Documento de Diseño e Implementación.

Sistema de Reserva de aulas - FCEFYN.

26/06/2017

Grupo cmd.

Benítez Jeremías, Rao Maximiliano, Monsierra Lucas.

Índice.

6. UNIT TESTS

7. REQUIREMENTS MATRIX  4

Historial de revisiones.

|  |  |  |  |
| --- | --- | --- | --- |
| Versión | Fecha | Observaciones | Autor |
| 1.0.0 | 23/06/17 | Creación del documento.  Se agregó la Introducción. | Monsierra, Lucas. |
| 1.0.1 | 24/06/17 | Se agregó la sección Diseño de datos. | Monsierra, Lucas. |

Introducción.

El Documento de Diseño de Software (SDD), es un documento que proporciona la documentación que se utilizará para ayudar en el desarrollo del software, proporcionando los detalles de cómo se debe construir.

Dentro del Documento de Diseño de Software se encuentran la documentación narrativa y gráfica del diseño de software para el proyecto, incluyendo diagramas de clase, de secuencia, paquetes, etc. y otra información de requisitos de soporte. Además se incluirán en éste las pruebas unitarias y la descripción de los patrones de diseño Observer, Singleton y Strategy utilizados para el desarrollo del proyecto.

## Propósito.

El propósito del SDD es proporcionar una descripción del diseño de un sistema lo suficientemente completo como para permitir el desarrollo del software, así tener una comprensión de lo que se va a construir y cómo se espera que se construya.

## Alcance.

Este SDD se centra en descripción de bajo nivel del sistema y sus partes críticas, haciendo hincapié en la generación de los documentos, diagramas, pruebas unitarias y la codificación de los documentos.

## Definiciones y Abreviaturas.

* BDD: Base de datos.
* DER: Diagrama de Entidad-Relación.
* SDD: Documento de Diseño de Software (Document Desing Software).

## Referencias.

Template SDD: <https://sovannarith.files.wordpress.com/2012/07/sdd_template.pdf>

Diseño de datos.

## Estructura de datos interna del software.

Los datos residirán localmente en la memoria y se organizarán en función de las clases definidas más adelante en este documento. Dado que nuestro sistema puede considerarse centrado en los datos, las clases que manejan los datos se aislarán y se accederá a través de un sistema Model-ViewController, descripto en el Documento de Arquitectura. Todos datos de necesarios se solicitarán a la BDD en la inicialización de la aplicación y se actualizarán según sea necesario en función de las acciones del usuario.

El almacenamiento permanente de la información del usuario se realizará utilizando una base de datos local, primeramente diseñada en MySQL, y luego transformada a SQLite para facilitar la interacción de ésta con la aplicación. La Sección 3.8 (Componente del servidor) cubre este tema con mayor detalle. Las consultas de ésta se harán utilizando el mismo lenguaje SQLite.

## Estructura global de datos.

La estructura de datos global de esta aplicación se caracteriza mejor por la base de datos. La estructura de la base de datos muestra los datos implicados en la aplicación en su sentido más puro. Los usuarios del sistema nunca accederán a ésta directamente, sino que emitirán peticiones.

## Estructura de datos temporales.

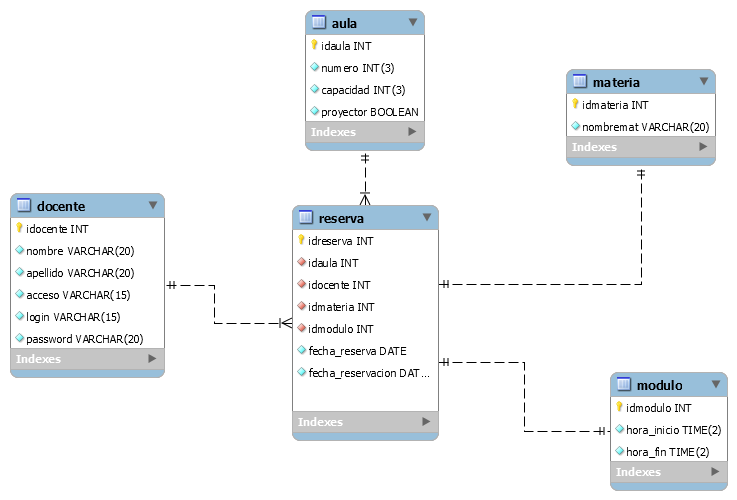
Las estructuras temporales de datos, se refieren a los objetos de datos que se crean en el sistema, los cuales sólo existirán durante el tiempo que la aplicación se ejecuta y posteriormente se destruirán.

