Universidad Zaragoza

Trabajo Fin de Grado

Monitorización web en tiempo real de alertas en entornos hospitalarios

Real-time web monitoring for alerts in hospital environments

Autor

Leticia Sánchez Romero

Director

Carlos Aisa Redondo

Ponente

Francisco Javier Zarazaga Soria

Grado en Ingeniería Informática
Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura

Junio 2023

This page is intentionally blank

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

D./Dª. _____,

con nº de DNI _____ en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) _____, (Título del Trabajo)

_____

_____

_____

_____

_____

_____,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, _____

Fdo: _____

This page is intentionally blank

AGRADECIMIENTOS

Agradezco a Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum. Nam quam nunc, blandit vel, luctus pulvinar, hendrerit id, lorem. Maecenas nec odio et ante tincidunt tempus. Donec vitae sapien ut libero venenatis faucibus. Nullam quis ante. Etiam sit amet orci eget eros faucibus tincidunt. Duis leo. Sed fringilla mauris sit amet nibh. Donec sodales sagittis magna. Sed consequat, leo eget bibendum sodales, augue velit cursus nunc,

y especialmente a los alumnos que hacen plantillas de LaTeX.

This page is intentionally blank

Monitorización web en tiempo real de alertas en entornos hospitalarios

RESUMEN

Ibernex es una compañía especializada en el diseño, desarrollo e integración de soluciones y servicios tecnológicos destinados al sector socio-sanitario. Realizan soluciones para automatizar y digitalizar la atención y experiencia de residencias u hospitales.

Actualmente la compañía tiene desarrollada una aplicación que se encarga de la gestión al completo de distintas funcionalidades dentro del sector comentado anteriormente. Esta aplicación se conecta con otras aplicaciones según las necesidades que tienen los distintos clientes de Ibernex.

Las soluciones que se han desarrollado hasta el momento son soluciones orientadas a aplicaciones de escritorio. Sin embargo, la empresa considera necesario que una buena opción sea utilizar alguna de sus funcionalidades en una aplicación web de tal manera que por ejemplo se pueda tener dicha aplicación ejectuando en monitores en una residencia u hospital.

Por esta razón, en este proyecto se desarrolla, cómo método de prueba de cara a que la empresa pueda reutilizar la implementación que consiedere necesaria, la funcionalidad de monitorizar alertas en tiempo real.

This page is intentionally blank

Índice

This page is intentionally blank

Capítulo 1

Introducción

En este capítulo se presenta el contexto del trabajo, así como la motivación del problema concreto que se aborda. Se explica los objetivos y sus limitaciones, además de las herramientas de trabajo utilizadas. Por último, se explica el esquema general de la memoria del proyecto.

1.1. Contexto de trabajo

Ibernex [1] es una empresa zaragonana especializada en el diseño, desarrollo e integración de soluciones y servicios tecnológicos destinados al sector sanitario y socio-sanitario. Estas soluciones se centran en automatizar y digitalizar la atención y experiencia de residencias u hospitales. La empresa cuenta con un largo recorrido de la mano del grupo Pikolín y recientemente se han incorporado dos nuevos accionistas mayoritarios con la intención de potenciar su presencia en el mercado internacional. Actualmente, Ibernex es líder en el sector en Iberia (España y Portugal) donde su principal core de negocio son los centros sociosanitaios, pero también alcanzó el 30 % de facturación en el mercado internacional durante el 2022, más concretamente en Latinoamérica potenciando la digitalización de los hospitales y ciudades de la salud.

Los clientes de la empresa, como se ha comentado anteriormente, son residencias u hospitales que quieren digitalizar el proceso de cuidado y atención de pacientes, además de otros procesos que puedan tener según sus necesidades.

La compañía realiza todo el proceso de construcción del producto.

- Por un lado, la fabricación del hardware que interactúa con el software. En esta parte se encuentran los terminales que están en las habitaciones de los pacientes. Estos terminales son pantallas en las que se pueden realizar distintas

acciones como por ejemplo disparar y codificar alarmas, registrar presencias de trabajadores, registrar tareas u otras actividades según el sector en el que se encuentren.

- Por otro lado, la implementación de Helpnex que es el software de gestión diseñado y desarrollado por Ibernex. Helpnex está diseñado para poder adaptarse a cada cliente ya sea por sus necesidades concretas, capacidades, como perfiles de trabajadores, entre otras. Además, las soluciones que ofrecen también se pueden integrar con otros sistemas de gestión como Resiplus (muy presente en residencias que quieren transformarse y digitalizarse, pero mantener su programa de gestión). La empresa se encuentra en continua innovación para seguir ofreciendo un sistema actualizado que ayude a optimizar la gestión de residencias y hospitales y así mejorar no solo la calidad de vida de los residentes y pacientes, sino también la labor de los trabajadores con el fin de que puedan invertir más tiempo en lo realmente importante, cuidar de las personas.

Los clientes que tiene la compañía actualmente trabajan mayormente con aplicaciones de escritorio. Pero Ibernex considera que sería una buena opción que en residencias u hospitales se pueda monitorizar alarmas mediante la visualización de estas en pantallas. Para esto han considerado que una buena solución sería tener una aplicación web en la que se pueda visualizar en tiempo real las alertas de la institución.

En este contexto la parte del trabajo a realizar (que se explica con más detalle en la siguiente sección) es hacer modificaciones en la implementación de Helpnex e implementar el frontal web, que será el que reciba las alertas en tiempo real, separado de la aplicación actual.

1.2. Objetivos y limitaciones

El objetivo del proyecto es desarrollar un sistema de información en forma de aplicación web que muestra la monitorización en tiempo real de alertas. El trabajo a realizar para llevar a cabo la construcción de la aplicación web consta de dos partes. En primer lugar, implementar un frontend completo desde cero separado de la aplicación actual de Ibernex, que recibirá las alertas en tiempo real y permitirá visualizar cierta información de estas alertas. En segundo lugar, el desarrollo de un backend que se realiza integrando código en Helpnex aprovechando algunas de las funcionalidades existentes y desarrollando las nuevas funcionalidades necesarias.

Estas alertas pueden ser de dos tipos:

- **Presencias**: son alertas que indican la presencia de un trabajador en una habitación. Estas presencias se muestran en los terminales y pueden ser de dos tipos:

    1. Identificadas: un trabajador pasa su tarjeta de identicación por el lector del terminal o intruduce su PIN personal en el terminal.
    2. No identificadas: mediante la pulsación de un botón en la pantalla táctil del terminal.

- **Alarmas**: son alertas generadas por los pacientes cuando hay una situación de emergencia desde los terminales que se encuentran en las habitaciones. Requieren que un miembro del personal sanitario vaya a la habitación para solucionar la emergencia. El ciclo de vida de estas alarmas pasa por tres estados:

    1. Disparada: la alarma ha sido generada y queda pendiente de ser aceptada.
    2. Aceptada: un trabajador indica al sistema que es consciente de la existencia de esa alarma y que va a hacerse cargo de ella.
    3. Atendida: un trabajador ha acudido a la habitación y va a realizar la acción necesaria para solucionar la emergencia. El trabajador puede identificarse introduciendo un PIN personal, pasando una tarjeta identificativa por el terminal o bien puede realizar una presencia anónima pulsando el botón apropiado en el terminal. Tras atender la emergencia la alarma queda pendiente de ser codificada por dicho trabajador.

Se tendrá en cuenta también que el inicio de sesión en la aplicación web se hará con las credenciales de usuario e indicando un puesto. Estos puestos son importantes ya que la información de las presencias se envían a todos los puestos, pero la información de las alarmas se filtra según el puesto.

Este sistema web servirá para pequeños y grandes escenarios. Es decir, será útil para una residencia pequeña en la que el número de clientes, habitaciones y por tanto terminales sea reducido. O el caso contrario en el que se quiera monitorizar las alertas de un hospital de gran tamaño con un alto número de clientes y terminales.

La limitación a la hora de realizar el proyecto es que, al tener ya una aplicación desarrollada con ciertas tecnologías y modelos de datos, se debe realizar el análisis y diseño del sistema en base a esas condiciones y teniendo en cuenta las necesidades

concretas de la empresa, que se podrán ver posteriormente en la sección de requisitos del sistema.

1.3. Herramientas de trabajo

Para desarrollar el proyecto descrito anteriormente es necesario trabajar con las herramientas que ya utiliza la empresa e introducir nuevas.

- El backend se desarrolla con el lenguaje de programación *C#* utilizando *.NET Framework 4.8* [2] ya que la aplicación de la empresa está implementada de esta forma. Además, como entorno de desarrollo se utilza *Microsof Visual Studio* [3].
- El frontend se desarrolla con *React* [4] y utilizando los lenguajes de programación *JavaScript*, *HTML* y *CSS* y teniendo *Visual Studio Code* [5] como editor de código fuente.
- Como software de control de versiones se utiliza *Git*. Para alojar el código del frontend se utiliza *GitHub*. Para alojar el código del backend se utiliza un repositorio de la empresa. Y para clonar el repositorio donde se aloja la aplicación Helpnex se utiliza el cliente *TortoiseGit* [6].

