MAMAS

Aula virtual para niños y niñas de infantil

[FECHA]

[NOMBRE DE LA COMPAÑÍA]
[Dirección de la compañía]

1. INTRODUCCION

Realizamos este proyecto ya que vemos necesaria una mejora en las aulas virtuales. Tienen una gran utilidad y se utilizan en todos los institutos y universidades, pero no están del todo implementadas en colegios donde son los padres los que utilizarán la aplicación y no los alumnos. Por lo tanto, creemos que vamos a aportar un extra de comunicación en los colegios y los padres van a poder estar más informados en todo momento y se van a sentir más partícipes en la vida académica de sus hijos.

2. ESTUDIO DE MERCADO:

Hay dos tipos principales de aplicaciones orientadas a mejorar el proceso educativo mediante la creación de aulas virtuales: las públicas que son financiadas por el Estado, como Papas 2.0 que pertenece a la Junta de Castilla-La Mancha, y las privadas, como ClassDojo o Google Classroom. Muchas universidades también tienen su propia aplicación de aula virtual, como la Universidad de Valencia. Pero estas últimas no serían parte de la competencia, ya que el *target* de nuestra aplicación son las escuelas de primaria e infantil, siendo los padres, los profesores y un administrador por centro los usuarios que la usarían. Así después de un pequeño estudio se puede ver que este tipo de app tiene hueco en el mercado actual.

Este tipo de aplicación cubre la necesidad de tener una comunicación a tiempo real y personalizada entre profesores y padres, así como involucrarles más en el proceso educativo del centro al poder tener un seguimiento mayor de lo que hacen sus hijos.

3. DESARROLLO

Arquitectura de desarrollo

Utilizaremos la mayoría de tecnologías y herramientas de **Microsoft**, ya que tienen muy buena **documentación** y están muy bien **conectadas entre ellas** para poder usarlas juntas. Las diferentes tecnologías y herramientas que se han utilizado son:

- **C#**

C# es un **lenguaje de programación orientado a objetos** desarrollado y estandarizado por **Microsoft** como parte de su plataforma **.NET**, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270).

Su sintaxis básica **deriva de C/C++** y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma.

- **Microsoft Visual Studio 2019**

Microsoft Visual Studio es un **entorno de desarrollo integrado** (IDE, por sus siglas en inglés) para sistemas operativos **Windows**. **Soporta múltiples lenguajes** de programación, tales como C++, **C#,** Visual Basic .NET, F#, Java, Python, Ruby y PHP, al igual que entornos de desarrollo web, como ASP.NET MVC, Django, etc.

Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así, se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos y consolas, entre otros.

Este IDE incluye características como la finalización automática de código, un sofisticado sistema de administración de proyectos y soluciones, una biblioteca exhaustiva de plantillas de proyecto, control de código fuente integrado, etc.

- **Microsoft SQL Server**

Microsoft SQL Server es un **sistema de gestión de base de datos relacional**, desarrollado por la empresa **Microsoft**.

El lenguaje de desarrollo utilizado (por línea de comandos o mediante la interfaz gráfica de **Management Studio**) es Transact-SQL (**TSQL**), una implementación del estándar ANSI del lenguaje SQL, utilizado para manipular y recuperar datos (**DML**), crear tablas y definir relaciones entre ellas (**DDL**).

SQL Server ha estado tradicionalmente disponible solo para sistemas operativos Windows de Microsoft, pero desde 2016 está disponible para GNU/Linux.

Puede ser configurado para **utilizar varias instancias en el mismo servidor físico**.

- **Xamarin**

Xamarin es un conjunto de herramientas de Microsoft para el desarrollo de aplicaciones móviles.

Xamarin es único en este entorno, ya que **ofrece un solo lenguaje** (C#), una **biblioteca de clases** y un **tiempo de ejecución que funciona en las plataformas móviles** iOS, Android y Windows Phone (el lenguaje nativo de Windows Phone ya es C#), al mismo tiempo que sigue compilando aplicaciones nativas (no interpretadas) con un rendimiento lo bastante bueno incluso para juegos exigentes.

Xamarin combina la potencia de las plataformas nativas y agrega una serie de **características propias muy eficaces**, incluidas las siguientes:

1.- Enlaces completos para los SDK subyacentes: Xamarin contiene enlaces para casi todos los SDK de plataforma subyacentes en iOS y Android.

2.- Interoperabilidad con Objective-C, Java, C y C++: Xamarin ofrece funciones para invocar directamente las bibliotecas de Objective-C, Java, C y C++, lo que le permite usar diversos tipos de código de terceros ya creado.

3.- Construcciones de lenguaje moderno: las aplicaciones de Xamarin se escriben en C#, con características como el lenguaje dinámico, construcciones funcionales como lambdas, LINQ, características de programación en paralelo, genéricos sofisticados, etc.

4.- Increíble biblioteca de clases base (BCL): las aplicaciones de Xamarin usan la BCL de .NET, una enorme colección de clases con características completas y optimizadas, como una eficaz compatibilidad con XML, bases de datos, serialización, E/S, cadenas y redes, por nombrar algunos.

5.- Moderno entorno de desarrollo integrado (IDE): Xamarin usa Visual Studio para Mac en Mac OS X y Visual Studio en Windows.

6.- Compatibilidad multiplataforma móvil: Xamarin ofrece una compatibilidad multiplataforma sofisticada con las tres principales plataformas móviles: iOS, Android y Windows Phone. **Es posible escribir aplicaciones de modo que compartan hasta el 90 % del código.**

Cuando se **compilan** aplicaciones de Xamarin, el resultado es un **paquete de aplicación**, ya sea un archivo .**app** en iOS o un archivo .**apk** en Android.

- **Postman**

Postman es una aplicación que se utiliza para probar API REST, nos será útil ya que se pueden crear espacios de trabajo donde uno vea las pruebas que ha hecho el otro.

- **.NET Core**

Es un framework de Microsoft que se utilizará para desarrollar una API REST (servidor en red) que maneje una estructura de datos hecha en SQL Server. La API REST se conecta en local a la base de datos de SQL Server, y cada aplicación se conectará en remoto a la API REST para utilizar la base de datos.

- **Swashbuckle**

Es la herramienta con la cual hemos creado la interfaz gráfica de la API, donde se explica lo que hace cada función, sus parámetros, qué devuelve y que códigos de respuesta, etc. Básicamente es el manual de uso de la API para que lo pueda usar cualquier persona.

  - Swashbuckle.AspNetCore.Swagger: modelo de objetos de Swagger y middleware para exponer objetos SwaggerDocument como puntos de conexión JSON.
  - Swashbuckle.AspNetCore.SwaggerGen: generador de Swagger que genera objetos SwaggerDocument directamente desde las rutas, controladores y modelos. Se suele combinar con el middleware de punto de conexión de Swagger para exponer automáticamente el JSON de Swagger.
  - Swashbuckle.AspNetCore.SwaggerUI: versión insertada de la herramienta de interfaz de usuario de Swagger. Interpreta el JSON de Swagger para crear una experiencia enriquecida y personalizable para describir la funcionalidad de la

API web. Incluye herramientas de ejecución de pruebas integradas para los métodos públicos.

- **Prism**

Prism es un framework que se utiliza para facilitar la implementación del patrón de diseño MVVM en Xamarin.

- **Microsoft Entity Framework**

Mircrosoft Entity Framework es un framework para facilitar el paso de una estructura de base de datos a código.

- **NewtonSoft**

- **RestSharp**

4. ESTRUCTURA DE LA APP

La idea es hacer un aula virtual para niños y niñas de infantil. En la app habrá una pantalla de log-in donde se podrán conectar padres y profesores. Habrá distintos módulos dentro de la app con distintas funcionalidades.

Cada perfil tendrá unos privilegios u otros y unas funcionalidades u otras dependiendo de si es profesor, padre o administrador.

- **Perfil Profesor**
    - **Selección de alumno:** Una vez hecho el log-in, el profesor podrá elegir el alumno con el que quiere interactuar. Después le saldrá una pantalla con los módulos.
    - **Módulo de Notas:** En este módulo los profesores podrán poner las notas de sus alumnos.