## Diagrama de Entidad-Relación.

En este diagrama se trabaja con un nivel de abstracción más bajo, ya que se muestran las tablas en donde se guardará la información de la aplicación.

Para el caso de los profesores por ejemplo, se guardarán en tabla un ID, el nombre, apellido, acceso, login y contraseña. Para las reservas, se querrá guardar un ID, el aula reservada, el docente que la reservó, la materia, fecha y módulo en el que se reservó. El aula a su vez tendrá un ID, un número, información sobre su capacidad y si posee proyector o no. Para las materias se almacenará un ID y su nombre, y por último, para los módulos se almacenarán con un ID, una fecha de inicio y una de fin.

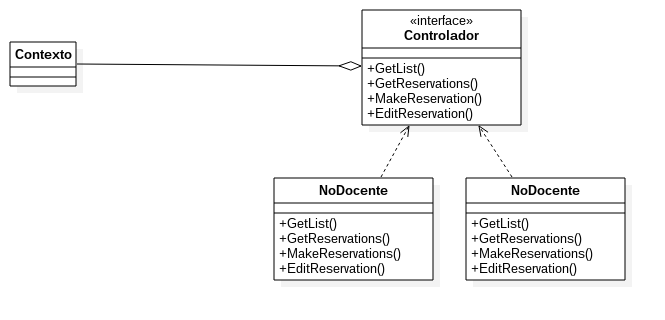
El sistema se debe diseñar para que solamente los usuarios Docentes puedan modificar el contenido de la tabla Reservas (de su propia reserva). Las otras tablas sólo podrán ser modificadas por los administradores de la aplicación. De esta manera se proporciona seguridad en los datos.



Patrones de diseño

La simplificación del desarrollo, y la posterior expansión, de este sistema de software, es el motivo que impulsó a adoptar tres patrones de diseño ampliamente utilizados en otros proyectos. A saber: Strategy, Singleton y Observer.

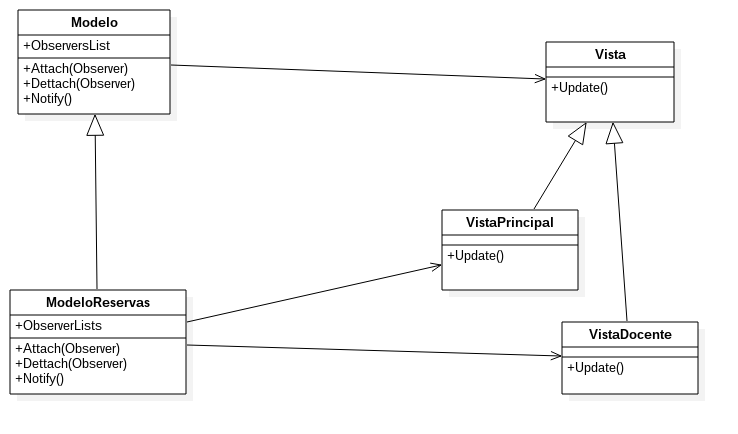
## Strategy.

En este caso, la utilización de este patrón nos permite la implementación de los dos tipos de usuario posibles, Docente y NoDocente, y el intercambio de los mismos en tiempo de ejecución, lo cual permitirá que los usuarios se registren como docentes en caso de tener una cuenta en el sistema, y accedan a las funcionalidades pertinentes.

Cabe destacar que, si bien la clase NoDocente implementa todos los métodos de Controlador, no toda la funcionalidad será accesible desde la misma. Así, en caso de que un no docente intente acceder a estos métodos restringidos, un mensaje de error será mostrado.