Por otro lado, este proyecto se está desarrollando de tal forma que la estudiante trabaja para la empresa de forma remota, estando la sede de la empresa en Zaragoza, España y la estudiante en Kaunas, Lituania. Por esta razón, cobran mayor relevancia *Gmail*, *Google Chat* y *Google Meet* como herramientas de trabajo utilizadas para mantener la comunicación con la empresa.

1.4. Esquema general de la memoria del proyecto

La estructura seguida en este documento es la siguiente:

- Previamente a esta sección se encuentran los agradecimentos y un resumen completo del proyecto realizado. A continuación el índice que resume esta sección.
- En este mismo capítulo se encuentran tres secciones distintas, además de esta misma. La primera, es el contexto del trabajo donde se explica qué empresa es Ibernex, cómo trabaja y quiénes son sus clientes, además de exponer cómo son sus soluciones y cómo encaja este proyecto en la empresa. La segunda, son los objetivos y limitaciones con los cuales se expone detalladamente qué es lo que se va a realizar y qué es lo que se va a conseguir al final y con qué limitaciones. Y por

último, se exponen las herramientas de trabajo utilizadas durante el desarrollo del proyecto.

- En el segundo capítulo se detalla el análisis y diseño de la aplicación web. En la primera sección se encuentran los requisitos funcionales del sistema acordados con la empresa. En la segunda, se explica la arquitectura software del sistema con ayuda de un diagrama. En la tercera, se expone la información relevante a los modelos y base de datos que utiliza la empresa utilizados en el contexto de este proyecto. Y por último se puede observar la interfaz de usuario de la aplicación web.

- En el tercer capítulo se pueden encontrar dos secciones en las que se proporciona de manera más detallada diversos aspectos de la implementación realizada para el frontend y backend respectivamente.

- En el cuarto capítulo se explica cómo se ha gestionado el proyecto, comentando la planificación inicial y final junto con un análisis explicativo de los posibles cambios.

- En el quinto capítulo se exponen las conclusiones del proyecto y se explica qué conocimientos se han adquirido, técnicamente y personalmente. Además, se proporciona información acerca de una continuación del trabajo en el futuro.

- Por último se puede encontrar detallada la bibliografía, la lista de figuras y la lista de tablas.

Adicionalmente, se incluyen los siguientes anexos que proporcionan mayor detalle, aclaraciones o explicaciones complementarias a los contenidos de los mencionados capítulos:

- Anexo A. Alternativas arquitecturas

- Anexo B. Decisión descarte SignalR

- Anexo C. Decisión descarte JWT

Capítulo 2

Análisis y diseño del sistema

En este capítulo se expone el análisis de requisitos funcionales, la arquitectura software, la base de datos y la interfaz del usuario.

2.1. Requisitos del sistema

Seguidamente (véase tabla 2.1) se presentan, en modo de tabla, los principales requisitos del sistema. Para su formulación se ha partido de los detalles incluidos en la sección 1.2 y se ha trabajado directamente con la empresa.

ID	**Requisito**
RF-1	Un usuario iniciará sesión en el sistema utilizando un identificador de usuario (nickname), contraseña y puesto.
RF-2	Un usuario verá como opciones del campo puesto en la autenticación, las existentes en los recursos de ese tipo de la base de datos. En caso de que no exista ningún recurso de ese tipo, verá la opción "Puesto único".
RF-3	Un usuario podrá cerrar su sesión.
RF-4	El sistema permitirá al usuario navegar en la aplicación mediante un menú lateral.
RF-5	El sistema mostrará al usuario un listado de plantas del edificio.
RF-6	El sistema mostrará alertas de dos tipos al usuario: alarmas y presencias activas.
RF-7	Las alarmas son generadas por los pacientes en una situación de emergencia desde un terminal. Son de uno de los tipos que se encuentran en la base de datos. Se muestran si se encuentran en cualquiera de los tres estados explicados en la sección 1.2. Los colores de estas alarmas según su estado son: rojo si está disparada; amarilla si está aceptada; y azul si está atendida.
RF-8	Las presencias activas son presencias de un trabajador en una habitación y pueden ser los dos tipos explicados en la sección 1.2.
RF-9	El sistema mostrará al usuario el número total de alarmas y presencias activas en cada planta.
RF-10	El sistema permitirá al usuario elegir una planta y ver la información de las alertas correspondiente a esa planta.

RF-11	El sistema mostrará al usuario un carrusel (presentación de tarjetas que se pueden recorrer de izquierda a derecha) con las alertas de la planta seleccionada.
RF-12	El usuario podrá ver las alertas más actuales en las primeras posiciones (izquierda) del carrusel.
RF-13	El sistema mostrará al usuario la información del paciente de las alarmas activas y el lugar y momento en el que se han disparado.
RF-14	El sistema mostrará al usuario la información del trabajador de las presencias activas identificadas y el lugar y momento en el que se han producido.
RF-15	El sistema mostrará al usuario el plano (imagen guardada en la base de datos) de la planta seleccionada.
RF-16	El sistema destacará al usuario en el plano las alertas activas en cada habitación coloreando la habitación según el tipo de alerta, y en caso de ser alarma también según su estado, siguiendo la siguiente prioridad: 1. Rojo: alerta de tipo alarma y estado disparada 2. Amarillo: alerta de tipo alarma y estado aceptada 3. Azul: alerta de tipo alarma y estado atendida 4. Verde: alerta de tipo presencia
RF-18	El sistema mostrará al usuario un icono en la habitación según la prioridad.
RF-19	El sistema permitirá clicar en los iconos para ver una tarjeta emergente con la información concreta de la alerta con mayor prioridad de las que se encuentran en el carrusel.
RF-20	El sistema mostrará al usuario el listado de alarmas pendientes con su información, sin filtrado por planta, e indicando además el tipo de alarma.
RF-21	El sistema mostrará al usuario el número total de alarmas y presencias de la institución.

Tabla 2.1: Requisitos funcionales del sistema

Los requisitos no funcionales definen los atributos de calidad del sistema describiendo de qué manera opera el sistema. Se presentan a coninuación (véase tabla 2.2).

ID	**Requisito**
RNF-1	Será necesario disponer de conexión a la LAN en la que esté instalada el sistema.
RNF-2	El sistema estará disponible siempre y cuando el PAServidor y los servicios implicados estén activos. (Veáse la explicación de la arquitectura en la sección 2.2).
RNF-3	El sistema será escalable y elástico para adaptarse a los diferentes tamaños de edificios, y por tanto al número de alertas activas, según la institución en la que se instale el sistema.

RNF-4	La conexión se realizará mediante websockets.
RNF-5	La aplicación web será capaz de reconectarse si pierde comunicación con el servicio.
RNF-6	El sistema está disponible para todos los navegadores más populares en sus versiones actualizadas.
RNF-7	Las contraseñas utilizadas en la autenticación se comparan con el PIN identificativo del trabajador en la base de datos.

Tabla 2.2: Requisitos no funcionales del sistema

2.2. Arquitectura software del sistema

El despliegue del sistema se realizará en una red de área local (LAN) ya que tiene que servir para una institución sanitaria o socio-sanitaria en la que se instalará el sistema configurando la base de datos correspondiente a la información de dicha institución.

Para ver las relaciones entre el software y su entorno se utiliza una vista de distribución estilo despliegue. (Veáse la *Figura 2.1*).

