

ULL-AR

Tecnología de realidad aumentada en entornos universitarios

Alejandro Hernández Padrón

TRABAJO DE FIN DE GRADO
La Laguna, 16 de Julio, 2018

Índice

- 1 Introducción
- 2 Herramientas y Tecnologías utilizadas
- 3 RA y RA en entornos universitarios
- 4 La aplicación ULL-AR
- 5 Back-end de la aplicación
- 6 Presupuesto
- 7 Summary and conclusions

Índice

- 1 Introducción
- 2 Herramientas y Tecnologías utilizadas
- 3 RA y RA en entornos universitarios
- 4 La aplicación ULL-AR
- 5 Back-end de la aplicación
- 6 Presupuesto
- 7 Summary and conclusions

Objetivos

- Programación de aplicaciones en Android.
- Realidad aumentada.
- Servicios de computacion en la nube.
- Repositorio online.
- Creación de una memoria técnica.
- La aplicación ULL-AR.

Índice

- 1 Introducción
- 2 Herramientas y Tecnologías utilizadas
- 3 RA y RA en entornos universitarios
- 4 La aplicación ULL-AR
- 5 Back-end de la aplicación
- 6 Presupuesto
- 7 Summary and conclusions

Herramientas y Tecnologías utilizadas

Android Studio

- Entorno de desarrollo integrado.
- Gradle.
- Sistema de depuración sencillo e intuitivo.

Herramientas y Tecnologías utilizadas

Node.js

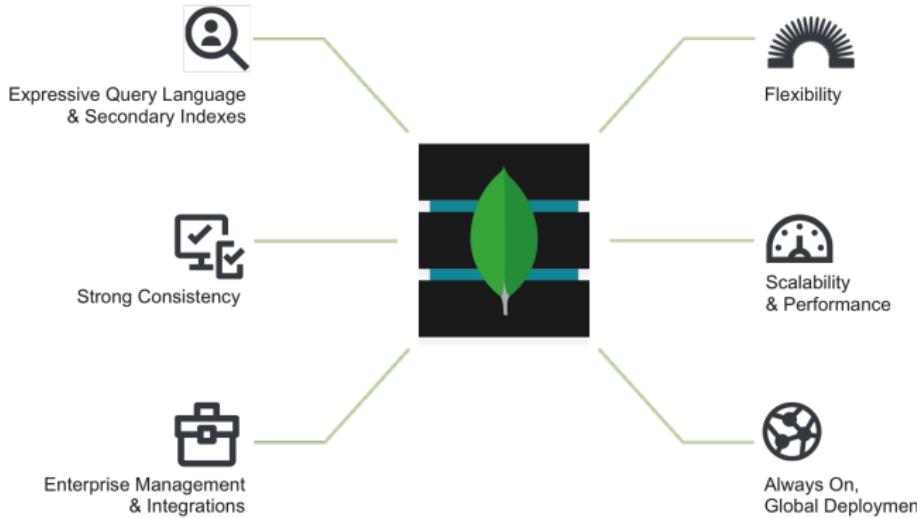
- ¿Qué es?.
- ¿Por qué se ha decidido utilizar esta tecnología?.



Herramientas y Tecnologías utilizadas

MongoDB

- Base de datos NoSQL.
- De esquema libre.
- ¿Porqué se ha decidido utilizar MongoDB?.



Herramientas y Tecnologías utilizadas

Heroku

- Plataforma como servicio de computación (PaaS) en la nube.
- Ejecuta las aplicaciones en “dynos”.
- Servidor de ULL-AR.



Herramientas y Tecnologías utilizadas

mLab

- Servicio de base de datos en la nube.
- Ofrece bases de datos MongoDB.
- Ejecuta sus máquinas en proveedores de servicios en la nube como AWS, Azure y Google Cloud.
- Base de datos de ULL-AR.



Herramientas y Tecnologías utilizadas

Google Maps

- API de Google Maps.
- Permite integrar los mapas de Google Maps en una aplicación Android.
- Permite:
 - Creación de marcadores, polígonos y superposiciones.
 - Cambiar la vista del mapa.
 - La posibilidad de elegir el tipo de mapa

Índice

- 1 Introducción
- 2 Herramientas y Tecnologías utilizadas
- 3 RA y RA en entornos universitarios
- 4 La aplicación ULL-AR
- 5 Back-end de la aplicación
- 6 Presupuesto
- 7 Summary and conclusions

Realidad Aumentada (RA)

- *¿Qué es la RA?.*
- *¿Cómo funciona esta tecnología?.*
- *¿Qué tipos de Realidad Aumentada existen?.*
- *¿Diferencias entre la Realidad Aumentada y la Realidad Virtual (RV)?.*
- *¿Qué es la Realidad Mixta?.*
- *Integración de la RA en Android Studio.*

¿Qué es la realidad aumentada?

- Permite expandir la información del mundo físico.
- Se encuentra en su mejor momento.
- Aplicaciones para todos los ámbitos.



¿Cómo funciona esta tecnología?

Componentes necesarios para la tecnología de RA:

- Dispositivo de visualización.
- Sistema de computación.
- Sensores: GPS, WIFI, Bluetooth, acelerómetro, giroscopio, cámara, etc.
- Software de RA.



¿Qué tipos de RA existen?

Existen distintos tipos de RA:

- Marker-based.
- Markerless.
- Projection-based.
- Superimposition-based.



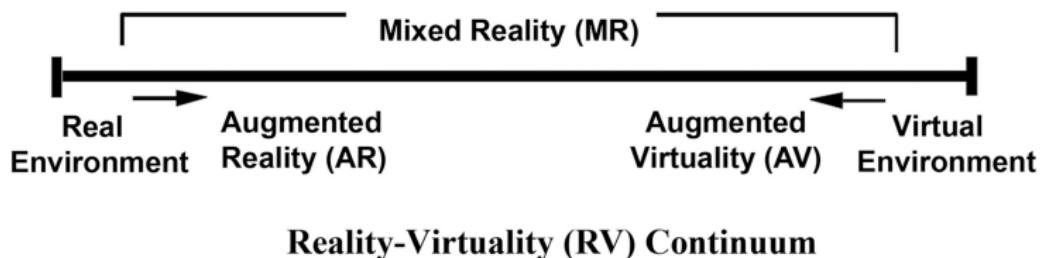
¿Diferencias entre la Realidad Aumentada y la Realidad Virtual (RV)?

