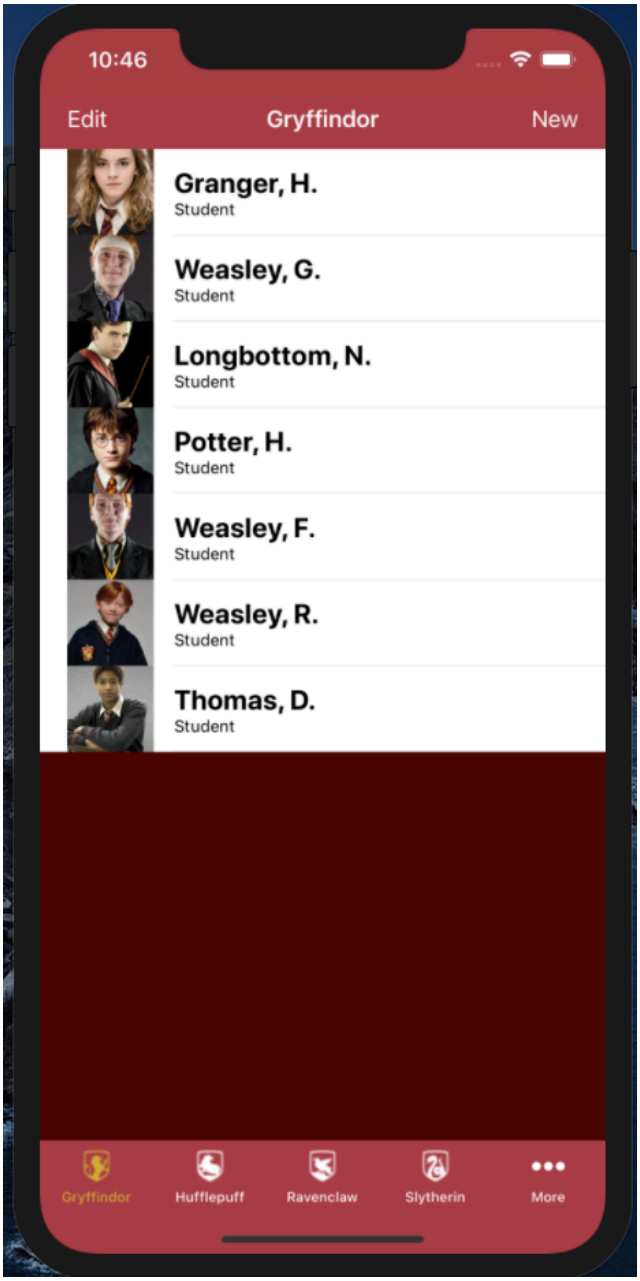
DumbleBook

Desarrollo de Aplicaciones Avanzadas: Programación en Swift



Autor: Alejandro Mateos Pedraza, alumno de Grado de Ingeniería Informática. USAL 2020.
Correo electrónico: alema@usal.es
Tutor: José Rafael García-Bermejo Giner (coti@usal.es)
Fecha de publicación: Viernes, 10 de enero de 2020
Versión: es.alema.master.DumbleBook.v.1.0

Descripción del proyecto: la aplicación desarrollada pretende simular un sistema de gestión de estudiantes, clasificados por escuelas. Ambientada en el mundo de Hogwarts, de la famosa saga de novelas 'Harry Potter', la aplicación permite desde la inscripción y eliminación de estudiantes, hasta la edición de la información relativa a cada uno de ellos. En el proyecto presentado se incorporan tres ramas: una inicial básica pero funcional que cumple los requisitos solicitados para ser admitida, y dos más que extienden el contenido del curso impartido e incorporan características adicionales, como a continuación se verá.

DESCRIPCIÓN DEL PROYECTO

Introducción. Descripción de las ramas implementadas

Como observará al analizar el proyecto, se dispone de un total de tres ramas, que se resumen como sigue:

A) **master**: Es la primera rama creada en el proyecto. Contiene las características esenciales para que el proyecto pueda ser sometido a evaluación.

B) **OptionalFeatures_DumbleChat**: la aplicación se rediseña por completo y se realiza la implementación de un chat para los alumnos existentes en la aplicación. Como se verá, la gestión del intercambio de mensajes se realiza a través de una base de datos en la nube alojada en un servidor Firebase.

