Grado en Ingeniería Informática

Trabajo Final de Grado

Aplicación para marketing con balizas

Supervisor:

Autor:

Rafael Amorós

Miguel Renau Ventura

Tutor académico:

Gabriel Recatalá Ballester

Fecha de lectura: 28 de Octubre de 2016
Curso Académico 2015/2016

Resumen

El proyecto que se describe en este documento consiste en la recepción de las promociones vigentes de una tienda cuando el cliente se encuentre cerca de ésta.

Para ello es necesario la implementación de un sistema de comunicación entre un dispositivo móvil (smartphone) y un servidor que se encarga de gestionar una base de datos. El dispositivo móvil se encarga de detectar balizas de los tipos iBeacons y EddyStones, y mostrar la información asociada a la baliza previa consulta al servidor.

Palabras clave

Smartphone, Aplicación Móvil, Baliza, iBeacon, EddyStone, Servidor, Base de datos, Bluetooth, Wifi

Keywords

Smartphone, App, Beacon, iBeacon, EddyStone, Server, Database, Bluetooth, Wi-Fi

Índice

Índice de figuras

1. Introducción

1.1 Contexto y motivación del proyecto

El proyecto cuya propuesta se presenta en este documento se ha desarrollado en Mestral Telecomunicaciones SL. Se trata de una empresa especializada en redes y telecomunicaciones con una experiencia del personal de la empresa, en algunos casos, de 25 años. Si bien en los inicios dedicó su labor profesional a la informática, fue en 1996 cuando apostó por las redes y telecomunicaciones. Siempre ha buscado la innovación como herramienta para ofrecer a sus clientes una tecnología que permitiera un ahorro de costes y una mayor productividad.

La motivación del proyecto es que cada una empresas adscritas al servicio que se va a dar pueda ofrecer contenidos y promociones personalizadas a sus clientes de forma rápida y sencilla.

A continuación y durante todo el proyecto se utilizarán los términos zona, baliza, aplicación móvil, base de datos y servidor. Cada zona corresponde a una tienda. Las balizas son los dispositivos utilizados para identificar cada una de las zonas desde el dispositivo móvil. La aplicación móvil es la que mediante el dispositivo móvil (smartphone) detecta cada una de las balizas. Una vez detectada una baliza, se consultará en la base de datos la información de la zona asociada. Para ello se utiliza como intermediario un servidor capaz de gestionar las peticiones y filtrar las promociones según el perfil de usuario de la aplicación.

El proyecto propuesto por la empresa consiste en el desarrollo de un sistema basado en una plataforma web para la gestión de beacons, y una aplicación para que el usuario de ésta reciba contenidos. El proyecto ha sido dividido en dos partes, la primera parte consta de la plataforma web y la gestión de una base de datos, y la segunda parte es la relacionada con la aplicación, la comunicación con la balizas y la base de datos.

Este proyecto trata de la segunda parte del sistema mencionado anteriormente, la relacionada con la aplicación y las balizas o beacons. La aplicación debe desarrollarse tanto para dispositivos Android como iOS.

La parte del proyecto relacionada con la plataforma web y la comunicación de esta con la BBDD ha sido realizada por Iván Chinchilla Mijas en su TFG.

Los beacons son dispositivos de bajo consumo que emiten una señal broadcast. Utilizan conexión bluetooth de bajo consumo para transmitir datos a dispositivos móviles sin necesidad de una sincronización de los aparatos.

Cada vez que un beacon se sincronice con un aparato móvil, éste recibirá la información que esté asociada al beacon. Para acceder a la información de cada beacon, así como a la del cliente, se consultará una base de datos. La base de datos se gestionará desde

la plataforma web, realizada por Iván Chinchilla. Una aplicación móvil (App) se encargará de, a partir de la señal del beacon, buscar la información que tiene asociada y mostrarsela al cliente.

Como función adicional, la App utilizará un sistema Indoor Positioning System [IPS] a partir de la detección de múltiples balizas para la localización de las tiendas y cálculo del camino más adecuado para el cliente, así como ajustar el contenido recibido al lugar en el que esté situado.

2. Descripción del proyecto

A continuación se describe tanto el funcionamiento del proyecto como los objetivos de éste. Cuando el usuario de la App esté situado cerca de alguna de las zonas, cada una de las balizas, ésta será detectada por la aplicación. La aplicación solicitará toda la información relacionada con la zona detectada, promociones y contenido asociado, a un servidor.

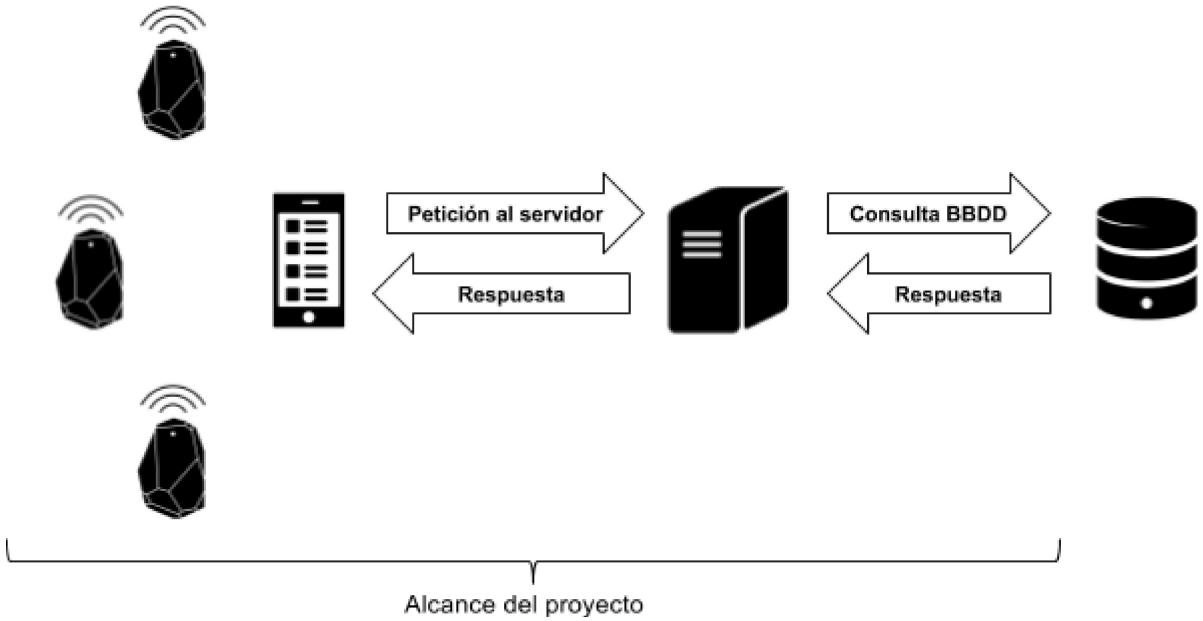


Figura 2-1 - Alcance del proyecto

El servidor es el que se encarga de hacer de intermediario entre la aplicación y la base de datos, el encargado de filtrar el contenido que le será mostrado al usuario de la App, así como controlar el acceso de este a la BBDD.

Por otra parte, y como está descrito en el proyecto de Iván, el usuario de la plataforma web es el que gestionará e introducirá todos los datos relacionados con las zonas y sus promociones, para posteriormente, mostrarlos al usuario de la App previo consulta.

2.1 Objetivos del proyecto

Se va a desarrollar un sistema de marketing basado en iBeacon y EddyStone que permitirá al usuario recibir toda clase de contenidos cuando se sitúe cerca de alguna de las balizas. Este sistema estará compuesto por:

- Una App tanto para dispositivos tanto Android como iOS que se encargará de detectar las balizas y mostrar el contenido asignado a éstas, mediante tecnología push [PUSH1] [PUSH2], previa consulta en la base de datos de la plataforma. Esta App también permitirá la geolocalización del usuario, además de mostrar las indicaciones pertinentes para llegar al lugar deseado desde la posición actual.

- La base de datos que recogerá todos los datos referentes a cada una de las balizas, así como los contenidos asociados y campañas.

- La plataforma web que permitirá la gestión de las balizas, creación de contenidos y campañas de marketing mediante la interacción con la base de datos.

La parte que se va a implementar en este proyecto es toda la relacionada con la App, desde el desarrollo de la App, la comunicación del dispositivo móvil con los iBeacons y EddyStone, y el intercambio de información con la base de datos compartida con la plataforma web.

El principal objetivo de este proyecto es que el usuario reciba las promociones asociadas a las zonas a las que se ha suscrito cuando se encuentre cerca de éstas.

El desarrollo e implantación de este sistema está dividido en tres partes:

- Desarrollo de la App

- Creación de la BBDD

- Comunicación de la App y la BBDD mediante un servidor

En el punto 4.3 Tecnologías utilizadas, se describen las cinco funcionalidades del proyecto que se está tratando en este documento. En primer lugar se explica la funcionalidad de los iBeacons y EddyStone, posteriormente el resto de éstas.

3. Análisis de requisitos

En este capítulo se analizan los requisitos de la aplicación y se delimita el alcance de la parte del proyecto descrita en esta memoria.

Se requiere una una aplicación móvil tanto para dispositivos Android como iOS. La aplicación se encargará, mediante un protocolo de comunicación Bluetooth, de detectar las balizas cercanas. Cada baliza tendrá un ID único asociado, para diferenciarla del resto. Cuando se detecte una de ellas, se consultará el contenido disponible en el servidor, para mostrarlo al usuario de la aplicación.

Las balizas a utilizar tienen dos modos de funcionamiento, un mod que simula las balizas de Apple, iBeacon, y otro que simula las de Google, EddyStone. Por motivos de compatibilidad, se requiere de una versión Android 4.3 o superior en el dispositivo móvil o smartphone. Más detalles en la sección 4.3 Tecnologías utilizadas.

3.1 Diagrama de casos de uso

En el análisis de los casos de uso se definen dos actores y dos casos de uso relacionados con la aplicación.

3.1.1 Actores

Los actores que interactúan con el sistema son los siguientes:

- Usuario cliente de la App: usuario que utiliza la aplicación para informarse sobre las promociones de las zonas seleccionadas.

- Usuario de la plataforma web: el encargado de introducir y modificar en la BBDD, vía web, las zonas y sus promociones. Más detallado por Iván Chinchilla Mijas en su TFG.

3.1.2 Casos de uso

A continuación se explican cada uno de los casos de uso.

Casos de uso de la App:

- Consulta de la promociones de una zona: el usuario cliente de la aplicación accede al contenido asociado a las zonas.
  Precondiciones: el usuario debe descargar la App y estar cerca de alguna de las zonas.

- Registro de los datos del usuario: el usuario cliente de la aplicación introduce sus datos personales para adaptar el contenido mostrado a sus características.

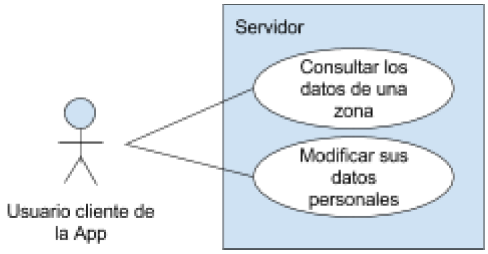


Figura 3.1.2-1 - Casos de uso de la aplicación

Los casos de uso de la plataforma web están detallados en el TFG de Iván Chinchilla.

