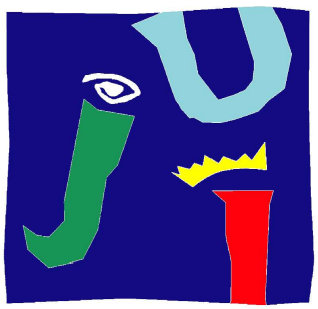

UNIVERSITAT JAUME·I

Grado en Ingeniería Informática

Trabajo Final de Grado

Uso de contenedores (Docker) en aplicación web ASP.NET CORE 2

Realizado por:
Aglaya Pons Martínez

Supervisado por:
Jose Luis Guillén

Tutorizado por:
Juan Carlos Amengual Argudo

Fecha de lectura: 19 de Septiembre de 2018
Curso académico: 2017/2018

Resumen

Hoy en día, la gestión de los datos de una empresa tiene que ser accesible desde cualquier lugar y desde cualquier dispositivo para sus empleados, ya que es posible que tengan que trabajar desde fuera de la empresa o no tienen un horario fijo y necesitan informar y estar informados continuamente.

En este proyecto se quiere desarrollar una aplicación web, junto a una base de datos, desde la cual se puedan manejar los datos de una escuela de música. De esta forma, la escuela podrá gestionar los datos de sus alumnos, de los miembros del personal y de las clases que imparte. Además, la aplicación también pretende ofrecer la comunicación unidireccional e individual de un miembro del personal a un alumno.

En este documento se detalla el proceso de desarrollo del proyecto para la creación de una aplicación web de gestión utilizando el *framework* ASP.NET Core y SQL Server para la gestión de la base de datos.

Palabras clave

ASP.NET, SQL Server, typescript, C#, Docker.

Keywords

ASP.NET, SQL Server, typescript, C#, Docker.

Índice general

Índice de figuras y tablas

Capítulo 1

Introducción

1.1 Contexto y motivación del proyecto

La empresa en la que se realiza el proyecto es CaliforniaDrims. Se trata de una pequeña empresa fundada en 2007 que, según definen en su página web [1], se dedica a implantar y administrar *Google Apps for Work* y a desarrollar aplicaciones para *Google Apps*. Explicándolo de otra manera, la empresa se dedica a desarrollar aplicaciones que se utilizan como extensiones para las herramientas de **Google**. Actualmente, la empresa también se dedica a desarrollar e implantar productos web de gestión en empresas de la zona de Castellón.

El proyecto propuesto consiste en desarrollar un sistema para gestionar el alumnado, el personal y las actividades de una escuela de música.

Este sistema debe ser accesible por el personal administrativo, de dirección y docente. Debe ser una aplicación web *responsive*, ya que se desea que sea accesible desde cualquier ubicación y desde cualquier tipo de dispositivo.

La motivación para la creación de este sistema se basa en que las herramientas online de **Google GSuite** interconectadas (hojas de cálculo, calendarios, etc.) que se emplean para la gestión del alumnado y sus actividades, tienen ciertas limitaciones. Con este sistema de gestión se quiere integrar toda la información en una sola herramienta web y que pueda ser utilizada por todo el personal.

1.2 Objetivos del proyecto

El proyecto que lleva a cabo la empresa para la escuela de música se ha dividido en dos subproyectos. Parte de uno de esos subproyectos es la creación de la base de datos y de la aplicación web de gestión que se detalla en este documento.

El otro subproyecto es la creación de una aplicación móvil desde la que los alumnos de la escuela puedan acceder a la información relacionada con su persona, como su información personal o los mensajes recibidos del personal de la escuela, ingresando sus credenciales en la aplicación. Esa información relacionada con el usuario que está accediendo a la aplicación móvil se extrae de la aplicación web de este proyecto. La aplicación móvil también ofrece otras funcionalidades no relacionadas con la aplicación web de gestión, como el acceso a las noticias relacionadas con la escuela de música. Esta aplicación móvil ha sido creada por Cristian Ionut, un compañero de la carrera que ha realizado las prácticas en la misma empresa.

Como se ha nombrado, los objetivos de este proyecto son el diseño y la creación de una base de datos para almacenar toda la información que la escuela de música quiere gestionar, y el análisis, diseño e implementación de una aplicación web que debe permitir mantener la información del alumnado, el profesorado y las clases. Además, este sistema también debe permitir que los

profesores puedan enviar mensajes a sus alumnos a través de la plataforma y éstos visualizarlos desde la aplicación móvil explicada anteriormente. Por tanto, el alcance del sistema es el siguiente:

• **Alcance funcional**. El sistema debe permitir al personal administrativo, de dirección y docente gestionar la información del alumnado, el profesorado y las clases. También debe permitir al profesorado enviar mensajes a sus alumnos a través de la plataforma.

• **Alcance organizativo**. Al tratarse de una aplicación de uso interno de la escuela de música, el alcance organizativo será igual al funcional, ya que la aplicación será utilizada por el personal de la escuela de música.

• **Alcance informático**. El sistema debe ser capaz de interactuar con el servidor de la base de datos que se emplee. Además, también se debe comunicar con la aplicación móvil que está desarrollando el compañero.

1.3 Estructura de la memoria

En esta sección se resume el contenido de cada capítulo de la memoria:

Capítulo 1: En este capítulo se describe tanto la empresa donde se ha realizado la estancia como los objetivos del proyecto que se ha realizado en ella.

Capítulo 2: En este capítulo se describe el proyecto y se presentan las tecnologías utilizadas.

Capítulo 3: Se presenta la metodología, la planificación y el seguimiento del proyecto. También se detalla un cálculo de los costes del proyecto.

Capítulo 4: En este capítulo se explica el análisis y el diseño del sistema, detallando los requisitos de datos y el diseño de la base de datos.

Capítulo 5: En este capítulo se detalla el proceso de implementación.

Capítulo 6: Se presentan las conclusiones desde el punto de vista formativo y se indican las posibles mejoras del proyecto.

Anexo A: En este anexo se documenta el diagrama de clases.

Anexo B: Muestra las sentencias necesarias para crear las tablas en la base de datos.

Anexo C: Muestra el contenido del fichero CSS utilizado para el diseño de la aplicación.

Capítulo 2

Descripción del proyecto

2.1 Descripción general

Actualmente, la escuela de música utiliza herramientas de **Google** para la gestión general de la escuela. Emplea los formularios de **Google** para la matriculación de nuevos alumnos y las hojas de cálculo para la gestión de alumnos, miembros del personal y las asignaturas que puede impartir. Para gestionar las clases con sus horarios y los eventos que organiza la escuela, utiliza **Google Calendar**.

El formulario que utiliza para la matriculación se muestra en la Figura 1. Las respuestas de este formulario se guardan automáticamente en una hoja de cálculo de **Google** a la que la persona encargada puede acceder fácilmente. Para el registro de nuevos miembros del personal y de asignaturas, tiene que realizar los registros manualmente.

Nuestro sistema no pretende sustituir todas las herramientas que la escuela de música está utilizando hasta el momento. Con la aplicación *web* se pretende gestionar a los alumnos, permitiendo el registro y la gestión desde la misma sustituyendo los formularios. De igual manera se quiere gestionar a los miembros del personal, a las asignaturas y a las clases para no tener que hacer uso de las hojas de cálculo. La gestión de las clases se utiliza simplemente para tener constancia de las clases que se están impartiendo con un horario oficial, pero se quiere seguir utilizando **Google Calendar** para cambios de horario de días específicos que no se van a reflejar en la aplicación *web*.

Aparte de estas funcionalidades, se quiere añadir el envío de mensajes por parte del personal a los alumnos, de forma personal y unidireccional.

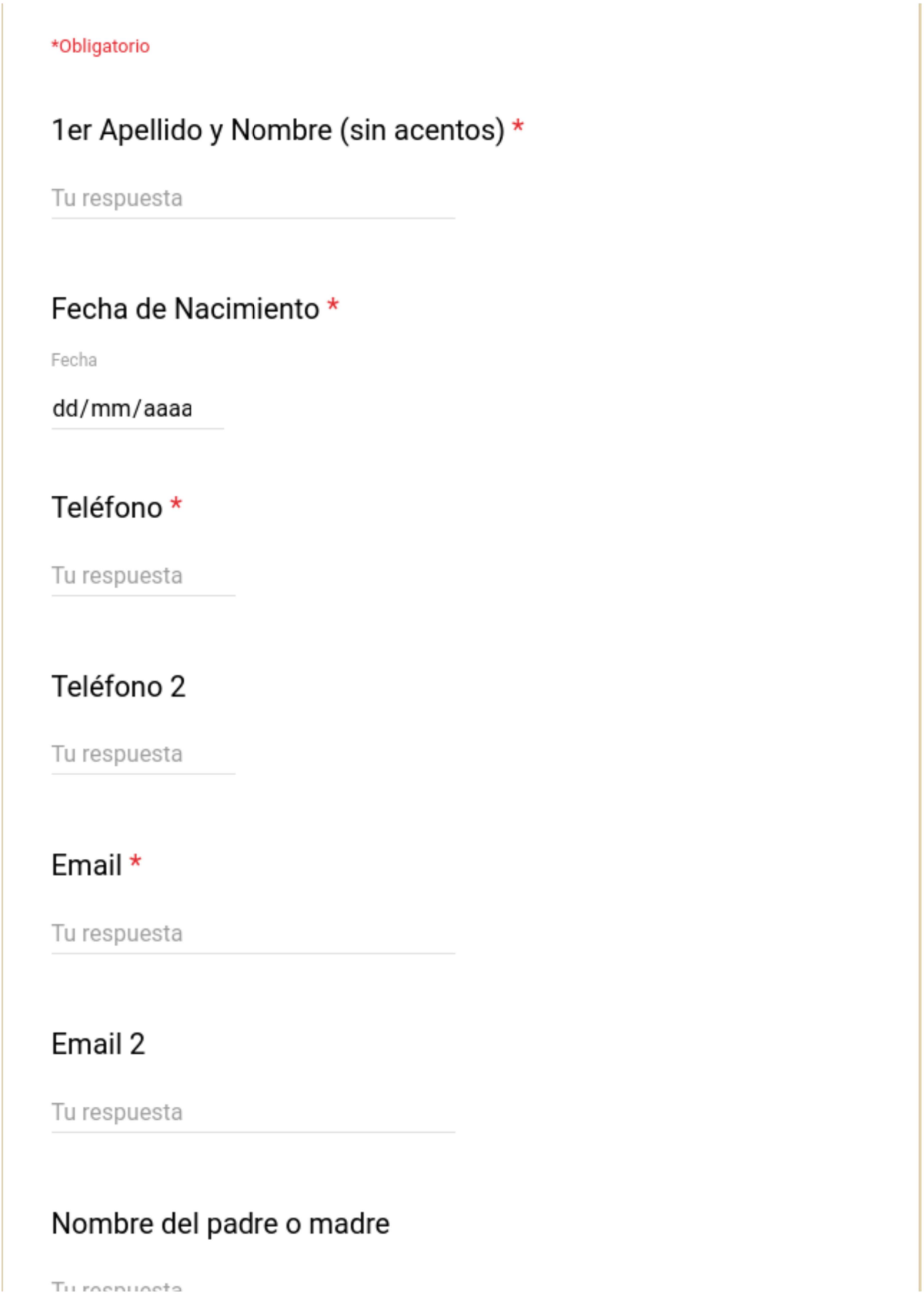

Figura 1: Formulario de matriculación de la escuela de música.

2.2 Herramientas utilizadas

Para la realización de este proyecto se han utilizado las herramientas que se detallan a continuación.

Se ha utilizado **Visual Studio** [2] como herramienta para el desarrollo del sistema. **Visual Studio** es un entorno de desarrollo integrado (IDE) que soporta múltiples lenguajes de programación y entornos de desarrollo web.

Este proyecto utiliza el *framework open source* multiplataforma **ASP.NET Core** [3], que se utiliza para desarrollar aplicaciones conectadas a Internet y basadas en la nube. Con este *framework* y junto a la plantilla libre **Serenity** [4] creada por Volkan Ceylan para **Visual Studio**, creamos nuestra aplicación web, utilizando **C# , Typescript** y **HTML** como lenguajes de programación y **CSS** para el estilo de la aplicación web.

Para tener un repositorio del trabajo realizado, se ha empleado **Visual Studio Team Services** [5].

El sistema para el manejo de bases de datos es **SQL Server** [6].

Se utiliza la plataforma **Azure** [7] para tener la aplicación en la nube. **Azure** es un producto de **Microsoft** que ofrece diversos servicios en la nube que permite crear, administrar e implementar aplicaciones utilizando múltiples herramientas y *frameworks*.

También se ha probado la utilidad de **Docker** [8] en nuestra aplicación para intentar reducir los costos del proyecto. **Docker** es una plataforma proveedora de contenedores donde en cada uno de esos contenedores se puede encapsular una aplicación con todo lo necesario para que se ejecute con éxito.

Se ha utilizado la herramienta **MagicDraw** [9] para realizar el diagrama de casos de uso y el diagrama de clases.

También se ha hecho uso de la herramienta *online* **NinjaMock** [10] para crear los prototipos de la visualización de pantallas de la aplicación *web*.

En este proyecto se ha usado diariamente el calendario que proporciona **Google** para llevar un diario de lo realizado día a día. Esta herramienta se llama **Google Calendar** [11].

Y la última herramienta que se ha utilizado ha sido **Vertabelo** [12], que nos ha permitido realizar el diseño lógico de la base de datos y obtener las sentencias para crear las tablas.

Capítulo 3

Planificación del proyecto

3.1 Metodología

Este proyecto no se ha realizado empleando una metodología ágil en concreto, aunque se puede considerar que se ha empleado una metodología de desarrollo que comparte ciertas similitudes con **Scrum**.

Se han realizado reuniones quincenales en las que han estado presentes el supervisor, un representante de la empresa cliente y el compañero que ha realizado la aplicación móvil. En estas reuniones se le presentaron al representante de la empresa cliente las funcionalidades implementadas tanto en la aplicación web como en la aplicación móvil para que fuesen validadas y pudiese proponer las modificaciones que considerase oportunas cuando ya había un producto mínimo viable. En las reuniones previas a la existencia de un producto mínimo viable, se le presentaron los *mockups* diseñados para ambas aplicaciones para que validara el diseño visual propuesto y, de esta forma, llevarlo a cabo. Estos *mockups* se encuentran más adelante, en la sección 4.3.2.

También se realizaron reuniones semanales con el supervisor y el compañero de la aplicación móvil para organizar el trabajo y establecer objetivos para la siguiente reunión. Cada uno de estos objetivos ha estado compuesto por diversas tareas que había que completar durante un cierto periodo de tiempo, que podría asemejarse a lo que se considera un *sprint* o iteración de desarrollo. Estos periodos no han sido regulares pero han tenido una duración media de 15 días. Por tanto, un objetivo puede no haber estado completo para la siguiente reunión pero parte de las tareas que lo componen sí. Se puede encontrar un ejemplo de un *sprint* o iteración en la sección 3.4.

3.2 Planificación

Para la planificación temporal de este proyecto se ha utilizado una estructura habitual para los proyectos *software* que se puede observar en la Tabla 1.

La duración total de las tareas es de 300 horas, que es la duración establecida para la estancia en prácticas. Hay que tener en cuenta que están planificadas también las reuniones semanales del equipo y las quincenales con el cliente pero estas no se muestran en la Tabla 1 al ser tareas repetitivas.

En la Figura 2 y en la Figura 3 se puede observar el diagrama de **Gantt**, donde se muestra las tareas que se han realizado en paralelo y la consecución entre todas ellas. En este diagrama se muestran, además de las tareas de la Tabla 1, las reuniones semanales del equipo y las reuniones quincenales con el cliente como tareas repetitivas.

