# EDUCAPP

## Aplicación de gestión educativa

### Descripción breve

Propuesta de proyecto de fin de ciclo.
C.F.G.S. Desarrollo de Aplicaciones Multiplataforma

Harold Hormaechea Garcia
Harold.hormaechea@gmail.com

# Contenido

Introducción ..... 2
Propuesta técnica de implementación ..... 3
Servidor ..... 3
Cliente ..... 3
Bases de datos ..... 5
Tablas empleadas ..... 6
Usuarios ..... 6
Cursos ..... 6
Tutores por curso ..... 7
Matriculaciones en cursos ..... 7
Exámenes ..... 7
Control de presencia ..... 8
Registro de notificaciones: ..... 8
Registro de usuarios tutelados: ..... 9
Esquema de tablas ..... 10

# Introducción

Este proyecto propone la creación de una herramienta de gestión de centros de estudios de fácil manejo por parte tanto de responsables de centro, profesores, alumnos, y tutores legales de éstos. El objetivo es la creación de un conjunto cliente – servidor encargado de mantener de forma persistente los detalles tanto de los cursos impartidos por parte del centro que gestione el servicio, como de las evaluaciones de los alumnos y sus controles de presencia y calificaciones.

Nuestros responsables de centros tendrán la posibilidad de registrar los cursos que sus instituciones imparten. Esto implica poder tanto otorgarles identificadores internos, como los nombres oficiales y las descripciones de dichos cursos. También se considera de importancia los aforos máximos de los cursos y las fechas de inicio y finalización, por lo que ellos serán los responsables de informarlas adecuadamente mediante una interfaz en el cliente apropiada para ello. Además, ellos (o personal autorizado por ellos) podrán incluir las matrículas de aquellos alumnos que sean asignados a los cursos, de modo que en cualquier momento se pueda identificar qué alumnos pertenecen a qué cursos en un año dado, pudiendo así acceder a todos sus datos de forma inmediata y sin ambigüedades.

Por otra parte, los profesores tendrán dos posibilidades principales. La primera, es pasar controles de presencia a diario. No será una función obligatoria a desempeñar por el profesorado, pero de ser empleada, permitirá generar alertas en caso de faltas de asistencia tanto a padres (o tutores legales) registrados en la aplicación como al propio alumno que se ausente. Además, también se les brindará la posibilidad de incluir las correcciones a entregas o exámenes en la aplicación, junto con archivos adjuntos. Entre estos archivos podrían encontrarse imágenes (o documentos) con las correcciones de exámenes, documentos de apoyo a alumnos, etcétera.

Los alumnos tendrán acceso a su lista de exámenes y cursos en los que hayan sido registrados. Recibirán notificaciones cuando las correcciones de sus exámenes estén en línea (tras haber sido subidas al servicio por los profesores), y también cuando tengan alguna falta de asistencia. Esto permitirá al alumno ser consciente de cualquier notificación generada por el profesorado hacia él, y mantener el listado de sus calificaciones del año accesible. Además dispondrán de un perfil de usuario.

El último de los perfiles de usuario con el que nos encontramos es el de los tutores legales. Éstos recibirán las mismas alertas que los alumnos que estén registrados 'bajo su cargo', de modo que de forma inmediata puedan recibir tanto las alertas de calificaciones subidas, como aquellas de absentismos injustificados, lo cual les permitirá tomar medidas inmediatas ante cualquier tipo de problema disciplinario o educativo de sus hijos o estudiantes tutelados.

Finalmente, todos los usuarios podrán disponer de perfiles personales que incluirán (de querer el usuario) una foto de perfil, entre otros detalles. Inicialmente, esos usuarios podrán registrarse sin necesidad de aprobación, aunque en una segunda fase de desarrollo es interesante incluir la posibilidad de que se requiera autorización por parte del responsable del centro para el registro de usuarios.

