RÍO NEGRO
UNIVERSIDAD NACIONAL

Licenciatura en Sistemas

IMPLEMENTACIÓN DE SOLUCIÓN MÓVIL PARA INTEGRAR A SECTORES SOCIALES EXCLUIDOS DEL MERCADO LABORAL

Alumno: Difabio Guillermo Alejandro
Director: Ing. Mauro Cambarieri

mcambarieri@unrn.edu.ar

Año: 2017

Agradecimientos

Mediante estas líneas quiero agradecer enormemente a cada una de las personas que colaboraron desde distintos lugares a lograr el presente trabajo, principalmente al director de esta tesina Ing. Mauro Cambarieri, quien supo brindarme todos sus conocimientos y experiencias logrando guiarme y motivarme para cumplir con eficiencia el objetivo final. Un agradecimiento muy especial a mis padres quienes desde un principio me inculcaron la importancia de esforzarse por lo que uno desea, a mis hermanos con los que comparto la pasión que genera esta profesión, quienes además fueron pilares fundamentales a lo largo de la carrera. Otro muy especial merece la comprensión, paciencia y el ánimo recibidos por mi compañera de vida quien me acompaño en el proceso de desarrollo de la tesina. Y por último quisiera hacer extensiva mi gratitud a mis compañeros del LIA y a todos los profesores que acompañaron este proceso de aprendizaje. A todos ellos, muchas gracias.

Índice

I. Introducción

Resumen

Este trabajo final presenta el desarrollo de una aplicación móvil con la función de ser un instrumento de intermediación en el mercado de trabajo.

Se trata de brindar a la población en general y a los colectivos más desfavorecidos en particular, como es el caso de los beneficiarios de los diferentes programas sociales que impulsa el estado, el acceso a una aplicación Android capaz de reducir la exclusión digital y mejorar la inserción de estos en el mercado de trabajo. La misma permitirá registrar y geolocalizar los usuarios que ofrezcan sus servicios de oficios de manera eventual y establecerá un lugar de encuentro con los respectivos demandantes, acercando a las partes sin participación de terceros, generando vínculos directos con gratuidad, eficiencia e inmediatez.

También aquí se presentan los estándares para el desarrollo y las tecnologías necesarias para su implementación. La construcción de un API REST[1] como arquitectura, hace que dicha aplicación sea totalmente escalable, brindando servicios que pueden ser usados por cualquier dispositivo o cliente que entienda HTTP, permitiendo en un futuro la implementación en otras plataformas.

Palabras Claves

Smartphones, E- Commerce, Android, Geolocalización, API RESTful, FCM, Java.

Un término que cada vez es más utilizado en el desarrollo del comercio electrónico es ‘Marketplace’. En la actualidad este modelo de negocio se encuentra en constante aumento, viene a ser el equivalente al mercado de toda la vida, donde cualquiera puede ir a vender sus productos y a comprar los del resto. Este concepto

[1] REST acrónimo en inglés de Representation State Transfer (traducción al español de Transferencia de Estado Representacional), interfaz entre sistemas que usan HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON.

tan sencillo se ha llevado en nuestra época, a Internet y a las opciones online, así, los fabricantes o proveedores suben sus productos o servicios para ofrecer online en el Marketplace, donde cualquier usuario puede acceder a ellos, comprarlos o pedirlos. A partir de este intercambio virtual nace un nuevo concepto llamado 'Economía Colaborativa', un sistema económico en el que se comparten e intercambian bienes y servicios a través de plataformas digitales, desde choferes privados y tintorerías hasta reservar alojamiento en viviendas privadas. Ejemplos de este tipo de App son Uber, una opción digital que conecta a pasajeros con conductores privados y Airbnb que permite publicar, descubrir y reservar viviendas privadas. Desde luego que este sistema no es nuevo en la economía, los intermediarios han existido desde los orígenes del capitalismo global, la diferencia está en la eficiencia con la que, por medio de complejos algoritmos, los sistemas de información brindan seguridad a la hora de acercar a las partes. De esta manera dichas tecnologías, pasaron a cumplir el rol de intermediarias y los usuarios podrán proveer bienes y servicios sin depender de un empleador.

II. Estado de la cuestión

II.1. Portales de trabajo

Existen en la actualidad sitios web donde se puede encontrar ofertas y demandas de trabajo, algunos de los más conocidos son:

Bumeran, el portal de empleos y reclutamiento online líder en Latinoamérica. ZonaJobs, la bolsa de empleo líder en Argentina, donde se pueden encontrar ofertas de empleo y oportunidades de trabajo en un área profesional. CompuTrabajo, es la web de empleo líder en Latinoamérica y ofrece servicios similares que ZonaJobs. Empleos, pertenece al periódico clarín y se encarga de resolver la demanda de recursos humanos e implementar la tercerización de servicios en distintas tareas a medida de cada cliente que solicita el servicio. Evalúan las capacidades de miles de trabajadores eventuales y efectivos para asegurar el personal y la administración adecuada para cada empresa.

También existen redes sociales de contactos profesionales, que permiten mantener un Currículum Vitae online, como son Linkedin, Xing, Viadeo, Plaxo, entre otros. Una de las más conocida es Linkedin, una red social que muestra un perfil laboral para aquellos que buscan conseguir un trabajo o bien una herramienta indispensable para los reclutadores que buscan recursos humanos. El usuario a parte de subir su foto y datos de contacto, puede especificar sus trabajos actuales, trabajos anteriores, experiencias, un breve extracto, intereses, cursos realizados, que títulos posee, donde estudió y mucho más.

Sin lugar a duda tanto las redes sociales como los portales mencionados anteriormente, ocupan un lugar imprescindible a la hora de que surjan nuevas oportunidades laborales, pero estos como el resto de las opciones del mercado de servicios públicos de empleo, dejan fuera de juego personas que hoy en día están excluidos del sistema por no calificar para los requisitos de reclutamiento formal.

II.2. Mercado de trabajo

El Instituto Nacional de Estadísticas y Censos (Indec) establece como principales indicadores del mercado de trabajo al Empleo y Desempleo, los cuales permiten hacer un seguimiento de la evolución de las tasas de actividad, empleo, desocupación y subocupación. Este organismo difundió las cifras de la Encuesta Permanente de Hogares (EPH) correspondiente al primer trimestre del año 2017.

Área geográfica	Tasas de						
	Actividad	Empleo	Desocupación	Ocupados demandantes de empleo	Subocupación	Subocupación demandante	Subocupación no demandante
Total 31 aglomerados urbanos	**45,5**	**41,3**	**9,2**	**14,1**	**9,9**	**6,6**	**3,3**
Aglomerados del interior	**43,1**	**40,0**	**7,0**	**12,8**	**8,7**	**6,5**	**2,2**
Regiones							
Gran Buenos Aires	**47,6**	**42,4**	**10,9**	**15,1**	**10,9**	**6,8**	**4,2**
Ciudad Autónoma de Buenos Aires (1)	55,8	51,4	7,9	12,8	7,8	4,5	3,2
Partidos del Gran Buenos Aires (1) (5)	45,6	40,2	11,8	15,8	11,9	7,4	4,4
Cuyo	**40,7**	**38,8**	**4,7**	**10,9**	**7,3**	**6,5**	**0,9**
Gran Mendoza (1) (5)	42,6	40,6	4,7	8,8	7,5	6,7	0,8
Gran San Juan (2) (5)	38,3	36,0	5,9	15,2	8,3	7,0	1,4
Gran San Luis (2) (3)	38,3	37,4	2,3	11,2	4,2	3,9	0,3
Noreste	**37,8**	**36,6**	**3,1**	**6,0**	**4,7**	**3,6**	**1,1**
Corrientes (2)	41,7	40,1	3,9	4,6	2,9	2,7	0,1
Formosa (2)	31,8	30,9	2,9	5,2	3,9	2,3	1,6
Gran Resistencia (2) (5)	35,1	34,7	1,4	5,7	4,8	3,7	1,1
Posadas (2)	40,7	39,0	4,1	8,3	7,0	5,0	2,0
Noroeste	**42,6**	**39,8**	**6,5**	**16,0**	**10,8**	**8,1**	**2,8**
Gran Catamarca (2)	46,4	41,4	10,8	13,4	3,0	2,4	0,6
Gran Tucumán-Tafí Viejo (1)	42,0	38,8	7,7	19,6	13,8	11,9	1,9
Jujuy-Palpalá (2)	44,0	42,1	4,2	17,0	15,5	10,0	5,5
La Rioja (2) (5)	42,5	40,5	4,8	8,5	6,4	5,6	0,8
Salta (1)	46,1	42,7	7,3	17,8	11,3	6,7	4,7
Santiago del Estero-La Banda (2) (5)	34,9	34,0	2,5	8,1	5,2	4,2	1,0
Pampeana	**45,3**	**41,3**	**8,8**	**14,0**	**9,6**	**6,9**	**2,7**
Bahía Blanca-Cerri (2)	46,2	42,6	7,6	8,3	5,6	4,3	1,4
Concordia (2) (4)	39,6	36,5	7,9	6,9	6,5	5,2	1,3
Gran Córdoba (1)	45,3	41,0	9,6	18,6	10,5	7,9	2,6
Gran La Plata (1)	45,4	41,8	8,0	13,1	11,7	9,0	2,7
Gran Rosario (1)	47,8	42,9	10,3	14,7	10,8	7,3	3,5
Gran Paraná (2) (4)	43,5	41,6	4,4	7,6	3,4	3,1	0,3
Gran Santa Fe (1)	41,8	39,7	5,2	5,6	7,0	5,0	2,0
Mar del Plata (1) (3)	45,9	41,1	10,4	19,7	11,7	7,9	3,9
Río Cuarto (2)	44,3	40,3	9,1	8,2	4,6	2,4	2,1
Santa Rosa-Toay (2)	42,5	38,9	8,6	9,0	9,4	7,1	2,3
San Nicolás-Villa Constitución (2) (5)	42,3	39,5	6,6	4,8	2,6	2,2	0,5
Patagónica	**42,0**	**39,7**	**5,5**	**7,7**	**4,0**	**3,3**	**0,8**
Comodoro Rivadavia-Rada Tilly (2)	39,4	37,3	5,3	9,8	3,9	3,2	0,8
Neuquén-Plottier (2)	41,4	39,4	5,0	6,2	3,2	2,8	0,5
Río Gallegos (2)	44,5	43,0	3,3	4,0	1,7	1,3	0,4
Ushuaia-Río Grande (2)	44,0	40,6	7,7	8,1	4,9	3,8	1,1
Rawson-Trelew (2) (5)	45,9	42,2	8,2	13,2	7,7	6,2	1,5
Viedma-Carmen de Patagones (2)	37,5	37,1	1,1	2,4	2,1	1,7	0,4
Total aglomerados de 500.000 y más habitantes	**46,7**	**42,0**	**10,1**	**15,2**	**10,8**	**7,1**	**3,7**
Total aglomerados de menos de 500.000 habitantes	**40,6**	**38,5**	**5,2**	**9,0**	**5,7**	**4,4**	**1,3**

Fig.II.1 - Principales indicadores por áreas geográficas. Primer trimestre de 2017.

La Encuesta Permanente de Hogares Continua es un programa nacional cuyo propósito es el relevamiento sistemático y permanente de los datos referidos a las características demográficas y socioeconómicas fundamentales de la población, vinculadas a la fuerza de trabajo. Su temática está orientada hacia la caracterización de la situación social integral de los individuos y los hogares, aunque los datos más difundidos son los relacionados con el mercado laboral.

Como se puede observar en la figura II.1, la desocupación en Argentina se ubicó en el 9,2 por ciento en el primer trimestre del año 2017, lo que representa 1.658.000 personas. Y observando los datos correspondientes a los distintos aglomerados urbanos del país, el nivel de desempleo en Viedma-Carmen de Patagones es de un 1,1 por ciento equivalente a 18.238 habitantes.

En la actualidad los gobiernos vienen desarrollando diferentes estrategias a nivel del empleo, que buscan contrarrestar esta brecha económica, atendiendo específicamente a la población joven.

A nivel nacional, se han creado dos programas, por un lado, el Programa Jóvenes con Más y Mejor Trabajo (PJMyMT), dependiente del Ministerio de Trabajo, Empleo y Seguridad Social de la Nación y, por otro lado, el Programa de Respaldo a Estudiantes Argentinos (PROG.R.ES.AR), dependiente de la ANSES, cuyos objetivos son generar oportunidades de inclusión social y laboral para jóvenes de dieciocho (18) a veinticuatro (24) años de edad, en situación de desempleo y con estudios primarios o secundarios incompletos.

A nivel provincial en Río Negro, se ha implementado recientemente a través del Ministerio de Desarrollo Social, el Programa “Emprender”, que tiene como objetivo fortalecer las capacidades productivas y de autogestión, habilidades o destrezas de las personas, para un mejor posicionamiento en el mercado laboral y productivo, según las necesidades del desarrollo local, articulando acciones con municipios, organizaciones comunitarias y juntas vecinales. Actualmente tiene una proyección anual de más de 600 talleres en toda la provincia, lo que referencia la participación de unas 11.000 personas que se capacitarán en distintos oficios. En el marco de dicho programa se estableció en la Ciudad de Viedma (Rio Negro), un convenio entre el Ministerio de Desarrollo Social y el Consejo Municipal de la Mujer, para

poner en marcha 14 cursos de formación laboral a través de las juntas vecinales en distintos barrios de la ciudad. Entre ellos se destacan, repostería, panificación y derivados, costura básica, peluquería, inglés y auxiliar de cuidador domiciliario.
Actualmente en la ciudad se llevan a cabo más de 90 talleres sobre un total de 600 previstos en toda la provincia.
Además, como todos los años, la Municipalidad de Viedma abre las inscripciones para los talleres de oficios que se dictan en las juntas vecinales de diferentes barrios de la Ciudad. En esta oportunidad, nueve talleres formarán parte del Programa Municipal “Quiero un Oficio”, con la posibilidad de desarrollar una capacitación profesional o micro emprendimiento, a través de trabajos manuales. Los talleres que se llevarán a cabo son Albañilería, Electricidad, Durlock (construcción en seco), Peluquería, Decoración de Tortas, Panadería, Cocina de Restaurante y Viandas, Corte y Confección y Diseño de Indumentaria.
Cada uno de estos programas dispone de un conjunto de prestaciones integradas y de apoyo a la construcción e implementación de un proyecto formativo y ocupacional, con el objetivo de proporcionarles oportunidades para desarrollar trayectorias laborales pertinentes y de calidad, adecuadas a sus perfiles, sus expectativas y sus entornos.

II.3. Dispositivos móviles

Los teléfonos inteligentes, también llamados smartphone, se han convertido en dispositivos imprescindibles para la sociedad actual, gracias a la evolución de tres factores, hardware, software e Internet. Esto se vincula con los beneficios que Internet ofrece (acceso a noticias, entretenimiento, videojuegos y redes sociales). Este servicio y los teléfonos Inteligentes logran una sinergia que día a día amplía la frontera de lo posible.

Los smartphones más destacados en la actualidad son iPhone, Android y Windows Phone. Dichas plataformas hacen uso eficientemente de todos los componentes de hardware que actualmente brindan los móviles como: pantallas táctiles (paneles LCD, IPS o AMOLED), cámara de foto y video, sensores embebidos (acelerómetros, giroscopio, luz), conectividad de corto alcance (Bluetooth y NFC) y largo alcance

(WIFI, 3G y 4G) entre los componentes más destacados. Al igual que los PCs que utilizan Windows, macOS (OSx), Linux, entre otros, los dispositivos móviles tienen sus sistemas operativos (SO) como Windows Phone, iOS, Android, etc.

Apple desarrollo iOS, sistema operativo basado en OSx, que da vida a dispositivos como el iPhone, el iPad, el iPod Touch o el Apple TV. Una de las limitaciones que presenta es que no permite la instalación de iOS en hardware de terceros. En las primeras versiones el código de desarrollo era en Object-C, luego se presentó en WWDC[2] 2014 Swift 3.0, un lenguaje de código abierto. Actualmente su sistema operativo se encuentra en la décima versión, mejor conocida como iOS 10.

Otro sistema operativo es Android, basado en Linux, fue creado por Android Inc. diseñado originalmente para cámaras fotográficas profesionales. Se fundó en 2003, luego fue comprada por Google en el 2005 y en 2007 fue lanzado al mercado con modificaciones para ser utilizado en dispositivos móviles como los teléfonos inteligentes y en tabletas como es el caso del Galaxy Tab de Samsung. Actualmente se encuentra en desarrollo para usarse en netbooks y PCs.

Las aplicaciones para Android se escriben y desarrollan en Java, aunque también se puede trabajar en C++. Una de las grandes características de este sistema operativo es su carácter abierto, para ello se distribuye bajo dos tipos de licencias, una que abarca todo el código del Kernel, que es GNU GPLv2 y la otra para el resto de componentes del sistema que se licencia bajo APACHE v2. En la actualidad los últimos modelos de Samsung y Motorola ofrecen la versión 6.0, código de nombre Marshmallow y desde la página oficial de Android se habla del lanzamiento del LG V20, siendo el primer smartphone con Android 7.0 Nougat integrado.

Por último, Windows Phone es un sistema operativo móvil desarrollado por Microsoft, como sucesor de Windows Mobile. Fue el primero en intentar unificar este SO para todas las plataformas, tanto que en enero de 2015 se dio de baja a Windows Phone, para enfocarse en un único sistema más versátil denominado Windows 10 Mobile, disponible para todo tipo de plataformas (teléfonos inteligentes, tabletas y computadoras). En la actualidad es la plataforma de escritorio más

[2] WWDC acrónimo en inglés de Worldwide Developers Conference (traducción al español de Conferencia Mundial de Desarrolladores de Apple).

popular en Argentina sin lugar a duda, pero los dispositivos móviles de la empresa no logran ser significativos respecto a Android y iPhone.

En la figura II.2 se puede ver la distribución de plataformas por SO. En el mercado de sistema operativos de teléfono inteligente, Android tiene una amplia ventaja con el 81,72 % del mercado total en el cuarto trimestre de 2016, seguido de iOS con el 17,85 %. En el año 2016 en general, Android también aumentó su cuota de mercado en 3,2 puntos porcentuales y fue el único sistema operativo que no paró de crecer año tras año.

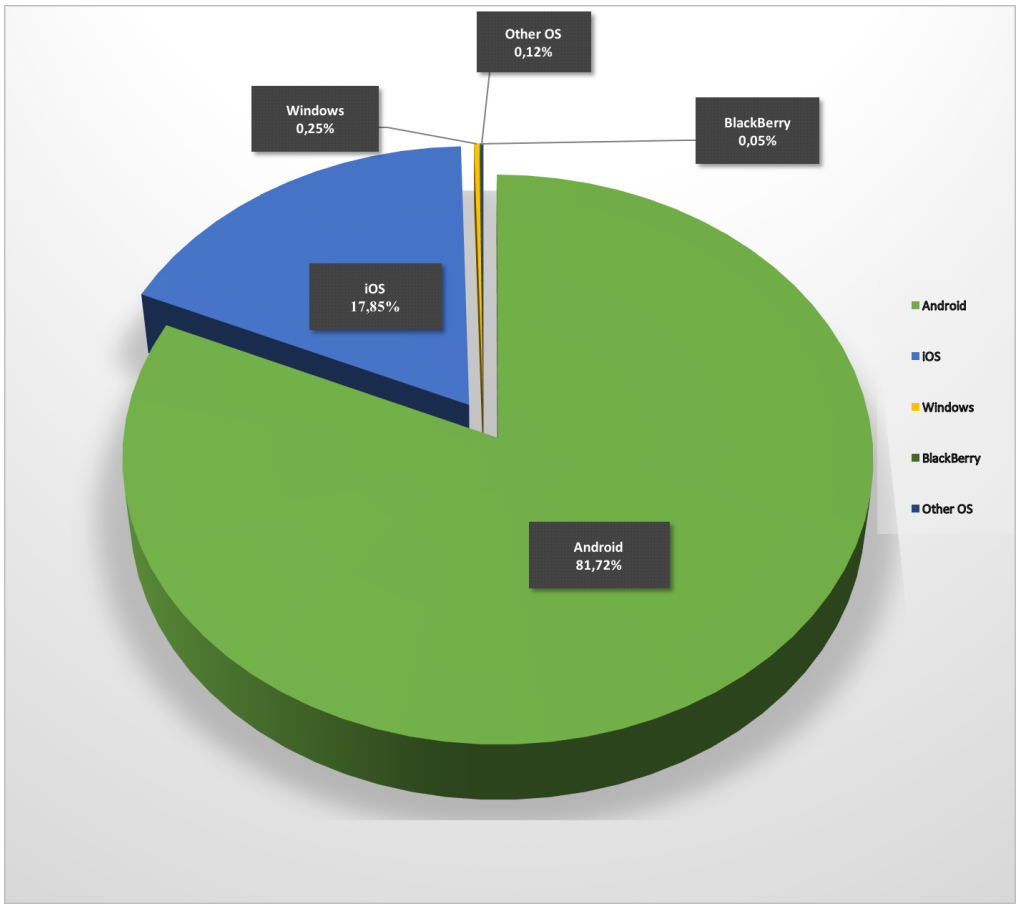


Fig.II.2 - Cuota de Mercado Mundial de Smartphones (Gartner febrero de 2017).

Para implementar una aplicación en alguna de las plataformas nombradas, se debe hacer un desarrollo en el lenguaje, IDE[3] y SDK[4] específico de cada SO. Por

[3] IDE acrónimo en inglés Integrated development environment (traducción al español de entorno de desarrollo integrado).

[4] SDK acrónimo en inglés Software Development Kit (traducción al español de kit de desarrollo de software).

ejemplo, si se quiere desarrollar en iPhone, se necesita el entorno Xcode sobre plataforma xOS. Para Windows phone se utiliza Visual Studio en Plataforma Windows y para Android se pueden utilizar todos los SO antes mencionados y también cualquier distribución de linux con el IDE Android Studio. En este punto Android tiene una ventaja que no restringe el SO del desarrollador y permite utilizar código abierto.

El mundo de las aplicaciones móviles está en pleno auge, tal es así que poco a poco están desplazando el desarrollo del software tradicional. Las ventajas de desarrollar una app móvil sobre un software de escritorio están claras, los móviles son una herramienta habitual (alta disponibilidad), tienen un gran alcance al mercado a través de las diferentes tiendas (Play Store, App Store, Microsoft Store, etc.), también cuentan con accesorios mucho más versátiles que un PC (GPS, cámara, altavoz, micrófono, etc.) y sobre todo, con una conexión de datos que permite tener un gran alcance y disponibilidad de Internet.

Cuando se habla del desarrollo de estas aplicaciones, se pueden distinguir tres aproximaciones, como la técnica de desarrollo de apps nativas, la de desarrollo híbrido y las responsive webs.

II.3.1. Aplicación nativa

La aplicación nativa está desarrollada y optimizada específicamente para un sistema operativo determinado y la plataforma de desarrollo del fabricante (Android, iOS, Windows 10, etc.).

Este tipo de aplicaciones se adaptan al 100% con las funcionalidades y características del dispositivo obteniendo así una mejor experiencia de uso. Sin embargo, el desarrollo de una aplicación nativa implica un mayor costo, puesto que si se desea realizar una aplicación multiplataforma se deberá crear una nueva versión para cada sistema operativo, multiplicando así los costos de desarrollo.

