PROYECTO INTEGRADO: Praxis

2ºDAM/I.E.S. Zaidín-Vergeles

Sergio Contreras Plata 2014/15

ÍNDICE

CÓDIGO FUENTE

El código fuente completo de la aplicación junto con su documentación se encuentra en la siguiente dirección:

https://www.dropbox.com/sh/kki03zg8xvvaso4/AAAK_4uzQRNs7V2wLbQTL8YYa?dl=0

OBJETIVO DEL PROYECTO

El objetivo del proyecto es proporcionar una herramienta para que los estudiantes de diferentes perfiles sanitarios puedan gestionar las prácticas clínicas que realizan en centros hospitalarios del Servicio Andaluz de Salud.

El alumno podrá solicitar la realización de las prácticas ofertadas, realizar un seguimiento diario (e incluso consultar su horario de prácticas) y finalizar el proceso evaluando la calidad de la formación recibida.

praxi

ugr Universidad de Granada

JUNTA DE ANDALUCIA
Servicio Andaluz de Salud
CONSEJERÍA DE SALUD y BIENESTAR SOCIAL

NECESIDADES Y REQUISITOS

Las necesidades y requisitos del proyecto son:

- La app realizada debe de estar dirigida al sistema operativo Android con la versión mínima 4.0.
- La app debe poder consultar el calendario de prácticas de un alumno.
- La app debe poder evaluar la labor un tutor clínico al finalizar una práctica clínica.
- El servicio web REST debe implementarse mediante JPA.
- El login del alumno se realizará mediante código QR.
- La aplicación debe ser segura, accesible y sencilla de usar.
- Los usuarios deben encontrar atractiva la aplicación.

DISEÑO DE LA SOLUCIÓN

Para diseñar la aplicación tendremos en cuenta las dos partes esenciales del proyecto. Por un lado, el servidor que mediante servicios web REST permitirá conectarse con la base de datos y realizar operaciones sobre ella, y por otro lado, el cliente en Android que permitirá comunicarnos con el servidor.

DISEÑO DE LA BASE DE DATOS

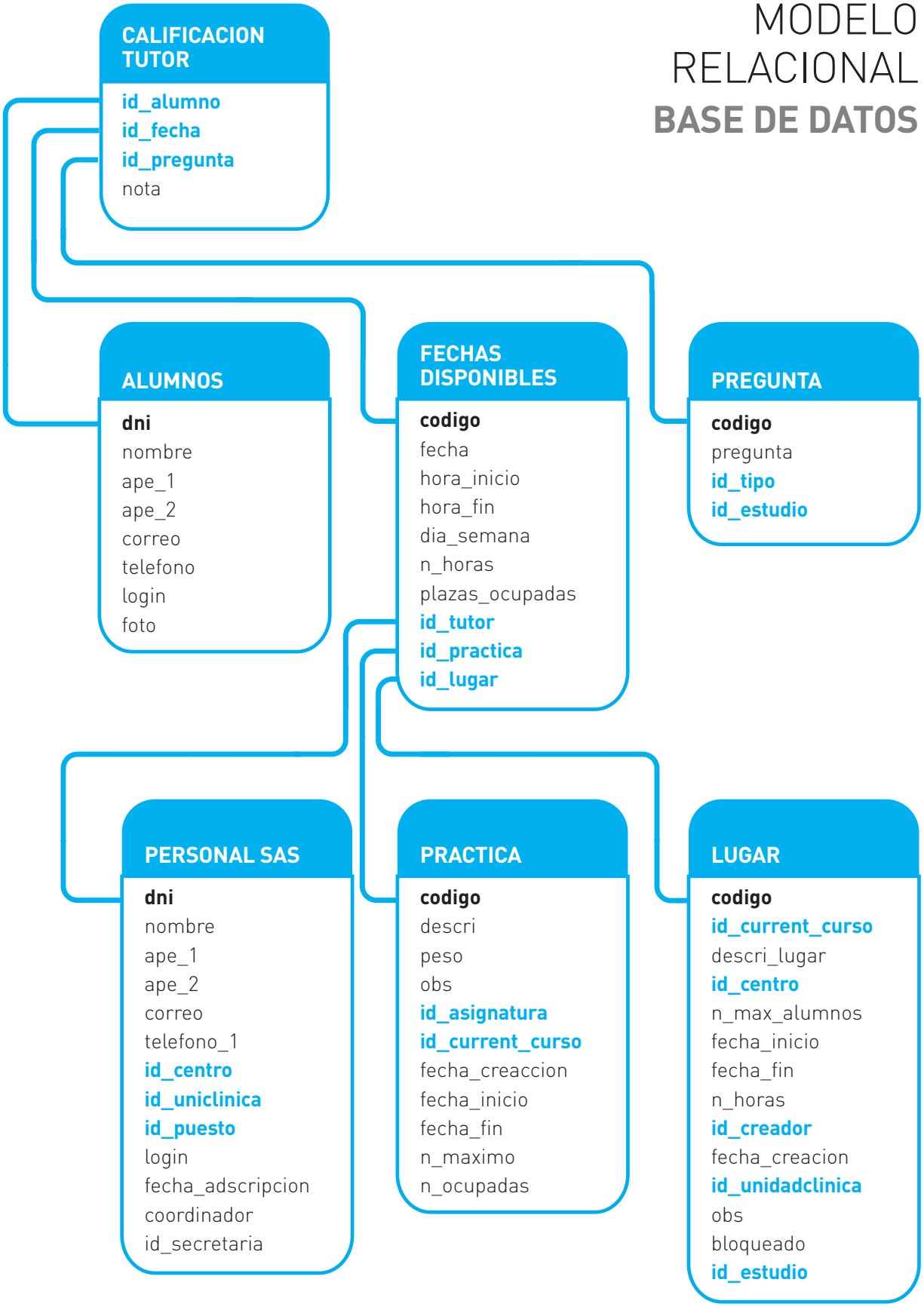
La base de datos de la aplicación se implementó en su momento para el cliente web de Praxis por lo que no era necesario la creación de una nueva y por lo tanto se utilizará la base de datos ya existente.

