UNIVERSIDAD NACIONAL DE LA PATAGONIA SAN JUAN BOSCO

FACULTAD DE INGENIERÍA

SEDE TRELEW

Ejercicio Práctico de Aprobación

“Aplicación móvil SIU GUARANÍ”

Cátedra: Taller de Nuevas Tecnologías

Docentes:

- Profesor Adjunto: Lic. Diego FIRMENICH
- JTP: Lic. Sebastián SCHANZ

Integrantes:

- CALFUQUIR, Jorge Nicolás
- PARRA, Iván Javier
- SANTOS, Carla Cecilia

Ciclo Lectivo 2017

Índice

Enunciado

Características de la aplicación móvil requerida:

- Permita establecer preferencias de app. (usuario y contraseña guaraní).
- Muestre notificaciones de mesas disponibles.
- Diariamente, sin la intervención del usuario, se verifique si hay o no mesa disponible.
- Listado de materias disponibles cuando hay mesa (para el usuario de la conf).
- Que permita inscribirse (pueden simular un servicio local que reciba el post).
- Si no hay mesa, mostrar que no hay mesa disponible aún.

Aplicación móvil SIU GUARANÍ

Introducción

Hemos desarrollado una aplicación móvil nativa en Android que realiza web scraping sobre el Sistema de Gestión de Alumnos de la Facultad de Ingeniería Sede Trelew de la UNPSJB llamado SIU GUARANÍ.
Esta aplicación permite al alumno:

- Iniciar sesión en el Sistema SIU GUARANÍ.
- Ver mesas de examen e inscripciones.
- Inscribirse y desincribirse en mesas de exámen.

Periódicamente, sin la intervención del usuario, se verifica si hay o no mesas disponibles como así también, si está inscripto o no a la misma. Se le informa al mismo a través de determinados mecanismos que ofrece Android.
Esta aplicación está destinada a los alumnos de las carreras Analista Programador Universitario (APU) y Licenciatura en Sistemas (Or. Planificación, Gestión y Control de Proyectos Informáticos) de la Facultad de Ingeniería Sede Trelew de la Universidad Nacional de la Patagonia San Juan Bosco (UNPSJB).
El enlace a SIU GUARANÍ es el siguiente: http://www.dit.ing.unp.edu.ar/v2070/www/
El código fuente se puede ver en https://github.com/nicocarp/scraping_guarani

Conceptos de Android

Introducción

El presente informe pretende explicar la teoría detrás de la aplicación que hemos desarrollado, específicamente el funcionamiento de la plataforma Android.

Android

Android es una plataforma basada en el sistema operativo Linux, diseñada principalmente para dispositivos móviles con pantalla táctil, como smartphones, tablets y también para smartwatchs, televisores y automóviles.
La elección de esta plataforma para el desarrollo de nuestra aplicación es muy simple, ya que es la que posee mayor cantidad de usuarios respecto a otras plataformas, es de código abierto, gratuita y está respaldada por una Alianza de 84 empresas mundialmente reconocidas llamada Open Handset Alliance (OHA), dentro de las cuales se encuentran:

- Operadores de redes: Telecom, Telefónica, Bodafone, etc.
- Fabricantes de móviles: Acer, Alcatel, Asus, Dell, Garmin, Huawei, Lenovo, LG, Motorola, Samsung, ZTE, etc
- Fabricantes de procesadores: ARM, Atheros, Intel, nVidia, Texas Instruments, etc.
- Fabricantes de software: Google, Ebay, etc.

Juntos desarrollan y sostienen Android. Se comprometen a distribuir y desarrollar comercialmente dispositivos y servicios móviles utilizando Android Platform.

Android combina:

- Software Libre
  - Plataforma de desarrollo libre y de código abierto. No se pagan royalties. Utiliza Apache V2.
  - Basada en software libre: Linux, WebKit, SQLite.
- Portabilidad
  - Aplicaciones de usuario escritas en Java.
- Estándares bien conocidos
  - Componentes reutilizables basados en estándares bien conocidos. Interfaz Gráfica (UI) basado en XML.

Android ofrece:

- Muchos servicios para las aplicaciones: Localización, bases de datos, reconocimiento de voz, sensores, etc.

- Alto nivel de seguridad: La seguridad es administrada por la plataforma explícitamente y también es posible hacerlo bajo demanda.
- Optimización: Kernel y máquina virtual optimizados para dispositivos móviles.
- Alta calidad para la reproducción de contenido multimedia: Basado OpenGl y distribuyendo codecs de audio y video más populares.
- Excelentes herramientas de desarrollo.
- Grandes comunidades de desarrollo.
- Mucha documentación y material.

En resumen, Android es una opción segura por su presente, por ser abierto y por sobre todo: por su futuro sostenido por el compromiso de esas grandes empresas en mantenerlo de esa manera. Dada la popularidad que ha alcanzado hoy en día la plataforma Android, ha hecho que esta cuente con la comunidad mundial más grande de desarrolladores, con múltiples eventos, concursos, competiciones y reuniones así como diversas vías de comunicación como foros y chats oficiales para fomentar la participación y la colaboración para encontrar mejoras e ideas para futuras versiones. Esta característica no es poca cosa para un desarrollador, ya que proporciona un recurso de apoyo y consulta invaluable a la hora de desarrollar.

Arquitectura de la Plataforma Android

La siguiente figura muestra los componentes principales del sistema operativo Android:

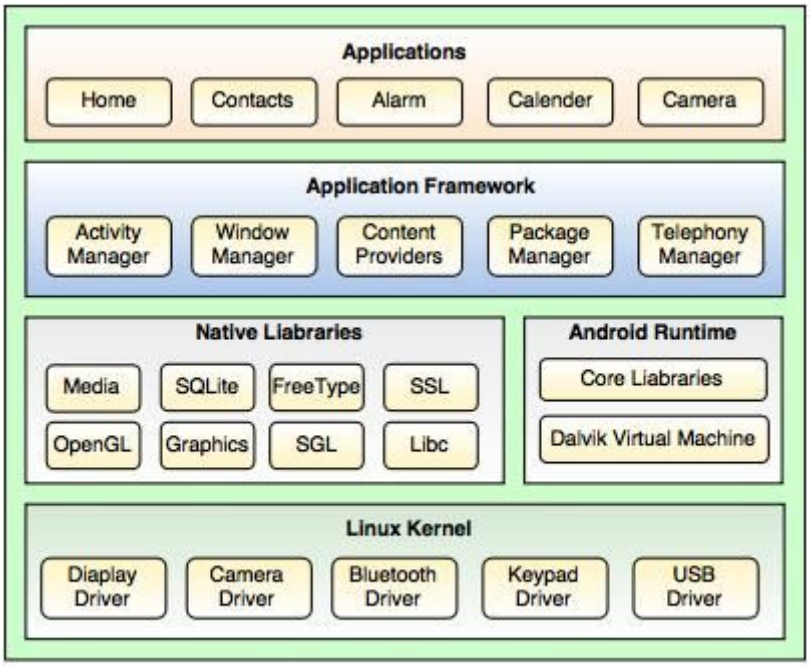


Fig. Android Architecture

Applications (Aplicaciones): En esta capa conviven las aplicaciones de usuario instaladas en el dispositivo.

- Aplicación de Inicio (pantalla principal).
- Contactos.
- Teléfono.
- Navegador.
- Galería.
- Correo electrónico..

Cualquiera de estas aplicaciones es sustituible y configurable por el usuario.

Applications Framework (Marco de trabajo de aplicaciones): los desarrolladores tienen acceso completo a los mismas API del entorno de trabajo usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.
Esta capa contiene y provee software común a las aplicaciones de usuario. Ofrece muchas funcionalidades que son utilizadas por las aplicaciones de usuario: como acceso a los sensores, servicios de localización, barra de notificación, etc. Las aplicaciones pueden compartir sus capacidades con las otras aplicaciones para fortalecer el reúso. A través del framework es posible que los usuarios intercambien aplicaciones para realizar tareas.

- Views System: Conjunto de vistas y componentes UI de las aplicaciones.
- Content providers: Mecanismo sencillo para acceder a datos de otras aplicaciones. (contactos, listas de llamadas, etc)
- Resource manager: Proporciona acceso a los recursos que no son codificados: imágenes, íconos, videos, etc.
- Notificación manager: Administra el área de notificaciones del dispositivo.
- Activity manager: Administra el ciclo de vida de todas las aplicaciones proporcionando un mecanismo de navegación entre ellas.

Runtime de Android:

- Librerías Java: Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java.
    - Clases básicas java.*, javax.*
    - Específicas de la plataforma: android.*
    - Internet/web services: org.*
- Máquinas Virtuales. Dalvik / ART: Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Están diseñadas para ambientes de recursos limitados (CPUs más lentas, menos memoria, batería limitada). La primera fue Dalvik y luego surgió ART (Android 4.4.x) que introdujo mejoras en el rendimiento general.