	Tarea	Duración	Comienzo	Fin	Predecesoras
1	Comunicación				
2	Estudio del caso	5 horas	12/02/2018	14/02/2018	
3	Formación en tecnologías a emplear	15 horas	12/02/2018	16/02/2018	
4	Definición del alcance y objetivos	5 horas	12/02/2018	14/02/2018	
5	Planificación				1
6	Definición de tareas	15 horas	19/02/2018	21/02/2018	
7	Desarrollo del cronograma	5 horas	22/02/2018	22/02/2018	6
8	Gestión de costes	10 horas	02/03/2018	12/03/2018	7
9	Modelado				5
10	Diagrama de casos de uso	3 horas	13/03/2018	13/03/2018	
11	Requisitos de datos	3 horas	13/03/2018	13/03/2018	
12	Diagrama de clases	8 horas	14/03/2018	15/03/2018	10, 11
13	Diseño				12
14	Diseño de interfaces	10 horas	16/03/2018	22/03/2018	
15	Diseño de la base de datos	11 horas	16/03/2018	22/03/2018	
16	Construcción				9
17	Desarrollo de la base de datos	40 horas	23/03/2018	05/04/2018	
18	Desarrollo del software				17
19	Desarrollo módulo alumnos	13 horas	06/04/2018	17/04/2018	
20	Desarrollo módulo personal	13 horas	06/04/2018	17/04/2018	
21	Desarrollo módulo asignaturas	13 horas	06/04/2018	17/04/2018	
22	Desarrollo módulo relación	46 horas	18/04/2018	30/04/2018	19, 20, 21
23	Pruebas unitarias	20 horas	02/05/2018	07/05/2018	22
24	Puesta en marcha				
25	Formación en Docker	25 horas	08/05/2018	14/05/2018	
26	Encapsulación del sistema en Docker	20 horas	15/05/2018	18/05/2018	25
27	Publicación en Azure	20 horas	21/05/2018	24/05/2018	26
28	Entrega del producto	0 horas	25/05/2018	25/05/2018	27
Total proyecto		**300 horas**			

Tabla 1: Estructura de la planificación del proyecto

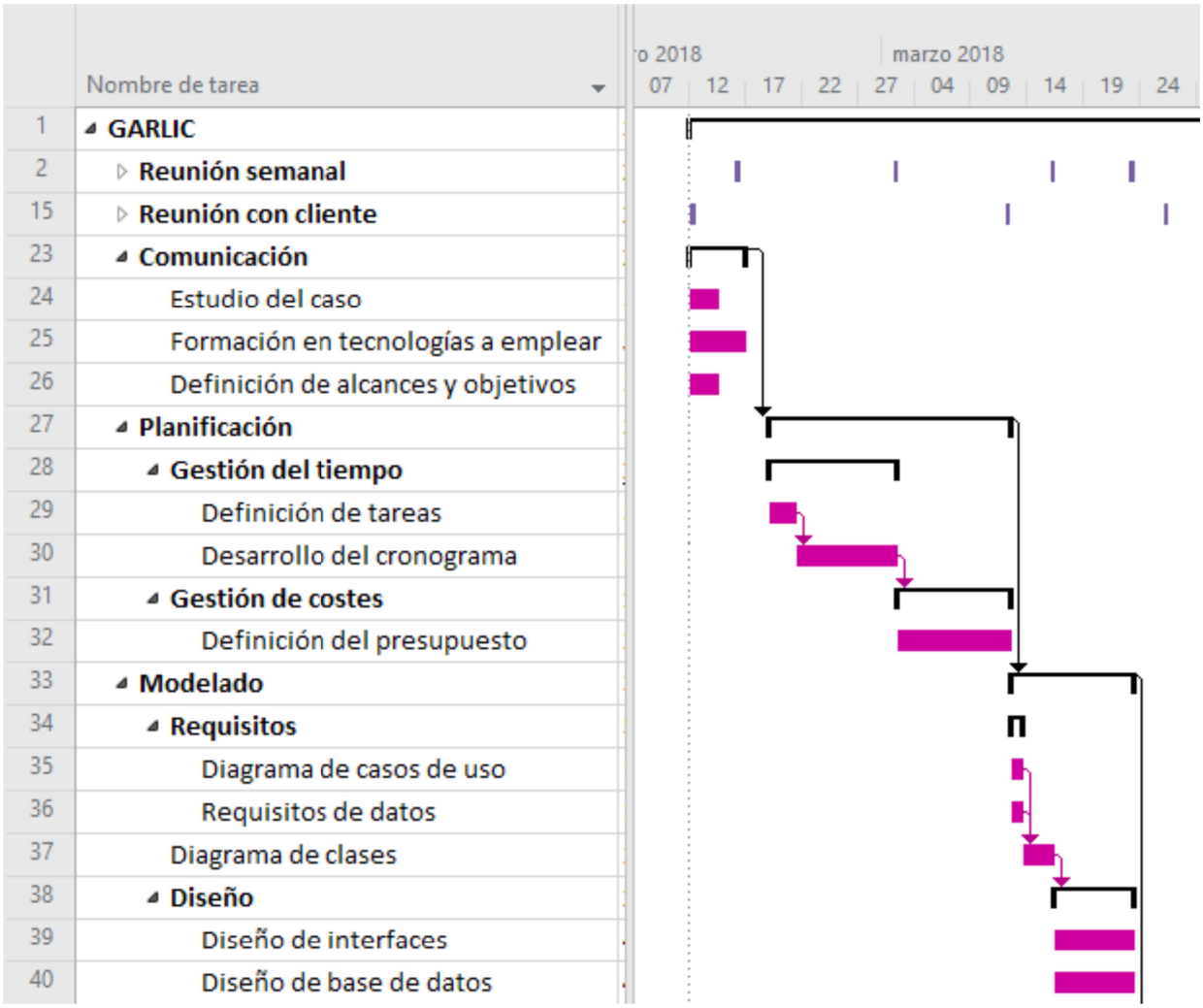

Figura 2: Primera parte del diagrama de Gantt del proyecto. Muestra desde el inicio del proyecto hasta el final del modelado del mismo.

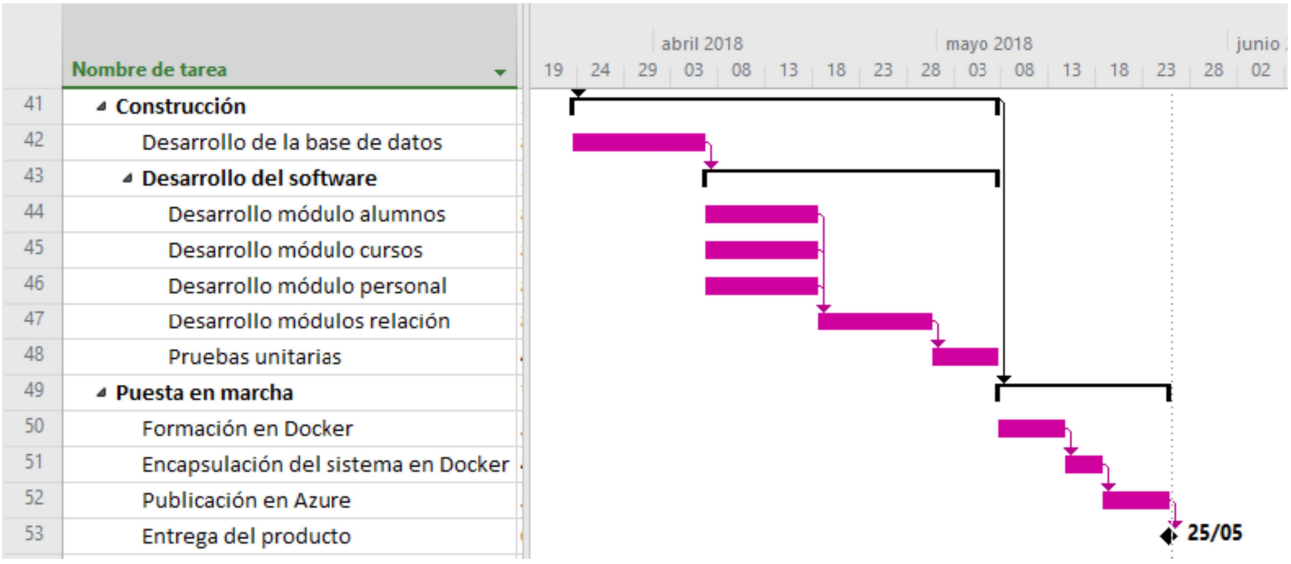

Figura 3: Segunda parte del diagrama de Gantt del proyecto. Muestra a partir de la fase de implementación y hasta la entrega final del producto.

3.3 Estimación de recursos y costes del proyecto

En este apartado detallamos los costes del proyecto.

Según la guía Hays [13], un desarrollador .NET sin experiencia tiene un sueldo base de 23.000€ al año en el 2018. Eso son aproximadamente 10€ la hora. Observando la Tabla 1, consideramos que se han realizado tareas de desarrollador a partir de la fase de comunicación y hasta el final del proyecto. Por tanto, en nuestro caso, el coste de contratar un desarrollador .NET durante 210 horas es de 2.100€.

Como también consideramos que se han realizado tareas de analista desde el inicio del proyecto hasta el final de la fase de modelado observando la Tabla 1, se ha consultado en la misma guía mencionada el sueldo de un analista de sistemas y este es de 28.000€ al año, que a la hora son 12,50€. Por tanto, durante 90 horas tiene un coste de 1.125€.

La suscripción a **Visual Studio Professional** es de 454,54€ al año, que al mes son 37,88€. El proyecto se va a realizar en 4 meses y no se va a realizar ningún otro proyecto durante ese tiempo en esa plataforma, por tanto, el coste de la suscripción durante ese tiempo se debe aplicar a este proyecto. El total de coste de Visual Studio es de 151,52€.

Los costes de **Azure** [14] para el proyecto son los siguientes:

- *App Service* con una conexión *IP SSL* por 79,06€ al mes.
- Dos instancias de *Azure SQL Database* por 1.241,83€ al mes.

Si tenemos en cuenta que la aplicación debe estar disponible por al menos un mes, el coste sería de 1.320,89€.

Con respecto al *hardware*, la empresa proporciona un equipo para la realización del proyecto con valor de 1.200€ y una vida útil de cuatro años. Por tanto, el coste proporcional a los cuatro meses de utilización para el proyecto es de 100€.

Para simplificar los datos y calcular el total, se ha creado la Tabla 2.

Sueldo desarrollador .NET	2.100€
Sueldo analista de sistemas	1.125€
Suscripción a Visual Studio Professional	151,52€
Costes Azure	1.320,89€
Coste *hardware*	100€
Total	4.797,41

Tabla 2: Resumen de los costes del proyecto.

3.4 Seguimiento del proyecto

Para realizar el seguimiento del proyecto se ha utilizado **Google Calendar**, tal como fue sugerido por el supervisor de la empresa. Cada día, al finalizar la jornada laboral, se anotaba tanto lo realizado en el día como las tareas que se habían quedado pendientes para poder retomarlas al día siguiente. Un ejemplo de la utilización de **Google Calendar** se muestra en la Figura 4.

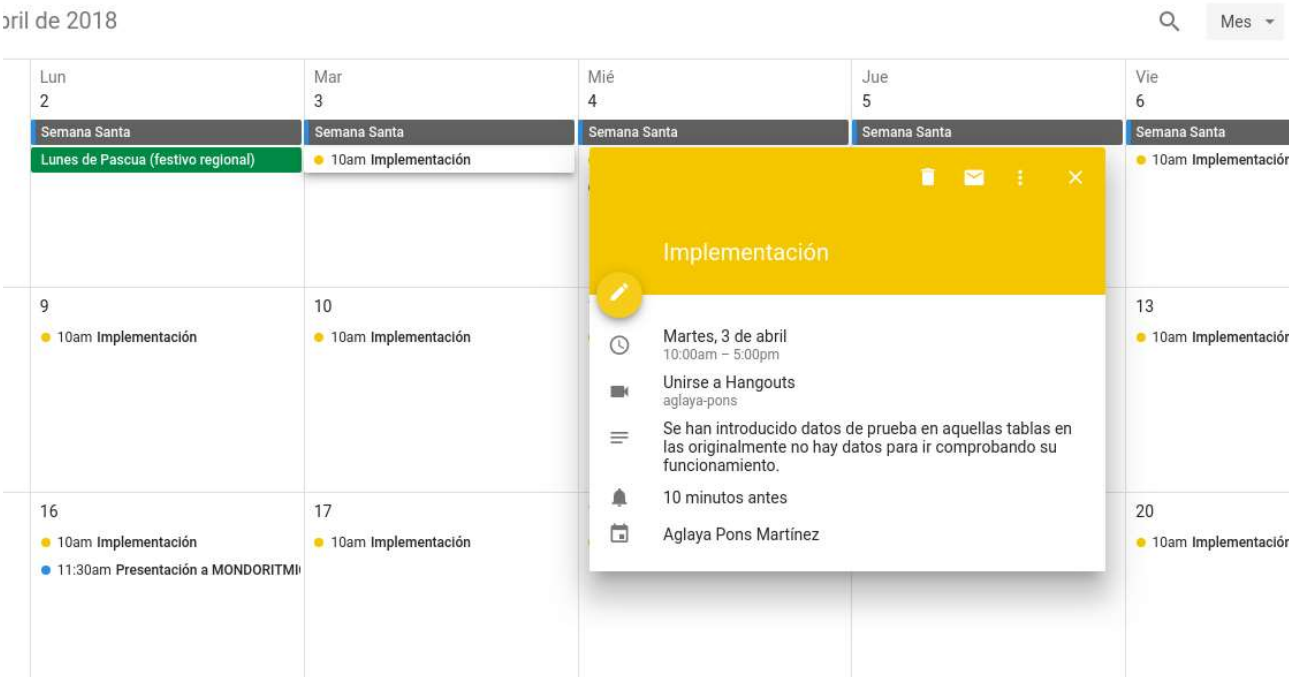

Figura 4: Ejemplo del seguimiento del trabajo realizado a través de Google Calendar.

Como se ha explicado anteriormente, se realizaron reuniones semanales con el supervisor y el compañero que ha implementado la aplicación móvil. En cada una de estas reuniones se establecían objetivos para la siguiente reunión y de esta forma hemos podido tener un seguimiento de lo que hemos hecho. En estas reuniones también se dio prioridad a algunas tareas de las que tenía que realizar, normalmente porque la aplicación móvil requería datos o funcionalidades que todavía no estaban desarrolladas en la aplicación *web*. Por tanto, esas tareas han sido las primeras en realizarse en los días posteriores a la reunión para que el compañero pudiera proseguir con las tareas relacionadas de su proyecto.

El seguimiento de las tareas se ha llevado a cabo mediante **Visual Studio Team Services**, donde también se ha almacenado un repositorio del proyecto. En la Figura 5 se puede observar la lista de tareas programadas en **Visual Studio Team Services** para una duración de 15 días aproximadamente y que coincide con el ejemplo mostrado de **Google Calendar** en la Figura 4.

Order	Work Item Type	Title	State	Story ...	Value Area	Iteration Path
1	User Story	Segunda presentación a MONDORITMIC	Closed		Business	GARLIC\Iteration 2
	Task	Implementar el módulo de mensajes	Active		Business	GARLIC\Iteration 2
	Task	Añadir nuevos atributos a las tablas	Closed		Business	GARLIC\Iteration 2
	Task	Diseñar mockups para las interfaces de usuario	Closed		Business	GARLIC\Iteration 2
	Task	Definir presupuesto	Closed		Business	GARLIC\Iteration 2
	Task	Entrega al cliente	Closed			GARLIC\Iteration 2
2	User Story	Migrar los datos de prueba	Resolved		Business	GARLIC\Iteration 2
	Task	Migrar de Excel a SQL Server	Closed			GARLIC\Iteration 2
	Task	Insertar datos	Closed			GARLIC\Iteration 2

Figura 5: Ejemplo del seguimiento llevado a cabo en Visual Studio Team Services.

En la Figura 5 se muestra el *sprint* o iteración 2 con los objetivos que se propusieron durante las reuniones para ese periodo de tiempo, desglosados por tareas. **Visual Studio Team Services** nos permite llevar un control del trabajo realizado y del trabajo pendiente, pudiendo modificar el estado de las tareas cuando éstas ya han sido completadas. En este ejemplo podemos ver que la tarea "Implementar el módulo de mensajes" no pudo ser completada durante ese *sprint* y por eso está marcada como activa. **Visual Studio Team Services** permite pasar las tareas inacabadas de un *sprint* al siguiente para tener constancia de que siguen activas.

Durante el desarrollo del proyecto se han producido algunos retrasos con respecto a la planificación inicial. La definición del presupuesto ha sufrido un retraso debido a que era necesario incluir unos costos que no se habían tenido en cuenta, como los costos de publicar en Azure, ya que había que incluir dos instancias de bases de datos. De todas formas, este retraso no ha provocado que se retrasaran otras tareas por lo que no se ha tomado ninguna medida al respecto.

Hay otras tareas que se han adelantado por petición del cliente. Esas tareas tienen que ver con el diseño visual de la aplicación, ya que es lo primero que el cliente quería validar. Por esa razón, se adelantó la creación de prototipos de diseño de pantallas. Aún así, este cambio en la planificación tampoco ha provocado retrasos en el resto de tareas.

Capítulo 4

Análisis y diseño del sistema

4.1 Análisis del sistema

4.1.1 Definición de requisitos

En este apartado definimos los requisitos que el cliente quiere que ofrezca el sistema a desarrollar. Estos requisitos se han extraído de los objetivos incluidos en la propuesta del cliente (véase la sección 1.2).

Primero vamos a definir los usuarios que van a interactuar con el sistema. A nuestra aplicación solo pueden tener acceso los miembros del personal, que hemos dividido entre administradores y profesores. Estos usuarios son los que denominamos actores y que definimos a continuación:

- El actor administrador representa a un miembro del personal de la escuela de música que tendrá acceso a la gestión interna del sistema.
- El actor profesor representa a un miembro del personal de la escuela de música que imparte clases.

En la Tabla 1 se definen las necesidades preliminares del sistema, que no solo ayudan a realizar el diagrama de casos de uso que se detalla más adelante sino también a especificar los requisitos.

En la Figura 6 podemos observar el diagrama de casos de uso. Este diagrama pretende documentar el comportamiento del sistema desde el punto de vista del usuario.

Los casos de uso se identifican a partir de las funcionalidades que el sistema debe implementar y que corresponden con las necesidades preliminares de la Tabla 3. Cada uno de estos casos de uso está definido de la Tabla 4 a la Tabla 12.

Actor	Qué quiere hacer	Comunicación con el sistema	
		Proporciona	Recibe
Administrador	Gestionar los alumnos	Parámetros para crear o editar un alumno	La lista de alumnos si la operación es correcta. Un mensaje de error en caso contrario.
	Gestionar el personal	Parámetros para crear o editar un miembro del personal	La lista del personal si la operación es correcta. Un mensaje de error en caso contrario.
	Crear un usuario	Parámetros para crear un usuario del sistema	No recibe nada si la operación es correcta. Un mensaje de error en caso contrario.
	Gestionar clase	Parámetros para crear o modificar una clase	La lista de actividades si la operación es correcta. Un mensaje de error en caso contrario.
	Gestionar asignaturas	Parámetros para crear o modificar una asignatura	La lista de asignaturas si la operación es correcta. Un mensaje de error en caso contrario.
Profesor	Consultar las clases que imparte		La lista de clases que imparte.
	Modificar perfil	Parámetros para modificar los datos existentes en la base de datos sobre sí mismo.	Los datos personales del individuo. Un mensaje de error si la operación no es correcta.
	Enviar mensaje a un alumno	Parámetros necesarios para el envío de mensajes	La lista de mensajes enviados por él mismo si la operación es correcta. Un mensaje de error en caso contrario.
	Consultar mensajes enviados.		La lista de mensajes enviados por él mismo.