Tablas

La base de datos contiene un total de 66 tablas de las cuales usaremos prioritariamente:

- PRA_CALIFICACION_TUTOR: almacena las notas que un alumno le da a un tutor clínico para una pregunta en una fecha determinada.
PRA_DET_FECHASDISPONIBLES: almacena las fechas en las que hay prácticas así como el horario, asignatura, tutor y lugar de la misma y las prácticas para esa fecha
PRA_DET_LUGARES: almacena los lugares donde se realizan las prácticas clínicas.
PRA_PRACTICA_CLINICA: almacena la información de una práctica clínica.
PRA_PREG_EVAL_DOCENTE: almacena las preguntas que se hacen para evaluar a un tutor según el grado en el que esté matriculado el alumno
PRA_RELA_FECHA_ALUMNO: almacena las fechas que se le asignan a cada alumno.
PRA_SAS_CENTRO: almacena los centros hospitalarios donde se realizan las prácticas clínicas.
PRA_SAS_PERSONAL: almacena los datos de los tutores clínicos que evalúan las prácticas.
PRA_UGR_ALUMNOS: almacena los datos de los alumnos de la UGR.
PRA_UGR_ASIGNATURAS: almacena las asignaturas pertenecientes a un grado universitario.
PRA_UGR_ESTUDIOS: almacena los grados universitarios.

MODELO RELACIONAL
BASE DE DATOS



DISEÑO DE LAS INTERFACES

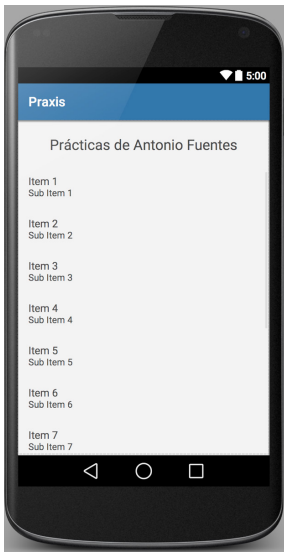
Para el diseño de interfaces se optó por seguir las nuevas guías de diseño ofrecidas por Google para Material Design en Android L y hacer que éste sea compatible con dispositivos con una versión anterior a la 5.0.