# Propuesta técnica de implementación

La implementación del proyecto es, necesariamente, en dos partes. La primera es un servidor encargado de gestionar, de forma concurrente y segura, las comunicaciones y solicitudes creadas en relación a las operaciones permitidas para los cuatro tipos de usuarios principales del servicio: gestores de centros, profesores, alumnos, y tutores legales. La segunda será un cliente orientado inicialmente a la plataforma Android (API 19) que gestionará las operaciones de cara al usuario.

Nos encontramos, debido a los datos sensibles potencialmente visibles de los usuarios, ante una situación en la que la seguridad será algo crítico. Por este motivo se ha decidido la implementación de las comunicaciones mediante el protocolo HTTPS sobre un servicio REST-like, de modo que ataques que impliquen la lectura de paquetes en transporte no sean eficaces. El resto de características son propias de los dos elementos, como se detalla a continuación.

## Servidor

Para facilitar la creación del servidor y poder cumplir todos los requisitos de seguridad y autenticación necesarios para una aplicación de este tipo, se ha decidido emplear java junto al framework Spring, que nos facilitará la labor de generación del servidor y la infraestructura y API REST que necesitamos para gestionar las conexiones entre cliente y servidor.

La seguridad será implementada mediante un contenedor basado en Tomcat dentro de Spring, que se configurará para admitir sólo conexiones HTTPS mediante certificado firmado. Además, la autenticación de usuarios registrados se realizará mediante OAuth2. Esto último nos permite mantener un listado de usuarios junto con sus privilegios y estados de cuentas, junto al acceso mediante tokens de sesión, fácilmente gestionable y obtenible para poder garantizar que los datos sólo serán accesibles por aquellos usuarios que tengan los privilegios necesarios para ello. Spring, además, provee de anotaciones que nos permitirán realizar esto de forma sencilla, limitando el acceso a métodos concretos a determinados perfiles de usuario sin necesidad de alterar nuestra lógica de procesos.

El Servidor declarará una API que deberá ser respetada por cualquier cliente que desee hacer operaciones sobre él. La API será declarada en un POJO java a compartir con los clientes que se creen para interactuar con él, y definirá los diferentes REST endpoints accesibles por ellos.

## Cliente

El cliente que operará en nuestro servicio será, inicialmente, una aplicación para Android con la API 19 como API mínima durante el desarrollo. Para las comunicaciones con el servidor, se aprovechará la API pública de este junto con las librerías RetroFit que nos permitirán emplear la API como una interfaz estándar de Java, para que cualquier operación con el servidor se realice de forma estandarizada, y segura. HTTPS será por supuesto obligatorio.

El cliente va a disponer de una serie de pantallas que permitirán al usuario acceder a las diferentes funciones expuestas por el servicio. La pantalla principal será de acceso al servicio, y presentará al usuario con la opción tanto de acceder al servicio introduciendo sus credenciales, como de registrarse como nuevo usuario. En éste último caso, la pantalla variará y se solicitará un número mayor de datos para crear un perfil completo. Esta pantalla de registro es la única que no requiere de estar autenticado para su utilización, por motivos obvios. Una vez se haya conectado el usuario, lo primero que verá será su propio perfil.

El resto de funciones dependerán del perfil de acceso que tenga el usuario. El objetivo principal será crear, mediante un "drawer" lateral o un menú superior, la lista de operaciones permitidas para dicho usuario. De ese modo, éste podrá seleccionar la que desee en cada momento, lo cual le presentará con una pantalla diferente en la que realizar la operación deseada, o donde visualizar los datos solicitados.

Desde el punto de vista tecnológico, las diferentes pantallas de interfaz de usuario serán implementadas usando "Fragments" en Android. Éste es un modo más eficaz de realizar los cambios entre diferentes partes de nuestra aplicación, al ahorrar cambios de 'Activity' al pasar de una parte a otra de la aplicación, además de facilitar la creación de interfaces diferentes en función de la orientación o tamaño de la pantalla del dispositivo móvil en el que nos encontremos. Por decidir queda si mantener un 'backstack' que capture el botón de retroceso del dispositivo móvil, o hacer que dicho botón simplemente salga de la actividad.