- En la actualidad se encuentra más avanzada que la RA.
- Muestra al usuario un entorno de escenas y objetos de aperiencia real.
- Dispone de distintos mecanismos de interacción.
- Aleja al usuario del entorno real.



¿Qué es la Realidad Mixta?

- Es la unión de la RA con la RV.
- Consiste en llevar el mundo real al mundo virtual.
- Requiere de mayor capacidad de procesamiento que la RA.

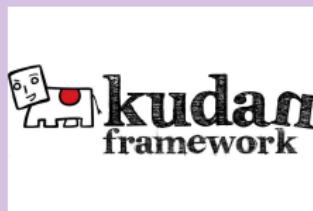


Integración de la Realidad Aumentada en Android Studio

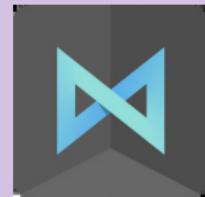
Vuforia



Kudan AR SDK



MaxST SDK



Aprendizaje

- La RA se encuentra preparada para comenzar a acercarse a las aulas.
- Capacidad para generar material didáctico y actividades para cualquier disciplina.
- Mejorar el aprendizaje.
- Experiencias más ricas e inmersivas.
- Cambia la forma en la que adquieren los contenidos de aprendizaje.

Aplicaciones en entornos universitarios

- Prácticas en laboratorios.
- Practicas de campo y visitas.
- Libros y documentos.
- Aprendizajes experimentales.
- Información sobre la universidad.

Índice

- 1 Introducción
- 2 Herramientas y Tecnologías utilizadas
- 3 RA y RA en entornos universitarios
- 4 **La aplicación ULL-AR**
- 5 Back-end de la aplicación
- 6 Presupuesto
- 7 Summary and conclusions

Requisitos de ULL-AR

- La aplicación se desarrollará para dispositivos con Android.
- Se implementarán técnicas de realidad aumentada basadas en la geolocalización.
- El servidor y base de datos de la aplicación estarán alojados en la nube.

Vídeo demostrativo de la aplicación ULL-AR

Inicio de la aplicación

Ventana de *Inicio de Sesión*

- Permitir al usuario identificarse con su cuenta de la ULL.
- Uso API de Google.
- Integrada en Firebase.



Ventana de Inicio de Sesión

```
1 public class LoginActivityULL extends AppCompatActivity implements ... {
2     private GoogleApiClient googleApiClient;
3     // Metodo que se ejecuta cuando se lanza la ventana
4     protected void onCreate(Bundle savedInstanceState) {
5         ...
6         // Opciones de la autentificacion que se desea realizar
7         GoogleSignInOptions gso = new GoogleSignInOptions.Builder(
8             GoogleSignInOptions.DEFAULT_SIGN_IN).requestEmail().build();
9         // Se crea una instancia de la API de google con las opciones
10        googleApiClient = new GoogleApiClient.Builder(this).enableAutoManage(
11            this, this).addApi(Auth.GOOGLE_SIGN_IN_API, gso).build();
12    }
13    // Manejo de eventos
14    public void onClick(View v) {
15        if (v.getId() == loginButton.getId()) { // Si se presiona loginButton
16            // Se crea y lanza el cuadro de dialogo de Google que permite
17            // autentificarse
18            Intent intent = Auth.GoogleSignInApi.getSignInIntent(
19                googleApiClient);
20            startActivityForResult(intent, 777);
21        }
22    }
23}
```

Ventana de *Inicio de Sesión*

```
1 private void handleSignInResult(GoogleSignInResult result) {  
2     if(result.isSuccess() == true) { // Exito en el inicio de sesion  
3         // Correo de la cuenta  
4         String userEmail = result.getSignInAccount().getEmail();  
5         // Se comprueba si es un correo de la ULL  
6         if(userEmail.matches("(.*)@ull.edu.es")) {  
7             Intent intent = new Intent(this, MainActivity.class);  
8             // Se ejecuta la ventana de ‘‘Inicio’’ de la aplicacion  
9             startActivity(intent);  
10        }else{  
11            logoutNotULLAccount(); // No es un correo universitario  
12        }  
13    }else{ ... } // Fallo al conectar con Google  
14 }
```

Modo de Realidad Aumentada

Ventana de Navegación en modo RA

- RA basada en geolocalización.
- Acceso a los sensores del dispositivo.
- Identificación de las instalaciones de la ULL.
- Obtención de las instalaciones.
- La clase ARNavigation.java es el activity principal encargado.



Acceso a la ubicación del dispositivo

AndroidManifest.xml

```
1 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
2 <uses-feature android:name="android.permission.LOCATION_HARDWARE" />
3 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

ARNavigation.java

```
1 // Si el usuario no ha concedido los permisos para utilizar el GPS en la
  // aplicación
2 if (ContextCompat.checkSelfPermission(this, Manifest.permission.
  ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
  // Se solicita el permiso para acceder a los datos de GPS de dispositivo
  requestPermissions(new String[]{Manifest.permission.ACCESS_FINE_LOCATION
    }, MY_PERMISSIONS_REQUEST_LOCATION);
5 }
```

Acceso a la orientación del dispositivo

```
1 locationManager = (LocationManager) getContext().getSystemService(Context.  
    LOCATION_SERVICE);  
2 locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 3000,  
    5, this);  
3 locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,  
    3000, 5, this);  
  
1 currentLocation = locationManager.getLastKnownLocation(LocationManager.  
    GPS_PROVIDER);
```

Acceso a la orientación del dispositivo

Orientación de un dispositivo Android

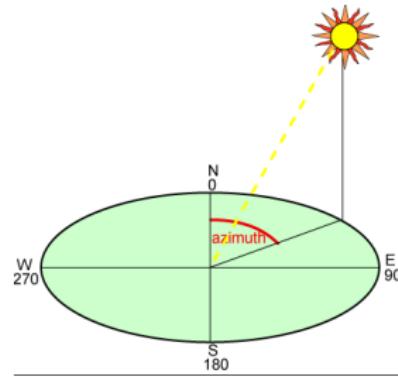
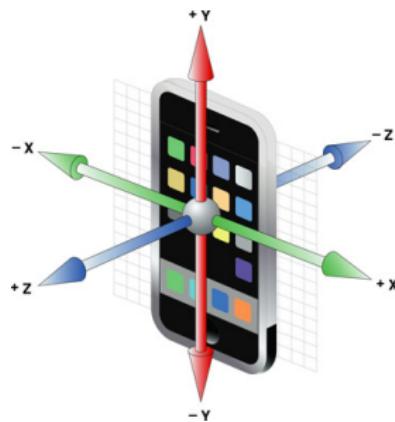
- Matriz de rotación que Android calcula a partir de magnetómetro y el acelerómetro.
- Se obtiene el valor de brújula magnética del dispositivo.
- Permite orientar al dispositivo en el espacio para poder identificar las instalaciones.