Libraries (Librerías): Android incluye un conjunto de librerías de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android. Esta capa de software controla gran parte del comportamiento que afecta drásticamente el rendimiento del dispositivo. Por ejemplo, la Renderización de gráficos y páginas web's, elementos de la pantalla.

- Bionic LibC: Es la librería de C re escrita para Android.
- Manejador de gestos y de pantalla.
- Codec's multimedia para audio y video.
- SQLite: bases de datos relacionales.

Linux Kernell (Núcleo Linux): Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software. El Android Linux Kernell es conocido como AOSP (Android Open Source Project).
El Kernel ofrece los siguientes servicios comunes:

- Seguridad.
- Administrador de memoria y de procesos.
- Sistema de archivos (ext4).
- Pila de red (network stack I/O).
- Controladores de hardware.

El Kernel ofrece los siguientes servicios específicos:

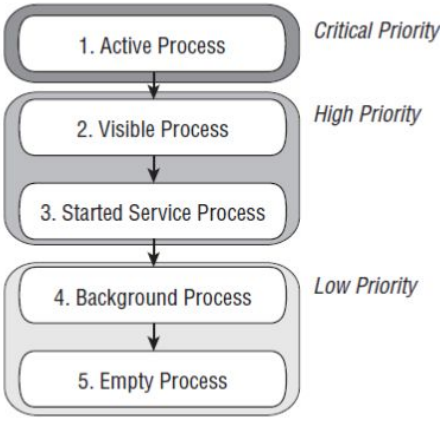
- Administración de energía.
- Memoria compartida.
- Controlador de memoria escasa (Low memory killer).
- Android Interprocess communication: Permite compartir datos y servicios de forma sofisticada apropiada a la plataforma. Se lo llama Binder.
- HAL: capa de abstracción de hardware relacionada con el software controlador sensores específicos.

Ciclo de vida de los procesos en la plataforma
Las aplicaciones Android son construidas en base a cuatro tipos de componentes básicos:

- Activities (Actividades): se ejecutan en primer plano, muestran pantallas de usuario.
- Services (Servicios): se ejecutan en segundo plano, realizan una tarea específica.
- Broadcast e Intent Receivers: son componentes para la comunicación entre aplicaciones y servicios.
- Content providers (Proveedores de Contenido): son componentes que permiten compartir datos entre aplicaciones.

La plataforma Android crea un proceso para cada una de las aplicaciones cuando estas son ejecutadas. Además, se encarga de administrar el ciclo de vida de las aplicaciones y de sus componentes manipulando los procesos en los que estas se ejecutan.

El ciclo de vida se muestra en el siguiente gráfico:



1. El "Active/foreground Process" es el proceso que contiene la actividad que el usuario está utilizando.
2. Los "Visible Process" son aquellos que contienen porciones visibles de la pantalla, pero que no están al frente.
3. Los "Service Process" son aquellos que ejecutan un servicio en segundo plano.
4. Los "Background Process" son aquellos que contiene una aplicación que no es visible al usuario.
5. Los "Empty Process" son aquellos que no contiene una aplicación, pero el proceso se mantiene para evitar la creación de uno nuevo cuando haga falta.

Los procesos continúan en memoria tanto como sea posible, aún cuando los usuarios hayan cerrado las aplicaciones (empty process). Cuando hace falta memoria, la plataforma decide que proceso finalizar.

Componentes de una aplicación

Como hemos mencionado anteriormente, pueden existir cuatro tipo de componentes básicos en una aplicación: activities, services, broadcast e intents receivers y content providers. A continuación haremos una breve explicación de cada uno, ya que la aplicación que hemos desarrollado los utiliza.

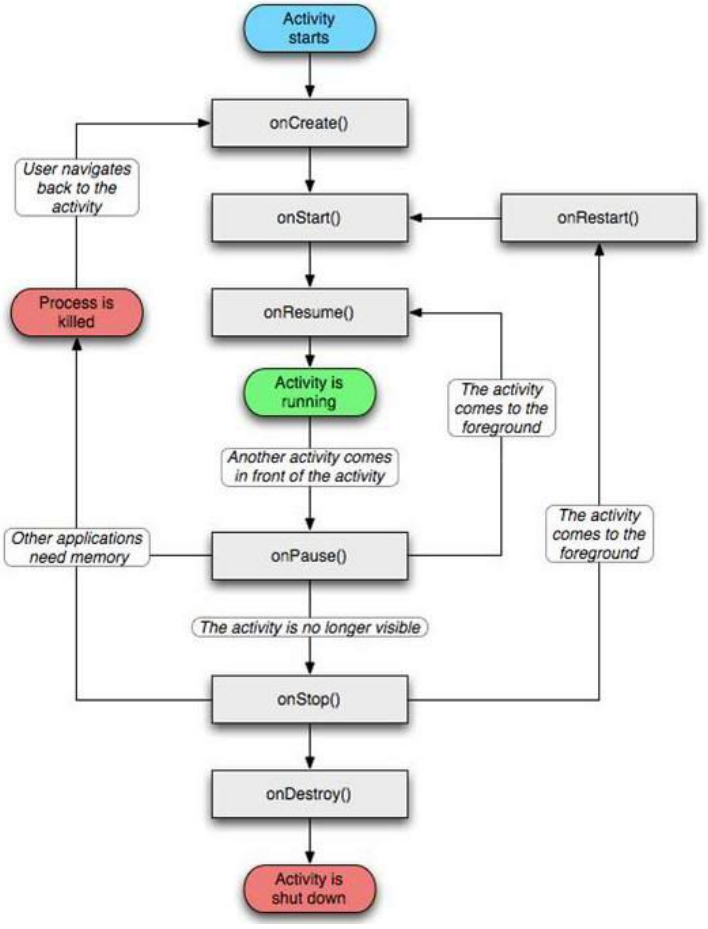
Todo componente de una aplicación debe estar declarado en el Manifest de la aplicación.

Activities

Las actividades, son piezas ejecutables que pueden ser instanciadas por el usuario o por la plataforma. Pueden interactuar con el usuario, o con otras actividades y servicios. Las

actividades muestran pantallas de usuario, y para cada actividad se muestra una pantalla al usuario.

El ciclo de vida de las actividades está diseñado para ahorrar recursos. Las actividades (que se corresponden con una pantalla de usuario) no están activas constantemente. El Framework manipula las actividades enviándoles mensajes para activarlas, iniciarlas, mandarlas al segundo plano, pausarlas, resumirlas, desactivarlas.



onCreate(): Es invocado cuando la Actividad es creada. Debe inicializar el estado de la actividad:

- Invocar a super.onCreate() para que Android realice su propia inicialización.
- Indicará cuáles son los componentes visuales de la actividad. Es decir, cuál es la interfaz de usuario que el usuario debe utilizar.
- Configurará las vistas como sea requerido.
- Tendrá oportunidad de reestablecer el estado anterior de la actividad, si es que lo hubiere disponible.

onStart(): Es invocado cuando la Actividad se ha vuelto visible.

- Las acciones a realizar son relativas a cada aplicación.
- Entre las acciones típicas puede considerarse las instanciaciones de managers de localización.

onResume(): Es invocado cuando la Actividad se ha vuelto visible y es posible interactuar con el usuario.

- Las acciones a realizar son relativas a cada aplicación.
- En cualquier caso dichas acciones deben estar relacionadas con comportamiento en primer plano, luego de haber sido pausada la aplicación.

onPause(): Es invocado cuando la Actividad ha perdido el foco.

- La actividad está a punto de ser pasada a un segundo plano.
- Es un buen lugar para para liberar recursos solicitados, posicionamiento, etc.
- Es el primer síntoma de que nuestra aplicación puede dejar de ser usada por un rato (potencialmente).
- Debe ejecutarse rápido.
- Si hay que persistir cosas, usamos onStop.

onStop(): Es invocado cuando la Actividad ya no es visible para el usuario.

- La actividad ha sido pasada a un segundo plano.
- No tenemos garantías de que este método sea invocado.
- Puede ser utilizado para tareas como persistir objetos en términos de caché (si se pierden no es tan dramático).

onRestart(): Es invocado cuando la Actividad ha sido parada y esta por ser iniciada nuevamente.

- Las acciones a realizar son relativas a cada aplicación.
- Hay que usar con cuidado este método porque si la actividad está siendo re-creada luego de haber sido finalizado su proceso por parte del framework, este método no será invocado.
- La utilización de este método es poco común, sin embargo el framework da la posibilidad de utilizarlo.

onDestroy(): Es invocado cuando la Actividad ha sido finalizada.

