

Trabajo Final:
Programación orientada a objetos



Alumno:

Alejandro, Gonzalez.	L.U: 248563.
Iara, Alfaro Lutz.	L.U: 248553.

E-mail:

ale280497@gmail.com
iaraalfarolutz@gmail.com

Introducción

Para el trabajo final de la cátedra de Programación Orientada a Objetos se nos solicitó desarrollar una aplicación mobile que le permita al usuario generar y almacenar distintos tipos de ubicaciones sobre un mapa tales como puntos líneas y polígonos en una base de datos remota.

Para esto fue necesario hacer uso de la API provista por Google Maps y el desarrollo de un backend que brindara los servicios necesarios para ser consumidos por la aplicación.

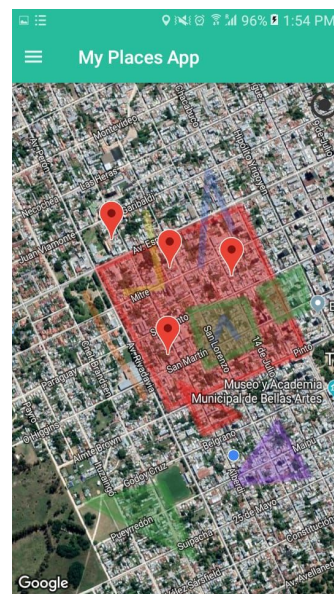
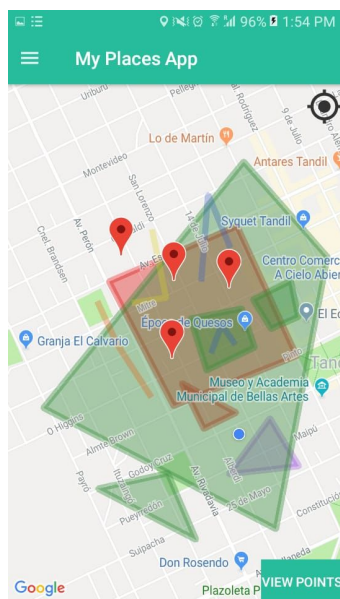
Desarrollo

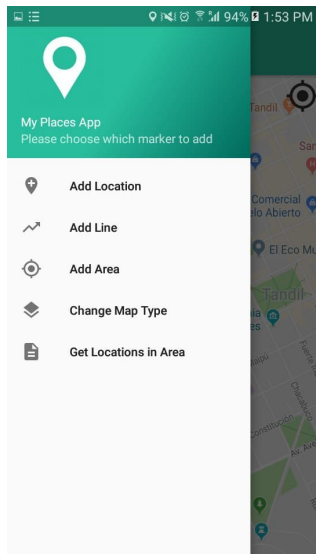
“MyPlacesApp” es una aplicación mobile para Android desarrollada en Java.

Para el diseño de la interfaz y las diferentes pantallas se aplicaron los conceptos vistos en la cátedra de “Introducción a la Programación de Dispositivos Móviles” tales como la utilización de activities e intents para realizar la transición entre las diferentes pantallas y transmitir información a través de las mismas.

Para poder acceder a las funcionalidades provistas por la SDK de Google Maps fue necesario crear un proyecto en la plataforma “Google Cloud” y solicitar una API key e incluirla en nuestro proyecto.