```
1 // Se accede a los sensores del dispositivo
2 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
3 // Se accede al calculo de la matriz de rotacion que proporciona el valor de
   la brujula magnetica del dispositivo
4 Sensor compass = mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
5 // Se escuchan a los cambios del sensor para actualizar los calculos
6 mSensorManager.registerListener(this, compass, SensorManager.
   SENSOR_DELAY_NORMAL);
```

Acceso a la orientación del dispositivo

Orientación de un dispositivo Android

- Android dispone de 3 ejes: x, y, z.
- Valor de acimut.
- Obtención de la orientación del dispositivo con respecto al norte magnético.



Acceso a la orientación del dispositivo

```
1 // Se escuchan los cambios en el sensor y se hacen los calculos
2 public void onSensorChanged(SensorEvent event) {
3     // Valor del sensor en grados
4     double radians = event.values[0];
5     // Se convierte a radianes
6     radians = Math.toRadians(radians);
7     // Se obtiene la ultima posicion registrada del GPS
8     LatLng lastPosition = getCurrentPos();
9     if (auxpos != null) { // Si la posicion no es nula
10         // Se le pregunta al objeto de la clase 'Navigation' las
11             instalaciones
12         // que se encuentran en esa direccion
13         allResultsSites = navULL.whatCanSee(lastPosition, radians);
14     }
15     // Si se obtiene al menos un resultado
16     if (allResultsSites != null) {
17         // Se obtiene la instalacion mas cercana, el indice 0 corresponde a
18             la mas cercana
19         nearSiteResult = allResultsSites.get(0);
... // Se muestra su informacion por pantalla para que usuario
    conozca la instalacion
... // Elementos a mostrar si se encuentra mas de una instalacion
```

Modelos encargados de la navegación

Clases que intervienen:

- `ULLSite.java`
- `Navigation.java`
- `Vector2D.java`

ULLSite.java

- Almacena la información referente a una instalación de la ULL.
- Se crea a partir del objeto JSON contenido en la base de datos.
- Contiene los atributos que permiten identificar una instalación.

Objeto JSON con la información de una instalación

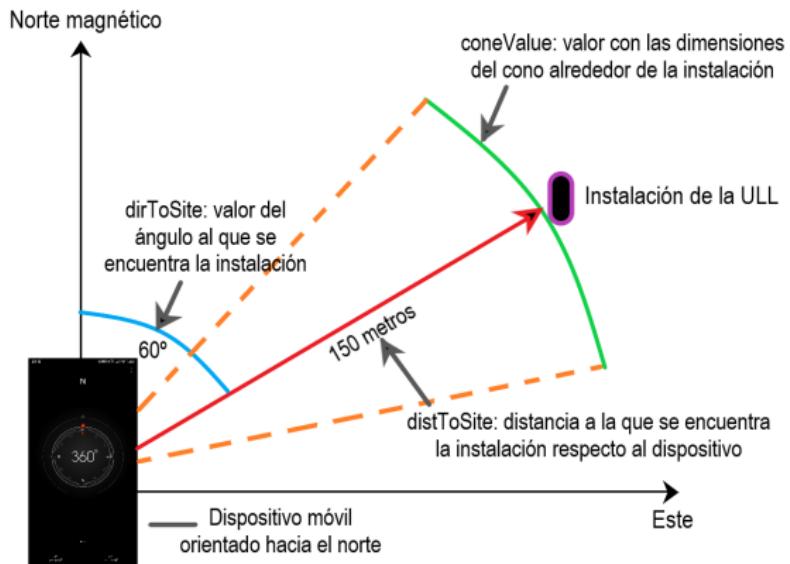
```
1  {
2      "_id": {
3          "$oid": "5b5a004efb6fc07c4c24a5aa"
4      },
5      "id": "esit",
6      "name": "Escuela Superior de Ingenieria y Tecnologia",
7      "position": {
8          "lat": "28.482965",
9          "long": "-16.322003"
10     },
11     "desc": "La Escuela Superior de Ingenieria y Tecnologia ...",
12     "imageLink": "https://www.ull.es/donde/assets/img/facultades/esit.jpg",
13     "canFind": [
14         {
15             "id": "ESIT",
16             "link": "https://www.ull.es/centros/escuela-superior-de-
17                 ingenieria-y-tecnologia/"
18         }
19     ]
}
```

ULLSite.java

```
1 public class ULLSite {  
2     private String id; // ID de la instalacion  
3     private String name; // Nombre  
4     private LatLng mapPoint; // Localizacion geografica  
5     private Vector2D point; // Vector2D de la ubicacion  
6     private String desc; // descripcion de la instalacion  
7     private String imageLink; // imagen de la instalacion  
8     // Enlaces de interes a las instituciones, grados, etc.  
9     private ArrayList<String> interestPoints; // Nombres  
10    private ArrayList<String> interestPointsLink; // Enlaces  
11    // Variables necesarias para identificar la direccion de la instalacion  
12    private double distToSite = -1; // Distancia a la instalacion  
13    private double dirToSite = -1; // Direccion en la que se encuentra  
14    private double coneValue = 0; // Valor del cono  
15    // Constructor  
16    public ULLSite(JSONObject object) {  
17        ... // Se construye el objeto con los atributos que se encuentran en  
18        // el objeto JSON de la instalacion.  
19    }  
20    ... // Metodos set() y get() de las variables  
}
```

Principales atributos que intervienen en la navegación

- “disToSite”
- “dirToSite”
- “coneValue”



Modelos encargados de la navegación

Navigation.java

- Clase encargada de identificar las instalaciones.
- Contiene una lista con todas las instalaciones.
- Recibe los datos de los sensores.
- Calcula la distancia, dirección y valor de cono para cada instalación.

Atributos de la clase Navigation.java