- No tenemos garantías de que este método sea invocado.
- Aquí podemos incorporar comportamiento relativo a la administración y liberación de recursos que hubieran sido liberados de todos modos ante una eventual desplanificaciónde nuestra actividad.

Recreación: Cuando la actividad es recreada, la plataforma se encarga de reconstruir el estado de sus componentes visuales. No es nuestra responsabilidad, por ejemplo, recomponer el estado de un cuadro de texto que el usuario hubiere completado de modo incompleto previo a que la actividad fuere pausada o parada. Si será nuestra responsabilidad guardar cualquier estado correspondiente a nuestro dominio que no sea relativo a los componentes de la interfaz.
Para ello podemos utilizar los métodos:

- onSaveInstanceState()
- onRestoreInstanceState()

No debemos olvidar nunca invocar a super en estos métodos, ya que es allí donde la plataforma salva y restaura el estado de los componentes de interfaz.
Allí debemos tener comportamiento liviano, de modo que no obtener una lenta renderización de la interfaz del usuario.
Nosotros hemos utilizado estos métodos para guardar el estado del progressBar cuando se está iniciando sesión. Si yo paso de portrait a landscape y mi progressBar se está ejecutando entonces se eliminará, porque cada vez que roto la pantalla las actividades son destruidas y recreadas. Por eso, en el método onSaveInstanceState() guardabamos el estado del progressBar (Visible/NoVisible), y en base a eso lo reconstruimos.

Services
Un servicio es un componente de aplicación que puede ejecutar operaciones a largo plazo en segundo plano. Estas operaciones son llevadas a cabo sin una interfaz de usuario. Los otros componentes de la Aplicación pueden lanzar un servicio y éste se continuará ejecutando en segundo plano incluso cuando el usuario pasa a utilizar otra Aplicación.

Los servicios pueden funcionar de dos modos:

- **Iniciados (Started):** Los inician otros componentes de la aplicación (ej. Una actividad) para realizar una tarea en segundo plano. Para ello, deben ejecutar el método startService(). Pueden correr en segundo plano tanto tiempo como sea requerido, incluso si el usuario deja de usar el componente que originó su ejecución. Generalmente realizan una única acción y finalizan sin devolver resultados a quién lo inició.
- **Ligados (Bounded):** Son servicios que se ejecutan en segundo plano con la intención de recibir solicitudes cliente/servidor, desde uno o varios componentes de la aplicación. Las aplicaciones se ligan a estos servicios a través del método bindService(). El servicio ligado podrá ofrecer una interfaz cliente servidor, permitiendo a los otros componentes interactuar con él enviando mensajes y recibiendo respuestas. El servicio finaliza cuando no hay componentes ligados a él.

Un servicio puede funcionar de ambas formas (iniciado y ligado). Para ello debe implementar dos callbacks: onStartCommand() y onBind().

Cabe aclarar que los servicios, por omisión, corren sobre el hilo de ejecución principal del proceso de aplicación que les dio lugar. Por ello debemos tener presente que si haremos operaciones que pueden bloquear el hilo de ejecución, necesitaremos un nuevo hilo para nuestro servicio, o bien para la tarea que requiere dicho procesamiento. Aquí es donde viene la separación entre Service e Intent Service. El primero, comparte el proceso de ejecución con la Aplicación. (Pueden ser iniciados o ligados). El segundo, es un tipo especial de Servicio que no comparte el proceso de ejecución con la Aplicación, son utilizados para realizar tareas en segundo plano que puedan bloquear el hilo principal.

Usamos un Service cuando requerimos un servicios en el segundo plano por un tiempo indeterminado e independientemente de los componentes que puedan interactuar con él.
No debe bloquear el UI THREAD. Se pueden lanzar tareas asincrónicas desde el servicio para realizar tareas de mayor procesamiento en otro hilo.

Un Intent Service está diseñado para ejecutar una determinada tarea en un hilo independiente al hilo principal de la aplicación. No es ni más ni menos que un tipo particular de servicio Android que se preocupará por nosotros de la creación y gestión del nuevo hilo de ejecución y de detenerse a sí mismo una vez concluida su tarea asociada. Es esencialmente útil si necesitamos realizar varias veces las mismas tareas y queremos apilar las solicitudes en un worker distinto al de la UI principal. Usualmente sin comunicación con el hilo principal.

Broadcast Receivers

Los BroadcastReceivers son componentes que nos permiten escuchar por eventos que se difunden en la plataforma. Responden a las solicitudes de otras aplicaciones. Los Broadcast Receivers, responden mensajes/anuncios enviados a todo el sistema. Los mensajes pueden provenir de la plataforma (Por ejemplo: batería baja) o de una aplicación. Nuestra aplicación escucha mensajes desde ambos lugares.
Para crear un Broadcast Receiver nuestra clase de extender de BroadcastReceiver y debe implementar el método onReceive(). Además, en el archivo AndroidManifest.xml indicamos qué evento debe atender.

Content Provider

Es un componente de aplicación que permiten compartir datos entre las aplicaciones. Un Content Provider utiliza una interfaz en forma de URI para permitirle a las otras aplicaciones acceder a sus datos. Las otras aplicaciones no necesitan saber que aplicación responde su solicitud, simplemente utilizan la uri. Por ejemplo: content://contacts/people

Android Studio

Se trata del entorno de desarrollo integrado (IDE) oficial para la plataforma Android.
Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas Microsoft Windows, macOS y GNU/Linux. Ha sido diseñado específicamente para el desarrollo de Android. Utiliza el lenguaje de programación Java.

Su última versión proporciona las siguientes características:

- Instant Run: Cuando haces clic en Run o Debug, la función Instant Run de Android Studio aplicará los cambios en el código y los recursos en la aplicación en ejecución. Esta interpreta de manera inteligente los cambios y a menudo los entrega sin reiniciar la app ni volver a compilar el APK, para poder ver los efectos de inmediato.
- Editor de código inteligente: Al ofrecer compleción avanzada de código, refactorización y análisis de código, el editor de código inteligente permite escribir un código más eficaz, trabajar más rápido y ser más productivo. A medida que uno escribe, Android Studio proporciona sugerencias en una lista desplegable.
- Integración de ProGuard y funciones de firma de aplicaciones.
- Renderizado en tiempo real.
- Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.
- Soporte para construcción basada en Gradle.
- Refactorización específica de Android y arreglos rápidos.
- Editor de diseño: editor visual con la función arrastrar y colocar componentes gráficos.
- Analizador de APK: para inspeccionar fácilmente el contenido de un APK.
- Estudio de recursos vectoriales: Android Studio facilita la creación de un nuevo recurso de imagen para cada densidad.
- Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones, y otros problemas.
- Plantillas para crear diseños comunes de Android y otros componentes.
- Soporte para programar aplicaciones para Android Wear.
- Soporte integrado para Google Cloud Platform, que permite la integración con Google Cloud Messaging y App Engine.
- Emulador rápido y cargado de funciones: Android Emulator se instala e inicia las apps más rápido que un dispositivo real. También te permite crear prototipos de una app y probarlos en todas las configuraciones de dispositivos Android: teléfonos, tablets y dispositivos Android Wear y Android TV. También se puede simular varias funciones de hardware, como la localización de GPS, la latencia de red y las funciones multitáctiles.

Los requisitos del sistema para las tres plataformas (Microsoft Windows, macOS y GNU/Linux) en su versión 3.x son:

- RAM: 3 GB de RAM mínimo, 8 GB de RAM recomendado más 1GB adicional para el emulador de Android.
- Espacio en Disco: 2 GB de espacio en disco para Android Studio, 4GB recomendados (500MB para la IDE y al menos 1.5 GB para Android SDK, imágenes de sistema de emulador y cachés).
- Versión de Java: Java Development Kit (JDK) 8.

Conceptos de Web Scraping

Web Scraping

Scraping es un término que, traducido al español, literalmente quiere decir "rascado". Sin embargo, en este contexto, se refiere a la limpieza y filtro de los datos. Web scraping es una técnica utilizada para extraer información de sitios web, es decir, es el proceso de recopilar información de forma automática de la Web. Conocer la estructura de una página web (DOM) es el primer paso para extraer y usar los datos.

Para el desarrollo de nuestra aplicación, tuvimos que scrapear la página de SIU GUARANÍ http://www.dit.ing.unp.edu.ar/v2070/www/
El primer paso consistió en descargar todas las páginas web de dicho sitio, porque las inscripciones a mesas de examen duran pocos días, y nosotros las necesitamos para poder scrapear datos. Para esto, utilizamos Wget. Se trata de una herramienta libre que permite la descarga de contenidos desde servidores web de una forma simple. Su nombre deriva de World Wide Web (w), y de «obtener» (en inglés get), esto quiere decir: obtener desde la WWW.
Una vez que obtuvimos todo el contenido que necesitábamos, tuvimos que empezar a estudiar la estructura del mismo, para así poder scrapear datos.