Tabla 3: Definición de necesidades preliminares.

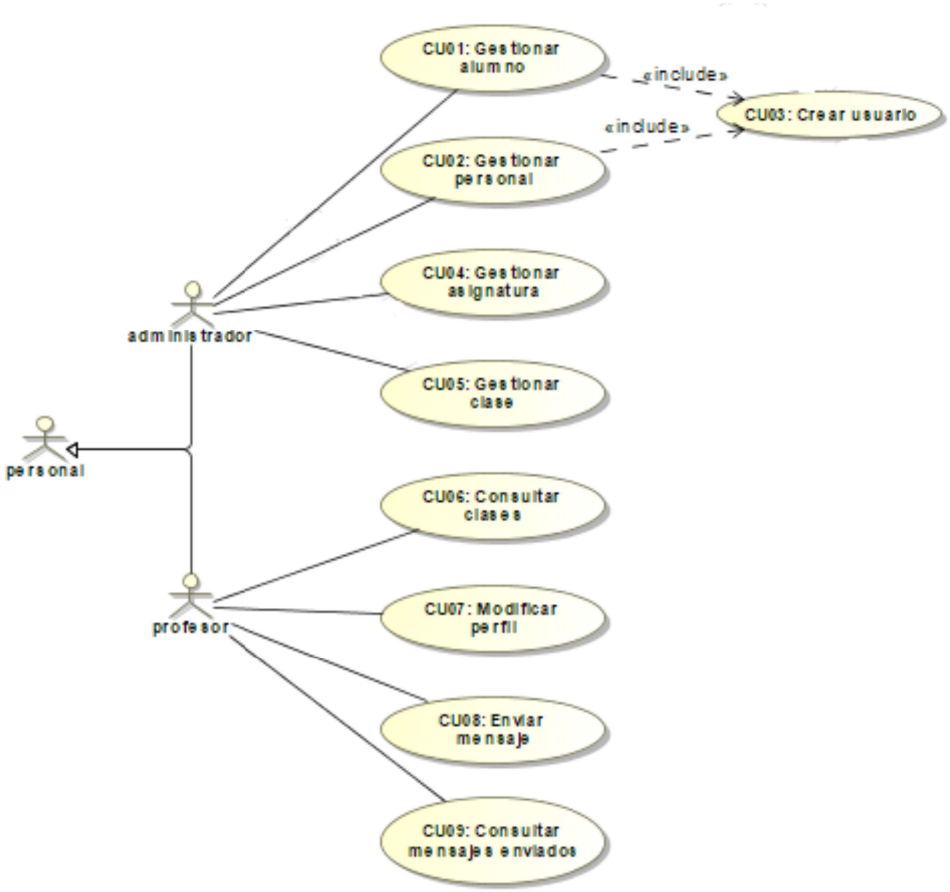

Figura 6: Diagrama de casos de uso del proyecto.

Gestionar alumno	
Identificador: CU01 Nombre: Gestionar alumno Autor: Aglaya Pons Martínez Fuente: Escuela de música	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: El administrador puede añadir nuevos alumnos y modificar los existentes.	
Secuencia de pasos habitual: 1. El administrador consulta los alumnos registrados en el sistema. 2. El administrador: (a) Selecciona el botón “crear” para añadir un nuevo alumno. (b) Selecciona la fila del alumno que quiere modificar. 3. El administrador introduce los datos necesarios. Excepción: Se muestra un mensaje de error si hay algún dato incorrecto o si no se ha introducido un valor en un campo obligatorio. Condición previa: El administrador debe estar registrado en el sistema.	

Tabla 4: Definición del caso de uso CU01, gestionar alumno.

Gestionar personal	
Identificador: CU02 Nombre: Gestionar personal Autor: Aglaya Pons Martínez Fuente: Escuela de música	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: El administrador puede añadir nuevos miembros del personal y modificar los existentes.	
Secuencia de pasos habitual: 1. El administrador consulta los miembros del personal registrados en el sistema. 2. El administrador: (a) Selecciona el botón “crear” para añadir un nuevo miembro. (b) Selecciona la fila del miembro que quiere modificar. 3. El administrador introduce los datos necesarios. Excepción: Se muestra un mensaje de error si hay algún dato incorrecto o si no se ha introducido un valor en un campo obligatorio. Condición previa: El administrador debe estar registrado en el sistema.	

Tabla 5: Definición del caso de uso CU02, gestionar personal.

Crear usuario	
Identificador: CU03 Nombre: Crear usuario Autor: Aglaya Pons Martínez Fuente: Escuela de música	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: El administrador puede crear un usuario para otorgarle acceso al sistema.	
Secuencia de pasos habitual: 1. El administrador consulta: (a) los alumnos registrados en el sistema. (b) Los miembros del personal registrados en el sistema. 2. El administrador selecciona al alumno o miembro del personal a quien quiere crearle un usuario. 3. El administrador introduce los datos necesarios. Excepción: Se muestra un mensaje de error si hay algún dato incorrecto o si no se ha introducido un valor en un campo obligatorio. Condición previa: El administrador debe estar registrado en el sistema. El alumno o miembro del personal al que se le quiere asignar un usuario, debe existir en la base de datos.	
Comentario: Los alumnos no tienen que acceder al sistema pero necesitan tener un usuario ya que acceden a sus datos a través de la aplicación móvil que se realiza en otro proyecto.	

Tabla 6: Definición del caso de uso CU03, crear usuario.

Gestionar asignatura	
Identificador: CU04 Nombre: Gestionar asignatura Autor: Aglaya Pons Martínez Fuente: Escuela de música	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: El administrador puede añadir nuevas asignaturas y modificar las existentes.	
Secuencia de pasos habitual: 1. El administrador consulta las asignaturas registradas en el sistema. 2. El administrador: (a) Selecciona el botón “crear” para añadir una nueva asignatura. (b) Selecciona la fila de la asignatura que quiere modificar. 3. El administrador introduce los datos necesarios. Excepción: Se muestra un mensaje de error si hay algún dato incorrecto o si no se ha introducido un valor en un campo obligatorio. Condición previa: El administrador debe estar registrado en el sistema.	

Tabla 7: Definición del caso de uso CU04, gestionar asignatura.

Gestionar clase	
Identificador: CU05 Nombre: Gestionar clase Autor: Aglaya Pons Martínez Fuente: Escuela de música	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: El administrador puede añadir nuevas clases y modificar las existentes.	
Secuencia de pasos habitual: 1. El administrador consulta las clases registradas en el sistema. 2. El administrador: (a) Selecciona el botón “crear” para añadir una nueva clase. (b) Selecciona la fila de la clase que quiere modificar. 3. El administrador introduce los datos necesarios. Excepción: Se muestra un mensaje de error si hay algún dato incorrecto o si no se ha introducido un valor en un campo obligatorio. Condición previa: El administrador debe estar registrado en el sistema. Debe existir en la base de datos los datos del alumno que quiere asistir a esa clase, la asignatura de la clase y el profesor que debe impartirla.	

Tabla 8: Definición del caso de uso CU05, gestionar clase.

Consultar clases	
Identificador: CU06 Nombre: Consultar clases Autor: Aglaya Pons Martínez Fuente: Escuela de música	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: El profesor puede consultar las clases que imparte.	
Secuencia de pasos habitual: 1. El profesor consulta las clases registradas en el sistema. 2. El sistema filtra las clases para mostrar solo aquellas que imparte. Excepción: No hay. Condición previa: El profesor debe estar registrado en el sistema.	

Tabla 9: Definición del caso de uso CU06, consultar clases.

Modificar perfil	
Identificador: CU07 Nombre: Modificar perfil Autor: Aglaya Pons Martínez Fuente: Escuela de música	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: El profesor puede modificar sus datos personales.	
Secuencia de pasos habitual: 1. El profesor consulta sus datos personales. 2. El profesor selecciona sus datos para modificarlos. 3. El profesor introduce los datos a modificar. Excepción: Se muestra un mensaje de error si hay algún dato incorrecto o si no se ha introducido un valor en un campo obligatorio. Condición previa: El profesor debe estar registrado en el sistema.	

Tabla 10: Definición del caso de uso CU07, modificar perfil.

Enviar mensaje	
Identificador: CU08 Nombre: Enviar mensaje Autor: Aglaya Pons Martínez Fuente: Escuela de música	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: El profesor puede enviar un mensaje a un alumno.	
Secuencia de pasos habitual: 1. El profesor selecciona el botón “crear” para enviar un mensaje. 2. El sistema muestra la lista de alumnos a los que puede enviar un mensaje. 3. El profesor selecciona al alumno al que quiere enviarle un mensaje. 4. El profesor introduce los datos necesarios. Excepción: Se muestra un mensaje de error si hay algún dato incorrecto o si no se ha introducido un valor en un campo obligatorio. Condición previa: El profesor debe estar registrado en el sistema.	

Tabla 11: Definición del caso de uso CU08, enviar mensaje.

Consultar mensajes enviados	
Identificador: CU09 Nombre: Consultar mensajes enviados Autor: Aglaya Pons Martínez Fuente: Escuela de música	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: El administrador puede consultar los mensajes que ha enviado.	
Secuencia de pasos habitual: 1. El profesor consulta los mensajes enviados. 2. El sistema filtra los mensajes para mostrar solo aquellos enviados por el profesor. 3. El profesor selecciona la fila del mensaje que quiere consultar. 4. El sistema muestra los datos del mensaje seleccionado. Excepción: No hay. Condición previa: El profesor debe estar registrado en el sistema.	

Tabla 12: Definición del caso de uso CU09, consultar mensajes enviados.

A continuación, se definen los requisitos de datos necesarios para el funcionamiento del sistema a desarrollar.

Alumno	
Identificador: RD01 Nombre: Alumno Autor: Aglaya Pons Martínez Fuente: Administrador	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Tipo: Entrada Datos específicos a almacenar: Apellido y nombre, fecha de matrícula, autorización de derechos de imagen, notas (observaciones), nombre del padre o la madre, teléfono, e-mail, cuenta bancaria, fecha de nacimiento, si el alumno tiene alguna enfermedad o alergia, si recibe alguna medicación y foto para el perfil.	
Detalles: La autorización de derechos de imagen es un booleano. La cuenta bancaria no es obligatoria ya que se puede abonar la cuota en efectivo.	
Comentarios: Un ejemplo de los datos es el siguiente: Amador Juan, 12/05/2018, Sí, No tiene instrumento, Amador Javier, 632145789, juanam@gmail.com, ES1234567891234567891234, 03/09/2000.	

Tabla 13: Definición del requisito de dato RD01, alumno.

Personal	
Identificador: RD02 Nombre: Personal Autor: Aglaya Pons Martínez Fuente: Administrador	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Tipo: Entrada Datos específicos a almacenar: Apellido y nombre, teléfono, e-mail, cargo.	
Detalles:	
Comentarios: Un ejemplo de los datos es el siguiente: Fuentes Macarena, 698457123, mfuentes@gmail.com, profesor.	

Tabla 14: Definición del requisito de dato RD02, personal.

Usuario	
Identificador: RD03 Nombre: Usuario Autor: Aglaya Pons Martínez Fuente: Administrador	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Tipo: Entrada Datos específicos a almacenar: nombre de usuario, contraseña	
Detalles:	
Comentarios: Un ejemplo de los datos es el siguiente: mfuentes, pass123	

Tabla 15: Definición del requisito de dato RD03, usuario.

Asignatura	
Identificador: RD04 Nombre: Asignatura Autor: Aglaya Pons Martínez Fuente: Administrador	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Tipo: Entrada Datos específicos a almacenar: categoría, nombre, descripción.	
Detalles:	
Comentarios: Un ejemplo de los datos es el siguiente: cuerda, guitarra eléctrica, Asignatura de guitarra eléctrica desde el nivel básico al avanzado.	

Tabla 16: Definición del requisito de dato RD04, asignatura.

Clase	
Identificador: RD05 Nombre: Clase Autor: Aglaya Pons Martínez Fuente: Administrador	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Tipo: Entrada Datos específicos a almacenar: nombre del alumno, nombre del profesor, nombre de la asignatura, fecha de inicio, fecha fin, hora, duración, aula y grupo de clase si es una clase grupal.	
Detalles: Las clases pueden ser tanto individuales como grupales pero, aunque la clase sea grupal, se debe realizar una entrada por cada alumno que asiste a ella.	
Comentarios: Un ejemplo de los datos es el siguiente: Amador Juan, Fuentes Macarena, Guitarra eléctrica, 01/02/2018, 11/06/2018, 17:00, 2, A103.	

Tabla 17: Definición del requisito de dato RD05, clase.

Mensaje	
Identificador: RD06 Nombre: Mensaje Autor: Aglaya Pons Martínez Fuente: Profesor	Fecha creación: 13/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Tipo: Entrada Datos específicos a almacenar: nombre del alumno, nombre del profesor, asunto, mensaje, fecha y hora de envío.	
Detalles: La fecha y hora de envía es creada automáticamente por el sistema.	
Comentarios: Un ejemplo de los datos es el siguiente: Amador Juan, Fuentes Macarena, Para la próxima clase, Acuérdate de traer la partitura de Medianoche, 26/03/2018.	

Tabla 18: Definición del requisito de dato RD06, mensaje.

4.1.2 Diagrama de clases

En esta sección se muestra parte del análisis del sistema con un diagrama de clases basado en los requisitos de datos. La Figura 7 muestra el diagrama de clases del sistema con sus atributos y operaciones. La documentación de este diagrama se encuentra en el anexo A.

Este diagrama se ha extraído a partir de la información obtenida con los requisitos de datos y los casos de uso.

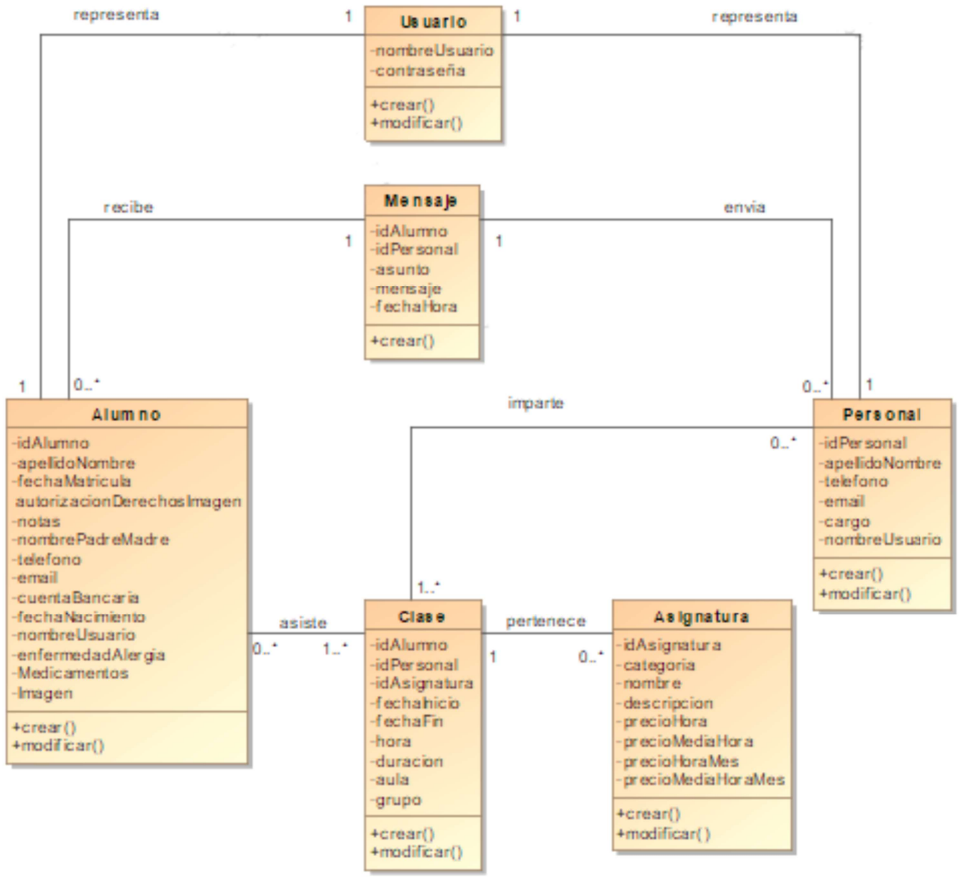

Figura 7: Diagrama de clases del proyecto.

4.2 Diseño de la arquitectura del sistema

En esta sección se incluye la arquitectura del sistema y el diseño de la base de datos.

4.2.1 Arquitectura del sistema

La arquitectura que debe seguir nuestra aplicación *web* para ofrecer las funcionalidades que el cliente quiere es la arquitectura cliente-servidor [15]. Un cliente realiza peticiones a un servidor que le ofrece respuestas.

En nuestro caso, el cliente puede ser un administrador o un profesor de la escuela de música que realiza peticiones al servidor a través de la aplicación *web*. El servidor *web* que estamos utilizando es el que ofrece **Azure** y la base de datos a la que se conecta la proporciona **SQL Server.**

La arquitectura definida está representada en la Figura 8.