Ventana de login

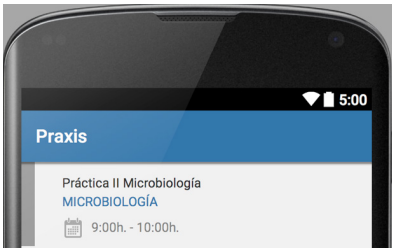
Se ha optado por un degradado de fondo con el logotipo de la app, las instrucciones para escanear el código QR y un botón que lanza la captura del código QR mediante la cámara.

Ventana principal y detalle del ítem de la lista

Se ha optado por una lista deslizable con el nombre del alumno y con varios menús en la barra de acción.

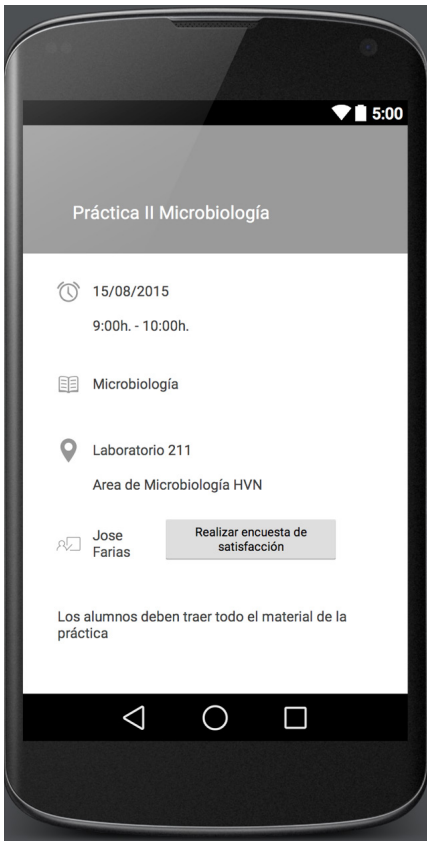


Se ha optado por mostrar en etiquetas de texto el título de la práctica, el nombre de la asignatura junto con el lugar y la fecha de la misma. También se ha incluido una franja de color que estaba pensada para que el alumno pudiera escoger el color de la práctica según su preferencia y que por defecto aparece gris.



Ventana secundaria

Se ha optado por una Toolbar con tamaño aumentado y el título de la práctica. El color de la Toolbar variaría en función del color que el usuario le hubiera asignado a la práctica. En el detalle se muestra en etiquetas de texto la fecha y horario de la práctica, la asignatura, la ubicación, el tutor que evalúa la práctica y en caso de que haya el botón de evaluar estará activado y las observaciones realizadas por el tutor.



Diálogo de encuesta y detalle ítem de la encuesta

Se ha optado por una ventana de diálogo con una lista de preguntas. Para cada pregunta aparece el título de la misma y una RatingBar con la que se puede calificar al tutor.

LÓGICA DE NEGOCIO

A la hora de implementar la aplicación se requiere un servidor que ataca la base de datos y un cliente que manda peticiones al servidor y recibe su respuesta mostrando los datos obtenidos. Empecemos viendo la implantación del servidor.

SERVIDOR WEB

El servidor web correrá bajo un Apache Tomcat. Se ha decidido usar la tecnología REST (Representational State Transfer) ya que su implementación es simple, ligera y rápida en comparación con SOAP. Se ha decidido usar el Framework Spring Data REST que permite exponer los repositorios JPA y generar los servicios web REST. Primero hay que generar los DTO (Data transfer object) que se usarán como entidades JPA. Mas tarde hay que implementar una capa de acceso a datos con los DAO (Data Access Object) que servirá como repositorio de Spring Data JPA. Una vez tengamos nuestro repositorio JPA se generan los servicios web REST a través de él.

Conexión a la base de datos

Para almacenar, organizar y obtener los datos la mayoría de las aplicaciones usan una base de datos relacional. Java EE permite acceder a las bases de datos mediante JDBC usando un objeto DataSource. Un objeto DataSource tiene un conjunto de propiedades (tales como la localización del servidor de la base de datos, el nombre de la base de datos, el protocolo de red, etc.) que identifica y describe el origen de datos real que representa. Un objeto DataSource puede nombrarse con un JNDI para que la API de persistencia de Java acceda a la base de datos a través de el fácilmente.