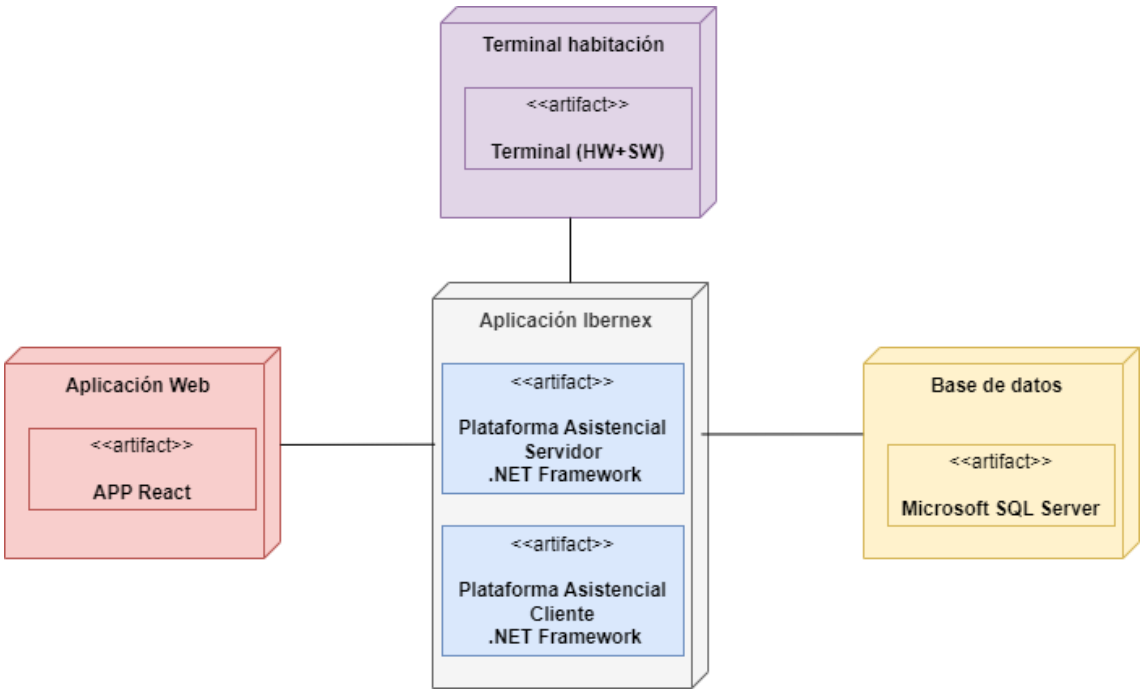

Figura 2.1: Diagrama de despliegue

Con este diagrama se observan los elementos que han estado implicados en el proyecto. No obstante, lo que se ha implementado ha sido: la aplicación web completa

y la funcionalidad añadida a la Plataforma Asistencial Servidor de la aplicación de Ibernex. A continuación se describen los elementos con mayor detalle:

- **Plataforma Asistencial Servidor (PAServidor)** es la aplicación servidor del sistema asistencial Helpnex. Se trata de una aplicación monolítica modular. Dos de los elementos que componen la arquitectura de la aplicación son:
  - Servicios: contienen la lógica del sistema Helpnex. Los servicios se cargan al inicio de la aplicación en función de la licencia adquirida. Cada servicio es el encargado de realizar una tarea específica. Algunos ejemplos de tareas son: comunicación con los terminales de habitación; gestionar las alarmas del sistema; comunicación con la aplicación web de monitorización; localización en interiores; comunicaciones SIP; entre otras.
  - Cola de eventos: es el mecanismo de comunicación que utilizan los servicios para transmitir información entre ellos. Cada servicio se suscribe a una serie de eventos.
- La **base de datos** es la que permite realizar la persistencia de datos de PAServidor, PACliente, la institución, los pacientes y resto de datos necesarios.
- La **aplicación web** que recibe los datos necesarios a monitorizar desde el nuevo servicio añadido a PAServidor para mostrarlos.
- **Plataforma Asistencial Cliente (PACliente)** es la encargada de gestionar la información del sistema Helpnex y guardarla en la Base de Datos. Se comunica con el PAServidor mediante una conexión socket TCP. La información se utiliza por PAServidor para realizar las gestiones necesarias. De igual forma, se compone de módulos que se gestionan mediante una licencia y permiten gestionar distintos elementos. Uno de los módulos sirve para configurar un sistema de reglas de notificación ante una alarma y esto es lo que permite al sistema generar las reglas y notificar por puestos la información que debe ser mostrada en la aplicación web.

  PACliente se ha utilizado en el proyecto para realizar la gestión de plantas, habitaciones, residentes, terminales, trabajadadores, recursos de los puestos y configuración de reglas, teniendo una institución ficticia generada con datos de prueba durante todo el desarrollo.
- El **terminal** es el hardware que se encuentra en la habitación. En este contexto sirve tanto para notificar y codificar las alarmas como para registrar las presencias de trabajadores en las habitaciones. Existen otras tareas adicionales que pueden

ser realizadas, pero que no están directamente relacionadas con el alcance de este proyecto.

La arquitectura aquí expuesta es la elegida entre las distintas opciones barajadas. Estas otras se pueden consultar con detalle en el Anexo A.

Para entender mejor la arquitectura del PAServidor se proporciona un diagrama más detallado. (Veáse la *Figura 2.2*).

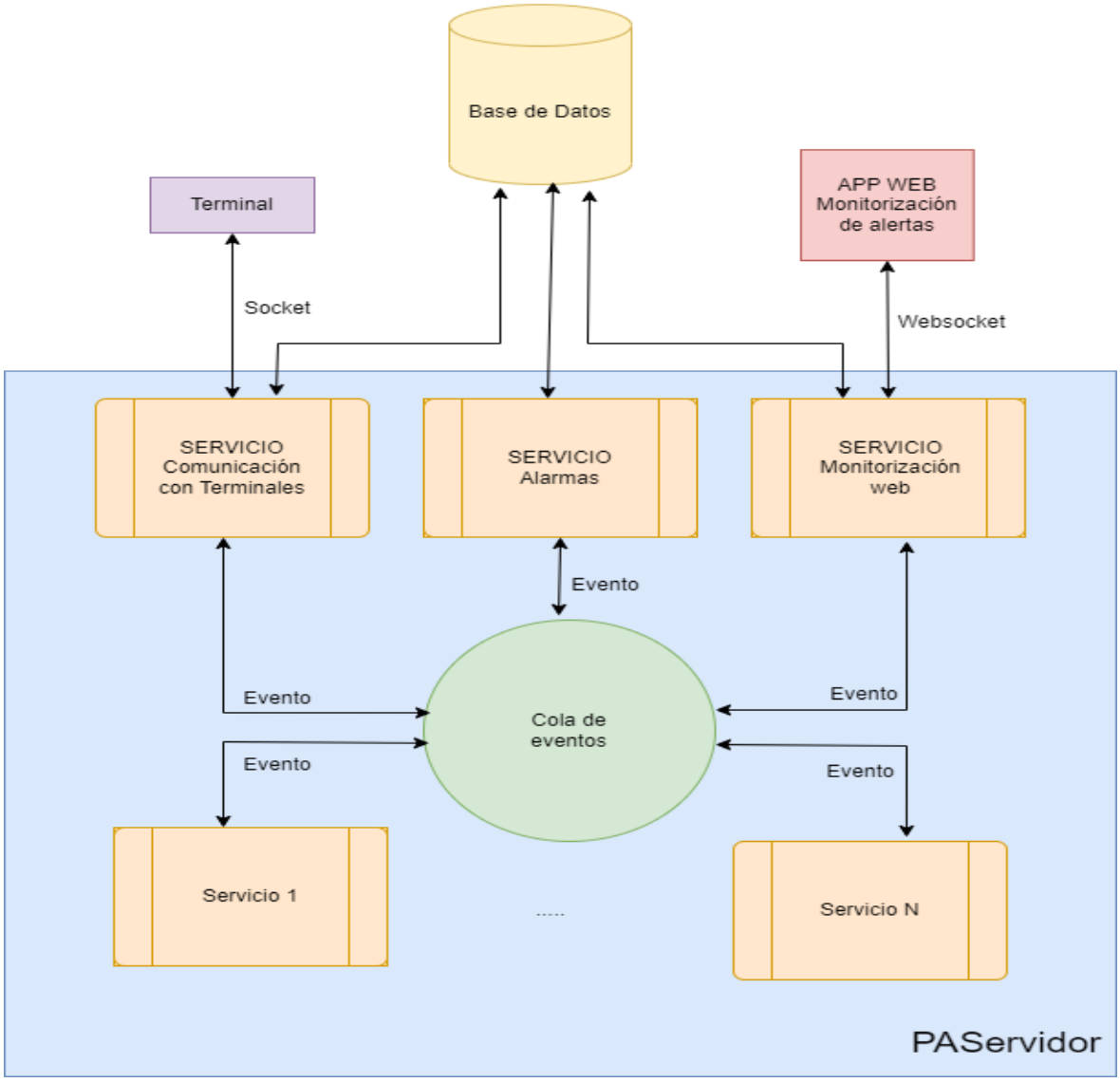

Figura 2.2: Diagrama arquitectura PAServidor

Se puede ver el comportamiento de esta arquitectura en el contexto de este proyecto en la sección 3.2.1.

2.3. Base de datos

Para llevar a cabo el desarrollo, fue proporcionada por parte de la empresa una base de datos de prueba configurada siguiendo el modelo de datos existente de la aplicación de Ibernex.

Parte de la información que se tiene que enviar a la web se puede obtener de los eventos que se reciben en el servicio, pero para otros datos es necesario acceder a la base de datos para obtenerlos. A continuación se exponen los diagramas de la base de datos utilizados en el contexto de este proyecto.

- En la *Figura 2.3* se observa el diagrama de la parte del login y la información de los trabajadores.

- En la *Figura 2.4* se observa el diagrama de los pacientes de la institución.

- En la *Figura 2.5* se observa el diagrama necesario para procesar las alarmas.