Desarrollo de la Aplicación Móvil SIU GUARANÍ

Funcionamiento general de la Aplicación

Cuando el usuario abre la aplicación por primera vez, se le mostrará la pantalla de login:

Al presionar sobre "Continuar" se realiza una petición a Guaraní para iniciar sesión (a través de una instancia de la clase Guarani.java). Si fue correcto, se disparará inmediatamente el servicio (Intent Service) en un hilo secundario que se encargará de scrapear mesas e inscripciones en SIU Guaraní y si existen las comunicará al usuario/aplicación a través de notifications y broadcast messages, guardandolas además en las preferencias. También se disparará una alarma (Alarm Manager) que permitirá repetir el servicio cada 12 hs. Mientras tanto, se le mostrará la pantalla principal al usuario, la cual leerá las preferencias para ver si existen mesas/inscripciones guardadas, y si es así las mostrará en el fragment de mesas/inscripciones. El usuario tendrá la opción de filtrar mesas/inscripciones por carrera.

Las mesas que están coloreadas de verde son aquellas a las que estamos inscriptos, las que están en color gris y contienen una fecha son aquellas mesas a las cuales no estamos inscriptos pero nos podemos anotar, y finalmente las mesas que están en color gris pero sin fecha son aquellas a las que no podremos anotarnos porque nos faltan correlativas.

Cuando queremos inscribirnos a una materia, bastará con presionar una mesa de examen que esté en color gris y con fecha, y aparecerá el siguiente AlertDialog que contendrá los detalles de la mesa.

Si aceptamos, se hará una petición a Guaraní para inscribirnos en la mesa. Y si no hubo error, se actualizará el fragment de mesas/inscripciones.

Si queremos desinscribirnos de una mesa, bastará con presionar la mesa de examen que está en color verde, y se mostrará un AlertDialog que contendrá los detalles de la inscripción.

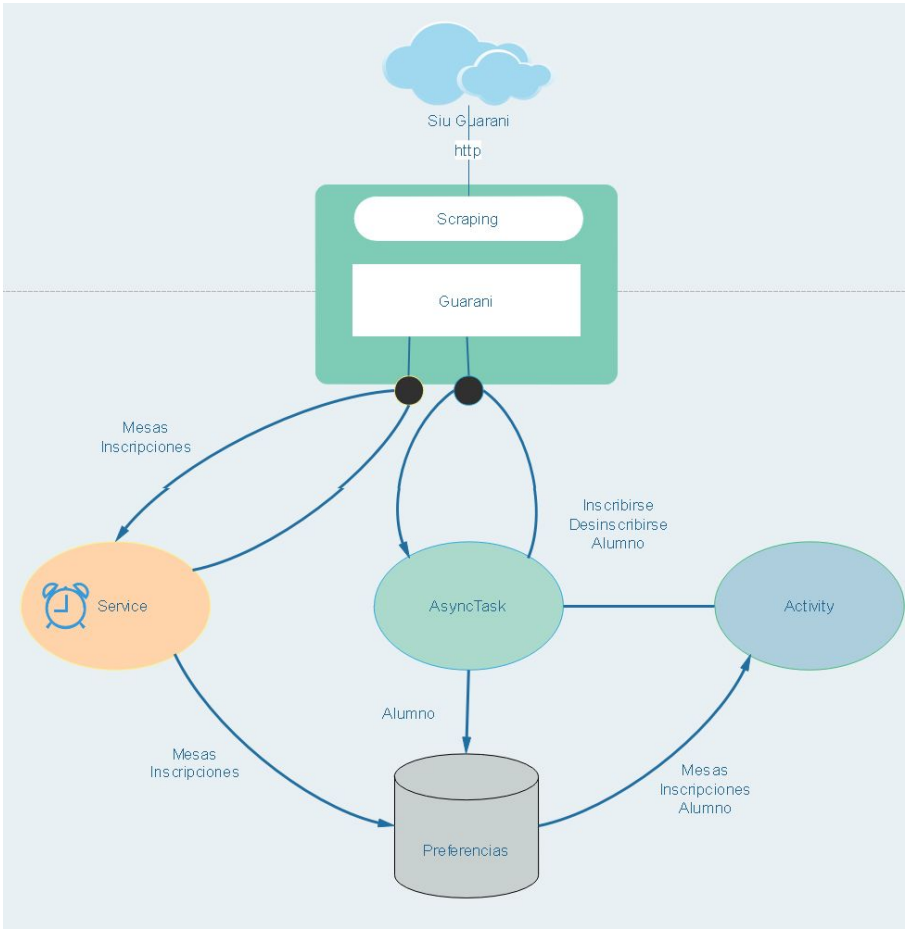
Si aceptamos, se realizará una petición a Guaraní para desinscribirnos de dicha mesa. Y si no hubo error, se actualizará el fragment de mesas/inscripciones.

Si hacemos click sobre una mesa que está en color gris y sin fecha, nos aparecerá otro AlertDialog que nos indicará que no podemos inscribirnos y nos mostrará el detalle de correlativas que necesitamos.

Finalmente, podremos cerrar sesión. Esto disparará una petición a Guaraní para cerrar la sesión que está actualmente abierta, cancelará la alarma de la aplicación, se matará al servicio

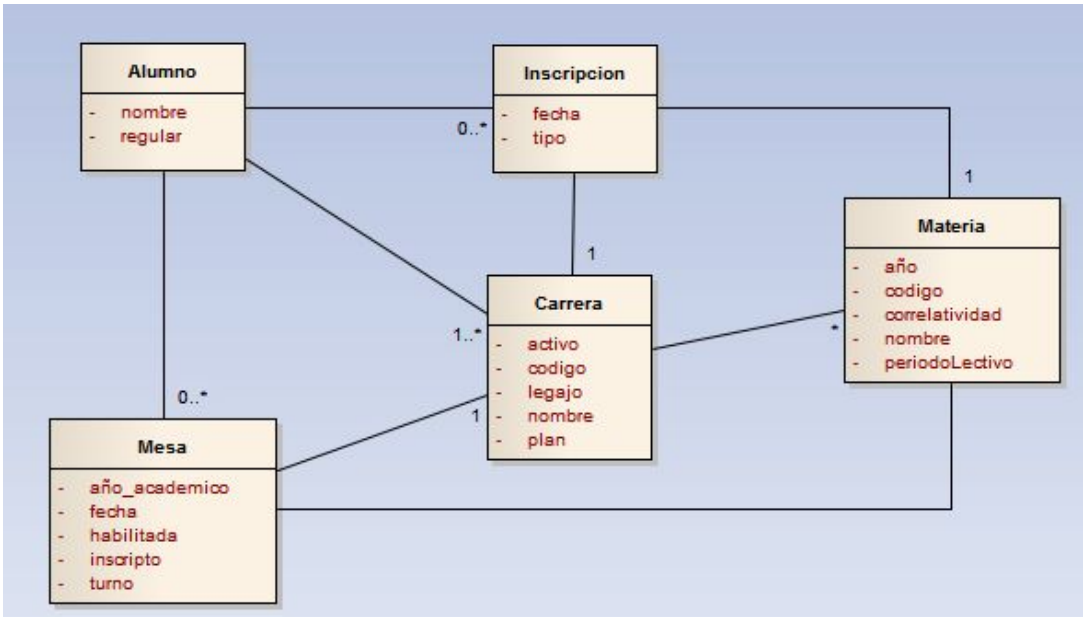
(si es que está corriendo), se actualizarán las preferencias, y finalmente nos mostrará la pantalla de usuario.