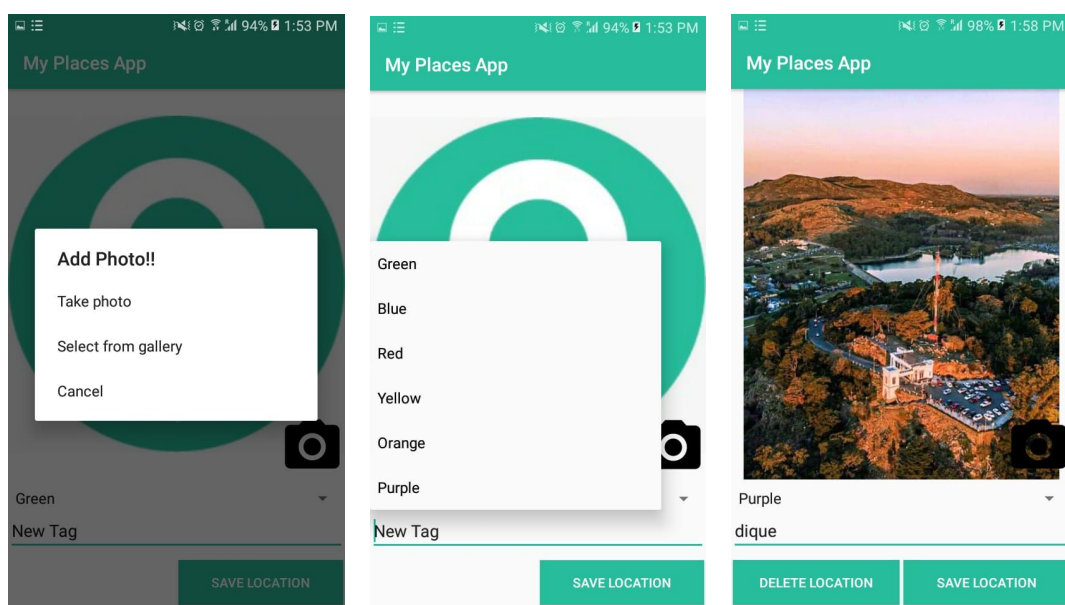
A partir de la SDK incluida se pudo hacer uso de las distintas funcionalidades para generar los distintos tipos de marcadores: pines, líneas y polígonos. Además se pudo definir el comportamiento que estos marcadores tomaban frente a distintos eventos (como ser clickeados).



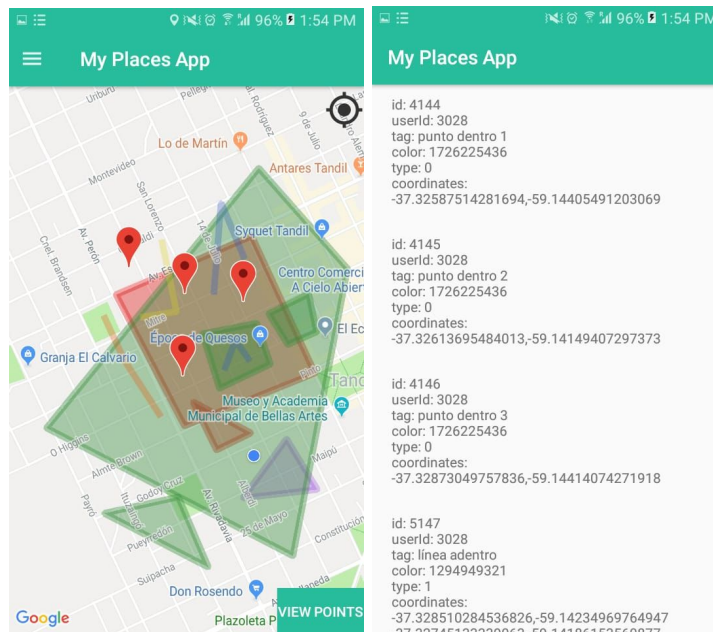


Para poder seleccionar y elegir entre cual de los distintos marcadores agregar al mapa se definió un menú lateral. A su vez, dentro de la vista principal se agregó un botón que redirige la vista hacia la ubicación actual del usuario. Para lograr esto, no solo es necesario que le brinde permisos a la aplicación de acceder a su ubicación, sino que también que su ubicación esté encendida.

Por otro lado, existe la posibilidad de editar los marcadores existentes al seleccionarlos. Se pueden editar tanto su etiqueta, como su color e imagen. Para lograr agregar una imagen fue necesario agregar cierta funcionalidad que permitiera tomar una foto o seleccionarla desde la galería. Un problema que se enfrentó al momento de querer agregar esto fue el gran tamaño de las imágenes, lo cual se solucionó al comprimir la imagen en formato JPEG y con una calidad del 50%. A su vez, también se optó traducir la imagen a un String Base64 para luego guardarla en la base de datos. Para poder acceder a las imágenes de la galería, es necesario que el usuario brinde permisos de almacenamiento.



Por otro lado, dentro del menú también se encuentra una opción denominada “Get Locations is Area” la cual permite definir un polígono y, al confirmar se puede observar los marcadores que el área definida por este contiene.