3.2 Diagrama de clases

3.2.1 Diagrama de clases de la App

En esta sección se describen mínimamente todas y cada una de las clases implementadas durante el desarrollo de la aplicación móvil para dispositivos Android, tanto las utilizadas en la aplicación final, como las usadas durante el desarrollo de ésta.

La siguiente clase se encarga de crear la pantalla principal, leer del dispositivo y cargar en memoria los datos de usuario y las preferencias de las zonas, sobre las que el usuario quiere recibir información.

contenidoListActivity extends FragmentActivity implements contenidoListFragment.Callbacks	
public static final String ARCHIVO_ESTADO_ZONAS	Almacena el nombre del archivo en el que se guardan, en el dispositivo local, los ajustes sobre zonas de las que se quiere recibir promociones.
public static final String ARCHIVO_DATOS_PERSONALES	Almacena el nombre del archivo en el que se guardan, en el dispositivo local, los datos personales del usuario.
public static MyNetworkManager nm	Almacena el objeto de la clase "**MyNetworkManager**" que se encarga de gestionar las peticiones al servidor. Utiliza los métodos "**getZona**" y "**setPerfil**" de "**AuxiliarCliente**" para gestionar la comunicación.
public static volatile AuxiliarCliente ac	Almacena el objeto de la clase "**AuxiliarCliente**" que se encarga de realizar las peticiones, enviar y recibir en el socket.
protected void onCreate(Bundle savedInstanceState)	Método encargado de crear la "Activity" principal, leer los ajustes, datos personales, e iniciar la comunicación con el servidor.
protected void onItemSelected(String id)	Gestiona las acciones realizadas en la pantalla principal. Se encarga de cambiar a la pantalla seleccionada por el usuario, pasada como argumento, "id".
private void leeDatos(String [] archivos)	Lee y carga en la aplicación los datos personales almacenados en el dispositivo local.
private void leeEstadoZonas(String [] archivos)	Lee y carga en la aplicación los ajustes de las zonas sobre las que se quiere recibir promociones.
private boolean existe(String [] archivos, String archbusca)	Comprueba que el archivo "**archbusca**" existe en la lista de archivos "**archivos**".

Figura 3.2.1-1 Clase contenidoListActivity

La clase "**contenidoListFragment**" es la que se encarga de construir la lista del menú principal, para que la clase anterior la pueda mostrar.

contenidoListFragment extends ListFragment	
private Callbacks mCallbacks	Almacena el objeto "Callbacks" devuelto cuando se selecciona un elemento de la lista del menú principal ("**contenidoListActivity**").
public interface Callbacks { **public void onItemSelected(String id);** **}**	Interfaz implementada en "**contenidoListActivity**" para el uso de callbacks con "**contenidoListFragment**".
public contenidoListFragment()	Constructor vacío necesario para instanciar la clase en "**contenidoListActivity**". En Android 4.2 es necesario implementar el constructor vacío para poder utilizarlo.
public static ArrayAdapter<String> miListAdapter	Almacena la lista de elementos a mostrar en el menú principal.
public void onCreate(Bundle savedInstanceState)	Crea el fragment ("**contenidoListFragment**"), y llama a "**actualizaLista**" para inicializar el menú principal con las promociones activas.
public void onAttach(Activity activity)	Inicializa los callbacks.
public void onDetach()	Resetea los callbacks a unos por defecto.
public void onListItemClick(ListView listView, View view, int position, long id)	Averigua que promoción corresponde al elemento seleccionado de la lista.
public void setActivatedOnItemClick(boolean activatedOnItemClicl)	Permite inicializar los elementos de la lista como "activados para seleccionar".
private void setActivatedPosition(int position)	Almacena la posición correspondiente al elemento seleccionado del menú principal.
public void actualizaLista()	Actualiza la lista de promociones visibles.

Figura 3.2.1-2 Clase contenidoListFragment

Las siguientes clases "**contenidoDetailActivity**" y "**contenidoDetailFragment**" han sido utilizadas durante el desarrollo de la aplicación para mostrar las zonas en el menú principal. Los elementos "**Datos personales**" y "**Ajustes**" del menú principal se tratan como si fueran un par de zonas más, aunque con peculiaridades. Se ha implementado de esta forma para facilitar el desarrollo y simplificar el funcionamiento de la aplicación.

La clase "**contenidoDetailActivity**" es la encargada de crear la pantalla de cada zona una vez seleccionada en el menú principal.

contenidoDetailActivity extends AppCompatActivity	
public void onCreate(Bundle savedInstanceState)	Crea la "Activity" de la zona.
public boolean onOptionsItemSelected(MenuItem item)	Cuando se le pase como argumento el "item" cuyo id sea el del menú principal, saldrá del detalle de la zona y la aplicación volverá a mostrar la lista de promociones.
public void setActionBar()	Modifica la barra superior para mostrar el título de la zona.

Figura 3.2.1-3 Clase contenidoDetailActivity

La clase "**contenidoDetailFragment**" se encarga de gestionar la vista de la zona mostrada.

contenidoDetailFragment extends Fragment	
public static final String ARG_ITEM_ID	Almacena el id del último elemento seleccionado en el menú.
public static final String MY_TITLE	Almacena el título del último seleccionado en el menú.
private Zona mItem	Almacena la zona a la que corresponde la promoción seleccionada.
public contenidoDetailFragment()	Constructor vacío.
public void onCreate(Bundle savedInstanceState)	Crea el "Fragment" y actualiza la zona, a la última seleccionada.
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)	Modifica la pantalla para que se ajuste al elemento seleccionado del menú principal.

Figura 3.2.1-4 Clase contenidoDetailFragment

A continuación se muestran las diferentes clases usadas para implementar los diferentes tipos de promociones clasificadas según su contenido, ya sea texto, imagen o url.

promociónTexto extends AppCompatActivity	
Zona mItem	Almacena la zona a la que corresponde la promoción seleccionada.
Contenido mContenido	Almacena la promoción seleccionada.
private TextView miTexto	Almacena la vista del texto que se muestra en pantalla.
public void onCreate(Bundle savedInstanceState)	Crea la pantalla de la promoción seleccionada. Llama a "**setActionBar**" con "Título zona" - "Título promoción" y actualiza el texto de la pantalla con la descripción de la promoción.
private void setActionBar(String heading)	Modifica la barra superior para mostrar la promoción.

Figura 3.2.1-5 Clase promocionTexto

promociónImagen extends AppCompatActivity	
Zona mItem	Almacena la zona a la que corresponde la promoción seleccionada.
Contenido mContenido	Almacena la promoción seleccionada.
private TextView miTexto	Almacena la vista del texto que se muestra en pantalla.
private ImageView miImagen	Almacena la vista de la imagen que se muestra en pantalla.
public void onCreate(Bundle savedInstanceState)	Crea la pantalla de la promoción seleccionada. Llama a "**setActionBar**" con "Título zona" - "Título promoción", actualiza el texto de la pantalla con la descripción de la promoción, y la respectiva imagen.
private void setActionBar(String heading)	Modifica la barra superior para mostrar la promoción.

Figura 3.2.1-6 Clase promocionImagen

promociónURL extends AppCompatActivity	
Zona mItem	Almacena la zona a la que corresponde la promoción seleccionada.
Contenido mContenido	Almacena la promoción seleccionada.
private TextView miTexto	Almacena la vista del texto que se muestra en pantalla.
private Button miBoton	Almacena la vista del botón con la URL a la que redirige la promoción.
public void onCreate(Bundle savedInstanceState)	Crea la pantalla de la promoción seleccionada. Llama a "**setActionBar**" con "Título zona" - "Título promoción", actualiza el texto de la pantalla con la descripción de la promoción y la URL del botón.
private void setActionBar(String heading)	Modifica la barra superior para mostrar la promoción.

Figura 3.2.1-7 Clase promocionURL

La clase "**contenidoPersonaActivity**" ha sido desarrollada para gestionar la pantalla relacionada con la opción "**Datos personales**". En ella se implementan los campos para introducir el nombre, la fecha mediante un desplegable y el sexo del usuario de la aplicación. El usuario puede introducir los datos, y hasta que no se solicite, pulsando el botón mostrado en pantalla, no se realizará la modificación. Al pulsar el botón se actualizarán los datos personales en el dispositivo local y se pedirá al servidor la modificación de la base de datos.

contenidoPersonaActivity extends AppCompatActivity	
private Toolbar toolbar	Almacena la barra superior de la pantalla.
private EditText inputNombre	Almacena la casilla en la que se introduce el nombre de usuario.
private TextInputLayout inputLayoutNombre	Almacena el "Layout" de la casilla del nombre.
private TextIntputLayout intputLayoutFecha	Almacena el "Layout" de la casilla de la fecha.
private TextView inputFecha	Almacena la casilla en la que se introduce la fecha.

private DatePicker dpResult	Almacena el desplegable de la fecha.
private RadioButton radioButton	Almacena el "RadioButton" para selecciona el sexo.
private Button btnSignUp	Almacena el botón que se usa para enviar los datos al servidor.
private static String nombre	Almacena el nombre del usuario.
private static String sexo	Almacena el sexo del usuario.
private static int year	Almacena el año de la fecha.
private static int month	Almacena el mes de la fecha.
private static int day	Almacena el día de la fecha.
private static int year_aux	Utilizado como auxiliar para mostrar el año por pantalla sin modificar el que se guarda en el dispositivo ni el de la base de datos.
private static int month_aux	Utilizado como auxiliar para mostrar el mes por pantalla sin modificar el que se guarda en el dispositivo ni el de la base de datos.
private static int day_aux	Utilizado como auxiliar para mostrar el día por pantalla sin modificar el que se guarda en el dispositivo ni el de la base de datos.
static final int DATE_DIALOG_ID	Constante para inicializar el desplegable de la fecha.
protected void onCreate(Bundle savedInstanceState)	Asigna los "Layout" a sus respectivos objetos, e inicializa los objetos anteriores. Crea la vista de "Datos personales" y carga los datos almacenados en el dispositivo.
public void setCurrentDateOnView()	Modifica la fecha de nacimiento con los datos almacenados en la variables "**year**", "**month**" y "**day**".
protected Dialog onCreateDialog(int id)	Crea e iniciliza el desplegable de la fecha con los datos almacenados en la variables "**year**", "**month**" y "**day**".
private DataPickerDialog.OnDateSetListener datePickerListener = new DatePickerDialog.OnDateSetListener(){ public void onDateSet(DatePicker view, int seletedYear, int seletedMonth, int selectedDay)	Modifica la fecha de nacimiento mostrada cuando se cierra el desplegable de ésta.