- **Perfil Padre**

- **Selección de hijo:** Una vez hecho el log-in, el padre podrá elegir al hijo (si solo tiene un hijo en el colegio solo le saldrá uno). Después le saldrá la pantalla con los módulos.
- **Módulo de Notas:** En este módulo los padres podrán ver las notas de su hijo. (Mismo botón que en el módulo Notas de profesores)

- **Perfil Administrador**
  - **Configuración asignaturas:** Módulo para insertar asignaturas.
  - **Configuración cursos:** Módulo para insertar cursos.
  - **Configuración alumnos:** Módulo para insertar alumnos.
  - **Configuración padres:** Módulo para insertar padres.
  - **Configuración profesores:** Módulo para insertar profesores.
  - **Eliminar y editar datos:** En este módulo se podrán eliminar o editar los datos de los cursos, asignaturas, alumnos, profesores y padres.

Principales casos de uso.

IDENTIFICACIÓN Y ENTRADA AL SISTEMA

Actores:	Usuarios registrados.
Propósito:	Identificar y autenticar un usuario ente el sistema.
Resumen:	El usuario introduce el nombre de usuario. El sistema comprueba si existe en la base de datos. Si existe el nombre de usuario pero es la primera vez que intenta acceder y no tiene una clave asignada, le pide que introduzca la clave que usará a partir de ahora. Si ya tiene una clave asignada, le pedirá que la introduzca y si es correcta permite el acceso, en caso de que el usuario o la clave sean incorrectos no permite el acceso.
Precondiciones:	El usuario no está conectado al sistema.

Curso normal de los eventos:

1. El usuario introduce un nombre de usuario correcto con clave ya asignada.
2. El sistema comprueba que el nombre de usuario existe.
3. El sistema muestra la pantalla para introducir la clave.
4. El sistema comprueba que la clave coincide para ese nombre de usuario.
5. El sistema muestra la pantalla inicial que corresponde al perfil del usuario.

Cursos alternos

Caso B:

1. El usuario introduce un nombre de usuario correcto sin clave asignada.
2. El sistema comprueba que el nombre de usuario existe.
3. El sistema muestra la pantalla para introducir la clave que usará a partir de ahora.
4. El sistema muestra la pantalla inicial que corresponde al perfil del usuario.

Caso C:

1. El usuario introduce un nombre usuario que no existe.
2. El sistema muestra un mensaje informando de que no existe el nombre de usuario.

Caso D:

1. El usuario introduce un nombre de usuario correcto con clave ya asignada.
2. El sistema comprueba que el nombre de usuario existe.
3. El sistema muestra la pantalla para introducir la clave.
4. El usuario introduce una clave incorrecta.
5. El sistema muestra un mensaje indicando que la clave introducida es incorrecta
6. El sistema muestra la pantalla inicial que corresponde al perfil del usuario.

ALTA DE UNA ASIGNATURA

Actores:	Administrador.
Propósito:	Dar de alta en el sistema una nueva asignatura.
Resumen:	El administrador introduce el nombre de la nueva asignatura. El sistema registra la nueva asignatura.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador accede mediante el botón *Añadir Asignatura*.
2. El sistema muestra el formulario que debe rellenarse con los datos de la asignatura.
3. El administrador introduce todos los datos y pulsa el botón *Añadir*.
4. El sistema comprueba que el nombre de la asignatura no está duplicado y da de alta la asignatura asignándole un identificador único; muestra un mensaje indicando que se ha realizado el alta.

Cursos alternos

Caso B:

3B. El administrador olvida introducir datos en alguno de los campos y pulsa el botón *Añadir.*

4B. El sistema muestra un mensaje indicando al usuario que debe rellenar todos los campos.

Caso C:

3C. El administrador introduce los datos de la nueva asignatura, pero ya existe una asignatura con el nombre introducido y pulsa el botón Añadir.

4C. El sistema muestra un mensaje indicando que ya existe una asignatura con ese nombre.

ALTA DE UN CURSO

Actores:	Administrador.
Propósito:	Dar de alta en el sistema un nuevo curso.
Resumen:	El administrador introduce el número y la letra del curso. El sistema registra el nuevo curso.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador accede mediante el botón *Añadir Curso*.
2. El sistema muestra el formulario que debe rellenarse con los datos del curso.
3. El administrador introduce todos los datos y pulsa el botón *Añadir*.
4. El sistema comprueba que no exista un curso con el mismo número y letra y da de alta el curso asignándole un identificador único; muestra un mensaje indicando que se ha realizado el alta.

Cursos alternos

Caso B:

3B. El administrador olvida introducir datos en alguno de los campos y pulsa el botón *Añadir.*

4B. El sistema muestra un mensaje indicando al usuario que debe rellenar todos los campos.

Caso C:

3C. El administrador introduce los datos del nuevo curso, pero ya existe un curso con el número y letra introducidos y pulsa el botón Añadir.

4C. El sistema muestra un mensaje indicando que ya existe un curso con ese número y letra.

ALTA DE UN PROFESOR

Actores:	Administrador.
Propósito:	Dar de alta en el sistema un nuevo profesor.
Resumen:	El administrador introduce nombre y apellidos del profesor. El sistema registra un nuevo usuario y un nuevo profesor con identificador de ese usuario.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador accede mediante el botón *Añadir Profesor*.
2. El sistema muestra el formulario que debe rellenarse con los datos del profesor.
3. El administrador introduce nombre y apellidos y pulsa el botón *Insertar y añadir cursos y asignaturas*.
4. El sistema registra un nuevo usuario con nombre creado automáticamente a partir del nombre y apellidos del profesor, sin clave, con identificador de perfil correspondiente a tipo profesor y con un identificador de usuario único.
5. El sistema registra un nuevo profesor con el nombre y apellidos introducidos, con identificador de usuario del usuario creado en el paso anterior y con un identificador de profesor único. El sistema muestra un mensaje indicando que el profesor ha sido creado.
6. El administrador pulsa el botón *Insertar y añadir cursos y asignaturas.*
7. El sistema muestra el formulario que debe rellenarse para introducir los cursos y asignaturas que imparte en profesor.
8. El administrador escoge de cada campo un curso y una asignatura existente de un combo box.

Cursos alternos

Caso B:

3B. El administrador olvida introducir datos en alguno de los campos y pulsa el botón *Añadir.*

4B. El sistema muestra un mensaje indicando al usuario que debe rellenar todos los campos.

Caso C:

6C. El administrador no pulsa el botón *Insertar y añadir cursos y asignaturas.*

7C. El administrador habrá creado un profesor sin asignarle cursos y asignaturas.

ALTA DE UN PADRE

Actores:	Administrador.
Propósito:	Dar de alta en el sistema un nuevo padre.
Resumen:	El administrador introduce los datos del padre. El sistema registra un nuevo usuario y un nuevo padre con identificador de ese usuario.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador accede mediante el botón *Añadir Padre*.
2. El sistema muestra el formulario que debe rellenarse con los datos del padre.
3. El administrador introduce nombre, apellidos, dirección y teléfono y pulsa el botón *Añadir.*
4. El sistema registra un nuevo usuario con nombre creado automáticamente a partir del nombre y apellidos del padre, sin clave, con identificador de perfil correspondiente a tipo padre y con un identificador de usuario único.
5. El sistema registra un nuevo padre con los datos introducidos, con identificador de usuario del usuario creado en el paso anterior y con un identificador de padre único. El sistema muestra un mensaje indicando que el padre ha sido creado.

Cursos alternos

Caso B:

3B. El administrador olvida introducir datos en alguno de los campos y pulsa el botón *Añadir.*

4B. El sistema muestra un mensaje indicando al usuario que debe rellenar todos los campos.

ALTA DE UN ALUMNO

Actores:	Administrador.
Propósito:	Dar de alta en el sistema un nuevo alumno.
Resumen:	El administrador introduce los datos del alumno. El sistema registra un nuevo alumno.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador accede mediante el botón *Añadir Alumno*.
2. El sistema muestra el formulario que debe rellenarse con los datos del alumno.
3. El administrador introduce nombre, apellidos, fecha de nacimiento y padre del alumno de un combo box con todos los padres de la base de datos y pulsa el botón *Añadir.*
4. El sistema registra un nuevo alumno con los datos introducidos y un identificador de alumno único.
5. El administrador pulsa el botón *Insertar alumno y añadir profesores.* El sistema muestra un mensaje indicando que el alumno ha sido creado.
6. El administrador escoge un profesor de una lista con todos los profesores en un combo box y pulsa el botón *Añadir*. El sistema muestra un mensaje indicando que el profesor ha sido añadido.