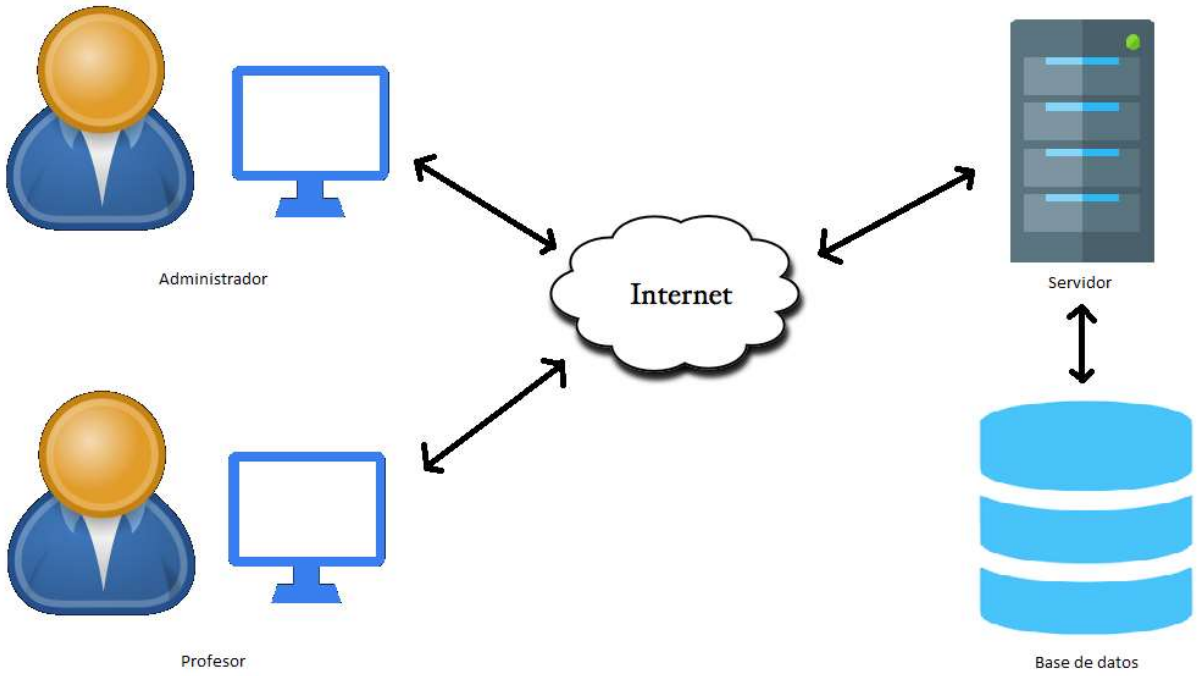

Figura 8: Propuesta tecnológica.

4.2.2 Diseño de la base de datos

En esta sección detallamos el diseño conceptual, lógico y físico de la base de datos que hemos creado para poder mantener los datos que el sistema debe manejar.

Es importante explicar que la plantilla **Serenity** que hemos utilizado para la creación de la aplicación *web*, ya crea por defecto una base de datos para la administración de usuarios de la aplicación. Por tanto, en lugar de crear una tabla para los usuarios, simplemente hay que relacionar las tablas Alumno y Personal de la base de datos que hemos creado con la base de datos que se crea por defecto para que puedan acceder al sistema. Aunque los alumnos no puedan tener acceso a esta aplicación web, necesitan tener las credenciales para acceder a la aplicación móvil, ya que ambas utilizan la misma base de datos para conceder oportunamente el acceso al sistema. Por tanto, todas las personas físicas que tengan una entrada en la base de datos, deben tener un usuario y una contraseña para poder tener acceso a sus datos.

4.2.2.1 Diseño conceptual

En la Figura 9 se presenta el esquema conceptual realizado a partir de los requisitos presentados en la sección 4.1. Este esquema describe a alto nivel la estructura de la base de datos y muestra la información que debe contener.

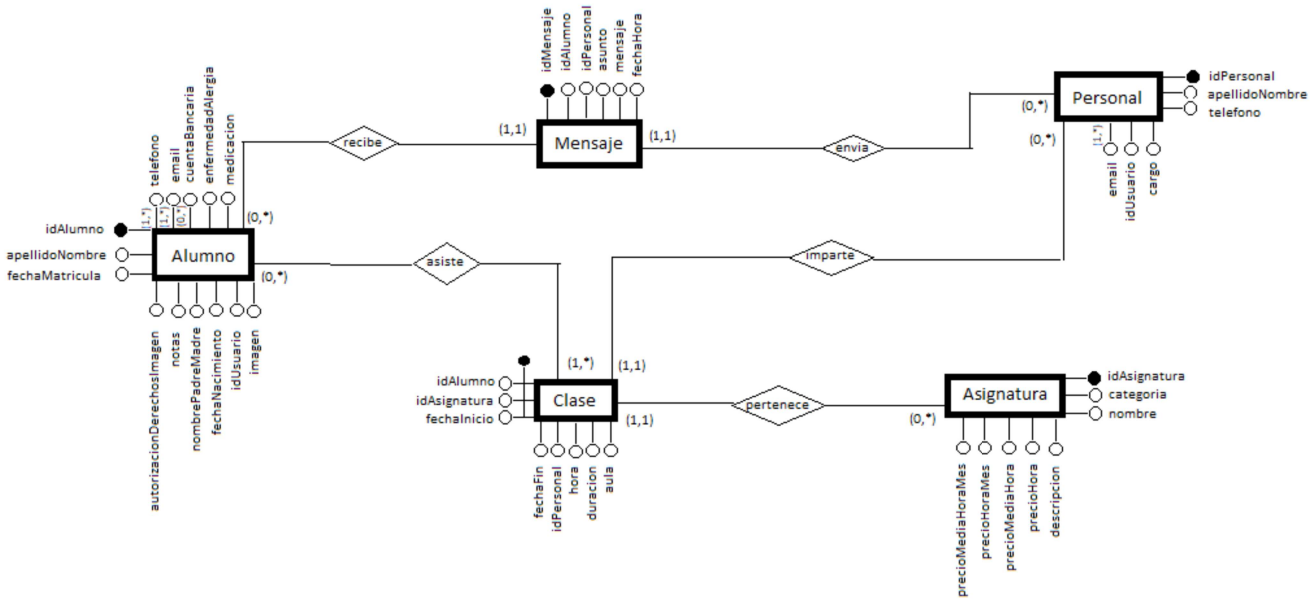

Figura 9: Diseño conceptual preliminar de la base de datos del proyecto.

4.2.2.2 Diseño lógico

A raíz del diseño conceptual, construimos el diseño lógico que se puede observar en la Figura 10. En este diseño se definen las tablas que forman parte de la base datos y también sus relaciones, claves y atributos.

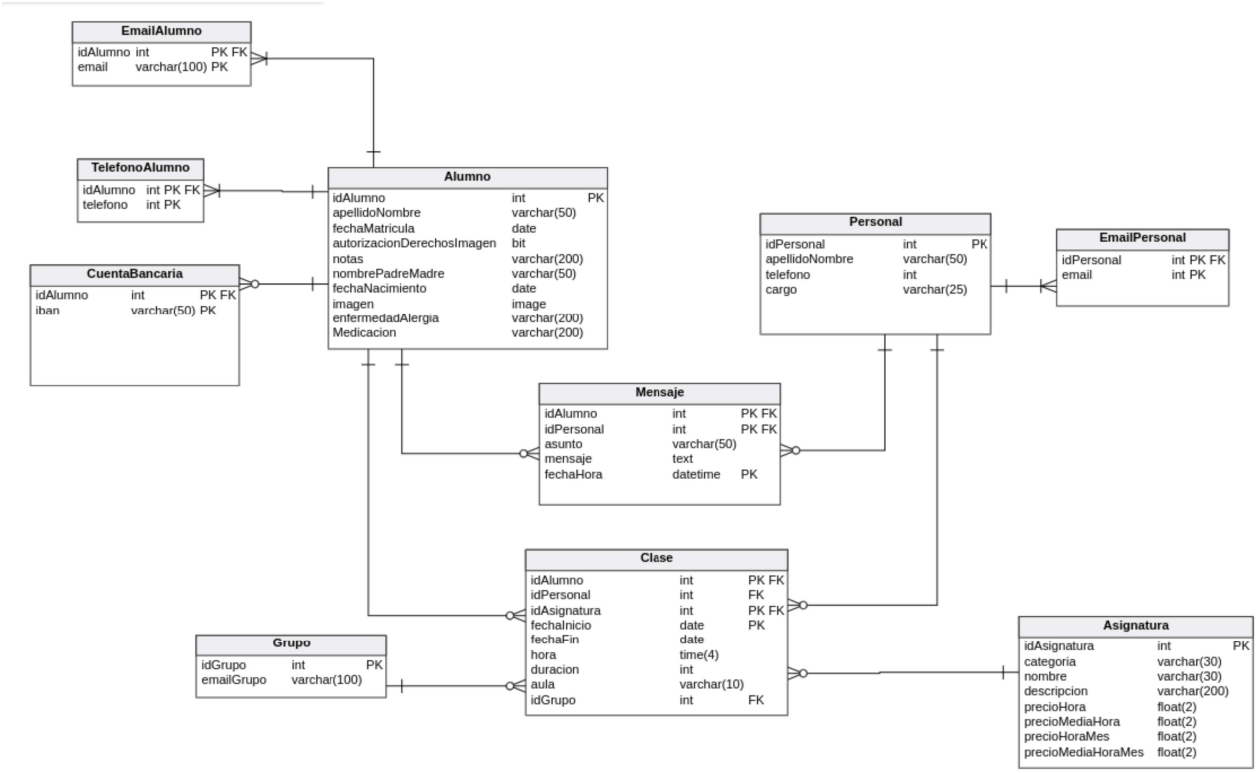

Figura 10: Diseño lógico de la base de datos del proyecto.

4.2.2.3 Diseño físico

En este apartado se incluyen las sentencias CREATE TABLE necesarias para la creación de las tablas con todos sus componentes en la base de datos. Debido a su extensión, estas sentencias se encuentran en el Anexo B.

4.3 Diseño de la interfaz

4.3.1 Diseño visual y definición de estilos

Para el diseño visual de la aplicación, la escuela de música nos ha proporcionado su color corporativo que es el naranja *hot cinnamon*. En hexadecimal #D1701D y RGB: 209, 112, 29. El color se puede observar en la Figura 11. Este color se combina con tonos blancos para una mejor visualización.

Figura 11: Color corporativo de la escuela de música.

Para todos los textos de la aplicación se emplea la fuente **Open Sans** y un tamaño de letra de 13 píxeles que proporciona por defecto la plantilla **Serenity** que hemos utilizado.

La escuela de música también ha proporcionado su logotipo e imágenes para que se muestren de fondo en la pantalla de inicio.

4.3.2 Prototipos

A continuación, mostramos algunos de los prototipos para el diseño de pantallas que se ha nombrado anteriormente con el fin de que el lector se pueda hacer una idea. Estos diseños era lo que se mostraba al cliente en las reuniones cuando todavía no existía un producto funcional. En el pie de cada imagen se especifica a qué pantalla o elemento representa. Estos prototipos han sido diseñados con la aplicación *web* **NinjaMock** que se ha nombrado en el capítulo 2. El contenido del fichero CSS del estilo se puede encontrar en el Anexo C.

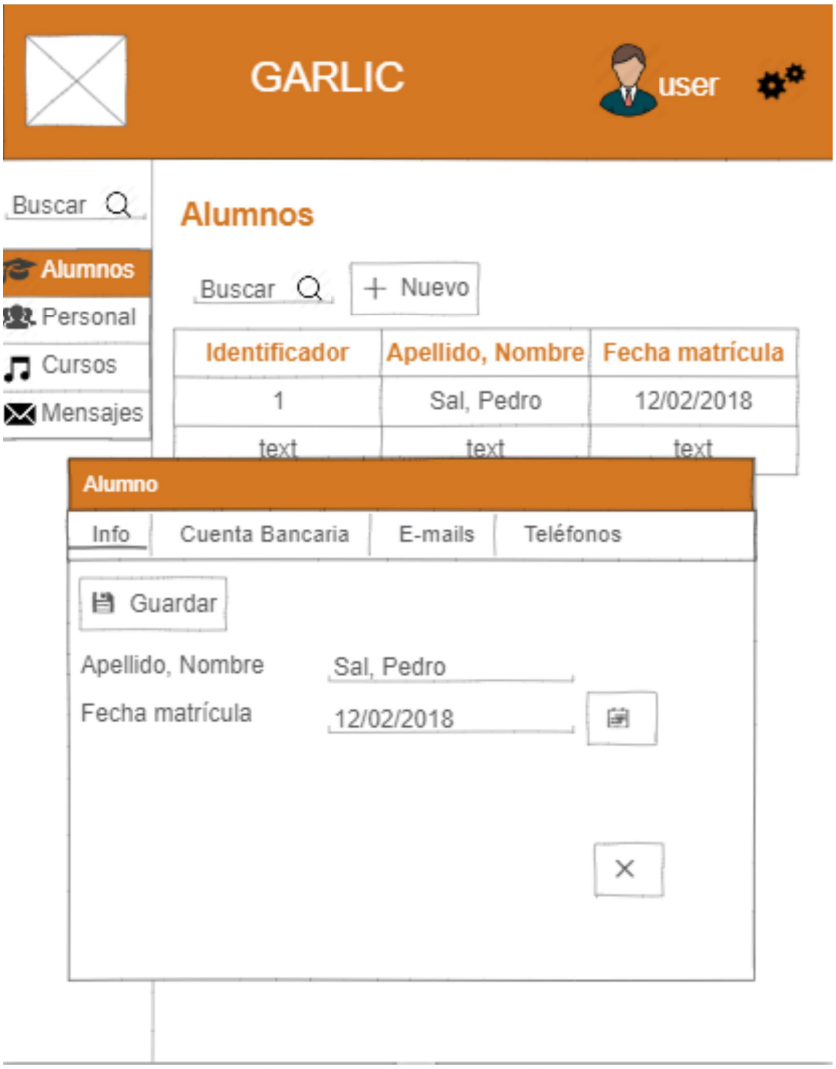

Figura 12: Diseño de pantalla del formulario para añadir un nuevo alumno.

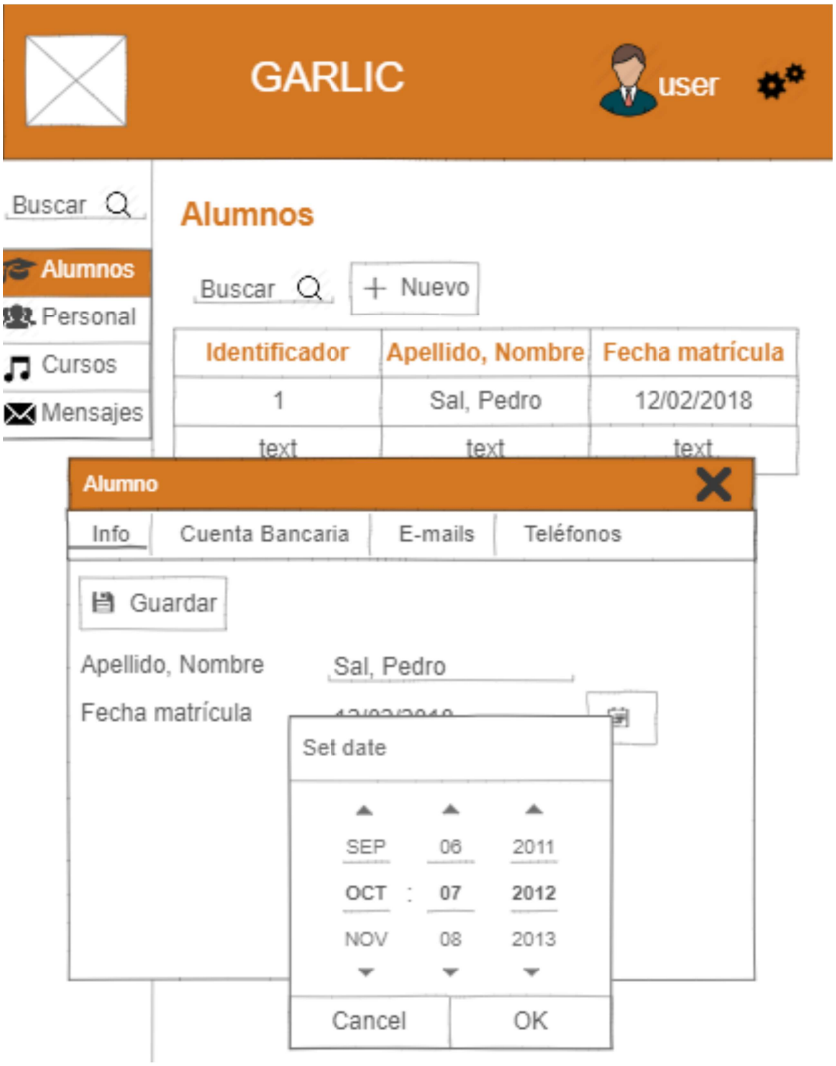

Figura 13: Diseño para el selector de fecha.