```
1 public class Navigation implements Serializable {
2     // Distancia en la que una instalacion se considera "cercana" en metros
3     private static final double NEAR_VALUE = 150;
4     // Cuando la distancia de una instalacion sea cercana se utilizaran los
5     // valores *_NEAR para los calculos en caso contrario se utilizaran *
6         _FAR
7     // Maximo valor del cono
8     private static final double MAX_CONE_GRADS_NEAR = Math.PI / 2;
9     private static final double MAX_CONE_GRADS_FAR = Math.PI / 8;
10    // Minimo valor del cono
11    private static final double MIN_CONE_GRADS = Math.PI / 20;
12    // Las variables SCALE_CONE contienen los valores que ajustan las
13        dimensiones del cono en funcion su distancia
14    // Variable que escala el cono
15    private static final double SCALE_CONE_NEAR = 0.0078;
16    private static final double SCALE_CONE_FAR = 0.00078;
17    // Distancia maxima y minima por defecto para considerar una instalacion
18    private double maxDist = 200;
19    private double minDist = 0;
20    private Vector2D currentPos; // Posicion actual del dispositivo
21    private double currentDir; // Direccion actual del dispositivo
        // Todas las instalaciones
        private ArrayList<ULLSite> allSites = new ArrayList<>();
```

Métodos de la clase Navigation.java

```
1 // Constructor del atributo allSites con el objeto JSON con las
   instalaciones
2 public Navigation(JSONArray jsonULLSitesAux) { ... }
3 // Calculos que permiten identificar las instalaciones
4 public ArrayList<ULLSite> whatCanSee(LatLng currentPosAux, double actualDir)
   { ... }
5 // Calcula la distancia entre dos ubicaciones geograficas
6 public double getDistanceBetween(Vector2D v1, Vector2D v2) { ... }
7 // Se comprueba si la direccion del dispositivo se encuentra dentro del cono
   de identificacion de la instalacion
8 private boolean isInCone(double directionToSite, double coneValue) { ... }
9 // Sirve para reorientar al norte magnetico, como inicio de rotacion, el
   angulo dado por el Vector2D.getAngleRad(Vector2D v)
10 private double recalculeAngVector2D(double angleRad) { ... }
11 // Se invierte el angulo
12 public double invertAng(double rad) { ... }
13 // Se rota -90 el angulo
14 public double rotateRad(double rad) { ... }
15 // Se calcula el valor del cono
16 public double calculateCone(double dist) { ... }
17 ... // Metodos Get() y Set() de los atributos
```

Método calculateCone(double dist)

```
1 public double calculateCone(double dist) {  
2     // Si es una instalacion "cercana" del dispositivo  
3     if (dist <= NEAR_VALUE) {  
4         // Se calcula el valor del coneValue restandole al valor maximo las  
5             distancia a la instalacion por la constante SCALE_CONE_NEAR que  
6             permite que esta se escale gradualmente.  
7         return MAX_CONE_GRADS_NEAR - dist * SCALE_CONE_NEAR;  
8     }else { // Si es "lejana"  
9         // Se calcula el valor del coneValue restandole al valor maximo las  
10            distancia a la instalacion por la constante SCALE_CONE_FAR que  
11            permite que esta se escale gradualmente en instalaciones lejanas  
12            .  
13            double auxCone = MAX_CONE_GRADS_FAR - dist * SCALE_CONE_FAR;  
14            if (auxCone < MIN_CONE_GRADS) {  
15                return MIN_CONE_GRADS;  
16            } else {  
17                return auxCone;  
18            }  
19        }  
20    }  
21}
```

Navigation.java

```
1 // Metodo que calcula el angulo que se obtiene al transformar el rectangulo
2 // formado por este punto y el punto v2 a coordenadas polares
3 public double getAngleRad(Vector2D v2) {
4     double dx = v2.getX() - getX(); // Se calculan las distancias en el eje
5     double dy = v2.getY() - getY(); // se obtiene el rectangulo formado por
6     // estos dos puntos
7     double radian = Math.atan2(dy, dx); // Se realiza la arcotangente para
8     // calcular el angulo del rectangulo formado
9     return radian; // Se devuelve el resultado
10 }
```

```
1 private double recalculaAng(double angleRad) {
2     double aux = rotateRad(angleRad); // Rota -pi/2
3     aux = invertAng(aux);           // Se invierte el angulo
4     return aux;                   // Se devuelve el resultado
5 }
```

whatCanSee(LatLng currentPosAux, double actualDir)

```
1 // Metodo que se encarga identificar las instalaciones en frente al dispositivo
2 // Recibe la posicion y orientacion actual del dispositivo
3 // Devuelve una lista de instalaciones indicando cual es la mas cercana
4 public ArrayList<ULLSite> whatCanSee(LatLng currentPosAux, double actualDir)
5 {
6     // Posicion actual del dispositivo
7     currentPos.set(actualPos.longitude, actualPos.latitude);
8     currentDir = actualDir; // Orientacion actual del dispositivo
9     int id = -1;
10    ArrayList<ULLSite> result = new ArrayList<>(); //Instalaciones encontradas
11    // Se calculan todas las instalaciones que se encuentran entre maxDist y minDist como posible instalaciones a identificar
12    for (int i = 0; i < allSites.size(); i++) {
13        double distToSite = getDistanceBetween(currentPos, allSites.get(i).getPoint());
14        if ((distToSite < maxDist) && (distToSite > minDist)) {
15            destSites.add(allSites.get(i));
16        }
17    }
18 }
```

whatCanSee(LatLng currentPosAux, double actualDir)