Cursos alternos

Caso B:

3B. El administrador olvida introducir datos en alguno de los campos y pulsa el botón *Añadir.*

4B. El sistema muestra un mensaje indicando al usuario que debe rellenar todos los campos.

Caso C:

6C. El administrador no pulsa el botón *Insertar alumno y añadir profesores.*

7C. El administrador habrá creado un alumno sin asignarle profesores.

BAJA DE UN ALUMNO

Actores:	Administrador.
Propósito:	Dar de baja un alumno.
Resumen:	El administrador selecciona el alumno de una lista de alumnos y pulsa el botón *Eliminar*. El sistema da de baja al alumno.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador pulsa el botón *Eliminar y editar datos.*
2. El sistema muestra una pantalla con distintas pestañas dependiendo del ítem que se quiera eliminar o editar de la base de datos.
3. El administrador pulsa en la pestaña de alumnos y escoge de una lista ordenada por el identificador, el alumno a eliminar.
4. El administrador pulsa el botón *Eliminar* y el sistema muestra un mensaje indicando que el alumno ha sido eliminado.

Cursos alternos

Caso B:

3B. El administrador no escoge un alumno de la lista.

4B. El administrador pulsa el botón *Eliminar* y el sistema muestra un mensaje indicando que debe escoger un alumno de la lista para poder llevar a cabo la baja del alumno.

BAJA DE UN PADRE

Actores:	Administrador.
Propósito:	Dar de baja un padre.
Resumen:	El administrador selecciona el padre de una lista de padres y pulsa el botón *Eliminar*. El sistema comprueba que no tenga alumnos asociados a su identificador de padre. El sistema da de baja al padre.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador pulsa el botón *Eliminar y editar datos.*
2. El sistema muestra una pantalla con distintas pestañas dependiendo del ítem que se quiera eliminar o editar de la base de datos.
3. El administrador pulsa en la pestaña de padres y escoge de una lista ordenada por el identificador, el padre a eliminar.
4. El administrador pulsa el botón *Eliminar.*
5. El sistema comprueba que no tenga alumnos asociados a su identificador de padre y lo elimina. El sistema muestra un mensaje indicando que el alumno ha sido dado de baja.

Cursos alternos

Caso B:

3B. El administrador no escoge un padre de la lista.

4B. El administrador pulsa el botón *Eliminar* y el sistema muestra un mensaje indicando que debe escoger un padre de la lista para poder llevar a cabo la baja del padre.

Caso C:

5C. El sistema muestra un mensaje indicando que el alumno no ha sido dado de baja porque tiene alumnos asignados

BAJA DE UN PROFESOR

Actores:	Administrador.
Propósito:	Dar de baja un profesor.
Resumen:	El administrador selecciona el profesor de una lista de profesores y pulsa el botón *Eliminar*. El sistema comprueba que no tenga alumnos asociados a su identificador de profesor. El sistema da de baja al profesor.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador pulsa el botón *Eliminar y editar datos.*
2. El sistema muestra una pantalla con distintas pestañas dependiendo del ítem que se quiera eliminar o editar de la base de datos.
3. El administrador pulsa en la pestaña de profesores y escoge de una lista ordenada por el identificador, el profesor a eliminar.
4. El administrador pulsa el botón *Eliminar.*
5. El sistema comprueba que no tenga alumnos asociados a su identificador de profesor y elimina el profesor. El sistema muestra un mensaje indicando que el profesor ha sido dado de baja.

Cursos alternos

Caso B:

3B. El administrador no escoge un profesor de la lista.

4B. El administrador pulsa el botón *Eliminar* y el sistema muestra un mensaje indicando que debe escoger un profesor de la lista para poder llevar a cabo la baja del profesor.

Caso C:

5C. El sistema muestra un mensaje indicando que el profesor no ha sido dado de baja porque tiene alumnos asignados

BAJA DE UNA ASIGNATURA

Actores:	Administrador.
Propósito:	Eliminar una asignatura.
Resumen:	El administrador selecciona la asignatura de una lista de asignaturas y pulsa el botón *Eliminar*. El sistema comprueba que no tenga profesores o notas asociadas a su identificador de asignatura. El sistema elimina la asignatura.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador pulsa el botón *Eliminar y editar datos.*
2. El sistema muestra una pantalla con distintas pestañas dependiendo del ítem que se quiera eliminar o editar de la base de datos.
3. El administrador pulsa en la pestaña de asignaturas y escoge de una lista ordenada por el identificador, la asignatura a eliminar.
4. El administrador pulsa el botón *Eliminar.*
5. El sistema comprueba que no tenga profesores ni notas asociadas a su identificador de asignatura y elimina la asignatura. El sistema muestra un mensaje indicando que la asignatura ha sido eliminada.

Cursos alternos

Caso B:

3B. El administrador no escoge un profesor de la lista.

4B. El administrador pulsa el botón *Eliminar* y el sistema muestra un mensaje indicando que debe escoger un profesor de la lista para poder llevar a cabo la baja del profesor.

Caso C:

5C. El sistema muestra un mensaje indicando que la asignatura no ha sido eliminada porque tiene profesores o notas asignadas.

BAJA DE UN CURSO

Actores:	Administrador.
Propósito:	Eliminar un curso.
Resumen:	El administrador selecciona el curso de una lista de cursos y pulsa el botón *Eliminar*. El sistema comprueba que no tenga profesores o notas asociados a su identificador de curso. El sistema elimina el curso.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador pulsa el botón *Eliminar y editar datos.*
2. El sistema muestra una pantalla con distintas pestañas dependiendo del ítem que se quiera eliminar o editar de la base de datos.
3. El administrador pulsa en la pestaña de cursos y escoge de una lista ordenada por el identificador, el curso a eliminar.
4. El administrador pulsa el botón *Eliminar.*
5. El sistema comprueba que no tenga profesores ni notas asociadas a su identificador de curso y elimina el curso. El sistema muestra un mensaje indicando que el curso ha sido eliminado.

Cursos alternos

Caso B:

3B. El administrador no escoge un curso de la lista.

4B. El administrador pulsa el botón *Eliminar* y el sistema muestra un mensaje indicando que debe escoger un curso de la lista para poder llevar a cabo la baja del curso.

Caso C:

5C. El sistema muestra un mensaje indicando que el curso no ha sido eliminado porque tiene profesores o notas asignadas.

MODIFICACIÓN DE UN ALUMNO

Actores:	Administrador.
Propósito:	Modificar un alumno.
Resumen:	El administrador selecciona el alumno de una lista de alumnos y pulsa el botón *Editar*. El sistema visualiza los datos del alumno, el administrador modifica los datos.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador pulsa el botón *Eliminar y editar datos.*
2. El sistema muestra una pantalla con distintas pestañas dependiendo del ítem que se quiera eliminar o editar de la base de datos.
3. El administrador pulsa en la pestaña de alumnos y escoge de una lista ordenada por el identificador, el alumno a modificar.
4. El administrador pulsa el botón *Editar* y el sistema muestra una nueva pantalla con los datos del alumno.
5. El administrador modifica los datos y pulsa el botón *Modificar datos*.
6. El sistema comprueba los datos introducidos y realiza las modificaciones mostrando un mensaje.

Cursos alternos

Caso B:

3B. El administrador no escoge un alumno de la lista.

4B. El administrador pulsa el botón *Editar* y el sistema muestra un mensaje indicando que debe escoger un alumno de la lista para poder llevar a cabo la modificación del alumno.

MODIFICACIÓN DE UN PADRE

Actores:	Administrador.
Propósito:	Modificar un padre.
Resumen:	El administrador selecciona el padre de una lista de padres y pulsa el botón *Editar*. El sistema visualiza los datos del padre, el administrador modifica los datos.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador pulsa el botón *Eliminar y editar datos.*
2. El sistema muestra una pantalla con distintas pestañas dependiendo del ítem que se quiera eliminar o editar de la base de datos.
3. El administrador pulsa en la pestaña de padres y escoge de una lista ordenada por el identificador, el padre a modificar.
4. El administrador pulsa el botón *Editar* y el sistema muestra una nueva pantalla con los datos del padre.
5. El administrador modifica los datos y pulsa el botón *Modificar datos*.
6. El sistema comprueba los datos introducidos y realiza las modificaciones mostrando un mensaje.

Cursos alternos

Caso B:

3B. El administrador no escoge un padre de la lista.

4B. El administrador pulsa el botón *Editar* y el sistema muestra un mensaje indicando que debe escoger un padre de la lista para poder llevar a cabo la modificación del padre.