Para ello hay que crear un pool de conexiones a través del origen de datos y especificarlo en la unidad de persistencia. En nuestro caso el archivo persistencia.xml contendrá las entidades, el origen de datos y algunas propiedades adicionales:

```
<jta-data-source>ojdbc/praxis</jta-data-source>
```

Entidades

Objeto de Transferencia de Datos (DTO) es un objeto que transporta datos entre procesos. La motivación de su uso tiene relación con el hecho que la comunicación entre procesos es usualmente realizada mediante interfaces remotas (ej. Servicios Web), donde cada llamada es una operación costosa. Como la mayor parte del costo de cada llamada está relacionado con el tiempo entre el cliente y servidor, una forma de reducir el número de llamadas es usando un objeto (el DTO) que agrega los datos que habrían sido transferidos por cada llamada, pero que son entregados en una sola invocación.

Las entidades se generan mediante el IDE a través de una conexión a la base de datos.

Repositorios

Un Objeto de Acceso a Datos (DAO por sus siglas en inglés) es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo. Un DAO siempre realiza operaciones atómicas contra la base de datos, nunca son necesarias las transacciones.

En su implementación a cada DAO se le añade la anotación @Repository para indicarle a Spring que esa clase es un repositorio.

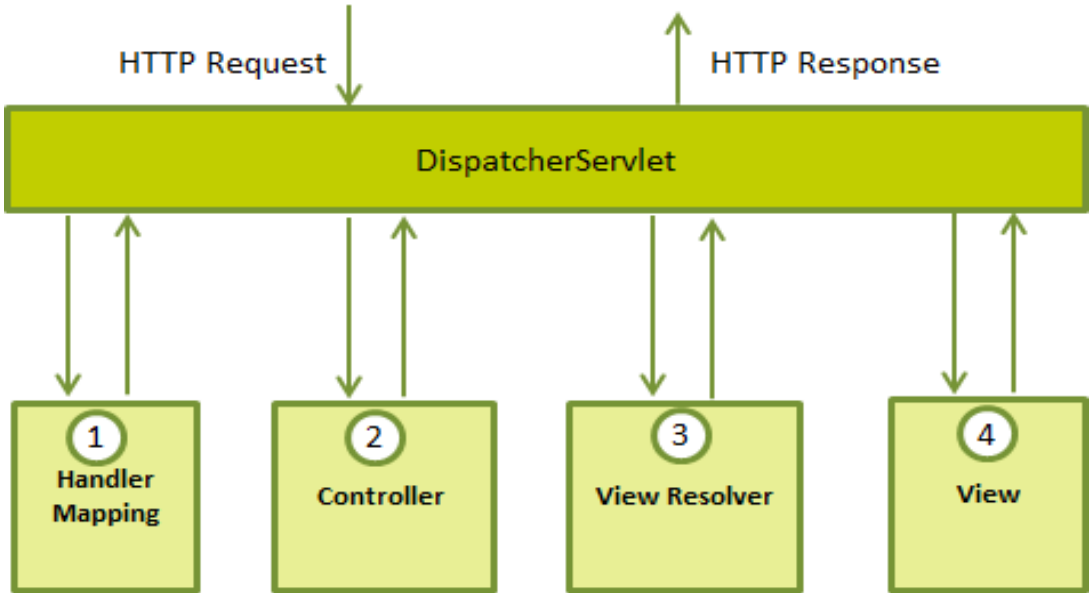
Servicios web

Para realizar operaciones contra la base de datos desde la aplicación web o móvil de Praxis se utilizan servicios web con Spring. Para ello se definen las interfaces que necesitemos y se implementan las operaciones que se realizarán mediante los repositorios de Spring.

Para indicar que nuestra clase es un servicio hay que añadir la anotación @Service a nuestras clases del servicio web.