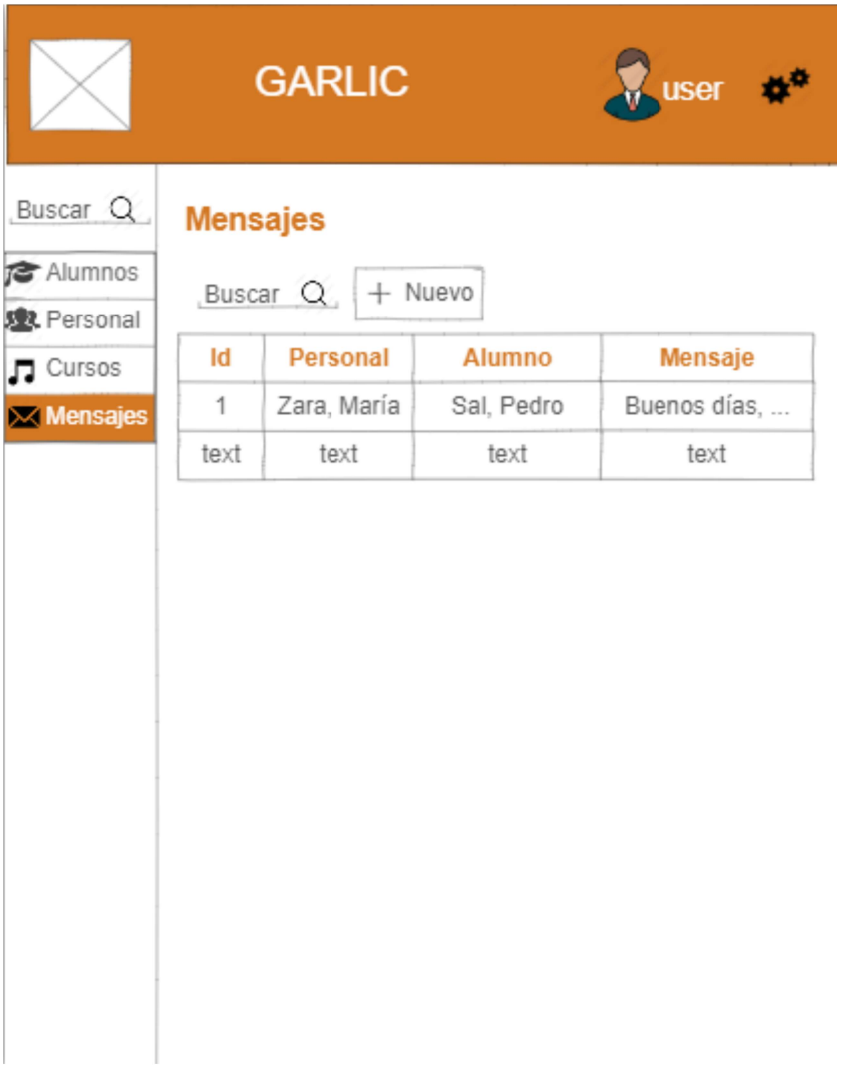

Figura 14: Diseño de la pantalla que lista los mensajes enviados. Si el usuario es administrador, verá todos los mensajes, mientras que si es profesor, solo verá aquellos que él mismo ha enviado.

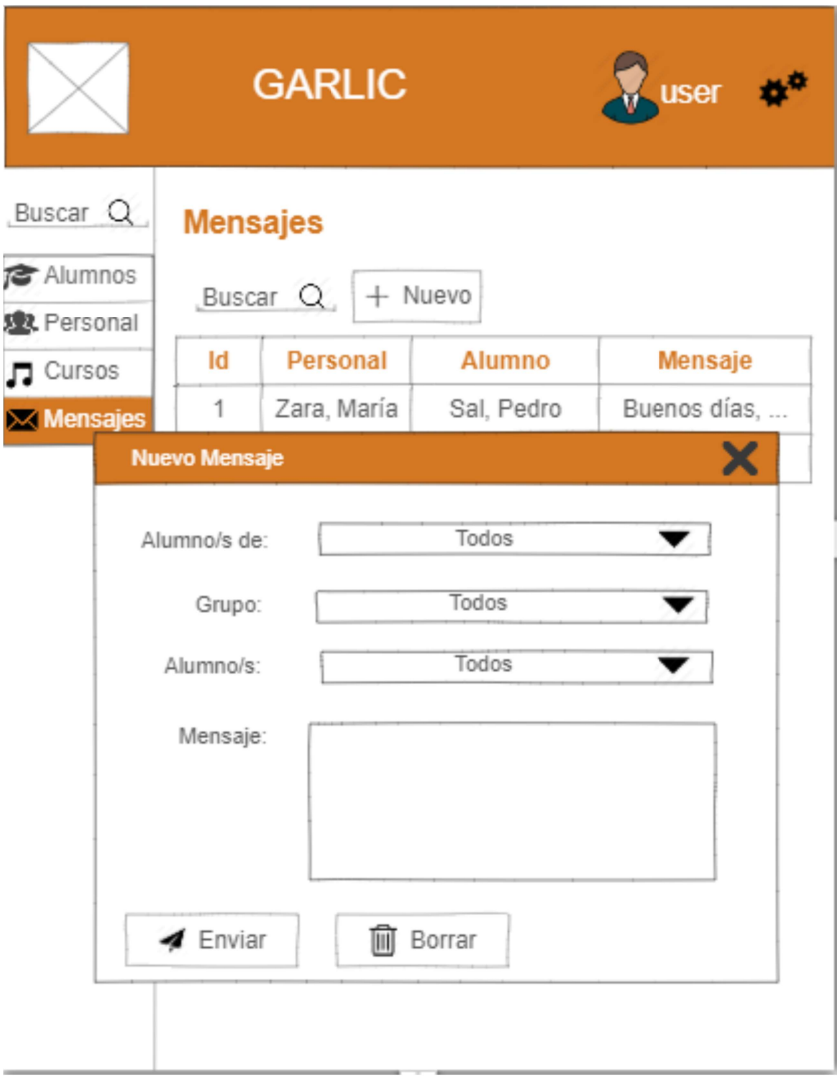

Figura 15: Diseño de pantalla de creación de mensaje.

Capítulo 5

Implementación

5.1 Detalles de implementación

La implementación del proyecto ha empezado con la creación de una primera versión de la base de datos en **SQL Server**. Los nombres de las tablas y los atributos que deben existir en la base de datos para almacenar la información que se debe mantener han sido extraídos de las hojas **Excel** que nos ha proporcionado el cliente, que es la herramienta que ellos han utilizado para mantener la información hasta el momento pero con los datos encriptados. En la Figura 16 se muestra un ejemplo de una de esas hojas.

	A	B	C	D	E	F	G
1	Marca Tempora	1er Apellido y Nc	Teléfono	Teléfono 2	Email	Email 2	Fecha Matrícula
2	8/02/2018 17:16:	XXXa, Pepa	XXX65607	XXX	XXX@gmail.com	XXX	12/02/2018
3	6/02/2018 13:13:	XXXALFONSO	XXX04565	XXX	XXX@bbg-aboga	XXX	6/02/2018
4	5/02/2018 21:55:	XXXe Jaume	XXX42724	XXX08338	XXX@yahoo.es	XXX@yahoo.es	5/02/2018
5	2/02/2018 7:56:4	XXXIG, ANA	XXX27034	XXX67469	XXX@gmail.com	XXX	8/02/2018
6	31/01/2018 18:37	XXXi Cesar	XXX69648	XXX	XXX@hotmail.co	XXX	1/02/2018
7	24/01/2018 19:5	XXX CARMEN	XXX63067	XXX	XXX@gmail.com	XXX	24/01/2018
8	22/01/2018 18:2	XXXa, Alex	XXX49477	XXX	XXX@gmail.com	XXX	22/01/2018
9	19/01/2018 11:47	XXX Marcos	XXX32464	XXX	XXX@gmail.com	XXX	15/01/2018
10	15/01/2018 19:0	XXXRicardo	XXX90805	XXX16014	XXX@hotmail.co	XXX@hotmail.co	17/01/2018
11	12/01/2018 18:4	XXX Blanca	XXX32705	XXX95223	XXX@gmail.com	XXX	12/01/2018
12	12/01/2018 9:15:	XXXut Musa	XXX86485	XXX	XXX@yahoo.cor	XXX	12/01/2018
13	11/01/2018 16:2	XXXía Anna	XXX51859	XXX	XXX@gmail.com	XXX	12/01/2018
14	8/01/2018 19:39:	XXXo Avila	XXX15149	XXX	XXX@hotmail.co	XXX	8/01/2018
15	8/01/2018 17:44:	XXXiz Pepa	XXX13941	XXX	XXX@gmail.com	XXX	17/01/2018
16	27/12/2017 19:47	XXX, Jaime	XXX35913	XXX59831	XXX@gmail.com	XXX	27/12/2017
17	20/12/2017 16:1	XXX MARIA	XXX72345	XXX72351	XXX@benet.es	XXX	1/01/2018
18	19/12/2017 17:2	XXXo, Siro	XXX09607	XXX	XXX@gmail.com	XXX	1/10/2017
19	13/12/2017 20:1	XXXon Roca	XXX76705	XXX21525	XXX@gmail.com	XXX	9/01/2018
20	28/11/2017 16:07	XXXez Laia	XXX36077	XXX80298	XXX@gmail.com	XXX	1/12/2017
21	28/11/2017 5:57:	XXXAntonio	XXX35830	XXX	XXX@gmail.com	XXX	28/11/2017
22	21/11/2017 18:34	XXX, Elisa	XXX62059	XXX08706	XXX@gmail.com	XXX@gmail.com	21/11/2017
23	20/11/2017 17:28	XXX, BALMA	XXX37833	XXX69684	XXX@gmail.com	XXX	7/11/2017
24	14/11/2017 9:46:	XXX TAMAYO	XXX94995	XXX	XXX@hotmail.co	XXX	14/11/2017
25	14/11/2017 9:42:	XXXYO TENA	XXX60142	XXX	XXX@gmail.com	XXX	14/11/2017
26	7/11/2017 18:27:	XXXatricia	XXX35618	XXX	XXX@hotmail.co	XXX	9/11/2017
27	6/11/2017 20:02:	XXX Adrian	XXX24057	XXX85569	XXX@gmail.com	XXX@yahoo.es	6/11/2017
28	6/11/2017 17:39:	XXXMartina	XXX23941	XXX57295	XXX@gmail.com	XXX	6/11/2017

ALUMNOS | PERSONAL | TALLERES_CURSOS | CURSOS_EXTENSION

Figura 16: Ejemplo de hoja Excel proporcionada por el cliente.

Las tablas y atributos que se han almacenado en la base de datos inicialmente se muestran en la Figura 17. Esta figura contiene el diseño lógico inicial que no se ha incluído en el apartado 4.2.2.2.

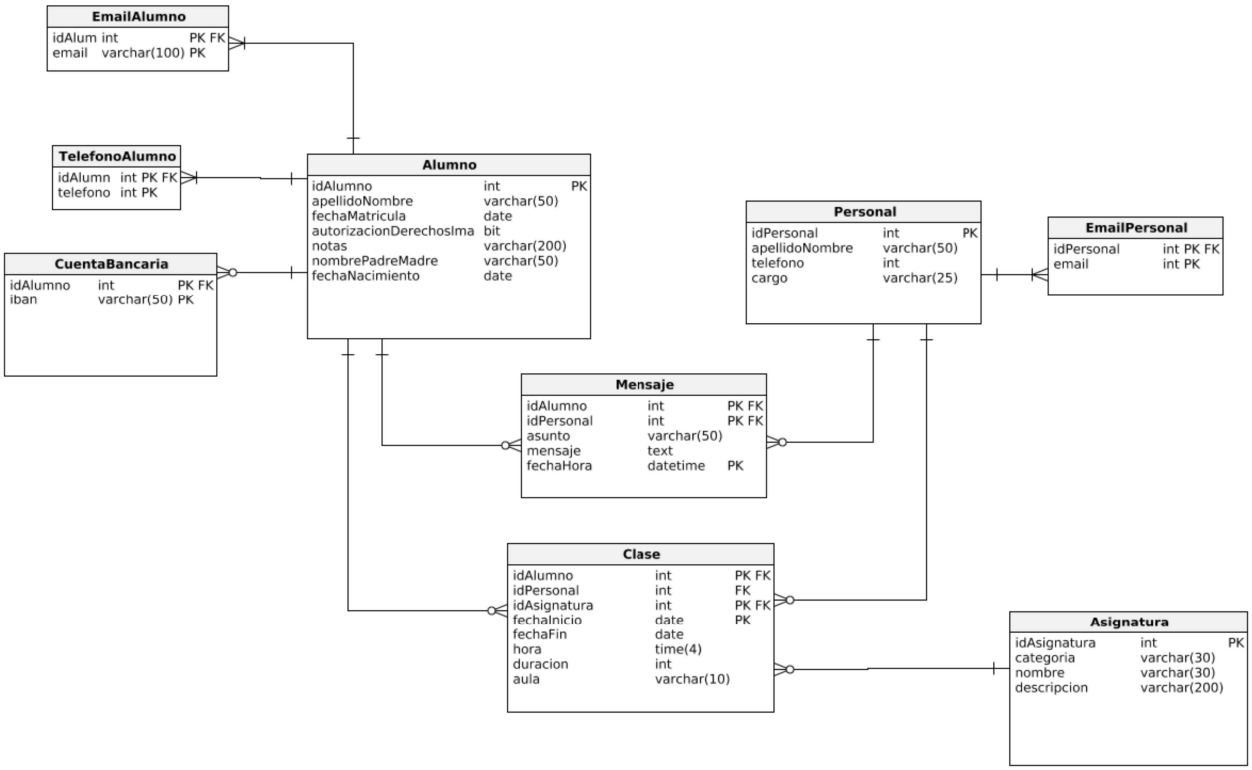

Figura 17: Diseño lógico inicial de la base de datos de la aplicación.

Como se ha mencionado anteriormente, la aplicación web ha sido desarrollada utilizando **Visual Studio** y la plantilla **Serenity**. Para conocer un poco más **Serenity** vamos a listar las características que ofrece y que están detalladas en su guía [16]. Estas características son las siguientes:
Un modelo de aplicación web basado en un servicio modular.

- Generador de código para producir servicios iniciales, código de la interfaz de usuario para una tabla SQL.
- Generación de código basado en T4 en el servidor para referenciar *script widgets* con *intellisense*. Validación del tiempo de compilación.
- Generación de código basado en T4 para proporcionar tiempo de compilación de prevención de errores tipográficos e *intellisense* mientras llama a servicios *AJAX* desde un *script*.
- Sistema de definición de formulario basado en atributos (prepara la interfaz de usuario en el lado servidor con una simple clase en C#).
- Enlace de datos continuo y automático a través de definiciones de formulario (formulario <-> entidad <-> servicio).
- Ayuda para el almacenamiento de datos en caché.
- Validación de caché automática.

- Sistema de configuración.
- Inicio de sesión simple.
- Informes. Solo proveen información, no tienen dependencia con el renderizado.
- *Script bundling*, minificación y versionado de contenido.
- Constructor fluído de SQL (SELECT/INSERT/UPDATE/DELETE)
- Micro ORM.
- H*andlers* customizables para REST como servicios que trabaja reutilizando información en clases de entidad y realiza validaciones automáticas.
- Menú de navegación basado en atributos.
- Localización de interfaz de usuario. Guarda la localización de textos en ficheros *json*, en la base de datos…
- Localización de datos. Utiliza un mecanismo de extensión de tabla que ayuda a localizar incluso datos introducidos por usuario, como tablas *lookup*.
- Sistema *script widget*. Inspirado por jQueryUI pero más adecuado para código C#.
- Validación del lado del cliente y del servidor. Basado en el *plugin* de validación de jQuery pero con dependencia abstracta.
- Registro de auditoría.
- Sistema para pruebas de integración basado en datos.
- *Scripts* dinámicos.
- Plantillas de *script*.

La plantilla **Serenity** ofrece un estilo adecuado para una aplicación web de gestión empresarial y también ejemplos de cómo adaptar la plantilla para ofrecer las funcionalidades deseadas.

Después de tener la primera versión de la base de datos y antes de comenzar a desarrollar la aplicación, se ha creado el proyecto en **Visual Studio** y se ha configurado su acceso a la base de datos, teniendo SQL como origen de datos. Para realizar esa configuración hay que seleccionar el servidor en el que se encuentra la base de datos a la que interesa acceder y el modo de autenticar al mismo. En la Figura 18 se puede observar una de las pantallas de la configuración.

Tras realizar la configuración, se ha compilado y ejecutado la aplicación base que crea **Serenity**. De esta forma se ha creado la base de datos de administración de usuarios que crea por defecto la plantilla y se ha podido visualizar la aplicación base. En la Figura 19 se muestran los ficheros de la plantilla que han creado la base de datos por defecto y la Figura 20 muestra parte del código de uno de esos ficheros.