C) **UsersFirebase**: se aprovecha el conocimiento adquirido sobre FireBase para aplicar la gestión de usuarios también al servidor online. De esta manera, los cambios que realice un usuario podrán ser vistos por cualquier otro con acceso a la aplicación.

La descripción del proyecto que se expone a continuación se realizará desde el punto de vista de la última rama implementada, por ser la que carga con la totalidad de la complejidad del proyecto e incorpora las características finales del producto.

StoryBoards. Descripción de las vistas

Hasta diez ficheros de storyboard son los que componen el proyecto para la administración de las diferentes vistas de la aplicación:

A) **LaunchScreen**: ventana que aparece en el momento del arranque.
B) **Main**: contiene un TabBarController que será el encargado de guiar al usuario por las diferentes rutas existentes en la aplicación. En el menú inferior, tendrá acceso a las cuatro casas. Por su parte, en la sección 'More' podrá visitar el área de visualización de los escudos y el chat, según lo desee.
C) **Gryffindor / Hufflepuff / Ravenclaw / Slytherin:** representa la parte máster de una vista de tipo maestro-detalle. Su contenido se basa en una tabla de alumnos. El usuario podrá interactuar con cada fila, que muestra un resumen relevante de cada alumno matriculado, para acceder a la vista de detalle del alumno en cuestión. En el menú superior, aparecen los botones de eliminación (Edit) y adición (New) de estudiantes. La pulsación de los mismos llevará al cliente a las correspondientes ventanas. La disposición es análoga para las cuatro casas de la escuela.
D) **Shields**: área de visualización de escudos. El recorrido por las diferentes escuelas se realiza a través de un control segmentado. Se puede comprobar también desde la misma sección el número de alumnos matriculados de cada escuela.
E) **StudentDetail**: vista de detalle de un alumno. Está formada por dos cuadros de texto que contienen el nombre y apellidos del alumno seleccionado, y que asimismo permiten su edición. Mismo caso con la foto inferior, que el cliente podrá cambiar accediendo a la cámara del dispositivo, si existe, o en todo caso

accediendo a la librería. La aplicación solicitará los permisos de privacidad adecuados para lo propio. Si el usuario desea guardar los cambios puede hacer uso del botón superior derecho de la ventana, que regresa a la sección maestra con guardado de los cambios efectuados.

F) **AddStudent**: panel de adición de nuevos estudiantes. Se indicará el nombre y apellidos del nuevo alumno, y se regresará a la vista maestra por dos vías: botón 'Cancel' (el alumno no se añade), o bien el botón 'Save', que implica la creación de un nuevo registro para el alumno. En función de la casa donde se esté efectuando la adición, el alumno se añadirá a ésta. La imagen con la que el usuario aparece tras la adición es una imagen predeterminada, que el cliente podrá cambiar accediendo al detalle del alumno.

G) **DumbleChat**: vista de chat, que inicialmente mostrará la ventana de login. Toda la gestión de acceso y el paso a la ventana de visionado de mensajes -tras validarse el usuario en cuestión- se gestiona por código a través de SwiftUI.

Controladores. Aspectos esenciales

A) **HogwartsViewController**: es un controlador único asociado a la vista maestra de cada una de las casas [véase Gryffindor / Hufflepuff / Ravenclaw / Slytherin en el capítulo anterior]. A continuación se exponen los detalles más relevantes de su implementación:

A.a) Uso de una **extensión** que nos permite ampliar las características del tipo UIImage. Resulta útil para enmarcar las imágenes en un cuadrado perfecto, y que no se visualicen de forma heterogénea.

A.b) En el **viewDidLoad** se ejecutan tres procedimientos esenciales: 1) registro de un observador de notificaciones, que captura el instante en el que se ha producido la lectura de datos, procedentes del servidor Firebase. 2) Obtención del nombre de la casa en la que estamos, dado que se utiliza un controlador único para gestionar las cuatro casas de forma indistinta. 3) Llamada al procedimiento de configuración de vistas, que proporciona un diseño apropiado y uniforme a la mismas.

A.c) Sobreescritura del método **prepare** del segue de acceso a la vista de detalle de alumnos: se utiliza para capturar la información del alumno seleccionado y establecer los valores de nombre, apellidos e imagen que el alumno tenga asignados en ese momento.

A.d) Métodos **tableView** para la gestión de la tabla maestra: se recogen los datos de alumnos registrados en la lógica de aplicación para configurar la tabla, celda a celda.