MODIFICACIÓN DE UN PROFESOR

Actores:	Administrador.
Propósito:	Modificar un profesor.
Resumen:	El administrador selecciona el profesor de una lista de profesores y pulsa el botón *Editar*. El sistema visualiza los datos del profesor, el administrador modifica los datos.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador pulsa el botón *Eliminar y editar datos.*
2. El sistema muestra una pantalla con distintas pestañas dependiendo del ítem que se quiera eliminar o editar de la base de datos.
3. El administrador pulsa en la pestaña de profesores y escoge de una lista ordenada por el identificador, el profesor a modificar.
4. El administrador pulsa el botón *Editar* y el sistema muestra una nueva pantalla con los datos del profesor.
5. El administrador modifica los datos y pulsa el botón *Modificar datos*.
6. El sistema comprueba los datos introducidos y realiza las modificaciones mostrando un mensaje.

Cursos alternos

Caso B:

3B. El administrador no escoge un profesor de la lista.

4B. El administrador pulsa el botón *Editar* y el sistema muestra un mensaje indicando que debe escoger un profesor de la lista para poder llevar a cabo la modificación del profesor.

MODIFICACIÓN DE UNA ASIGNATURA

Actores:	Administrador.
Propósito:	Modificar una asignatura.
Resumen:	El administrador selecciona la asignatura de una lista de asignaturas y pulsa el botón *Editar*. El sistema visualiza los datos de la asignatura, el administrador modifica los datos.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador pulsa el botón *Eliminar y editar datos.*
2. El sistema muestra una pantalla con distintas pestañas dependiendo del ítem que se quiera eliminar o editar de la base de datos.
3. El administrador pulsa en la pestaña de asignaturas y escoge de una lista ordenada por el identificador, la asignatura a modificar.
4. El administrador pulsa el botón *Editar* y el sistema muestra una nueva pantalla con los datos de la asignatura.
5. El administrador modifica los datos y pulsa el botón *Modificar datos*.
6. El sistema comprueba que no exista ya una asignatura con ese nombre y realiza las modificaciones mostrando un mensaje.

Cursos alternos

Caso B:

3B. El administrador no escoge una asignatura de la lista.

4B. El administrador pulsa el botón *Editar* y el sistema muestra un mensaje indicando que debe escoger una asignatura de la lista para poder llevar a cabo la modificación de la asignatura.

Caso C:

6C. El sistema muestra un mensaje indicando que no se ha podido modificar porque ya existe una asignatura con ese nombre.

MODIFICACIÓN DE UN CURSO

Actores:	Administrador.
Propósito:	Modificar un curso.
Resumen:	El administrador selecciona el curso de una lista de cursos y pulsa el botón *Editar*. El sistema visualiza los datos del curso, el administrador modifica los datos.
Precondiciones:	El administrador está conectado al sistema.

Curso normal de los eventos:

1. El administrador pulsa el botón *Eliminar y editar datos.*
2. El sistema muestra una pantalla con distintas pestañas dependiendo del ítem que se quiera eliminar o editar de la base de datos.
3. El administrador pulsa en la pestaña de cursos y escoge de una lista ordenada por el identificador, el curso a modificar.
4. El administrador pulsa el botón *Editar* y el sistema muestra una nueva pantalla con los datos del curso.
5. El administrador modifica los datos y pulsa el botón *Modificar datos*.
6. El sistema comprueba que no haya un curso con el mismo número y letra introducidos y realiza las modificaciones mostrando un mensaje.

Cursos alternos

Caso B:

3B. El administrador no escoge un curso de la lista.

4B. El administrador pulsa el botón *Editar* y el sistema muestra un mensaje indicando que debe escoger un curso de la lista para poder llevar a cabo la modificación del curso.

Caso C:

6C. El sistema muestra un mensaje indicando que no se ha podido modificar porque ya existe un curso con ese número y letra.

ALTA DE UNA NOTA

Actores:	Profesor.
Propósito:	Dar de alta en el sistema una nueva nota.
Resumen:	El profesor escoge un alumno e introduce los datos de la nota. El sistema registra una nueva nota.
Precondiciones:	El profesor está conectado al sistema.

Curso normal de los eventos:

1. El profesor elige un alumno de una lista con todos los alumnos ordenados por identificador y pulsa el botón *Siguiente.*
2. El sistema muestra una pantalla con el botón *Notas* y el botón *Chat.*
3. El profesor pulsa el botón *Notas*.
4. El profesor introduce los datos de la nota y comprueba que no haya una nota con los mismos datos y pulsa el botón *Añadir.*
5. El sistema registra una nueva nota con los datos introducidos y un identificador de nota único. También se introduce esa nota en la lista de notas del alumno.

Cursos alternos

Caso B:

4B. El profesor olvida introducir datos en alguno de los campos y pulsa el botón *Añadir.*

5B. El sistema muestra un mensaje indicando al usuario que debe rellenar todos los campos.

Caso C:

4C. El sistema comprueba los datos y muestra un mensaje indicando que ya existe una nota con esos datos para ese mismo alumno y no introduce la nota.

CONSULTA DE LAS NOTAS

Actores:	Padre.
Propósito:	Consultar las notas de su hijo.
Resumen:	El padre escoge un alumno de los que constan como hijos suyos y obtiene una lista de sus notas.
Precondiciones:	El padre está conectado al sistema.

Curso normal de los eventos:

1. El padre elige un alumno de una lista de los que constan como hijos suyos y pulsa el botón *Siguiente.*
2. El sistema muestra una pantalla con el botón *Notas* y el botón *Chat.*
3. El padre pulsa el botón *Notas*.
4. El sistema muestra un listado con los datos de las notas de su hijo.

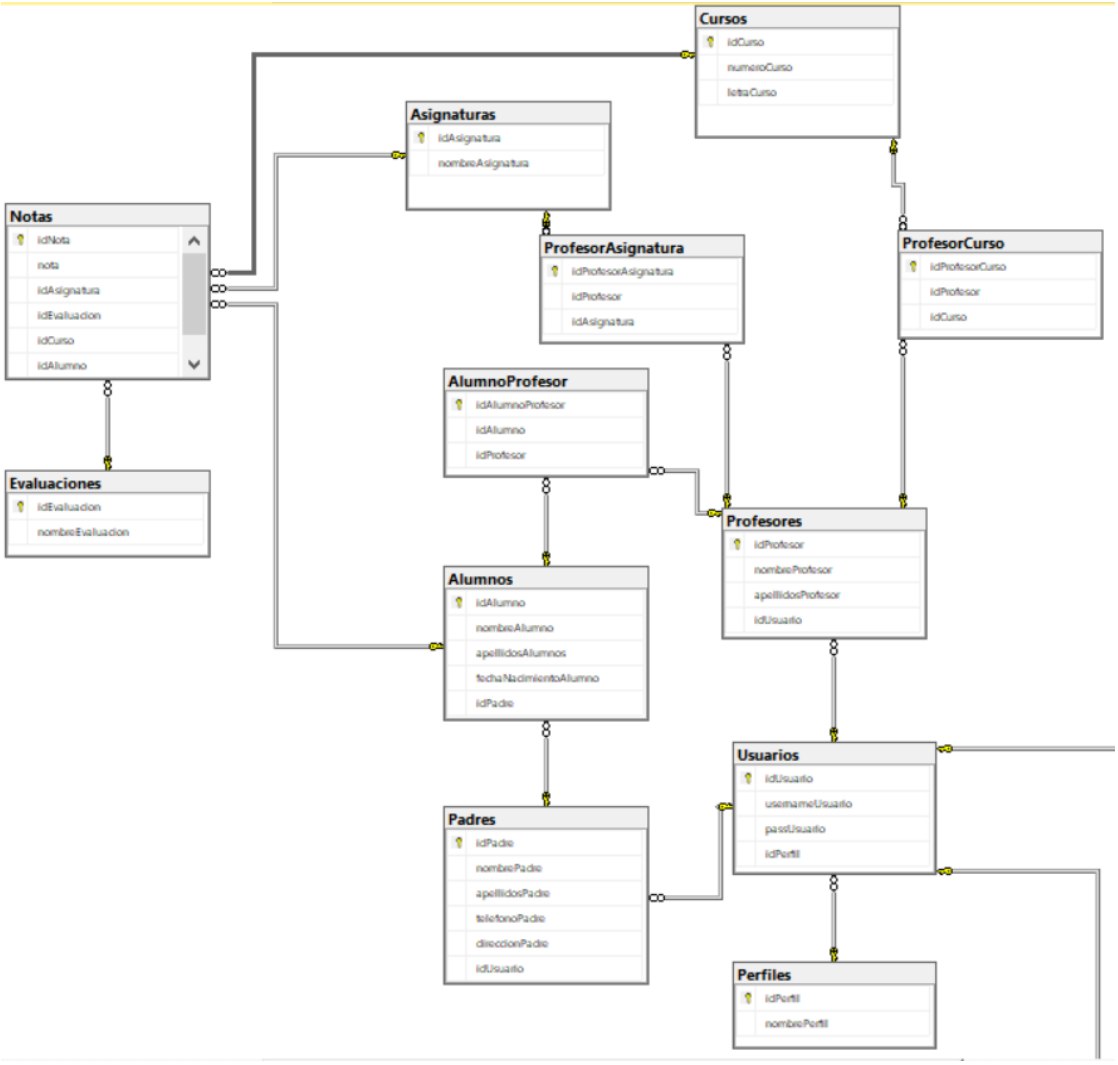
Cursos alternos

Caso B:

1B. El padre olvida escoger un alumno de la lista y pulsa el botón *Notas*.