II.3.2. Aplicación web

Existe la posibilidad de implementar una única solución que se pueda acceder en todos los SO móviles antes mencionados. Esta característica es posible mediante tecnología web en lo que se conoce como aplicaciones multiplataforma.

En este sentido HTML5[5] aporta muchos componentes para el desarrollo de aplicaciones móviles y un creciente número de frameworks[6] que permiten desarrollar aplicaciones móviles web cada vez más parecidas a las Nativas, no solo en lo estético sino también en rendimiento.

La aplicación web es la opción más sencilla y económica de crear un sistema móvil, puesto que al realizar un único desarrollo se reducen al máximo los costos de producción. La desventaja que presenta es que ofrece una peor experiencia de uso, puesto que ignora las características del dispositivo y una menor seguridad ya que depende de la que ofrezca el propio navegador.

II.3.3. Aplicación híbrida

Este tipo de aplicación aprovecha al máximo la versatilidad de un desarrollo web y tiene la capacidad de adaptación al dispositivo como una app nativa. Permite utilizar los estándares de desarrollo web (HTML5) y aprovechar algunas de las funcionalidades del dispositivo tales como la cámara, el GPS o los contactos. Además, implica un menor costo que una aplicación nativa y una mejor experiencia de uso que una aplicación web.

Sin embargo, tiene un rendimiento inferior al de una aplicación nativa debido a que cada página debe ser renderizada desde el servidor y supone una mayor dificultad de desarrollo.

II.4. Arquitectura de Software

La arquitectura de software conforma la columna vertebral de cualquier sistema y constituye uno de sus principales atributos de calidad.

Generalmente las arquitecturas de aplicaciones empresariales se componen de tres capas bien definidas: presentación, modelo de negocios y persistencia. Esta división permite que cada componente tenga su responsabilidad y esconda su lógica a las demás, permitiendo escalar de forma más sencilla y generando un producto más mantenible, otro factor de calidad.

[5] HTML (HyperText Markup Language) que significa Lenguaje de Marcado para Hipertextos.
[6] Framework, entorno de trabajo o marco de trabajo.

En la capa de presentación los objetos trabajan directamente con las interfaces de negocios, implementando el patrón Model View Controller. En este, el modelo (Model) es modificable por las funciones de negocio, siendo éstas solicitadas por el usuario, mediante el uso de un conjunto de vistas (View) que solicitan dichas funciones de negocio a través de un controlador (Controller), que es quien recibe las peticiones de las vistas y las procesa.

La capa de negocio está formada por servicios implementados por objetos de negocio. Estos delegan gran parte de su lógica en los modelos del dominio que se intercambian entre todas las capas. Finalmente, la capa de persistencia facilita el acceso a los datos y su almacenamiento en una base de datos. En Android este flujo de trabajo involucra Actividades y Fragmentos**.**

A veces se presentan tareas de interfaz, como mostrar un elemento, ocultarlo, etc. Pero para esto hay una lógica en donde se decide que hacer de acuerdo a un evento. Por lo tanto, hay que segmentar y especializar cada una de las capas, para ello se utilizaran los patrones de diseño Model View Presenter (MVP) y Clean.

El MVP es un patrón derivado del MVC comúnmente usado en la construcción de interfaces gráficas. En la aplicación Android el usuario va a interactuar con la vista, la cual no debería tener lógica en cuanto a lo que está mostrando y en general va a ser la actividad quien lo haga.

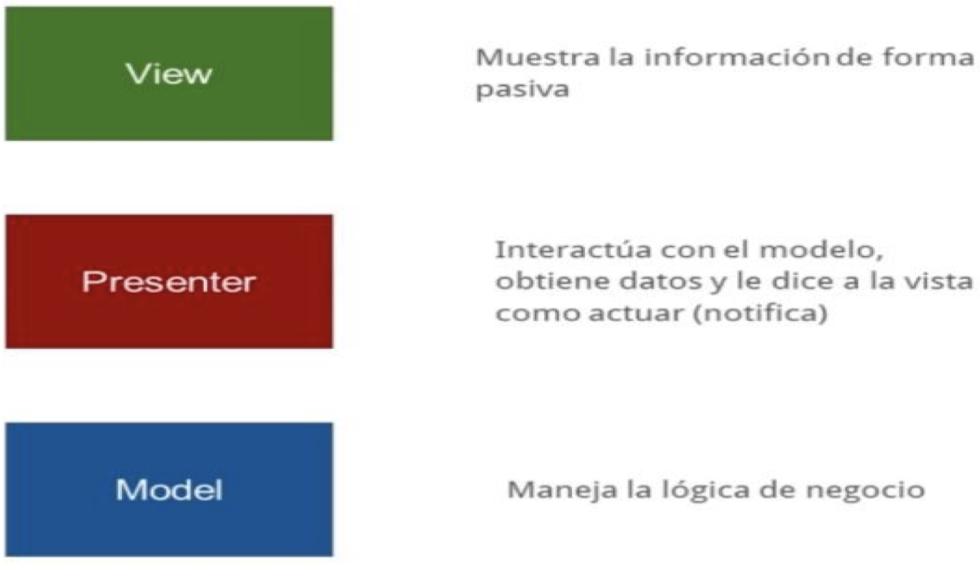


Fig.II.3 - Componentes del Modelo MVP.

La vista cuando necesite algo va a acudir al presentador, este le dará formato y accederá a métodos tanto de la vista como del modelo. Cuando el presentador necesite de estos datos los traerá al modelo, ya que este tiene la lógica del negocio. Una vez que obtenga los datos, enviará el resultado a la vista, para que esta publique los resultados.

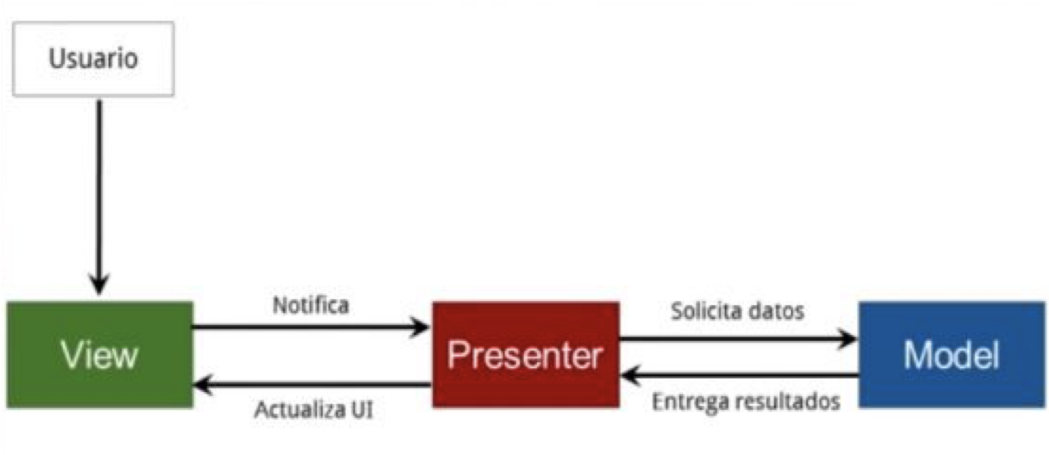


Fig.II.4 - Comunicación entre los componentes del Modelo MVP.

El Presentador va a tener una instancia de la vista y una instancia del modelo. Entonces al implementar el presentador, este, incluirá una instancia del modelo a través de la cual se ejecutará algún método para obtener datos que se puedan devolver a la vista. El modelo va a concluir y tendrá que reportarle de regreso al presentador, y este a la vista. Para esto se trabaja con un callback[7] que se implementará en el presentador y habrá una instancia en el modelo. Se reemplazará el callback por EventBus[8] y se usará para generar una comunicación entre el modelo y el presentador. El modelo emitirá un evento y el presentador lo capturará para luego realizar alguna acción sobre la vista.

El modelo además va a tener una instancia de esta interfaz, por ejemplo, un método Succes o Error que lo manda a llamar cuando sucede el evento correspondiente, el presentador implementa estos métodos y a través de ellos se comunica con la Vista.

La otra arquitectura llamada Clean, lo que busca es desacoplar, lo que permitirá tener una interfaz gráfica con la que el usuario va a interactuar, esta va a conectarse de alguna forma con un repositorio que tendrá los datos a través de un Interactor.

[7] En programación de computadoras, una devolución de llamada.
[8] EventBus es una biblioteca de código abierto para Android.

El repositorio eventualmente podría obtener datos utilizando la lógica de dominio o de una base de datos, haciendo uso de entidades que son los objetos que modelan el contenido, esto regresara hacia la interfaz para mostrarlo. Al combinar Clean con MVP se utilizará MVP para la parte frontal, es decir para la Vista y el Presentador, pero no habrá un modelo.

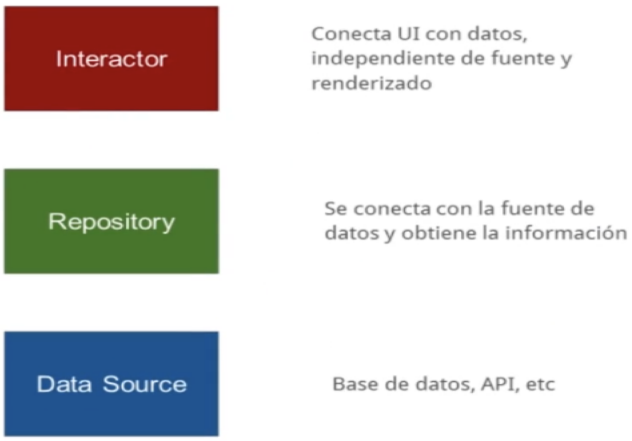


Fig.II.5 - Modelo combinando MVP y Clean.

En vez de tener un modelo donde toda la lógica de negocio esté acoplada, se hará uso de un Interactuador, que será invocado por el Presentador y a su vez este invocará al Repositorio. La ventaja de esto es que el interactuador no tiene que saber mucho de ninguno de los dos casos, solo que alguien lo va a llamar y que cuando alguien lo haga tiene que llamar a su vez al repositorio.

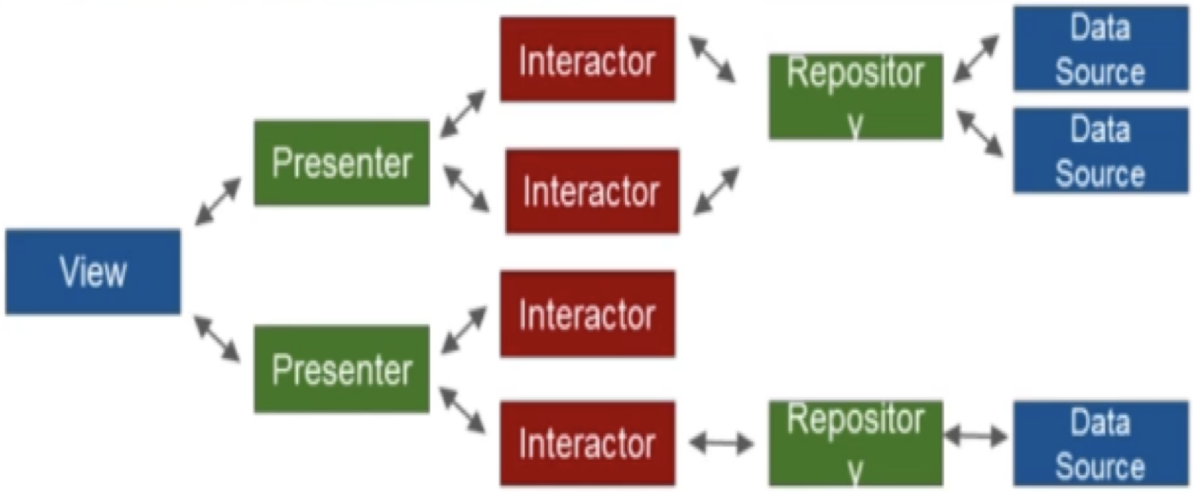


Fig.II.6 - Comunicación entre los componentes de MVP y Clean.

En el diagrama de la figura II.6 se puede observar una vista que se conecta con el presentador, el presentador podría tener más de un interactuador y este se comunica con el repositorio, que a su vez va con la fuente de Datos. El modelo ya no hará mucho, sino que delegará en el repositorio y la fuente de datos. La combinación de MPV y Clean dará buenos resultados a largo plazo, en mantenibilidad, testing, confiabilidad, etc.

II.4.1. Api REST

La necesidad de tener una comunicación en tiempo real entre servidor y clientes cada día cobra mayor importancia. Un recurso web es simplemente algo disponible en la Web, como puede ser una página web HTML tradicional, un documento, un archivo de audio y un archivo de imagen.

Representational State Transfer (REST), es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. Utiliza direcciones web (URI) para acceder y manipular recursos utilizando verbos (métodos de solicitud HTTP) dentro de los cuales se destacan POST, GET, PUT y DELETE, estas corresponden a operaciones de creación, lectura, actualización y eliminación respectivamente. Dichas operaciones se equiparán a las operaciones CRUD en bases de datos (CLAB en castellano: crear, leer, actualizar, borrar) que se requieren para la persistencia de datos.

Ventajas que ofrece REST para el desarrollo:

1. Separación entre el cliente y el servidor: El protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos, eso tiene algunas ventajas cuando se hacen desarrollos, por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.

2. Visibilidad, fiabilidad y escalabilidad: La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una

de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back, eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.

3. API REST: Siempre es independiente del tipo de plataformas o lenguajes, se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js,lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

II.4.2 Comunicación en tiempo real

Firebase Cloud Messaging (FCM) es la nueva versión del Google Cloud Messaging (GCM) bajo la marca Firebase. Se hereda la infraestructura central de GCM, con nuevos SDK[9] para que el desarrollo mensajería en la nube sea más fácil. Es una solución multiplataforma que permite enviar, de forma gratuita y segura, mensajes y notificaciones. Dentro de las ventajas que encontramos al utilizar FCM, se puede identificar, por ejemplo, la tarea de notificarle a una app cliente que hay un correo electrónico nuevo u otros datos están disponibles para la sincronización. También se puede utilizar para mensajería instantánea, donde un mensaje puede transferir una carga de hasta 4 KB a una app cliente.

[9] Un SDK (Software Development Kit), o kit de desarrollo de software, es un conjunto de herramientas que ayudan a la programación de aplicaciones para un entorno tecnológico particular

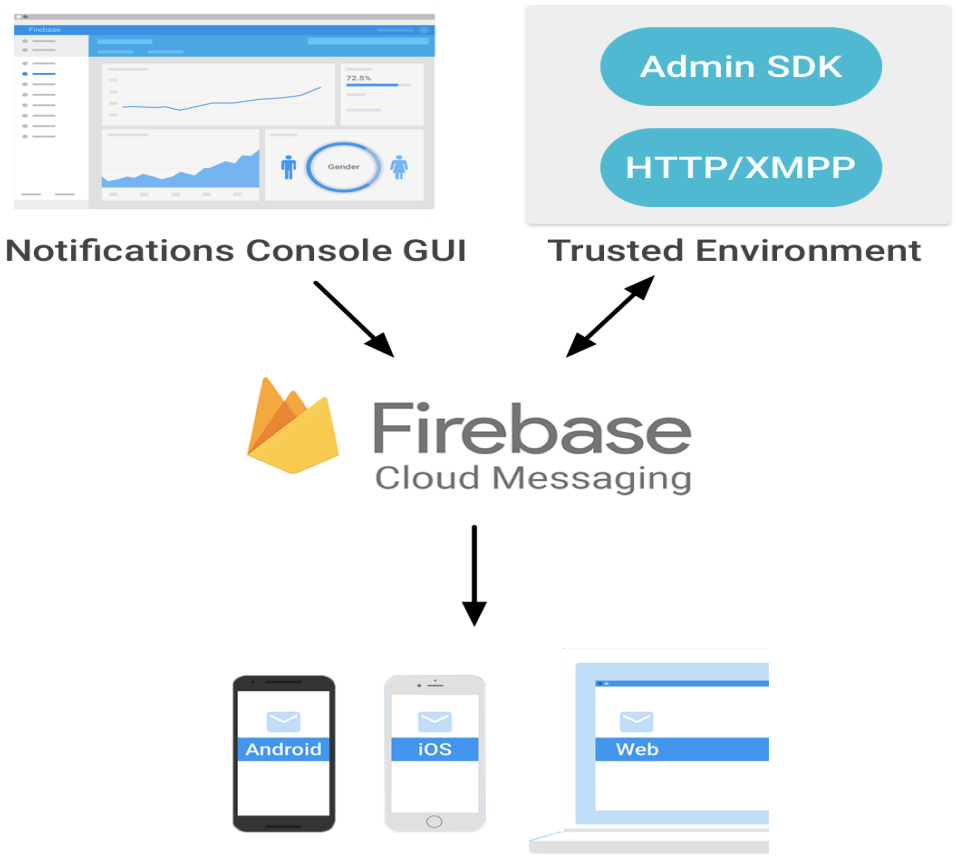


Fig.II.7 - Esquema de implementación Firebase Cloud Messaging.

Como se puede observar en la figura II.7, FCM proporciona una conexión fiable y eficiente entre el servidor y los dispositivos que permite enviar y recibir mensajes y notificaciones en Android, iOS y la web sin costo alguno. También se puede ver la implementación, que incluye un servidor de app en tu entorno que interactúa con FCM a través del protocolo de HTTP o XMPP y una app cliente. Además, incluye la consola de Notificaciones, que se usara para enviar notificaciones a las apps cliente.

Las notificaciones de Firebase están basadas en Firebase Cloud Messaging y comparten el mismo SDK para el desarrollo del cliente. Esta tecnología es de gran utilidad para el desarrollo de la aplicación, facilita el proceso de enviar mensajes de marketing, avisos de nuevas actualizaciones de software, notificaciones al usuario oferente de solicitudes de contratación, entre otras funciones. Además, permitirá implementar un chat donde el usuario demandante del servicio y el que realizó la publicación, puedan despejar dudas y hasta llegar a un acuerdo.

III. Problema a resolver e importancia de resolverlo

Si bien una gran cantidad de jóvenes se incorporan a los programas impulsados por las distintas políticas públicas que establece el estado y culminan diferentes prestaciones, el problema fundamental que persiste es la generación de instancias que posibiliten iniciar su carrera laboral.

Otro impedimento a la hora de salir al mercado de trabajo es el no poder calificar para los requisitos de reclutamiento formal requeridos habitualmente.

Hoy en día contar con un currículum vítae es fundamental, el mismo es un documento que sirve de presentación ya que recoge información de índole personal (datos biográficos, residencia), educativa (académica), formativa (profesional) y laboral (experiencia, habilidades y conocimientos), ofrece la primer imagen de uno y se entrega al contratista o a la empresa donde se desea ingresar a trabajar. Si el mismo es seleccionado habitualmente se pasa a una entrevista de trabajo, la cual consta de un diálogo que se sostiene entre el postulante y el contratista o en el caso de ser una empresa con los representantes de recursos humanos de la misma.

Partiendo de esta necesidad social de generar la inserción laboral y de la metodología utilizada en las aplicaciones descritas en el capítulo anterior, es que se plantea una plataforma que permita el intercambio de servicios entre usuarios demandantes y beneficiarios de los programas impulsados por las distintas políticas públicas que establece el estado, como son el Programa Jóvenes con Más y Mejor Trabajo (PJMyMT), dependiente del Ministerio de Trabajo, Empleo y Seguridad Social de la Nación y el Programa de Respaldo a Estudiantes Argentinos (PROG.R.ES.AR), dependiente del ANSES.

Se implementará una articulación de los módulos de alfabetización digital dirigidos a los inscriptos de los programas descriptos anteriormente, con la capacitación sobre la aplicación digital desarrollada, a fin de aprovechar el espacio de formación para capacitarlos en temas referidos al uso de Internet, como herramienta que permita el ofrecimiento de los servicios para los integrantes del curso.

Los usuarios de la aplicación buscarán llamar la atención de los demandantes intentando destacarse por sobre los demás postulantes, y eso dependerá del grado de completitud que presente su CV. Dentro de dicha app, se brindarán las herramientas necesarias para que la carga del mismo sea lo más sencilla posible, además este se nutrirá de las calificaciones recibidas sobre los servicios prestados. Las áreas que deberá incluir necesariamente son, datos personales, experiencia laboral y formación académica. Si se prefiere, se podrá agregar información adicional como idiomas, cursos de especialización y otras actividades. De esta manera los CV cargados en la aplicación permitirán a los usuarios lograr por una parte mayor visibilidad de sus conocimientos y experiencia en determinados oficios frente a los demandantes de los mismos y por otro lado incorporarlos a la Sociedad del Conocimiento a través de la construcción de sus identidades digitales.

III.1. Android 4.4 KitKat como solución

Gracias al gran avance de la tecnología móvil en la actualidad, se ha dispuesto como propósito del presente trabajo crear una aplicación basada en la plataforma Android para teléfonos inteligentes, ya que está lidera el mercado mundial de SO.

Es cierto que contar con una web móvil podría ser la solución, pero como se vio en el apartado anterior las aplicaciones móviles nativas permiten el aprovechamiento de las funcionalidades de los dispositivos, como acceder a los contactos, la cámara o la geolocalización y la mayoría de ellas no necesita una conexión a internet demasiado potente, a diferencia de lo que sí ocurre con las páginas web adaptadas a terminales móviles. Se decidió utilizar la plataforma Android, ya que además de ser la plataforma con mayor crecimiento del mercado, es menos restrictiva que otras plataformas nativas considerando la libertad que ofrece al desarrollar una aplicación, permitiendo que esté disponible para cualquier usuario y dispositivo físico. Lo cual resultó ser de utilidad para compartir y mostrar la aplicación en varios dispositivos y así poder ser evaluada por usuarios. Además, luego de investigar las posibilidades del desarrollo en Android, se llegó a la conclusión de que permite implementar todas las funcionalidades planteadas, que fueron necesarias para el desarrollo de esta aplicación, esto se podrá ver a lo largo del documento. Finalmente, una de las

razones que llevó en primer lugar a considerar la plataforma Android fue que ya se disponía de un dispositivo móvil con este sistema operativo, el cual sirvió para la realización de las diferentes pruebas de funcionalidad.

Al momento de comenzar el proyecto, el sistema Android había lanzado la versión 6.0 Marshmallow con nuevas funciones, como permisos en tiempo de ejecución, funciones de ahorro de energía, nueva tecnología de asistencia y otras características importantes.

En la figura III.1 se puede observar una captura de pantalla de la ayuda que brinda Android Studio a la hora de seleccionar la versión a desarrollar, la cual recomienda utilizar 4.4 KitKat abarcando el 73,9 % de la distribución de dispositivos.