}	
private String nombreDelMes(int mes)	Devuelve el nombre del mes a partir del número (0-11), para ser mostrado por pantalla.
private void submitForm()	Método al que se llama cuando se pulsa el botón para enviar los datos al servidor. Llama a: "**modificaFecha**", "**actualizaDatos**" y "**guadaDatosEnFichero**". Avisa a "**MyNetworkManager**" para que envíe al servidor una petición "**setPerfil**". Por último muestra por pantalla un mensaje avisando que los datos han sido modificados.
private void modificaFecha()	Actualiza "**year**", "**month**" y "**day**" con "**year_aux**", "**month_aux**" y "**day_aux**" para almacenarlos.
private void actualizaDatos()	Modifica el objeto de la clase "**Persona**" con la fecha de "**year**", "**month**" y "**day**". Cambia el nombre de usuario y sexo al último introducido.
private boolean guardaDatosEnFichero()	Almacena los datos personales en el dispositivo local.
public void onRadioButtonClicked(View view)	Gestiona y almacena la selección del "RadioButton" del sexo del usuario.
private boolean validateName()	Comprueba que el usuario ha introducido un nombre, y avisa en caso de que no lo haya hecho.
private boolean validateDate()	Comprueba que se ha introducido una fecha.
private boolean isValidDate()	Comprueba que la fecha introducida no es posterior a la actual.
private void requestFocus(View view)	Remarca el campo(nombre o fecha) en caso de que tenga un valor no válido.
private class MyTextWatcher implements TextWatcher{}	Utilizado en el input del nombre para comprobar que sea válido.
private void setActionBar(String heading)	Modifica la barra superior para mostrar "Datos personales".

Figura 3.2.1-8 Clase contenidoPersonaActivity

La clase "**Persona**" es la que almacena los datos personales del usuario, así como el ID que usará el sistema para identificarlo en la base de datos.

Persona	
private static String id_persona	Almacena el ID del usuario, no visible para éste.
private static String nombre	Almacena el nombre del usuario.
private static Calenda fechaNacimiento	Almacena la fecha de nacimiento.
private static String sexo	Almacena el sexo del usuario.
public Persona()	Constructor vacío. Usado en caso de que no se encuentren datos almacenados en dispositivo sobre el usuario al iniciar la aplicación.
public Persona(Calendar fechaNacimiento, String sexo)	Constructor con los datos básicos para ajustar las promociones a su perfil.
public static String getId_persona()	Devuelve el ID del usuario. Solo se utiliza al mandar las peticiones al servidor.
public static String getNombre()	Devuelve el nombre del usuario.
public static void setNombre(String nombre)	Modifica el nombre del usuario.
public static Calendar getFechaNacimiento()	Devuelve la fecha de nacimiento.
public static void setFechaNacimiento(Calendar fechaNacimiento)	Modifica la fecha de nacimiento.
public static String getSexo()	Devuelve el sexo del usuario.
public static void setSexo(String sexo)	Modifica el sexo del usuario.
public static String toString()	Utilizado durante el desarrollo de la aplicación para mostrar los datos del usuario.
public static String getData()	Utilizado durante el desarrollo de la aplicación para mostrar los datos del usuario.

Figura 3.2.1-9 Clase Persona

La clase "**Zona**" representa cada una de las balizas, en ella se almacenan el ID de la baliza, el título de la zona, descripción, la lista de promociones que tiene asociadas, y si el usuario desea o no recibir las promociones de ésta.

Zona	
private String id	Almacena el ID de la zona (ID de la baliza).
private String titulo	Almacena el título/nombre de la zona.
private String descripcion	Almacena la descripción de la zona.
private ArrayList<Contenido> listaContenidos	Almacena la lista de promociones asociadas a la zona.
private boolean activa	Almacena las preferencias del usuario sobre la zona: - "true" si quiere recibir promociones - "false" si no las quiere recibir
private static contenidoListFragment miListFragment	Almacena el "listFragment" del menú principal. Por problemas de visibilidad y refresco del menú, se ha tenido que implementar de esta forma.
public Zona(String id, String titulo, String descripcion, boolean activa)	Constructor de una zona.
public String getId()	Devuelve el ID de la zona.
public void setId(String id)	Modifica el ID de la zona.
public String getTitulo()	Devuelve el título de la zona.
public void setTitulo(String titulo)	Modifica el título de la zona.
public String getDescripcion()	Devuelve la descripción de la zona.
public void setDescripcion(String descripcion)	Modifica la descripción de la zona.
public void addContenido(Contenido contenido)	Añade la promoción pasada como argumento a la lista de promociones.
public void removeContenido(Contenido contenido)	Elimina la promoción pasada como argumento de la lista de promociones.
public List<Contenido> getListaContenidos()	Devuelve la lista de promociones de la zona..

public List<Contenido> getContenidosActivos()	Devuelve la lista de promociones que están en vigor.
public void setListaContenidos(ArrayList<Contenido> listaContenidos)	Modifica la lista de promociones de la zona.
public boolean isActiva()	Devuelve la preferencia del usuario sobre si quiere o no recibir las promociones de la zona.
public void setActiva(boolean activa)	Modifica la preferencia del usuario sobre la zona.
public static void setMiListFragment(contenidoListFragment mLF)	Permite actualizar la lista del menú principal cuando se modifiquen las preferencias sobre las zonas.
public String toString()	Utilizado durante el desarrollo de la aplicación para mostrar los datos de la zona.
public String getData()	Utilizado durante el desarrollo de la aplicación para mostrar los datos de la zona.

Figura 3.2.1-10 Clase Zona

En la clase descrita a continuación se implementan las promociones, clasificándolas según el contenido a mostrar.

Contenido	
public static final String TEXTO	Constante usada para identificar si el contenido a mostrar es texto.
public static final String IMAGEN	Constante usada para identificar si el contenido a mostrar es un imagen.
public static final String URL	Constante usada para identificar si el contenido a mostrar es una URL.
public static final String VIDEO	Constante usada para identificar si el contenido a mostrar es un video.
private String titulo	Almacena el título de la promoción.
private String tipo	Almacena el tipo de la promoción (descuento, cupón, entrada...).

private String descripcion	Almacena la descripción de la promoción
private String nombreUbicacion	Almacena la ruta en el dispositivo del contenido a mostrar(en caso de que no sea solo texto).
private Calendar fechaInicio	Almacena la fecha de inicio de la promoción.
private Calendar fechaFin	Almacena la fecha de fin de la promoción.
private Contenido()	Constructor vacío.
public Contenido(String titulo, String tipo, String descripcion, String imagen, Calendar fechaInicio, Calendar fechaFin)	Constructor de la clase.
public String getTitulo()	Muestra el título de la promoción.
public void setTitulo(String titulo)	Modifica el título de la promoción.
public void getTipo()	Muestra el tipo de la promoción.
public void setTipo(String tipo)	Modifica el tipo de la promoción.
public String getDescripcion()	Muestra la descripción de la promoción
public void setDescripcion(String descripcion)	Modificar la descripción de la promoción.
public String getNombreUbicacion()	Devuelve la ruta del contenido a mostrar.
public void setNombreUbicacion()	Modifica la ruta del contenido a mostrar.
public Calendar getFechaInicio()	Devuelve la fecha de inicio de la promoción.
public void setFechaInicio(Calendar fechaInicio)	Modifica la fecha de inicio de la promoción.
public Calendar getFechaFin()	Devuelve la fecha de fin de la promoción.
public void setFechaFin(Calendar fechaFin)	Modifica la fecha de fin de la promoción.
public boolean isActivo()	Devuelve si la promoción está aún vigente o no.
public String getData()	Utilizado durante el desarrollo de la aplicación para mostrar los datos de la zona.

Figura 3.2.1-11 Clase Contenido

La siguiente clase se encarga de gestionar cada una de las zonas y sus promociones.

GestionZonas	
public static final String DATOS_PERSONALES_ID	Constante con el ID de la “zona” “Datos personales”.
public static final String DATOS_PERSONALES_TITLE	Constante con el nombre de la “zona” “Datos personales”.
public static final String AJUSTES_ID	Constante con el ID de la “zona” “Ajustes”.
public static final String AJUSTES_TITLE	Constante con el nombre de la “zona” “Ajustes”
public static final Zona ZONA_DATOS_PERSONALES	Constante del objeto de la clase “**Zona**” que hace referencia a los “Datos personales”.
public static final Zona ZONA_AJUSTES	Constante del objeto de la clase ”**Zona**” que hace referencia a los “Ajustes”.
public static Persona PERSONA	Objeto de la clase “**Persona**” en el que se guardan todos los datos personales del cliente en el dispositivo local cuando la aplicación está en funcionamiento.
public static volatile List<String> ITEMS	Lista de los ID de las zonas (cada uno de los ID de las balizas más los ID de “Datos personales” y “Ajustes”).
public static volatile Map<String, Zona> ITEM_MAP	Mapa con todas las zonas, incluyendo “Datos personales” y “Ajustes”.
public static Zona ultimaZonaSeleccionada	Almacena la última zona seleccionada en el menú principal.
public static int indiceUltimaSeleccion	Almacena el índice en “**listaContenidos**” (de la última zona seleccionada) de la última promoción seleccionada.
public static void cargaContenido()	Utilizado durante el desarrollo cuando aún no se había implementado la conexión con la base de datos. Tiene la función de cargar unas pocas zonas predefinidas para comprobar el correcto funcionamiento de la App.
private static int cambiaDia(int dia)	Comprueba si se trata de un día válido.

private static int cambiaMes(int mes)	Comprueba si se trata de un mes válido.
private static void loadItem(Zona item)	Añade una zona al mapa "**ITEM_MAP**", y su ID a "**ITEMS**".
public synchronized static void addZona(Zona zona)	Utilizado durante el desarrollo. Llama a "**loadItem**".
public synchronized static void setZona(Zona zona)	Modifica una zona en "**ITEM_MAP**".
public synchronized static List<String> getItemList()	Devuelve los títulos de todas las zonas activas (según las preferencias del usuario), incluyendo "Datos personales" y "Ajustes".
public synchronized static List<String> getPromociones()	Devuelve los títulos de todas las promociones vigentes asociadas a las zonas de las que el usuario quiere recibir contenido.
public synchronized static List<Zona> getZonas()	Devuelve una lista de zonas (de la clase "**Zona**"), menos "Datos personales" y "Ajustes".
public synchronized static List<String> zonasActivas()	Devuelve una lista con los títulos de las zonas sobre las que se quiere recibir promociones. Menos "Datos personales" y "Ajustes".
public synchronized static Zona buscarZonaPorTitulo(String titulo)	Devuelve el objeto de la clase "**Zona**" que corresponde al título pasado como argumento.
public synchronized static int getIndice(String titulo)	Devuelve el índice en "**ITEMS**" de la zona cuyo título se pasa como argumento.
public synchronized static Zona buscarContenidoPorTitulo(String titulo)	Busca la zona a partir del título de una de sus promociones, y modifica "**ultimaZonaSeleccionada**" e "**indiceUltimaSeleccion**". Utilizado para identificar la promoción que se ha seleccionado en el menú principal, y así posteriormente mostrar la.
public String toStringZona(String id)	Utilizado durante el desarrollo de la aplicación para mostrar los datos de la zona cuyo id se pasa como argumento.

Figura 3.2.1-12 Clase GestionZonas

Las siguientes cuatro clases son las encargadas de gestionar la comunicación con el servidor.

MyNetworkManager extends Thread	
AuxiliarCliente auxCliente	Almacena el objeto que se encarga de utilizar los sockets.
public static volatile boolean setPerfil	Variable que se utiliza por el botón "Modificar" desde la pantalla de "Datos personales" para avisar que se ha modificado el perfil, y se debe actualizar en el servidor.
public MyNetworkManager(AuxiliarCliente auxCliente)	Constructor de la clase que recibe como argumento un objeto de la clase "**AuxiliarCliente**".
public void run()	Se encarga de gestionar "**auxCliente**", tanto pedir que solicite las promociones de una zona al servidor, como la modificación del perfil.