Servlet

El modelo-vista-controlador del framework Spring está diseñado en base a un DispatcherServlet que maneja todas las peticiones y respuestas HTTP tal y como se ve en el siguiente diagrama:



Se necesita mapear las peticiones que el Servlet queremos que trate especificándolo en el archivo web.xml. En nuestro caso queremos que trate las peticiones relacionadas con los servicios web REST:

```
<servlet>
	<servlet-name>mvc-dispatcher</servlet-name>
	<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
	<load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
		<servlet-name>mvc-dispatcher</servlet-name>
		<url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

Controlador

Para manejar las peticiones HTTP se define un controlador que servirá para especificar que acción se realizará dependiendo de la petición que le llega.

Para ello a nuestra clase se le añade la anotación @RestController y se le inyectan los servicios web mediante la anotación @Autowired.

Con la anotación @RequestMapping nos aseguramos que las peticiones HTTP con una determinada URL se procesen con el método deseado.

CLIENTE ANDROID

La app en Android se usará para mostrar las prácticas disponibles para un alumno y calificar al tutor clínico que le haya evaluado esa práctica.

Los datos se obtendrán a través de un cliente REST que usará los servicios web definidos en el servidor web. Para realizar este cliente se ha optado por usar el framework Retrofit, que permite elaborar una API REST mediante una interfaz en Java de una manera rápida, sencilla y segura.

Funcionamiento de la app

Cuando el usuario abra la app por primera vez se le pedirá que escanee un código QR que se generará en su perfil de la aplicación web. Este código es una combinación de la URL con la base del servidor web + el login del usuario encriptado mediante AES.

Una vez iniciada la sesión del alumno, se cargará un listado de prácticas disponibles ordenadas por fecha. En la barra de acción el alumno podrá filtrar las prácticas por día, semana o mes, así como mostrar la ayuda o cerrar la sesión. Si se deja pulsado un ítem de la lista se podrá añadir a Google Calendar.

Para ver el detalle de una práctica clínica hay que pulsar sobre el ítem y se nos mostrarán los detalles en una ventana nueva o a la derecha del listado dependiendo de la orientación del dispositivo.

Si hay disponible un tutor para evaluar, el botón de evaluar tutor estará activado y cuando se pulse se mostrará un diálogo con las preguntas a calificar dependiendo del grado universitario en el que esté matriculado el alumno.

Librerías

Se han añadido las siguientes librerías con Gradle:

- Librerías de soporte en Android:
    'com.android.support:appcompat-v7:21.0.+'
    'com.android.support:support-v4:21.0.3'

- Retrofit:
    'com.squareup.retrofit:retrofit:1.9.0'

- Jackson JSON Converter
    'com.fasterxml.jackson.core:jackson-databind:2.5.3'

- Soporte Jackson con Retrofit:
    com.squareup.retrofit:converter-jackson:1.9.0'

- Soporte OkHttp con Retrofit:
    'com.squareup.okhttp:okhttp:2.3.0'

- Lector de códigos QR integrado ZXING:
    'com.journeyapps:zxing-android-embedded:2.3.0@aar'
    'com.journeyapps:zxing-android-legacy:2.3.0@aar'
    'com.journeyapps:zxing-android-integration:2.3.0@aar'
    'com.google.zxing:core:3.2.0'

- Android TextView autoescalable:
    'me.grantland:autofittextview:0.2.1'

Manifiesto

En el manifest.xml se han definido tres actividades y se conceden los permisos de Internet, acceso al estado de la red, lectura y escritura en memoria y lectura y escritura en el calendario.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="es.hvn.praxis" >

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.WRITE_CALENDAR" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".actividades.Principal"
            android:label="@string/app_name" >
        </activity>
        <activity
            android:name=".actividades.Secundaria"
            android:label="@string/title_activity_secundaria"
            android:theme="@style/NoActionBar">
        </activity>
        <activity
            android:name=".actividades.Login"
            android:label="@string/title_activity_login" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

POJOS

La app usa dos objetos Java principalmente. El objeto RestAlumno y PregEvalDocente.

RestAlumno

Esta clase representa los datos de un alumno y su lista de prácticas. Se ha reducido al mínimo esta clase ya que se partía de la clase Alumno que dependía de 32 clases mas debido a las claves foráneas de la base de datos y esto daba problemas a la hora de procesar la respuesta JSON ya que era demasiado grande. Sus propiedades son:

```
String dniAlumno;
String nombreCompleto;
int codigoEstudio;