A.e) Métodos **unwind** de regreso a la vista maestra: proporcionan la vuelta atrás desde los paneles de detalle y adición, con guardado o cancelación de cambios, según corresponda.

B) **DetailViewController:** controlador asociado a la vista de detalle de alumnos. Se basa en el uso de delegados para la gestión de las modificaciones oportunas.

B.a) Uso de **delegados** para gestión de cambios de nombre y apellido: se capturan todas las pulsaciones de teclado del usuario sobre los textField, de tal manera que en el momento en que el texto cambia, actualizamos una serie de variables de clase que se usan desde el controlador de las casas para actualizar la información de alumno, una vez se regresa con guardado de cambios.

B.b) Uso de **delegados** para gestionar el cambio de imagen: se incorpora un gestureRecognizer que captura la pulsación sobre la imagen que aparece en la ventana. Cuando se detecta pulsación se efectúa una llamada a los métodos que, mediante un imagePicker permiten acceder a la cámara del dispositivo, o en caso de ausencia, a la librería del mismo.

C) **AddStudentViewController**: panel de adición de nuevos estudiantes. De forma análoga al controlador anterior, se capturan mediante delegación las pulsaciones del usuario sobre los textField para reconocer el texto introducido.

D) **ShieldsViewController**: el método central de este controlador reside en houseSelection(…). Aquí se gestionan los cambios de selección en el control segmentado para visualizar los datos de cada una de las respectivas casas (escudo y número de alumnos).

E) **DumbleChatView**: representa el añadido más importante a la hora de extender los requerimientos del enunciado. Implementa novedades en dos planos. En el área del código, se ha optado por usar la sintaxis de SwiftUI (el nuevo modelo de desarrollo de interfaces concebido por apple). Y en segundo lugar, en la forma de acceder a la información, dado que los mensajes se leen y escriben a través de un servidor on-line alojado en la plataforma Firebase (de Google). Estos son algunos de sus aspectos esenciales:

E.a) **DumbleChatView**: presenta la ventana de login. El diseño de interfaz, como es básico en SwiftUI, se gestiona mediante pilas de objetos, ya sea en el plano vertical, horizontal y frontal (superposición de objetos). Entre las funciones implementadas, cabe destacar: getImage (que devuelve la imagen asociada al usuario para mostrarla sobre la sección de login -en caso contrario mostrará la predefinida) y checkLogin (que realiza la comprobación de usuarios y gestión de acceso a la sala de chat. En lo que respecta a este acceso, una vez que se ha validado que el usuario es correcto, se crea una NavigationLink que a través e un botón nos conduce a la página de mensajes.

E.b) **MsgPage**: vista (View), enlazada a partir del botón de Join de la sección de login que muestra una lista (List de SwiftUI) de mensajes enviados, que debe leerse de Firebase. El proceso es como sigue: se crea un objeto observador [véase la clase observer] que recoge los mensajes que se han intercambiado y se crea una Lista que se rellena con el campo messages de ese observador. Este campo messages está definido como un tipo de dato que contiene un id, usuario emisor y contenido del mensaje. De esta manera, cada fila de la List se rellena con cada uno de los mensajes (messages) del observador (msg). Se establecen distintas propiedades para cada fila en función de si el mensaje corresponde al propio usuario o se ha enviado por otro. Esto último se hace dentro de la estructura MsgRow. En resumen, se tiene:

E.b.a) **MsgPage**: zona de mensajería. Se basa en una pila vertical que contiene, en este orden, la List de mensajes intercambiados y el área de escritura (mediante textField y botón Send que añade el mensaje escrito al servidor Firebase).

E.b.b) **observer**: clase observadora que recopila de Firebase los mensajes transmitidos e implementa una función de adición de nuevos mensajes (asociada al botón Send que se ha comentado).

E.b.c) **datatype**: estructura que construye un mensaje por su id, usuario emisor y contenido.

E.b.d) **MsgRow**: Vista de configuración de cada fila de la List de mensajes enviados. Permite: establecer un rectángulo de fondo coloreado para identificar los mensajes entre los propios y ajenos, visualizar la fotografía del emisor, colocar a un lado los mensajes propios y al opuesto los ajenos…

E.b.e) **getImage**: procedimiento de obtención de las imágenes de los usuarios que acceden al chat.

Para más información puede acceder al mismo código del proyecto que se adjunta.