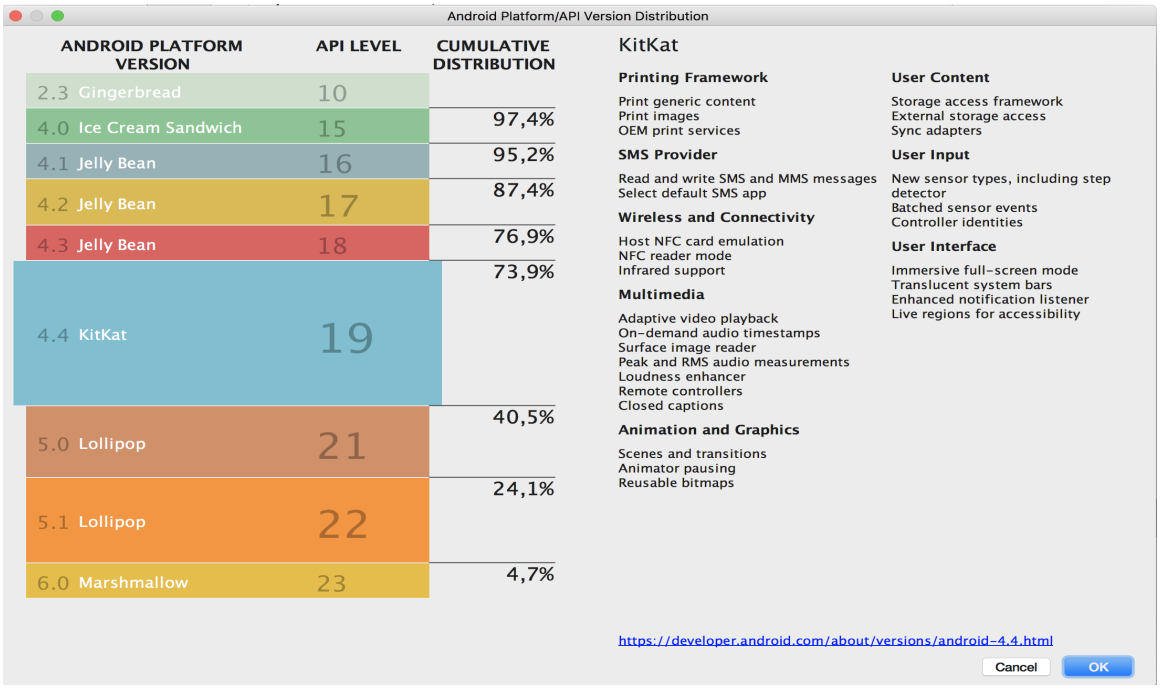


Fig.III.1 Distribución de versión de Plataforma/API Android.

En comparación con versiones anteriores como Gingerbread, Ice Cream Sandwich y Jelly Bean representa un porcentaje menor, pero si se escoge una versión muy baja pensando en abarcar una mayor cuota de mercado, se perderan nuevas e importantes funciones. Poniendo en la balanza estas dos características (cuota de mercado y funcionalidad) se decidió trabajar con la versión 4.4 KitKat, ya que, si se tomaba la decisión de desarrollar en Marshmallow, obtendría lo último en

funcionalidad, pero el número de usuarios a alcanzar iba a ser muy acotado, debido a que no hay muchos dispositivos que cuenten con ella hasta el momento.

III.2. Objetivos de la aplicación

El objetivo general de este proyecto es desarrollar un sistema fácil de usar, que sirva como intermediario entre personas que prestan servicios a la comunidad y quienes desean contratar alguno de ellos.

Otros objetivos también importantes, relativos al sistema global, son:

- Comunicación bidireccional entre las dos plataformas (Web y Android) a través del API REST.
- Demostración del funcionamiento del sistema completo de la arquitectura.

Por otro lado, los objetivos específicos de la plataforma Android son:

- Las aplicaciones deben tener una interfaz de usuario intuitiva y fácil de usar.
- Las aplicaciones deben estar realizadas para dispositivos que tengan al menos la versión 4.4.
- Las aplicaciones se deben ajustar a pantallas de entre 3,2 y 5 pulgadas smartphones.

IV. Solución

En este capítulo se explica cómo se desarrolló la implementación que forma parte de la solución del problema planteado en capítulos anteriores. Primero se describe la planificación del proyecto en general y como se realizó el seguimiento del mismo. También se detallan las herramientas de trabajo que se utilizaron en el desarrollo y las tecnologías relevantes como Java EE, Android y Api[10] REST.

Para explicar el desarrollo completo de la solución, se presentan las HU que formaron parte del proyecto divididas en seis Sprint.

IV.1 Planificación

A la hora de llevar a cabo un proyecto, es muy importante tener en cuenta el proceso de trabajo, el tiempo disponible, el resultado que se quiere obtener y la capacidad del equipo de desarrollo. En la actualidad, obtener un resultado rápido y un producto que se pueda ver, mostrar y utilizar, se ha vuelto crucial para el éxito. En consecuencia, nacieron los procesos de desarrollo de software iterativos que intentan evitar las desventajas de las metodologías tradicionales, uno de ellos es la metodología ágil que se enfoca en el usuario y los resultados.

Existen varias metodologías de este tipo, pero todas ellas se guían por los mismos valores, definidos en el Manifiesto Ágil. “Se centran en el individuo y sus interacciones, más que en el proceso y las herramientas utilizadas; en desarrollar software que funcione en lugar de obtener una buena documentación; en la colaboración con el cliente y en responder a los cambios en lugar de seguir una planificación”.

Para llevar adelante el desarrollo del proyecto se siguió un proceso iterativo tomando como guía algunos principios de la metodología ágil. Entre ellos se encuentran, aceptar que los requerimientos cambien en cualquier etapa del desarrollo, realizar entregas tempranas y continuas de software y comunicar el progreso al equipo mediante reuniones cara a cara. Se aplicó este proceso, ya que, al ser un trabajo final de grado, se requiere completarlo en un tiempo limitado.

[10] API acrónimo en Inglés de Application Programming Interface

Además de esta manera el director puede estar involucrado en el proceso de trabajo, pudiendo aconsejar y ver el resultado de cada entrega desde otro punto de vista.

Los ciclos de trabajo se planificaron con una duración sugerida en el proceso iterativo, de aproximadamente 3 semanas, dependiendo de la complejidad de las tareas a realizar en cada uno. Por cada ciclo finalizado, se realizó una reunión con el objetivo de observar y discutir las modificaciones realizadas, también para plantear los pasos a seguir en el siguiente ciclo, tanto las mejoras a realizar en la aplicación, como las nuevas funcionalidades requeridas.

En la siguiente lista se muestra la descomposición de tareas del proyecto:

1. Gestión de Proyecto (52 horas):
    a. Realizar reuniones con el dueño del producto (22 horas). Todos los jueves de cada mes, se llevó a cabo una reunión vía videoconferencia con quien es la dueña del producto, con el fin de verificar y validar los avances del proyecto y avanzar con la toma de requisitos.
    b. Documentar objetivos y redacción de la introducción (20 horas). Durante los dos primeros meses se confeccionaron los objetivos que deberá cumplir el desarrollo de la aplicación para satisfacer las necesidades del cliente.
2. Backend (310 Horas):
    a. Análisis, en esta se etapa se realizaron las siguientes actividades:
        i. Identificación de los participantes del sistema.
        ii. Establecimiento del catálogo de requisitos.
        iii. Creación de tareas en Trello.
    b. Diseño, en esta se etapa se realizaron las siguientes actividades:
        i. Establecimiento de cómo deben almacenarse los datos.
        ii. Creación del diagrama de clases y diseño del plan de pruebas.

c. Implementación, en esta etapa se realizaron las siguientes actividades:
    i. Se construyeron las capas según el modelo M.V.C.[11]
    ii. Se desarrollaron los servicios REST que permiten la comunicación con la aplicación móvil.
d. Prueba: Se realizaron pruebas unitarias de los módulos construidos y pruebas de integración.

3. Frontend, Aplicación Móvil (340 Horas):
    a. Análisis: Se analizó la situación de la aplicación y se establecieron una serie de directrices para el desarrollo.
    b. Diseño: Se establecieron criterios para realizar las interfaces de la aplicación. Se crearon Mockups. Se implementó la comunicación del Móvil con el Backend.
    c. Implementación: Se desarrollaron los servicios y las actividades. También se crearon las interfaces de configuración.
    d. Prueba: Se realizaron pruebas unitarias de los módulos construidos y pruebas de integración.

Duración Total estimada: 702 Horas.

IV.2 Seguimiento del proyecto

El seguimiento del proyecto se realizó a través de las herramientas de Issue Boards de GitLab, Google Drive y Trello. Las mismas son aplicaciones basadas en web que permiten el seguimiento y documentación del proyecto. La fundamentación de la elección está basada en que cumplen con los requisitos básicos de la metodología Scrum.

Trello permite realizar los tableros propios de la metodología (Henrik Kniberg, 2007) es un gestor de tareas que permite el trabajo de forma colaborativa mediante

[11] M.V.C. Patrón de Diseño Model View Controller.

tableros (board) propios de la metodología (Henrik Kniberg, 2007) compuestos de columnas que representan distintos estados.

Se basa en el método Kanban para gestión de proyectos, con tarjetas que viajan por diferentes listas en función de su estado. Así, se puede tener una lista de cosas pendientes, en proceso o terminadas.

Por otro lado, GitLab presenta Issue Boards, una herramienta que proporciona una interfaz visual donde los miembros del equipo pueden seguir el estado de sus proyectos. Las Issue son una gran manera de realizar un seguimiento de tareas, mejoras y errores para el proyecto, ya que pueden compartirse y debatirse con el resto de su equipo.

Por su parte Google Drive permite el tratamiento de documentos on-line y su colaboración en tiempo real, incluyendo un servicio de almacenamiento en nube que permite visualizar, editar y compartir documentos. Además, brinda respaldo automático, control de versiones y revisión sin conexión a Internet. Ambas herramientas permitirán el seguimiento y documentación del proyecto a lo largo de su desarrollo.

Se destaca que al utilizar Scrum en este proyecto se realizó una adaptación del mismo de acuerdo con el estilo del proyecto y el equipo del trabajo. Los roles para este proyecto se dividieron de la siguiente manera:

- Scrum Master (director del proyecto): Mauro Cambarieri.
- Team:
    1. Difabio, Guillermo Alejandro (Líder equipo Móvil).
    2. Ricket, German (Desarrollador Móvil).
    3. Arce, Rodrigo (Desarrollador Móvil).
    4. Difabio, Federico Ezequiel (Desarrollador Java).
    5. Mauro Cambarieri (Líder equipo Java).
- Product Owner (dueño del producto): Alejandra Viadana (Conocedora del dominio).

Durante la primera semana se estableció la pila del producto y los objetivos a cumplir en cada sprint. Se decidió que estos tuviesen una duración de 2 semanas.

A continuación, se muestra una breve descripción del desarrollo del proyecto por cada sprint:

- **Sprint 0**
  - **Definición de Arquitectura**
  - **Mockup de la Aplicación**
  - **Metodología de trabajo**

- **Sprint 1**
  - **HU.1 Registro de usuarios.**
    - HU.1.1 Registro de usuario con email.

      Registro de usuario solicitando datos como nombre, apellido, email y contraseña. Todos requeridos para poder completar el registro. Antes de completar la operación se debe comprobar que el email ingresado no esté en uso por otro usuario.
    - HU.1.2 Registro de usuario con Google.

      Registro de usuario utilizando la Api de Google para tomar la información de la cuenta como email, nombre y apellido. Luego se pasará a la pantalla de Registro con dichos datos precargados en el formulario.
    - HU.1.3 Registro de usuario con Facebook.

      Registro de usuario utilizando la Api de Facebook para tomar la información de la cuenta como email, nombre y apellido. Luego se pasará a la pantalla de Registro con dichos datos precargados en el formulario.
  - **HU.2 Login con Redes Sociales.**
    - HU.2.1 Login con email y contraseña.

      El usuario del sistema se autentifica en el mismo como Oferente y Demandante de servicios, haciendo uso de sus credenciales (email y contraseña).

- HU.2.2 Login con Google.

  El usuario del sistema se autentifica en el mismo como Oferente y Demandante de servicios, haciendo uso de las credenciales de Google (email y contraseña).
- HU.2.3 Login con Facebook.

  El usuario del sistema se autentifica en el mismo como Oferente y Demandante de servicios, haciendo uso de las credenciales de Facebook (email y contraseña).

- **Sprint 2**
  - **HU.3 Gestión de Perfil de Usuarios.**
    - HU.3.1 Cargar datos de Perfil.

      Formulario que contenga todos los datos correspondientes al perfil de usuario, en el cual podemos identificar Nombre, Apellido, Email, Teléfonos, Fecha de nacimiento, Domicilio completo, y una presentación del usuario.
    - HU.3.2 Upload de Imagen de Perfil.

      Seleccionar una imagen del dispositivo móvil, ya sea desde la galería o usando la cámara para tomar una foto. Subir esta imagen como foto del perfil al servidor.
    - HU.3.3 Cambiar estado.

      Cambiar el estado (Ocupado/ Disponible) de la persona tocando la foto de perfil de la barra de navegación.
  - **HU.4 Gestión de Curriculum Vitae.**
    - HU.4.1 Administración de experiencia laboral.

      Alta, Listado, Baja y Modificación de la sección experiencia laboral de CV. Formulario que contenga los datos correspondientes experiencia laboral, en el cual podemos identificar Titulo del trabajo, Lugar de trabajo, Descripción, Fecha desde y Fecha Hasta.

        - HU.4.2 Administración de idiomas.

          Alta, Listado, Baja y Modificación de la sección idiomas del CV. Formulario que contenga los datos correspondientes a idiomas, en el cual podemos identificar Nombre y Nivel.
        - HU.4.3 Administración de educación.

          Alta, Listado, Baja y Modificación de la sección educación del CV. Formulario que contenga los datos correspondientes a educación, en el cual podemos identificar Nivel de educación, Institución, Título de la carrera, Descripción de educación, Fecha desde y Fecha Hasta.
        - HU.4.4 Administración de cursos.

          Alta, Listado, Baja y Modificación de la sección cursos del CV. Formulario que contenga los datos correspondientes a cursos, en el cual podemos identificar Nombre y Asociado con.

- **Sprint 3**
    - **HU.5 Gestión de Servicios**
        - HU.5.1 Administración de servicios.

          Alta, Listado, Baja y Modificación de servicios. Formulario que contenga los datos correspondientes de servicio, en el cual podemos identificar Titulo de servicio, Imágenes, Descripción, Tipo de cotización, Valor, Ubicación (Provincia y Localidad), Categoría y Subcategoría.
        - HU.5.2 Administración de imágenes del servicio.

          Seleccionar una o más imágenes del dispositivo móvil, ya sea desde de la galería o usando la cámara para tomar una foto. Subir al servidor estas imágenes como galería de fotos del servicio a publicar.

- **Sprint 4**
    - **HU.6 Listado de Servicios.**

      Listar los servicios almacenados en la base de datos por los usuarios.
    - **HU.7 Filtro de Servicios.**

      Filtrar servicios almacenados en la base de datos del sistema por categoría, nombre y estado.

- **Sprint 5**
    - **HU.8 Mapa de Servicios Georreferenciados**
        - HU.8.1 Filtro de Servicios Georreferenciados.

          Filtrar servicios almacenados en la base de datos del sistema por categoría, nombre, estado y distancia.
        - HU.8.2 Publicar ubicación.

          Permitir al usuario oferente que tenga GPS del móvil activado publicar su ubicación (latitud y longitud) actual, para que otros puedan encontrarlo.
        - HU.8.3 Información de Usuario Georreferenciado.

          Ver Información detallada de un usuario en particular al presionar sobre los marcadores, que identifican a usuarios oferentes activos con su posición GPS pública.
- **Sprint 6**
    - **HU.9 Gestión de Contratación**
        - HU.9.1 Comentarios de publicaciones.

          La primera comunicación del usuario demandante de servicio con el usuario que publica estos mismos, es a través de comentarios, que permitirán despejar dudas previas a solicitar una contratación.
        - HU.9.2 Proceso de contratación.

          El proceso de contratación está compuesto por distintos estados. Una vez que el usuario demandante quiera contratar

un servicio lo primero será enviar al usuario oferente una notificación y se registrará una solicitud de contratación (Estado PENDIENTE), la cual tendrá una fecha de creación de solicitud, datos del usuario demandante y un tiempo para responder. El usuario oferente deberá aceptar la solicitud para dar curso a dicha operación. Esta tendrá diferentes estados (PENDIENTE, ACEPTADA, EN CURSO, CANCELADA, FINALIZADA). Si la contratación es CANCELADA deberá tener un motivo, esta acción se podrá realizar desde cualquier estado. Para que la contratación pase a estado FINALIZADA, previamente el sistema habilitará los parámetros de puntuación y comentario.

- HU.9.3 Calificación de usuarios.

  Al momento de finalizar con la contratación el usuario demandante del servicio deberá calificar el trabajo realizado por proveedor del servicio. Para ello la app tendrá que realizar una serie de preguntas definidas de antemano para ayudar con la calificación.

- **HU.10 Servicio de Notificaciones**
  - HU.10.1 Notificar nuevos mensajes.

    Notificar a los usuarios sobre nuevos mensajes. Para ello se mostrará un mensaje en la barra de notificaciones del dispositivo móvil y luego de ser leído se ocultará.
  - HU.10.2 Notificar nuevas solicitudes.

    Notificar a los usuarios sobre nuevas solicitudes de Contratación de servicios. Para ello se mostrará un mensaje en la barra de notificaciones del dispositivo móvil y luego de ser leído se ocultará.

IV.3 Herramientas utilizadas

Dada la gran variedad de herramientas disponibles para llevar adelante la implementación de la aplicación, se realizó un estudio para determinar cuáles se adaptan mejor al proyecto. Se debió seleccionar la metodología de trabajo, el lenguaje de programación a utilizar, el tipo de base de datos para el dispositivo móvil y para el servidor implementado en conjunto con la aplicación, editores de código, bibliotecas externas y el ambiente de desarrollo.

IV.3.1. Sistema de control de versiones de código

GitLab es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git[12]. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis[13] y un sistema de seguimiento de errores, todo ello publicado bajo una Licencia de código abierto. Contiene todo lo necesario para llevar a cabo el proyecto bajo la metodología ágil, desde la creación de la idea a la producción de la misma.

IV.3.2. Documentación del proyecto

Una buena documentación es clave para el éxito de cualquier proyecto. Haciéndola accesible permitirá a los demás desarrolladores aprender acerca del mismo y además si es fácil de actualizar asegura que esta sea útil y se mantenga relevante.

Una vez más GitLab resulta de gran utilidad, ya que a la hora de generar documentación ofrece dos maneras de hacerlo, por una parte, los archivos de texto llamados README, los cuales son una manera rápida y sencilla para que otros usuarios puedan aprender más acerca de su trabajo, ya que es con lo primero que se encuentran al ingresar al proyecto. El mismo deberá contener solamente la información necesaria para que los desarrolladores comiencen a utilizar y contribuir

[12] Software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

[13] Es el nombre que recibe un sitio web, cuyas páginas pueden ser editadas directamente desde el navegador, donde los usuarios crean, modifican o eliminan contenidos que, generalmente, comparten.

sobre el proyecto. Para la creación de dicho archivo se siguió un formato, el cual debía contener:

Nombre del proyecto: Es lo primero que los desarrolladores verán al desplazarse hacia abajo en el README.

Descripción: Se describe la importancia del proyecto y lo que hace.

Tabla de Contenidos: Opcionalmente, se incluye una tabla de contenidos con el fin de permitir a otras personas a navegar rápidamente sobre el README.

Instalación: Se explica a otros usuarios cómo instalar el proyecto a nivel local.

Uso: Se indica a otras personas sobre el uso del proyecto después de que se haya instalado. Aquí también se incluyen capturas de pantalla.

Créditos: Se incluye una sección de créditos con el fin de resaltar a los autores del proyecto.

Licencia: Por último, incluye una sección para la licencia del proyecto.

Todas estas secciones se crean con el fin de orientar a los desarrolladores de inmediato a los aspectos más importantes del proyecto, de manera tal que comiencen a utilizar y contribuir sobre el mismo lo antes posible.
Por otro lado, todo repositorio en GitLab cuentan con un wiki, el cual se puede configurar y presentar información en profundidad acerca de su proyecto de una manera útil.

IV.3.2. Entorno de trabajo

Para desarrollar la aplicación móvil, se utilizó Android Studio, el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y se basa en Intellij Idea (Google, 2015). Desde el 2015 tiene su primera versión estable 1.0 y actualmente está la versión 2.3.3 que funciona correctamente para los fines del proyecto. Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, ofrece aún más funciones que aumentan la productividad durante la compilación de apps para Android, como un sistema de compilación basado en Gradle, un emulador rápido con varias funciones, un entorno unificado que permite realizar desarrollos para todos los dispositivos Android, Instant Run

para aplicar cambios mientras tu app se ejecuta sin la necesidad de compilar un nuevo APK, integración de plantillas de código y el plugin de Git para facilitar la importación/exportación de código, etc. Además del IDE, Google ofrece un SDK[14] que permite desarrollar cualquier tipo de aplicación para Android.

El desarrollo de las aplicaciones en Java, como es el caso de la Web y el backend que brinda los servicios REST, se realizó con el IDE Intellij Idea 2017.1.1 con licencia para estudiante. Al utilizar el mismo entorno de trabajo para ambas aplicaciones facilitó el desarrollo. Para la gestión de todos los componentes del proyecto y manejo de dependencias se utilizó Gradle.

IV.3.3 Tecnologías y arquitectura del sistema

La arquitectura de software conforma la columna vertebral de cualquier sistema y constituye uno de sus principales atributos de calidad. El documento de IEEE Std 1471-2000 define: “La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución”.

La aplicación se ejecutará sobre un esquema cliente/servidor, con los procesos e interfaz de usuario ejecutándose en los clientes y estos solicitando requerimientos al servidor que cumple su proceso.

Patrones de diseño

MVP (Model View Presenter) es un patrón derivado del conocido MVC (Model View Controller), permitie conseguir que una misma lógica pudiera tener vistas totalmente diferentes e intercambiables.

[14] SDK Significa Kit de Software de Desarrollo.

Librerías empleadas

Con el objetivo de facilitar el desarrollo de ciertas partes del proyecto, se utilizaron varias librerías externas, entre la cuales se puede identificar:

Retrofit: Librería pensada para peticiones API, es un cliente REST para Android y Java, desarrollada por Square que permite hacer peticiones GET, POST, PUT, PATCH, DELETE y HEAD; gestionar diferentes tipos de parámetros y convertir automáticamente la respuesta a un POJO (Plain Old Java Object).

Glide: Es un marco rápido y eficiente de gestión de medios de código abierto y carga de imágenes para Android que envuelve la decodificación de los medios de comunicación, memoria y almacenamiento en caché de disco, y la puesta en común de recursos en un simple y fácil de usar interfaz.

DbFlow: Una biblioteca robusta, potente y muy simple de base de datos. ORM Android con procesamiento de anotaciones, que proporciona una API potente y fácil de manejar interacciones.

Butter Knife: Es una librería que facilita la tarea de relacionar los elementos de las vistas con el código en aplicaciones Android. Evita tener que utilizar findViewById y simplificará el código.

EventBus: Es un bus de eventos de publicación / suscripción optimizado para Android. Simplifica la comunicación entre los componentes, desacopla los remitentes y receptores de eventos. Funciona bien con Actividades, fragmentos y subprocesos de fondo.

Gson: Es una biblioteca de Java que se puede utilizar para convertir objetos Java en su representación JSON. También se puede utilizar para convertir una cadena JSON a un objeto Java equivalente.

IV.4 Desarrollo de la solución

Para el desarrollo de la solución se utilizaron los siguientes artefactos que propone la metodología ágil Scrum y los diagramas del lenguaje UML:

- Diagramas de Clase.
- Historias de Usuario.
- Tablero de Tareas.
- Diagramas de transición de estados (DTE).