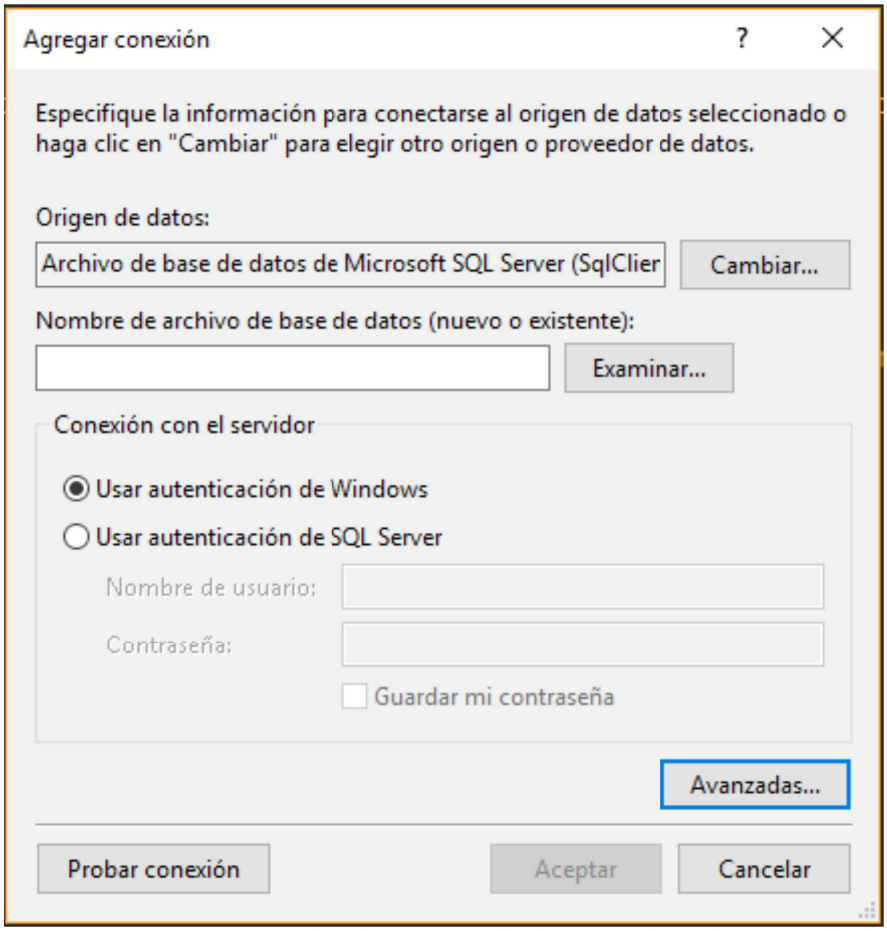

Figura 18: Configuración del origen de datos de la aplicación de Visual Studio.

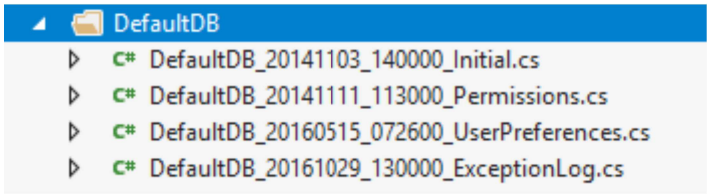

Figura 19: Ficheros de la plantilla Serenity que han creado la base de datos por defecto.

```
public class DefaultDB_20141111_113000_Permissions : AutoReversingMigration
{
    public override void Up()
    {
        this.CreateTableWithId64("UserPermissions", "UserPermissionId", s => s
            .WithColumn("UserId").AsInt32().NotNullable()
                .ForeignKey("FK_UserPermissions_UserId", "Users", "UserId")
            .WithColumn("PermissionKey").AsString(100).NotNullable()
            .WithColumn("Granted").AsBoolean().NotNullable().WithDefaultValue(true));

        Create.Index("UQ_UserPerm_UserId_PermKey")
            .OnTable("UserPermissions")
            .OnColumn("UserId").Ascending()
            .OnColumn("PermissionKey").Ascending()
            .WithOptions().Unique();

        this.CreateTableWithId32("Roles", "RoleId", s => s
            .WithColumn("RoleName").AsString(100).NotNullable());

        this.CreateTableWithId64("RolePermissions", "RolePermissionId", s => s
            .WithColumn("RoleId").AsInt32().NotNullable()
                .ForeignKey("FK_RolePermissions_RoleId", "Roles", "RoleId")
            .WithColumn("PermissionKey").AsString(100).NotNullable());

        Create.Index("UQ_RolePerm_RoleId_PermKey")
            .OnTable("RolePermissions")
            .OnColumn("RoleId").Ascending()
            .OnColumn("PermissionKey").Ascending()
            .WithOptions().Unique();

        this.CreateTableWithId64("UserRoles", "UserRoleId", s => s
            .WithColumn("UserId").AsInt32().NotNullable()
                .ForeignKey("FK_UserRoles_UserId", "Users", "UserId")
            .WithColumn("RoleId").AsInt32().NotNullable()
                .ForeignKey("FK_UserRoles_RoleId", "Roles", "RoleId"));
```

Figura 20: Parte del código de creación de las tablas de la base de datos que ha creado Serenity.

Para que las tablas de nuestra base de datos se muestren también en la aplicación *web*, hemos utilizado el generador de código *Sergen* que nos ofrece la plantilla. Ejecutando *Sergen* desde la terminal, hemos insertado las tablas de la base de datos en el proyecto de **Visual Studio** una a una, creando automáticamente los ficheros necesarios para poder visualizar los datos que tenemos almacenados. Para insertar las tablas hay que ejecutar el código de la Figura 21, seleccionar la base de datos a la que se quiere conectar y después seleccionar la tabla que se quiere insertar. En este punto todavía no había datos que mostrar, por lo que simplemente se ha podido observar, al ejecutar la aplicación, que las tablas con sus atributos se visualizan correctamente.

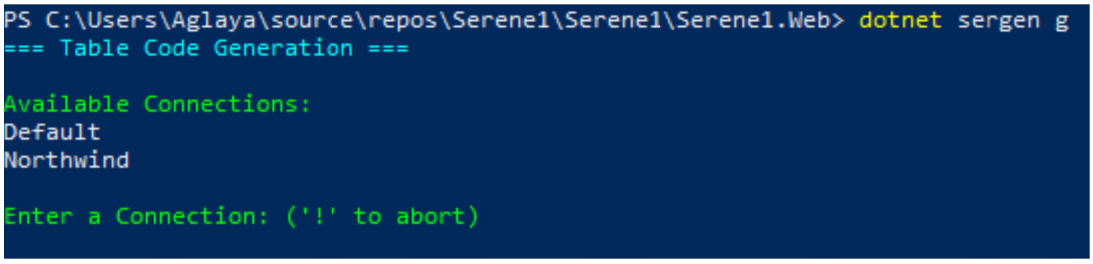

Figura 21: Código para insertar los ficheros de una tabla en el proyecto de Visual Studio.

Después de insertar las tablas en el proyecto de **Visual Studio** se ha realizado la migración de datos desde los documentos **Excel** que ha proporcionado el cliente a la base de datos. Estos datos proporcionados contenían información sobre alumnos, personal y asignaturas de forma codificada y que hemos utilizado como datos de prueba.

La migración de datos se ha realizado directamente desde **SQL Server**. Al realizar la migración se ha creado una tabla en la base de datos por cada hoja **Excel** introducida y a partir de estas tablas se han insertado los datos oportunos a través de sentencias *Insert* en las tablas que habíamos creado anteriormente. Para realizar la migración se ha utilizado el asistente para importación y exportación de **SQL Server** y del que se puede observar una de sus pantallas en la Figura 22.

Para el resto de tablas de las que no hay datos proporcionados por el cliente, se han creado datos de prueba para comenzar a comprobar el funcionamiento de la aplicación web.

Al tener todas las tablas con datos se ha vuelto a ejecutar la aplicación para comprobar que éstos se muestran correctamente pero nos ha dado errores de ejecución. Habíamos configurado algunas tablas con claves compuestas pero **Serenity** da errores si algún atributo de una clave compuesta es clave ajena a otra tabla y que no nos había dado error al no tener datos anteriormente. Por tanto, se ha tenido que poner un identificador en cada una de las tablas afectadas. Por ejemplo, en la tabla Mensajes habíamos establecido en un principio que la clave primaria estaba compuesta por el identificador del alumno que recibe el mensaje, por el identificador del miembro del personal que envía el mensaje y la fecha y hora de envío. Ahora, la clave primaria de la tabla mensajes es un identificador del mensaje.

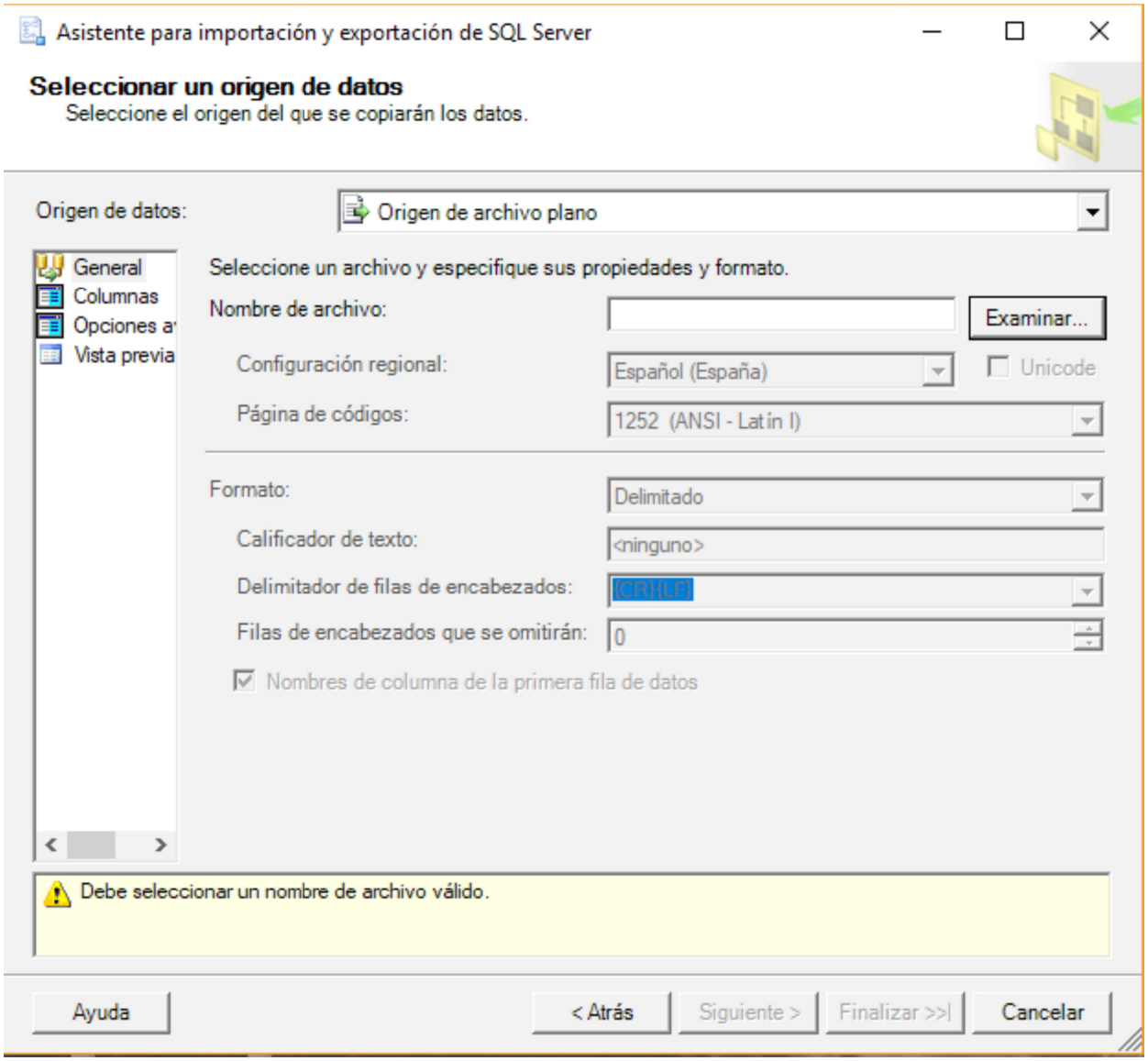

Figura 22: Pantalla para importar datos a una base de datos desde el asistente para importación y exportación de SQL Server.

Después de resolver esos errores y tras una de las reuniones con el cliente, se han añadido nuevos atributos en algunas de las tablas que son los que nombramos a continuación:

- En la tabla Alumno se ha añadido un atributo para la imagen del alumno, otro para indicar si tiene alguna enfermedad o alergia y otro para indicar si toma alguna medicación.
- En la tabla Asignatura se han añadido campos para indicar el precio de las asignaturas para media hora y una hora si son clases sueltas y media hora y una hora a la semana por mes.

Además, también se ha añadido una tabla nueva llamada Grupo. Esta tabla tiene como atributos un identificador, un nombre del grupo y una dirección de correo electrónico para poder enviar mensajes a dicho grupo. Una fila de esta tabla representa a un grupo de alumnos que asiste a

una misma clase. De esta manera, resulta más sencillo enviar un mensaje al grupo desde la aplicación web.

El resultado final de la base de datos tras estos cambios, es el mostrado en el apartado 4.2.2.2.

Después de la modificación de la base de datos se ha implementado la funcionalidad de enviar mensajes del personal al alumno. El envío de mensajes es unidireccional y, en principio, solo se puede enviar de uno a uno.

En la funcionalidad de envío de mensajes, también debe haber una opción para notificar por correo electrónico aunque el funcionamiento no entra dentro del alcance de este proyecto. Aún así, como nos ha parecido que íbamos a buen ritmo de acuerdo a la planificación, hemos decidido probar a implementarlo. El envío por correo electrónico nos ha dado algunos problemas debido a que había que cambiar algunas configuraciones de la cuenta **Gmail** desde la que se deben enviar los correos porque hay que otorgar diversos permisos. Hemos puesto un correo por defecto para efectuar las pruebas y funciona correctamente. De todas formas, no se ha completado la implementación para que se envíe al correo del alumno por lo que esta funcionalidad dentro de mensajes se queda como futura mejora del proyecto.

Después también se ha realizado la gestión de usuarios para que las personas registradas en el sistema puedan acceder con su usuario y contraseña, y para dar permisos de lectura y escritura en las distintas páginas a los distintos grupos de usuario.

La gestión de usuarios ha sido sencilla de realizar gracias al código que incluye la plantilla y a la base de datos que crea por defecto. Desde la propia aplicación web se han creado los roles de usuario y se le han otorgado los permisos para el acceso a cada sección de la aplicación. Después, tras insertar un nuevo alumno o un miembro del personal, se ha añadido un nuevo elemento en el formulario que muestra la información del alumno o miembro del personal para poder otorgarle un nombre de usuario y un rol. Este formulario se muestra en la Figura 23.

Al crear un usuario también se debe crear una contraseña pero como la creación del usuario se realiza por parte del personal de administración, se ha establecido una contraseña por defecto como se muestra en el código de la Figura 24. Esta contraseña puede ser cambiada después desde la misma aplicación web si es un miembro del personal o desde la aplicación móvil si se trata de un alumno.

Para relacionar una fila de la tabla Alumno o Personal de nuestra base de datos con una fila de la tabla Usuario de la base de datos por defecto hemos incluido un nuevo atributo en ambas primeras tablas para almacenar el identificador de usuario de la última tabla.

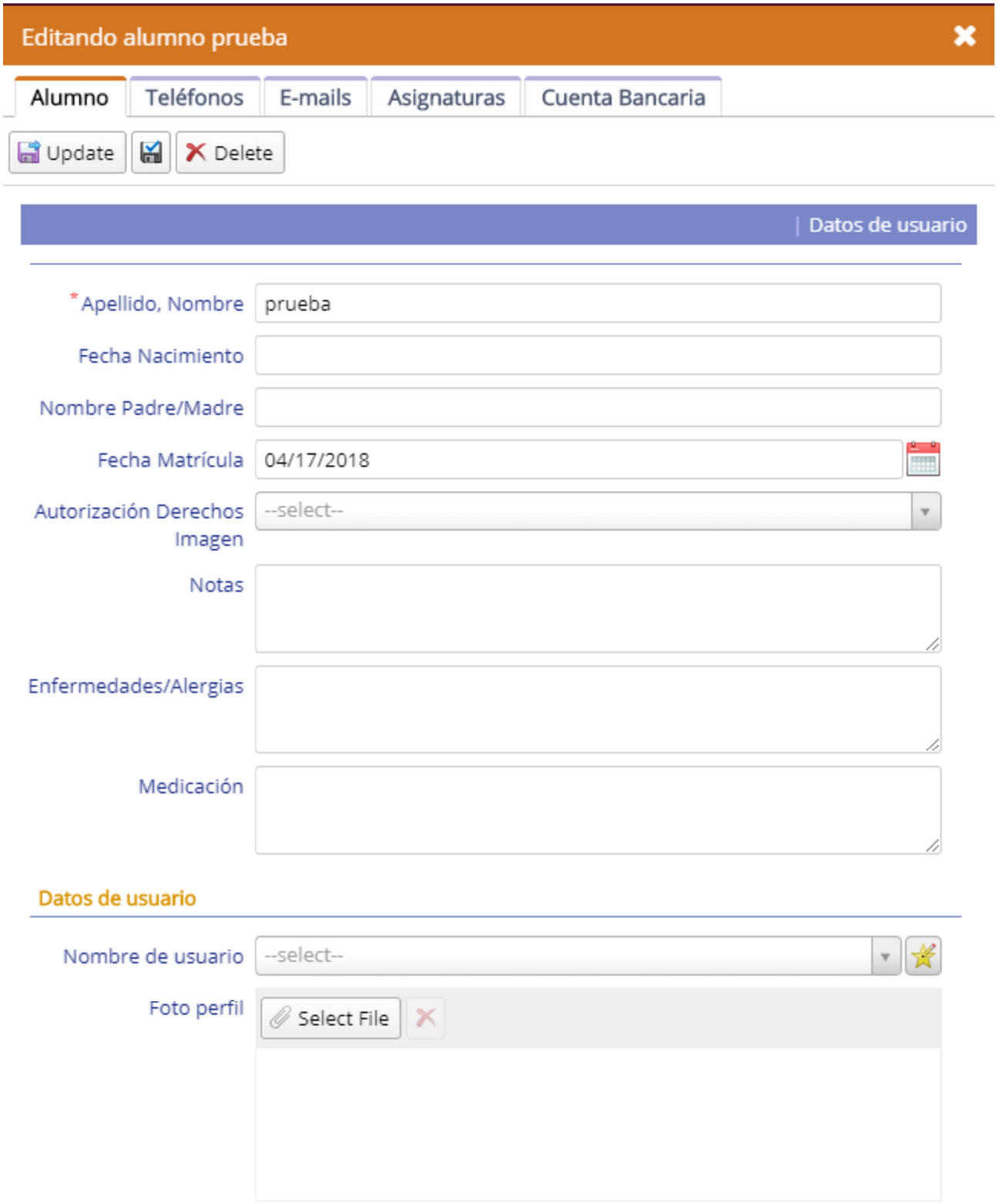

Figura 23: Formulario para añadir o modificar un alumno.

```
[DisplayName("Password"), Size(50), NotMapped, DefaultValue("pass123")]
public String Password
{
    get { return Fields.Password[this]; }
    set { Fields.Password[this] = value; }
}

[DisplayName("Confirm Password"), Size(50), NotMapped, DefaultValue("pass123")]
public String PasswordConfirm
{
    get { return Fields.PasswordConfirm[this]; }
    set { Fields.PasswordConfirm[this] = value; }
}
```

Figura 24: Código en el que se establece una contraseña por defecto al crear un nuevo usuario.