Figura 3.2.1-13 Clase MyNetworkManager

AuxiliarCliente	
private MyStreamSocket mySocket	Almacena el socket que se usa para la comunicación con el servidor.
private InetAddress serverHost	Almacena dirección IP del servidor.
private int serverPort	Almacena el puerto con el que se conecta servidor.
private static volatile String id_zona	Almacena el ID de la zona a consultar. Se utilizan los métodos "**setIdZona**" y "**getIdZona**" para acceder y modificar el valor.
public AuxiliarCliente(String hostName, String portNum) throws IOException	Inicializa y establece la conexión con el servidor mediante un "StreamSocket".
public void fin()	Avisa al servidor y cierra la conexión.
public synchronized void getZona()	Pide al servidor todas las promociones de

throws IOException	la zona cuyo ID esté en "**id_zona**".
public synchronized void setPerfil(String id, String nombre, Calendar, fechaNacimiento, String sexo) throws IOException	Envía al servidor una petición de modificación del perfil del usuario con sus datos personales.
public synchronized String getIdZona()	Devuelve el ID de la zona almacenado en "**id_zona**".
public synchronized void setIdZona(String id)	Modifica el ID de la zona almacenado en "**id_zona**".

Figura 3.2.1-14 Clase AuxiliarCliente

MyStreamSocket extends Socket	
Socket socket	Almacena el socket.
BufferedReader input	Almacena el objeto que se usa para recibir los datos.
PrintWritter output	Almacena el objeto para enviar los datos.
public MyStreamSocket(InetAddress acceptorHost, int acceptorHost) throws SocketException, IOException	Inicializa el objeto "**socket**" de la clase con los datos pasados como argumento.
public MyStreamSocket(Socket socket) throws IOException	Asigna el socket pasado como argumento al objeto "**socket**" de la clase.
private void setStreams() throws IOException	Usado en ambos constructores. Inicializa los objetos "**input**" y "**output**" de la clase con la entrada y salida del objeto "**socket**".
public void sendMessage(String message) throws IOException	Escribe en el socket el dato pasado como argumento para posteriormente enviarlo.
public String receiveMessage() throws IOException	Recibe y devuelve el mensaje escrito en la entrada, "**input**", del objeto "**socket**".
public void close() throws IOException	Cierra el socket para finalizar la conexión.

Figura 3.2.1-15 Clase MyStreamSocket

Las siguientes clases son las utilizadas para implementar la lista de “CheckBox” utilizada en la sección de “**Ajustes**”.

MiCheckBox implements Comparable<CheckBox>	
private String texto	Almacena el texto del “CheckBox”.
private boolean checked	Almacena el estado del “CheckBox”.
public MiCheckBox(String texto, boolean checked)	Constructor de la clase con el texto y el estado del “CheckBox”.
public void setChecked(boolean value)	Modifica el estado del “CheckBox”.
public void getChecked()	Devuelve el estado del “CheckBox”.
public String getText()	Devuelve el texto del “CheckBox”.
public void setText(String texto)	Modifica el texto del “ChecBox”.
public int compareTo(MiCheckBox other)	Permite comparar los “CheckBox” por su texto.

Figura 3.2.1-16 Clase MiCheckBox

MiCheckBoxList extends ListActivity	
private MiCheckBoxListAdapter mListAdapter	Almacena la lista de objetos de la clase “**MiCheckBox**”, uno por cada zona.
private List<Zona> lista_zonas	Almacena una lista de las zonas. Por temas de visibilidad y para evitar fallos se ha preferido duplicar la lista cuando se crea una nueva pantalla de “Ajustes”.
public void onCreate(Bundle saveInstanceState)	Crea la pantalla con un objeto de la clase “**MiCheckBox**” por cada zona, con el estado actual de cada una de ellas.
protected void onListItemClick(ListView l, View v, int position, long id)	Marca o desmarca el “CheckBox” seleccionado, dependiendo del estado anterior.
protected void onDestroy()	Almacena en el fichero “**ARCHIVO_ESTADO_ZONAS**” de la clase “**contenidoListActivity**” las preferencias del usuario (el estado de cada una de las zonas).

Figura 3.2.1-17 Clase MiCheckBoxList

MiCheckBoxListAdapter extends BaseAdapter	
private Context mContext	Almacena el contexto de la pantalla.
private List<MiCheckBox> mItems	Almacena la lista de objetos "**MiCheckBox**", uno por cada zona.
public MiCheckBoxListAdapter(Context context)	Constructor de la clase, al que se le pase el contexto anterior.
public void addItem(MiCheckBox it)	Añade a la lista "**mItems**" el elemento pasado como argumento.
public long getItemId(int position)	Devuelve la posición en la lista del elemento solicitado.
public View getView(int position, View convertView, ViewGroup parent)	Devuelve una vista de la clase "**MiCheckBoxListView**" para no perder el estado de los "CheckBox".

Figura 3.2.1-18 Clase MiCheckBoxListAdapter

MiCheckBoxListView extends LinearLayout	
private TextView texto	Almacena el texto del "CheckBox" mostrado por pantalla.
private CheckBox checkBox	Almacena el "CheckBox" mostrado por pantalla.
private MiCheckBox checkBoxText	Almacena el objeto de la clase "**MiCheckBox**". Usado para simplificar la gestión de los "CheckBox" y permitir la creación de una lista de éstos.
public MiCheckBoxListView(Context context, MiCheckBox aCheckBoxifiedText)	Constructor de la clase. Inicializa "**checkBoxText**" con el valor de "**aCheckBoxText**", y relaciona la acción de seleccionar "**checkBox**" con la misma de "**checkBoxText**".
public void toggleCheckBoxState()	Modifica el estado del "CheckBox".
public void setCheckBoxState(boolean bool)	Modifica el estado del "CheckBox" al pasado como argumento.
public boolean getCheckBoxState()	Devuelve el estado del "CheckBox".

Figura 3.2.1-19 Clase MiCheckBoxListView

3.2.2 Diagrama de clases del servidor

A continuación se explican las clases que componen el servidor.

La clase MyDataAccess es la interfaz entre el servidor y la base de datos, se encarga de gestionar las peticiones a la base de datos, tanto consultas de zonas como modificaciones de perfil.

MyStreamSocket es una implementación básica de sockets de tipo stream. La clase ServidorSockets es la que se encarga de aceptar las peticiones de la App e iniciar HiloServidor para gestionarlas. HiloServidor inicializa un objeto de la clase MyDataAccess, prepara la consulta y la envía a MyDataAccess para que la realice en la base de datos. Cuando HiloServidor recibe el resultado de la consulta, lo envía a la App en formato json [JSON].

MyDataAccess	
private String _user	Almacena el nombre de usuario de la base de datos.
private private String _pass	Almacena la contraseña asociada al usuario anterior.
private String _db	Almacena el nombre de la base de datos.
private String _dirIP	Almacena la dirección IP en la que se encuentra la base de datos.
private String _url	Almacena la dirección url con los datos anteriores.
private Connection conn	Objeto de la clase "**Connection**" que permite la comunicación con la base de datos.
public MyDataAccess()	Establece la conexión con la base de datos utilizado el objeto "**conn**" con "**_url**" como dirección.
public ResultSet getQuery(String _query)	Método usado para realizar las consultas en la base de datos. Recibe como argumento la consulta a realizar.
public void setQuery(String _query)	Método usado para realizar los "**UPDATE"** en la base de datos. Recibe como argumento la actualización de los datos personales del usuario.

Figura 3.2.2-1 - Clase MyDataAccess

La siguiente clase es idéntica a la que recibe el mismo nombre implementada en la aplicación. Su funcionalidad es permitir la conexión entre las dos partes del sistema.

MyStreamSocket extends Socket	
Socket socket	Almacena el socket.
BufferedReader input	Almacena el objeto que se usa para recibir los datos.
PrintWritter output	Almacena el objeto para enviar los datos.
public MyStreamSocket(InetAddress acceptorHost, int acceptorHost) throws SocketException, IOException	Inicializa el objeto "**socket**" de la clase con los datos pasados como argumento.
public MyStreamSocket(Socket socket) throws IOException	Asigna el socket pasado como argumento al objeto "**socket**" de la clase.
private void setStreams() throws IOException	Usado en ambos constructores. Inicializa los objetos "**input**" y "**output**" de la clase con la entrada y salida del objeto "**socket**".
public void sendMessage(String message) throws IOException	Escribe en el socket el dato pasado como argumento para posteriormente enviarlo.
public String receiveMessage() throws IOException	Recibe y devuelve el mensaje escrito en la entrada, "**input**", del objeto "**socket**".
public void close() throws IOException	Cierra el socket para finalizar la conexión.

Figura 3.2.2-2 - Clase MyStreamSocket

ServidorSockets	
static int puertoServidor	Almacena el puerto del servidor.
static ServerSocket mySocketStream	Almacena el objeto de la clase ServerSocket que se encarga de recibir las peticiones de la aplicación.
public void main(String [] args)	Método principal del servidor. En él se reciben y aceptan las peticiones de la aplicación. Instancia e inicia un nuevo "**Thread**" con un objeto de la clase "**HiloServidor**" para cada petición aceptada.

Figura 3.2.2-3 - Clase ServidorSocket

HiloServidor implements Runnable	
private static final int DIA	Almacena el campo de la clase "**Calendar**" en el que se almacena el día. Utilizado en los métodos "**getZona**" y "**setPerfil**" para evitar la repetición de código.
private static final int MES	Almacena el campo de la clase "**Calendar**" en el que se almacena el mes. Utilizado en los métodos "**getZona**" y "**setPerfil**" para evitar la repetición de código.
private static final int ANYO	Almacena el campo de la clase "**Calendar**" en el que se almacena el año. Utilizado en los métodos "**getZona**" y "**setPerfil**" para evitar la repetición de código.
private static MyDataAccess bbdd	Objeto de la clase "**MyDataAccess**" que almacena la conexión con la base de datos.
MyStreamSocket myDataSocket	Almacena el socket para permitir la comunicación con la aplicación.
HiloServidor(MyStreamSocket myDataSocket)	Asigna el socket pasado como argumento al objeto "**myDataSocket**" de la clase.
public void run()	Gestiona la petición de la aplicación. Ya sea obtener los datos de una zona, modificar el perfil o cerrar la conexión.
private String getZona(String id_zona)	Recibe como argumento el id de la zona sobre la que se quiere recibir las promociones, y devuelve un String en formato JSON con éstas. Para ello hace uso del método "**getQuery**" del objeto "**bbdd**" de la clase.
private synchronized void setPerfil(String id, String nombre, String fechaNacimiento, String sexo)	Recibe como argumentos el id, nombre, fecha de nacimiento y sexo del usuario de la aplicación. Hace uso del método "**setQuery**" del objeto "**bbdd**" de la clase para modificar su perfil.

Figura 3.2.2-4 - Clase HiloServidor

3.3 Diseño de la interfaz

En esta sección se muestran las diferentes pantallas de la interfaz de usuario. La figura 3.3-1 se corresponde a la pantalla principal y menú de la aplicación. En esta lista se muestra cada uno de los elementos de la lista corresponde a una de las promociones a las que el usuario está suscrito y las opciones para modificar los datos personales y ajustes.