Arquitectura de la Aplicación



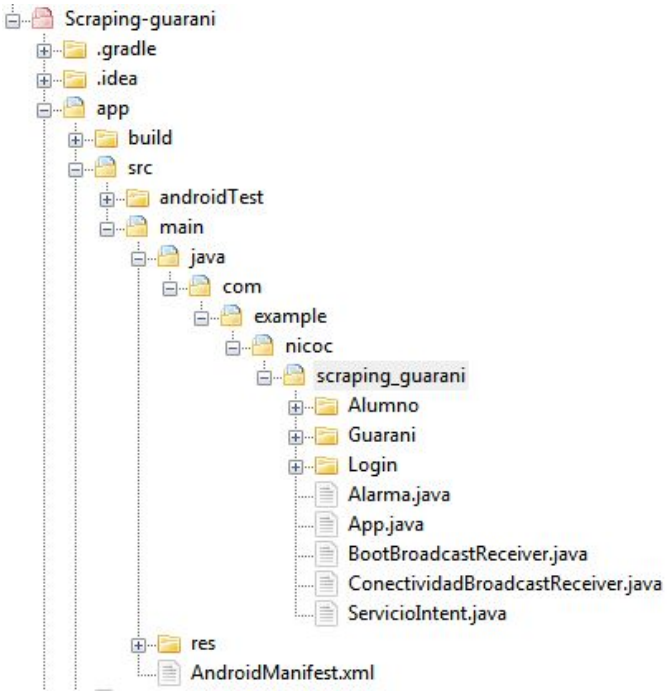
Modelo de Dominio GUARANÍ

Para el desarrollo de la aplicación nos basamos en el siguiente modelo que hemos realizado:



Estructura de carpetas de la Aplicación

La estructura de carpetas de nuestra aplicación "Scraping-guarani" se muestra en la siguiente imagen:



La carpeta Java contiene los archivos fuentes de la aplicación. Podemos ver que hay 3 carpetas principales:

- **Alumno**: contiene la funcionalidad relacionada con los fragments de mesas, AsyncTask para inscribirse/desinscribirse a una mesa de examen y desloguearse de

GUARANÍ, y clases para la gestión de la Interfaz de Alumno y del alumno que se ha logueado.

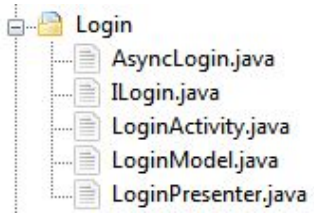
- **Guaraní**: contiene por un lado las clases que representan el modelo de dominio GUARANÍ (Alumno, Carrera, Inscripción, Mesa, Materia, Auth) y por otro lado, la clase Guarani.java que encapsula toda la funcionalidad de scraping hacia el sistema SIU GUARANÍ.
- **Login**: contiene la clase LoginActivity.java que es la que se encarga de mostrar la pantalla de login al usuario. Contiene un AsyncTask para iniciar sesión llamado AsyncLogin.java.

El resto de las clases que están sin una carpeta particular, es decir:

- Alarma.java
- BootBroadcastReceiver.java
- ConectividadBroadcastReceiver.java
- ServicioIntent.java

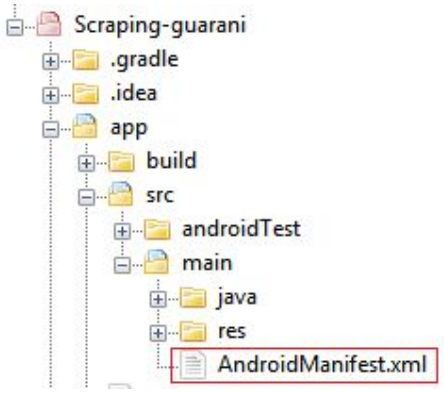
Encapsulan toda la funcionalidad del servicio que se encarga de consultar las mesas/inscripciones a GUARANÍ (scrapeando datos a partir de los métodos que provee la clase Guarani.java).

Se optó por seguir estrictamente el patrón arquitectónico MVP (Modelo-Vista-Presentador) . Por ejemplo se desplegamos la carpeta "Login" veremos lo siguiente:

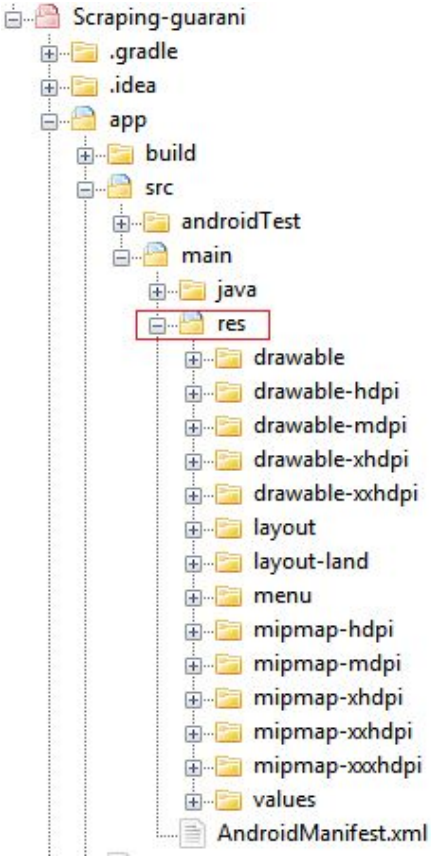


Modelo: está representado por la clase LoginModel.java. Maneja todas las operaciones con datos necesarios para el logueo del usuario.
Vista: está representada por la clase LoginActivity.java. La vista es una interfaz pasiva que exhibe datos (el modelo) y órdenes de usuario de las rutas (eventos) al presentador para actuar sobre los datos.
Presentador: está representado por la clase LoginPresenter,java. Se encarga de recibir las órdenes del usuario y de solicitar los datos al modelo para comunicárselos a la vista.

El archivo Manifest Contiene el manifesto de la aplicación principal, esto es, la declaración de cada componente de aplicación (activity, services, etc.), los permisos, etc.:



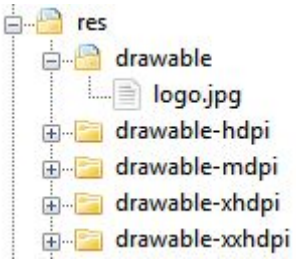
La carpeta "res" es la que contiene todos los archivos de recursos (resources) utilizados por la aplicación: imágenes, videos, cadenas de texto, etc.



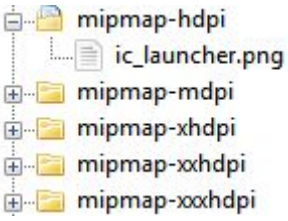
res/drawable: Contiene los elementos gráficos para las distintas resoluciones y densidades de pantallas.

- drawable-ldpi (densidad baja)
- drawable-mdpi (densidad media)
- drawable-hdpi (densidad alta)
- drawable-xhdpi (densidad muy alta)

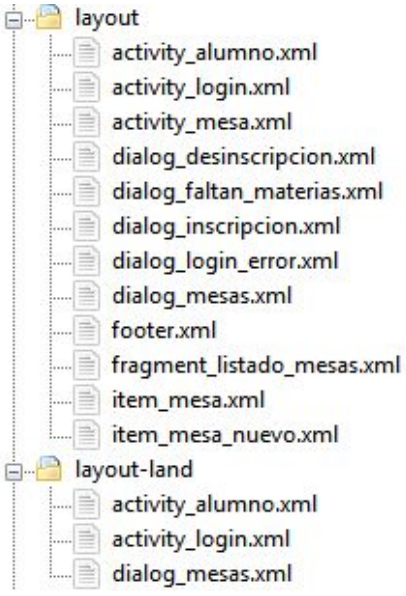
Dentro de esta carpeta nosotros hemos puesto el logo de la aplicación llamado "logo.png".



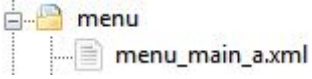
res/mipmap: Contiene los iconos para las distintas densidades de pantallas. Aquí incorporamos el ícono de la aplicación llamado "ic_launcher.png".



res/layout: Contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica. La carpeta layout incorpora los XML cuando estamos en modo Portrait (más alto que ancho), mientras que la carpeta layout-land contiene los XML cuando estamos en modo Landscape (más ancho que largo).

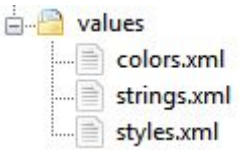


res/menu: Contiene la definición XML de los menús de la aplicación.



El archivo menu_main_a.xml lo usamos cuando estamos en la activity AlumnoActivity.java y contiene un item que nos permite cerrar sesión.

res/values: Contiene archivos XML de recursos de la aplicación, como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), arrays de valores (arrays.xml), etc.