## Observer.

Este patrón define un tipo especial de relación uno-a-muchos entre clases. La principal función de dicha relación, es la capacidad de poder notificar a los objetos de una clase, cuando el estado de otra cambió.

En este proyecto, el patrón Observer se utiliza para mantener dos vistas actualizadas según los datos que ocurran en la base de datos. Si un Docente realiza una nueva reserva, tanto la vista principal (encargada de mostrar el listado de aulas con sus reservas si las hubiere), como la vista de docentes (en la cual, una vez registrados, los mismos pueden obtener detalles sobre sus reservas); son notificadas instantáneamente, y las mismas pueden modificar los valores mostrados. 

Esto nos permite ofrecer en todo momento la información más actual de la base de datos, evitando así conflictos que pudieran surgir por la visualización de datos que ya no sean válidos.

## Singleton.

El patrón Singleton fue utilizado en la representación de la base de datos dentro de la aplicación. Esto da la posibilidad de acceder a la base de datos mediante una única instancia de la clase que la warpea, evitando así un uso ineficiente de la memoria el cual, al crecer la base de datos, podría afectar fuertemente el funcionamiento del sistema.

Diseño de Interfaz de Usuario.

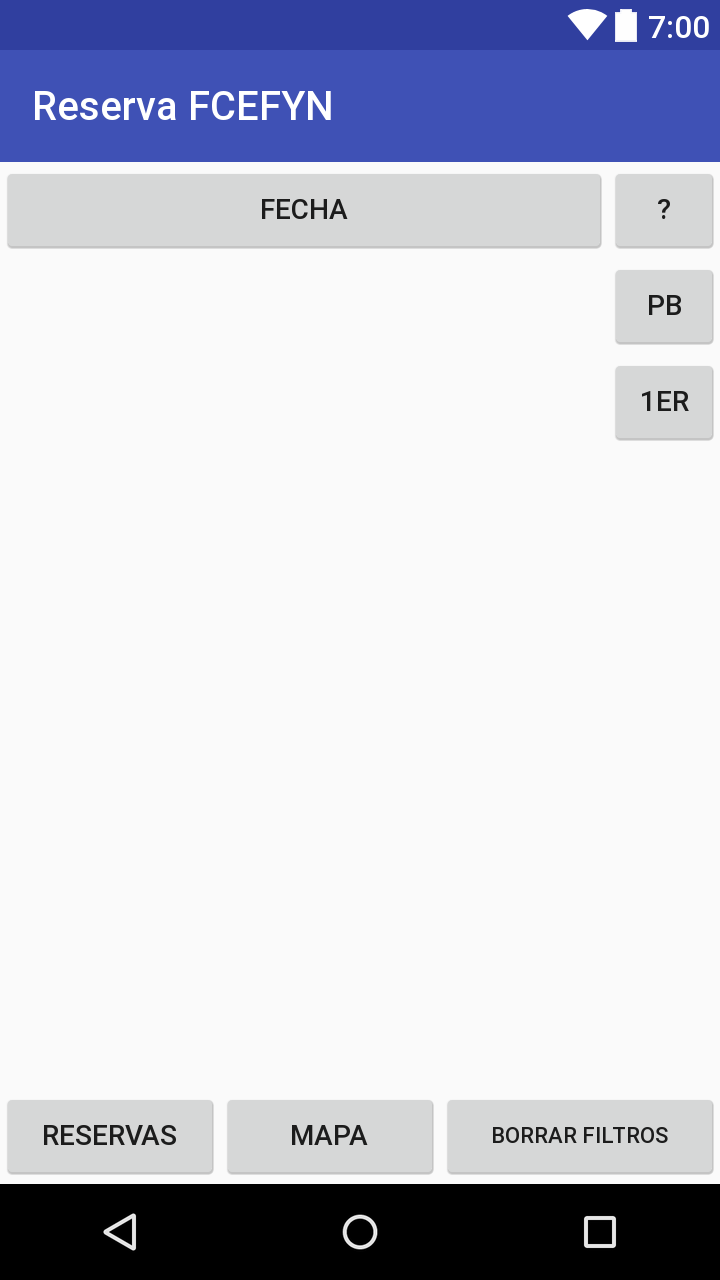
La interfaz de usuario consiste en un conjunto de menús a través de los cuales el usuario puede interactuar con datos de la BDD. Estos menús incluyen una variedad botones, los cuales permitirán la elección de filtros específicos a la tabla que se mostrará en el medio de la pantalla. El usuario interactuará con los menús a través de la pantalla táctil del dispositivo.