2B. El sistema muestra un mensaje indicando al usuario que debe escoger un alumno.

5. ESTRUCTURA DE LA BASE DE DATOS



En la estructura principal de la BBDD tiene 12 tablas.

- Primero tenemos las tablas principales para la gestión del login y los usuarios de la app:

Usuarios: Usamos esta tabla principalmente para el Login. Cada vez que se crea un Padre o un Profesor también se crea su usuario.

Perfiles: con idPerfil (clave primaria y autonumérica) y nombrePerfil (PADRE, PROFESOR o ADMIN por defecto).

Padres: En esta tabla tendremos la información de los padres. Cada padre tendrá un idUsuario asociado.

Profesores: En esta tabla tendremos la información de los profesores. Cada profesor tendrá un idUsuario asociado.

Alumnos: En esta tabla tendremos la información de los alumnos que tendrán un idPadre asociado.

- Después podemos ver tablas que se usan para relaciones muchos a muchos como son:

ProfesorAsignatura: Tabla que contiene un idProfesor y un idAsignatura, ya que un profesor puede dar varias asignaturas y una asignatura puede ser dada por varios profesores. Así se relaciona la tabla **Profesores** con la tabla **Asignaturas**

ProfesorCurso: Tabla que contiene un idProfesor y un idCurso, ya que un profesor puede dar en varios cursos y un curso puede tener varios profesores dando clase. Así se relaciona la tabla **Profesores** con la tabla **Asignaturas**

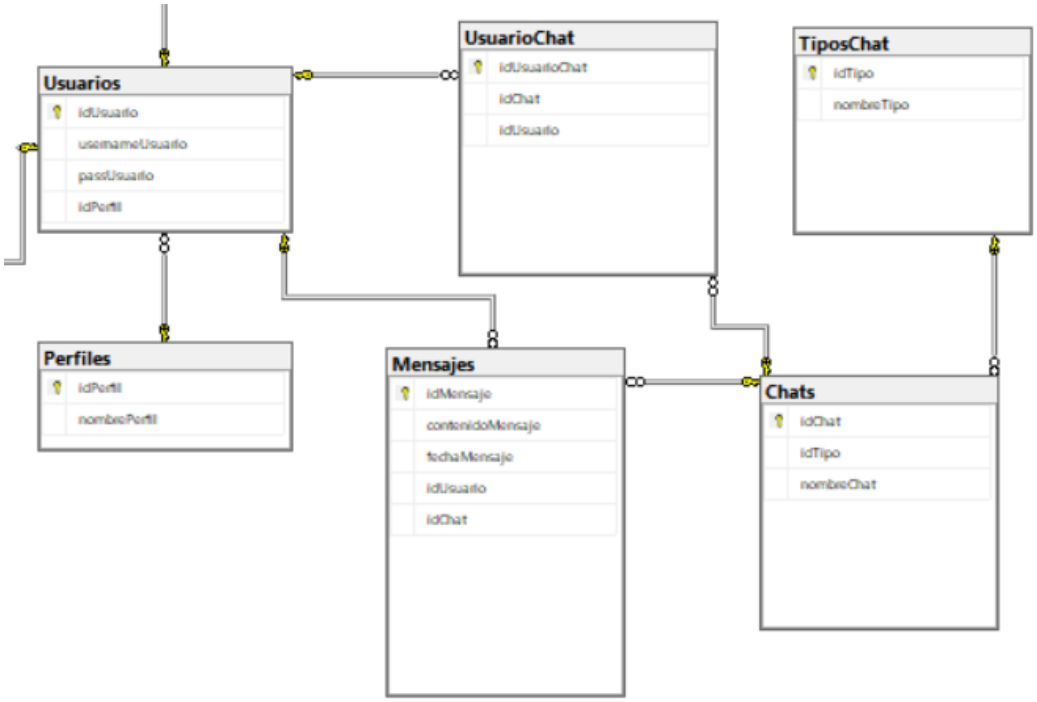
AlumnoProfesor: Tabla que contiene un idAlumno y un idProfesor, ya que un profesor puede dar clase a varios alumnos y un alumno puede recibir clase de varios profesores. Asi se relaciona la tabla **Alumnos** con la tabla **Profesores**

- Por otro lado, tenemos la parte de las notas:

Notas: En esta tabla va la nota, que es un comentario del tipo “Progresa adecuadamente” o “Necesita esforzarse más” y después los id de asignatura, curso, alumno y evaluación.

Evaluaciones: Con el idEvaluacion y el nombre de la evaluación, que vendrán por defecto en la base de datos y serán 1, 2 y 3.

- Por último tenemos otra serie de tablas que están en la base de datos pero que todavía no hay funciones en la app que las utilicen, pero que se podrían implementar en una próxima versión. Estas tablas son las tablas de chat.

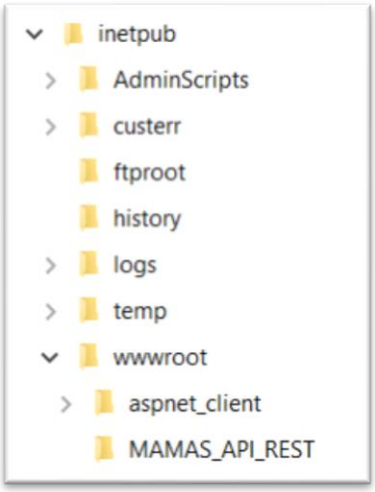


Chats: Esta tabla sería para las distintas salas de chat con su idTipo que la relaciona con **Tipos**, los cuales pueden ser chat privado o chat grupal.

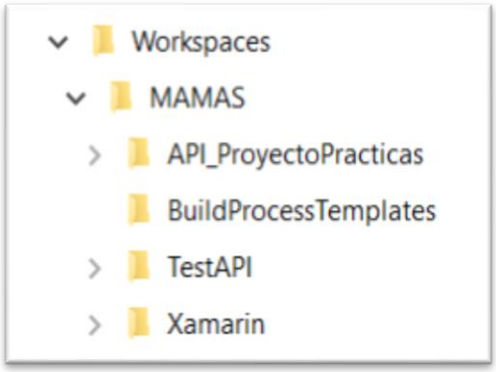
UsuarioChat es la tabla que relaciona a los usuarios con los chats ya que en un chat hay varios usuarios y un usuario puede estar en varios chats.

Mensajes, por último, es la tabla donde irían guardados todos los mensajes que se escriben en la app, relacionados con su sala de grupo y con el usuario que lo escribe.

6. ESTRUCTURA DEL SOFTWARE

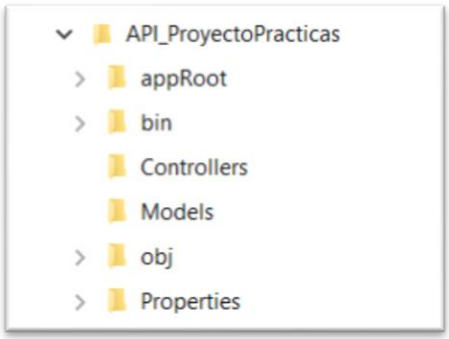


Antes de empezar con los directorios del directo, cabe destacar la localización de la API REST que se utiliza para publicarla en un servidor IIS. En C: encontramos el directorio inetpub\wwwroot que es generalmente el que se usa para guardar los servidores. Aquí es donde se guarda la carpeta MAMAS_API_REST, con todos los ficheros de configuración necesarios para poder conectarse a la API.