Las comunicaciones con la red se realizarán mediante un servicio vinculado activo de forma continua, que síncronamente solicitará los datos usando RetroFit. Las comunicaciones entre la aplicación y el servicio serán realizadas mediante el mecanismo de IPC AIDL o llamadas directas, según tiempo disponible para la realización del proyecto.

# Bases de datos

En el proyecto, la estructura de datos principal se encontrará en el servidor. Éste gestionará una serie de tablas, creadas mediante Hibernate a través del framework Java Persistence API (JPA) de Java Enterprise, con la finalidad de simplificar y ayudar a la gestión de las tablas de objetos subyacentes, además de otorgarles una integridad referencial automatizada por este sistema, y permitir el acceso simple a los datos mediante interfaces Java. Este sistema ha sido elegido por proporcionar varias ventajas respecto a una implementación directa en base de datos:

- La creación de tablas se automatiza mediante anotaciones en las clases de datos. Esto evita errores en la creación de las mismas e inconsistencias.
- Al quedar vinculados los objetos con el cómo son almacenados, se evita el tener que realizar conversiones o manipulaciones directas sobre ellos o sus referencias externas.
- JPA automatiza la creación de tablas auxiliares que contengan uniones de otros elementos sin necesidad de declaraciones explícitas, lo que agiliza la ampliación del número de tablas en el caso de desear añadir funcionalidades, y previene errores de integridad referencial o duplicidad de información.
- Permite realizar consultas simples tanto usando una serie de palabras clave en los nombres de métodos como consultas complejas usando consultas explícitas en un subconjunto del lenguaje HQL.

La solicitud de datos al sistema encapsulador JPA se realiza mediante repositorios CRUD de Spring, lo cual facilitará la adquisición de datos y el uso de interfaces para crear una fachada que nos permita, de ser necesario, alterar el sistema de gestión de bases de datos empleado por nuestro servidor, sin necesidad de alterar la implementación lógica del programa en ningún otro aspecto que inyectándole la nueva clase gestora de persistencia que implemente dicha interfaz.

# Tablas empleadas

Dadas las utilidades de JPA y el almacenamiento de objetos con integridad referencial entre ellos, he considerado la implementación de las siguientes tablas, cuya nomenclatura está sujeta a cambio:

## Usuarios

| | | |
|---|---|---|
| **userName** | [ String ] | < PK > |
| **password** | [ String ] | |
| **firstName** | [ String ] | |
| **lastName** | [ String ] | |
| **profileDescrip** | [ String ] | < Texto de presentación en el perfil de usuario > |
| **NIF** | [ String] | |
| **Authorities** | [ String[] ] | < Descriptor de privilegios de usuario – Oauth2 > |
| **phoneNumbers** | [ String ] | < @ElementCollection, genera tabla auxiliar > |
| **address** | [ objeto ] | < Objeto tipo: Address > |

Esta tabla contendrá la lista de usuarios de todo tipo registrados en el sistema. El tipo de usuario (estudiante, profesor, tutor legal,…) será definido por la interpretación de las "authorities" heredadas de OAuth2, el sistema de autenticación elegido para la aplicación servidor. Esto permite evitar redundancia en datos (producida por ejemplo si incluimos algún identificador de tipo de usuario en el objeto, o herencia) y simplificar la implementación. Por otra parte, nos garantiza que en situaciones excepcionales como que un tutor sea alumno de algún curso, el servidor sea capaz de gestionarlo sin tener que realizar arreglos inconvenientes.
Por último, el caso del atributo phoneNumbers es especial. He decidido emplear la anotación @ElementCollection, que permite definir una tabla auxiliar que contenga estos valores, gestionada automáticamente por JPA.
Por último, indicar que el fetching de relaciones será realizado en modo "eager", que permitirá transferencia en red de objetos completos con sus objetos vinculados, sin perder referencias a claves ajenas.