## Descripción de la Interfaz de Usuario.

Cada menú consistirá en varios componentes GUI, como botones y objetos de lista. Estos componentes se arreglarán de tal manera que el usuario será capaz de comprender rápidamente el propósito de cada uno y realizar cualquier tarea de manera eficiente. Una descripción detallada de estos menús y sus interacciones entre sí se agrega más adelante.

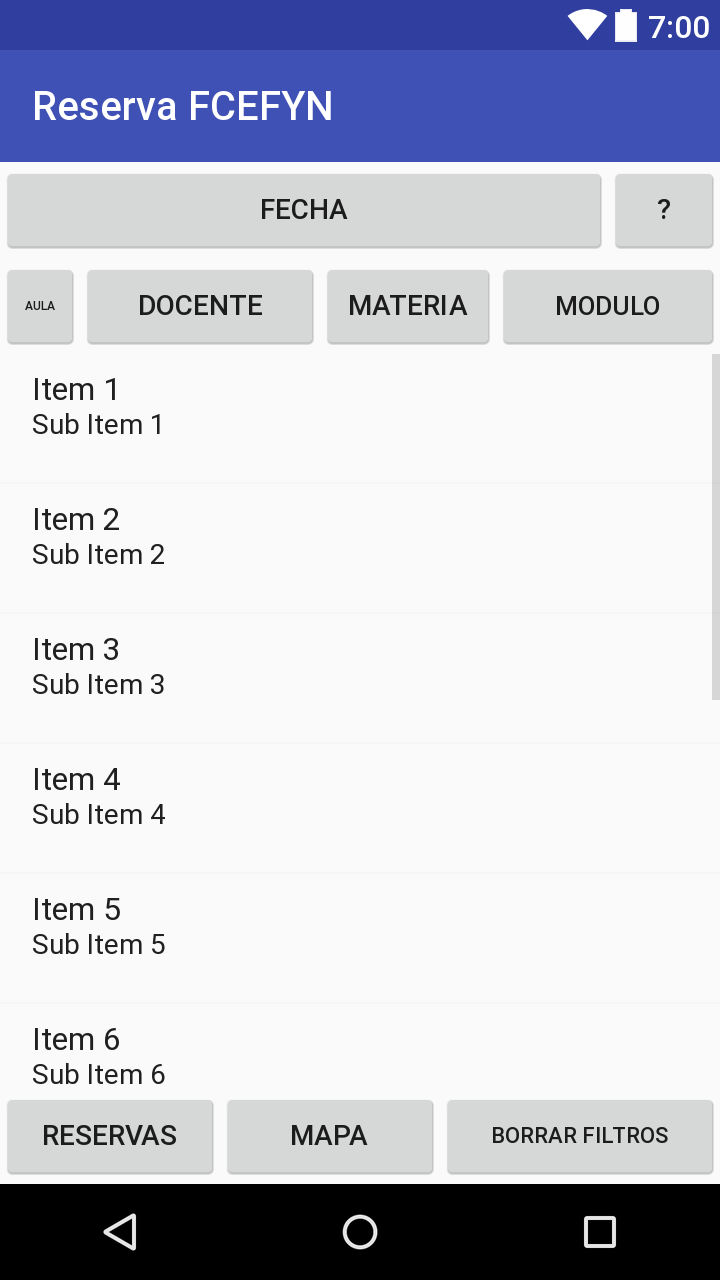
Nuestro sistema se basa en este lienzo común para todos, que dependiendo de la acción del usuario, mostrará distintas tablas, imágenes o texto de ayuda.