Cuando el usuario de la App seleccione una de las promociones podrá acceder a su contenido (Figura 3.3-2). En caso de seleccionar Datos personales o Ajustes accederá a sus respectivas pantallas, Figura 3.3-3 y Figura 3.3-4, pudiendo modificar sus datos personales o la lista de zonas sobre las que se quiere recibir promociones.

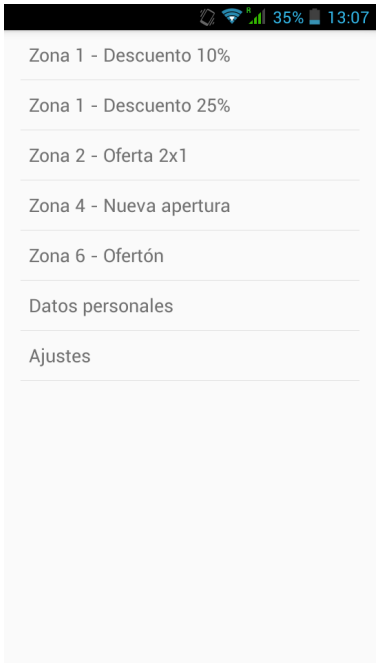


Figura 3.3-1
Menú principal

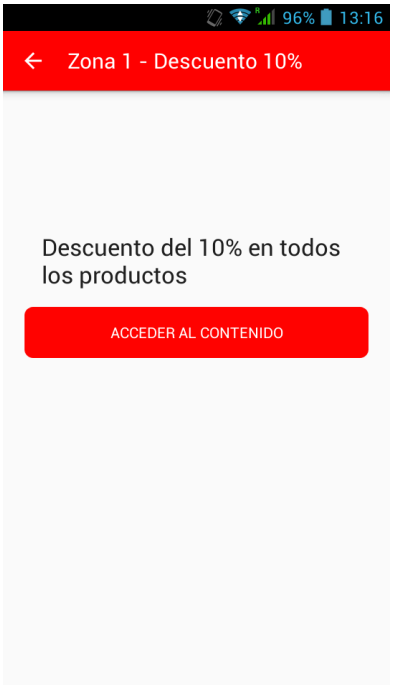


Figura 3.3-2
Información sobre la promoción

Figura 3.3-3
Datos Personales

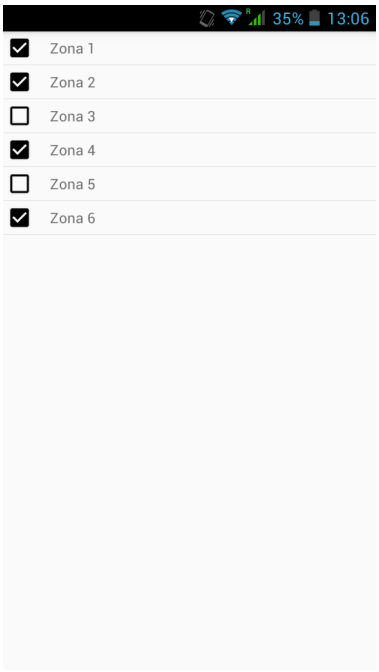


Figura 3.3-4
Ajustes

4. Planificación del proyecto

4.1 Documentación

El método de trabajo que se va a seguir consta de tres grandes bloques, estos son:

- Desarrollo de la propuesta técnica: la parte inicial del proyecto, en la que se define el proyecto, sus características y requisitos. En esta parte se especificarán las herramientas a utilizar así como los lenguajes de programación.

    - Documentación: se informará sobre la documentación proporcionada, además cuando sea necesario se buscará información adicional que complemente a la anterior.

    - Planificación: se definirán y programará las tareas a realizar para la realización del desarrollo técnico del proyecto. También se dispondrá a la redacción de la propuesta técnica.

- Desarrollo técnico del proyecto:

    - Desarrollo de la App: se dispondrá a la realización de una interfaz de usuario básica para comprobar el correcto funcionamiento de la comunicación con las balizas. Posteriormente, con el desarrollo de la comunicación con la BBDD se procederá a realizar una interfaz de usuario más amigable y con mejor aspecto. A continuación, se implementará y añadirá el sistema de geolocalización IPS mediante el método matemático de la trilateración[TRILAT] a partir de las señales RSSI [RSSI] de las balizas.

    - Para finalizar el desarrollo de la App y durante el transcurso de este, se realizarán toda clase de pruebas para comprobar el correcto funcionamiento de esta. También se testeará que todos los cambios realizados a la BBDD desde la plataforma web aparezcan correctamente en la App.

    - Una vez realizadas todas las comprobaciones se procederá a la implantación y entrega del sistema.

- Documentación y presentación del Trabajo Final de Grado: durante la realización del proyecto se realizarán informes semanales sobre lo realizado hasta la fecha. Además, durante el transcurso de este también se documentará y redactará la memoria técnica del TFG explicando todo el trabajo realizado. Finalmente se procederá a la presentación del TFG ante el tribunal.

4.2 Metodología y definición de tareas

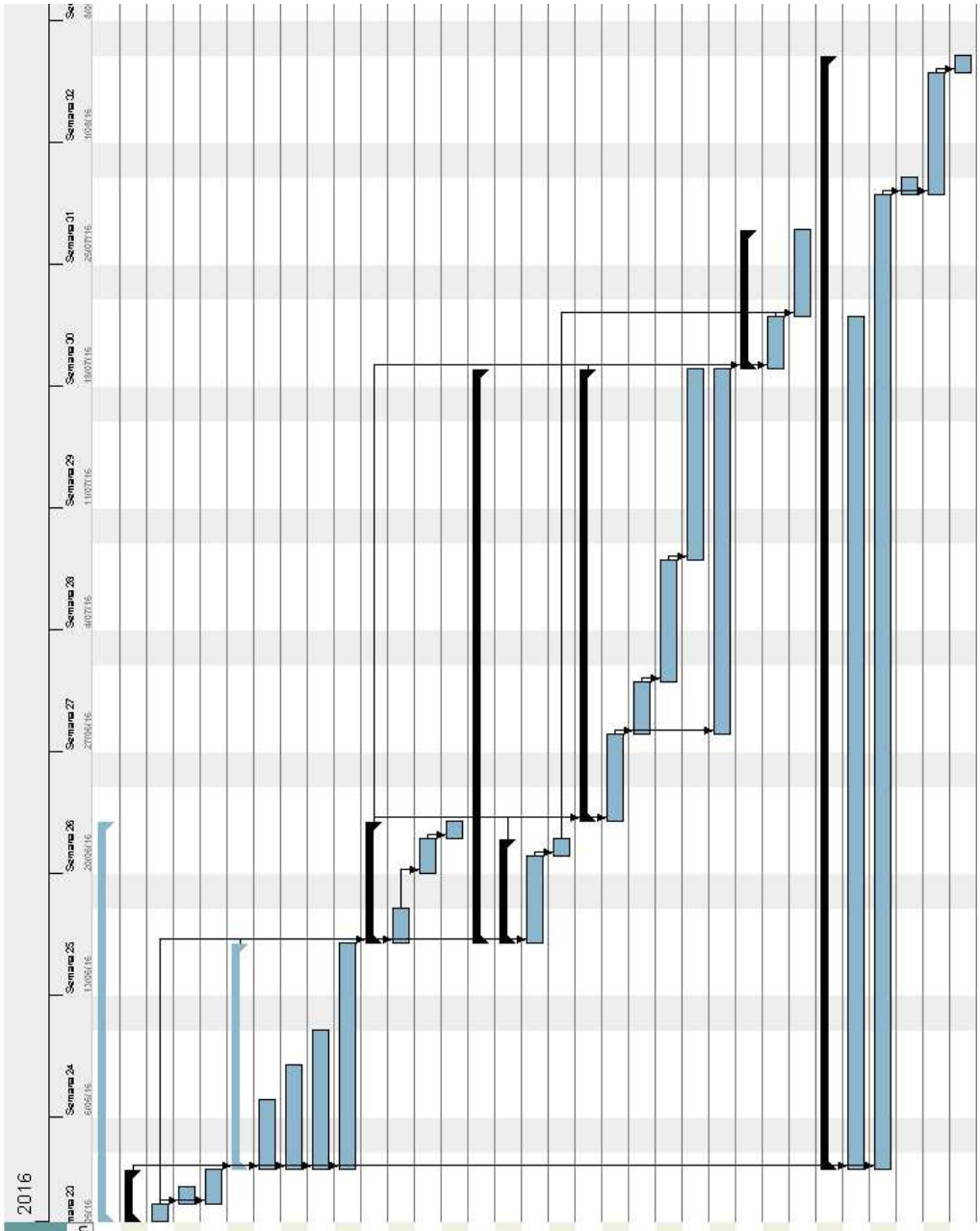
En la siguiente figura se muestra una planificación estimada de las tareas a realizar durante el transcurso del proyecto:

Nº	Descripción	Tiempo(h)	Dependencias
1	**Desarrollo de la propuesta técnica**	**78 h**	
1.1	**Inicio**	**11 h**	
1.1.1	Definir el proyecto con el tutor y el supervisor	1	
1.1.2	Definir el método de trabajo y la documentación	5	1.1.1
1.1.3	Definir formato y estándares de trabajo	5	1.1.1
1.2	**Documentar y planificar el proyecto**	**45 h**	
1.2.1	Documentarse sobre Android y su IDE	16	1.1
1.2.2	Documentarse sobre iOS	16	1.1
1.2.3	Documentarse sobre la base de datos a utilizar	8	1.1
1.2.4	Identificar alcance y objetivos	5	1.1
1.3	**Planificar el proyecto**	**22 h**	
1.3.1	Definir tareas y planificar fechas	12	1.2
1.3.2	Redactar y documentar la propuesta técnica	10	1.3.1
1.3.3	Entregar la propuesta técnica	0	1.3.2
2	**Desarrollo técnico del proyecto**	**235 h**	
2.1	**Crear la base de datos**	**30 h**	
2.1.1	Diseñar y crear la base de datos	25	1.2
2.1.2	Insertar datos iniciales en la base de datos	5	2.1.1
2.2	**Desarrollo de la App móvil**	**200 h**	
2.2.1	Programación de la interfaz usuario	40	1.3
2.2.2	Programación de la comunicación con las balizas	40	2.2.1
2.2.3	Programación de la comunicación con la BBDD	32	2.2.2

2.2.4	Programación de la geolocalización	56	2.2.3
2.2.5	Pruebas	32	2.2.1
2.3	**Puesta en marcha**	**5 h**	
2.3.1	Implantación	5	2.2
2.3.2	Entrega final	0	2.3.1
3	**Documentación y presentación TFG**	**139 h**	
3.1	Redacción de informes semanales	8	1.1
3.2	Redacción de la memoria técnica	100	1.1
3.3	Entrega de la memoria técnica	0	3.2
3.4	Preparación de la presentación oral	30	3.2
3.5	Presentación oral	1	3.4

Figura 4.2-1 - Planificación inicial

En la página siguiente se muestra el diagrama de Gantt sobre la planificación inicial, Figura 4.2-2 Diagrama sobre la planificación inicial.