IV.4.1. Historias de Usuario

Las historias de usuario (HU) van a tener el formato que sugiere la bibliografía consultada que se tomó para la documentación del desarrollo. En la figura IV.1 se puede ver un ejemplo donde se definen las siguientes columnas:

- ID: Identificador.
- Nombre: Un nombre descriptivo.
- Imp: Importancia, valor numérico de 0 a 100 que establece la importancia del negocio.
- Est: Estimación, un valor numérico entero, que representa el número de días que llevaría desarrollar la HU (un día está compuesto de 8 horas de un recurso).
- Cómo probarlo: Una descripción de cómo probar el sistema para comprobar que hace lo que el dueño del producto desea.
- Notas: Cualquier observación o comentario que puede ayudar al desarrollo de la tarea.

ID	Nombre	Importancia	Estimacion	Cómo Probarlo	Nota
1	Login con Redes Sociales	50	5	Iniciar sesión en la aplicación haciendo uso de cuentas como Gmail, Facebook y twitter.	Usuario del sistema, tendrá username y clave para autenticarse en el sistema. La aplicación tomará datos de cuentas de Redes Sociales como Correo Electrónico, Nombre ,etc..
2	Mapa de Servicios Georeferenciados	100	6	Iniciar sesión con un usuario que tenga su posición activa y consular por servicios cercanos a el en el Mapa de Servicios.	Mostrar en el Mapa personas que ofrecen la prestación de algún servicio y están cerca del usuario que realiza la consulta.
3	Administración del perfil de Usuario	60	15	Modificar datos del Usuario (Nombre, Apellido, email, etc). Cambiar contraseña	Se desarrollará el módulo de administración de perfil de usuario: Trabajos realizados, Referencias, Documentación(CV adjunto) de los usuarios oferentes.

Fig.IV.1. Ejemplo del formato de las Historias de usuario.

Cada HU puede descomponerse en una o varias tareas para implementar la funcionalidad descripta. A continuación, solo se presentarán en detalle las actividades más relevantes del sistema, como es la definición de la arquitectura en el Sprint 0, la comunicación de la aplicación con las SDK de Facebook y Google para el inicio de sesión y registro de usuario planteado en el Sprint 1, Mapa de Servicios Georeferenciados en el Sprint 5 y por último la gestión de contrataciones y notificaciones planteado para el Sprint 6.

Sprint 0

Definición de Arquitectura

Dentro de la metodología Scrum, se puede definir un Sprint 0, en el cual se determinan cuestiones de tecnologías y arquitectura de software que el sistema podría requerir para la implementación (Anurag Prakash, 2013). Dentro de este Sprint, se analizó la comunicación de la aplicación móvil con el backend.

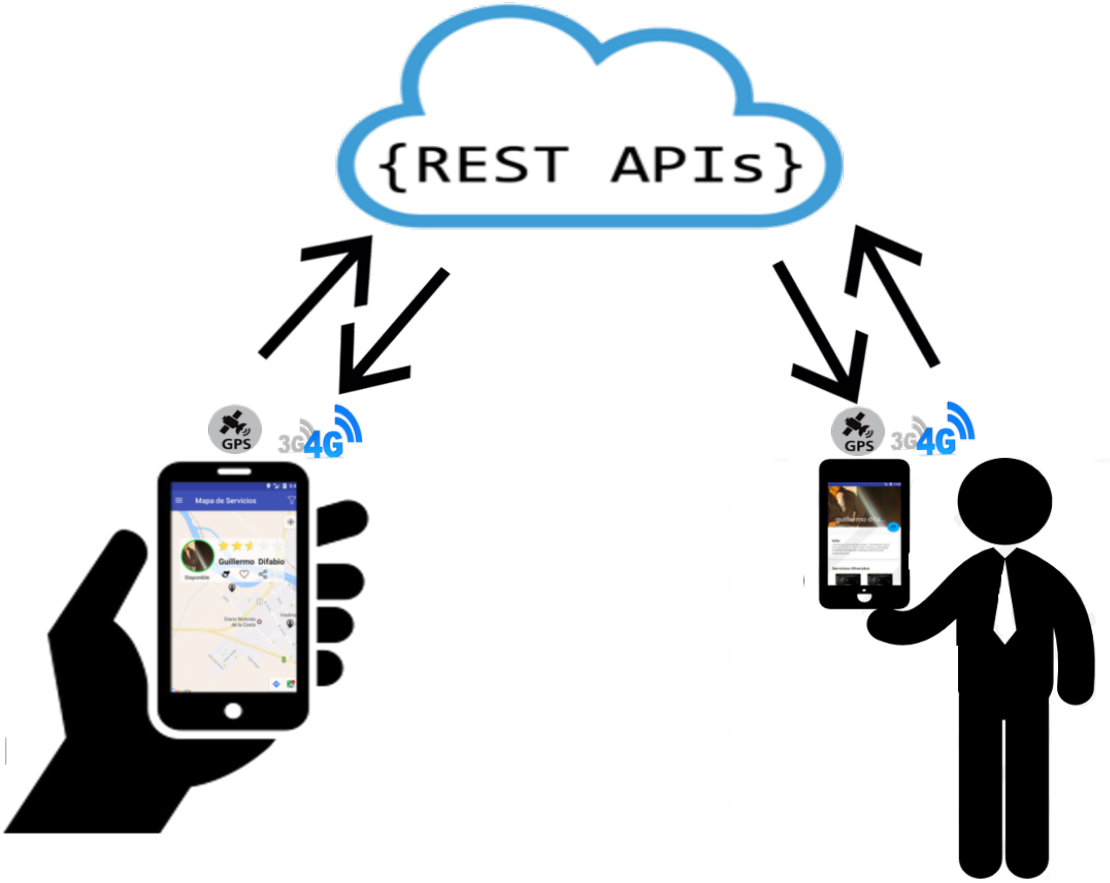


Fig.IV.2. Esquema de conexión.

En la figura IV.2 se puede ver el esquema de la arquitectura propuesta. La conexión entre los dispositivos móviles con el servicio en la nube mediante WIFI o 3G/4G, a partir de la cual cada usuario va a poder ofrecer servicios y/o contratar los ya publicados por terceros, según lo requieran. También se puede observar el uso del

sistema GPS[15] integrado en la mayoría de los smartphones, que permitirá al demandante de servicios geolocalizar a los usuarios oferentes que estén dentro del radio de su ubicación. Para la aplicación propuesta se diseñó una arquitectura que se basa en dos partes bien definidas:

Backend: basado en tecnologías Java. Para el desarrollo de este módulo se utilizó una arquitectura en capas lógicas bien definidas: la capa de presentación, la capa de negocio y la capa de persistencia. El principio para la separación en capas es que cada una esconde su lógica al resto y solo brinda puntos de acceso a dicha lógica. En la capa de presentación los objetos trabajan directamente con las interfaces de negocios, implementando el patrón arquitectónico Model-View-Controller con Spring MVC brindando servicios REST. La capa de negocio está formada por servicios implementados por objetos de negocio. Estos delegan gran parte de su lógica en los modelos del dominio que se intercambian entre las capas. Finalmente, la capa de persistencia facilita el acceso a los datos y su almacenamiento en una base de datos mediante la tecnología Spring Data JPA con el soporte de Hibernate. La integración de las capas está a cargo de Spring a través de su contenedor de inversión de control (IoC) de Beans y su paradigma de Inyección de Dependencias. Spring es un framework que aporta aún más funcionalidades a esta arquitectura además de su comportamiento como contenedor, por lo que las posibilidades sobre esta arquitectura quedan abiertas para la integración de nuevos módulos como Spring AOP, y el módulo de seguridad de autenticación y autorización como Spring Security.

El diseño de la arquitectura y la metodología de trabajo permitió la implementación a los temas inherentes definidos en cada uno de los Sprints: registro de usuario, gestión de perfil, de servicios, de contrataciones y administración de consultas y operaciones. La aplicación utilizada como intermediaria permite informatizar el proceso tomando y generando interfaces de información.

Frontend: La interfaz estará representada por la página web y la aplicación móvil, los cuales estarán basados en tecnologías Javascript y Móvil respectivamente. La

[15] El Sistema de Posicionamiento Global, más conocido por sus siglas en inglés, GPS, es un sistema que permite determinar en toda la Tierra la posición de un objeto con una precisión de hasta centímetros.

interfaz móvil se encuentra definida por una aplicación nativa para dispositivos con Sistema Operativo Android. Para el desarrollo de la misma se implementó el patrón Model View Presenter (MVP), este es un derivado del conocido Model View Controller (MVC). Permite separar la capa de presentación de la lógica de la misma, de tal forma que todo lo relacionado con el funcionamiento de la interfaz queda separado de la presentación en pantalla. Idealmente el patrón MVP permitiría conseguir que una misma lógica pudiera tener vistas totalmente diferentes e intercambiables.

Por otra parte, la aplicación WEB, permite la administración y la gestión de los usuarios de la aplicación móvil y también la gestión de la información que será utilizada por esta. Para el desarrollo de este módulo se utilizó AngularJS, que es un framework Javascript[16] que permite escribir aplicaciones web basado en el patrón MVC, pero del lado del cliente. AngularJS funciona como una single-page application, dado que renderiza en principio un solo HTML base, y luego va reemplazando la vista principal con sectores HTML más pequeños, que corresponden a cada página de la aplicación.

Mockup de la Aplicación

Antes de comenzar con el desarrollo de la aplicación se estableció un prototipo inicial de al menos una parte de la funcionalidad del sistema, el cual permitió definir en conjunto con el dueño del producto la interfaz de dicha aplicación. A continuación, se pueden observar algunas de las pantallas diseñadas en esta etapa:

[16] Javascript es un lenguaje de programación interpretado.

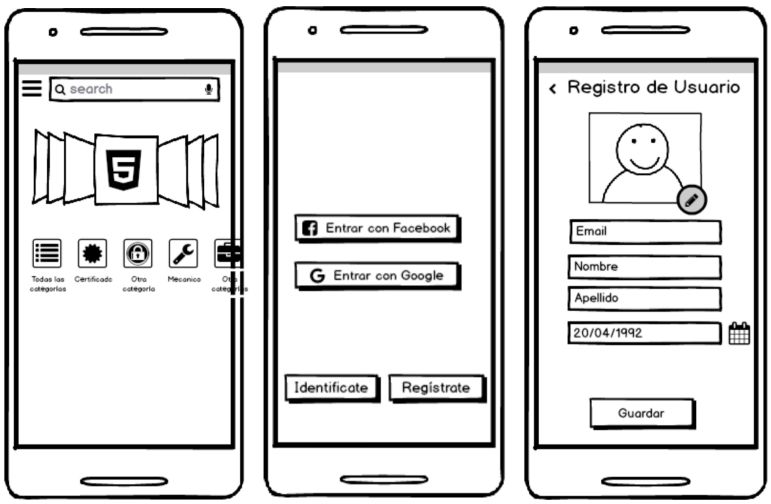


Fig.IV.3. Mockup de la aplicación.

Metodología de trabajo

Como se explicó al principio de este capítulo el equipo de desarrollo utilizó GitLab, un software de control de versiones diseñado para desarrollar en equipo y realizar el mantenimiento de versiones de la aplicación, controlando los cambios que se produzcan en los distintos archivos del proyecto.

Se utilizó en principio las Issues de GitLab, las cuales se crean por cada tema sobre el que quiere trabajar, ya sea un problema por resolver o un requerimiento nuevo tal como se puede ver en la figura IV.4.

Fig.IV.4. Issue contratación de servicios.

También se hizo uso de los Milestone el cual permitió agrupar issues, y asignarle el tiempo estimado para resolverlo, como por ejemplo en la figura IV.5 se ve el milestone correspondiente al Sprint 5.

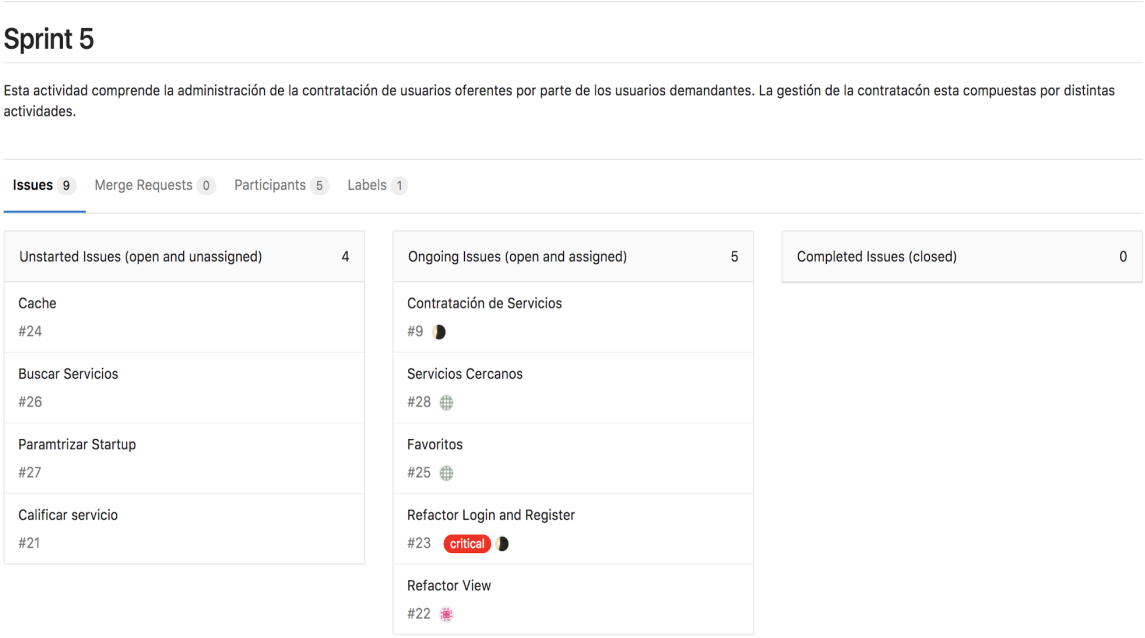


Fig.IV.5. Milestone correspondiente al Sprint 5.

Para comenzar con el desarrollo se configuraron en la herramienta GitLab dos branch principales, master y develop. En Android Studio se pude tener en el mismo proyecto, pero en diferentes ramas el código de compilación de la aplicación estable (branch master) y por otro lado el código para desarrollar y testear (branch develop). Por lo tanto, los desarrolladores trabajarán únicamente en el branch develop y por cada requerimiento(issue) se creará un nuevo branch tomando como base este, hasta finalizar el requerimiento y una vez probado se realizará el merge al branch develop. Tras las pertinentes pruebas, se fusiona (merge) esa rama y se envía (push) a la rama de producción master, la cual se utilizará para crear los releases (cierre de Sprint).

Esto permitió acceder y recuperar versiones, por ejemplo, tras detectar un error que se cometio durante el desarrollo, y continuar otra vez a partir de un punto anterior. También permitió desarrollar y mantener diferentes versiones o ramas de forma

paralela, por ejemplo, una versión principal (branch master) y otra de test para realizar pruebas (branch develop).

Sprint 1

Para el seguimiento de este Sprint, se implementó el siguiente tablero de Trello, en el cual se pueden ver en detalle todas las tareas de las HU planteadas para esta interacción. Aquí se desarrollaron las tareas de registro de usuario y autenticación del sistema.

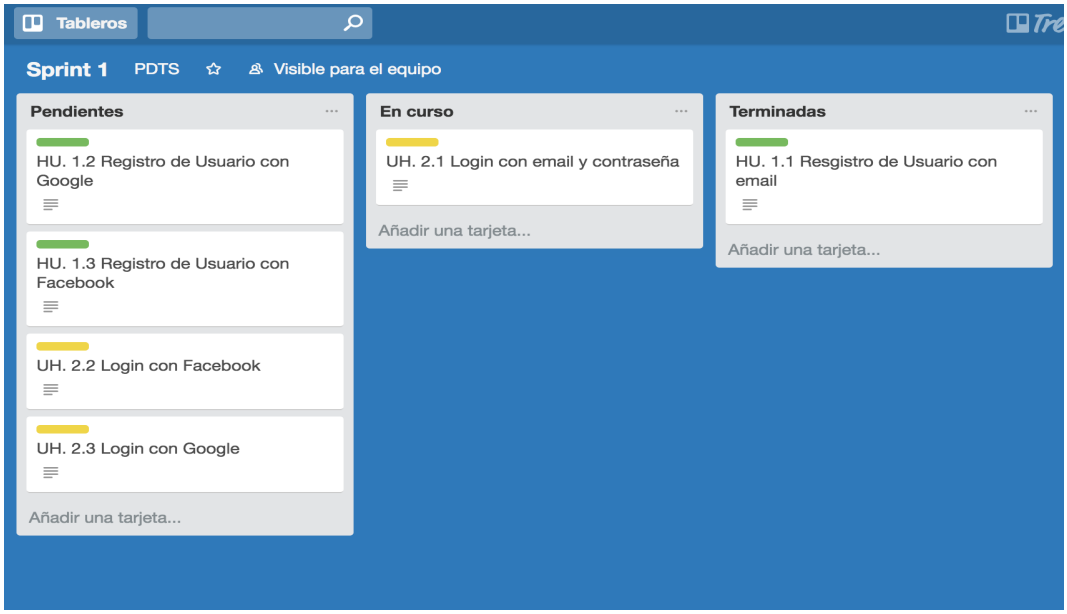


Fig.IV.6. Tareas de la HU 1 y 2 del Tablero Scrum.

HU.1 Registro de usuarios

Detalle de la H.U.

- **ID**: 1
- **Nombre**: Registro de usuarios.
- **Importancia**: 100
- **Estimación**: 5
- **Como probarlo**: Registrar un usuario, completando el formulario con datos como nombre, apellido, email y contraseña.
- **Nota**: Todos los campos del formulario deberán ser requeridos.

Esta HU es un Registro de usuario con un esfuerzo de 5 días de trabajo. En el desarrollo de esta actividad se creó una actividad llamada BaseActivity que va a contener todas las operaciones y elementos en común de las demás Activity del sistema, como el menú de navegación lateral.

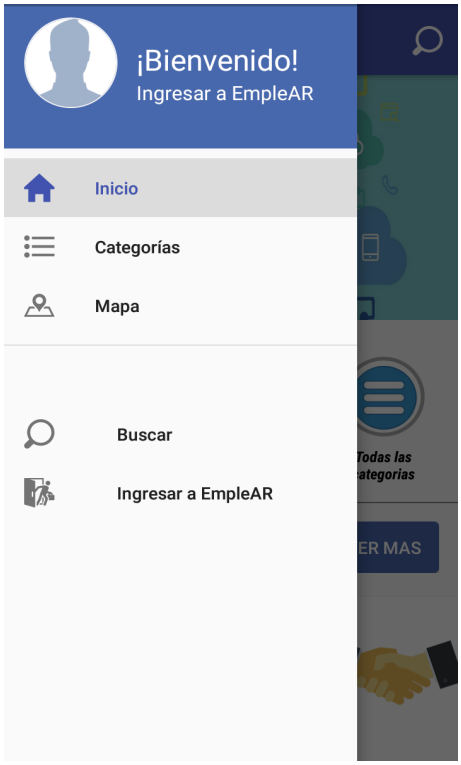


Fig.IV.7. Menú de navegación lateral.

Para crear el menú de la figura IV.7 se construyó un layout con un nodo DrawerLayout, el cual tendrá dos hijos, el contenido principal y un NavigationView. Un layout para el header, donde se añadiran un header con un ImageView y un textView para construir un perfil de la persona logueada y un menú para inflar las opciones de la lista.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <group android:checkableBehavior="single">
       <item
           android:id="@+id/nav_inicio"
           android:icon="@drawable/ic_navigateview_home"
           android:title="Inicio" />
              <item
```

```
            android:id="@+id/nav_perfil"
            android:icon="@drawable/ic_navigateview_user"
            android:title="Mi cuenta" />
             <item
            android:id="@+id/nav_map"
            android:icon="@drawable/ic_navigateview_placeholder_on_map"
            android:title="Mapa" />
    </group>
    <group android:checkableBehavior="single">
        <item android:title="">
            <menu>
                <item
                    android:id="@+id/nav_buscar"
                    android:title="     Buscar"
                    android:icon="@drawable/ic_navigateview_loupe"/>
                <item
                    android:id="@+id/nav_ingresar"
                    android:title="     Ingresar a EmpleAR"
                    android:icon="@drawable/ic_navigateview_in"/>
            </menu>
        </item>
    </group>
</menu>
```

Se diseñaron activities que extienden de BaseActivity, dado que la mayoría de estas van a contener un menú deslizante con sus respectivas funciones por cada ítem. La clase BaseActivity implementa NavigationView.OnNavigationItemSelectedListener permitiendo reutilizar esta funcionalidad a quienes extienden de ella.