Por ejemplo, la vista de Ayuda mostrará en la parte central un texto de ayuda donde se detalla el manual de usuario.

Para las demás vistas simplemente se agregan o quitan botones.

Ahora se mostrará una vista de la sección Mapa.

Como se puede apreciar en este lienzo, se agregaron 2 botones, “PB” y “1ER”, los cuales les permiten a los usuarios intercambiar la imagen del plano de la facultad que se muestra en el medio de la pantalla.

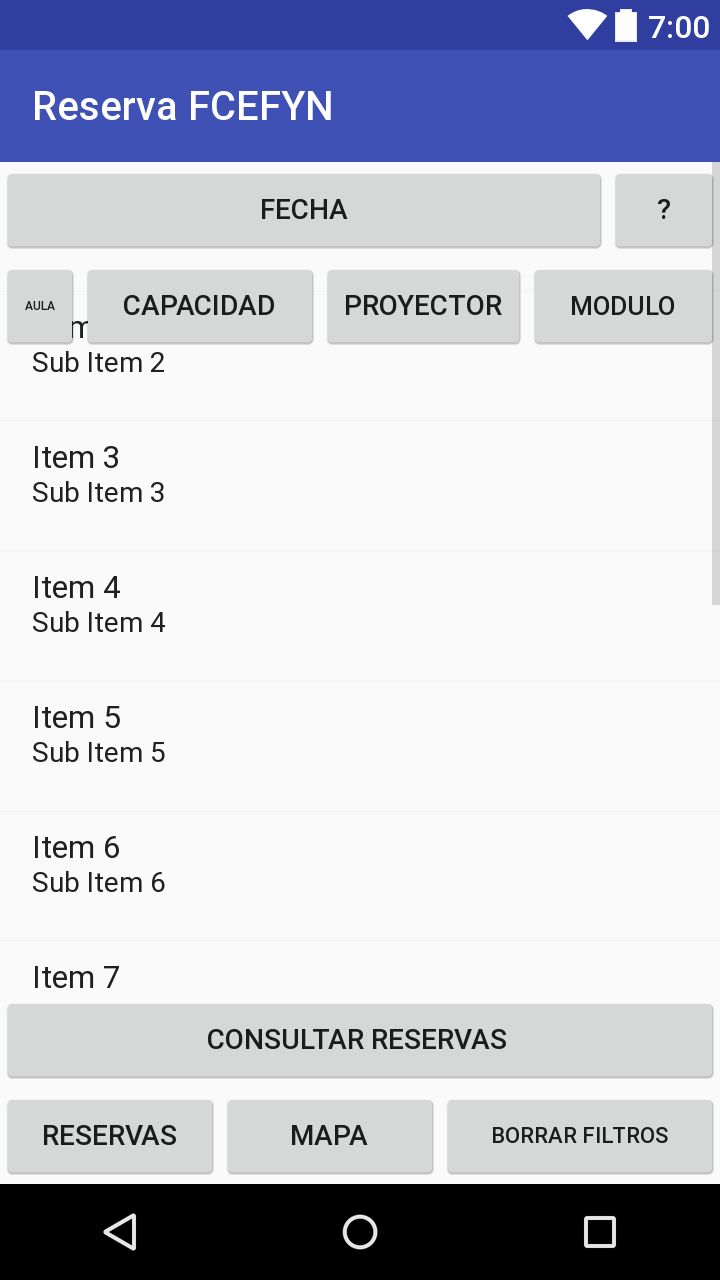
La vista inicial cuando un usuario abre la aplicación tendrá este formato, donde en la parte central de la pantalla se mostrará un listado de reservas para un determinado día. El botón “Fecha” permitirá cambiarla y así filtrar en la base de datos las reservas de otro día.

El botón “Aula” permite elegir un aula determinada para mostrar las reservas para un aula.

El botón “Docente”, al igual que el anterior, permite elegir un docente determinado para filtrar y mostrar sus reservas.

Los botones “Materia” y “Módulo” tienen un comportamiento similar a los 2 anteriores.