Nombre	Fecha de inicio	Fecha de fin
Desarrollo de la propuesta técnica	31/05/16	22/06/16
Inicio	31/05/16	2/06/16
Definir el proyecto con el tutor y el supervisor	31/05/16	31/05/16
Definir el método de trabajo y la documentación	1/06/16	1/06/16
Definir el formato y estándares de trabajo	1/06/16	2/06/16
Documentar y planificar el proyecto	3/06/16	15/06/16
Documentarse sobre Android y su IDE	3/06/16	6/06/16
Documentarse sobre iOS	3/06/16	8/06/16
Documentarse sobre la base de datos a utilizar	3/06/16	10/06/16
Identificar el alcance y objetivos	3/06/16	15/06/16
Planificar el proyecto	16/06/16	22/06/16
Definir tareas y planificar fechas	16/06/16	17/06/16
Redactar y documentar la propuesta técnica	20/06/16	21/06/16
Entregar la propuesta técnica	22/06/16	22/06/16
Desarrollo técnico del proyecto	16/06/16	18/07/16
Crear la base de datos	16/06/16	21/06/16
Diseñar y crear la base de datos	16/06/16	20/06/16
Insertar los datos iniciales en la base de datos	21/06/16	21/06/16
Desarrollo de la App móvil	23/06/16	18/07/16
Programación de la in terfaz de usuario	23/06/16	27/06/16
Programación de la comunicación con las balizas	28/06/16	30/06/16
Programación de la comunicación con la BBDD	1/07/16	7/07/16
Programación de la geolocalización	8/07/16	18/07/16
Pruebas	28/06/16	18/07/16
Puesta en marcha	19/07/16	26/07/16
Implantacción	19/07/16	21/07/16
Entrega final	22/07/16	26/07/16
Documentación y presentación TFG	3/06/16	5/08/16
Redacción de informes semanales	3/06/16	21/07/16
Redaccción de la memoria técnica	3/06/16	28/07/16
Entrega de la memoria técnica	29/07/16	29/07/16
Preparación de la presentación oral	29/07/16	4/08/16
Presentación oral	5/08/16	5/08/16

4.3 Tecnologías utilizadas

A continuación se describen las tecnologías utilizadas, algunas de las herramientas de trabajo, y algunas de las que deberían usarse en las partes no implementadas.

4.3.1 Funcionalidad de los iBeacons y EddyStones

La funcionalidad de las balizas, iBeacons y EddyStone, es la que permite asociar contenidos a espacios físicos. Esto permite acceder a diferentes contenidos dependiendo del lugar en el que se encuentre el usuario.

Mediante la emisión de un ID único se podrá identificar cada una de las balizas para posteriormente acceder al contenido asociado previa consulta al servidor.

- iBeacon: dispositivo y protocolo basado en balizas desarrollado por Apple, permite la asociación de contenidos a las balizas [iBeacon1] [iBeacon2].

  El tipo de baliza del que se disponía durante la estancia en prácticas era ContextBeacon Pro [EBEACON1] disponible en la tienda de la web easiBeacon [EBEACON2], utilizadas en la solución informática de Marketing de Proximidad easyContext [ECONT1] [ECONT2]. Compatible con iBeacon y EddyStone.

- EddyStone: el homólogo de Google al iBeacon de Apple. [EDDY1] [EDDY2].

- Buetooth Low Energy [BLUELE] y Bluetooth 4.0 [BLUE41] [BLUE42]: son dos estándares de comunicación Bluetooth diferentes, utilizados por iBeacon y EddyStone respectivamente.

4.3.2 Funcionalidad de los dispositivos smartphone Android e iOS

El dispositivo smartphone es el que se encarga de detectar las balizas descritas anteriormente y mostrar las promociones asociadas. En primer lugar el dispositivo deberá detectar la baliza(zona). En el mismo momento en el que se detecte la zona el dispositivo realizará una consulta al servidor solicitando toda la información disponible, tanto contenidos como promociones en vigor, para mostrarla posteriormente.

Además el dispositivo móvil permite el registro del usuario para posteriormente ofrecerle contenidos adaptados a su perfil.

- Android (Sistema Operativo): sistema operativo basado en linux y desarrollado para dispositivos móviles [ANDRSO].

  Para la comunicación con dispositivos iBeacon es necesaria la versión 4.3 [ANDR43] o superior de este SO y Bluetooth 4.0.

- Android (lenguaje): el lenguaje de programación de los dispositivos que utilizan el SO Android, JAVA con librerias específicas para dicho SO.

- Android Studio: es el entorno de desarrollo integrado para la plataforma Android.

- iOS: sistema operativos utilizado por los dispositivos móviles de Apple, iPhone entre otros [IOS]. Para el desarrollo, validación y realización de pruebas es necesario disponer de una licencia de desarrollador, dispositivo móvil con iOS y Mac.

4.3.3 Funcionalidad de una base de datos

La funcionalidad de la base de datos permite el registro y modificación de las zonas y sus contenidos mediante una página web externa. También se permite por parte del usuario la creación y modificación de un perfil básico previa comunicación con el servidor.

En la base de datos se almacenará cada zona, con su ID y las promociones asociadas. Así como el perfil de cada usuario. Más detallado en el TFG de Iván Chinchilla.

4.3.4 Funcionalidad de un servidor

El servidor es la interfaz entre la BBDD y la App. Su función principal es gestionar la base de datos sin que el usuario final pueda acceder a ella directamente. Se encarga de filtrar todas las peticiones del usuario así como de enviarle el contenido solicitado.

Cuando el servidor reciba el ID de una de las zonas, devolverá todas las promociones activas asociadas, si el usuario ha introducido sus datos personales, éstas se adaptarán a su perfil.

4.3.5 Funcionalidad de la localización en interiores

La localización en interiores o IPS(Indoor Positioning System) en la que se encarga de saber en cada momento el punto exacto donde se encuentra el usuario con su smartphone. Esto permite trazar rutas desde el dispositivo móvil hasta las balizas detectadas.

El método para calcular la posición respecto de las balizas es la trilateración, por el cual se puede calcular la posición del usuario de la App a partir de la distancia entre tres balizas y su smartphone. La distancia entre las balizas se calcula mediante una función predefinida en la biblioteca de uso de la balizas al pasarle como argumento el RSSI.

Una vez calculada la posición del usuario, se le podrían ofrecer servicios como orientarlo en el edificio, e incluso, guiarlo de forma que pueda acceder a todas las promociones siguiendo un ruta óptima.

5. Implementación, pruebas y documentación

5.1 Desarrollo del proyecto

El desarrollo del proyecto se ha realizado en tres etapas, desarrollo de la App, desarrollo del servidor e implementación de la comunicación entre servidor y App.

Durante el desarrollo de la aplicación primero se ha realizado la lógica de la App, comprobando el correcto funcionamiento de ésta, desde la asociación de promociones a sus zonas hasta la gestión de los datos personales. Una vez terminada una primera versión de la gestión interna de la App se ha realizado la interfaz gráfica de ésta, teniendo en cuenta que cada acción debe reflejarse en lógica.

Posteriormente se ha llevado a cabo la creación e introducción en la base de datos de los datos necesarios para el proyecto.

Una vez terminada la App y la base de datos se ha realizado el desarrollo del servidor, un servidor básico cuya función es ser la interfaz entre la aplicación y la base de datos. Su implementación permite una gran modularidad y escalabilidad para implementar posteriormente mejoras si se desean.

Con la App y servidor desarrollados se ha realizado la comunicación entre los dos mediante el envío de archivos JSON utilizando sockets de tipo stream.

En la sección 5.2.1 Diagrama hasta el fin de la estancia, se encuentran el diagrama de Gantt y el desglose aproximado de las horas invertidas en el proyecto hasta el fin de la estancia en prácticas.

En la sección 5.2.2 Diagrama con lo realizado después de la estancia, se encuentran el diagrama de Gantt y el desglose de horas correspondiente al desarrollo de la parte del proyecto descrita en esta memoria.

En la siguiente sección se describen los problemas surgidos durante el desarrollo de la parte del sistema descrita en esta memoria.

5.2 Problemas y cambios sobre la planificación inicial

En este apartado se explican los problemas surgidos durante la realización del proyecto, tanto los surgidos durante la estancia en prácticas como los posteriores.

Los primeros problemas surgieron por no tener la infraestructura necesaria para el desarrollo del proyecto. La falta de ordenadores y dispositivos móviles con los que desarrollar supuso un problema al principio y un cambio en la planificación. La primera semana solo disponíamos, Iván y yo, de un ordenador para documentarnos. Tampoco disponía de un smartphone ni dispositivo iOS con el que desarrollar, aunque no le dimos mucha importancia, pues se nos prometieron ordenadores y un dispositivo android para realizar el trabajo. Por la carencia de material informático decidí utilizar mi propio portátil y smartphone personal. Aunque la segunda semana nos entregaron los ordenadores, decidí seguir usando mi portátil por problemas de temperatura del portátil facilitado por la empresa.

Durante el desarrollo de la App para Android tuve varios problemas a la hora de la implementación por no tener casi conocimientos sobre el lenguaje y sus librerías, y que el material relacionado con la comunicación con la balizas, así como la documentación sobre implementación de algunas de las funcionalidades resulta imprecisa y confusa.

Después de preguntar un par de veces sobre el material necesario para el desarrollo en iOS(iPhone, licencia y un dispositivo Mac) y no recibir respuesta, el tutor y yo decidimos abandonar el desarrollo de la App para iOS.

Al no disponer de ningún dispositivo compatible con el protocolo de comunicación utilizado por las balizas(Bluetooth 4.0 y BLE), se decidió cambiar esta comunicación por un protocolo de comunicación WiFi con un servidor.

Posteriormente surgieron algunos problemas, aunque bastante más comunes y fáciles de resolver. En principio la App debía comunicarse directamente con la bbdd, pero se decidió usar un servidor como interfaz entre ambos para facilitar su implementación y modularidad del sistema. Se decidió utilizar este servidor para la comunicación WiFi.

El desarrollo del servidor ocasionó algunos problemas que fueron resueltos con facilidad, como por ejemplo la comunicación de éste con la bbdd y con la App. Se decidió utilizar objetos de la clase Connection para que la implementación entre la comunicación entre bbdd y servidor fuera trivial, y el uso del formato JSON para lo comunicación entre servidor y App.

Se decidió suprimir la funcionalidad de la localización en interiores por sobrepasar el tiempo de desarrollo de la planificación inicial y la cercanía en las fechas de entrega de la memoria.