```
public abstract class BaseActivity extends AppCompatActivity implements
NavigationView.OnNavigationItemSelectedListener {

   private MenuItem inicioItem;

   private MenuItem loginItem;

   private User user;

   private Profile profile;

   public void navigateToLogin() {

       Intent intent = new Intent(this, LoginActivity.class);

       intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP                                    |
Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_NEW_TASK);

       startActivity(intent);

   }

public boolean onNavigationItemSelected(@NonNull MenuItem menuItem) {

   switch (menuItem.getItemId()) {

       case R.id.nav_inicio:

           navigateToMainScreen();

           break;

       case R.id.nav_ingresar:

           navigateToLogin(this);

           break;

         }

   closeDrawer();

}
```

También se creó una pantalla (Fig.IV.8) que va ser utilizada para las demás HU de este Sprint, ya que va a tener los enlaces hacia las pantallas de registro de usuario con redes sociales, registro con email, login con redes sociales y login con email.

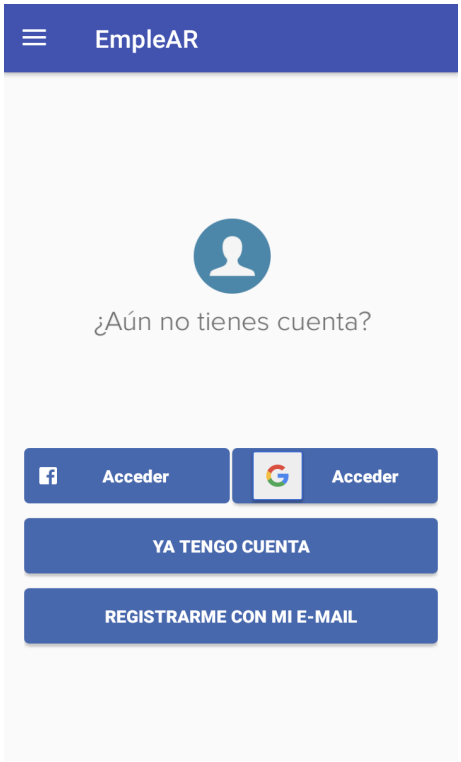


Fig.IV.8. Pantalla general de registro/login.

Para esta implementación se desarrolló un método que va a ser utilizado luego del proceso de login o registro de usuario en el sistema, el cual tiene como objetivo guardar los datos de sesión, como lo son todos los atributos de la clase User y la clase Profile, en las preferencias de Android.

Al guardar en preferencias se utiliza la API SharedPreferences, un objeto SharedPreferences tiene asociado un archivo que contiene pares clave-valor y proporciona métodos simples para leer y escribir dichos pares accediendo a través de su clave. Para tal propósito se encapsuló en una clase llamada SessionPref todas las clases y funcionamiento necesario para poder guardar, borrar, leer y editar datos en la Preferencias de Android. A continuación, se puede ver el código de la implementación descripta anteriormente:

```
public class SessionPrefs {

   public static final String PREFS_NAME = "USER_PREFS";

   private static SessionPrefs INSTANCE;

   private final SharedPreferences mPrefs;
```

```
    private SharedPreferences.Editor editor;
    private User user;
    private Profile profile;
    private Token token;

public User getUser() {
    user = new User();
    profile = new Profile();
    user.setProfile(profile);
   user.getProfile().setFirstname(mPrefs.getString(Constants.USER_FIRSTNAME, null));
    user.getProfile().setLastname(mPrefs.getString(Constants.USER_LASTNAME, null));
    user.setEmail(mPrefs.getString(Constants.USER_EMAIL, null));
    user.setId(mPrefs.getLong(Constants.USER_ID, 0));
    user.setUsername(mPrefs.getString(Constants.USER_USERNAME, null));
    return user;
}
public void setUser(User user) {
    if (user != null) {
 editor.putString(Constants.USER_FIRSTNAME, user.getProfile().getFirstname());
 editor.putString(Constants.USER_LASTNAME, user.getProfile().getLastname());
 editor.putString(Constants.USER_EMAIL, user.getEmail());
 editor.putLong(Constants.USER_ID, user.getId());
 editor.putString(Constants.USER_USERNAME, user.getUsername());
editor.putLong(Constants.PROFILE_BIRTHDATE, user.getProfile().getBirthDate());
            editor.apply(); }
}
}
```

Como se puede observar en el código anterior se utilizó por un lado la clase SharedPreferences, que permite recuperar valores de un archivo de preferencias compartidas, llamando a métodos como getInt() y getString(), proporcionando la clave del valor que se desea recuperar y, opcionalmente, un valor predeterminado para mostrar si no se encuentra la clave. Por otro lado, para realizar operaciones de

escritura en el archivo de preferencias compartidas, se utiliza el SharedPreferences.Editor, llamando a los métodos putInt(), putString(), etc. Luego, se llama a commit() para guardar los cambios.

Fue necesario aplicar el patrón de diseño "Singleton", el cual garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella. Este patrón se implementa haciendo privado el constructor de la clase y creando (en la propia clase) un método que crea una instancia del objeto si este no existe. A continuación se verá cómo se implementó este patrón, donde se creara la clase "SessionPref" con un constructor que va a ser privado y un método llamado 'getSingletonInstance()' que será el encargado de crear una instancia de esta clase si no se ha creado todavía:

```
public static SessionPrefs getSingletonInstance(Context context) {
   if (INSTANCE == null) {
        INSTANCE = new SessionPrefs(context);
   }
   return INSTANCE;
}

private SessionPrefs(Context context) {
   mPrefs = context.getApplicationContext()
           .getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
   editor = mPrefs.edit();
   mIsLoggedIn = !TextUtils.isEmpty(mPrefs.getString(Constants.USER_TOKEN, null));
}
```

HU.1.1 Registro de usuario con email

El detalle de la tarea HU.1.1 se puede ver en la figura IV.9. En la misma se detallan con más precisión los requisitos de la tarea y mediante el progreso del desarrollo de la tarea, se incorpora más información, comentarios y documentación.

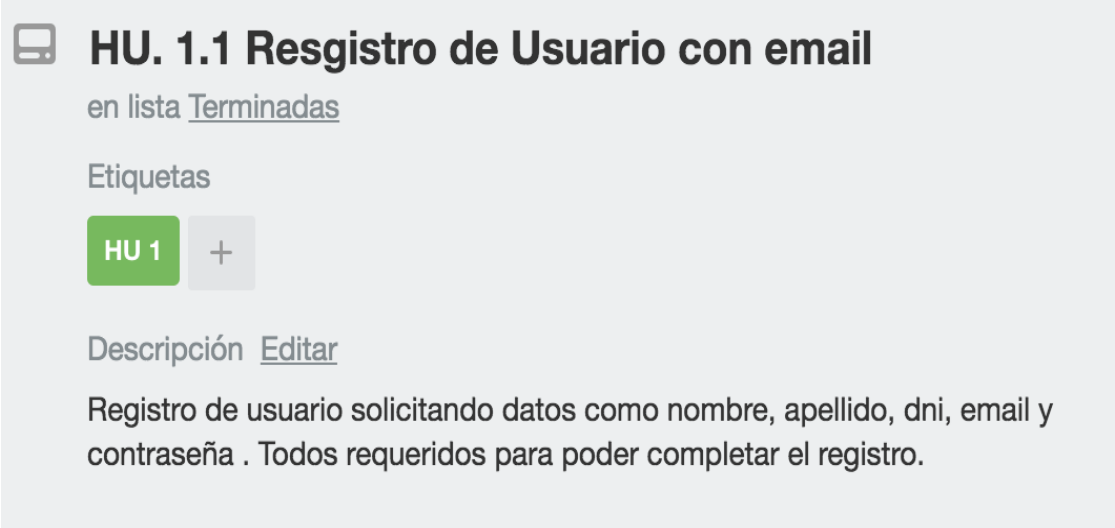


Fig.IV.9. Detalle de la Tarea Alta Usuario de la HU 1.1

La implementación de esta HU comenzó con la construcción del modelo y la lógica de negocio en primer lugar.

Modelo y lógica de negocio

El primer paso para realizar esta tarea fue crear el modelo que representa un usuario mediante un conjunto de Clases y la creación de la interfaz para poder utilizar la librería Retrofit que permite la comunicación con el backend y convertir su API REST en una interface Java. Dado que es una librería, se definieron las clases, métodos y componentes que serán utilizados para realizar la petición POST y registrar a la persona.

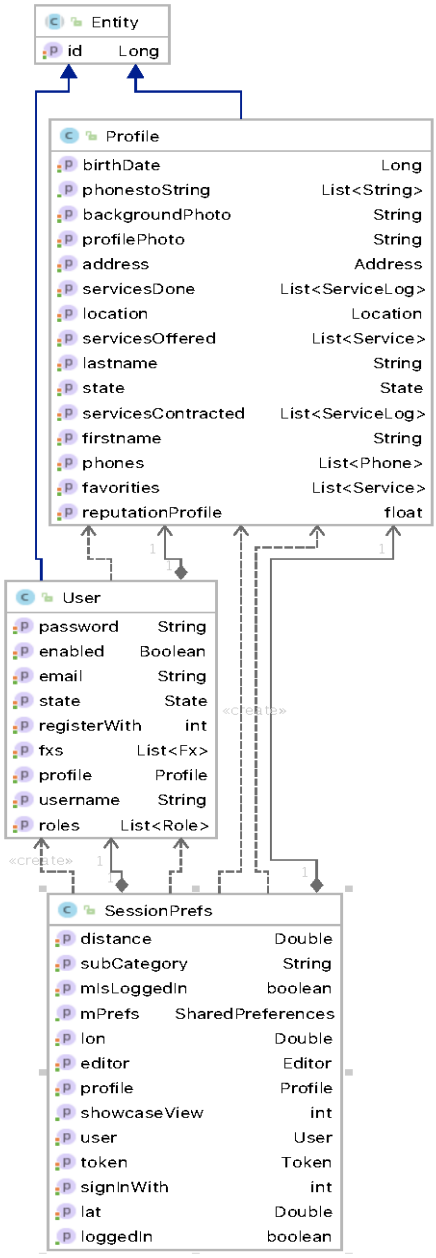


Fig.IV.10. Implementación de la Clase User, SessionPrefs y Profile.

En la figura IV.10 se puede ver un diagrama de clase donde se representan todos los componentes necesarios para la implementación.

Tanto la clase User cómo Profile extienden de la clase Entity, esto permite generalizar las implementaciones de las demás entidades, dado que todas las entidades van a tener una clave ID de tipo Long, con sus métodos getter y setter.

Para la comunicación con el backend desde la aplicación Android se utilizó la librería Retrofit. El primer paso para usar dicha librería es añadir la dependencia en el proyecto vía Build.gradle(Module:app)

```
compile 'com.squareup.retrofit2:retrofit:2.0.2'
compile 'com.squareup.retrofit2:converter-gson:2.0.2'
compile 'com.squareup.retrofit2:converter-scalars:2.1.0'
```

Las aplicaciones en Android poseen un archivo principal de configuraciones, este archivo se llama manifest.xml. Como su nombre lo indica, es un manifiesto de lo que la aplicación puede hacer con el sistema operativo. En este manifiesto se definen todos los componentes que la aplicación tendrá, además de los permisos sobre elementos de hardware y de software. Para utilizar la librería Retrofit será necesario añadir permisos de internet:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Por último, crear una interface y definir cada método, los cuales deberán tener una anotación HTTP que proporciona la solicitud y la URL relativa. Hay cinco anotaciones incorporadas: @GET, @POST, @PUT, @DELETE, y @HEAD que se especifican en las anotaciones URL relativas del recurso. La URL base de solicitud estará conformada por el protocolo de comunicación, el dominio, el puerto y el nombre del servicio web, como por ejemplo http://dominio:puerto/backend/. Mediante el uso de Callback, se hace la petición asíncrona. Por defecto, Retrofit solo puede deserializar la respuesta HTTP en la clase ResponseBody en formato JSON. Para guardar la respuesta en un objeto Java se utiliza la clase GsonConverterFactory.

La clase ApiClient realiza una implementación (Fig IV.11) de todas las interfaces. Para la HU descrita se utilizará la interfaz UserService, que contiene los métodos necesarios para poder realizar la operación de registro y autentificación.

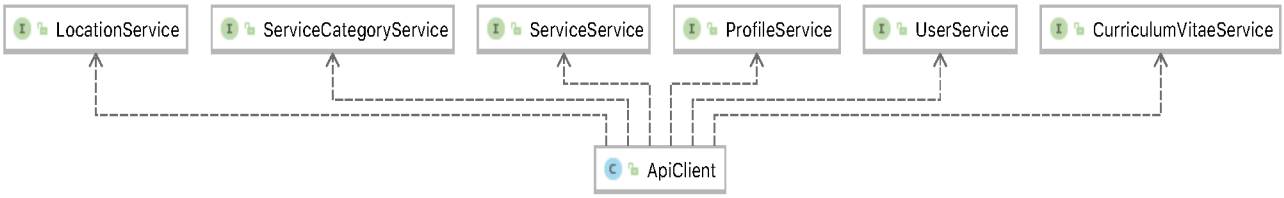


Fig.IV.11. Implementación de la Clase ApiClient.

El código de la clase ApiClient se describe a continuación:

```
public class ApiClient extends AppCompatActivity {
  private Retrofit retrofit;
  private final static String BASE_URL = "http://dominio:puerto/backend/";
  public ApiClient() {
     retrofit = new Retrofit.Builder()
           .baseUrl(BASE_URL)
           .addConverterFactory(GsonConverterFactory.create())
           .build();
  }
  public UserService getUserService() {
     return retrofit.create(UserService.class);
  }
  public ProfileService getProfileService() {
     return retrofit.create(ProfileService.class);
  }
...
}
```

El método register(@Body User user) es utilizado para el proceso de registro de usuario, este invoca al método de la API REST “public/register” del backend, el cual validará que el usuario no exista en la base de datos del sistema, resolviendo la petición mediante un mensaje de éxito de la clase (HttpStatus.OK) o un mensaje de error de la clase (HttpStatus.NOT_FOUND). A continuación, se muestra la implementación de la interface User Service:

```
public interface UserService {
  @GET("api/users/{id}")
  Call<User> getUserById(@Header("Authorization") String authorization, @Path("id") Long userId);
  @POST("api/users/{id}/reset_password")
  Call<User> resetPass(@Header("Authorization") String authorization, @Path("id") Long userId);
  @POST("public/register")
  Call<User> register(@Body User user);
  @Headers("Content-Type: application/json")
  @POST("auth/")
  Call<Token> Auth(@Body LoginBody loginBody);
  @DELETE("api/users/{id}")
  Call<User> deleteUser(@Header("Authorization") String authorization, @Path("id") Long userId);
}
```

En el siguiente fragmento de codigo se muestra la clase SignUpActivity, donde se implementa la interface de vista SignUpView, la cual utilizará componentes del tipo TextInputLayout para poder solicitar los datos del usuario a registrar.

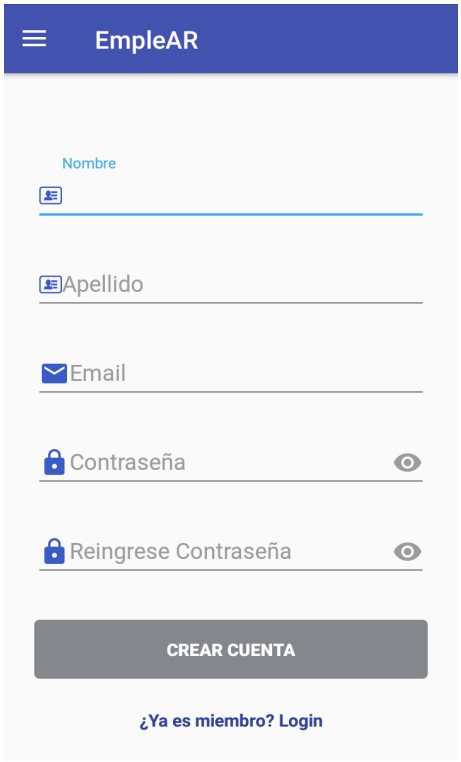


Fig.IV.12. Pantalla de Registro de usuario.

En la figura IV.12 se observa la pantalla de registro de usuario con todos los campos del formulario a completar para realizar dicha operación.

En Android, cada pantalla de la aplicación habitualmente se carga desde un fichero XML que actúa de recurso. Desde SignUpActivity se hace el llamado al archivo xml content_sign_up en el método onCreate, como se muestra a continuación:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.content_sign_up);
    ButterKnife.bind(this);
}
```

El método setContentView(R.layout.content_sign_up) guarda una referencia de la clase estática R.layout, esta permite acceder a la constante content_sign_up que representa el layout en formato xml. El nombre de este layout es

content_sign_up.xml, el cual debe estar en el directorio /res/layout y su contenido es el siguiente:

```
<ScrollView
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:fitsSystemWindows="true">
   <LinearLayout
       android:orientation="vertical"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"

       android:paddingTop="56dp"
       android:paddingLeft="24dp"
       android:paddingRight="24dp">
              <!-- Name Label -->
       <android.support.design.widget.TextInputLayout
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:layout_marginTop="8dp"
           android:layout_marginBottom="8dp"
           android:textColor="@color/primary_dark"
           android:textStyle="normal|bold">
           <EditText android:id="@+id/input_name_singup"
               android:layout_width="match_parent"
               android:layout_height="wrap_content"
               android:drawableStart="@drawable/ic_info"
               android:inputType="textPersonName"
               android:hint="@string/signup.message.name" />
       </android.support.design.widget.TextInputLayout>
        <android.support.design.widget.TextInputLayout
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:id="@+id/text_input_password"
    android:layout_marginBottom="8dp"
    android:textColor="@color/primary_dark"
    android:textStyle="normal|bold">
    <EditText android:id="@+id/input_password_singup"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:drawableStart="@drawable/ic_lock_black_24dp"
        android:inputType="textPassword"
        android:hint="@string/signup.message.password"/>
</android.support.design.widget.TextInputLayout>
<!-- Password Re-enter Label -->
      <!-- Signup Button -->
<android.support.v7.widget.AppCompatButton
    android:id="@+id/btn_signup"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="24dp"
    android:layout_marginTop="24dp"
    android:background="@drawable/button_login_style"
    android:padding="12dp"
    android:text="@string/signup.button.signup"
    android:textColor="@android:color/white"
    android:textStyle="bold" />
<!-- Signin Link-->
<TextView android:id="@+id/link_login"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="24dp"
```

```
            android:text="@string/signup.notice.message.signin"
            android:gravity="center"
            android:textColor="@color/primary_dark"
            android:textStyle="normal|bold" />
    </LinearLayout>
</ScrollView>
```

Para la validación de formularios de interfaz de usuario se utilizó Android Saripaar, una biblioteca basada en reglas sencilla. Para hacer uso de esta librería es necesario añadir la dependencia en el proyecto vía Build.gradle(Module:app)

```
compile 'com.mobsandgeeks:android-saripaar:2.0.2'
```

Las validaciones son de estilo declarativa utilizando anotaciones como @NotEmpty que indica que dicho campo es requerido, @Email que ese campo va a ser de tipo email, entre otros que se pueden observar en el siguiente fragmento de código:

```
public class SignUpActivity extends BaseActivity implements
NavigationView.OnNavigationItemSelectedListener, SignUpView,
Validator.ValidationListener {
    private static final String TAG = "SignUpActivity";
    @NotEmpty(message = "Este campo es requerido.")
    @BindView(R.id.input_lastname_singup)
    EditText inputLastName;
    @NotEmpty(message = "Este campo es requerido.")
    @BindView(R.id.input_name_singup)
    EditText inputName;
    @NotEmpty(message = "Este campo es requerido.")
    @Email(message = "Ingrese un email valido.")
    @BindView(R.id.input_email_singup)
    EditText inputEmail;

    @BindView(R.id.text_input_password)
    TextInputLayout textInputPassword;
    @NotEmpty(message = "Este campo es requerido.")
```

```
    @Password(min = 4, scheme = Password.Scheme.ANY, message = "Contraseña
incorrecta.")
    @BindView(R.id.input_password_singup)
    EditText inputPassword;
    @BindView(R.id.text_input_reEnterPassword)
    TextInputLayout textInputReEnterPassword;
    @NotEmpty(message = "Este campo es requerido.")
    @Password(min = 4, scheme = Password.Scheme.ANY, message = "Contraseña
incorrecta.")
    @BindView(R.id.input_reEnterPassword_singup)
    EditText inputReenterPassword;
    @BindView(R.id.btn_signup)
    AppCompatButton btnSignup;
    @BindView(R.id.link_login)
    TextView signinLink;
{
```

HU.2.1 Login con email y password

En esta HU se mostrará la implementación de MVP y Clean que se utilizará para el desarrollo de toda la aplicación Android como se mencionó en el Capítulo IV.3. MVP permite separar los datos y la lógica de negocio de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Se utiliza MVP para la parte frontal, es decir para la Vista y el Presentador, y para no tener un modelo donde toda la lógica de negocio esté acoplada, se utilizará la clase Interactuador, que va a ser invocado por la clase Presenter y a su vez este invoca a la clase Repository. Por lo tanto, el proyecto tendrá una estructura como la que se puede observar en la figura IV.13.

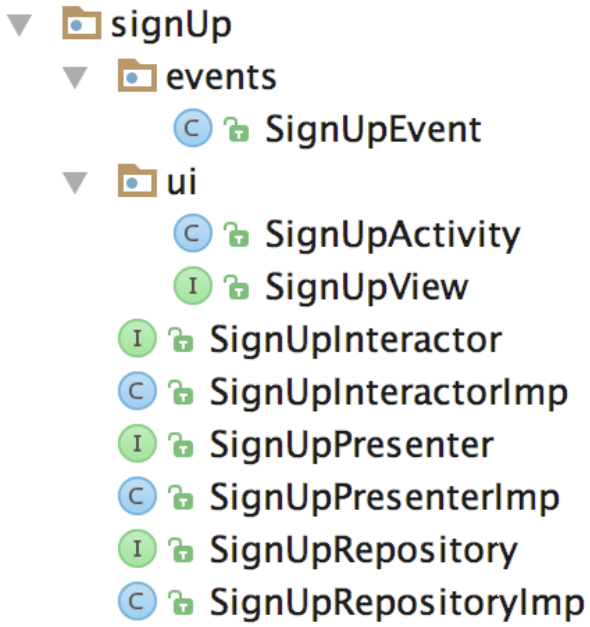


Fig.IV.13. Estructura de la carpeta singUp con MVP y Clean.

Lo primero a tener en cuenta es que el usuario va a interactuar con la vista, la cual no debería tener lógica en cuanto a los que está mostrando y en general va a ser la Actividad, por ejemplo, la vista del login, tendra la interface con los siguientes métodos:

```
public interface SignInView {

   void enableInputs();

   void disableInputs();

   void showProgress();

   void hideProgress();

   void handleSignIn();

   void handleSignUp();

   void navigateToSignUpScreen();

   void loginError(String error);

   void signInSuccess();

   Context getContext();

}
```

La implementación de la interface SignInView está dada por la clase SignInActivity. A continuación, se muestran los métodos e implementarlos:

```
public class SignInActivity extends BaseActivity implements
NavigationView.OnNavigationItemSelectedListener, SignInView,
Validator.ValidationListener {

   @NotEmpty(message = "Este campo es requerido.")

   @Email(message = "Ingrese un email valido.")

   @BindView(R.id.input_email)

   EditText inputEmail;

   @NotEmpty(message = "Este campo es requerido.")

   @Password(min = 4, scheme = Password.Scheme.ANY, message = "Contraseña
incorrecta.")

   @BindView(R.id.input_password)

   EditText inputPassword;

   @BindView(R.id.btn_signin)

   AppCompatButton btnSignin;

   @BindView(R.id.link_signup)

   TextView signupLink;

   @BindView(R.id.progressBar)

   ProgressBar progressBar;


   private SignInPresenter signInPresenter;

     @Override

   protected void onCreate(Bundle savedInstanceState) {

       super.onCreate(savedInstanceState);

       setContentView(R.layout.activity_main);

       ButterKnife.bind(this);

       signInPresenter = new SignInPresenterImp(this);

       signInPresenter.onCreate();

    }

   @Override

   public void enableInputs() {

       setInputs(true);

   }
```

```
    @Override
    public void disableInputs() {
        setInputs(false);
    }
    @Override
    public void showProgress() {
        progressBar.setVisibility(View.VISIBLE);
    }
    @Override
    public void hideProgress() {
        progressBar.setVisibility(View.GONE);
    }
    @Override
    public void handleSignIn() {
        signInPresenter.validateLogin(inputEmail.getText().toString(),
inputPassword.getText().toString());
    }
       @Override
    public void handleSignUp() {
        this.navigateToSignUpScreen();
    }
    @Override
    public void loginError(String error) {
        inputPassword.setText("");
        Toast.makeText(getBaseContext(),                    error.toString(),
Toast.LENGTH_LONG).show();
    }
    @Override
    public void navigateToSignUpScreen() {
        Intent intent = new Intent(this, SignUpActivity.class);
        intent.putExtra("signUpWith", Constants.SIGNIN_EMAIL);
        startActivity(intent);
```

```
    }
    @Override
    public void signInSuccess() {
        navigateToMainScreen();
    }
    private void setInputs(boolean enabled) {
        inputEmail.setEnabled(enabled);
        inputPassword.setEnabled(enabled);
        btnSignin.setEnabled(enabled);
    }
    @Override
    public Context getContext() {
    return this.getApplicationContext();
    }
  }
```

Como se observa en el código anterior, SignInActivity tendrá una instancia de la clase Presenter, de esta manera cuando la vista necesite algo va a acudir a este. El Presenter podrá acceder a métodos tanto de la vista como del interactuador y tendrá la siguiente interfaz:

```
public interface SignInPresenter {
    void onCreate();
    void onDestroy();
    void validateLogin(String email, String password);
    void onEventMainThread(SignInEvent event);
}
```

En la implementación de la clase SignInPresenter se puede identificar la comunicación con el repositorio. Este se comunicara a su vez con la vista, como es el caso del método onEventMainThread() dependiendo el caso envía un mensaje de

error y habilita los input nuevamente o le da a la vista la orden de que pase a otra pantalla:

```
public class SignInPresenterImp implements SignInPresenter {
   private EventBus eventBus;
   private SignInView signInView;
   private SignInInteractor signInIteractor;
   public SignInPresenterImp(SignInView loginView){
       this.signInView = loginView;
       this.eventBus = GreenRobotEventBus.getInstance();
       this.signInIteractor = new SignInInteractorImp();
   }
   @Override
   public void validateLogin(String email, String password) {
       if (signInView != null){
           signInView.disableInputs();
           signInView.showProgress();
           signInIteractor.doSignIn(email, password, signInView.getContext());
       }
   }
   private void onSignInSuccess(){
       if (signInView != null){
           signInView.signInSuccess();
       }
   }
   private void onSignInError(String error){
       if (signInView != null){
           signInView.hideProgress();
           signInView.enableInputs();
           signInView.loginError(error);
       }
```

```
    }
    @Override
    public void onCreate() {
        eventBus.register(this);
    }
    @Override
    public void onDestroy() {
        signInView = null;
        eventBus.unregister(this);
    }
    @Override
    @Subscribe
    public void onEventMainThread(SignInEvent event) {
        switch (event.getEventType()) {
            case SignInEvent.onSignInError:
                onSignInError(event.getErrorMesage());
                break;
            case SignInEvent.onSignInSuccess:
                onSignInSuccess();
                break;
        }
    }
}
```

El Presentador además de tener una instancia de la vista, tendrá una instancia del interactuador, este va a incluir en su implementación una instancia del repositorio a través de la cual se ejecutará algún método para obtener datos que se puedam devolver a la vista. EventBus debe registrarse en función del ciclo de vida de la actividad, es decir se registrará en el método onCreate y se anulará en el onDestroy. El método suscripto es onEventMainThread() ya que tiene la anotación @Subscribe y será llamado cuando se registre un nuevo evento.