## Cursos

| | | |
|---|---|---|
| **Id** | [ Int ] | < PK, autogenerada> |
| **courseTag** | [ String ] | < Nombre descriptivo, por ejemplo DISTDAM2015AD > |
| **courseFullNam** | [ String ] | < Nombre del curso en lenguaje natural > |
| **courseDescrip** | [ String ] | < Descripción completa del curso o presentación > |
| **maxAttendants** | [ Integer ] | < Número máximo de asistentes al curso > |
| **beginDate** | [ Date ] | < Fecha de comienzo del curso > |
| **endDate** | [ Date ] | < Fecha de finalización del curso > |
| **classroom** | [ Objeto] | < Aula en la que se impartirá. Tabla auxiliar autogenerada > |

La tabla cursos incluye todos los cursos, activos o no, que se hayan desarrollado y gestionado a través de nuestro servicio. Esto nos permite conseguir que haya un seguimiento completo de los mismos

aún después de su finalización, pudiendo acceder mediante las tablas vinculantes de objetos a listas de antiguos alumnos, listas de exámenes realizados, evaluaciones, aulas utilizadas, etcétera. Será esperable, en este caso, que un mismo curso realizado en diferentes sesiones o años tenga diferentes identificadores, que permitan separar claramente una sesión de otra. El ID es un entero autogenerado, pero se permitirá la creación de un descriptor legible (courseTag) que tendrá la restricción "unique" y brindará la posibilidad de que los cursos de diferentes años tengan un identificador secundario único entre ellos (p.ej. que incluya nombre corto y año lectivo) pero manteniendo el nombre formal del curso (courseFullName) visible.

## Tutores por curso

| | | |
|---|---|---|
| **id** | [ Integer ] | < PK, autogenerado > |
| **course** | [ Objeto ] | < FK, objeto de tabla Cursos > |
| **teacher** | [ Objeto ] | < FK, objeto de tabla Usuarios > |

Esta tabla de objetos incluirá aquellos que facilitan identificar qué tutores hay para cada curso. Dado que puede darse la circunstancia de que haya más de un tutor por curso, y que es probable que cada tutor imparta más de uno a lo largo de la vida de este gestor, se maneja esta tabla a parte. Es cierto que podría ser autogenerada por JPA, pero he decidido mantenerla separada para poder, por una parte separar más los objetos unos de otros (lo que implica que no habrá referencias circulares que generen objetos masivos a transferir), y por otra hacer demostración de ambas alternativas de creación de tablas de relación: manualmente (ésta) y autogeneradas.

## Matriculaciones en cursos

| | | |
|---|---|---|
| **id** | [ Integer ] | < PK, autogenerado > |
| **course** | [ Objeto ] | < FK, objeto de tabla Cursos > |
| **student** | [ Objeto ] | < FK, objeto de tabla Usuarios > |
| **finalGrade** | [ Float ] | |

Similar a como ocurre en la tabla anterior, en esta gestionamos las relaciones entre los alumnos y los cursos en los que participan, mediante una tabla creada manualmente. Además incluyo el atributo notaFinal, que podrá ser rellenado para dejar constancia de la nota al final del curso realizado de ese alumno. Éste último atributo está sujeto a modificación, y ser incluido en una tabla a parte que relacione este registro de alumnos matriculados con sus notas por curso.

## Exámenes

| | | |
|---|---|---|
| **id** | [ Integer ] | < PK, autogenerado > |
| **evaluator** | [ Objeto ] | < FK, objeto de la tabla Usuarios > |
| **matricula** | [ Objeto ] | < FK, objeto de la tabla Matriculaciones en cursos > |
| **examName** | [ String ] | < Nombre o título del examen > |
| **examComment** | [ String ] | < Comentarios sobre la corrección por el profesor > |
| **examDate** | [ Date ] | < Fecha de realización > |
| **examMark** | [ Float ] | < Nota asignada al examen > |
| **data** | [ String ] | < URL a archivo > |

Esta tabla incluirá la información sobre cualquier examen (o tarea) realizada por un alumno. Permite desde indicar los datos básicos del examen como quién lo realiza (a través de la matrícula asociada del curso y el alumno) hasta añadir datos mediante un archivo que es vinculado en este objeto. Esto último podría ser empleado, por ejemplo, para incluir una imagen al examen corregido, o archivos de ayuda o apoyo al alumno.
He considerado adecuado incluir también el nombre del corrector, que a pesar de estar implícito que será el tutor del curso, puede en circunstancias excepcionales ser un tercero. Esto permite realizar seguimientos de las correcciones y asociarlas a quien las realizó, útil para reclamaciones o solución de problemas.