- En la *Figura 2.6* se observa el diagrama necesario para procesar las presencias.

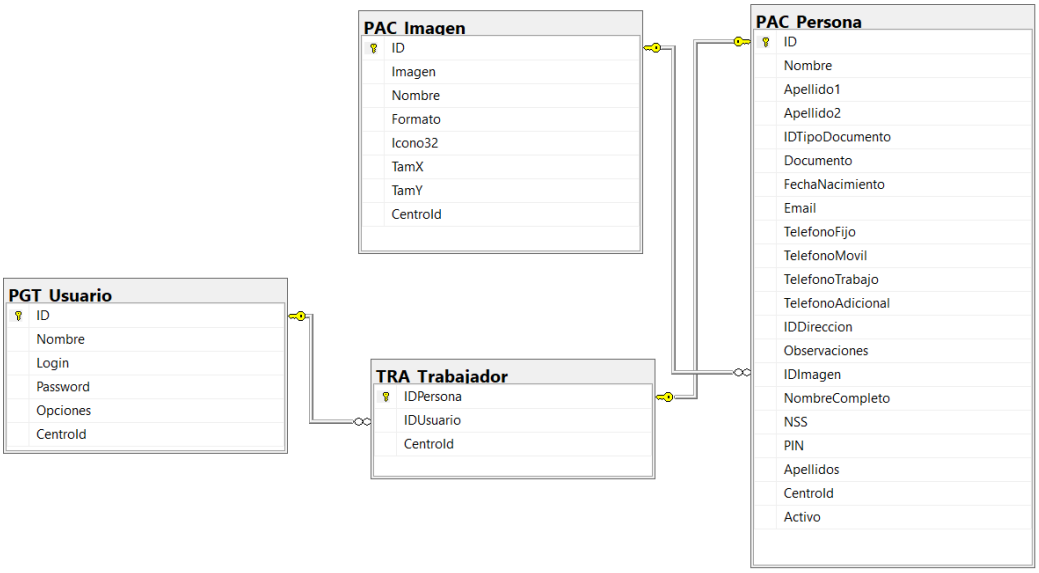

Figura 2.3: Diagrama base de datos - Trabajadores

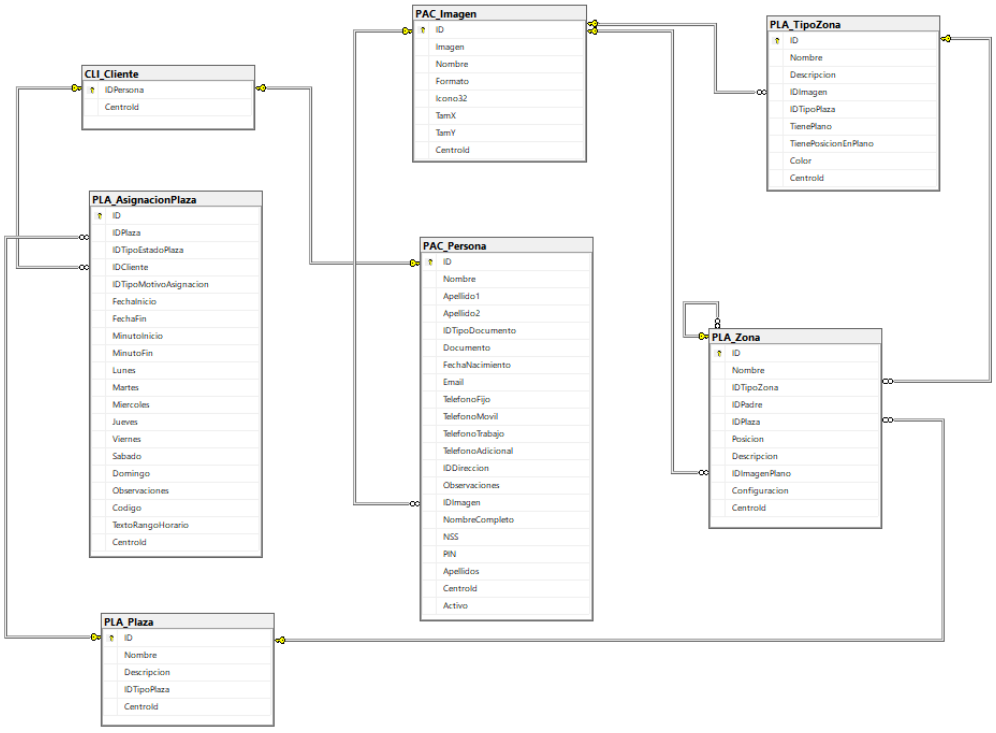

Figura 2.4: Diagrama base de datos - Clientes

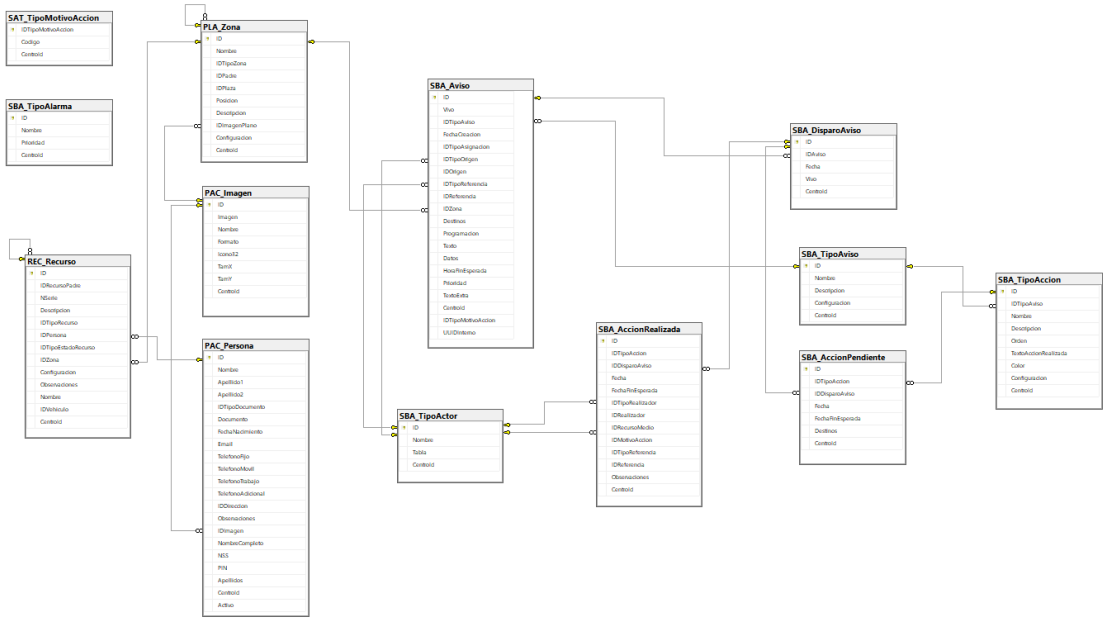

Figura 2.5: Diagrama base de datos - Alarmas

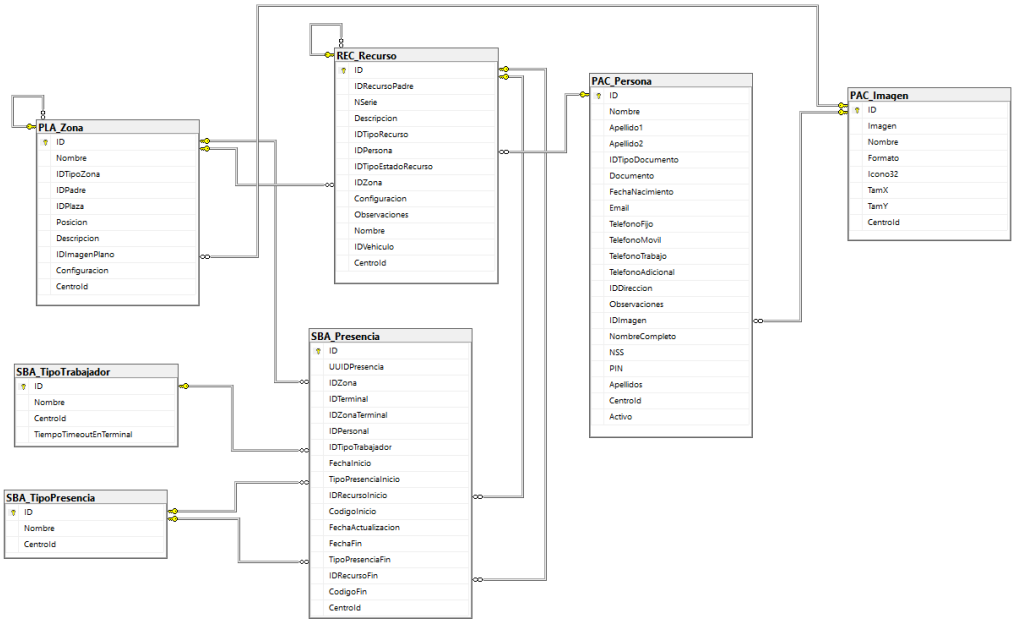

Figura 2.6: Diagrama base de datos - Presencias

2.4. Interfaz de usuario

TODO: Poner la interfaz final del usuario cuando esté terminada

Capítulo 3

Implementación

En este capítulo se exponen ciertas explicaciones relacionadas con el desarrollo del frontend (implementación de la aplicación web) y el backend (implementación del nuevo servicio en Helpnex).