List<RestFechaDisponible> listaPracticas;
```

Contiene sus respectivos getter y setter y un constructor por parámetros. Para que el conversor Jackson pueda parsear la respuesta que le llega en JSON y convertirla a un objeto de la clase AlumnoRest, se añaden las anotaciones @ JsonProperty a cada propiedad de la clase y en los parámetros del constructor.

RestFechaDisponible

Esta clase representa una práctica clínica. Como se ha mencionado anteriormente esta clase partía de la clase FechaDisponible, pero al depender de tantas clases se ha reducido al mínimo con los datos de una práctica, asignatura, lugar y tutor. Sus propiedades son:

```
Integer codigo;
Date horaInicio;
Date horaFin;
String descripcionPractica;
String observaciones;
String descripcionAsignatura;
String descripcionLugar;
Integer plazasocupadas;
Integer nmaxalumnos;
String centroSas;
String nombreCompletoTutor;
```

Contiene sus respectivos getter y setter y un constructor por parámetros. Para que el conversor Jackson pueda parsear la respuesta que le llega en JSON y convertirla a un objeto de la clase FechaDisponible, se añaden las anotaciones @ JsonProperty a cada propiedad de la clase y en los parámetros del constructor.

PregEvalDocente

Esta clase representa una pregunta de la encuesta de satisfacción. Sus propiedades son:

```
Integer codigo;
String pregunta;
Estudio estudio;
PregEvalDocenteTipo tipo;
```

Contiene sus respectivos getter y setter y un constructor por parámetros. Para que el conversor Jackson pueda parsear la respuesta que le llega en JSON y convertirla a un objeto de la clase PregEvalDocente, se añaden las anotaciones @ JsonProperty a cada propiedad de la clase y en los parámetros del constructor.

PregEvalDocenteTipo

Esta clase representa el tipo de pregunta de la encuesta de satisfacción. Sus propiedades son:

```
Integer codigo;
String descri;
```

Contiene sus respectivos getter y setter y un constructor por parámetros. Para que el conversor Jackson pueda parsear la respuesta que le llega en JSON y convertirla a un objeto de la clase PregEvalDocenteTipo, se añaden las anotaciones @JsonProperty a cada propiedad de la clase y en los parámetros del constructor.

Actividades

Login

La actividad Login es la principal de la app y se encargará de comprobar que el usuario inicie sesión correctamente al arrancar la app.

Para realizar el inicio de sesión, el usuario tendrá que escanear desde la app un código QR que se le proporcionará desde el perfil de la aplicación web Praxis. En caso de que ya lo ha hecho se cargará el login del usuario almacenado en Shared Preferences y se lanzará la actividad Principal.

```
private void login() {
        android.content.SharedPreferences sp = getSharedPreferences(getString(R.string.
sp_dni), Context.MODE_PRIVATE);
        String dni = getString(R.string.sp_dni);
        if(!getFromSharedPreferences(sp, dni).equals("null")){
            Intent intent = new Intent(Login.this, Principal.class);
            intent.putExtra(getString(R.string.idalumno), getFromSharedPreferences(sp,
dni));
            startActivity(intent);
            this.finish();
        } else {
            setContentView(R.layout.activity_login);
        }
    }
```

Si se ha cargado la interfaz de login el usuario podrá pulsar sobre el botón de escanear QR y se lanzará una nueva Actividad que permitirá escanear mediante la librería integrada ZXING.

```
new IntentIntegrator(this).initiateScan();
```

El resultado del scan se obtendrá en onActivityResult y se desencriptará mediante AES utilizando la clase auxiliar AES incluida en el paquete utils del proyecto.

```
AES.decrypt(input, key);
```

Una vez desencriptada y comprobando que los valores sean correctos se almacenan en Shared Preferences y se vuelve a comprobar el inicio de sesión lanzando la actividad Principal.

Principal

En la actividad principal se cargan los fragmentos Lista y Detalle dependiendo de la orientación del dispositivo y se le asigna un escuchador al fragmento Lista que nos permitirá controlar los clics que se realicen sobre los ítems de la lista.