Modelo. Casas y alumnos

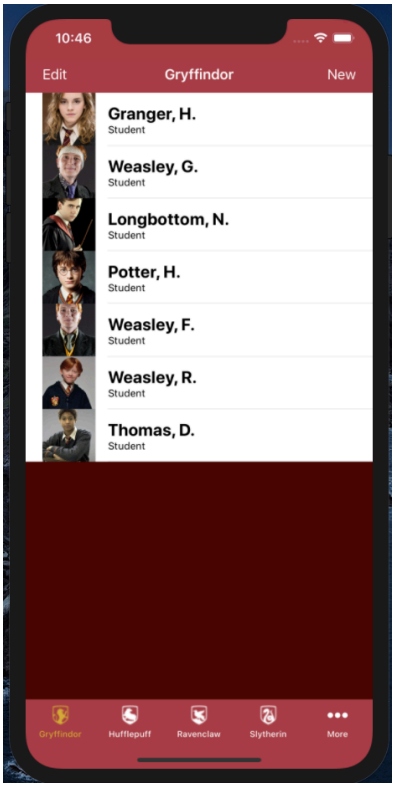
Se dispone de un modelo de alumnos, que se utiliza para la creación de objetos de tipo alumno dado un nombre, apellido y alojamiento, y de un modelo de casas, que gestiona las relaciones entre estos alumnos y con el alojamiento en el que residen. Se detallan brevemente:

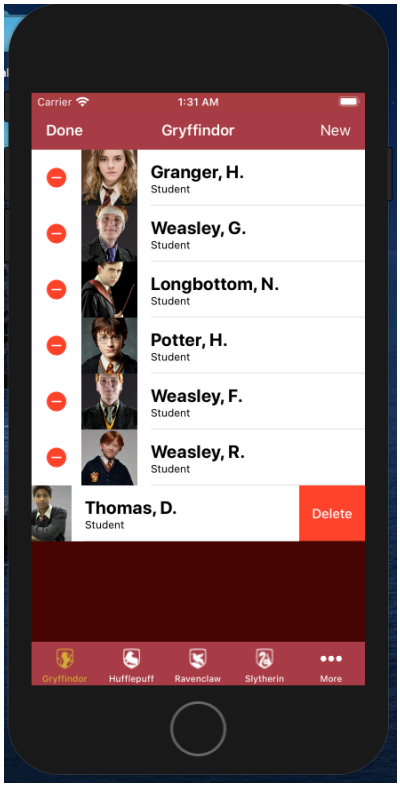
A) **Student**. Estos son los métodos esenciales:
A.a) **init(name, surname, houseName):** a partir de estos parámetros iniciales se crea un objeto alumno, al que se agrega información adicional calculada automáticamente a partir de los argumentos proporcionados (userName y passWord para es chat, identificador para la imagen, nombre abreviado, etc).

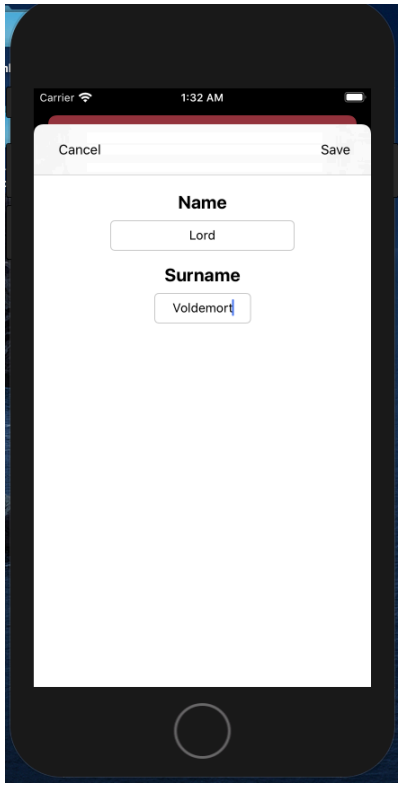
A.b) Se implementa la interfaz **Codable** para gestionar la capa de persistencia. El almacenamiento físico de alumnos de nueva creación se gestiona mediante fichero json.