El código de la inteface del interactuador será:

```
public interface SignInInteractor {

   void doSignIn(String email, String password, Context context);

}
```

Y su implementación:

```
public class SignInInteractorImp implements SignInInteractor {

   private static final String TAG = "SignInInteractorImp";

   private SignInRepository signInRepository;


   public SignInInteractorImp(){

       signInRepository = new SignInRepositoryImp();

   }

   @Override

   public void doSignIn(String email, String password, Context context) {

       signInRepository.signIn(email, password, context); }
```

El repositorio va a concluir, tiene que reportarle de regreso al presentador y este a la vista. Para esto trabajamos con EventBus que permite realizar la comunicación desde el repositorio hacia el presentador, donde el repositorio emite un evento, el presentador lo captura y realiza alguna acción sobre la vista. El repositorio también tiene su interface:

```
public interface LoginRepository {

   void checkSession(Context context);

}
```

En su implementación, se puede observar el método signIn(email,password) que se encarga de iniciar sesión con el email y password pasado como parámetro. Si el resultado de esta operación es positivo devuelve a la clase LoginPresenter los datos del usuario logueado, si no devuelve un mensaje de error. Esta devolución se realiza utilizando el método eventBus.post(event), que publica un evento. Esto hará que el método onEventMainThread del presentador que está registrado lo reciba y ejecute la acción correspondiente.

```
public class SignInRepositoryImp implements SignInRepository {
   private ApiClient apiClient;
   private UserService apiService;
   private ProfileService profileService;

   public SignInRepositoryImp() {
       apiClient = new ApiClient();
       apiService = apiClient.getUserService();
       profileService = apiClient.getProfileService();
   }
   @Override
   public void signIn(final String email, final String password, final Context
context) {
       final SessionPrefs sessionPrefs = SessionPrefs.get(context);
       LoginBody loginBody = new LoginBody(email, password);
       Call<Token> call = apiService.Auth(loginBody);
       call.enqueue(new Callback<Token>() {
        @Override
        public void onResponse(Call<Token> call, Response<Token> response) {
               int statusCode = response.code();
               final Token token = response.body();
              if (response.isSuccessful()) {
                   sessionPrefs.setToken(token);
                  Call<Profile>                        call2                        =
profileService.getProfile(sessionPrefs.getToken().getToken());
                   call2.enqueue(new Callback<Profile>() {
                 @Override
              public    void    onResponse(Call<Profile>    call2,    Response<Profile>
response) {
                          int statusCode = response.code();
                          if (response.isSuccessful()) {
                              Profile profile = response.body();
```

```
                                sessionPrefs.setProfile(profile);
                                postEvent(SignInEvent.onSignInSuccess);
                            }else {
                postEvent(SignInEvent.onSignInError, "Usuario o Password Incorrecto");
                                }
                            }
                        @Override
                        public void onFailure(Call<Profile> call2, Throwable t) {
    postEvent(SignInEvent.onSignInError, "Failure " + t.getLocalizedMessage());
                            }
                        });
                } else {
        postEvent(SignInEvent.onSignInError, "Usuario o Contraseña Incorrecta");
                }
            }
        });
       }
    private void postEvent(int type, String errorMessage) {
        SignInEvent signInEvent = new SignInEvent();
        signInEvent.setEventType(type);
        if (errorMessage != null) {
            signInEvent.setErrorMesage(errorMesage);
        }
        EventBus eventBus = GreenRobotEventBus.getInstance();
        eventBus.post(signInEvent);
   }
}
```

Para terminar con la implementación de MVP y Clean se puede ver en la figura IV.14 el caso donde la clase SignInRepository devuelve un mensaje de error a la clase SignInPresenter y este le indica a la vista que imprima dicho mensaje.

Fig.IV.14. Pantalla de Login.

HU.2.2 Login con Facebook

Todas las clases correspondientes a la implementación de esta HU estarán agrupadas en la carpeta login, por lo tanto, el proyecto tendrá una estructura como la que se puede observar en la figura IV.15.

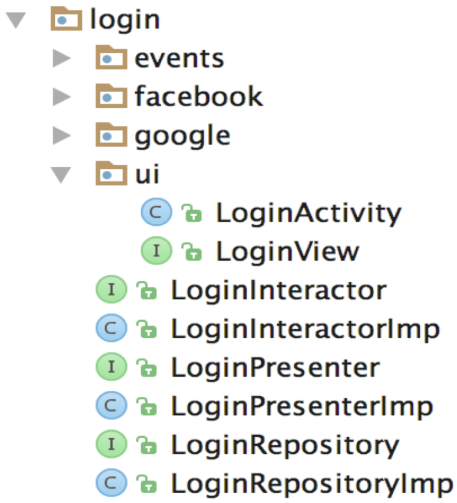


Fig.IV.15. Estructura de la carpeta login con MVP y Clean.

Para esta HU se utilizó la SDK de Facebook para Android, el inicio de sesión con Facebook tiene integración automática con Facebook Lite. Si las personas no tienen

instalada la aplicación de Facebook para Android, el inicio de sesión usa Facebook Lite para mostrar la pantalla de inicio de sesión y obtener las credenciales.

En el proyecto, en el archivo build.gradle (module: app) se agrega la siguiente instrucción de compilación a la sección dependencies para compilar la versión más reciente del SDK del inicio de sesión con Facebook:

```
compile 'com.facebook.android:facebook-android-sdk:4.26.0'
```

En el archivo Manifest se agregó la meta-data, donde @string/facebook_app_id indicara el nombre de la aplicación que se creo en la plataforma de Facebook:

```
<meta-data
   android:name="com.facebook.sdk.ApplicationId"
   android:value="@string/facebook_app_id" />
```

El modo más sencillo de agregar el inicio de sesión con Facebook a la aplicación es agregar LoginButton desde el SDK. El botón LoginButton es un elemento de la interfaz de usuario que reúne funciones disponibles en LoginManager, de modo que cuando alguien hace click en el botón, el inicio de sesión se inicia con los permisos establecidos en LoginManager. El botón controla el estado de inicio de sesión y muestra el texto en función del estado de autenticación.

Para agregar el botón de inicio de sesión con Facebook, primero se debe agregar al archivo XML de diseño con el nombre completo de la clase, com.facebook.widget.LoginButton:

```
<com.facebook.login.widget.LoginButton
   android:id="@+id/btnLoginFacebook"
   android:layout_width="match_parent"
 />
```

La aplicación solo puede iniciar la sesión de una persona a la vez, y LoginManager establece el AccessToken y el Profile actual para esa persona. El SDK de Facebook guarda estos datos en las preferencias de uso compartido y los configura al principio de la sesión. Por lo tanto en el método signIn() que se implementa en la clase LoginRepositoryFacebbokImp, se comprobara si una persona ya tiene sesión

iniciada mediante AccessToken.getCurrentAccessToken() y Profile.getCurrentProfile().

Luego la Clase GraphRequest tiene un método newMeRequest que llama al extremo /user/me para recuperar los datos de usuario del token de acceso correspondiente. Se podrá obtener datos de la persona logueada con facebook como name, first_name, birthday, last_name, email, etc. Con estos datos se loguea a la persona en nuestra app, para esto primero se verifica con el email si ya tiene una cuenta creada, si no se envía a completar el formulario de registro con dichos datos precargados.

```
public void signIn(final Context context) {
   final SessionPrefs sessionPrefs = SessionPrefs.get(context);
   GraphRequest request = GraphRequest.newMeRequest(
           AccessToken.getCurrentAccessToken(),
           new GraphRequest.GraphJSONObjectCallback() {
               @Override
               public void onCompleted(
                       JSONObject object,
                       GraphResponse response) {
                   try {
                       final String first_name = object.getString("first_name");
                       final String last_name = object.getString("last_name");
                       final String email = object.getString("email");
                       final String birthday = object.getString("birthday");
                       final String avatar =
object.getJSONObject("picture").getJSONObject("data").getString("url");

                       //Comprobar que esté en la BD y sinó agregarlo
                       dataMovil.setText(email);
                       Call<User> call = apiService.findByEmail(dataMovil);

                       call.enqueue(new Callback<User>() {
                           @Override
                public void onResponse(Call<User> call, Response<User> response) {
                               int statusCode = response.code();
                               User userApi = response.body();
                               if (response.isSuccessful()) {
                                  signIn(userApi, context);

                               } else {//Si no existe usuario
                                   User user = new User();
                                   user.setEmail(email);
                                   user.setRegisterWith(Constants.SIGNIN_FACEBOOK);
                                   Profile profile = new Profile();
                                   profile.setFirstname(first_name);
                                   profile.setLastname(last_name);
                                if (avatar != null) {
                                       profile.setProfilePhoto(avatar);
                                   }
                               user.setProfile(profile);
                               postEvent(FacebookEvent.signupRequired,user,profile);
```

```
                                }
                            }

                            @Override
                            public void onFailure(Call<User> call, Throwable t) {
                                Log.d(TAG, t.getLocalizedMessage());
                                sessionPrefs.logOut();
                        postEvent(FacebookEvent.onLoginError, t.getLocalizedMessage());
                                // Log error here since request failed
                            }
                        });
                        //Fin comprobar BD
                    } catch (JSONException e) {
                        e.printStackTrace();
                        postEvent(FacebookEvent.onLoginError,
e.getLocalizedMessage());
                    }
                }
            });
    Bundle parameters = new Bundle();
    parameters.putString("fields",
"id,name,first_name,birthday,picture.type(large),last_name,email");
    request.setParameters(parameters);
    request.executeAsync();
}
```

HU.2.3 Login con Google

Para integrar Google Sign-In en la aplicación, hay que configurar el proyecto en Android Studio y agregar un archivo de configuración el cual proporciona información específica del servicio para la aplicación. Para obtenerlo, es necesario crear un proyecto en la consola de desarrolladores de Google y proporcionar el nombre del paquete de la aplicación que se encuentra en el archivo Manifest.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

package="ar.edu.unrn.lia.loginapp"
```

Además de crear el archivo de configuración, se proporcionó el hash SHA-1 del certificado de firma. Luego se copió el archivo google-services.json, que se descargó desde la consola de desarrolladores, al directorio app/ del proyecto Android Studio, como se muestra en la figura IV.16.

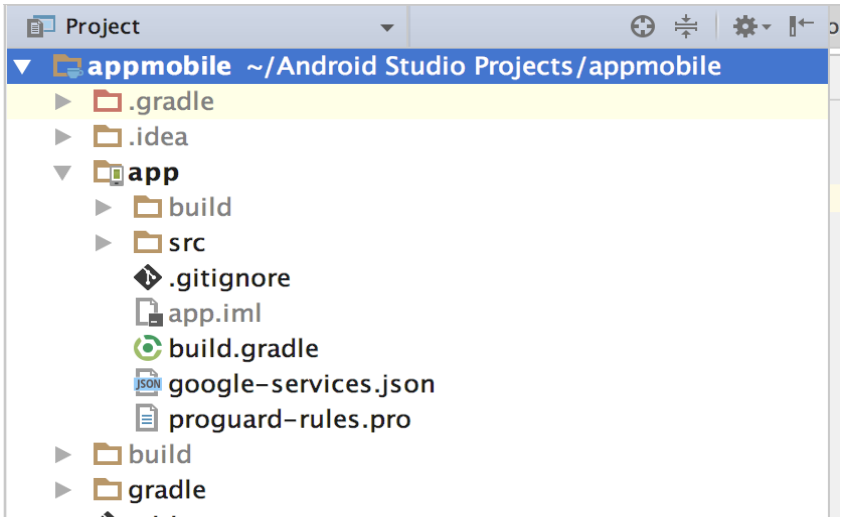


Fig.IV.16. Estructura del proyecto Android.

Por último se agregó el complemento al proyecto actualizando los archivos de build.gradle a nivel proyecto y de aplicación como sigue:

build.gradle de proyecto

```
classpath 'com.google.gms:google-services:3.0.0'
```

build.gradle de aplicación

```
apply plugin: 'com.google.gms.google-services'
```

Después de realizar todas las configuraciones necesarias se agregó el botón de inicio de sesión de Google a la aplicación:

```
<com.google.android.gms.common.SignInButton
   android:id="@+id/btnLoginGoogle"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
 />
public class LoginPresenterGoogleImp implements LoginPresenterGoogle,
GoogleApiClient.OnConnectionFailedListener {
    private GoogleApiClient mGoogleApiClient;
   @Override
   public void createGoogleClient(LoginActivity loginActivity) {
       mGoogleApiClient = GoogleApiBase.getInstance(loginActivity);
```

```
    }
   @Override
    public void onStart() {
        mGoogleApiClient.connect();
    }
   @Override
    public void signIn(LoginActivity loginActivity) {
        Intent signInIntent =
Auth.GoogleSignInApi.getSignInIntent(mGoogleApiClient);
        loginActivity.startActivityForResult(signInIntent, RC_SIGN_IN_G);
    }
@Override
public void onActivityResult(LoginActivity loginActivity, int requestCode, int
resultCode, Intent data) {
     if (requestCode == RC_SIGN_IN_G) {
        GoogleSignInResult result =
Auth.GoogleSignInApi.getSignInResultFromIntent(data);
        handleSignInResult(result, loginActivity);
    }
}
```

En el método signIn se inicia el intento, donde se le solicita al usuario que seleccione una cuenta de Google para iniciar sesión, como se puede observar en la figura IV.17.

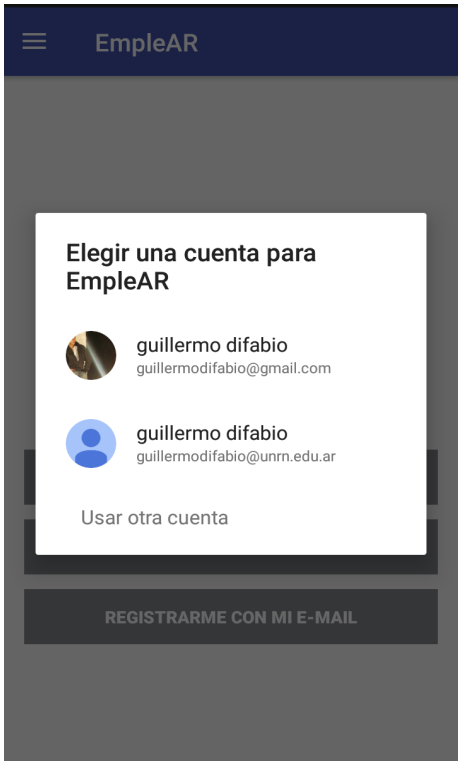


Fig.IV.17. Pantalla de login con Google.

En el método onActivityResult de la actividad se recupera el resultado de inicio de sesión con getSignInResultFromIntent. Si el inicio de sesión se realizó correctamente, se llamará al método getSignInAccount para obtener un objeto GoogleSignInAccount que contiene información sobre el usuario, como el nombre, foto de perfil, apellido, email, etc. En el siguiente fragmento de código se puede observar que el método signIn recibe como parámetro un objeto GoogleSignInAccount :

```
@Override
public void signIn(GoogleSignInAccount acct, final Context context) {
   final SessionPrefs sessionPrefs = SessionPrefs.get(context);
   final String email = acct.getEmail();
   final String first_name = acct.getGivenName();
   final String last_name = acct.getFamilyName();
   final String avatar;
   avatar = acct.getPhotoUrl().toString();
```

```
    dataMovil.setText(email);
    Call<User> call = apiService.findByEmail(dataMovil);
    call.enqueue(new Callback<User>() {
        @Override
        public void onResponse(Call<User> call, Response<User> response) {
            int statusCode = response.code();
            User userApi = response.body();
             if (response.isSuccessful()) {
                 signIn(userApi, context);
            } else {//Si no existe usuario
                 User user = new User();
                user.setEmail(email);
                Profile profile = new Profile();
                profile.setFirstname(first_name);
                profile.setLastname(last_name);
                 if (avatar != null) {
                    profile.setProfilePhoto(avatar);
                }
                user.setProfile(profile);
                postEvent(GoogleEvent.signupRequired, user, profile);
            }
        }
        @Override
        public void onFailure(Call<User> call, Throwable t) {
            sessionPrefs.logOut();
            postEvent(GoogleEvent.onLoginError, t.getLocalizedMessage());
              }
    });
}
```

Con estos datos se loguea a la persona en la app, para esto primero se verifica con el email si ya tiene una cuenta creada, si no se envía a completar el formulario de registro con dichos datos precargados como se puede ver en la figura IV.18.

Fig.IV.18. Pantalla de Registro con Google.

Luego de completar el registro se procede a mostrar el usuario logueado en la aplicación con su nombre, apellido, foto de perfil y todas las opciones del menú que tiene habilitadas como se puede ver en la figura IV.19.

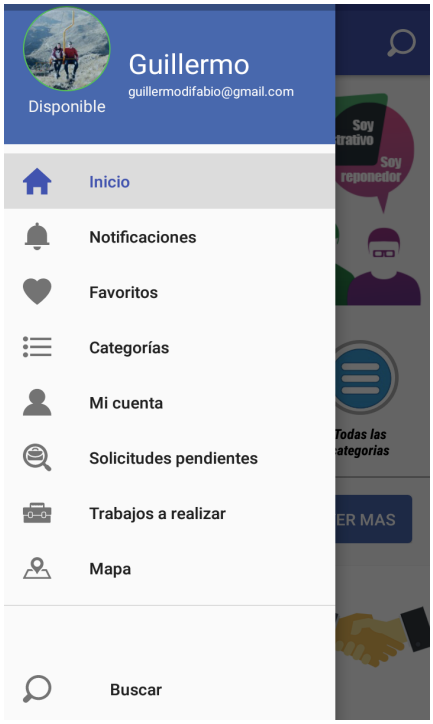


Fig.IV.19. Menú de navegación.

Sprint 5

Para el seguimiento del Sprint, se implementó el siguiente tablero de Trello, en el cual se pueden ver en detalle todas las tareas de las HU planteadas para esta interacción. Aquí se desarrollaron las tareas relacionadas al Mapa de servicios.

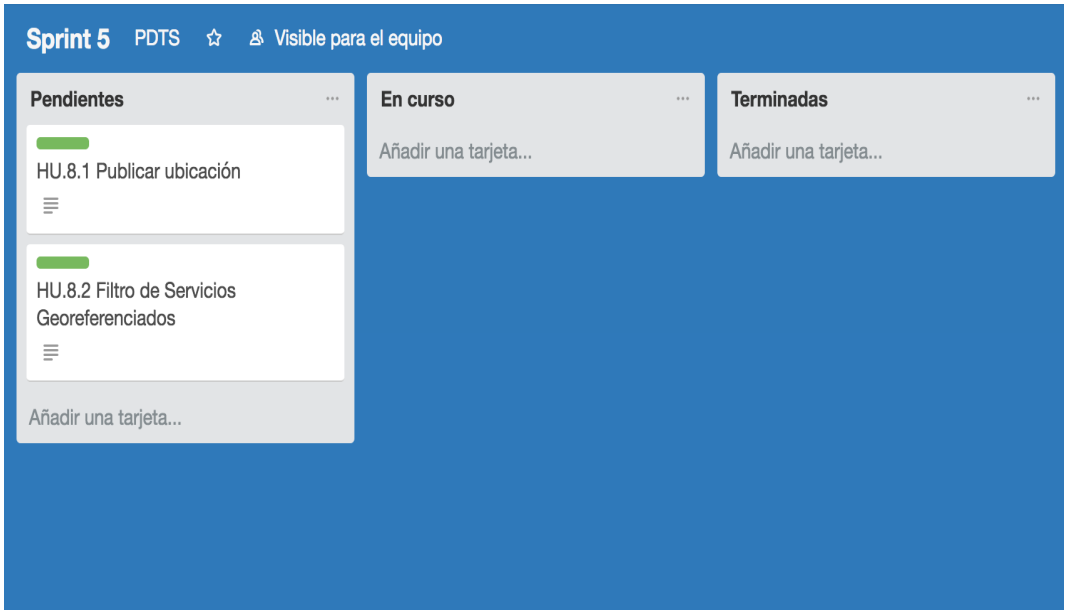


Fig.IV.20. Tareas de la HU 8 del Tablero Scrum.

HU.8 Mapa de Servicios Georreferenciados

Para resolver esta HU se utilizó la API de ubicación disponible en los servicios de Google Play, que facilitan la incorporación de conocimientos de ubicación a la aplicación con el seguimiento automatizado de ubicaciones, la geocodificación y el reconocimiento de actividades.

HU.8.1 Publicar ubicación

En este caso en particular se utilizó el método que permite recuperar la última ubicación conocida de un dispositivo Android, que suele ser equivalente a la ubicación actual del usuario. La aplicación deberá solicitar permisos de ubicación para utilizar los servicios de ubicación. Android ofrece dos permisos de ubicación: ACCESS_COARSE_LOCATION y ACCESS_FINE_LOCATION. El permiso que se elija determina la exactitud de la ubicación devuelta por la API. Para esta HU se requiere que la ubicación sea lo más exacta posible, por lo tanto, se solicitó el siguiente permiso con el elemento uses-permission en el manifiesto de aplicación:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android

   package="ar.edu.unrn.lia.loginapp">

   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

</manifest>
```

Pero esto no es todo, a partir de Android 6.0 (nivel de API 23), los usuarios conceden permisos a las apps mientras se ejecutan, no cuando instalan la app. Si esta no tiene el permiso que necesita, debe llamar al método requestPermissions() para solicitar los permisos correspondientes. Este funciona de manera asincrónica, realiza la devolución inmediatamente y, cuando el usuario responde al cuadro de diálogo, el sistema llama al método callback de la app con los resultados y pasa el mismo código de solicitud que la app le pasó a requestPermissions(). El siguiente código verifica en el método onConnected si la app tiene permiso para acceder la ubicación del teléfono y solicita el permiso si es necesario:

```
@Override
 public void onConnected(@Nullable Bundle bundle) {
     //Android 6.0 permisos
```

```
        if (ActivityCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this,
                    new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION,
android.Manifest.permission.ACCESS_COARSE_LOCATION}, REQUEST_LOCATION);
        }else {
            createLocationRequest();

LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
mLocationRequest, this);
        }

    }
```

Al hacerlo en el método onConnected de la actividad MapsActivity el diálogo aparecerá ni bien arranca la actividad como se ve en la figura IV.21.