```
@Override
    protected void onPostCreate(Bundle savedInstanceState) {
        super.onPostCreate(savedInstanceState);
        Lista fragmentLista = (Lista)getSupportFragmentManager().findFragmentById(R.
id.fragment_lista);
        fragmentLista.setEscuchador(this);
        Detalle fragmentDetalle = (Detalle)getSupportFragmentManager().
findFragmentById(R.id.fragment_detalle);
        horizontal = fragmentDetalle != null && fragmentDetalle.isInLayout();
    }

@Override
    public void onItemSelected(RestFechaDisponible practica) {
        if(horizontal) {
           ((Detalle)getSupportFragmentManager().findFragmentById(R.id.fragment_
detalle)).setDetalle(practica);
        } else {
            // Mostrar detalle en actividad Secundaria
            Intent intent = new Intent(Principal.this, Secundaria.class);
            intent.putExtra("practica", practica);
            startActivity(intent);
        }
    }
```

Secundaria

En esta actividad se carga el fragmento Detalle y se carga el detalle de la práctica que le llega como parámetro.

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_secundaria);
        Detalle fragmentDetalle = (Detalle)getSupportFragmentManager().
findFragmentById(R.id.fragment_detalle);
        if(fragmentDetalle != null && fragmentDetalle.isInLayout()) {
            RestFechaDisponible practica = (RestFechaDisponible)getIntent().
getSerializableExtra("practica");
            fragmentDetalle.setDetalle(practica);
        }
    }
```

Adaptadores

En la app se usan dos adaptadores personalizados para los ítems de los ListView reutilizando los componentes de la interfaz con el patrón ViewHolder para reducir la carga de la aplicación.

En el AdaptadorPracticas se muestra el detalle de la práctica y en el AdaptadorEncuesta se muestra la pregunta de la encuesta y un RatingBar para calificar al tutor. Este RatingBar tiene un escuchador que manda un Broadcast al fragmento Detalle cuando se produce un cambio de calificación.

Fragmentos

La app dispone de dos fragmentos independientes que controlan las acciones que se realizan sobre ellos usando el patrón Lista/Detalle.

Lista

En este fragmento se hace una llamada al servicio web de alumnos para obtener los datos del alumno según el login.

Para llevarlas a cabo se usa nuestra ApiRest con Retrofit que contiene los métodos necesarios para atacar la base de datos. Para inicializar nuestra API le indicamos la URL base del servidor web, se le asigna un cliente OkHttp que se encarga de las peticiones y respuestas HTTP, un conversor que nos permite parsear la respuesta en JSON y convertirla a objetos Java y se le añade una cabecera para todas las peticiones:

```
        RequestInterceptor requestInterceptor = new RequestInterceptor() {
            @Override
            public void intercept(RequestInterceptor.RequestFacade request) {
               request.addHeader("Accept", "application/json");
            }
        };

        RestAdapter restAdapter = new RestAdapter.Builder()
                .setEndpoint(URL_BASE)
                .setClient(new OkClient(getClient()))
                .setConverter(new JacksonConverter())
                .setLogLevel(RestAdapter.LogLevel.FULL)
                .setRequestInterceptor(requestInterceptor)
                .build();

        api = restAdapter.create(ApiRest.class);
```

Una vez inicializada la ApiRest, para obtener los datos del alumno hacemos una llamada al método infoAlumno() pasándole el ID del alumno como parámetro y después se inicializará el ListView con las prácticas obtenidas.

```
RestAlumno alumno = api.infoAlumno(idAlumno);
```

Para añadir una práctica a Google Calendar cuando se selecciona un ítem de la lista, se lanza un intent con los datos del evento:

```
Intent intent = new Intent(Intent.ACTION_INSERT)
                    .setData(CalendarContract.Events.CONTENT_URI)
                    .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, cal.
getTimeInMillis())
                    .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, cal2.
getTimeInMillis())
                    .putExtra(CalendarContract.Events.TITLE, restFechaDisponible.
getDescripcionPractica())
                    .putExtra(CalendarContract.Events.EVENT_LOCATION,
restFechaDisponible.getDescripcionLugar());
            startActivity(intent);
```

Detalle

En este fragmento se muestran los detalles de la práctica al hacer clic sobre un ítem del fragmento lista utilizando el método setDetalle() que se llama desde el fragmento lista. La actividad Principal se encargará de mostrar este fragmento en la actividad Secundaria o en ella misma dependiendo de la orientación.