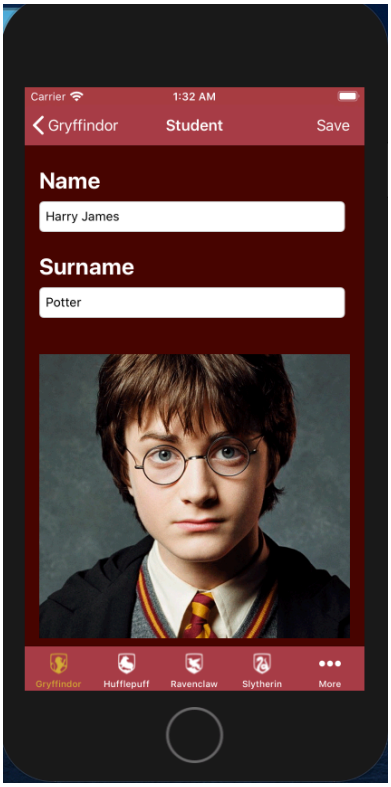
B) **House**. A continuación se describen los métodos principales de la clase:
B.a) **init( ):** lectura de alumnos procedente de la base de datos de Firebase.
B.b) **studentsCount(houseName):** devuelve el número de estudiantes alojados en la casa que se proporciona
B.c) **addStudent(student, houseName, index):** añade un estudiante a la casa proporcionada en el índice que se especifica.
B.d) **editStudent(student, houseName, newName, newSurname):** edición de estudiantes, a los que se proporciona nuevo nombre y apellido. Para que la edición tenga lugar, el student proporcionado debe existir en la base de datos actual.
B.e) **updateFileOfStudents(filename, data):** reescribe el vector que se pasa en el parámetro data en el fichero json identificado por filename.
B.f) **Métodos de acceso a la base de datos remota:** readStudentsFromFirebase() -lectura-, readInitialStudentsFromFirebaseCollectionToVector() -llenado del vector de estudiantes predefinidos-, addStudentToFirebase() -escritura-, removeStudentFromFirebase() -eliminación. Los procedimientos en cada caso son semejantes en función de la tarea solicitada.