Ahora si, en cuanto a los directorios del proyecto en sí, tenemos como directorio raíz **Workspaces**. El directorio de **Workspaces** se crea cuando creas un área de trabajo en dev.azure.com, en este caso nuestro grupo de trabajo está en https://dev.azure.com/dmgproyectos/MAMAS/, siendo **MAMAS** el directorio que se va a crear dentro de **Workspaces**. Hay dos directorios (o proyectos) principales, uno para la API, **API_ProyectoPracticas**, y otro para la interfaz gráfica y las funciones que recogen la información de las funciones de la API, que devuelven información en formato JSON, **Xamarin**. Por otro lado, tenemos otro directorio con un proyecto de prueba, TestAPI. **BuildProcessTemplates** es el directorio que crea azure para poder compartir el proyecto entre los miembros del mismo grupo de trabajo.

7.1. El directorio de la API (API_ProyectoPracticas)

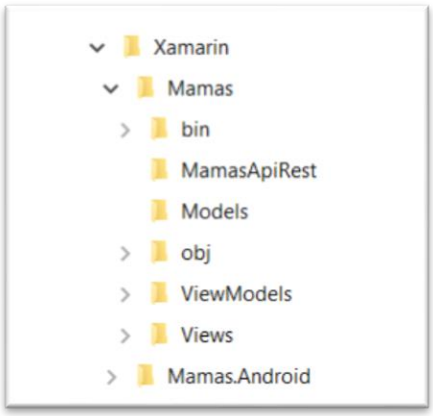


Está subdividido a su vez en los **controladores** y los **modelos**. Los modelos son creados a partir de la base de datos en SQL Server utilizando un comando de la consola de .Net Core. Estos modelos son clases cuyos atributos representan los campos y relaciones de las distintas tablas de la base de datos. Cuando se ejecuta ese comando, se crea también la clase MAMASContext.cs, que contiene la conexión a la base de datos y relaciona o contextualiza los nombres de los campos de las tablas y sus relaciones, con los atributos de las clases de los modelos.

En este directorio se encuentra también la clase **Startup.cs**, que es la que contiene el código relacionado con la herramienta Swashbuckle para crear la página web de la API, donde se explica su funcionalidad, los parámetros de las funciones y en resumen todo lo necesario para poder utilizarla, es decir, un manual de usuario de la API.

7.2. El directorio de la Interfaz Gráfica de la app, (directorio Xamarin)

Dentro del proyecto de Xamarin nos encontramos con Mamas. Que es el proyecto con código compartido. Se divide en los siguiente directorios:



7.2.1. MAMASAPIREST: Es el directorio que contiene las clases para recoger la información de las funciones de la API, que devuelven información en formato JSON.

La clase más importante de este submódulo es **MamasApi.cs**, que contiene toda la configuración que ejecuta correctamente la llamada a la API: valida que la URL esté bien formada, prepara la petición certificando que los parámetros correspondan a los necesarios para una determinada función de la API, establece qué códigos de respuesta serán calificadas como Error (en nuestro caso 400 como error general y 404 cuando no se encuentra un ítem), establece protocolos de seguridad y lo más importante: contiene las funciones que van a usarse para transformar la información recogida en formato JSON al tipo de objeto que se les indique. Hay un total de 8 funciones, que dependen del tipo (GET, POST, PUT o DELETE) y de si devuelven un objeto individual o un array de objeto.

En este directorio se encuentran también las clases que contienen las funciones encargadas de hacer la llamada a su correspondiente función de la API y recoger su información utilizando las funciones de **MamasApiRest.cs**, con nombres como: MamasApiAlumnos.cs, MamasApiProfesores.cs, que en realidad forman parte de MamasApi ya que son Partial Class, es decir en realidad son la misma clase pero separada en distintos archivos .cs ya que si no sería demasiado grande y así es más fácil de organizar. Cada una de estas clases contiene las funciones

encargadas de hacer la llamada a su correspondiente función de la API y recoger su información.

Hay otras dos clases importantes en este directorio que son plantillas para recoger la información de la API, de las cuales heredarán las clases del submódulo Models, y que son:

a. **MamasApiRestResponseBase.cs**: Contiene atributos comunes a todas las clases del modelo de datos de este módulo, siendo el más importante el atributo JSON, que es un string con el JSON devuelto por la función de la API. Serán las funciones de MamasApi.cs las encargadas de transformar este JSON en el objeto del tipo indicado.
b. **MamasApiRestResponseArrayBase.cs**: Para cuando hay que recoger varios objetos que heredan de MamasApiRestResponseBase.cs.

7.2.2. MODELS:

Están las mismas clases que los modelos del módulo de la API, con la única diferencia que heredan de la clase MamasApiRestResponseBase. También están las clases que heredan de MamasApiRestResponseArrayBase, para cuando una función de la API devuelve un array.

7.2.3. VIEWMODELS:

Aquí nos encontramos las clases que van a trabajar con los datos de los modelos y que van a estar conectadas con las vistas, a través de bindings, siguiendo el patrón de diseño MVVM. Así por ejemplo tendríamos un LoginViewModel.cs que es el encargado de coger los datos del usuario de los models, hacer lo que sea con ellos y comunicar los cambios a las vistas, en este caso a LoginPage.xaml. Para mayor organización en la raíz se han dejado los viewmodels de profesores y padres y en una carpeta los del administrador

7.2.4. VIEWS

Aquí nos encontramos los archivos .xaml que serán las vistas de nuestra app. Todo el código que trabaja sobre estas vistas estará en los view models correspondientes. Para mayor organización en la raíz se han dejado las views de profesores y padres y en una carpeta las del administrador

7.2.5. MAMAS.ANDROID

A parte de Mamas, tenemos Mamas.Android. En este directorio tenemos el proyecto de Android, que en nuestro caso no tiene nada de código ya que hemos podido hacer todo con código compartido.

8. DISEÑO

En cuanto al diseño se podría decir que, si tenemos en cuenta el conjunto de proyectos, el patrón de diseño se asemeja al patrón MVC, ya que tenemos los modelos y los controladores en el proyecto de la API y las vistas podrían ser la app móvil en general.

Un ejemplo de modelo sería el siguiente:

```
1. namespace API_ProyectoPracticas.Models
2. {
3.     public class Alumnos
4.     {
5.         public Alumnos()
6.         {
7.             AlumnoProfesor = new HashSet<AlumnoProfesor>();
8.             Notas = new HashSet<Notas>();
9.         }
10.
11.              public int IdAlumno { get; set; }
12.              public string NombreAlumno { get; set; }
13.              public string ApellidosAlumnos { get; set; }
14.              public DateTime FechaNacimientoAlumno { get; set; }
15.              public int IdPadre { get; set; }
16.
17.              public virtual Padres
   IdPadreNavigation { get; set; }
18.              public virtual ICollection<AlumnoProfesor> AlumnoPr
   ofesor { get; set; }
19.              public virtual ICollection<Notas> Notas { get; set;
    }
20.          }
21.      }
```