Para consumir los servicios brindados por el backend fue utilizado el framework Retrofit, el cual brinda un cliente REST que permite acceder a los servicios.

La estructura de uso de Retrofit requirió la implementación de una interfaz que definiera los diferentes métodos HTTP brindados por el backend, junto con su path, headers y body en caso de ser necesarios.

A su vez fue necesario definir ciertos DataTransferObjects tales como: ImageDTO, UserDTO y LocationDTO, que implementan la interfaz serializable y los cuales fueron utilizados tanto para ser enviados en los cuerpos de los request, como para contener la información del cuerpo del response del servicio.

El backend fue implementado en el lenguaje C#, haciendo uso del framework de Microsoft “.NET Core”, el cual provee un conjunto de librerías y soluciones predefinidas que facilitan el desarrollo de aplicaciones web. Este mismo aporta un gestor de paquetes gratuito llamado Nugget que facilita el acceso a librerías

proporcionadas por Microsoft para acelerar y facilitar el desarrollo de los proyectos.

Entre los servicios que provee el Backend se encuentran:

- Creación de un usuario.
- Servicio de Login.
- Creación de un marcador.
- Modificación de marcador.
- Eliminación de un marcador.
- Recuperar todas ubicaciones del usuario.
- Obtener una ubicación dado su Id.
- Guardar una Imagen.
- Obtener una imagen dado su Id.
- Dada un área obtener los marcadores del usuario que se encuentran dentro.

El backend cuenta con servicios que no requieren autorización tales como la inscripción de un nuevo usuario y el "login" de un usuario, mientras que otros servicios se encuentran "protegidos".

El servicio de login retorna un Java Web Token (JWT) el cual, además de proveer acceso a dichos servicios protegidos, posee ciertas "Claims" que contienen información del usuario y acerca del mismo token.

Para acceder a la base de datos fue utilizado "Entity Framework Core", dicha herramienta funciona como un ORM, permitiendo trabajar con objetos o "entidades" que representan las diferentes tablas de la base de datos. De esta forma, se evita tratar con sentencias SQL dentro del código, sino que se utilizan sentencias LINQ (Language Integrated Query).

Por otro lado para abstraer lo referido a la capa de datos de la lógica de los servicios se optó por utilizar el patrón "Repository" que emplea interfaces para definir los métodos de acceso a la base de datos.

Entre las ventajas que conlleva este patrón se encuentran:

- Abstracción del tipo de base de datos de las capas de negocio.

- Facilita la testeabilidad, ya que se se pueden generar Mocks de las interfaces.
- Posibilidad de reutilización de las interfaces en otras capas.

Para ir almacenando la información fue utilizada una base de datos relacional (SQL Server). En primer lugar existe la tabla Usuario, que tal como su nombre lo define contiene los datos de un usuario. De la misma forma se estableció la tabla Location y, a su vez se decidió definir una tabla Images a parte, la cual se encuentra separada de la tabla de Location con el objetivo de alivianar el contenido de esta y, de esta forma facilitar también los métodos que acceden a estas tablas. El lograr una cierta independencia entre estas tablas posibilita que en un futuro, se logre almacenar más información acerca de las imágenes sin que se encuentre asociada a la ubicación solamente.

La base de datos corre dentro de un contenedor de Docker, haciendo uso de una imagen de SQL Server descargada de Docker Hub.

Para asegurar la persistencia de la información en dicha base de datos fueron utilizados los “volúmenes” provistos por docker, gracias a estos, al reiniciar la base de datos por cualquier motivo, ésta recupera información existente en dicho volumen.

Esta puede ser iniciada utilizando el comando “docker-compose up -d” desde la carpeta contenedora del Docker-compose.yml.

Conclusión

Para concluir, la realización de este trabajo fue muy útil para en cierta manera integrar lo visto en varias cátedras, ya que incluye conceptos de la programación orientada a objetos tales como abstracción de comportamientos, la utilización de interfaces y clases abstractas, entre otros. Por otro lado también se utilizaron conceptos asociados al desarrollo de aplicaciones móviles para lograr desarrollar el frontend de la mejor manera, utilizando activities, intents, definiendo permisos del dispositivo y el consumo de la API Rest.