3.1. Implementación del frontend

El frontend de la aplicación web se realiza con React utilizando los lenguajes de programación *javascript*, *HTML*, y *CSS*.

Para la comunicación del frontend con el backend mediante WebSockets se utiliza la librería *reconnecting-websocket* [7]. Se decide utilizar esta frente a otras porque reconecta automáticamente con el servicio si la conexión se cierra por alguna razón y es compatible con *WebSocket Browser API* [8].

La aplicación web es una aplicación visual, en la que no se puede realizar ninguna acción sobre las alertas. Se recibe por websockets toda la información necesaria a monitorizar que procesa el nuevo servicio implementado en PAServidor. Por esta razón, la mayoría de la implementación de esta parte del proyecto se centra en interfaz de usuario aunque también se realiza el tratado de la información que llega para tener los datos de la manera indicada. Cabe destacar que el filtrado por plantas de las alertas, la ordenación de las alertas por más actuales, el filtrado de alertas por prioridad y el mapeado por habitaciones, se realiza en esta parte del proyecto.

Para implementar la aplicación web se ha considerado útil contar con el apoyo de ciertas librerías. Algunas interesantes de mencionar son:

- *Bootstrap* [9] y *React-Bootstrap* [10] como herramientas para la implementación de la interfaz de usuario.

- *react-multi-carousel* [11] para mostrar el carrusel de alertas en una de las pantallas principales de la aplicación.
- *react-icons* [12] ya que provee distintos tipos de iconos para la aplicación.
- *react-moment* [13] como apoyo para utilizar las fechas y horas de las alertas y ordenarlas para mostrar en el carrusel las más actuales primero.
- *react-img-mapper* [14] para facilitar el mapeado de las áreas de las habitaciones en los planos de las plantas y poder mostrar al usuario las habitaciones coloreadas por prioridad junto con el icono de la alerta con más prioridad dentro de dicha habitación y la tarjeta emergente con la información de dicha alerta.

Durante el desarrollo de la aplicación se ha seguido cierta estructura a la hora de dividir los módulos para organizar el código. De esta forma, ha sido más fácil realizar una implementación clara durante el desarrollo. Además, de cara a que la empresa pueda reutilizar el código hace que sea más sencillo comprenderlo. Se puede ver la organización realizada en tiempo de desarrollo con una vista de módulos en forma de diagrama de paquetes. (Véase *Figura 3.1*).

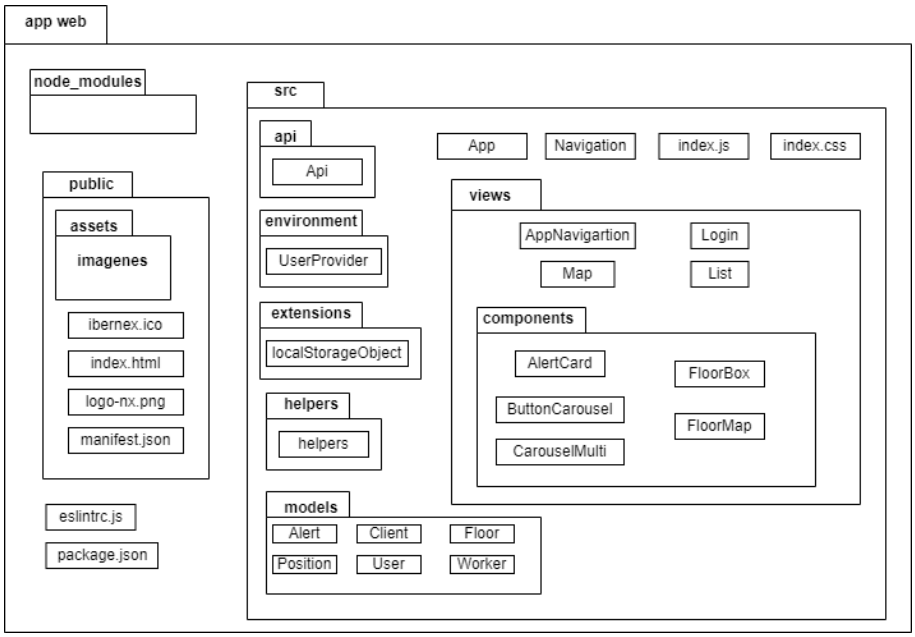

Figura 3.1: Diagrama de paquetes frontend

En el diagrama se pueden ver distintos paquetes que cumplen distintas funcionalidades. Se explican a continuación:

app web contiene *package.json* que es un archivo de metadatos que contiene información sobre el proyecto como las dependencias; **node_modules** que contiene

todos los paquetes y dependencias de npm que se especifican en el archivo anterior; y *eslintrc.json* que es el archivo de configuración para ESLint y que se ha utilizado para tener un código limpio y libre de errores. Por otro lado, se tienen los paquetes:

- **public** que contiene archivos estáticos para el navegador. *assets* contiene imágenes para la aplicación; *manifest.json* y *index.html* que son archivos de configuración de la página web; y otros iconos como el utilizado como favicon.

- **src** Contiene distintos modulos esenciales que son el punto de entrada de la aplicación, la navegación y la configuración de ciertos estilos de la aplicación. Además, se puede ver la organización escogida a la hora de implementar la aplicación con la explicación de la funcionalidad de los paquetes que contiene:

    1. **api**: módulo relacionado con la conexión con el backend.
    2. **environment** y **localstorageObject**: módulos que gestionan y proporcionan información del usuario en todo el contexto de la aplicación.
    3. **models**: módulo que contiene los distintos modelos de datos utilizados en la aplicación web.
    4. **views**: módulo que contiene las vistas principales de la aplicación y **components** que contiene distintos módulos utilizados por dichas vistas.
    5. **helpers**: módulo que contiene un módulo que es utilizado por los módulos de views y que contiene funciones auxiliares.

Otra de las decisiones tomadas en la implementación es utilizar en el login la validación del formulario de bootstrap. De esta forma, los datos no se envían a backend a no ser que todos los campos estén completados. Además, en backend se hace la validación correspondiente, y si hay algún problema se informa al usuario de dicho problema mediante una tarjeta emergente.

Como se explicará en la siguiente sección, el frontend tiene la información de los clientes y trabajadores desde el inicio de sesión, y cuando le llega información de nuevas alertas debe completar cierta información con la que ya tiene. Por esta razón, para que sea rápido y eficiente, se decide guardar esta información en diccionarios para un fácil y rápido acceso al dato concreto a utilizar.

3.2. Implementación del backend

El backend de la aplicación se realiza con *.NET Framework 4.8*, ya que como se ha explicado en secciones anteriores la implementación que se realiza es concretamente añadir funcionalidad a la aplicación existente.

Para la comunicación con el frontend mediante WebSockets se utiliza la librería *websocket-sharp* [15] en su versión *1.0.3.0*. Se decide utilizar esta librería ya que en otra de las funcionalidades existente en la aplicación se utiliza (aunque en una versión anterior), y se cree conveniente utilizar algo similar de cara a que internamente en la empresa puedan entender mejor la implementación o reutilizar el código. La instalación de esta librería se realiza mediante el propio Visual Studio (que se ha comentado la sección de herramientras de trabajo que es el entorno de desarrollo utilizado para implementar el backend) utilizando la opción de administrar paquetes NuGet e instalando el nombrado.

Al principio la empresa propuso realizar la comunicación del backend con el frontend utilizando *SignalR* [16] pero finalmente se descarta por incompatibilidad con la aplicación actual y simplicidad en la arquitectura software. Se puede ver más información referente a esta decisión en el Anexo B.

De los servicios y eventos nombrados en la arquitectura detallada del PAServidor en la sección 2.2, el servicio que se ha implementado para añadir funcionalidad a la aplicación existente y que sirve como backend de la aplicación web, es el *Servicio de monitorización web* que se puede observar en la sección nombrada, concretamente en la *Figura 2.2*.

Durante la implementación del servicio se han tomado distintas decisiones. Algunas de ellas se exponen a continuación.