Y su controlador (solo algunas funciones porque si no quedaría muy largo):

```
1. namespace API_ProyectoPracticas.Controllers
2. {
3.     [Produces("application/json")]
4.     [Route("api/[controller]")]
5.     [ApiController]
6.     public class AlumnosController : ControllerBase
7.     {
8.         /// <summary>
9.         /// Devuelve los datos de todos los alumnos
10.              /// </summary>
11.              /// <returns>El JSON de todos los alumnos</returns>
```

```
12.             /// <response code="200">Si existen
    alumnos</response>
13.             /// <response code="404">Si no existen
    alumnos</response>
14.             [HttpGet()]
15.             public IActionResult Get()
16.             {
17.                 MAMASContext context = new MAMASContext();
18.
19.                 if (!context.Alumnos.Any())
20.                 {
21.                     var details = new ProblemDetails()
22.                     {
23.                         Title = "No hay alumnos en la base de
    datos",
24.                         Detail = $"No hay alumnos en la tabla
    Alumnos de la base de datos",
25.                         Status = 404
26.                     };
27.                     return new ObjectResult(details)
28.                     {
29.                         ContentTypes = { "application/problem+j
    son" },
30.                         StatusCode = 404,
31.                     };
32.                 }
33.                 else
34.                     return Ok(context.Alumnos);
35.             }
36.
37.             /// <summary>
38.             /// Devuelve los alumnos de un mismo padre
39.             /// </summary>
40.             /// <param name="idPadre"> El identificador del
    padre</param>
41.             /// <returns>El JSON de los alumnos</returns>
42.             /// <response code="200">Si el padre existe y tiene
    alumnos hijos</response>
43.             /// <response code="404">Si el padre no
    existe</response>
44.             /// <response code="400">Si no tiene alumnos
    hijos</response>
45.             [HttpGet("Padre/{idPadre}")]
46.             /// <summary>
47.             /// Busca un Alumno
48.             /// </summary>
49.             /// <param name="idAlumno"> El id del
    alumno</param>
50.             /// <param name="aux"> Parámetro auxiliar</param>
51.             /// <returns>El JSON del alumno</returns>
52.             /// <response code="200">Si el alumno
    existe</response>
53.             /// <response code="404">Si el alumno no
    existe</response>
54.             [HttpGet("Alumno/{idAlumno}")]
55.             public IActionResult GetAlumno(int idAlumno)
56.             {
57.                 MAMASContext context = new MAMASContext();
58.                 var alumno = context.Alumnos.SingleOrDefault(p
    => p.IdAlumno == idAlumno);
59.                 if (alumno == null)
```

```
60.                 {
61.                     var details = new ProblemDetails()
62.                     {
63.                         Title = "No existe el alumno",
64.                         Detail = $"No existe el alumno de id:
   {idAlumno}",
65.                         Status = 404
66.                     };
67.                     return new ObjectResult(details)
68.                     {
69.                         ContentTypes = { "application/problem+j
   son" },
70.                         StatusCode = 404,
71.                     };
72.                 }
73.                 else
74.                 {
75.                     return Ok(alumno);
76.                 }
77.             }
78.
79.             /// <summary>
80.             /// Crea un alumno
81.             /// </summary>
82.             /// <param name="nombreAlumno"> El nombre del
   alumno</param>
83.             /// <param name="apellidosAlumno"> Los apellidos
   del alumno</param>
84.             /// <param name="fechaNacimiento"> La fecha de
   nacimiento del alumno</param>
85.             /// <param name="idPadre"> El identificador del
   padre del alumno</param>
86.             /// <returns>El JSON del alumno</returns>
87.             /// <response code="201">Si el alumno ha sido
   creado</response>
88.             [HttpPost("{nombreAlumno}/{apellidosAlumno}/{fechaN
   acimiento}/{idPadre}")]
89.             public IActionResult
   Post(string nombreAlumno, string apellidosAlumno, DateTime
   fechaNacimiento, int idPadre)
90.             {
91.                 MAMASContext context = new MAMASContext();
92.
93.                 Alumnos alumno = new Alumnos();
94.                 alumno.NombreAlumno = nombreAlumno;
95.                 alumno.ApellidosAlumnos = apellidosAlumno;
96.                 alumno.FechaNacimientoAlumno = fechaNacimiento;
97.                 alumno.IdPadre = idPadre;
98.                 context.Alumnos.Add(alumno);
99.                 context.SaveChanges();
100.                return CreatedAtRoute("", new
101.                {
102.                    nombreAlumno = alumno.NombreAlumno,
103.                    apellidosAlumno = alumno.ApellidosAlumnos,
104.                    fechaNacimiento = alumno.FechaNacimientoAlu
   mno,
105.                    idPadre = alumno.IdPadre
106.                }, alumno);
107.            }
108.
109.
```

Ahora pasamos a ver como se usan estos controladores y modelos en la app.

Ya dentro de la app, para pasar esos modelos a vistas, se puede observar que se utiliza un patrón de diseño **MVVM** (Model, View, ViewModel) en el que en los ViewModel se trabaja con los datos de los Model y se manejan esos datos utilizando las funciones de los Controller o controladores, aislando completamente el código de las vistas, para así solo tener que comunicar los cambios a las vistas a través de Bindings.

Como se ha explicado antes, para recoger las funciones de los Controller de la API utilizamos la clase MamasAPI y sus respectivas partial class del directorio MamasApiRest del proyecto Mamas. Una clase de ejemplo sería:

```
1. namespace Mamas.MamasApiRest
2. {
3.     public partial class MamasApi
4.     {
5.         public ArrayAlumnos AlumnosGet()
6.         {
7.             var result = (ArrayAlumnos)null;
8.             var wsMethod = "Get()";
9.             var uri = $"Alumnos";
10.
11.                 result = CallWebServiceGetCommon<ArrayAlumnos,
   Alumnos>(uri, wsMethod);
12.                 return result;
13.             }
14.
15.             public ArrayAlumnos AlumnosGet(int idPadre)
16.             {
17.                 var result = (ArrayAlumnos)null;
18.                 var wsMethod = "Get()";
19.                 var uri = $"Alumnos/Padre/{idPadre}";
20.
21.                 result = CallWebServiceGetCommon<ArrayAlumnos,
   Alumnos>(uri, wsMethod);
22.                 return result;
23.             }
24.
25.             public Alumnos AlumnoGet(int idAlumno)
26.             {
27.                 var result = (Alumnos)null;
28.                 var wsMethod = "Get()";
29.                 var uri = $"Alumnos/Alumno/{idAlumno}";
30.
31.                 result = CallWebServiceGetCommon<Alumnos>(uri,
   wsMethod);
32.                 return result;
33.             }
34.
35.             public Alumnos
   AlumnosPost(string nombreAlumno, string apellidosAlumno,
   DateTime fechaNacimiento, int idPadre)
36.             {
```

```
37.                 var result = (Alumnos)null;
38.                 var wsMethod = "Post()";
39.                 var uri = $"Alumnos/{nombreAlumno}/{apellidosAl
   umno}/{fechaNacimiento.ToShortDateString().Replace('/','-
   ')}/{idPadre} ";
40.
41.                 result = CallWebServicePostCommon<Alumnos>(uri,
   wsMethod, null);
42.                 return result;
43.             }
44.
45.             public Alumnos AlumnosDelete(int idAlumno)
46.             {
47.                 var result = (Alumnos)null;
48.                 var wsMethod = "Delete()";
49.                 var uri = $"Alumnos/{idAlumno}";
50.
51.                 result = CallWebServiceDeleteCommon<Alumnos>(ur
   i, wsMethod);
52.                 return result;
53.             }
54.
55.             public Alumnos
   AlumnosPut(int idAlumno, string nombreAlumno, string apellidosAl
   umno, DateTime fechaNacimiento, int idPadre)
56.             {
57.                 var result = (Alumnos)null;
58.                 var wsMethod = "Put()";
59.                 var uri = $"Alumnos/{idAlumno}/{nombreAlumno}/{
   apellidosAlumno}/{fechaNacimiento.ToShortDateString().Replace('/
   ','-')}/{idPadre}";
60.
61.                 result = CallWebServicePutCommon<Alumnos>(uri,
   wsMethod, null);
62.                 return result;
63.             }
64.
65.
66.
67.         }
68.     }
```

Después en los ViewModel nos conectaríamos a la API publicada en el servidor IIS, que está en la red local de nuestro ordenador:

```
1.             mamas = new MamasApi()
2.             {
3.                 Ruta = new Uri("https://huawei-david:8443/api"),
4.             };
```

Y utilizaríamos las funciones de las clases de MamasApiRest para comunicarnos con las funciones de la API. Por ejemplo:

```
var resp = mamas.AlumnosPost(NombreAlumno,ApellidosAlumno,
FechaNacimientoAlumno, PadreSeleccionado.IdPadre);
```

Siendo la variable resp el json devuelto por la función, que se mapeará a objeto y por lo tanto se podrá ver toda su información muy fácilmente.

En cuanto al paso de la información entre los ViewModel y las View tenemos los Binding. En el .xaml se hace un Binding por ejemplo a la propiedad Text de un Entry (equivalente a TextField)

```
<Entry x:Name="nombreAlumno"
                    Text="{Binding Path=NombreAlumno}"
                    Placeholder="Nombre del Alumno"/>
```

Después en el ViewModel debe haber una propiedad con el mismo nombre que hemos puesto en el Path del Binding

```
        public string NombreAlumno
        {
            get { return _nombreAlumno; }
            set { SetProperty(ref _nombreAlumno, value); }
        }
```

Y por último, haciendo uso del framework Prism, solo nos hará falta ir a app.cs y añadir esta línea al método `RegisterTypes`

```
containerRegistry.RegisterForNavigation<ConfAlumnosPage,
ConfAlumnosViewModel>();
```

Siendo los tipos del RegisterForNavigation<ClaseView, ClaseViewModel>

Con este patrón de diseño, queda un proyecto bien organizado y fácil de entender.