## Control de presencia

| | | |
|---|---|---|
| **Id** | [ Integer ] | < PK, autogenerado > |
| **alumnMat** | [ Objeto ] | < FK, tabla Matriculaciones en cursos > |
| **date** | [ Date ] | < Fecha en la que se realiza este control de presencia > |
| **isPresent** | [ Boolean ] | < True si el alumno está presente en esta clase, este día > |

Esta tabla permite hacer el seguimiento de presencia de los alumnos independientemente del curso en el que estén registrados. Dado que un alumno podría estar registrado en varias clases el mismo día, y acudir a unas sí y a otras no, usamos la matrícula del alumno como uno de los datos del control de presencia.
Un caso práctico sería que el tutor quisiera realizar un control de presencia al comenzar la clase. Se solicitaría la creación de un objeto control de presencia para todos los alumnos registrados en la tabla de Matriculaciones de Cursos para el curso en cuestión con la fecha actual pre-creada. Una vez rellenado en la aplicación, sería devuelto al servidor que lo almacenará tras verificar la integridad de los datos y su coherencia. En el futuro, se podría consultar la presencia de un alumno a lo largo de toda su vida de estudiante en todas las clases, por clase, por fecha, etcétera.
Además, estos controles de presencia generarán alertas para usuarios definidos como seguidores de los alumnos (tutores legales, padres).

## Registro de notificaciones:

| | | |
|---|---|---|
| **id** | [ Integer ] | < PK, autogenerado > |
| **notifregistry** | [ Objeto ] | < FK, entidad de notificación que almacena la relación > |
| **notifCreator** | [ Objeto ] | < FK, tabla usuarios, profesor que genera la notificación > |
| **notifType** | [ String ] | < Descriptor del tipo de notificación (examen, falta) > |
| **detailMessage** | [ String ] | < Descripción creada por el profesor para este evento > |
| **isRead** | [ Boolean ] | < Define si la notificación ya ha sido leída por el receptor > |

Esta tabla almacena objetos de registro para las notificaciones que serán enviadas a los usuarios. Incluye una referencia a una entrada de nuestro registro de usuarios tutelados, que de por si incorpora tanto al estudiante "tutelado" como a aquel que tenga acceso a sus notificaciones, un objeto 'detalles' que incluirá datos básicos sobre el motivo de la notificación, y un descriptor "estaLeida" que permitirá que el usuario notificado sólo reciba prioritariamente aquellas notificaciones respecto a las que no haya tomado acción alguna aún.

Estas notificaciones podrán ser utilizadas desde para publicar notas de exámenes (se generaría notificación al usuario y a sus "seguidores" registrados como tales) como para informar de ausencias injustificadas a padres o tutores.

## Registro de usuarios tutelados:

| | | |
|---|---|---|
| **id** | [ Integer ] | < PK, autogenerado > |
| **student** | [ Objeto ] | < FK, estudiante por el que se genera la notificación > |
| **follower** | [ Objeto ] | < FK, usuario registrado como 'tutor legal > |
| **beginDate** | [ Date ] | < Fecha de inicio del seguimiento > |
| **endDate** | [ Date ] | < Fecha de finalización del seguimiento > |

Este será el registro de tutores legales o padres que recibirán las notificaciones de absentismos y calificaciones de cada alumno. Nos permitirá de forma eficaz saber quién debe ser informado de éstas, de modo que se generen tantas notificaciones en un momento dado como padres o tutores tenga registrado el alumno.
Además, se añaden los campos "beginDate" y "endDate" que informan del periodo en el que se está realizando el seguimiento de ese alumno por parte de ese tutor. Las notificaciones sólo serán generadas durante ese periodo para esta relación. En el caso de que endDate sea nulo, asumiremos que no hay fecha fin establecida y se podrán generar notificaciones para esta relación.