```
1  for (int i = 0; i < destSites.size(); i++) {
2      // Se calcula la direccion, distancia y valor del cono de cada
3      // instalacion a partir de la actual ubicacion del dispositivo
4      double dirToSite = recalculeAng(currentPos.getAngleRad(...));
5      double distToSite = getDistanceBetween(...);
6      double coneValue = calculateCone(distToSite);
7      // Se comprueba si el dispositivo esta orientado hacia dentro del
8      // cono que se forma en la direccion de la instalacion
9      if (isInCone(dirToSite, coneValue)) { ...
10         // Se guarda esta instalacion como resultado
11         result.add(destSites.get(i));
12         // Si es la instalacion mas cercana
13         if (nearSiteDist > distToSite) {
14             nearSiteDist = distToSite; // Actualizamos la distancia
15             id = i; // Se guarda el indice de la instalacion mas cercana
16         }
17     }
18     if (id != -1) { // Si se ha encontrado alguna instalacion
19         // La instalacion mas cercana la se guarda en la primera posicion
20         result.add(0, destSites.get(id));
21         return result; // Se devuelven las instalaciones
22     }
23 }
```

Obtención de la información

- La información de las instalaciones la provee el servidor que a su vez se comunica con la base de datos.
- La clase `GetData.java` se encarga de conectar con el servidor y manejar la respuesta.
- Los radios máximos y mínimos con los que identificar las instalaciones están guardados en las Shared Preferences.

GetData.java

```
1 public class GetData extends AsyncTask<String, Void, String> {
2     protected String doInBackground(String... strings) {
3         try {
4             URL url = new URL(strings[0]); // url a la que se realiza la
5                                         // peticion
6             HttpURLConnection urlConnection = url.openConnection();
7             urlConnection.setConnectTimeout(10000); // Tiempo de espera
8                                         // maximo
9             urlConnection.setRequestMethod("GET"); // Metodo de la conexion
10            urlConnection.setRequestProperty("Content-Type", "application/
11                                         // json"); // Tipo de contenido esperado como respuesta
12            urlConnection.connect();           // Se realiza la conexion
13            ... // Cuando se recibe la respuesta se lee, se crea y se devuelve
14                                         // un string con su contenido
15        }catch (IOException e) { ... }
16    }
17 };
18 }
```

Métodos encargados de la obtención de la información

```
1 private void getSitesFromDB() {  
2     GetData getSites = new GetData();  
3     String sites = getSites.execute("https://server-ull-ar.herokuapp.com/api  
        /ull-sites").get(); //Conectamos con el servidor  
4     JSONArray array = new JSONArray(sites);  
5     // Se crea una instancia de la clase "Navigation" con todas las  
        instalaciones obtenidas de la base de datos  
6     navULL = new Navigation(array);  
7 }
```

```
1 private void getRadius() {  
2     //Obtenemos las preferencias que contiene los radios de las  
        circunferencias  
3     settingsPref = PreferenceManager.getDefaultSharedPreferences(getApplicationContext()  
());  
4     String auxMaxRadius = settingsPref.getString("maxRadius", "null");  
5     String auxMinRadius = settingsPref.getString("minRadius", "null");  
6     // Se guarda el valor "maxRadius" y "minRadius" en el objeto navULL  
7     navULL.setMaxDist(Integer.parseInt(auxMaxRadius));  
8     navULL.setMinDist(Integer.parseInt(auxMinRadius));  
9 }
```

Visualización

- El activity “ARNavigation” es el encargado de la visualización de la técnica de RA.
- Este activity hereda de la clase “ARActivity” del SDK de Kudan.

arnavigation.xml

```
1 <android.support.constraint.ConstraintLayout ...>
2     ... <!-- Imagenes de fondo de los TextView para facilitar la lectura de
3         los mensajes-->
4     <!-- Texto informativo de que se ha encontrado una instalacion -->
5     <TextView android:id="@+id/seenText" android:text="Viendo actualmente"
6         .../>
7     <!-- Nombre de la instalacion encontrada -->
8     <TextView android:id="@+id/ullSiteText" android:text="Facultad de
9         Ciencias ..." .../>
10    <!-- Texto por defecto cuando no se apunta a ninguna instalacion -->
11    <TextView android:id="@+id/..." android:text="Apunte a una instalacion
12        ..." .../>
13    <!-- Boton que accede a la ventana con la informacion de la instalacion
14        -->
15    <Button android:id="@+id/moreInfoButton" .../>
16    <!-- Texto del boton "moreInfoButton" -->
17    <TextView android:id="@+id/moreInfoText" android:text="Mas informacion"
18        .../>
19    <!-- Boton que despliega la lista con el resto de instalaciones
20        encontradas -->
21    <Button android:id="@+id/..." android:text="Encontradas X instalaciones
22        ..." />
23 </android.support.constraint.ConstraintLayout>
```

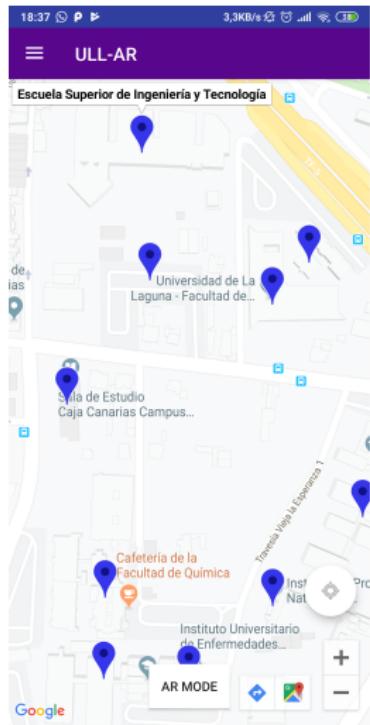
Fragments en Android

- ¿Qué es un fragmento?
- ¿Qué beneficios ofrecen para la aplicación?
- Fragmentos de ULL-AR:
 - MapsFragment.
 - HomeFragment.
 - AboutFragment.

Fragmentos de ULL-AR

MapsFragment

- Nos proporciona un mapa generado por la API de Google Maps.
- En este mapa permite:
 - Dibujar las instalaciones de la ULL.
 - Ubicar al dispositivo.
 - Dibujar dos circunferencias que actúan como rango de aparición de las instalaciones.



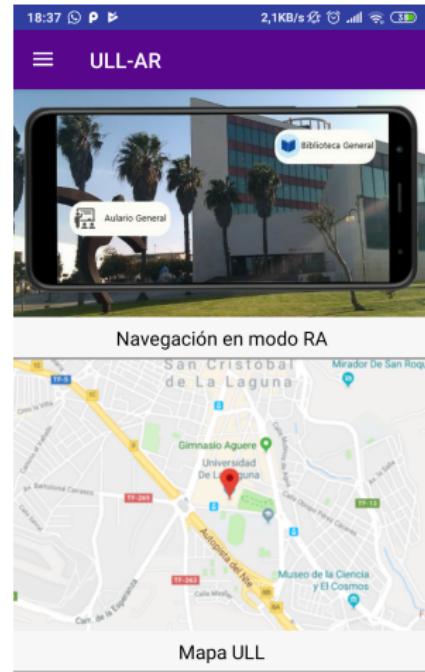
MapsFragment.java