DAVID: 4 (REPASAR Y AMPLIAR SI SE PUEDE), 6, 7 (TERMINAR) Y 8

JORGE: CASOS DE USO

9. PRUEBAS

El plan de pruebas se ha llevado a cabo durante el desarrollo de la aplicación y una vez completado todo el desarrollo.

Durante el desarrollo las pruebas se realizaban con datos que se adaptaban a los requisitos del módulo que se estuviera implementando en cada momento. Estas

pruebas consistían en comprobar el tratamiento de la información por parte del módulo, y el almacenamiento y recuperación de la información en la base de datos.

Al finalizar todo el desarrollo se realizan pruebas siguiendo los flujos de información más habituales en la app y comprobando que toda la funcionalidad se lleva a cabo.

1.- Primero se necesita una cuenta de Usuario en la BBDD que sea administrador, además de meter los perfiles y las evaluaciones que serán datos por defecto

2.- Después el administrador podrá crear, por norma general, primero cursos y asignaturas, después padres y profesores y por último los alumnos asociados a esos padres y profesores.

3.- Una vez creados los datos, o según se van creando, el administrador podrá borrarlos (si las restricciones de la BBDD lo permiten) o modificarlos.

4.- Cuando los padres y profesores tengan creada su cuenta de usuario podrán logearse y elegir su contraseña

5.- Los profesores podrán elegir un alumno ya creado y ponerle sus notas

6.- Una vez puestas las notas, los padres podrán verlas.

La app se ha probado tanto en un emulador como en un dispositivo móvil físico (Huawei Mate 10) y la API se ha probado publicada en la red local en el servidor IIS del PC desde el emulador, desde el móvil y desde una aplicación de consola de prueba; y sin publicar, es decir, desde localhost teniendo que ejecutar el proyecto de la API cada vez que queramos usarla.

A continuación, el conjunto de pruebas realizadas en la app

LOGIN

- Introducir nombre de usuario correcto
- Introducir nombre de usuario incorrecto
- Elegir contraseña cuando se entra por primera vez
- Escribir contraseña correcta
- Escribir contraseña incorrecta

ADMINISTRADOR

- Insertar asignatura
- Insertar curso
- Insertar profesor
- Insertar padre
- Insertar alumno
- Insertar cualquier registro sin rellenar todos los campos de texto
- Modificar asignatura
- Modificar curso
- Modificar profesor
- Modificar padre
- Modificar alumno
- Insertar cualquier registro sin rellenar todos los campos de texto
- Eliminar asignatura
- Eliminar curso
- Eliminar profesor
- Eliminar padre
- Eliminar alumno
- Eliminar asignatura o curso con profesor/es asignado/s
- Eliminar profesor con alumno/s asignado/s
- Eliminar padre con alumno/s asignado/s

PROFESOR

- Intentar entrar sin elegir alumno
- Elegir un alumno y entrar
- Elegir módulo de notas
- Insertar nota
- Intentar insertar nota al mismo alumno, curso, asignatura y evaluación

PADRE

- Intentar entrar sin elegir hijo
- Elegir hijo y entrar

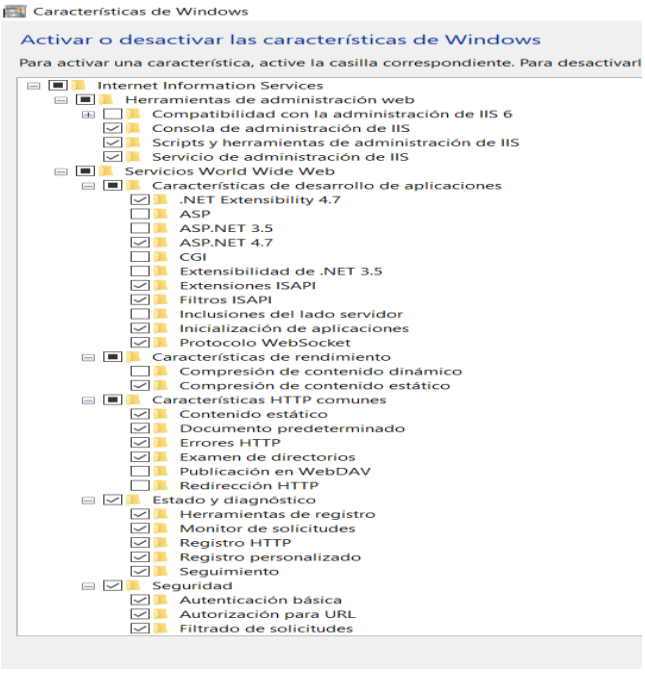
- Elegir módulo de notas, donde se deberían ver las notas puestas a ese alumno

10. MANUAL DE INSTALACIÓN

En cuanto a la instalación de la app, la única forma de instalarla ahora mismo, ya que no se ha generado apk aún, es conectar el móvil al PC donde se va a ejecutar el proyecto (el móvil debe tener la depuración USB activada). Por tanto será necesario Visual Studio.

También será necesario el script de la base de datos de donde la app va a coger la información. El entorno es Microsoft SQL Server Management Studio.

Por último, será necesario o bien el proyecto de la API y ejecutarlo cada vez que queramos usar la app, o publicar en nuestro PC un servidor IIS (para esto último necesitamos .NET con la última versión de SDK y runtime). Para poder montar el servidor, en *Características de Windows -> Activar o desactivar características de Windows* debes activar lo siguiente:



Después, accediendo al administrador de IIS (poniendo IIS en el buscador de Windows saldrá), creamos un nuevo sitio y lo configuramos con el nombre de dominio que queramos, http o https, puerto y el directorio donde queramos guardarlos (como norma general en *C:\inetpub\wwwroot*). Por último nos vamos a la API, click dercho en el proyecto y le damos a Publicar. Aquí igualmente configuramos a nuestro gusto y elegimos el directorio donde está nuestro sitio web.

Una vez instalado y configurado todo esto. El último paso será instalar la aplicación y conectar el WI-FI del móvil a la red local compartida por el ordenador, ya que el servidor en este caso está en la red de nuestro ordenador.

11. PRESUPUESTO

Teniendo en cuenta que en Guadalajara un proyecto con desarrollo y análisis cuesta en torno a 45/50€ la hora, y que el proyecto está previsto para unas 40h, el presupuesto inicial sería entre 1800 y 2000€.

12. AÑADIR MEJORAS FUTURAS:

En la base de datos hay tablas para una futura implementación de un servicio de mensajería mediante chats entre usuarios. Las tablas son: Chats, Mensajes, TiposChat y UsuarioChat. Las relaciones y la conversión a clases del modelo de datos ya están hechas.

Otra posible mejora que hubiésemos hecho con más tiempo, habría sido crear un tablón de anuncios, en el que los profesores pudiesen escribir mensajes, fotos y vídeos para todos los alumnos de una clase, un curso o del centro.

Por último, se podría mejorar el estilo de la interfaz gráfica, ya que el actual está enfocado simplemente a la funcionalidad de la aplicación.

13. BIBLIOGRAFÍA:

- https://docs.microsoft.com/es-es/
    - Documentación de Microsoft
- www.stackoverflow.com
    - Para consultar problemas específicos
- https://docs.microsoft.com/es-es/aspnet/core/tutorials/first-mvc-app/?view=aspnetcore-2.2
    - Tutorial para crear el MVC

- https://docs.microsoft.com/es-es/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-2.2&tabs=visual-studio
    - Para crear la web de la API
- https://www.lucidchart.com
    - Para crear el diagrama entidad relación de la base de datos
- https://www.entityframeworktutorial.net/efcore/create-model-for-existing-database-in-ef-core.aspx
    - Tutorial para crear las clases a partir de la base de datos ya creada, lo que se llama estructura DataBaseFirst
- https://www.connectionstrings.com/sql-server/
    - En esta página se puede ver la estructura de las cadenas de conexión para conectarse a la base de datos. En este caso a una base de datos SQLServer
- www.variablenotfound.com
    - Blog para ver ejemplos de cosas que van saliendo nuevas