Fig.IV.21. Diálogo de Permisos.

Cuando el usuario responde, el sistema invoca el método onRequestPermissionsResult() y le pasa la respuesta del usuario.

```
    @Override
    public void onRequestPermissionsResult(int requestCode, String permissions[],
int[] grantResults) {
        switch (requestCode) {
```

```
            case REQUEST_LOCATION: {
                // If request is cancelled, the result arrays are empty.
                if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                    createLocationRequest();

LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
mLocationRequest, this);


                }
                }
        }
    }
```

Se utilizó la clase BaseLocation que encapsula todos los métodos de Location.

```
public abstract class BaseActivityLocation extends BaseActivity implements
        GoogleApiClient.ConnectionCallbacks,
        GoogleApiClient.OnConnectionFailedListener,
        LocationListener {

    private GoogleApiClient mGoogleApiClient;
    private LocationRequest mLocationRequest;
    public static final int REQUEST_LOCATION = 2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TAG = this.getClass().getSimpleName();
        setupGoogleAPIClient();
    }

    private void setupGoogleAPIClient() {
        mGoogleApiClient = new GoogleApiClient.Builder(this)
                .addApi(LocationServices.API)
                .addConnectionCallbacks(this)
                .addOnConnectionFailedListener(this)
                .build();
    }

    private void createLocationRequest() {
        mLocationRequest = LocationRequest.create();
        mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
        mLocationRequest.setInterval(60000);// Update location every second
             mLocationRequest.setSmallestDisplacement(10);    }
    @Override
    protected void onStart() {
        super.onStart()
        mGoogleApiClient.connect();
    }
    @Override
    protected void onStop() {
        super.onStop();
        if (mGoogleApiClient.isConnected()) {
            mGoogleApiClient.disconnect();
        }
    }
```

```
    @Override
    public void onConnectionSuspended(int i) {
        mGoogleApiClient.connect();
    }

    public abstract void onConnectionFailed(@NonNull ConnectionResult
connectionResult);

    public abstract void onLocationChanged(Location location);
}
```

El método abstracto onLocationChanged será implementado en la clase que desee realizar alguna acción cuando la posición de GPS cambie, por ejemplo, MapActivity. En dicho método se verifica si fue concedido el permiso para acceder a la posición del teléfono y de ser así se habilita el botón que permite compartir la ubicación actual del usuario logueado como se observa en el siguiente código:

```
public class MapsActivity extends BaseActivityLocation implements MapsView,
OnMapReadyCallback, GoogleMap.InfoWindowAdapter {

    private static final int COD_FILTER = 1;
    private SessionPrefs sessionPrefs;
    private User user;
    private Profile profile;
    private GoogleMap mMap;
    private MapsPresenter mapsPresenter;

@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);

    //Initialize Google Play Services
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
            mMap.setMyLocationEnabled(true);

        }
    } else {
        mMap.setMyLocationEnabled(true);
    }
    mMap.setOnMarkerClickListener(new GoogleMap.OnMarkerClickListener() {
        @Override
        public boolean onMarkerClick(Marker marker) {
                  return false;
        }
    });
```

```
    mMap.setOnInfoWindowClickListener(new GoogleMap.OnInfoWindowClickListener() {
        @Override
        public void onInfoWindowClick(Marker marker) {

            String[] profile = marker.getTitle().split("▪");
            Intent intent = new Intent(MapsActivity.this, ProfileActivity.class);
            String profileId = profile[0];
            String firstName = profile[1] + " " + profile[2];
            String profilePhoto = profile[3];
            intent.putExtra("firstName", firstName);
            intent.putExtra("profileId", Long.valueOf(profileId));
            intent.putExtra("profilePhoto", profilePhoto);
            startActivity(intent);
        }
    });
}

@Override
public void onLocationChanged(Location location) {

    final ar.edu.unrn.lia.loginapp.api.model.Location loc = new
ar.edu.unrn.lia.loginapp.api.model.Location();
    loc.setLatitude(location.getLatitude());
    loc.setLongitude(location.getLongitude());
    final LatLng latLng = new LatLng(location.getLatitude(),
location.getLongitude());
    mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
    float zoom = (mMap.getCameraPosition().zoom);
    animCam(zoom);
    if (isLogin()) floatingActionButton.setVisibility(View.VISIBLE);
    floatingActionButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(getApplicationContext(), "Compartiendo ubicación" +
latLng.toString(), Toast.LENGTH_LONG).show();
            mapsPresenter.saveLocation(loc);
        }
    });
    sessionPrefs.setLat(location.getLatitude());
    sessionPrefs.setLon(location.getLongitude());
    if (sessionPrefs.getDistance() != null && sessionPrefs.getDistance() != 0.0) {
        mMap.clear();//Borro marcadores
        float zoom2 = 17;
        zoom2 = (float) (zoom2 - (sessionPrefs.getDistance() / 2.0));
        animCam(zoom2);
        mapsPresenter.getProfiles(createFilter(location.getLatitude(),
location.getLongitude(), sessionPrefs.getDistance(),
sessionPrefs.getSubCategory()));

    }
}}
```

El resultado obtenido luego de ejecutar el código se muestra en la figura IV.22.

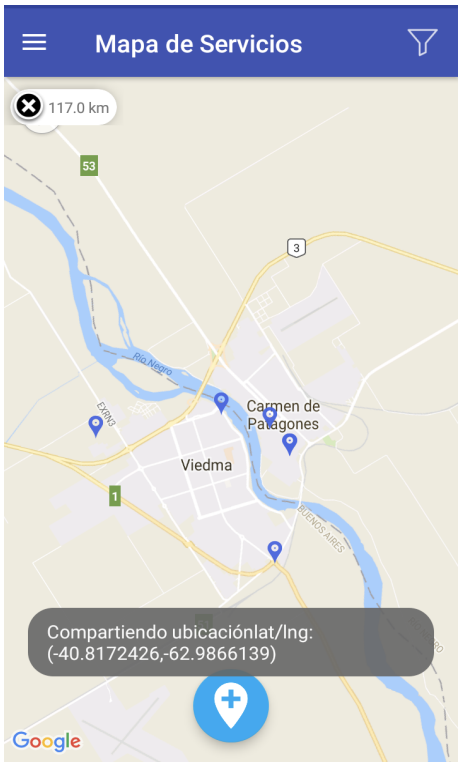


Fig.IV.22. Compartiendo ubicación en mapa de servicios georeferenciados.

Sprint 6

Para el seguimiento del Sprint, se implementó el siguiente tablero de Trello, en el cual se pueden ver en detalle todas las tareas de las HU planteadas para esta interacción. A continuacion se muestran las tareas de la Gestión de Contratación y además se aprovechó dicho sprint para agregar y terminar ciertas funcionalidades.

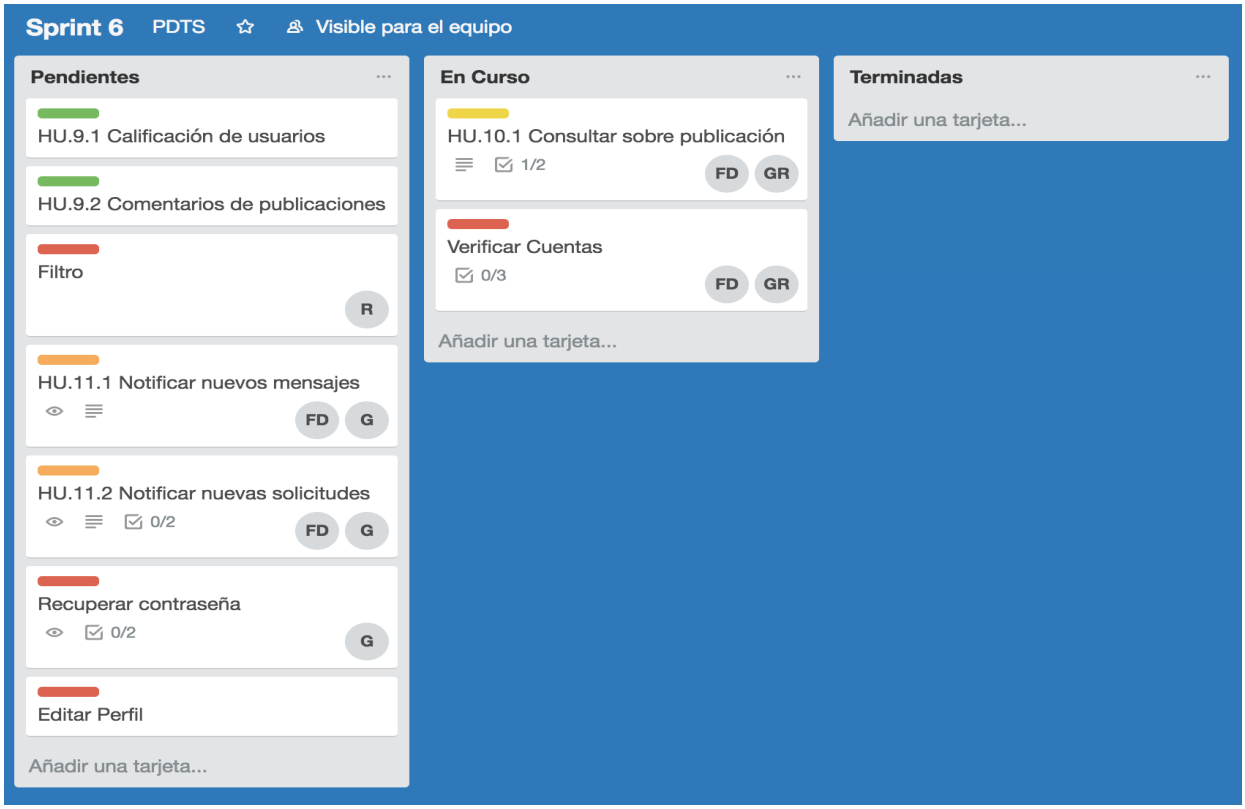


Fig.IV.23. Tareas de la HU 9 del Tablero Scrum.

HU.9 Gestión de Contratación

HU.9.2 Proceso de contratación

En esta HU se desarrolló el proceso de contratación y todos sus estados. En la figura IV.24 se puede ver el Diagrama de estado de la contratación, donde los últimos estados posibles son el de finalizada o cancelada.

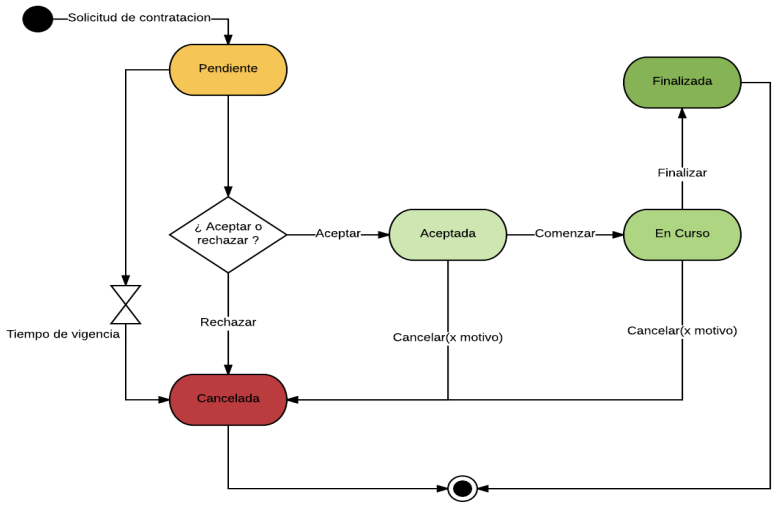


Fig.IV.24. Diagrama de estado de una contratación.

HU.10 Servicio de Notificaciones

En esta HU fue necesario configurar Firebase y el SDK de FCM, que permite enviar y recibir mensaje y notificaciones en la aplicación.

Lo primero fue crear un proyecto en la consola de desarrollo de Firebase, luego se descargó el archivo de configuración que esta genera y para finalizar el proceso, se agregó al módulo del proyecto, es decir en /app. Después de configurar FCM en el proyecto se agregó la dependencia al archivo build.gradle a nivel de app:

```
compile 'com.google.firebase:firebase-messaging:11.0.4'
```

Se creó en el manifiesto de la aplicación un servicio que extiende de FirebaseMessagingService. Esto es obligatorio para administrar los mensajes además de recibir notificaciones en la app en segundo plano.

```
<service
   android:name=".MyFirebaseMessagingService">
   <intent-filter>
       <action android:name="com.google.firebase.MESSAGING_EVENT"/>
   </intent-filter>
</service>
```

En la figura IV.25 se puede ver la notificación en la barra de notificaciones al recibir una solicitud de contratación.

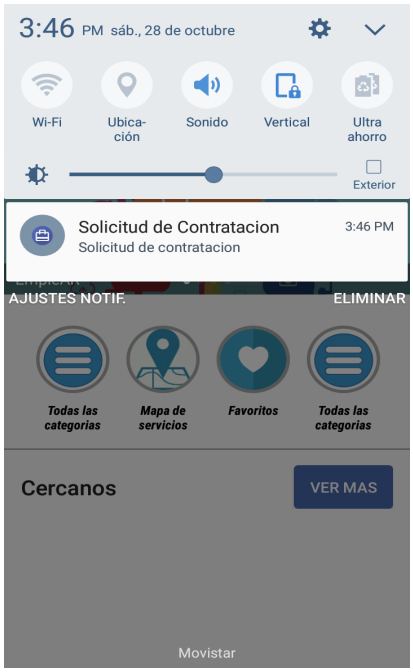


Fig.IV.25. Notificación de solicitud de Contratación.

V. Verificación de resultado

En este capítulo se desarrolló un conjunto de pruebas de verificación, validación y stress que permitieron determinar el funcionamiento completo de la aplicación. El propósito principal de la evaluación es encontrar errores y defectos del sistema a fin de corregirlos. Es importante comprobar que el sistema cumpla con los requerimientos establecidos por el usuario y tenga un rendimiento adecuado en el ambiente donde se encuentre instalado. Otro aspecto importante por evaluar, son las características de seguridad relacionadas con el ingreso no autorizado de usuarios, de manera que estos no puedan realizar modificaciones donde no son permitidas.

V.1 Condiciones generales de la prueba

El backend está alojada en un servidor CRX con procesador Intel Xeon y 16 Gram. Este servidor es equipamiento de Laboratorio de Informática Aplicada (LIA) de la Universidad Nacional de Río Negro (UNRN).

Para la aplicación Android se utilizó el celular Samsung Galaxy J5 Prime con Android 6.0.1, el cual se conectó al servidor mediante servicio de 4G de la empresa Movistar.

V.2 Pruebas de verificación y validación

Se convocaron alumnos de la carrera Licenciatura en Sistemas que no tenían conocimiento alguno sobre la aplicación. Con el propósito de verificar y validar el funcionamiento del prototipo presentado en el Capítulo IV, se mostrará un ejemplo hipotético. Con el propósito de verificar y validar el funcionamiento del prototipo presentado en el Capítulo IV, se mostrarán funcionalidades como: registro de usuario y autenticación.

A continuación, se detallarán los casos de pruebas realizados:

V.2.1 Prueba de registro de usuario

Con este caso de prueba se pretende mostrar el funcionamiento del registro de usuario a la aplicación. Para ello el usuario deberá completar un formulario con los datos requeridos por el sistema para generar el registro. Este método verifica si el email ingresado no está siendo usado por otro usuario, de manera tal que, si dicho mail no existe en la base de datos, permitirá continuar con el registro.

Para probarlo se convocó al alumno Nicolás Castro, quien procedió a realizar la carga de los datos solicitados por la app como se puede observar en la figura V.1.

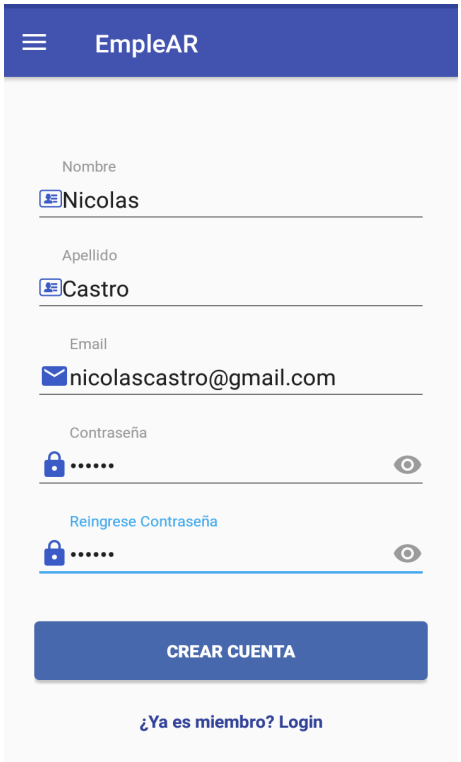


Fig.V.1. Registro de usuario.

V.2.1.1 Resultados obtenidos

Como se esperaba, los resultados fueron correctos según el conocimiento empírico previo. Durante el desarrollo de la prueba se pudo observar que el registro fue exitoso ya que Nicolás Castro no se encontraba registrado con el email nicolascastro@gmail.com en la aplicación.

V.2.2 Prueba de registro de usuario con email existente

Con esta prueba se pretende mostrar el caso contrario al anteriormente desarrollado. Es decir, el caso donde el usuario a registrar ingresa un mail que está siendo utilizado por otro, por lo tanto, el sistema deberá informar este error.

V.2.2.1 Resultados obtenidos

Como se esperaba, los resultados fueron correctos, la aplicación no dejó continuar con el proceso de registro y devolvió un mensaje “email en uso”.

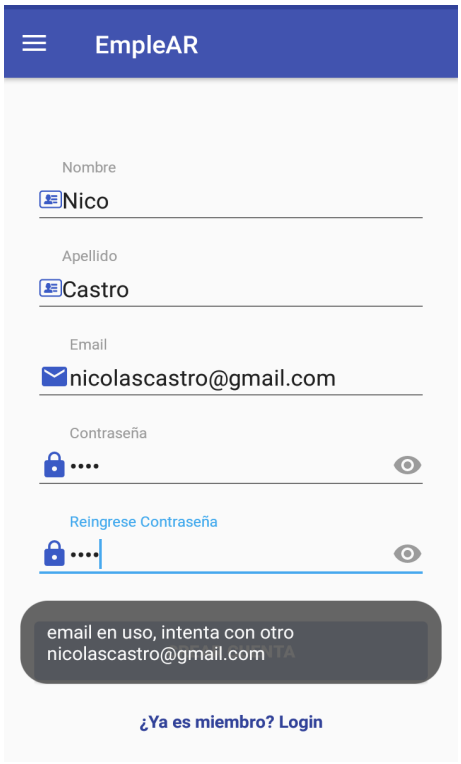


Fig.V.2. Mensaje de email en uso.

V.2.3 Prueba sobre mostrar ítem del menú con usuario logueado

En este caso se probaron las funcionalidades que necesitan de un ingreso autorizado. Al iniciar sesión en la aplicación el usuario deberá tener más opciones del menú que cuando ingresa como invitado.

V.2.3.1 Resultados obtenidos

Como se esperaba, los resultados fueron correctos, luego de pasar el proceso de verificación de credenciales, el sistema le otorgó acceso al usuario y como resultado este puede encontrar todas las opciones del menú que tiene

habilitadas. A continuación, se muestran las figuras Fig.V.3 y V.4 donde se puede ver el menú antes de la prueba y después de la prueba respectivamente.

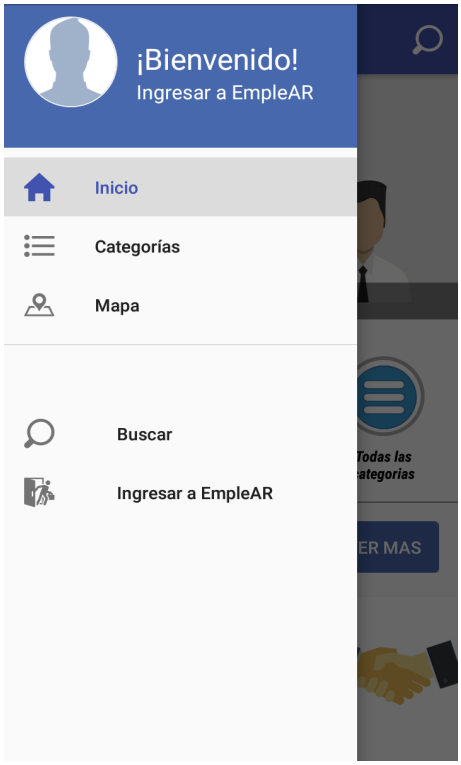


Fig.V.3. Menú de usuario invitado.

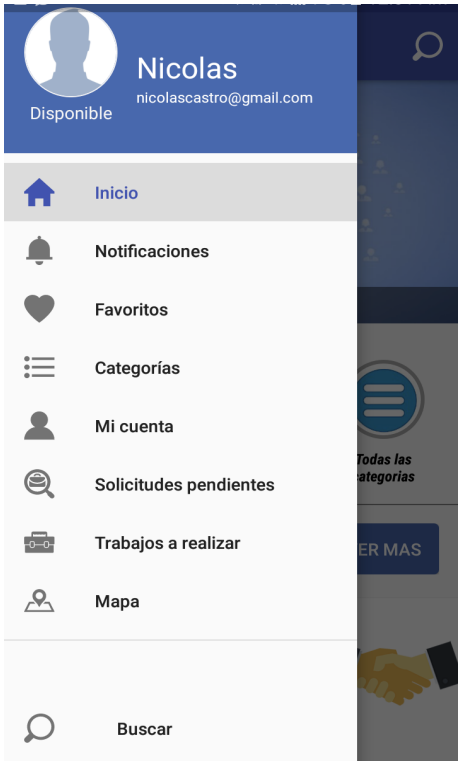


Fig.V.4. Menú de usuario logueado.

En la figura V.4 se puede observar que el usuario tiene los ítems habilitados que debería tener un usuario logueado.

V.3 Pruebas de rendimiento

Con la intención de medir la capacidad de carga y realizar pruebas de rendimiento para la aplicación, se utilizó la herramienta Apache JMeter. Esta es un software de código abierto, diseñado para ser utilizado como prueba de carga para analizar y medir el desempeño de una variedad de servicios, con énfasis en aplicaciones web.

Cabe aclarar, que dicha herramienta se ejecuta sobre un pc, por lo que para llevar a cabo las pruebas con una conexión similar a la que se cuenta en un smartphone, 4G, fue necesario realizar desde el dispositivo móvil, un anclaje de red con el objetivo de realizar las pruebas en condiciones que más se ajustan a la realidad.