```
1 public class MapsFragment extends Fragment implements ... {  
2     // Todas las instalaciones y su marcadores  
3     private ArrayList<ULLSite> allSites= new ArrayList<ULLSite>();  
4     private ArrayList<Marker> allMarkers = new ArrayList<Marker>();  
5     private Circle circleMax;      // Circulo mayor dibujado en el mapa  
6     private Circle circleMin;      // Circulo menor  
7     private int maxRadius;        // Radio del circulo mayor  
8     private int minRadius;        // Radio del circulo menor  
9     private boolean showRadius;   // Se indica si se dibujan los circulos o no  
10    // Metodo que se ejecuta cuando se lanza el fragment  
11    public View onCreateView( ... ) {... }  
12    // Metodo que se ejecuta cuando la vista ya esta creada  
13    public void onViewCreated(View view, ...) { ... }  
14    // Metodo que dibuja cuando el mapa de Google Maps ya este creado  
15    public void onMapReady(GoogleMap googleMap) { ... }  
16    // Metodo que actualiza los elementos de los mapas a la ubicacion actual  
17    public void onLocationChanged(Location location) { ... }  
18    // Metodo que muestra las ubicaciones entre "circleMax" y "circleMin"  
19    private void showSitesOnRadius(LatLng position) { ... }  
20    private void drawAllSites() { ... } // Se dibujar las instalaciones en  
21        el mapa  
22    .... // Resto de metodos y atributos  
}
```

Fragmentos de ULL-AR

HomeFragment

- Primera vista que aparece cuando el usuario se autentifica en la aplicación.
- Uso del modelo de “RecyclerView” y adaptadores de Android.
- La clase “ItemHome” contiene los atributos a mostrar.



ItemHomeAdapter.java

```
1 public class ItemHomeAdapter extends RecyclerView.Adapter<...> {  
2     private List<ItemHome> items; //Lista de items a mostrar  
3     private int layout;           //Vista con los items  
4     private OnItemClickListener itemClickListener; //Manejo de eventos  
5     //Constructor con los atributos  
6     public ItemHomeAdapter(List<ItemHome> items, int layout ...){ ... }  
7     // Metodo que asigna "layout" a la vista del parametro "parent"  
8     public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent...){ ... }  
9     // Se enlaza cada item con su vista  
10    public void onBindViewHolder(@NonNull Viewholder holder, int pos){ ... }  
11    // Clase que muestra y conecta cada item con su vista  
12    public static class Viewholder extends RecyclerView.ViewHolder {  
13        public TextView itemName; //Atributos con los elementos  
14        ...                      //de la vista  
15        public Viewholder(View itemView) { // Constructor con el layout  
16            this.itemName = itemView.findViewById(R.id.textView_home_item);  
17            ... // Se enlaza los atributos de los con su vista  
18        }  
19        // Se asigna a cada vista el contenido de su item correspondiente  
20        public void bind(final ItemHome itemHome, final ...) {  
21            this.itemName.setText(itemHome.getName()); // Se asigna el texto  
22            ... //Se asignan el resto de elementos de la vista  
23    }
```

HomeFragment.java

```
1 public class HomeFragment extends Fragment implements View.OnClickListener {  
2     private List<ItemHome> itemsHome; // Lista de items a mostrar  
3     private RecyclerView recyclerView; // Vista de la lista de items  
4     private RecyclerView.Adapter homeAdapter; // Adaptador  
5     private RecyclerView.LayoutManager homeLayoutManager; //Layout  
6     public View onCreateView( ... ) { ... } //Constructor  
7     // Creamos la lista de items con su nombre, imagen y si es un enlace web  
8     private void setAllItems() {  
9         ... // Resto de items  
10        itemsHome.add(new ItemHome("Pagina de la ULL", "home_ull_site", true,  
11                           "www.ull.es")); // Item con enlace externo  
12    }  
13    public void onViewCreated(View view, @Nullable Bundle ... ) {  
14        // RecyclerView  
15        recyclerView =getActivity().findViewById(R.id.recyclerView_Home);  
16        // Se instancia el objeto ItemHomeAdapter con las lista de items  
17        homeAdapter = new ItemHomeAdapter(itemsHome, R.layout.  
18                                         adapter_home_item new ItemHomeAdapter.OnItemClickListener() {  
19                                             public void onItemClick(ItemHome item, int position) { ... }});  
20        //Se asigna el tipo layout y el adaptador a la vista RecyclerView  
21        homeLayoutManager = new LinearLayoutManager(getContext());  
22        recyclerView.setLayoutManager(homeLayoutManager);  
23        recyclerView.setAdapter(homeAdapter);  
24    }
```

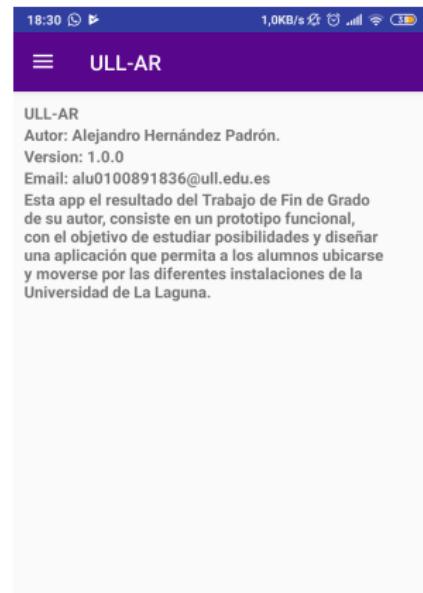
fragment_home.xml