# Esquema de tablas

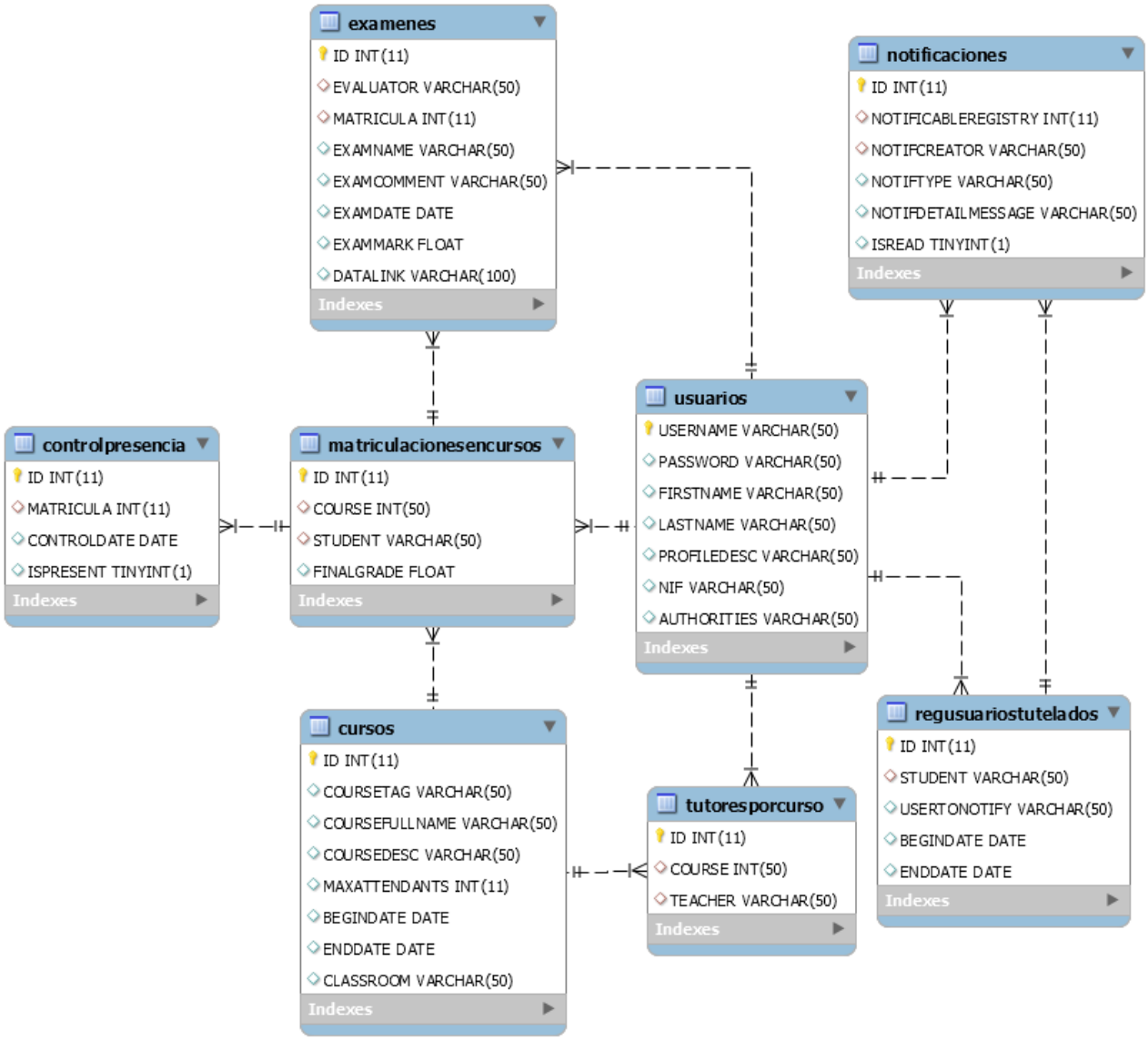


*Ilustración 1 Diagrama de tablas*