Desde el código de la aplicación también ha sido necesario realizar modificaciones. En los ficheros que manejan el acceso a la base de datos se han añadido los permisos de lectura y escritura oportunos para que, dependiendo del tipo de usuario que se haya registrado en el sistema, pueda visualizar o modificar la información de dicha tabla. El código para otorgar permisos se puede observar en la Figura 25.

```
[ConnectionKey("Escuela"), TableName("[dbo].[Mensaje]")]
[DisplayName("Mensajes"), InstanceName("Mensaje"), TwoLevelCached]
[ReadPermission("Administration:Personal")]
[ModifyPermission(PermissionKeys.Admin)]
[InsertPermission(PermissionKeys.Profesor)]
```

Figura 25: Código para establecer permisos en el módulo Mensajes. Solamente puede insertar un nuevo mensaje un usuario con rol de profesor y solo puede modificar el administrador.

También se ha deshabilitado la modificación de determinados atributos de una tabla y eso se ha realizado desde los ficheros en los que se indican qué atributos deben aparecer al visualizar los datos desde la tabla y del formulario para modificarlos. Un ejemplo se encuentra en la Figura 26.

```
[Updatable(false)]
public Int32 IdAlumno { get; set; }
```

Figura 26: Código para deshabilitar la edición de un campo.

La aplicación solo mostrará la información relacionada con el usuario que esté registrado en el sistema a excepción del personal administrativo que tenía acceso a todo. Este filtro de información se realiza a través del código que se muestra en la Figura 27.

```
protected override void ApplyFilters(SqlQuery query)
{
    base.ApplyFilters(query);

    var user = (UserDefinition)Authorization.UserDefinition;
    if (!Authorization.HasPermission(PermissionKeys.Admin))
    {
        if (Authorization.HasPermission(PermissionKeys.Alumno))
        {
            query
                .Where(fld.IdAlumnoUserId == user.UserId);
        }
        else if (Authorization.HasPermission(PermissionKeys.Profesor))
        {
            query
                .Where(fld.IdPersonalUserId == user.UserId);
        }
    }
}
```

Figura 27: Código para filtrar quién puede acceder al módulo en el que se encuentra.

Después de realizar la gestión de usuarios, la aplicación tenía el aspecto de la Figura 28, siendo el usuario registrado un profesor de la escuela de música.

Figura 28: Visualización del módulo Suscripciones que muestra la lista de clases y sus atributos que imparte el profesor que está accediendo al sistema.

Hasta este punto, la aplicación web ya ha cumplido los requisitos especificados por lo que se ha comenzado a leer documentación sobre **Docker** [17], a instalarlo en el sistema y a hacer pruebas de su funcionamiento siguiendo el tutorial de la documentación. La empresa no ha trabajado

anteriormente con **Docker** por lo que ha decido probar su uso con nuestra aplicación para analizar si ofrece un ahorro de costos mayor que ejecutándola localmente.

Desde **Visual Studio** es muy sencillo agregar la compatibilidad de un proyecto con **Docker** [18] siempre que sea un proyecto de **.NET Core**, ya que **Visual Studio** no permite agregar esta compatibilidad con otro tipo de proyecto.

Si antes de crear un proyecto nuevo ya se sabe que se quiere compatibilizar con **Docker**, al crearlo hay que habilitarlo activando la casilla *Enable Docker Support* como se muestra en la Figura 29. En el caso de que el proyecto ya esté creado, que es lo que nos sucede a nosotros, hay que habilitarlo desde el menú Proyecto como se muestra en la Figura 30.

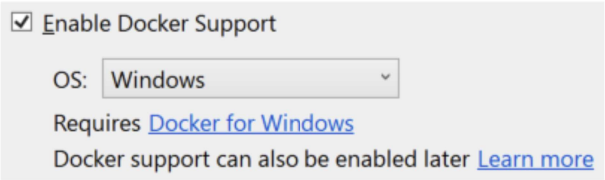

Figura 29: Opción para habilitar la compatibilidad del proyecto con Docker desde Visual Studio.

Docker utiliza características del núcleo **Linux** para el aislamiento de recursos. De esta forma, cuando una aplicación está en un contenedor, la vista de ésta está aislada del sistema operativo del equipo físico en el que se encuentra. Dentro de un contenedor se pueden almacenar todos los recursos que una aplicación necesita para ser ejecutada. Por tanto, si el contenedor de una aplicación se mueve a otro dispositivo con diferentes características e incluso con diferente sistema operativo, la aplicación contenida en el contenedor se va a seguir ejecutando correctamente.

Actualmente, **Docker** también utiliza contenedores con características del núcleo **Windows** pero éstos no pueden ser utilizados en dispositivos con un sistema operativo diferente a **Windows**. Por esta última razón, el sistema operativo que hemos seleccionado para el contenedor de nuestra aplicación es **Linux**.

Aplicando **Docker** a nuestro proyecto desde **Visual Studio** no hay que realizar ninguna configuración porque, al habilitarlo, crea todos los ficheros necesarios para que se pueda ejecutar en un contenedor. Para ejecutar la aplicación desde el contenedor hay que ejecutar el comando de la Figura 31 desde la terminal. Ahora, la aplicación se puede ejecutar desde un contenedor en un entorno de desarrollo y se puede seguir actualizando mientras está lanzada.

Después de probar la aplicación con **Docker**, el supervisor decidió que no se ejecutaría de esta manera ya que no ofrece ningún ahorro de costos significativo para este proyecto.

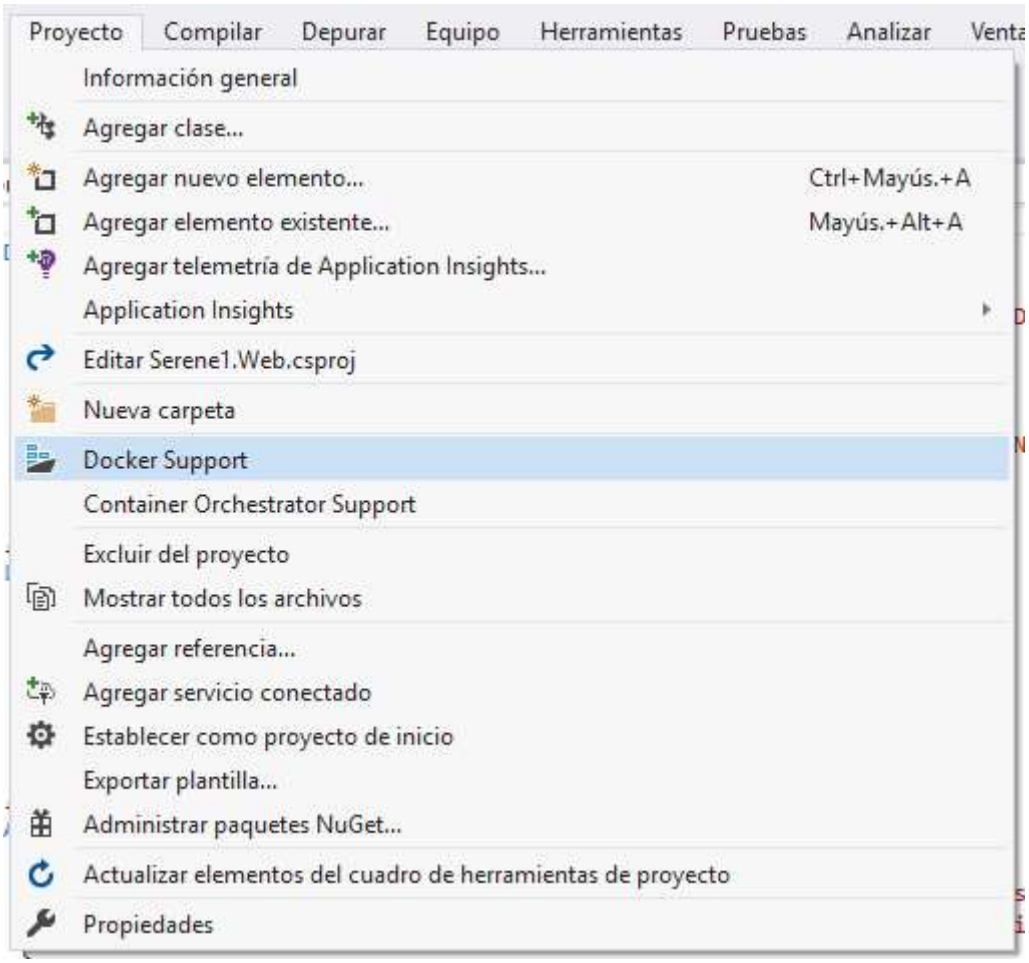

Figura 30: Opción para habilitar la compatibilidad con Docker de un proyecto ya creado en Visual Studio.

```
PS C:\Users\Aglaya\source\repos\Serene1\Serene1\Serene1.Web> docker-compose up
```

Figura 31: Comando para ejecutar una aplicación dentro de un contenedor Docker.

Tras las pruebas con **Docker** se ha vuelto a modificar la aplicación ya que la aplicación móvil relacionada precisaba unos datos a los que no tenía acceso directo. En este caso concreto, la aplicación móvil estaba accediendo al módulo mensajes pero no le era posible acceder al nombre del miembro del personal que ha enviado el mensaje ya que la base de datos solo le devolvía el identificador. Para solucionar este problema, se ha creado una vista en la base de datos y se ha creado un módulo especial para mostrar esa vista simplemente a los alumnos. De esta forma, el usuario ya puede ver el nombre de la persona que le ha enviado un mensaje y no su identificador.

5.2 Pruebas y despliegue

Se han realizado pruebas unitarias para comprobar el correcto funcionamiento de los módulos durante el desarrollo del proyecto. Estas pruebas han consistido en insertar datos en cada uno de los módulos y comprobar que se muestran todos los datos, incluidos los de las claves ajenas. Usualmente, se ha realizado parte de estas pruebas durante las presentaciones al cliente, ya que quería asegurarse de que los módulos ofrecían las funcionalidades deseadas.

Por ejemplo, para insertar toda la información de un alumno, hay que insertar también datos en las pestañas Teléfonos y Emails que se muestran en la Figura 23. Estas dos pestañas representan otras tablas de la base de datos a las que no se pueden acceder desde un módulo propio de la aplicación, sino que solo se puede acceder a ellas a través del formulario del módulo Alumnos. Como se muestra en la Figura 32, se habilitó que se relacionara por defecto el identificador del alumno del que se está editando información para no tener que introducirlo manualmente desde la pestaña y así evitar errores al añadir un nuevo teléfono en este caso. Por tanto, lo que se ha probado en este caso concreto es que, internamente, las tablas están relacionadas correctamente y que las claves ajenas no han provocado ningún error de inserción.

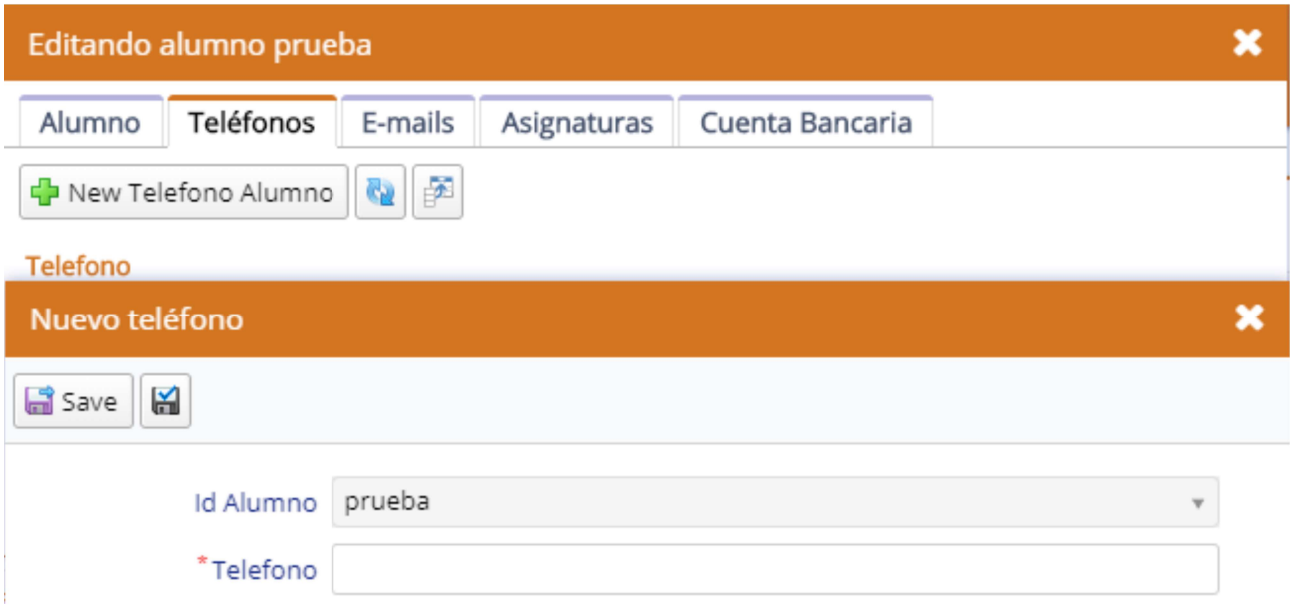

Figura 32: Formulario para insertar un nuevo teléfono del alumno "prueba".

Por último, se ha publicado la aplicación en la nube que ofrece el servidor de **Azure**. Ahora, la aplicación puede ser accesible desde cualquier dispositivo y desde cualquier lugar aunque, de momento, el acceso está restringido solo a la red de la empresa ya que la aplicación todavía no es pública y falta añadir funcionalidades que no entraban dentro del alcance de este proyecto.

Capítulo 6

Conclusiones

El resultado final ha sido satisfactorio. Se han realizado todas las tareas propuestas en el periodo de tiempo establecido y el cliente se ha mostrado satisfecho con el trabajo realizado.

A pesar de que se ha decidido no utilizar **Docker** para el proyecto, el estudio sobre esta plataforma nos puede ser útil para futuros proyectos.

Este proyecto se ha desarrollado con éxito gracias a la formación que he recibido durante el transcurso del grado. Varias de las asignaturas cursadas han sido de gran ayuda tanto para la realización del proyecto en la estancia como para el contenido de esta memoria. Esas asignaturas son las que listo a continuación:

- EI1020 – Bases de datos.
- EI1023 – Fundamentos de ingeniería de software.
- EI1027 – Diseño e implementación de sistemas de información.
- EI1030 – Análisis de sistemas de información.
- EI1036 – Tecnologías web para los sistemas de información.
- EI1038 – Diseño de sistemas de bases de datos.

Considero que ha sido un proyecto bastante completo ya que he tenido que realizar todas las partes de un proyecto informático como son el análisis, el diseño y la implementación. Además, también se ha tenido que diseñar e implementar una base de datos.

Lo que considero que me ha ocupado más tiempo ha sido entender el funcionamiento de la plantilla **Serenity**, ya que tiene sus peculiaridades y los lenguajes de programación utilizados (C# y Typescript) no son con los que más hemos trabajado en la carrera.

Pienso que me resultará muy útil haberme documentado sobre **Docker**, ya que el uso de contenedores para la ejecución de aplicaciones está en auge.

La estancia en la empresa ha sido satisfactoria. El trato recibido ha sido espléndido y el trabajo ha sido dinámico. La empresa tiene su sede dentro de la escuela de música cliente por lo que siempre ha sido posible realizar consultas al cliente fuera de las reuniones si era necesario.

El cliente ha propuesto algunas mejoras durante las reuniones que se han realizado pero no se han podido llevar a cabo por falta de tiempo. Estas mejoras van relacionadas con el envío de mensajes. El cliente quiere que se pueda enviar una notificación al correo electrónico cuando un profesor le envíe un mensaje a un alumno -como ya comenzamos a implementar-, y también enviar mensajes a un grupo de alumnos.

Además, también quiere que se envíe un correo electrónico cuando se registre un usuario para que pueda cambiar su contraseña desde la aplicación móvil si es un alumno o desde la aplicación web si es un miembro del personal.