Tecnologías y Herramientas utilizadas en el desarrollo de la Aplicación

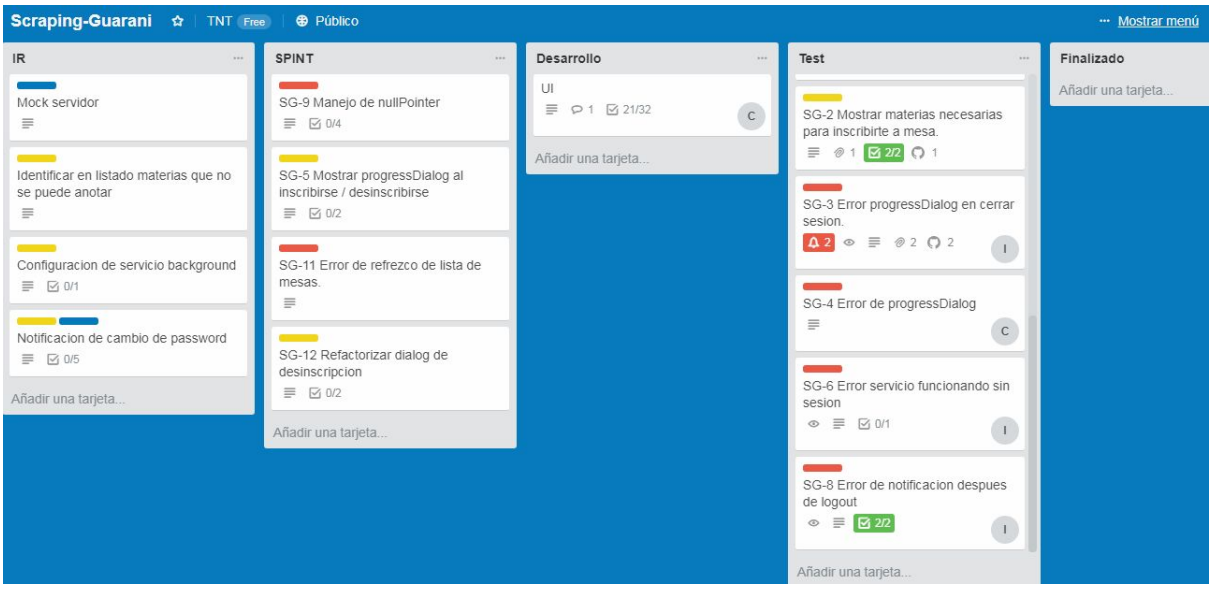
Para desarrollar aplicaciones Android necesitamos:

- Sun's Java Development Kit (JDK, v7)
- Android Software Developer's Kit (SDK)
- Android Studio

WGet: Se trata de una herramienta libre que permite la descarga de contenidos desde servidores web de una forma simple. Su nombre deriva de World Wide Web (w), y de «obtener» (en inglés get), esto quiere decir: obtener desde la WWW. Lo utilizamos para descargarnos las páginas web necesarias de SIU GUARANÍ.

Trello: Para especificar la funcionalidad de la app y ver qué es lo que está haciendo cada uno en un determinado momento utilizamos la herramienta Trello. Trello es un software de administración de proyectos. Emplea el sistema kanban, para el registro de actividades a través de tarjetas virtuales. Permite organizar tareas, agregar listas, adjuntar archivos, etiquetar eventos, agregar comentarios y compartir tableros. Trello es un tablero virtual en el que se pueden colgar ideas, tareas, imágenes o enlaces.

El tablero de Trello se puede ver en: https://trello.com/b/b76gl2IC/scraping-guarani



GSON; hemos utilizado GSON. Gson (también conocido como Google Gson) es una biblioteca de código abierto para el lenguaje de programación Java que permite la serialización y deserialización entre objetos Java y su representación en notación JSON.
Nosotros usamos esta librería para manejar los objetos (alumnos, mesas, inscripciones, etc.) y guardarlos en las preferencias como JSON. JSON es el acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. JSON es un formato de texto para la serialización de datos estructurados.
Para utilizar GSON simplemente deberemos añadir una línea en el archivo .gradle para incluirlo en nuestro proyecto:

compile 'com.google.code.gson:gson:2.3.1'

```
String mesas_json_guardadas = preferences.getString("mesas", "");
Type collectionType = new TypeToken<ArrayList<Mesa>>(){}.getType();
ArrayList<Mesa> mesas_guardadas = new Gson().fromJson(mesas_json_guardadas,
collectionType);
```

ButterKnife: es una librería desarrollada y mantenida por Jake Wharton (Square Inc.) que nos facilita la tarea de relacionar los elementos de las vistas con el código en las aplicaciones Android. Nos evita tener que utilizar findViewById y simplifica el código. No solo podemos utilizar Butter knife para inyectar vistas, también ofrece otro tipo de anotaciones para manejar eventos como OnClick(), OnLongClick(), etc. Para instalar Butter Knife solo tendremos que añadir la siguiente dependencia al archivo build.gradle del proyecto:

compile 'com.jakewharton:butterknife:7.0.1'

Ejemplo de uso de Butter Knife:

```
1  public class MainActivity extends Activity{
2
3      @Bind(R.id.my_textview) TextView myTextview;
4
5      @Override
6      protected void onCreate(Bundle savedInstanceState) {
7          super.onCreate(savedInstanceState);
8          setContentView(R.layout.main_activity);
9          ButterKnife.bind(this);
10
11         myTextview.setText("Texto de ejemplo");
12     }
13
14     @OnClick(R.id.my_textview)
15     public void submit() {
16         Toast.makeText(this, "Pulso sobre el TextView", Toast.LENGTH_LONG).show();
17     }
18 }
```

JSOUP: es una librería Java que facilita mucho la labor de scrapeo. Provee una API para extraer y manipular datos. La manera de seleccionar los elementos del DOM es similar a jQuery. Permite:

- Scrapear y parsear HTML desde una URL, archivo o String.
- Buscar y extraer datos, usando DOM o selectores CSS.
- Manipular los elementos, atributos y texto HTML.
- Entre otros.

Web Scraping desde la aplicación

El Web Scraping (o Scraping) son un conjunto de técnicas que se utilizan para obtener de forma automática el contenido que hay en páginas web a través de su código HTML.
Toda la funcionalidad de scraping hacia la página de SIU GUARANÍ está encapsulada dentro de la clase java Guarani. También encapsula comportamiento para loguearse/desloguearse en SIU GUARANÍ e inscribirse/desinscribirse a una mesa de examen da través peticiones HTTP, por lo que deben ejecutarse en tareas en segundo plano (ya que son tareas pesada que bloquearán el UI THREAD). La clase Guarani es un singleton (es decir, la idea es que haya una única instancia Guarani y proporcionar un punto de acceso global a ella), con métodos synchronized para evitar inconsistencias entre hilos.
Presenta una API que nos permite:

- Realiza una inscripción a una mesa, para lo cual requiere una instancia del Alumno que se inscribirá a la mesa, instancia de la Mesa a la que se va a inscribir, el tipo de inscripción (regular o libre).
- Devolver un listado de las inscripciones de un alumno. Representan inscripciones que pueden ser canceladas.
- Desinscribirse de una mesa de examen, identificándola por código de carrera y materia.
- Obtener todas las mesas de examen habilitadas para inscripción.
- Loguearse/Desloguearse. Si existe una sesion activa, cerramos sesion contra el servidor guarani.
- Obtener los datos del alumno, nombre, carreras y respectivas materias.

El Servicio de la Aplicación

Parte del objetivo de nuestra aplicación consiste en scrapear los datos de mesas e inscripciones de los alumnos desde la página de SIU GUARANÍ. Esta tarea es muy pesada, por lo que si la ejecutamos desde el UI THREAD nos aparecerá el Dialog ANR (Application Not Responding). Para solucionar este problema, incluimos estas tareas en un tipo especial de servicio llamado Intent Service. La clase que lo implementa se llama ServicioIntent.java.
Básicamente la diferencia que existe entre un Servicio y un Intent Service es que el Servicio comparte el proceso de ejecución con la Aplicación, mientras que el Intent Service no lo hace, es decir, corre en otro hilo de ejecución, por lo que no bloqueará al UI THREAD. Además, los Intent Service están diseñados justamente para realizar tareas pesadas, como

pueden ser subir/descargar un archivo, geofencing, y en nuestro caso, realizar scraping. Usualmente los Intent Service no se comunican con el hilo principal.

A modo de comentario, hemos hecho la prueba de ejecutar la aplicación con un Start Service corriendo como un hilo de ejecución a partir (extendiendo de la clase Thread) y con un Intent Service, para ver si existía alguna diferencia de rendimiento. Llegamos a la conclusión a través de las pruebas que el Intent Service ejecutaba su tarea mucho más rápidamente que el Start Service, y eso nos llevó a tomar la decisión de utilizar un Intent Service en nuestra aplicación.

Cuando instanciamos un IntentService, se ejecuta automáticamente el método

onHandleIntent(Intent intent)

el cual posee el código que especifica el servicio que debe realizarse. El intent pasado como parámetro se puede usar para pasar al servicio los datos de entrada necesarios.
Las tareas que realiza nuestro Intent Service son las siguientes:

- Verifica si hay conexión (WiFi, 3G, 4G). Si no hay, termina.
- Mientras haya conexión y el usuario esté logueado, se realiza scraping a SIU GUARANÍ para obtener las mesas de examen y las inscripciones.
- Comunicar al usuario cuando haya nuevas mesas e inscripciones.