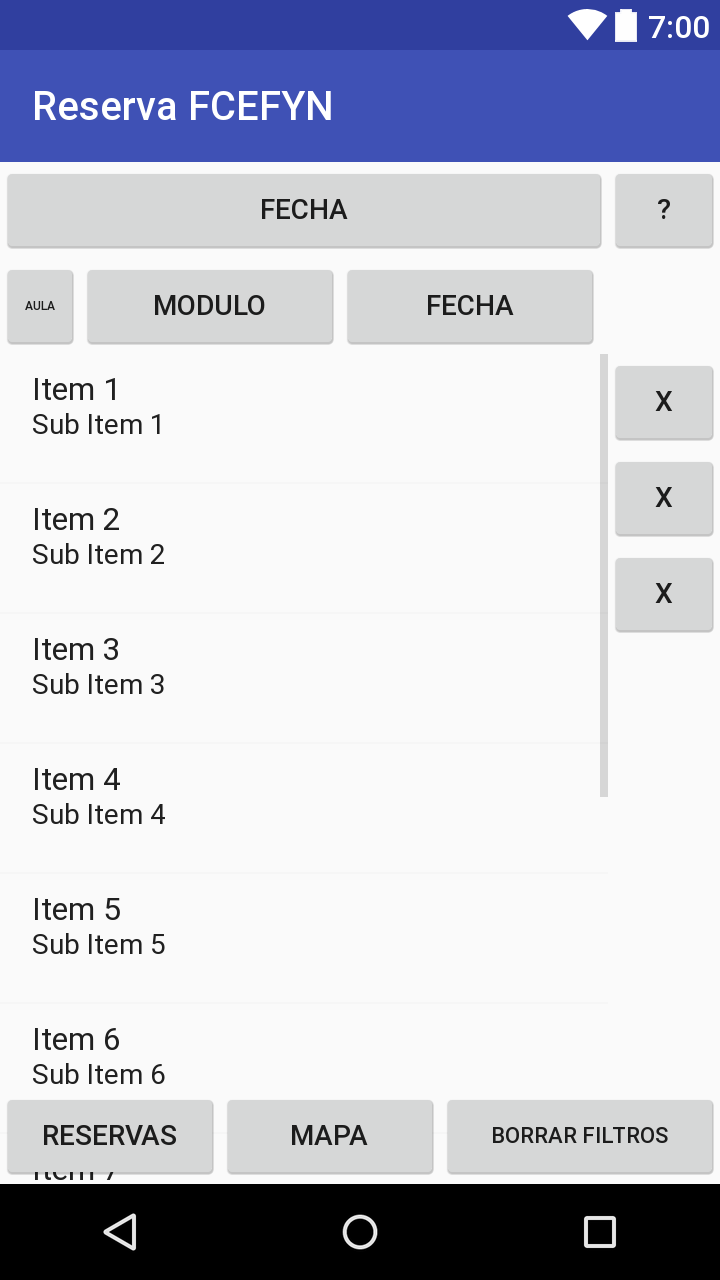
Mediante le botón “Reservas” se accederá a una vista de “Identificación”, la cual posee campos para completar con un usuario y contraseña y un botón “Ingresar”. Luego de ser validados los datos ingresados, se pasa a una vista de la sección Reservas.



Esta es la vista de la sección “Reservas”, donde aparecen botones nuevos. Como el de “Capacidad” que permite ordenar los datos ascendente o descendentemente, “Proyector” dependiendo si el aula posee o no, y por “Módulo” como se describió anteriormente.

El botón “Consultar Reservas” permite cambiar a otra vista donde se muestran las reservas realizadas por el docente logueado.

Por último el botón “Borrar Filtros” elimina todo filtro aplicado anteriormente.

Esta vista es luego de haber seleccionado el botón “Consultar Reservas”. Como vemos, también vuelven a cambiar algunos botones. Aparece un botón fecha el cual permite ordenar las reservas del docente, y los demás botones ya fueron descriptos.

En la parte lateral derecha aparecen botones “X” cuya función es cancelar las reservas.

Luego de seleccionar alguno de éstos, aparecerá una ventana de confirmación, al igual que cuando se realiza la reserva de un aula. Dicha ventana tendrá 2 opciones, “Confirmar” o “Cancelar” la acción.