```
1 <LinearLayout ...>
2     <!-- RecyclerView, contenedor en el que iran la vista de cada item de la
         clase ItemHome -->
3     <android.support.v7.widget.RecyclerView android:id="@+id/
         recyclerView_Home" ... />
4 </LinearLayout>
```

Fragmentos de ULL-AR

AboutFragment

- Información general de la aplicación:
 - Nombre.
 - Autor.
 - Version.
 - Email.
 - Descripción de la aplicación.



Instalaciones de la ULL

Ventana de *Todas las instalaciones*

- Permite al usuario acceder a las instalaciones de la BD.
- Realizar una búsqueda.
- Uso de adaptadores para mostrar información básica de cada instalación.
- Permite acceder a la ficha de información de cada instalación.

The screenshot shows a mobile application interface with a purple header bar containing the text "ULL-AR". Below the header are three cards, each representing a different university installation:

- Ingeniería Agraria**: A card featuring a photo of a modern white building complex. The text describes it as a teaching facility located in the Agrarian Engineering section of the Polytechnic Superior of Engineering.
- Edificio Calabaza Decano**: A card featuring a photo of a red brick building with large windows. The text describes it as the Doctorado y Estudios de Posgrado, which is responsible for organizing academic teachings and processes, leading to various postgraduate degrees.
- Módulo B Educación**: A card featuring a photo of a white multi-story building. The text describes it as the Faculty of Education of the University of La Laguna, created by Decree in 1995, and responsible for administrative management, planning, organization, and control of university education, leading to various academic titles.

SiteAdapter.java

```
1 public class SiteAdapter extends BaseAdapter implements Filterable { ...
2     // Instalaciones a mostrar con el filtro de busqueda aplicado
3     private ArrayList<ULLSiteSerializable> filteredSites;
4     public SiteAdapter( ... ) { .. } // Constructor
5     // Se crea la vista de cada item con su imagen, nombre y descripcion
6     public View getView(int position ...) { ... }
7     // Filtro de busqueda a aplicar sobre la lista a mostrar
8     public Filter getFilter() {
9         return new Filter() {
10            FilterResults performFiltering(CharSequence charSequence) {
11                String charString = charSequence.toString(); // Filtro
12                if (charString.isEmpty()) { ... } // No hay filtro
13                else { ... } // Se crea un lista las instalaciones que
14                    coinciden con los caracteres de "charString"
15                // Instalaciones filtradas
16                filterResults.values = filteredSites;
17                return filterResults; // Se devuelve el resultado del filtro
18            }
19            protected void publishResults(... FilterResults filterResults) {
20                filteredSites = (ArrayList<ULLSite...>) filterResults.values;
21                notifyDataSetChanged(); // Se indica al adaptador que el
22                                // contenido de la lista ha sido modificado
23            }
24        };
25    }
```

SitesListActivity.java

```
1 public class SitesListActivity extends AppCompatActivity { ...
2     private ListView listSites; // Vista con la lista de instalaciones
3     private SitesArray sitesToShow; // Lista de instalaciones
4     private SearchView searchView; // Barra superior de búsqueda
5     SiteAdapter siteAdapter; // Adaptador de las instalaciones
6     protected void onCreate(Bundle savedInstanceState) { ...
7         //La lista de instalaciones se obtiene del activity anterior
8         sitesToShow = (SitesArray) getIntent().getSerializableExtra("sitesToShow");
9     }
10    // Metodo que instancia el objeto la clase "SiteAdapter"
11    private void showDinamicSites() {
12        // ListView que muestra la lista de las instalaciones
13        listSites = findViewById(R.id.listSites);
14        //Instanciamos el Adaptador con la lista de instalaciones sitesToShow
15        siteAdapter = new SiteAdapter(this, R.layout.site_item, sitesToShow);
16        listSites.setAdapter(siteAdapter); //Adaptador del ListView
17        //Manejo de eventos, se ejecuta la ventana con la ficha de
18        //informacion
19        listSites.setOnItemClickListener(...){} //de la instalacion
    }
```

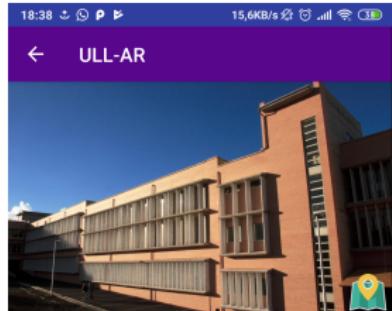
SitesListActivity.java

```
1  public boolean onCreateOptionsMenu(Menu menu) {
2      ...//Configuramos la barra de busqueda
3      //Eventos cuando se escriba se aplica el filtro en el adaptador
4      searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener
5          () {
6              @Override // Cuando se presione el boton de busqueda
7              public boolean onQueryTextSubmit(String query) {
8                  siteAdapter.getFilter().filter(query); // Filtro a aplicar
9                  return false; }
10             @Override // Cuando el texto cambia
11             public boolean onQueryTextChange(String newText) {...}
12         });
13         return super.onCreateOptionsMenu(menu); // Se devuelve la barra de
14             busqueda
15     }
```

Instalaciones de la ULL

Ventana de *Información de la instalación*

- Muestra información detallada de la instalación.
- Permite acceder a la ruta a de la instalación en Google Maps.
- Muestra los enlaces de interés relacionados.



Descripción:

La Escuela Superior de Ingeniería y Tecnología es el órgano encargado de la organización de las enseñanzas y los procesos académicos, administrativos y de gestión conducentes a la obtención de títulos de carácter oficial y validez en todo el territorio nacional que se imparten en ella y demás títulos que establezca la legislación vigente. Las titulaciones que se imparten en la Escuela corresponden al ámbito de la Ingeniería Informática y de la Ingeniería Industrial.

Instituciones y enlaces de interés:

ESIT

Grado en Ingeniería Informática

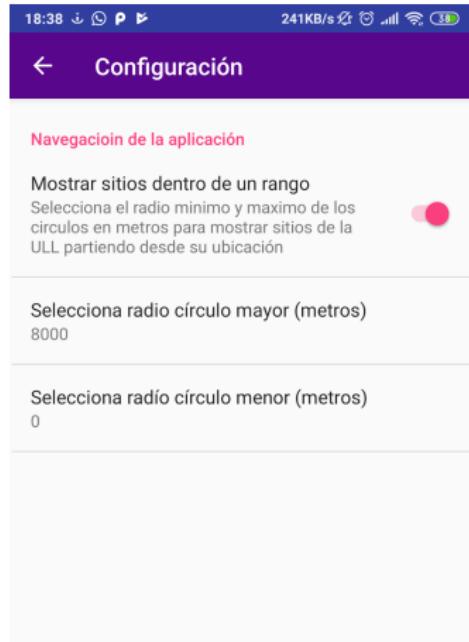
SiteDescriptionActivity.java