5.2.1 Diagrama hasta el fin de la estancia

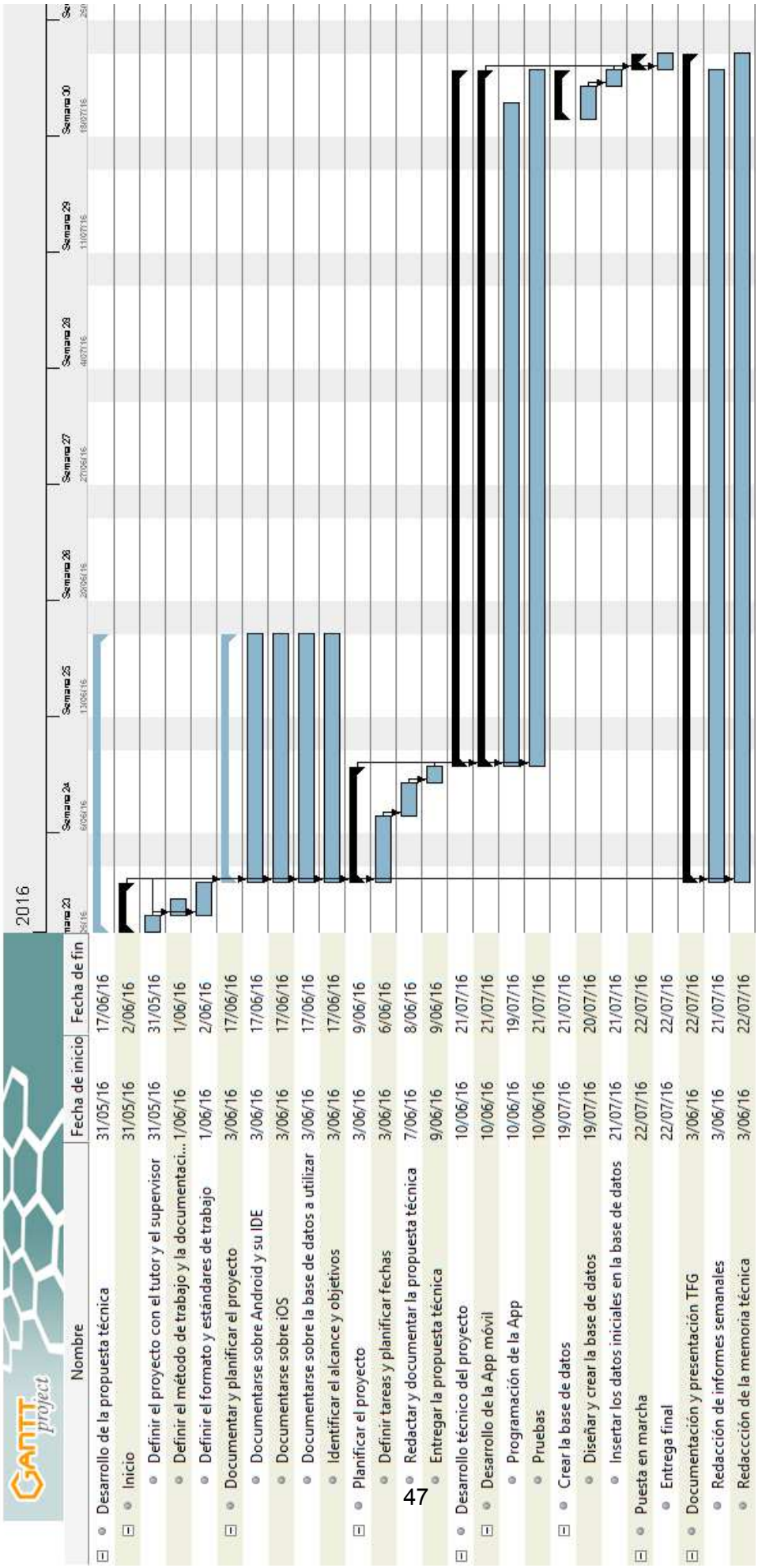
En este apartado se muestra el trabajo realizado durante la estancia en prácticas. Las figuras mostradas a continuación difieren en gran medida de la planificación inicial por los motivos descritos en éste mismo punto, 5.2 Problemas y cambios sobre la planificación inicial.

Nº	Descripción	Tiempo(h)	Dependencias
1	**Desarrollo de la propuesta técnica**	**99 h**	
1.1	**Inicio**	**11 h**	
1.1.1	Definir el proyecto con el tutor y el supervisor	1	
1.1.2	Definir el método de trabajo y la documentación	5	1.1.1
1.1.3	Definir formato y estándares de trabajo	5	1.1.1
1.2	**Documentar y planificar el proyecto**	**70 h**	
1.2.1	Documentarse sobre Android y su IDE	26	1.1
1.2.2	Documentarse sobre iOS	26	1.1
1.2.3	Documentarse sobre la base de datos a utilizar	8	1.1
1.2.4	Identificar alcance y objetivos	10	1.1
1.3	**Planificar el proyecto**	**18 h**	
1.3.1	Definir tareas y planificar fechas	12	1.1
1.3.2	Redactar y documentar la propuesta técnica	6	1.3.1
1.3.3	Entregar la propuesta técnica	0	1.3.2
2	**Desarrollo técnico del proyecto**	**248 h**	
2.1	**Desarrollo de la App móvil**	**224 h**	
2.1.1	Programación de la App	180	1.3
2.1.5	Pruebas	44	2.1.1
2.2	**Crear la base de datos**	**24 h**	
2.2.1	Diseñar y crear la base de datos	22	2.1
2.2.2	Insertar datos iniciales en la base de datos	2	2.2.1
2.3	**Puesta en marcha**	**0 h**	

2.3.2	Entrega final	0	2.2
3	**Documentación y presentación TFG**	**108 h**	
3.1	Redacción de informes semanales	8	1.1
3.2	Redacción de la memoria técnica	100	1.1

Figura 5.2.1-1 - Planificación realizada hasta el fin de la estancia

En la siguiente página se muestra el diagrama de Gantt, Figura 5.2.1 - Diagrama de Gantt sobre la estancia, según lo descrito en la planificación anterior.



Nombre	Fecha de inicio	Fecha de fin
Desarrollo de la propuesta técnica	31/05/16	17/06/16
Inicio	31/05/16	2/06/16
Definir el proyecto con el tutor y el supervisor	31/05/16	31/05/16
Definir el método de trabajo y la documentaci...	1/06/16	1/06/16
Definir el formato y estándares de trabajo	1/06/16	2/06/16
Documentar y planificar el proyecto	3/06/16	17/06/16
Documentarse sobre Android y su IDE	3/06/16	17/06/16
Documentarse sobre iOS	3/06/16	17/06/16
Documentarse sobre la base de datos a utilizar	3/06/16	17/06/16
Identificar el alcance y objetivos	3/06/16	17/06/16
Planificar el proyecto	3/06/16	9/06/16
Definir tareas y planificar fechas	3/06/16	6/06/16
Redactar y documentar la propuesta técnica	7/06/16	8/06/16
Entregar la propuesta técnica	9/06/16	9/06/16
Desarrollo técnico del proyecto	10/06/16	21/07/16
Desarrollo de la App móvil	10/06/16	21/07/16
Programación de la App	10/06/16	19/07/16
Pruebas	10/06/16	21/07/16
Crear la base de datos	19/07/16	21/07/16
Diseñar y crear la base de datos	19/07/16	20/07/16
Insertar los datos iniciales en la base de datos	21/07/16	21/07/16
Puesta en marcha	22/07/16	22/07/16
Entrega final	22/07/16	22/07/16
Documentación y presentación TFG	3/06/16	22/07/16
Redacción de informes semanales	3/06/16	21/07/16
Redaccción de la memoria técnica	3/06/16	22/07/16

5.2.2 Diagrama con lo realizado después de la estancia

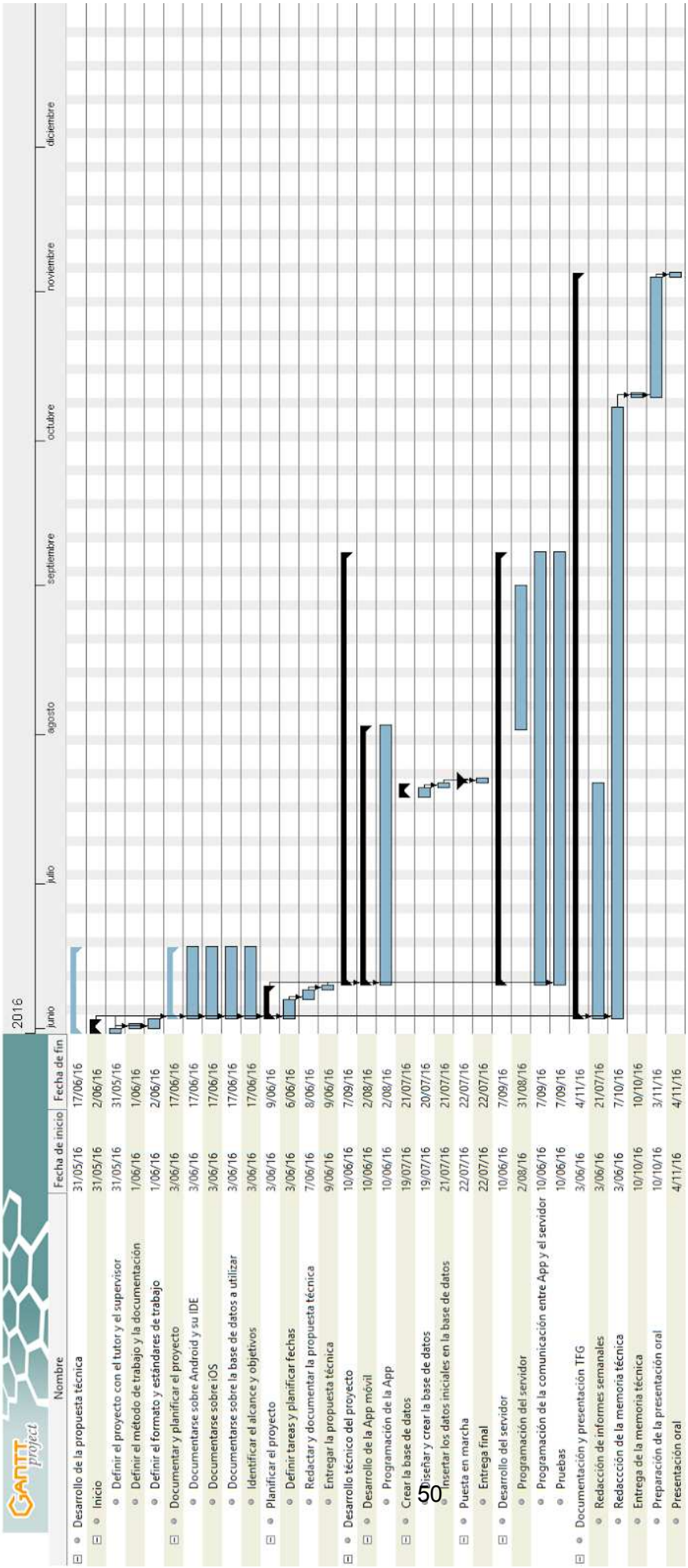
A continuación se muestra la distribución final del tiempo en la realización del trabajo. En este diagrama se muestra, además de lo realizado durante la estancia, el trabajo realizado hasta terminar el proyecto.