Hay que aclarar que para que nuestro servicio pueda consultar si hay conexión WiFi , 4G, etc, se deben poner los siguientes permisos dentro del AndroidManifest.xml de la aplicación:

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

La clase ConnectivityManager es la que se encarga de responder consultas sobre el estado de la conectividad de red. Además notifica a las aplicaciones cuando la conectividad de red cambia.
Las principales responsabilidades de esta clase son:

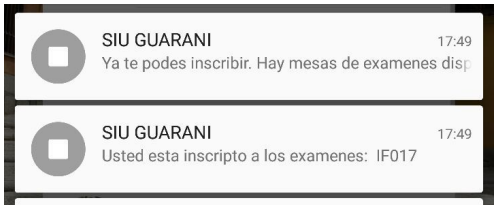
- Monitorear la conectividad de red (Wi-Fi, GPRS, UMTS, etc.).
- Enviar intents broadcast cuando la conectividad de red cambia.
- Intentar "conmutar" a otra red cuando se pierde la conectividad de una red.
- Provee una API que permite a las aplicaciones hacer consultas del estado de grano grueso o fino de las redes disponibles.
- Provee una API que permite a las aplicaciones hacer peticiones y seleccionar redes para su tráfico de datos.

Para comunicar al alumno la existencia de nuevas mesas e inscripciones utilizamos dos componentes que nos ofrece Android que son las notificaciones y los broadcasts. La idea es que la notificación sólo se muestre cuando el usuario no está usando la app. En caso de que

de que la esté usando, no habrá notificación, y será el broadcast el que se encargue de comunicarle y realizar las acciones pertinentes.

Las notificaciones se componen de un título, un mensaje y un ícono pequeño, son administradas por el Notification Manager de la plataforma, y se muestran en el área de notificaciones de nuestro dispositivo. Nosotros le agregamos la capacidad de hacer vibrar y sonar el dispositivo para que el usuario se de cuenta inmediatamente de la presencia de una notificación. También le agregamos un Pending Intent, para que cuando el usuario presione sobre la notificación abra inmediatamente la aplicación con el listado de mesas e inscripciones actualizadas.

La clase NotificationManager se encarga de gestionar las notificaciones que se deben mostrar en la barra de notificaciones del dispositivo. Para poder mostrar una notificación, por lo tanto, necesitaremos crear un objeto de esta clase y configurar los parámetros necesarios de la notificación que queremos mostrar. Las notificaciones pueden tomar diferentes formas, pueden hacer vibrar el dispositivo, prender LEDs, etc.
Un Pending Intent es parecido a un Intent. Se ofrece a una aplicación externa o de terceros, para proporcionarle permiso para ejecutar determinado fragmento del código de una aplicación. Se utiliza a menudo con clases como NotificationManager, AlarmManager o AppWidgetManager.



Por otro lado, cuando comunicamos a través de broadcasts, lo que hacemos es crear una instancia de LocalBroadcastManager el cual se encargará de difundir a toda la aplicación el intent que le hemos agregado. Nosotros simplemente le definimos un nombre al intent, para que los componentes que estén escuchando puedan capturarlo. En nuestra app, al intent le incorporamos el mensaje "MesasActivity". La clase AlumnoActivity es la que estará a la escucha de estos broadcast (solamente cuando esté siendo usada por el usuario) y se encargará de actualizar el fragment de mesas e inscripciones, y mostrar un Toast al usuario con el mensaje "Nuevas mesas de examen disponibles" o "Actualizando inscripciones".

La clase LocalBroadcastManager es la que ayuda a registrar o enviar broadcasts de intents a objetos locales dentro de nuestro proceso. Esto tiene una serie de ventajas sobre enviar un broadcast global a partir del método sendBroadcast(Intent):

- Los datos que estamos broadcasteando no sobrepasarán la aplicación, por lo que no hay que preocuparse por la filtración de datos privados.

- Es imposible para otras aplicaciones enviar estos broadcast hacia mi aplicación, por lo que no hay que preocuparse por tener agujeros de seguridad que puedan explotar.
- Es más eficiente que enviar un broadcast global a través del sistema.

La Alarma de la Aplicación

El Intent Service nos resuelve el problema de las tareas pesadas. Debemos ejecutar ese servicio con una cierta frecuencia para poder scrapear datos de las mesas e inscripciones. Esto lo podemos realizar utilizando la clase AlarmManager.
Esta clase proporciona acceso a los servicios de alarma del sistema. Esto permite programar aplicaciones que se ejecuten en algún momento en el futuro. Cuando se activa una alarma, el Intent que se ha registrado en ella se transmite por el sistema, iniciando automáticamente la aplicación de destino si no se está ejecutando. Las alarmas registradas se conservan mientras el dispositivo está dormido (y opcionalmente pueden reactivar el dispositivo si se apagan durante ese tiempo), pero se borrará si se apaga y se reinicia.
La clase AlarmManager está diseñada para casos en los que se desee que el código de la aplicación se ejecute en un momento específico.
Si bien esta clase tiene varios métodos, nosotros hemos utilizado uno llamado

setRepeating(…).

Su estructura es la siguiente:
void setRepeating (int type,

long triggerAtMillis,

long intervalMillis,

PendingIntent operation)

Éste método permite programar una alarma repetitiva. Lo bueno de este método es que nos permite repetir nuestras aplicaciones en un período de tiempo especificado. Esta alarma continuará repitiéndose hasta que se elimine explícitamente con el método cancelar(AlarmManager.OnAlarmListener). Si el tiempo de disparo establecido está en el pasado, la alarma se activará de inmediato, con un conteo de alarmas que dependerá de cuán lejos en el pasado el tiempo de disparo sea relativo al intervalo de repetición.

Parámetros	
type	int: tipo de alarma. RTC_WAKEUP: Alarm time in System.currentTimeMillis() (wall clock time in UTC). despertará el dispositivo cuando se apaga. RTC: Alarm time in System.currentTimeMillis() (wall clock time in UTC). Esta alarma no activa el dispositivo; si se apaga mientras el dispositivo está dormido, no se enviará hasta la próxima vez que el dispositivo se despierte. ELAPSED_REALTIME_WAKEUP: Alarm time in SystemClock.elapsedRealtime()(tiempo desde el inicio, incluido el modo de suspensión), que activará el dispositivo cuando se apague. ELAPSED_REALTIME: Alarm time in SystemClock.elapsedRealtime()(tiempo desde el inicio, incluido el modo de suspensión), esta alarma no activa el dispositivo; si se apaga mientras el dispositivo está dormido, no se enviará hasta la próxima vez que el dispositivo se despierte.
triggerAtMillis	long: tiempo en milisegundos a partir del cual la alarma se activará.
intervalMillis	long: intervalo en milisegundos entre repeticiones posteriores de la alarma.
operation	PendingIntent: Acción a realizar cuando la alarma se dispara.

En pocas palabras, lo que hicimos nosotros fue configurar una alarma cuyo horario de comienza será a las 12:00 PM, y se repetirá cada 12 hs, ejecutando el servicio definido en ServiceIntent. Todo esto está contenido en la clase Alarma.java, la cual extiende de Thread.

Los Broadcast Receivers de la Aplicación

Nuestra aplicación está compuesta por dos Broadcast Receivers: BootBroadcastReceiver.java y ConectividadBroadcastReceiver.java. El primero de ellos se encarga de reiniciar la alarma cuando el dispositivo se apaga o se reinicia, ya que ante estos dos casos la alarma muere, por lo que el servicio no se ejecutará nunca más. Para poder hacer esto, debemos definir el siguiente permiso dentro del AndroidManifest.xml de la aplicación:

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

La clase ConectividadBroadcastReceiver.java se encarga simplemente de monitorear constantemente la conectividad de la red (WiFi, 3G, 4G). Pero sólo lo va a hacer cuando nosotros queremos que lo haga. La necesidad de esto surge en la clase ServicioIntent. Éste, antes de scrapear, pregunta si hay conectividad. En caso de que no haya, utiliza las preferencias y setea el valor de la preferencia "aviso" a "true" y luego muere. El broadcast, detecta este cambio del valor de la preferencia de falso a verdadero y se activa. Cuando detecta conectividad, lo que hace es reiniciar de nuevo el servicio.

Los Dialogs de la Aplicación

La plataforma Android nos provee distintos modos de brindar información al usuario, las notificaciones (que ya las hemos mencionado) son uno de estos modos. Otra forma son los AlertDialogs. Estos muestran una ventana que toma el foco de la aplicación a la espera de que el usuario haga algo, normalmente confirmar, aceptar o cancelar una acción.

En principio, los diálogos de Android los podremos utilizar con distintos fines, en general:

- Mostrar un mensaje.
- Pedir una confirmación rápida.
- Solicitar al usuario una elección (simple o múltiple) entre varias alternativas.