Finalmente el desarrollo del Backend de la aplicación permitió aplicar ciertos conocimientos obtenidos en el Taller de Desarrollo Web tales como el desarrollo de servicios RestFul capaces de ser accedidos independientemente de la aplicación que los consuma.

Instrucciones de uso

1. Desde la carpeta BackendMyPlacesApp ejecutar en la terminal el siguiente comando: "docker-compose up -d". Si no hubo problema, debería aparecer el mensaje "database ready".
2. Correr los scripts adjuntos de creación de tablas.
3. Abrir el proyecto WebApiObjetos desde VisualStudio y correrlo. Si se desea modificar el puerto donde se publican los servicios se debe modificar el archivo launchSettings.json dentro de Properties.
La duración del token de validación y la clave de encriptado del token se encuentran en Resources.resx para ser modificadas(actualmente dura 1 hora el token).

El puerto al que se conecta a la base y el usuario y contraseña para la misma se encuentran en appsettings.json

Usuario:sa

Contraseña:Passw0rd@@


4. Para lograr que acceder a los servicios desde el celular seguir las siguientes instrucciones:
(<https://developers.google.com/web/tools/chrome-devtools/remote-debugging/local-server>)

La Url Base que utiliza retrofit para acceder a los servicios puede ser cambiada desde Strings.xml .

5. Abrir el proyecto desde AndroidStudio y con el dispositivo conectado por usb, ejecutarlo.

Scripts de creación de las tablas de la base (autogenerados):

Users:


	Column Name	Data Type	Allow Nulls
	Id	int	<input type="checkbox"/>
	UserName	nvarchar(MAX)	<input type="checkbox"/>
	Password	nvarchar(MAX)	<input type="checkbox"/>
	Email	nvarchar(MAX)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

```

USE [LocationsDB]
GO
/***** Object: Table [dbo].[Users]  Script Date: 20/08/2019 15:25:04 *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Users](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [UserName] [nvarchar](max) NOT NULL,
    [Password] [nvarchar](max) NOT NULL,
    [Email] [nvarchar](max) NULL,
    CONSTRAINT [PK_Users] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

```

Locations:

	Column Name	Data Type	Allow Nulls
	Id	int	<input type="checkbox"/>
	UserId	int	<input type="checkbox"/>
	Tag	varchar(50)	<input type="checkbox"/>
	Color	int	<input type="checkbox"/>
	Type	smallint	<input type="checkbox"/>
	Coordinates	varchar(MAX)	<input type="checkbox"/>
	ImageId	int	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

```

USE [LocationsDB]
GO

/***** Object: Table [dbo].[Locations]  Script Date: 20/08/2019 15:25:13 *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

```

```

CREATE TABLE [dbo].[Locations](
    [Id] [int] IDENTITY(100,1) NOT NULL,
    [UserId] [int] NOT NULL,
    [Tag] [varchar](50) NOT NULL,
    [Color] [int] NOT NULL,
    [Type] [smallint] NOT NULL,
    [Coordinates] [varchar](max) NOT NULL,
    [ImageId] [int] NULL,
CONSTRAINT [PK_Locations] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

```

```

ALTER TABLE [dbo].[Locations] WITH CHECK ADD CONSTRAINT [FK_Locations_Images] FOREIGN
KEY([ImageId])
REFERENCES [dbo].[Images] ([Id])
GO

```

```

ALTER TABLE [dbo].[Locations] CHECK CONSTRAINT [FK_Locations_Images]
GO

```

Images:

	Column Name	Data Type	Allow Nulls
PK	Id	int	<input type="checkbox"/>
	UserId	int	<input type="checkbox"/>
	Picture	nvarchar(MAX)	<input type="checkbox"/>
			<input type="checkbox"/>

```

USE [LocationsDB]
GO
/***** Object: Table [dbo].[Images] Script Date: 20/08/2019 15:22:52 *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

```

```
CREATE TABLE [dbo].[Images](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [UserId] [int] NOT NULL,
    [Picture] [nvarchar](max) NOT NULL,
    CONSTRAINT [PK_Images] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE [dbo].[Images] WITH CHECK ADD CONSTRAINT [FK_Images_Users] FOREIGN KEY([UserId])
REFERENCES [dbo].[Users] ([Id])
GO

ALTER TABLE [dbo].[Images] CHECK CONSTRAINT [FK_Images_Users]
GO
```