Cuando un usuario inicia sesión en la aplicación web se envía una carga de datos iniciales con la información acerca de la configuración del centro (plantas con su información y sus planos), los clientes, los trabajadores, y las alertas actuales en el sistema. La decisión se toma valorando la cantidad de datos a enviar y el acceso a la base de datos. Se cree conveniente realizarlo de esta forma, para que la condiguración esté al inicio en la aplicación y en el caso de los clientes y los trabajadores, para no realizar accesos a la base de datos cada vez que se tenga que enviar información

actualizada de alertas, y sea la aplicación web la que complete dicha información. De esta manera el backend es más eficiente y los envíos de la información es más rápido, pero el frontend tiene que hacer cierto procesado con la información que le llega para completarla como se ha comentado anteriormente.

Como se ha mencionado en secciones anteriores, las presencias se mandan a todos los puestos, pero las alarmas se deben enviar a los puestos que cumplan ciertas reglas.

Para saber a qué puestos se debe enviar una alarma, se recibe un evento con esa información, y se utiliza un diccionario cuya clave es cada uno de los puestos disponibles y el valor es un diccionario cuyas claves son los identificadores de las alarmas que deben ser enviadas a ese puesto. Por otro lado, se tiene otro diccionario cuya clave es el nombre del puesto y los valores son un diccionario cuyas claves son los identificadores de las alarmas y el valor son las alarmas (con la información concreta ya procesada) que se envían a cada puesto. Esto es así porque se reciben distintos tipos de eventos según sea para saber a qué puesto notificar en la web, actualización de una alarma, o fin de una alarma.

En el caso de las presencias se tiene un diccionario en el que la clave es el identificador de la presencia y el valor es la presencia a enviar. Se decide utilizar esta colección para almacenar los datos para que los accesos al insertar, actualizar o eliminar una alarma o presencia concreta sean más rápidos. Cabe destacar que en la implementación se ha completado implementado también parte del procesado de alertas de tipo tareas con otro diccionario similar. (Se puede ver más información sobre esto en la sección 5.3 que se habla de trabajo futuro).

Para enviar la información de las alertas se barajó la opción de mandar solo la alerta actualizada y tener cierta lógica en el frontend, o si hacer la lógica completa y envío de todas las alertas actuales en el sistema desde el backend. Finalmente, se optó por esta segunda opción ya que se consideraba que para lo que se necesitaba en estos momentos era la mejor opción para no perder ningún dato si se desconectaba la web del servicio. Para la primera opción, la solución a esto hubiese sido que al reconectar se pidiera la información actual completa.

Durante el desarrollo de la aplicación se ha seguido cierta estructura a la hora de dividir los módulos para organizar el código igual que en el frontend y por las mismas razones. Se puede ver la organización con una vista de módulos en forma de diagrama de paquetes. (Véase *Figura 3.2*).

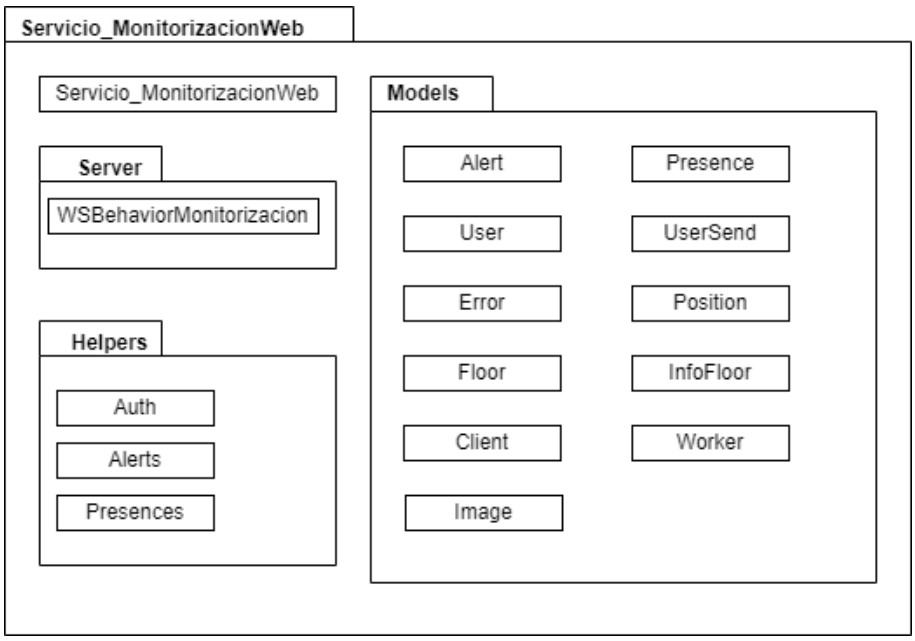

Figura 3.2: Diagrama de paquetes backend

Se observa como se tiene un módulo principal **Servicio_MonitorizacionWeb** que es donde se procesa la llegada de eventos. Por otro lado se tiene el módulo **Server** donde se encuentra la lógica para recibir y enviar información a la aplicación web. Los módulos **Helpers** y **Models** son utilizados por los dos anteriores.

3.2.1. Comportamiento servicios y eventos

Respecto al comportamiento de los servicios y eventos implicados, a continuación se exponen uno ejemplo de funcionamiento tanto para alarma como para presencia.

El funcionamiento para el caso de las alarmas es el siguiente:

1. El paciente pulsa el botón del terminal para solicitar ayuda.

2. El terminal notifica al *servicio de comunicación con terminales*, mediante el socket, que alguien ha pulsado el botón.

3. El *servicio de comunicación con terminales* envía un evento *NuevaAlarma* con toda la información relativa al terminal.

4. El *servicio de alarmas*, que está suscrito al evento *NuevaAlarma*, recibe el evento y lo procesa. Como resultado envía un evento *NuevaAccionPendiente* y *NotificarAlarmaEnMonitorWeb*.

5. El *servicio de monitorización web* en primer lugar recibe el evento *NuevaAccionPendiente* al que está suscrito pero se ignora ya que aún no se ha recibido el evento *NotificarAlarmaEnMonitorWeb*. Seguido se recibe el evento *NotificarAlarmaEnMonitorWeb* y se procesa, envíando mediante websocket la información conveniente a la aplicación web.

6. En la aplicación web aparece la nueva alarma. Después se efectuará una presencia en el terminal. (Véase el comportamiento específico para las presencias).

7. El *servicio de monitorización web* recibe un evento *NuevaAccionPendiente* cuando se actualiza la alarma y lo procesa si ha llegado el evento *NotificarAlarmaEnMonitorWeb* para actualizar la alarma indicada. El resultado de la actualización es envíado mediante el websocket a la aplicación web.

8. En la aplicación web aparece la alarma actualizada y después en el terminal un trabajador codifica la alarma y elimina su presencia. (Véase el comportamiento específico para las presencias).

9. El *servicio de monitorización web* recibe el evento *DisparoAvisoFinalizado* cuando se codifica una alarma y lo procesa si le ha llegado el evento de *NotificarAlarmaEnMonitorWeb* para eliminar la alarma indicada. El resultado de la actualización de las alarmas en el sistema es envíado mediante el websocket a la aplicación web.

10. En la aplicación web desaparece la alarma actualizada.

El funcionamiento para el caso de las presencias es el siguiente:

1. Un trabajador se identifica en la habitación pasando su tarjeta personal por el lector o con el su PIN.

2. El terminal notifica al *servicio de comunicación con terminales*, mediante el socket, que alguien está presente.

3. El *servicio de comunicación con terminales* envía un evento *NuevaPresencia* con toda la información relativa a la presencia.

4. El *servicio de alarmas* que está suscrito al evento *NuevaPresencia* recibe el evento y lo procesa. Como resultado envía un evento *NotificarEstadoPresencias*.

5. El *servicio de monitorización web* recibe el evento *NotificarEstadoPresencias* que puede ser de las presencias actuales, de una presencia nueva, actualizada o

eliminada y lo procesa. Como resultado envía la información necesaria mediante el websocket a la aplicación web.

6. En la aplicación web aparece, se actualiza o desaparece la presencia.

Capítulo 4

Gestión del proyecto

TODO: Esta sección no la subdividiría. Pon una planificación inicial, la final, y comenta el porqué de las variaciones

Capítulo 5

Conclusiones

5.1. Conclusiones

El proyecto ha finalizado con éxito cumpliendo con los objetivos descritos en la sección 1.2 y los requisitos expuestos en la sección 2.1.

Se ha desarrollado un sistema de información en forma de aplicación web que monitoriza en tiempo real alertas, ya sean presencias identificadas o no, o alarmas con sus distintos estados.

Para esto se ha desarrollado desde cero el frontend de la aplicación; se han estudiado las distintas alternativas de arquitectura software, partiendo del esquema propuesto inicialmente por la empresa; se han analizado las opciones de comunicación para las distintas arquitecturas; y se ha implementado correctamente el backend de la aplicación añadiendo la lógica adicional necesaria en la aplicación existente. Todo esto aplicando las decisiones finales debatidas con la empresa referentes a la arquitectura del sistema y la comunicación entre el frontend y backend.

5.2. Conocimientos adquiridos

En esta sección se presentan los conocimientos técnicos y personales adquiridos con la realización de este proyecto.