Nº	Descripción	Tiempo(h)	Dependencias
1	**Desarrollo de la propuesta técnica**	**99 h**	
1.1	**Inicio**	**11 h**	
1.1.1	Definir el proyecto con el tutor y el supervisor	1	
1.1.2	Definir el método de trabajo y la documentación	5	1.1.1
1.1.3	Definir formato y estándares de trabajo	5	1.1.1
1.2	**Documentar y planificar el proyecto**	**70 h**	
1.2.1	Documentarse sobre Android y su IDE	26	1.1
1.2.2	Documentarse sobre iOS	26	1.1
1.2.3	Documentarse sobre la base de datos a utilizar	8	1.1
1.2.4	Identificar alcance y objetivos	10	1.1
1.3	**Planificar el proyecto**	**18 h**	
1.3.1	Definir tareas y planificar fechas	12	1.1
1.3.2	Redactar y documentar la propuesta técnica	6	1.3.1
1.3.3	Entregar la propuesta técnica	0	1.3.2
2	**Desarrollo técnico del proyecto**	**328 h**	
2.1	**Desarrollo de la App móvil**	**244 h**	
2.1.1	Programación de la App	200	1.3
2.1.5	Pruebas	44	2.1.1
2.2	**Crear la base de datos**	**24 h**	
2.2.1	Diseñar y crear la base de datos	22	2.1
2.2.2	Insertar datos iniciales en la base de datos	2	2.2.1
2.3	**Puesta en marcha**	**0 h**	

2.3.2	Entrega final	0	2.2
2.4	**Desarrollo del servidor**	**60 h**	
2.4.1	Programación del servidor	40 h	2.3
2.4.2	Programación de la comunicación entre App y servidor	40 h	2.4.2
2.4.3	Pruebas	20 h	2.4.3
3	**Documentación y presentación TFG**	**139 h**	
3.1	Redacción de informes semanales	8	1.1
3.2	Redacción de la memoria técnica	100	1.1
3.3	Entrega de la memoria técnica	0	3.2
3.4	Preparación de la presentación oral	30	3.2
3.5	Presentación oral	1	3.4

Figura 5.2.2-1 - Planificación realizada hasta el fin del proyecto

En la siguiente página se muestra el diagrama de Gantt, Figura 5.2.2-2 Diagrama de Gantt sobre el proyecto final, sobre la planificación anterior.


GANTT project
2016
junio
julio
agosto
septiembre
octubre
noviembre
diciembre
Nombre
Fecha de inicio
Fecha de fin
Desarrollo de la propuesta técnica 31/05/16 17/06/16
Inicio 31/05/16 2/06/16
Definir el proyecto con el tutor y el supervisor 31/05/16 31/05/16
Definir el método de trabajo y la documentación 1/06/16 1/06/16
Definir el formato y estándares de trabajo 1/06/16 2/06/16
Documentar y planificar el proyecto 3/06/16 17/06/16
Documentarse sobre Android y su IDE 3/06/16 17/06/16
Documentarse sobre iOS 3/06/16 17/06/16
Documentarse sobre la base de datos a utilizar 3/06/16 17/06/16
Identificar el alcance y objetivos 3/06/16 17/06/16
Planificar el proyecto 3/06/16 9/06/16
Definir tareas y planificar fechas 3/06/16 6/06/16
Redactar y documentar la propuesta técnica 7/06/16 8/06/16
Entregar la propuesta técnica 9/06/16 9/06/16
Desarrollo técnico del proyecto 10/06/16 7/09/16
Desarrollo de la App móvil 10/06/16 2/08/16
Programación de la App 10/06/16 2/08/16
Crear la base de datos 19/07/16 21/07/16
Diseñar y crear la base de datos 19/07/16 20/07/16
Insertar los datos iniciales en la base de datos 21/07/16 21/07/16
Puesta en marcha 22/07/16 22/07/16
Entrega final 22/07/16 22/07/16
Desarrollo del servidor 10/06/16 7/09/16
Programación del servidor 2/08/16 31/08/16
Programación de la comunicación entre App y el servidor 10/06/16 7/09/16
Pruebas 10/06/16 7/09/16
Documentación y presentación TFG 3/06/16 4/11/16
Redacción de informes semanales 3/06/16 21/07/16
Redaccción de la memoria técnica 3/06/16 7/10/16
Entrega de la memoria técnica 10/10/16 10/10/16
Preparación de la presentación oral 10/10/16 3/11/16
Presentación oral 4/11/16 4/11/16

5.3 Validación y pruebas

Durante todo el desarrollo se han ido realizando pruebas para comprobar el correcto funcionamiento de todas y cada una de las funcionalidades tanto por separado como al añadirlas al conjunto de proyecto. Desde comprobación del correcto funcionamiento de la lógica del proyecto, las interfaces gráficas, hasta la gestión de la base de datos.

Como ejemplo de las pruebas realizadas, primero se programó el funcionamiento interno de gestión de las zona con una pantalla mínima para poder lanzar la aplicación en el móvil de pruebas y comprobar mediante la consola de debug del programa Android Studio que se mostraban todos los datos correctamente. Posteriormente se añadió una pantalla básica para mostrarlos en el dispositivo en vez de en el ordenador. Se fueron añadiendo las diferentes funcionalidades, como por ejemplo la lista del menú principal y comprobando que se mostraban todas.

Después se añadieron los ajustes para personalizar las zonas mostradas. Donde surgieron algunos problemas, la lista principal no se actualizaba. Pero se consiguió solucionar compartiendo un objeto del mismo tipo de la lista con el modificador “static”.

Durante el desarrollo de la App se comprobaron tanto el correcto funcionamiento de ésta como los mensajes de debug mostrados en el ordenador fueran los esperados.

Una de las partes finales del desarrollo de la App fué modificar la pantalla principal para que en vez de mostrar las zonas con sus descripciones, se mostraran las promociones asociadas.

En cuanto a la parte del servidor, una vez implementado, se comprobó que todas las consultas y modificaciones a la base de datos tuvieran efecto y se realizaran correctamente.

5.4 Entrega

La entrega a la empresa del trabajo llevado a cabo durante la estancia en prácticas se ha realizado, a petición del supervisor, mediante la entrega de una copia comprimida de una carpeta con todos los archivos que componen la aplicación. El depósito se ha realizado tanto en los ordenadores locales, como en la máquina virtual del servidor.

6. Conclusiones

La realización de este proyecto me ha servido para aumentar mi experiencia en la tecnologías utilizadas Java, mySQL, JSON. Además también para aprender aquellas que no conocía tanto, como son Android y su IDE, y la comunicación con las balizas. Así como la metodología a usar para el cálculo de la posición del usuario de la App a partir del RSSI de éstas.

El hecho de haber desarrollado la App en Android me ha servido para aumentar mis conocimientos sobre dicho lenguaje, ya que mis conocimientos previos se deben a la creación de Apps muy básicas con un par de botones.

Aunque no haber podido realizar la App para iOS y no disponer de un dispositivo compatible con las balizas a sido un poco frustrante, me ha servido para informarme como se debería hacer y tener conocimiento teóricos sobre ambos temas.

En cuanto a mi experiencia en la estancia en prácticas, y pese a los problemas de infraestructura surgidos, me he sentido muy agusto, formando parte del equipo con Iván y los demás compañeros de trabajo.

Cabe destacar que al haber trabajado en una empresa, he podido experimentar lo que es desarrollar en un entorno real, con compañeros y plazos de entrega comunes. Así como tomar decisiones de acuerdo al resto del equipo para que el proyecto pudiera realizarse.

El cambio de requisitos y los posteriores y múltiples cambios en las especificaciones del sistema me resultaron al principio un poco chocantes, e incluso desconcertantes, pues no llegaba a enter el propósito de la realización de las prácticas si no se realizaba lo planificado inicialmente. Pero posteriormente he podido comprobar que aunque el proyecto realizado difiera bastante del inicial, el objetivo de la asignatura es darse cuenta de como se trabaja en un entorno real y los cambios en la planificación que pueden surgir.

Por último me gustaría destacar la modularidad del proyecto así como mejoras que podrían realizarse. La utilización de un servidor como intermediario entre la base de datos y el usuario final de la App y la definición de sus métodos, permite cambiar cualquiera de los elementos del sistema sin afectar al funcionamiento del resto. También se podrían añadir mejoras como una mayor clasificación entre usuarios para ajustar mejor las promociones a sus gustos con pocos cambios. Aumentar la funcionalidad de la App para compartir contenidos entre los usuarios, permitir el posicionamiento mediante GPS, la implementación del pago sin contactos, o incluso el desarrollo de una App para la creación de nuevas promociones. Además, las balizas podrían utilizarse para activar y compartir contenido entre aplicaciones.

Bibliografía

[IPS] Wikipedia. (17 septiembre 2016). *Indoor positioning system.* Recuperado 28 septiembre, desde
https://en.wikipedia.org/wiki/Indoor_positioning_system

[PUSH1] qodeblog. (4 febrero 2015). *¿Qué son las notificaciones push?.* Recuperado 28 septiembre, desde
http://qode.pro/blog/que-son-las-notificaciones-push/

[PUSH2] Wikipedia. (11 agosto 2016). *Tecnología Push.* Recuperado 28 septiembre, desde
https://es.wikipedia.org/wiki/Tecnolog%C3%ADa_Push

[RSSI] Wikipedia. (17 junio 2016). *Received signal strengh indication*. Recuperado 20 junio 2016, desde
https://en.wikipedia.org/wiki/Received_signal_strength_indication

[TRILAT] Wikipedia. (1 junio 2016). *Trilateration*. Recuperado 9 junio 2016, desde
https://en.wikipedia.org/wiki/Trilateration

[JSON] Wikipedia. (25 agosto 2016). *Json*. Recuperado 7 septiembre 2016, desde
https://es.wikipedia.org/wiki/JSON

[IBEACON1] Wikipedia. (22 septiembre 2016). *iBeacon*. Recuperado 28 septiembre 2016, desde
https://en.wikipedia.org/wiki/IBeacon

[IBEACON2] easyContext. *iBeacon: tecnología beacon de Apple*. Recuperado 28 septiembre 2016, desde
https://www.easycontext.com/ibeacon-tecnologia-beacon-apple/

[EBEACON1] easiBeacon store. *ContextBeacon Pro*. Recuperado 9 junio, desde
http://store.easibeacon.com/ContextBeacon-Pro_p_28.html

[EBEACON2] easiBeacon. *iBeacons made easy*. Recuperado 9 junio, desde
http://easibeacon.com/

[ECONT1] easyContext. *easyContext, engage with your costumers*. Recuperado 28 septiembre, desde
https://www.easycontext.com/

[ECONT2] easyContext. *¿Qué es la tecnología beacon?*. Recuperado 28 septiembre, desde
https://www.easycontext.com/beacons/

[EDDY1] Wikipedia. (17 junio 2016). *EddyStone(Google)*. Recuperado 28 septiembre, desde https://en.wikipedia.org/wiki/Eddystone_(Google)

[EDDY2] easyContext. *EddyStone: Google's beacon technology*. Recuperado 28 septiembre, desde
https://www.easycontext.com/en/beacons-eddystone-google/

[BLUELE] Wikipedia. (12 septiembre 2016). *Bluetooth low energy*. Recuperado 28 septiembre, desde
https://en.wikipedia.org/wiki/Bluetooth_low_energy

[BLUE41] bluetooth. (2016). *The Story Behind Bluetooth Technology*. Recuperado 28 septiembre, desde
https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth

[BLUE42] Wikipedia. (23 septiembre 2016). *Bluetooth*. Recuperado 28 septiembre 2016, desde
https://en.wikipedia.org/wiki/Bluetooth

[ANDRSO] Android. (2014). *Android.* Recuperado 3 junio, desde
https://www.android.com/intl/es_es/

[ANDR43] Android. *Android - 4.3 Jelly Bean*. Recuperado 3 junio 2016,
desde
https://www.android.com/intl/es_es/versions/jelly-bean-4-3/

[IOS] Apple. (2016). *iOS - iOS 10 - Apple (ES)*. Recuperado 6 junio 2016, desde
http://www.apple.com/es/ios/ios-10/