Todas estas mejoras han sido estudiadas con el supervisor y se llevarán a cabo en un desarrollo futuro.

Bibliografía

[1] CaliforniaDrims. https://californiadrims.com/es/ [Consulta: Junio de 2018]

[2] Microsoft. Visual Studio. https://www.visualstudio.com/es/vs/ [Consulta: Junio de 2018]

[3] Microsoft. Introducción a ASP.NET Core. https://docs.microsoft.com/es-es/aspnet/core/?view=aspnetcore-2.1 [Consulta: Junio de 2018]

[4] Visual Studio. What is Serenity Platform. https://marketplace.visualstudio.com/items?itemName=VolkanCeylan.SereneSerenityApplicationTemplate#overview [Consulta: Junio de 2018]

[5] Microsoft. Visual Studio Team Services. https://www.visualstudio.com/es/team-services/ [Consulta: Junio de 2018]

[6] Microsoft. SQL Server. https://www.microsoft.com/es-es/sql-server/sql-server-2017 [Consulta: Junio de 2018]

[7] Microsoft Azure. https://azure.microsoft.com/es-es/ [Consulta: Junio de 2018]

[8] Docker. https://www.docker.com/ [Consulta: Junio de 2018]

[9] No Magic. MagicDraw. https://www.nomagic.com/products/magicdraw [Consulta: Junio de 2018]

[10] NinjaMock. https://ninjamock.com/ [Consulta: Junio de 2018]

[11] Google. Google Calendar. https://calendar.google.com/calendar [Consulta: Junio de 2018]

[12] Vertabelo. https://www.vertabelo.com/ [Consulta: Junio de 2018]

[13] Guía Hays. Guía del mercado laboral 2018. [Consulta: Julio de 2018]

[14] Microsoft Azure. Calculadora de precios. https://azure.microsoft.com/es-es/pricing/calculator/?service=container-service [Consulta: Junio de 2018]

[15] Wikipedia. Cliente-servidor. https://es.wikipedia.org/wiki/Cliente-servidor [Consulta: Junio de 2018]

[16] Gitbooks. Guía Serenity. https://volkanceylan.gitbooks.io/serenity-guide/ [Consulta: Junio de 2018]

[17] Adam Freeman. Essential Docker for ASP.NET Core MVC. [Consulta: Mayo de 2018]

[18] Microsoft. Visual Studio Tools para Docker con ASP.NET Core https://docs.microsoft.com/es-es/aspnet/core/host-and-deploy/docker/visual-studio-tools-for-docker?view=aspnetcore-2.1 [Consulta: Junio de 2018]

Anexo A

Documentación del diagrama de clases

Clase: Alumno	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: Un objeto de esta clase representa a un alumno de la escuela de música.	
Asociaciones: • Con la clase “Clase” • Con la clase “Mensaje” • Con la clase “Usuario”	

Tabla 19: Documentación de la clase Alumno.

Clase: Personal	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: Un objeto de esta clase representa a un miembro del personal de la escuela de música.	
Asociaciones: • Con la clase “Clase” • Con la clase “Mensaje” • Con la clase “Usuario”	

Tabla 20: Documentación de la clase Personal.

Clase: Asignatura	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: Un objeto de esta clase es, en su mayoría, un instrumento que se utiliza para impartir una clase. También puede representar a un conjunto de instrumentos o a otro tipo de actividades.	
Asociaciones: • Con la clase “Clase”	

Tabla 21: Documentación de la clase Asignatura.

Clase: Clase	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: Un objeto de esta clase representa la suscripción de un alumno a una determinada asignatura durante un tiempo específico.	
Asociaciones: • Con la clase “Asignatura” • Con la clase “Alumno” • Con la clase “Personal”	

Tabla 22: Documentación de la clase Clase.

Clase: Mensaje	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: Un objeto de esta clase es un mensaje que un miembro del personal envía a un alumno.	
Asociaciones: • Con la clase “Alumno” • Con la clase “Personal”	

Tabla 23: Documentación de la clase Mensaje.

Clase: Usuario	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Descripción: Un objeto de esta clase representa a un usuario del sistema.	
Asociaciones: • Con la clase “Alumno” • Con la clase “Personal”	

Tabla 24: Documentación de la clase Usuario.

Asociación: Recibe	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Clases: Alumno_Mensaje Un alumno recibe un mensaje. Descripción: Un alumno recibe un mensaje enviado por un miembro del personal.	
Multiplicidad: • Rol 1: Alumno 0..* • Rol 2: Mensaje 1	
Comentarios: Cuando se crea el mensaje ya debe existir el alumno al que se le quiere enviar.	

Tabla 25: Documentación de la asociación Recibe.

Asociación:Envia	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Clases: Personal_Mensaje Un miembro del personal envía un mensaje. Descripción: Un miembro del personal envía un mensaje a un alumno.	
Multiplicidad: • Rol 1: Personal 0..* • Rol 2: Mensaje 1	
Comentarios: Cuando se crea el mensaje ya debe existir el alumno al que se le quiere enviar.	

Tabla 26: Documentación de la asociación Envia.

Asociación: Asiste	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Clases: Alumno_Clase Un alumno asiste a una clase. Descripción: Un alumno está suscrito a una clase de una asignatura determinada.	
Multiplicidad: • Rol 1: Alumno 0..* • Rol 2: Clase 1..*	

Tabla 27: Documentación de la asociación Asiste.

Asociación: Imparte	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Clases: Personal_Clase Un miembro del personal imparte una clase. Descripción: Un miembro del personal con rol de profesor, imparte una clase de una asignatura determinada.	
Multiplicidad: • Rol 1: Personal 0..* • Rol 2: Clase 1	

Tabla 28: Documentación de la asociación Imparte.

Asociación: Pertenece	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Clases: Clase_Asignatura Una clase pertenece a una asignatura. Descripción: En una clase se imparte una asignatura determinada.	
Multiplicidad: • Rol 1: Clase 1 • Rol 2: Asignatura 0..*	
Comentarios: La asignatura asociada debe estar registrada en el sistema.	

Tabla 29: Documentación de la asociación Pertenece.

Asociación: Representa	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Clases: Usuario_Alumno Un usuario representa a un alumno. Descripción: Un alumno tiene un usuario y una contraseña para poder acceder al sistema.	
Multiplicidad: • Rol 1: Usuario 1 • Rol 2: Alumno 1	
Comentarios: Un alumno no debería poder acceder a esta aplicación pero tiene una entrada en la tabla usuario para acceder a sus datos a través de la aplicación móvil realizada en otro proyecto y que emplea la misma base de datos.	

Tabla 30: Documentación de la asociación Representa.

Asociación: Representa	
Autor: Aglaya Pons Martínez	Fecha creación: 15/03/2018 Fecha revisión: Fecha aprobación: Versión: 1.0
Clases: Usuario_Personal Un usuario representa a un miembro del personal. Descripción: Un miembro del personal tiene un usuario y una contraseña para poder acceder al sistema.	
Multiplicidad: • Rol 1: Usuario 1 • Rol 2: Personal 1	
Comentarios: Por defecto, cuando se cree un usuario para un miembro del personal tendrá solo permisos de profesor. Si se quiere que ese usuario tenga permisos de administrador, será el administrador que lo registre en el sistema el que deba otorgarle dichos permisos manualmente.	

Tabla 31: Documentación de la asociación Representa.

Anexo B

Diseño físico de la base de datos

En esta sección se muestran las sentencias CREATE TABLE para introducir las tablas en la base de datos. Se han extraído al realizar el esquema lógico en la aplicación web Vertabelo. Las sentencias son las siguientes:

```
-- tables

-- Table: Alumno

CREATE TABLE Alumno (
    idAlumno int  NOT NULL,
    apellidoNombre varchar(50)  NOT NULL,
    fechaMatricula date  NOT NULL,
    autorizacionDerechosImagen bit  NOT NULL,
    notas varchar(200)  NOT NULL,
    nombrePadreMadre varchar(50)  NOT NULL,
    fechaNacimiento date  NOT NULL,
    imagen image  NOT NULL,
    enfermedadAlergia varchar(200)  NOT NULL,
    Medicacion varchar(200)  NOT NULL,
    CONSTRAINT Alumno_pk PRIMARY KEY  (idAlumno)
);


-- Table: Asignatura

CREATE TABLE Asignatura (
    idAsignatura int  NOT NULL,
    categoria varchar(30)  NOT NULL,
    nombre varchar(30)  NOT NULL,
    descripcion varchar(200)  NOT NULL,
    precioHora float(2)  NOT NULL,
    precioMediaHora float(2)  NOT NULL,
```

```
    precioHoraMes float(2)  NOT NULL,
    precioMediaHoraMes float(2)  NOT NULL,
    CONSTRAINT Asignatura_pk PRIMARY KEY  (idAsignatura)
);

-- Table: Clase
CREATE TABLE Clase (
    idAlumno int  NOT NULL,
    idPersonal int  NOT NULL,
    idAsignatura int  NOT NULL,
    fechaInicio date  NOT NULL,
    fechaFin date  NOT NULL,
    hora time(4)  NOT NULL,
    duracion int  NOT NULL,
    aula varchar(10)  NOT NULL,
    idGrupo int  NOT NULL,
    CONSTRAINT Clase_pk PRIMARY KEY  (idAlumno,idAsignatura,fechaInicio)
);

-- Table: CuentaBancaria
CREATE TABLE CuentaBancaria (
    idAlumno int  NOT NULL,
    iban varchar(50)  NOT NULL,
    CONSTRAINT CuentaBancaria_pk PRIMARY KEY  (idAlumno,iban)
);

-- Table: EmailAlumno
CREATE TABLE EmailAlumno (
    idAlumno int  NOT NULL,
    email varchar(100)  NOT NULL,
```

```
    CONSTRAINT EmailAlumno_pk PRIMARY KEY  (idAlumno,email)
);

-- Table: EmailPersonal
CREATE TABLE EmailPersonal (
    idPersonal int  NOT NULL,
    email int  NOT NULL,
    CONSTRAINT EmailPersonal_pk PRIMARY KEY  (idPersonal,email)
);

-- Table: Grupo
CREATE TABLE Grupo (
    idGrupo int  NOT NULL,
    emailGrupo varchar(100)  NOT NULL,
    CONSTRAINT Grupo_pk PRIMARY KEY  (idGrupo)
);

-- Table: Mensaje
CREATE TABLE Mensaje (
    idAlumno int  NOT NULL,
    idPersonal int  NOT NULL,
    asunto varchar(50)  NOT NULL,
    mensaje text  NOT NULL,
    fechaHora datetime  NOT NULL,
    CONSTRAINT Mensaje_pk PRIMARY KEY  (idAlumno,idPersonal,fechaHora)
);

-- Table: Personal
CREATE TABLE Personal (
    idPersonal int  NOT NULL,
```

```
    apellidoNombre varchar(50)  NOT NULL,
    telefono int  NOT NULL,
    cargo varchar(25)  NOT NULL,
    CONSTRAINT Personal_pk PRIMARY KEY  (idPersonal)
);

-- Table: TelefonoAlumno
CREATE TABLE TelefonoAlumno (
    idAlumno int  NOT NULL,
    telefono int  NOT NULL,
    CONSTRAINT TelefonoAlumno_pk PRIMARY KEY  (idAlumno,telefono)
);

-- foreign keys
-- Reference: Alumno_EmailAlumno (table: EmailAlumno)
ALTER TABLE EmailAlumno ADD CONSTRAINT Alumno_EmailAlumno
    FOREIGN KEY (idAlumno)
    REFERENCES Alumno (idAlumno);

-- Reference: Alumno_TelefonoAlumno (table: TelefonoAlumno)
ALTER TABLE TelefonoAlumno ADD CONSTRAINT Alumno_TelefonoAlumno
    FOREIGN KEY (idAlumno)
    REFERENCES Alumno (idAlumno);

-- Reference: Clase_Alumno (table: Clase)
ALTER TABLE Clase ADD CONSTRAINT Clase_Alumno
    FOREIGN KEY (idAlumno)
    REFERENCES Alumno (idAlumno);

-- Reference: Clase_Asignatura (table: Clase)
```

```
ALTER TABLE Clase ADD CONSTRAINT Clase_Asignatura
    FOREIGN KEY (idAsignatura)
    REFERENCES Asignatura (idAsignatura);

-- Reference: Clase_Grupo (table: Clase)
ALTER TABLE Clase ADD CONSTRAINT Clase_Grupo
    FOREIGN KEY (idGrupo)
    REFERENCES Grupo (idGrupo);

-- Reference: Clase_Personal (table: Clase)
ALTER TABLE Clase ADD CONSTRAINT Clase_Personal
    FOREIGN KEY (idPersonal)
    REFERENCES Personal (idPersonal);

-- Reference: CuentaBancaria_Alumno (table: CuentaBancaria)
ALTER TABLE CuentaBancaria ADD CONSTRAINT CuentaBancaria_Alumno
    FOREIGN KEY (idAlumno)
    REFERENCES Alumno (idAlumno);

-- Reference: EmailPersonal_Personal (table: EmailPersonal)
ALTER TABLE EmailPersonal ADD CONSTRAINT EmailPersonal_Personal
    FOREIGN KEY (idPersonal)
    REFERENCES Personal (idPersonal);

-- Reference: Mensaje_Alumno (table: Mensaje)
ALTER TABLE Mensaje ADD CONSTRAINT Mensaje_Alumno
    FOREIGN KEY (idAlumno)
    REFERENCES Alumno (idAlumno);

-- Reference: Mensaje_Personal (table: Mensaje)
```

```
ALTER TABLE Mensaje ADD CONSTRAINT Mensaje_Personal
    FOREIGN KEY (idPersonal)
    REFERENCES Personal (idPersonal);

-- End of file.
```

Anexo C

Diseño CSS

En esta sección se muestra el código CSS para el estilo de la aplicación *web*.

```
.skin-azure-light .control-sidebar-dark + .control-sidebar-bg {
    background: #d1701d;
}

.skin-azure-light .main-header .navbar {
    background-color: #d1701d;
    z-index: 999;
    min-height: initial;
}

.skin-azure-light .main-header .sidebar-toggle {
    font-size: 2.5rem;
    color: #fff;
    line-height: 20px;
    border-right: 1px solid #d1701d;
    padding: 15px 15px;
}

.skin-azure-light .main-header .logo {
    font-family: 'Open Sans';
    color: #fff;
    z-index: 1000;
    background-color: #d1701d;
    padding: 0 0;
}

.skin-azure-light .navbar-nav > li > a {
    border-left: 1px solid #d1701d;
    color: #fff;
}

.skin-azure-light .grid-title,
.skin-azure-light .panel-titlebar {
    color: #d1701d;
    font-size: 17px;
}

.skin-azure-light .s-Panel {
    border-radius: 0;
    border-top: 3px solid #d1701d;
}

.skin-azure-light .main-header li.user-header {
    background-color: #d1701d;
}

.skin-azure-light .s-QuickSearchBar .quick-search-icon {
    background: #d1701d;
```

```
}

.skin-azure-light .slick-header-column.ui-state-default {
    color: #d1701d;
}

.skin-azure-light .slick-cell > a {
    color: #d1701d;
}

.skin-azure-light .content-wrapper,
.skin-azure-light .right-side {
    background-color: #edf5fb;
}

.skin-azure-light .sidebar-menu > li:hover > a,
.skin-azure-light .sidebar-menu > li.active > a {
    color: #fff;
    border-left-color: #d1701d;
    background-color: #d1701d;
}

    .skin-azure-light .sidebar-menu > li:hover > a i.nav-icon,
    .skin-azure-light .sidebar-menu > li.active > a i.nav-icon {
        color: #fff;
    }

.skin-azure-light .sidebar-menu .nav-icon {
    color: #d1701d;
}

.skin-azure-light .sidebar-menu > li > a > .nav-icon {
    color: #d1701d;
}

.skin-azure-light .sidebar-menu > li > ul > li:hover > a,
.skin-azure-light .sidebar-menu > li > ul > li.active > a {
    color: #fff;
    background-color: #d1701d;
    border-radius: 13px;
    border-right: 4px solid #d1701d;
    border-left: 4px solid #d1701d;
    box-shadow: 0px 0px 2px white;
    padding-left: 15px;
}

    .skin-azure-light .sidebar-menu > li > ul > li:hover > a .nav-icon,
    .skin-azure-light .sidebar-menu > li > ul > li.active > a .nav-icon {
        color: #fff;
    }

.skin-azure-light .sidebar-form input[type="text"],
.skin-azure-light .sidebar-form .btn {
    background-color: #eaf1f7;
}
```

```
.skin-azure-light .sidebar-menu > li li li:hover > a,
.skin-azure-light .sidebar-menu li li li.active > a {
    color: #d1701d;
}

    .skin-azure-light .sidebar-menu > li li li:hover > a .nav-icon,
    .skin-azure-light .sidebar-menu li li li.active > a .nav-icon {
        color: #d1701d;
    }

.skin-azure-light .sidebar-form input[type="text"] {
    color: #666;
}

.skin-azure-light .main-header .navbar {
    min-height: 62px;
    margin-left: 0;
    background: #d1701d;
}

.skin-azure-light section.content > .s-DataGrid {
    border-radius: 0;
    border-top: 2px solid #d1701d;
    box-shadow: 0 1px 3px rgba(0, 0, 0, 0.3);
}

.skin-azure-light .main-header .logo {
    position: absolute;
    left: 6rem;
    border-left: 1px solid #d1701d;
    top: 6px;
}

.main-header .logo i{
    background: url();
}
```