```
1 public class SiteDescriptionActivity extends ListActivity { ...
2     ULLSiteSerializable actualULLSite; // Instalacion a mostrar
3     // Lista de enlaces de la instalacion
4     ArrayList<String> listItems = new ArrayList<String>();
5     @Override // Metodo que inicia el activity
6     public void onCreate(Bundle savedInstanceState) { ...
7         // Vista principal
8         setContentView(R.layout.activity_site_description);
9         // El objeto con la instalacion proviene del activity anterior
10        actualULLSite = getIntent().getSerializableExtra("actualULLSite");
11        setUI();
12        setListSites();
13    }
14    // Metodo que carga los textos, imagenes y enlaces de la instalacion a
15    // mostrar en la vista
16    public void setUI() { ... }
17    // Este metodo crea un adaptador con los enlaces de la instalacion y
18    // carga en el navegador del dispositivo el enlace cuando es
    // seleccionado
    public void setListSites() { ... }
}
```

Preferencias del usuario

Ventana de *Información de la instalación*

- Ventana donde van los ajustes de la aplicación
- Se pueden configurar los radios de búsqueda de las instalaciones.



Índice

- 1 Introducción
- 2 Herramientas y Tecnologías utilizadas
- 3 RA y RA en entornos universitarios
- 4 La aplicación ULL-AR
- 5 Back-end de la aplicación
- 6 Presupuesto
- 7 Summary and conclusions

Back-end de la aplicación

Base de datos

- Base de datos de MongoDB.
- Se encuentra en la plataforma de mLab.
- Contiene todas la información referente a las instalaciones.

Servidor

- Implementado en Node.js.
- Se encuentra en funcionamiento en la plataforma de Heroku.
- Conecta con la base de datos.

server.js

```
1 //Librerias
2 var express = require('express'); // Servidor
3 var mongoose = require('mongoose'); // Para conectar con MongoDB
4 // Metodos para manejar el contenido de las respuestas a la peticiones
5 var bodyParser = require('body-parser');
6 //Configuracion del servidor
7 var server = express(); // Se instancia el servidor
8 server.use(bodyParser.urlencoded({extended: false})); //Formato Querystring
9 server.use(bodyParser.json()); //Manejo de peticiones JSON
10 // Se usa el fichero que se encuentra en ./routes/api.js
11 // Se realiza la conexion con la base de datos con la url correctamente
12 // guardada en PROD_MONGODB
13 mongoose.connect(process.env.PROD_MONGODB, function (error) { ... });
14 server.get('/', function (req, res) { ... }) // Ruta raiz del servidor
15 // Se le indica al servidor que el archivo /routes/api.js se encargue de las
16 // solicitudes que se reciben la url /api
17 api = require('./routes/api');
18 server.use('/api', api);
19 // Se ejecuta el servidor en el puerto de produccion o en el 3000
20 server.listen(process.env.PORT || 3000, function() { ... });
```

models/ullSites.js

```
1 // Se declara el variable restful para manejar las peticiones
2 var restful = require('node-restful');
3 // Se usa mongoose para conectarse a la BD
4 var mongoose = restful.mongoose;
5 // Estructura de las instalaciones de la ull contenidos en la base de datos
6 var ullSitesSchema = new mongoose.Schema({
7     id: String,
8     name: String,
9     position: {
10         lat: String,
11         long: String
12     },
13     desc: String,
14     imageLink: String,
15     canFind: [
16         {
17             id: String,
18             link: String
19         }
20     }
21 })
22 // Se devuelve el modelo para poder utilizarlo en otros ficheros
23 // Este modelo se conectara con colección de la base de datos "ull_sites"
24 module.exports = restful.model('ull_sites', ullSitesSchema);
```

routes/api.js

```
1 var express = require('express');
2 var router = new express.Router();
3 // Modelo que maneja la peticion a la base de datos
4 var ullSites = require('../models/ullSites');
5 // Se selecciona los metodos que puede responder
6 // Al ser una aplicacion sencilla solo se necesitan manejar peticiones get
7 ullSites.methods(['get']);
8 // Se indica al router la url que gestionara las peticiones
9 ullSites.register(router, '/ull-sites');
10 module.exports = router; //Exportamos este modulo
```

Back-end de la aplicación

Despliegue

- Archivo Procfile.
- Subir el repositorio a Heroku.
- URL : *https://server-ull-ar.herokuapp.com/api/ull-sites*.

Procfile

```
1 web: node server.js
```

Índice

- 1 Introducción
- 2 Herramientas y Tecnologías utilizadas
- 3 RA y RA en entornos universitarios
- 4 La aplicación ULL-AR
- 5 Back-end de la aplicación
- 6 Presupuesto
- 7 Summary and conclusions

Coste almacenamiento y despliegue

- Publicar la aplicación en el la Google Play Store: 25 dolares.
- Servido en Heroku: 25 euros/mes, 300 euros al año.
- Base de datos de mLab: Sin coste.

Desarrollo de ULL-AR

- Obtener información de las instalaciones: 300-400 euros.
- Programador con conocimientos en Android.
- Salario: coste 10 euros/hora de trabajo por 320 horas. hacen un total de 3200 euros.

Mantenimiento de la aplicación

- Programador en Android con conocimientos en Node.js.
- Encargado de la actualización de la información de la base de datos y del mantenimiento y posibles mejoras de la aplicación.
- Tarifa: 20 euros/hora. El coste total se calcula mediante la contratación de una “bolsa de horas” de trabajo del programador.

Índice

- 1 Introducción
- 2 Herramientas y Tecnologías utilizadas
- 3 RA y RA en entornos universitarios
- 4 La aplicación ULL-AR
- 5 Back-end de la aplicación
- 6 Presupuesto
- 7 Summary and conclusions

Summary and conclusions

Conclusions

Nowadays, beacon technology is still on a development stage. By itself can be quite limited due to the way it works. The physical and positional limitations require a deep analysis in order to place devices the best way.

PENDIENTE

ULL-AR

- Repositorio en *GitHub*:

<https://github.com/alehdezp/TFG-ULL-AR>

Alejandro Hernández Padrón

alu0100891836@ull.edu.es