5.2.1. Conocimientos técnicos

En cuanto a conocimientos técnicos, se destaca que para la parte de frontend se había trabajado anteriormente con *React* por lo que se tenía experiencia previa en desarrollo de aplicaciones web con dicha tecnología, además de con los lenguajes de programación utilizados para esta parte del proyecto. Aunque se ha podido trabajar

aspectos que anteriormente no habían sido abordados como trabajar con el mapeado de imágenes.

Para el control de versiones ya se había trabajado con *Git* y *GitHub* anteriormente, por lo que ha sido fácil seguir trabajando con estas herramientas.

Por el contrario, las herramientas de la parte del backend no se habían utilizado, por lo que ha sido parte del reto de este proyecto familiarizarse con el lenguaje *C#* junto con la tecnología *.NET Framework* utilizando como entorno *Microsoft Visual Studio*. Esto es beneficioso ya que se añaden todos estos conocimientos de cara a tener más aptitudes para la salida al mundo laboral.

Además, ha sido la primera vez que se ha tenido la oportunidad de trabajar con websockets ya que anteriormente, solo se había colaborado en proyectos de desarrollo en equipo, sin asumir la responsabilidad específica de trabajar con esta tecnología.

De igual manera, se ha conocido *SignalR*, que aunque finalmente no se haya utilizado en el proyecto, se estuvieron haciendo algunas pruebas con dicha tecnología, y para un futuro ya se tiene conocimiento de lo que es y para qué sirve.

5.2.2. Conocimientos personales

En cuanto a los conocimientos personales adquiridos, en primer lugar cabe mencionar la experiencia adquirida con los conocimientos técnicos expuestos anteriormente.

Cabe destacar principalmente que ha sido la primera experiencia laboral realizando un proyecto software en un entorno relacionado con el grado cursado, ya que se ha realizado este Trabajo de Fin de Grado como estudiante en prácticas en Ibernex. También ha sido un reto y un aprendizaje trabajar de forma remota por estar realizando el último año del grado en Kaunas, Lituania. Gracias a esto se ha aprendido a sobrellevar y solucionar las dificultades que se han tenido, sobre todo cuando surgían inconvenientes con la conexión a la red de la empresa, o la diferencia de poder consultar algo estando en la empresa rodeada del resto del equipo de desarrollo frente a estar a distancia y depender de distinto uso horario.

Esta situación ha sido beneficiosa en el sentido en que que se han mejorado las habilidades de organización para lidiar con dichas dificultades, y mantener al

mismo tiempo los compromisos académicos con la universidad de destino y laborales simultáneamente.

5.3. Trabajo futuro

La implementación realizada solo incluye funcionalidad para monitorizar en la web los dos tipos de alerta comentados en secciones anteriores: las presencias y las alarmas.

Inicialmente se incluía también la posibilidad de tener tareas como tipo de alerta, por lo que se comenzó parte de la implementación en backend para administrar la llegada de este tipo de eventos, y en frontend, el diseño de modelos e implementación del código se pensó de manera que fuera posible reutilizar o facilitar, sirviendo de base, algunos aspectos del código, como poder usar el mismo modelo de alerta con cualquier tipo. Por esta razón una línea de trabajo futura, siguiendo el contexto del proyecto realizado, sería incluir las alertas en la aplicación web y que estas fueran igualmente monitorizadas. Para esto se tendría que continuar la implementación e incluir cierta funcionalidad tanto en backend como en frontend.

Se destaca que la aplicación web desarrollada sirve de base y prueba a Ibernex para ver cómo es incluir la implementación web en la empresa y ver cómo funciona en este caso la monitorización de alertas. Por lo que, la principal línea futura de trabajo para la empresa, es que gracias a este proyecto pueden analizar si les sería interesante sacar a producción y comenzar a desarrollar soluciones de su aplicación compatibles con web.

Capítulo 6

Bibliografía

[1] Ibernex. URL: https://ibernex.es/ (Fecha de último acceso: 25/05/2023).

[2] .NET Framework 4.8. URL: https://dotnet.microsoft.com/en-us/download/dotnet-framework/net48 (Fecha de último acceso: 13/02/2023).

[3] Microsof Visual Studio. URL: https://visualstudio.microsoft.com/es/ (Fecha de último acceso: 13/02/2023).

[4] React. URL: https://react.dev/ (Fecha de último acceso: 17/02/2023).

[5] Visual Studio Code. URL: https://code.visualstudio.com/ (Fecha de último acceso: 13/02/2023).

[6] TotoiseGit. URL: https://tortoisegit.org/ (Fecha de último acceso: 16/02/2023).

[7] Librería reconnecting-websocket. URL: https://www.npmjs.com/package/reconnecting-websocket (Fecha de último acceso: 15/04/2023).

[8] WebSocket Browser API. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (Fecha de último acceso: 15/04/2023).

[9] Bootstrap. URL: https://getbootstrap.com/ (Fecha de último acceso: 27/05/2023).

[10] React Bootstrap. URL: https://react-bootstrap.github.io/ (Fecha de último acceso: 27/05/2023).

[11] Librería react-multi-carousel. URL: https://www.npmjs.com/package/react-multi-carousel (Fecha de último acceso: 17/03/2023).

[12] Librería react-icons. URL: https://react-icons.github.io/react-icons/ (Fecha de último acceso: 27/05/2023).

[13] Librería react-moment. URL: https://www.npmjs.com/package/react-moment (Fecha de último acceso: 24/05/2023).

[14] Librería react-img-mapper. URL: https://www.npmjs.com/package/react-img-mapper (Fecha de último acceso: 24/05/2023).

[15] Librería websocket-sharp. URL: http://sta.github.io/websocket-sharp/ (Fecha de último acceso: 28/04/2023).

[16] SignalR. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/signalr (Fecha de último acceso: 13/04/2023).

[17] Librería microsoft/signalr. URL: https://www.npmjs.com/package/@microsoft/signalr (Fecha de último acceso: 10/04/2023).

[18] JWT. URL: https://jwt.io/ (Fecha de último acceso: 17/04/2023).

[19] Create an ASP.NET Core app with React in Visual Studio. URL: https://learn.microsoft.com/en-us/visualstudio/javascript/tutorial-asp-net-core-with-react?view=vs-2022 (Fecha de último acceso: 12/04/2023).

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Alternativas arquitecturas

En el proceso de diseño de la arquitectura software del proyecto se evaluan distintas opciones con el objetivo de seleccionar la más adecuada para cumplir con los objetivos y requisitos de la empresa. Los principales motivos de este análisis son:

- Tener o no un middleware que actúe de servidor web para comunicar la aplicación web y la aplicación existente de Ibernex (Helpnex).
- El problema de compatibilidad de utilizar SignalR (Véase Anexo B) para comunicar la aplicación de Ibernex y la aplicación web.

En un principio, el diseño de arquitetura propuesto por la empresa es el que se puede ver en la *Figura A.1*. Lo que proponían incluía un middleware para realizar una autenticación mediante API REST utilizando JWT (Véase Anexo C) y la comunicación entre los componentes de la arquitecura con websockets utilizando SignalR.

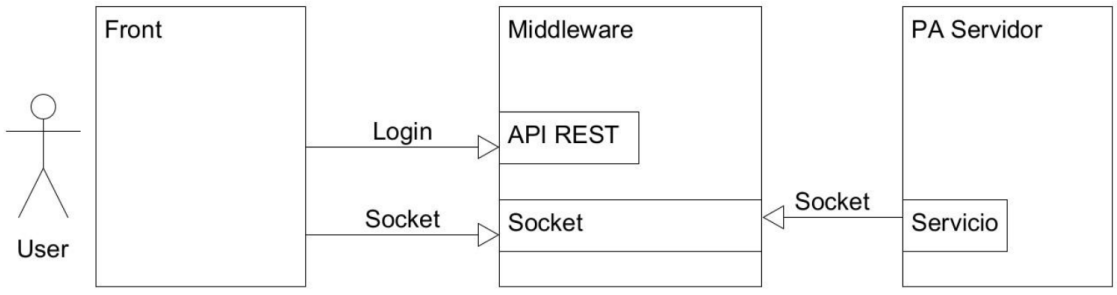

Figura A.1: Diagrama propuesto por la empresa

Durante el desarrollo del proyecto la empresa decide que no es necesario utilizar una autenticación por JWT en estos momentos, por lo que tener un middleware con API REST no tiene sentido si el resto de la comunicación se puede hacer por websockets.

Ahora, lo que se necesita es la comunicación mediante websockets entre el nuevo servicio implementado en PAServidor y la aplicación web. La empresa había propuesto

utilizar SignalR como tecnología para realizar dicha comunicación. El problema se encuentra cuando haciendo pruebas de funcionamiento, para utilizarla en la implementación, se ve la incompatibilidad de SignalR con .NET Framework 4.8 que es la tecnología con la que se implementa Helpnex.