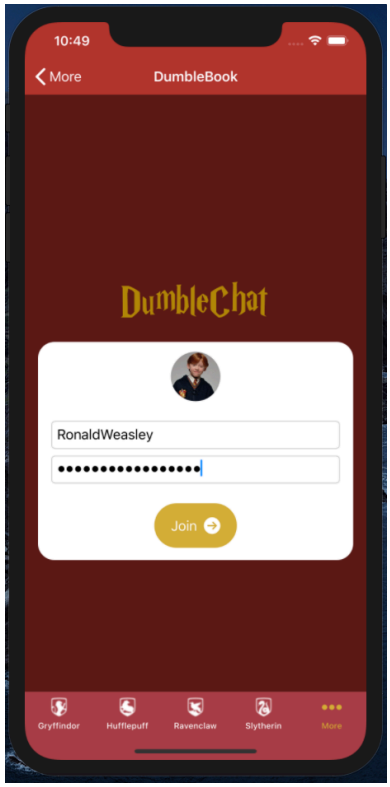
Anexo. Imágenes de la aplicación

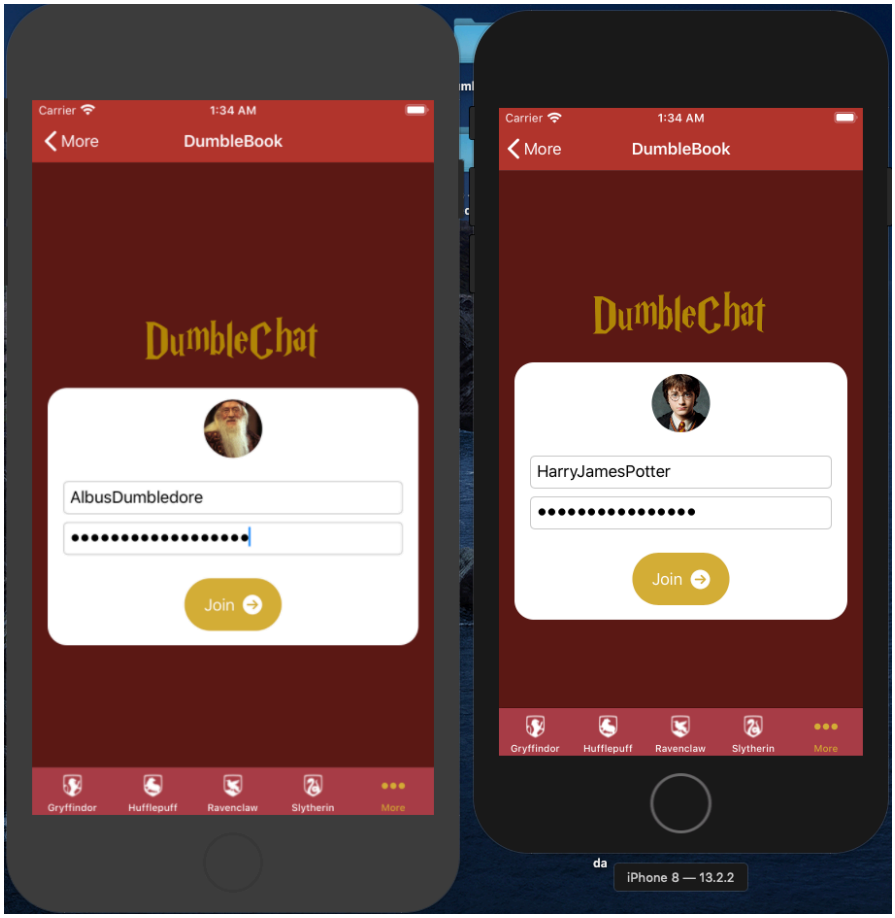


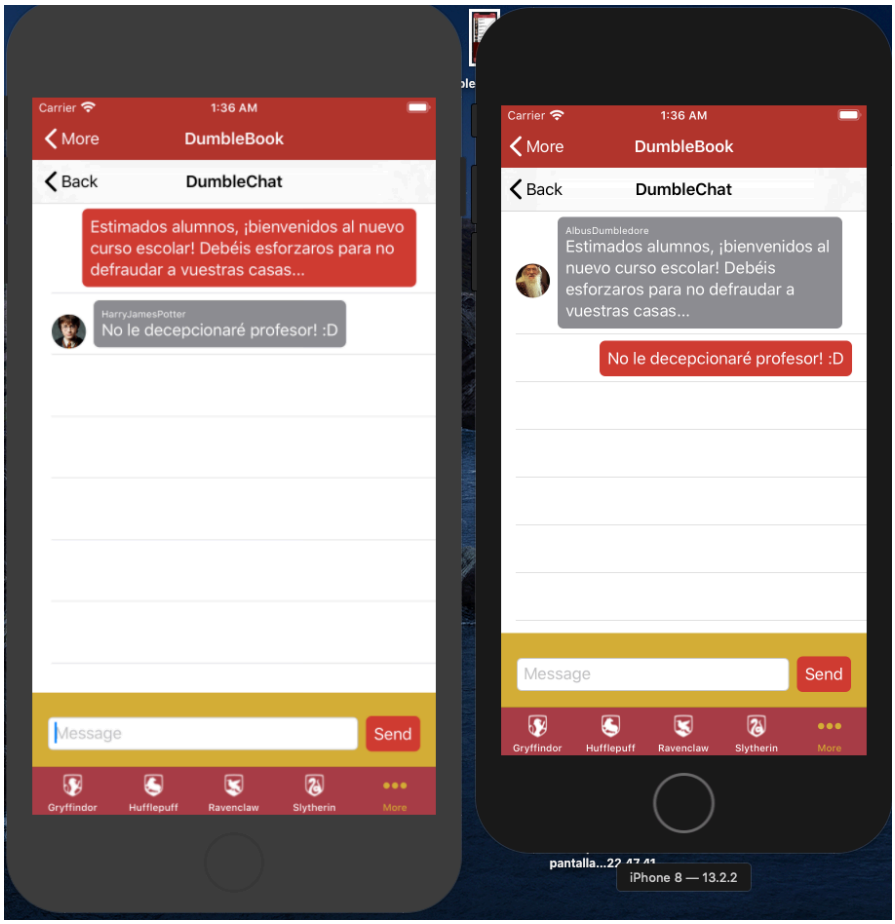









Carrier
1:34 AM
More
DumbleBook
DumbleChat
AlbusDumbledore
Join
Gryffindor
Hufflepuff
Ravenclaw
Slytherin
More
HarryJamesPotter
iPhone 8 — 13.2.2


Carrier
1:36 AM
More
DumbleBook
Back
DumbleChat
Estimados alumnos, ¡bienvenidos al nuevo curso escolar! Debéis esforzaros para no defraudar a vuestras casas...
HarryJamesPotter
No le decepcionaré profesor! :D
Message
Send
Gryffindor
Hufflepuff
Ravenclaw
Slytherin
More
AlbusDumbledore
iPhone 8 — 13.2.2