Por último para mostrar un ejemplo del comportamiento de la aplicación al iniciar, se adjunta una última captura, donde se pueden ver los campos y botones descriptos en la vista inicial, pero además, se muestran las reservas para un día determinado.

## Componentes.

Android ofrece una gran cantidad de componentes GUI útiles. Los componentes que se van a utilizar en este proyecto (para este prototipo) incluyen son los siguientes:

* **ListView:** El componente ListView se utiliza para mostrar una lista de texto con una barra de desplazamiento. Permite al usuario desplazarse y seleccionar los objetos que aparecen en la lista.
* **TextView:** El componente TextView se utiliza para mostrar texto de forma lineal. Usado para ubicar en el ListView cada dato de las tablas.
* **Button:** El componente Button permite a los usuarios interactuar empujándolo. Cuando se pulsa un botón crea un evento que puede ser manejado por un EventHandler fuera de la clase Button. Nuestro sistema manejará la mayor parte de su interacción con los usuarios a través de botones, donde cada uno tendrá un EventHandler para manejar esta interacción.
* **RelativeLayout:** Este componente permite la colocación arbitraria de componente dentro de los límites. Nuestro sistema utiliza este componente en cada uno de sus menús para organizar los botones y texto de una manera lógica.

Diseño de Componentes.

Nuestro sistema se divide en 2 componentes principales: una aplicación Android y una base de datos SQLite.

La aplicación Android, a su vez se divide en 3 partes: el componente funcional o controlador (escrito en Java), el modelo (escrito en Java) y el componente gráfico o visto (escrito en XML).

El controlador forma el núcleo de la aplicación. Proporciona funcionalidad a todos los botones, cuadros de texto y otros elementos en pantalla que permiten al usuario acceder a todas las funciones proporcionadas por la aplicación. Recibe la entrada y aplica filtros, realiza reservas, cambios de vista, etc.

La vista, como su nombre lo indica, es simplemente la interfaz gráfica de usuario. Muestra todos los botones, cuadros de texto y otros elementos en pantalla que permiten al usuario acceder a todas las funciones proporcionadas por la aplicación.

El componente de base de datos SQLite, que proporciona almacenamiento centralizado para datos sincronizados.

## Clase BDDHelper.

Esta clase es la encargada de leer la BDD desde el archivo SQLite. Para ello extiende de SQLiteAssetHelper, el cual es un manejador de BDD que permite levantarla de forma automatizada.

Para su funcionamiento se debe colocar el archivo. sqlite dentro de una carpeta llamada “assets/database” dentro del directorio de la aplicación. Luego, pasando como parámetro el nombre del archivo de BDD dentro del constructor de esta clase, la carga de la BDD se realizará automáticamente.

**dbh** = **new** BDDHelper(**this**);

## Clase GestorDeDatos.

Una vez leída la BDD, esta clase posee métodos para manejarla y realizar consultas. La instancia de la clase anterior permite obtener una BDD “readable” a través de un método propio de la librería.

db = dbh.getReadableDatabase();

Este objeto de de la clase SQLiteDatabase, posee todos los métodos que permiten realizar las consultas a la base de datos y las cuales se almacenarán en variables de tipo Cursor.

**public** Cursor getReservas(String fecha) {...}

**public** Cursor getReservasDocente(String idocente) {...}

**public** Cursor getAulasLibres(String fecha) {...}

Un cursor es una colección de filas que se obtienen luego de realizar una consulta a la BDD. Estas variables cursores son las que se almacenarán en las clases que componen al modelo.