Por esta razón, se buscan alternativas para utilizar SignalR. Las propuesta que se le proponen a la empresa son:

1. Seguir utilizando un Middleware (sin API REST) que siga utilizando SignalR para comunicarse con el frontend React y websockets (sin SignalR) para comunicarse con PAServidor.

2. Realizar la comunicación entre aplicación web y servicio de PAServidor directamente con websockets sin utilizar SignalR.

Para la primera opción el diagrama de despliegue que se propone a la empresa es el que se puede ver en la *Figura A.2*. Aquí se pueden ver una aplicación web React, un servidor web que hace de proxy con ASP .NET Core (que sí es compatible con SignalR) y PAServidor, como tres proyectos independientes. Pero, para esta solución, la empresa pensaba que lo que se haría sería algo similar teniendo un único proyecto incluyendo la aplicación web y el servidor web (un proyecto en el entorno de desarrollo de Microsoft Visual Studio en el que se crea una aplicación ASP.NET Core con React [19]) y por otro lado el PAServidor (Véase *Figura A.3*)

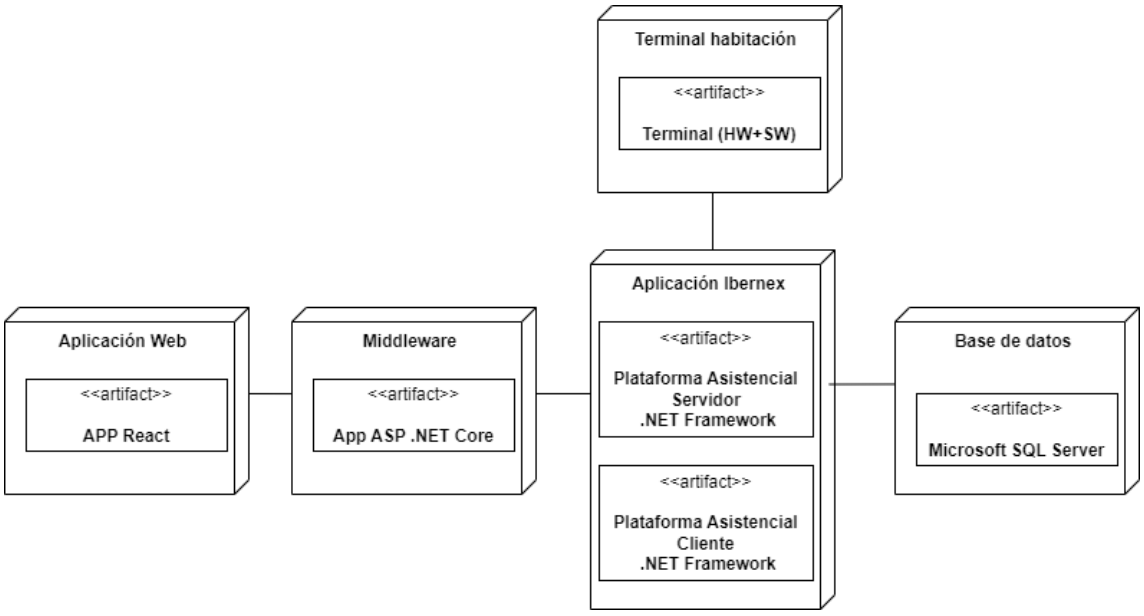

Figura A.2: Diagrama de despliegue con Middleware

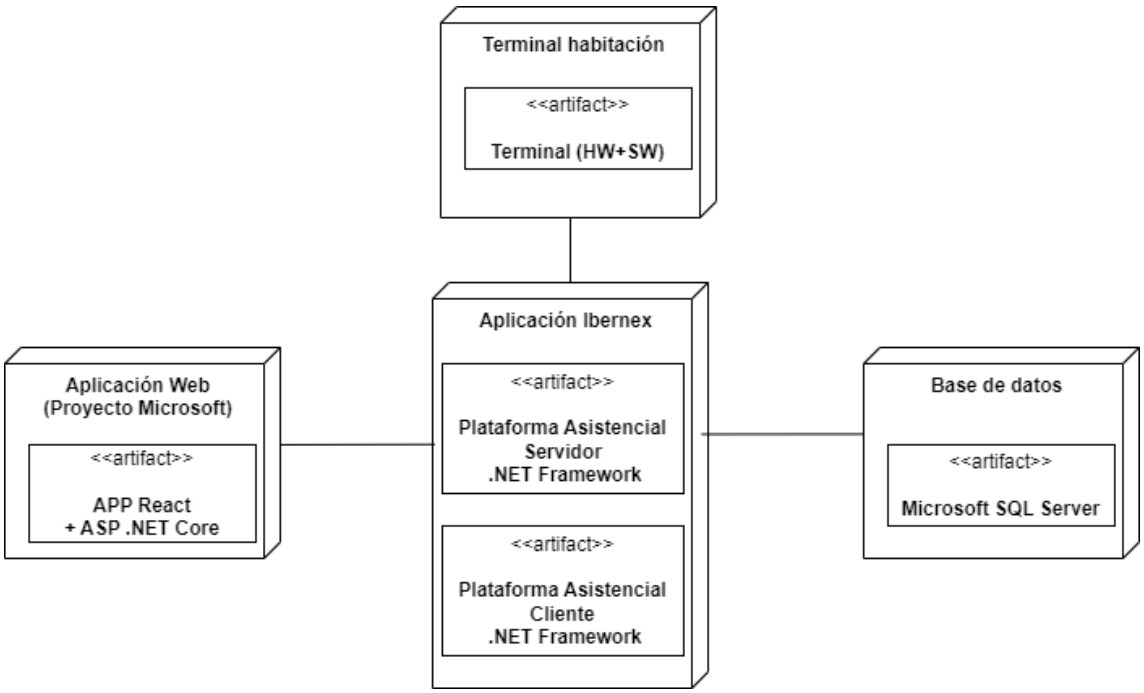

Figura A.3: Diagrama de despliegue - único proyecto

Finalmente, se decide junto con la empresa que es innecesario tener un servidor web si la comunicación se puede hacer directamente entre la aplicación web y PAServidor, aunque sea sin utilizar SignalR (ya que la empresa no contaba con la incompatibilidad).

Por esta razón, para la segunda opción el diagrama de despliegue que se propone a la empresa es el de la arquitectura final elegida para el proyecto que es la que se puede ver en la *Figura 2.1* que se encuentra en una de las secciones anteriores.

Anexos B

Decisión descarte SignalR

SignalR [16] es una biblioteca de código abierto que simplifica la adición de funcionalidad web en tiempo real a las aplicaciones.

En un principio SignalR es la opción a utilizar en la implementación propuesta por la empresa para establecer la comunicación entre el backend y el frontend.

La decisión de descartar SignalR se toma en el momento en el que se observa que no es compatible con .NET Framework 4.8 (y sí lo es con ASP.NET Core) y durante la evaluación de las distintas opciones de arquitecturas software para el sistema, explicadas con detalle en el Anexo A.

La razón de esta decisión es que, para utilizar SignalR, la arquitectura elegida debería de haber sido la que contiene un middleware entre el frontend y backend, de tal forma que este middleware estuviese implementado con ASP.NET Core para que fuese compatible. Con esta arquitectura la comunicación entre middleware y frontend podría haber sido totalmente compatible utilizando en React la librería *microsoft/signalr* [17]. Pero para comunicar con el backend se debería de haber utilizado en la aplicación Helpnex otra librería para websockets como *websocket-sharp* [15] que es la finalmente escogida a utilizar teniendo en cuenta la arquitectura final elegida.

Por lo que se toma la decisión teniendo en cuenta utilizar o no SignalR y la elección de una arquitectura más compleja o más simple y que sea la más rentable en el momento para la empresa.

Anexos C

Decisión descarte JWT

Inicialmente la empresa propuso utilizar como método de autenticación JSON Web Tokens (JWT) [18] para cuando se realizara el inicio de sesión almacenar en local storage el token.

Esto se planteaba para el caso de que la arquitectura software fuese la que se planteaba inicialmente en la que se tenía un middleware ASP.NET Core como capa intermedia entre el backend, es decir la aplicación existente, y el frontend. Pero finalmente se decide junto con la empresa que la arquitectura no va a tener esa capa intermedia, ya que no la consideran necesaria. (Se puede ver más información sobre las distintas arquitecturas planteadas en el Anexo A).

Por lo que la razón de esta decisión es que, al utilizar una arquitectura sin capa intermedia, en la que la comunicación entre el frontend y el backend se realiza mediante websockets, no se considera necesario meter este tipo de autenticación. Sino que el inicio de sesión se realiza utilizando directamente los websockets. Esto es así también, porque el entorno de despliegue en el que estaría la aplicación web sería en la red de área local (LAN) del hospital o residencia.