En la figura Fig.V.5 se puede ver la interfaz de usuario de JMeter y la configuración del Grupo de Hilos, donde se pueden personalizar los números de solicitudes a enviar. En este caso se definimos "Número de subprocesos = 1000", "Periodo de subida = 1" y "Conteo de bucle = 1". Esto permite simular 1000 solicitudes diferentes cada un segundo por única vez.

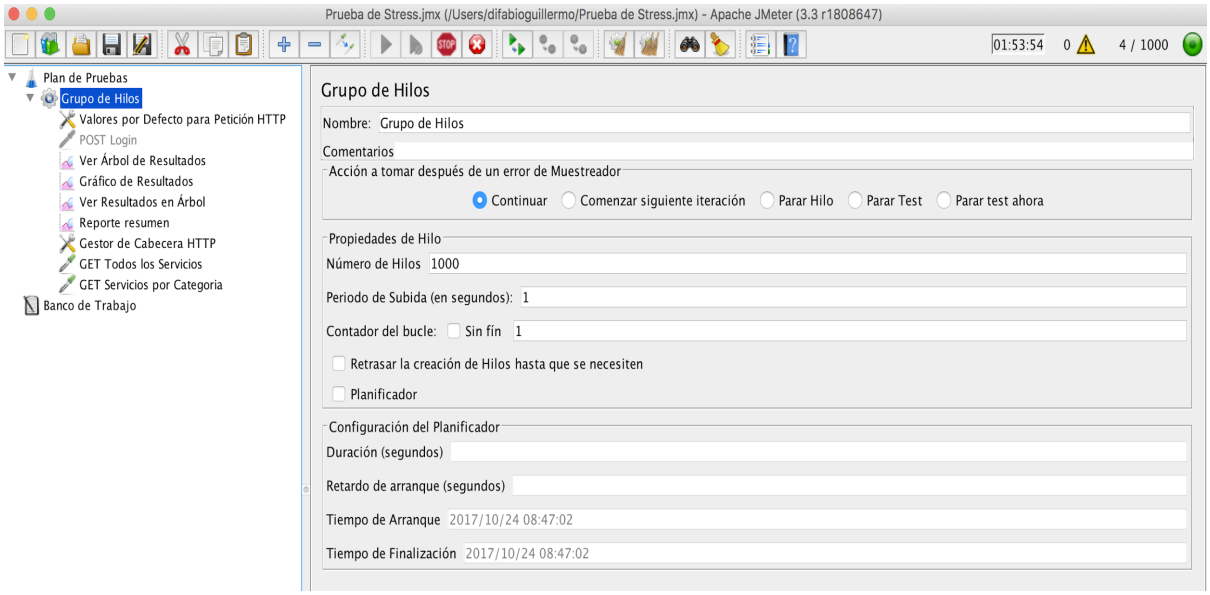
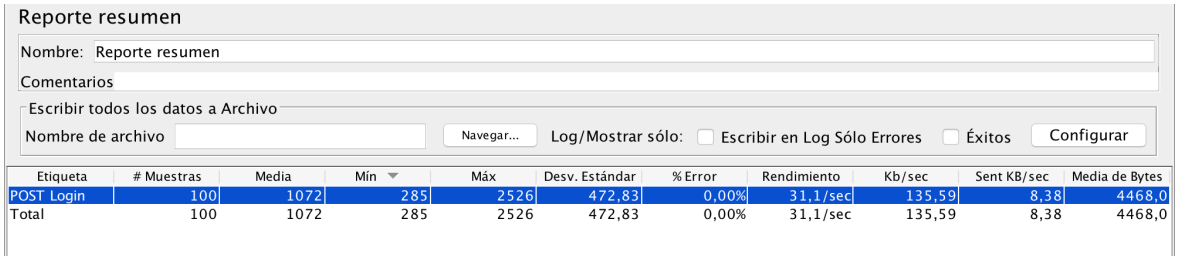


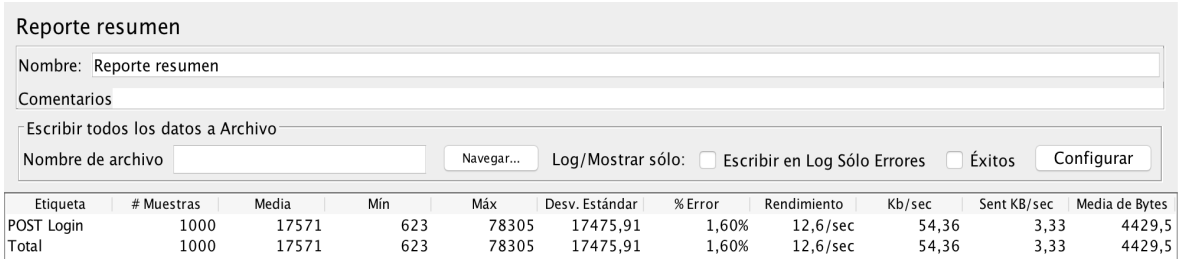
Fig.V.5. Configuración de Hilos en JMeter.

Estas pruebas se realizan para medir la respuesta de la aplicación a distintos volúmenes de carga esperados (cantidad de usuarios y/o peticiones). Se probó la velocidad de respuesta al procesar el ingreso de 100 y 1000 usuarios en forma simultánea. El resultado obtenido se puede observar en las figuras V.6 y V.7 donde se muestra un reporte en el que se detalla, entre otras cosas, el porcentaje de error a la hora de realizar las peticiones. En el caso de la figura V.6 se probó con 100 peticiones simultáneas y no fallo ninguna. En el siguiente se probó con 1000 y los resultados se pueden ver en la figura V.7, donde se generó un 1,60% de peticiones fallidas, por lo que si se tiene cuenta la diferencia se obtiene el número de peticiones exitosas, esto da un valor de 984 (1000 - 16), es decir, que este servicio soporta un volumen de usuarios menor o igual a este número de manera concurrente.



Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
POST Login	100	1072	285	2526	472,83	0,00%	31,1/sec	135,59	8,38	4468,0
Total	100	1072	285	2526	472,83	0,00%	31,1/sec	135,59	8,38	4468,0

Fig.V.6. Reporte resumido del método Login generado con JMeter.



Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
POST Login	1000	17571	623	78305	17475,91	1,60%	12,6/sec	54,36	3,33	4429,5
Total	1000	17571	623	78305	17475,91	1,60%	12,6/sec	54,36	3,33	4429,5

Fig.V.7. Reporte resumido del método Login generado con JMeter.

Por lo que en conclusión, el método de Login responde con normalidad hasta un número de 1000 peticiones simultáneas, luego a mayor número de peticiones, mayor es el porcentaje de error. A partir de esto se puede determinar que el servidor y la arquitectura es apta para la cantidad de peticiones concurrentes probadas.

También se realizó una prueba de estrés, para encontrar el volumen de datos o de tiempo en que la aplicación comienza a fallar o es incapaz de responder a las peticiones. Son pruebas de carga o rendimiento, pero superando los límites esperados en el ambiente de producción.

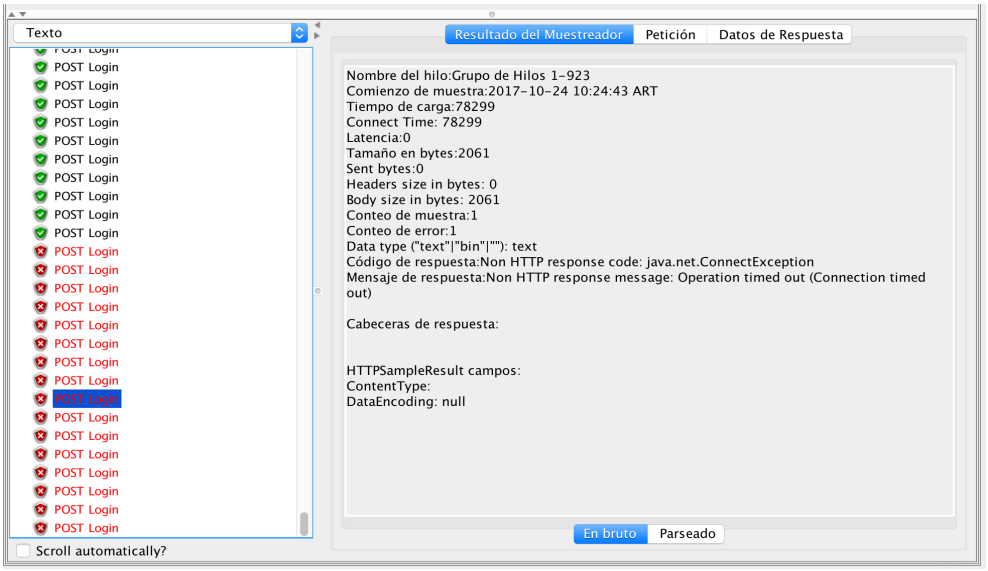


Fig.V.8. Ejemplo de Timed out.

Se trató de encontrar la cantidad de usuarios simultáneos, en que la aplicación deja de responder (timed out) en forma correcta a todas las peticiones. Para ello se probó la petición que devuelve todos los servicios con un número de 10000 peticiones simultáneas. Lo cual resolvió todas con normalidad hasta llegar a la muestra numero 846 donde dejó de responder y devolvió Operation timed out (Connection timed out) como se muestra en la figura V.8.

V.4 Pruebas de interfaz de usuario Android

Para esta prueba se utilizó Espresso, la Api de Google que permite simular interacciones de usuario. De esta forma, se pudo automatizar interacciones del usuario, para poder entonces testear sus resultados. Probar las interacciones de los usuarios en una sola aplicación garantiza que estos no encuentren resultados inesperados o que tengan una experiencia deficiente al interactuar con la aplicación. Las pruebas de Espresso se pueden ejecutar en dispositivos con Android 2.3.3 (API nivel 10) y superior. Una ventaja clave del uso de Espresso es que proporciona una sincronización automática de las acciones de prueba con la interfaz de usuario de la aplicación que se está probando. Este detecta cuando el hilo principal está inactivo, por lo que es capaz de ejecutar los comandos de prueba en el momento adecuado, lo que mejora la fiabilidad de las pruebas. Esta capacidad evita tener que agregar soluciones temporales, como Thread.sleep() en el código de prueba.

Se utilizó la ActivityTestRule, que permite iniciar la actividad que se desea probar en el marco de prueba, antes de cada método anotado con @Test y @Before. El marco gestiona el cierre de la actividad una vez que finaliza la prueba y se ejecutan todos los métodos anotados con @After. A continuación, un ejemplo del uso de ActivityTestRule:

```
@RunWith(AndroidJUnit4.class)
public class TestLogin {
   @Rule
   public ActivityTestRule<SignInActivity> mActivityRule = new ActivityTestRule<>(
           SignInActivity.class);
   @Rule
   public ActivityTestRule<MainActivity> mainActivityRule = new ActivityTestRule<>(
           MainActivity.class);
}
```

Para encontrar la vista, se llamó al método onView() y se pasó como parámetro un verificador de vistas que especifica la vista a la que se dirige con el método withId(). El siguiente fragmento de código muestra cómo se escribieron las pruebas sobre la pantalla de Login. Se escribió una clase TestLogin con dos métodos, el primero testSocialError()para probar el caso en el que el email y contraseña fueron incorrectos mostrando un mensaje de “Usuario o Contraseña Incorrecta" y el otro testSocialSucces()para probar el caso donde fueron correctos, pasando a la pantalla principal con los datos del usuario logueado. Estos métodos acceden a los campos EditText correspondientes a email y contraseña, ingresa la cadena de texto en ambos, cierran el teclado virtual y luego hace clic en un botón INGRESAR. Por último, se verifica si se cumple lo esperado en cada método.

```
   @Test
   public void testSocialError() {
       String message="Usuario o Contraseña Incorrecta";
     onView(withId(R.id.input_email))
             .perform(typeText("admin@admin.com"), closeSoftKeyboard());
     onView(withId(R.id.input_password))
               .perform(typeText("passwordfalse"), closeSoftKeyboard());
     onView(withId(R.id.btn_signin)).perform(click()); // Click botón de INGRESAR
       SignInActivity activity = mActivityRule.getActivity();
onView(withText(message)).inRoot(withDecorView(not(is(activity.getWindow().getDecor
View())))).
               check(matches(isDisplayed()));
   }
```

```
    @Test
    public void testSocialSucces() {
        String email="admin@admin.com";
        onView(withId(R.id.input_email))
                .perform(typeText(email), closeSoftKeyboard());
        onView(withId(R.id.input_password))
                .perform(typeText("passwordtrue"), closeSoftKeyboard());
        onView(withId(R.id.btn_signin)).perform(click()); // Click sobre INGRESAR
onView(withId(R.id.email_nav_header)).check(matches(withText(email)));
    }
}
```

En la figura V.9 se puede ver cómo se ejecutan los métodos anteriores desde Android Studio sobre el emulador de Android.

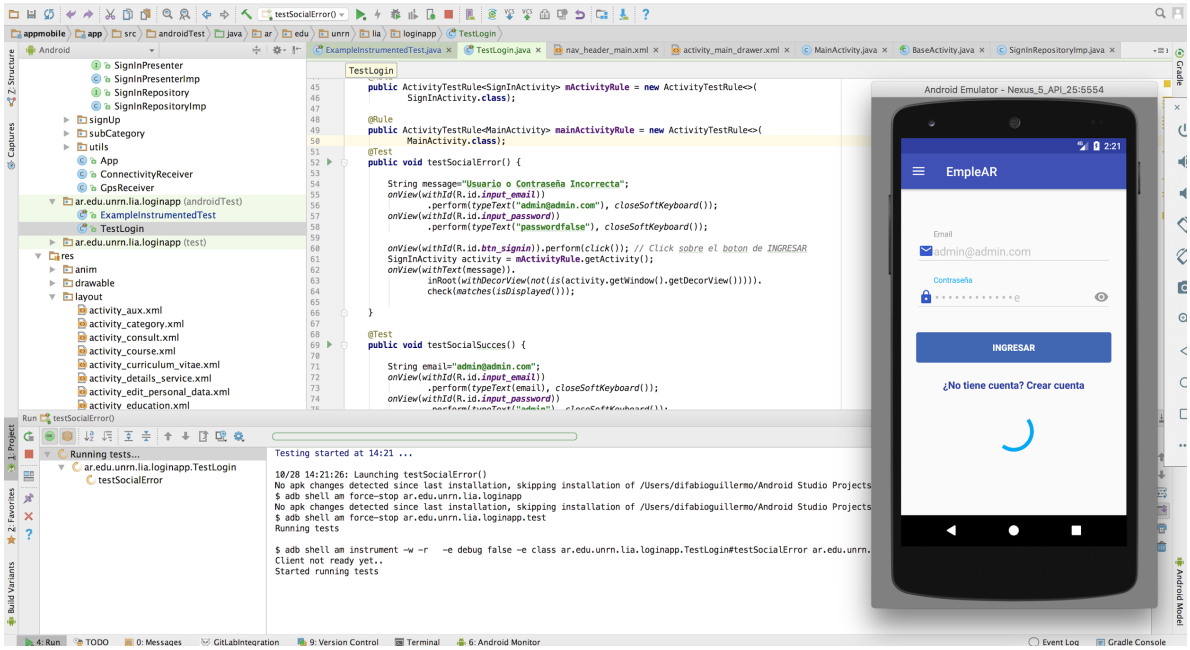


Fig.V.9. Test Android Studio.

Por último, en la figura V.10 se puede observar el resultado de los Test, donde se probó con un email y contraseña incorrectos, el método testSocialSucces paso y el método testSocialError no. Por lo que se puede establecer que los test están cumpliendo su función como corresponde.

Fig.V.10. Resultado del test.

VI. Conclusiones y líneas futuras

VI.1 Conclusiones

Con el desarrollo de esta aplicación se pretende implementar los conocimientos adquiridos durante el transcurso de la carrera con el fin de poder brindarle solución a una problemática socioeconómica, que no sólo estará reducida a los ciudadanos viedmenses sino que permitirá expandirla a nivel nacional. La potencialidad de la misma se basa en el impacto en la comunidad, sin costo alguno, a través de la generación de empleo y la inclusión de personas que hoy en día están fuera del sistema por no calificar para los requisitos de reclutamiento formal. El desarrollo de las TICs y el incremento en el uso de las redes posibilita la construcción de relaciones más horizontales y un mayor acceso a la información de calidad.

Haciendo un análisis de la aplicación se puede observar que la misma plantea la inserción al mercado laboral desde otra lógica, es decir, a través de la co-construcción conjunta de sus propios CVs, ya que quienes contraten un servicio, podrán calificar al contratado una vez que este haya culminado su prestación, para que otros interesados lo vean, contribuyendo a la mejora en la calidad del servicio y a generar confianza. De esta manera, el actual proyecto generará un impacto positivo en la sociedad dado que, como toda red social, estimula el desarrollo de valores de colaboración y cooperación.

También se identificaron quienes serán los posibles beneficiarios de la misma, por un lado, estarán los directos, personas vinculadas a los distintos planes sociales convertidos en usuarios de la aplicación móvil que prestarán servicios susceptibles de ser contratados a través de la misma, como también potenciales usuarios demandantes de servicios particulares que la hayan descargado.

Por otro lado los beneficiarios indirectos, serán empresas que podrán visibilizar usuarios que ofrecen sus servicios particulares y cuentan con antecedentes generados a partir de la valoración que realizaron los usuarios demandantes, también lo serán los organismos del estado nacional, provincial o municipal ya que

los datos que se generan como, las categorías de contrataciones, cantidad de usuarios demandantes, oferentes y geo-localización de los servicios prestados, son de utilidad para el análisis y posterior formulación de políticas públicas en relación al mercado laboral.

En cuanto al diseño de la arquitectura y la metodología de trabajo, ambos permitieron la implementación de los temas inherentes definidos en cada uno de los Sprints, como registro de usuario, gestión de perfil, de servicios, de contrataciones y administración de consultas y operaciones. Así mismo, permitieron el desarrollo en paralelo del backend y el frontend, acelerando el proceso, fortaleciendo el trabajo en equipo, aumentando la escalabilidad y mejorando la portabilidad de la interfaz permitiendo migrarla a otro tipo de plataformas.

VI.2 Líneas futuras

En próximos proyectos sería interesante utilizar la arquitectura propuesta para poder implementar la aplicación en otras plataformas, ya que, el API REST de backend como se explicó en capítulos anteriores nos permitirá desarrollar cualquier frontend que entienda solicitudes HTTP. Aprovechando estas caracteristicas es que se plantea como próximo objetivo el desarrollo de una aplicación en iOS que pueda interactuar con el backend de la misma manera que lo hace la ya desarrollada en Android.

VI.2.1 Implementar solución en iOS

Con el objetivo de lograr una aplicación multiplataforma sin dejar a ningún usuario fuera del alcance de la aplicación, será necesario un nuevo desarrollo para que pueda ser utilizada en dispositivos con sistema operativo iOS.

Para este tipo de desarrollo es indispensable contar con un Mac, ya que las herramientas para llevar a cabo la implementación solo están disponibles para Mac OSX o Hackintosh. También se necesitará Xcode, el entorno de desarrollo de Apple para todos sus dispositivos y SO, que se encargará de proporcionar el SDK de iOS, el cual dispone de todas las herramientas, compiladores y frameworks necesarios.

Es gratuito y se puede obtener desde la Mac App Store o desde la página de desarrolladores de Apple.

VI.2.2 Validar datos con organismos estatales competentes

Los CV cargados en la aplicación permitirán a los usuarios lograr por una parte mayor visibilidad de sus conocimientos y experiencia en determinados oficios frente a los demandantes de los mismos y por otro lado incorporarlos a la sociedad del conocimiento a través de la construcción de sus identidades digitales. Sin embargo, la identidad digital de una persona puede coincidir o no con la de su vida real. Para poder lograr una homogeneidad entre ambas se propone desde dicha aplicación consumir servicios prestados por distintos Organismos estatales que permitan verificar los datos declarados en el CV digital, por ejemplo.

VI.2.3 Visualización de datos para la toma de decisiones en políticas públicas

Se propone desarrollar un módulo que permitirá extraer conclusiones, tendencias y patrones a partir del crecimiento de la cantidad de información que en un futuro generará la aplicación. Esta información será útil para respaldar decisiones relacionadas al desarrollo de las distintas políticas públicas y es aquí donde la visualización de datos hace su aporte significativo.

VI.2.4 Versión premium

El modelo de la app se ubica en el segmento de desarrollo de aplicaciones móviles que son de descarga gratuita, aunque existe la posibilidad de monetizar a partir de incorporar “compras incrustadas” conocidas como “pagos inApp”. Si bien la descarga está pensada como un servicio gratuito existen complementos que harán mejorar la reputación online consiguiendo la verificación total de la cuenta personal.

Teniendo en cuenta la importancia de la gratuidad de la aplicación para los usuarios, a cambio de esto se incluyen pequeños banners de publicidad dentro de la misma. Esta estrategia funciona con el denominado “coste por click”, cada vez que el usuario haga click sobre la publicidad, genera automáticamente un ingreso de dinero para la organización que administra la aplicación.

En una segunda etapa de la aplicación, se consideró que se mantenga la descarga gratuita, pero con posibilidades de que los usuarios compren una versión Premium para extender las funcionalidades de la misma. La estrategia consistirá en promocionar la versión Premium con descarga gratuita durante un tiempo limitado (15 días o un mes) para que puedan probar y luego esté dispuesto a pagar por ella. Esta estrategia estará dirigida a determinados usuarios a los que se llegará a través de campañas puntuales de promoción.

VI.2.5 Arquitectura de la solución

En el transcurso del desarrollo de la tesina y particularmente en la evaluación de la Arquitectura de la aplicación, fueron surgiendo modificaciones producto del avance de las tecnologías, como en el caso de la plataforma Android, servicios web, nuevos framework, patrones de diseño y herramientas que podrían mejorar el producto de este trabajo.

Bibliografía

- Apple, https://www.apple.com/es/swift. [Accedido Octubre. 2017].
- Caules, C. L. (2012). Arquitectura java sólida.
- Developers Facebook, https://developers.facebook.com/docs. [Accedido Octubre. 2017].
- Developers Google, https://developers.google.com. [Accedido Octubre. 2017].
- Firebase, https://firebase.google.com. [Accedido Octubre. 2017].
- Gargenta, M. (2011). Learning Android. Sebastopol, CA: O'Reilly.
- Gartner, http://www.gartner.com/technology/home.jsp. [Accedido Octubre. 2017].
- GitLab, https://about.gitlab.com. [Accedido Agosto. 2017].
- Google, Android Studio, https://developer.android.com/studio/index.html. [Accedido Mayo. 2017].
- JMeter, http://jmeter.apache.org. [Accedido Octubre. 2017].
- Kniberg, H. (2007). Scrum and xp from the trenches: How we do scrum.
- Manifiesto Ágil, http://agilemanifesto.org. [Accedido Junio. 2017].
- Ministerio de Desarrollo Humano, http://viedma.gov.ar/noticias/desarrollo-humano. [Accedido Junio. 2017].
- Retrofit, http://square.github.io/retrofit/.[Accedido Octubre. 2017].
- Saripaar, https://github.com/ragunathjawahar/android-saripaar. [Accedido Octubre. 2017].
- Square Open Source, http://square.github.io. [Accedido Agosto. 2017].