en nuestro sistema.
* El diseño de la API no reflejará la estructura de BBDD para evitar posibles inyecciones SQL.
* La pulseras de uso serán solo de modo lectura, evitando que al ser de escritura y lectura algún usuario con conocimientos más avanzados pueda cambiar los identificadores de las mismas pulseras.
* Validación de los campos tanto en el lado del cliente como en el lado del servidor.
* Para mantener la sesión de usuario, se llevará a cabo el cifrado de login y contraseña en un fichero, de modo que cuando el usuario acceda de nuevo, pueda acceder a dichos datos de sesión y entrar sin autenticarse.

**Autenticación**

* Inicio de sesion con usuario y contraseña, que será cifrada con un conjunto de funciones hash criptográficas SHA512.
* La seguridad en el proceso de autenticación se va a basar en la implementación de token para garantizar la identidad del usuario.
* Cada vez que se mande una petición al servidor, se adjuntará el token actual, y el servidor devolverá uno nuevo. Dicho token, caducará tras un periodo determinado de tiempo. Por eso decimos que utilizaremos tokens de un solo uso.

**Especicación API Rest y del modelo de datos**

**Especificación API Rest**

**Login**

POST /rest/login -> Login usuario, empleado y administrador.

**Usuario**

GET/rest/usuario -> Listar todos los usuarios.

GET/rest/usuario/id -> Detalle de un usuario.

GET/rest/usuario/pulsera/id -> Listar todas las pulseras de un usuario(id).

GET/rest/usuario/producto/id -> Listar todos los productos de un usuario(id).

PUT/rest/usuario -> Crear un nuevo usuario.

POST/rest/usuario/id -> Modificar un usuario (modificación parcial).

POST/rest/usuario/pulsera ->Asignar una pulsera a un usuario.

DELETE/rest/usuario/id -> Eliminar un usuario.

**Producto**

GET/rest/producto -> Listar todos los productos.

GET/rest/producto/id -> Detalle de un producto.

GET/rest/producto/categoria -> Listar categorias de los productos.

GET/rest/producto/categoria/id -> Listar productos de una categoría.

GET/rest/producto/empleado ->Devolver empleado asignado a un producto.

GET/rest/producto/reservas/id -> devuelve los usuarios por producto.

PUT/rest/producto -> Crear un nuevo producto.

POST/rest/producto/id -> Modificar un producto.

PUT/rest/producto/categoria -> Crear categorías de un producto.

POST/rest/producto/categoria/id -> Modificar una categoría.

POST/rest/producto/categoria -> Asignar una categoría a un producto.

POST/rest/producto/estancia -> Asignar una estancia a un producto.

DELETE/rest/producto/id -> Eliminar un producto.

DELETE/rest/producto/categoria/id -> Eliminar categoría.

**Estancia**

GET/rest/estancia -> Listar las estancias.

GET/rest/estancia/id -> Detalle de una estancia.

GET/rest/estancia/capacidad/id -> aforo máximo de una estancia.

GET/rest/estancia/aforo ->aforo de las estancias públicas.

GET/rest/estancia/aforo/id ->aforo tiempo real de una estancia.

PUT/rest/estancia -> Crear una estancia.

POST/rest/estancia/id -> Modificar una estancia.

DELETE/rest/estancia/id -> Eliminar una estancia.

**ticket**

GET/rest/ticket -> Listar pagos (tickets).

GET/rest/ticket/id -> Información de un pago (ticket).

PUT/rest/ticket -> Crear pago (ticket).

**Pulsera**

GET/rest/pulsera -> Listar pulseras.

GET/rest/pulsera/id -> Detalle de una pulsera.

PUT/rest/pulsera -> Dar de alta una pulsera.

POST/rest/pulsera/id -> Modificar una pulsera.

DELETE/rest/pulsera/id -> Dar de baja una pulsera.

**Esquema de la base de datos relacional**