Una vez cargados los detalles, si existe un tutor para evaluar, el botón para hacer la encuesta de satisfacción del tutor estará habilitado. Al pulsar sobre él se hará una llamada al método preguntasEvaluacion() de nuestra API REST pasándole el código del grado matriculado como parámetro.

```
preguntasTutor = (ArrayList<PregEvalDocente>) api.preguntasEvaluacion(params[0]);
```

Una vez cargadas las preguntas de la encuesta se mostrará un diálogo con una lista de preguntas para evaluar. Al pulsar el botón aceptar se hará una llamada al método guardaCalificacionTutor de la API REST con la nota, el alumno, la pregunta y la fecha de la práctica para almacenarla en la base de datos.

```
api.guardaCalificacionTutor(idAlumno, idFecha, idPregunta, nota);
```

Para guardar las calificaciones desde el diálogo de la encuesta se usa un BroadcastReceiver que está escuchando cuándo varía la puntuación del RatingBar:

```
private BroadcastReceiver receptor = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            if(getResultCode() == Activity.RESULT_OK) {
                Bundle b = intent.getExtras();
                Encuesta encuesta = (Encuesta) b.getSerializable(getString(R.string.
intent_rating));
                calificaciones.set(encuesta.getPosition(), encuesta);
                Log.v("position", encuesta.getPosition()+"");
                Log.v("rating", encuesta.getRating());
            }
        }
    };
```

Clases de acceso a datos

Para el acceso a datos se ha implementado una interfaz con Retrofit que permitirá realizar las peticiones GET y POST por HTTP al servidor web. Para ello se usan las anotaciones de Retrofit @GET y @POST en los métodos de la interfaz.

```
public interface ApiRest {

    @GET("/getInfoAlumno/{dni}")
    public RestAlumno infoAlumno(@Path("dni") String dni);

    @GET("/getPreguntasEvaluacion/{codigo}")
    public List<PregEvalDocente> preguntasEvaluacion(@Path("codigo") Integer codigo);

    @POST("/guardaCalificacionTutor/{idAlumno}/{idFecha}/{idPregunta}/{nota}")
    public void guardaCalificacionTutor(@Path("idAlumno") String idAlumno,
                                        @Path("idFecha") Integer idFecha,
                                        @Path("idPregunta") Integer idPregunta,
                                        @Path("nota") String nota,
                                        Callback<Boolean> success);


}
```

Utilidades

Se han añadido las clases auxiliares SharedPreferences y AES en el paquete utils para poder trabajar con SharedPreferences y desencriptar con AES fácilmente.

IMPLEMENTACIÓN Y DESPLIEGUE DE LA APLICACIÓN.

Para la implementación de la aplicación se han utilizado los programas Android Studio e IntelliJ IDEA.

El servidor y aplicación web están alojados en un servidor Tomcat del hospital.

La aplicación Android está en proceso de subida a la Google Play Store en la cuenta de la Junta de Andalucía.

PRUEBAS

La aplicación se ha probado entre el Departamento de Desarrollo del Hospital Virgen de las Nieves y varios alumnos de la Facultad de Medicina para sugerir cambios y mejoras.

BIBLIOGRAFÍA

Praxis: http://www.hospitalgranada.es/normopraxis/
Material Design: http://www.google.com/design/spec/material-design/introduction.html
Spring Data JPA: http://projects.spring.io/spring-data-jpa/
Spring Data REST: http://projects.spring.io/spring-data-rest/
Spring Web MVC: http://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm
Spring RESTful Web Services: https://spring.io/guides/gs/rest-service/
Retrofit: http://square.github.io/retrofit/
Jackson JSON converter: http://jackson.codehaus.org/
Compatibilidad Material Design: https://chris.banes.me/2014/10/17/appcompat-v21/
Procesamiento códigos QR: https://github.com/zxing/zxing
JavaEE: http://docs.oracle.com/javaee/6/api/overview-summary.html
RESTful Web Services: https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html
Data Sources and Connection Pools: http://docs.oracle.com/javaee/6/tutorial/doc/bncjj.html
DTO: http://es.wikipedia.org/wiki/Objeto_de_Transferencia_de_Datos_%28DTO%29
DAO: http://es.wikipedia.org/wiki/Data_Access_Object