Dichas consultas se realizan a través de la función rawQuery(select, **null**) perteneciente a la librería SQLiteDatabase. Ésta recibe como parámetro un String llamado “select”, el cuál posee la consulta en leguaje SQLite. Por ejemplo, para el caso de las reservas:

**public** Cursor getReservas(String fecha) {  
 String select = **"select idreserva as \_id, numero as Aula, (apellido || ', '||nombre) as Docente, nombremat as Materia,(hora\_inicio||'-'||hora\_fin) as Modulo from reserva r\n"** +  
 **"join aula a join docente d join materia mat join modulo m \n"** +  
 **"where r.idaula = a.idaula and r.idocente = d.idocente and r.idmateria = mat.idmateria and r.idmodulo = m.idmodulo and r.fecha\_reserva = "** + fecha;  
 Cursor c = **db**.rawQuery(select, **null**);  
 **return** c;  
}

## Clases AulasLibres, ResDocente, Reservas.

Estas clases son parte del componente Modelo. Poseen una variable de tipo cursor en la que se almacenará la consulta de cada una, junto con un método que retorna dicho cursor.

* Para el caso de AulasLibres, la consulta devolverá un cursor cuyas filas son campos de número de aula, capacidad, proyector y módulo en una fecha determinada.

**public** AulasLibres (GestorDeDatos gdd, String fecha){  
 **libres** = gdd.getAulasLibres(fecha);  
}  
  
**public** Cursor obtenerAulasLibres () {  
 **return libres**;  
}

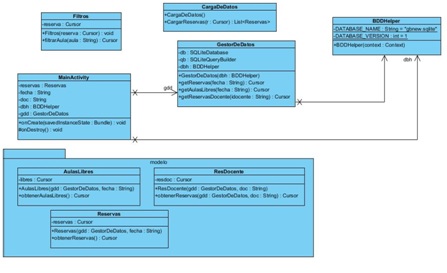
* Para el caso de ResDocente, la consulta devolverá un cursor cuyas filas son campos de aula, módulo y fecha para un docente determinado que debe ir como parámetro.

**public** ResDocente (GestorDeDatos gdd, String doc) {  
 **resdoc** = gdd.getReservasDocente(doc);  
}  
  
**public** Cursor obtenerReservas (GestorDeDatos gdd, String doc) {  
 **return resdoc**;  
}

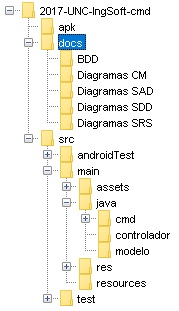
* Y por último Reservas, la consulta devolerá un cursor cuyas filas son campos de aula, docente, materia y módulo en una fecha determinada.

**public** Reservas(GestorDeDatos gdd, String fecha) {  
 **reservas** = gdd.getReservas(fecha);  
}  
  
**public** Cursor obtenerReservas() {  
 **return reservas**;  
}

Por último se adjunta un diagrama de clases preliminar, de las funciones del sistema implementadas hasta el momento.



Y el esquema de directorios hasta el momento es:



Unit Test.

Cada clase se probará individualmente para asegurarse de que las funciones y constructores están funcionando correctamente. Luego, una vez que el programa se ensambla, se probarán conjuntos de funciones con las pruebas de integración y por último todo el conjunto para asegurar que todos los componentes funcionen correctamente.

## Casos de prueba y resultados esperados.

Los tipos de pruebas a realizar se especifican a continuación, incluyendo tanto detalle como sea posible en esta etapa. El énfasis aquí está en Black-box y White-box Testing:

* **White-box Testing:** En este método de prueba de software el desarrolador conoce la estructura interna del software que va a probar. Mientras se implementa cada clase, el desarrollador asignado a esa clase probará para asegurarse de que cada función está funcionando. El desarrollador es totalmente responsable de depurar su propio código porque la sobrecarga de compartir código entre desarrolladores se ha considerado demasiado costosa. Sin embargo, todo el código será accesible a través del sistema de control de versiones proporcionado, por lo que esta regla puede ser violada si es necesario.
* **Black-box Testing:** En cambio en este método de prueba, el Tester es consciente de lo que el programa debe hacer, pero no tiene el conocimiento de cómo lo hace. Comprende la mayoría del proceso de prueba. Esto se realizará después de que todos los componentes estén montados y consistirá en ejecutar todas las situaciones posibles que puedan ocurrir en el uso de la aplicación.

## Pruebas unitarias.

* **Carga de la BDD:**
* **Consulta de la BDD:**