Para construir nuestros diálogos de alerta utilizamos la clase AlertDialog.Builder que es muy flexible y nos da la posibilidad de construir diálogos tan complejos y específicos como necesitemos.

La clase AlertDialog.Builder ya nos provee todo lo necesario para mostrar un AlertDialog simple con unas pocas líneas de código, con un título, un mensaje y un botón, como por ejemplo:

Nosotros hemos definido Dialogs más complejos, primero definiendo parte de su estructura en un archivo layout.xml, para luego inflarlo a partir del mismo. Realizar un 'inflate' en Android es instanciar un xml para poder añadirlo a una jerarquía de vistas. El LayoutInflater es un servicio propio de Android que nos facilita el mecanismo de "inflado" de los XML. Básicamente lo que queremos decir con "Inflar" es agregar una vista (.xml) a nuestros dialogs en tiempo de ejecución.

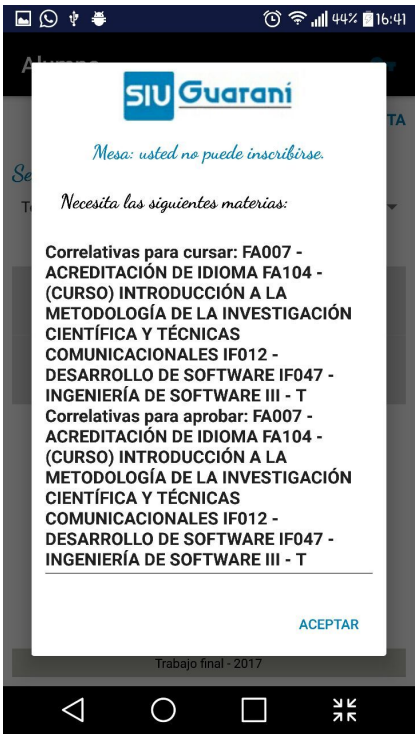
Nosotros hemos definido los siguientes layouts para inflar los Dialogs:

- **dialog_desinscripcion.xml:** lo usamos para mostrar un AlertDialog que permite desinscribirte de una mesa de examen, mostrando los datos de inscripción.
- **dialog_faltan_materias.xml:** lo usamos para mostrar un AlertDialog que le indica al usuario que no puede inscribirse a una mesa de examen porque le faltan materias correlativas.
- **dialog_inscripcion.xml:** lo usamos para mostrar un AlertDialog que permite inscribirte de una mesa de examen. Muestra los datos de dicha mesa.

La siguiente imagen muestra el Dialog de desinscripción:

La siguiente figura muestra el Dialog de inscripción:

La siguiente Figura muestra el Dialog de correlativas:



AsyncTask en nuestra Aplicación

Todos los componentes de una aplicación Android, tanto las actividades como los broadcast receivers se ejecutan en el mismo hilo de ejecución, llamado hilo principal, main thread o GUI thread. Es el hilo donde se ejecutan todas las operaciones que gestionan la interfaz de usuario de la aplicación.

Cualquier operación larga o costosa que realicemos en este hilo va a bloquear la ejecución del resto de componentes de la aplicación así como también la interfaz, produciendo al usuario un efecto de lentitud, bloqueo, o mal funcionamiento, esto es algo que deberíamos evitar siempre, como ya hemos mencionado.

Android monitorea las operaciones realizadas en el hilo principal y detecta aquellas que superen los 5 segundos, en este caso se muestra el famoso mensaje de "Application Not Responding" (ANR) y el usuario debe decidir entre forzar el cierre de la aplicación o esperar a que termine la ejecución de la misma. Es por eso que para ejecutar operaciones de larga duración se utilizan los procesos en segundo plano. Existen algunas alternativas a la hora de ejecutar tareas en segundo plano en Android:

- Crear nosotros mismos de forma explícita un nuevo hilo para ejecutar nuestra tarea.
- Utilizar la clase auxiliar AsyncTask proporcionada por Android.
- Utilizar la clase IntentService proporcionada por Android.

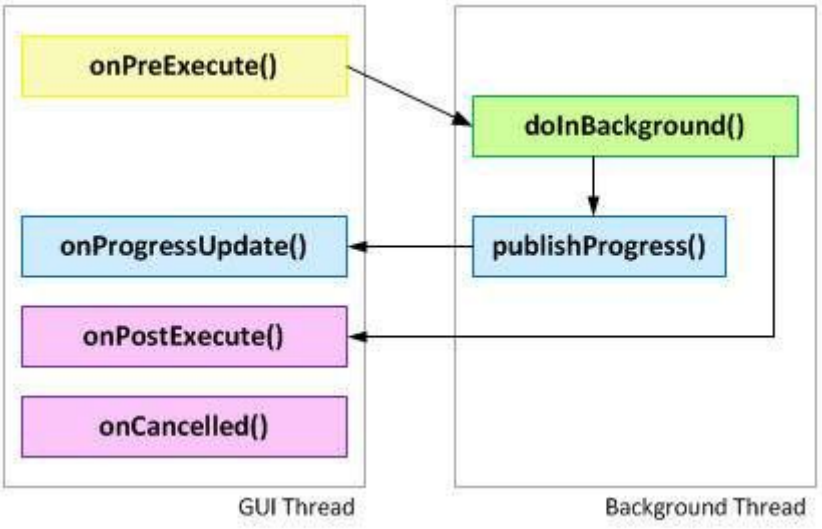
La primera opción no es demasiado utilizada ya que con la segunda, es más que suficiente para tratar los problemas de ejecución de procesos costosos en cuantos a tiempo.

Android provee la posibilidad de realizar tareas asincrónicas a través de la clase AsyncTask. Nos permite realizar operaciones en segundo plano y luego utilizar sus resultados en el hilo principal. Esto evita que debamos controlar los hilos de ejecución por nosotros mismos.

La forma básica de utilizar la clase AsyncTask consiste en crear una nueva clase que extienda de ella y sobrescribir varios de sus métodos entre los que repartiremos la funcionalidad de nuestra tarea. Estos métodos son los siguientes:

- **onPreExecute().** Se ejecutará antes del código principal de nuestra tarea. Se suele utilizar para preparar la ejecución de la tarea, inicializar la interfaz, etc.
- **doInBackground().** Contendrá el código principal de nuestra tarea.
- **onProgressUpdate().** Se ejecutará cada vez que llamemos al método **publishProgress()** desde el método doInBackground().
- **onPostExecute().** Se ejecutará cuando finalice nuestra tarea, o dicho de otra forma, tras la finalización del método doInBackground().
- **onCancelled().** Se ejecutará cuando se cancele la ejecución de la tarea antes de su finalización normal.

El método doInBackground() se ejecuta en un hilo secundario (por lo tanto NO podremos interactuar con la interfaz). Sin embargo todos los demás se ejecutan en el hilo principal, lo que quiere decir que dentro de ellos podremos hacer referencia directa a nuestros controles de usuario para actualizar la interfaz.
Dentro de doInBackground() tendremos la posibilidad de llamar periódicamente al método publishProgress() para que automáticamente desde el método onProgressUpdate() se actualice la interfaz si es necesario.
El siguiente gráfico clarifica lo anteriormente mencionado:



Por otro lado, al extender una nueva clase de AsyncTask tendremos que indicar tres parámetros de tipo:

private class MyTask extends AsyncTask<A, B, C> { ... }

El primero indica tipo de parámetros que recibirá la tarea para ser ejecutada. Es decir, el tipo de datos que recibiremos como entrada de la tarea en el método doInBackground().

El segundo indica el progreso, es decir, el tipo correspondiente a la unidad de progreso de la tarea que correrá en segundo plano. Es decir, el tipo de datos con el que actualizaremos el progreso de la tarea, y que recibiremos como parámetro del método onProgressUpdate() y que a su vez tendremos que incluir como parámetro del método publishProgress().

El tercero indica el tipo de resultado. Es decir, el tipo de datos que devolveremos como resultado de nuestra tarea, que será el tipo de retorno del método doInBackground() y el tipo del parámetro recibido en el método onPostExecute().

No todas las tareas necesitan de las tres cosas. Utilizamos Void si algo no es necesario.

private class MyTask extends AsyncTask<Void, Void, Void> { ... }

Cabe mencionar que todo lo que podemos hacer con una AsyncTask lo podemos hacer con un Intent Service. A diferencia de las AsyncTask, un IntentService no proporciona métodos que se ejecuten en el hilo principal de la aplicación y que podamos aprovechar para "comunicarnos" con nuestra interfaz durante la ejecución. Éste es el motivo principal de que los IntentService sean una opción menos utilizada a la hora de ejecutar tareas que requieran cierta vinculación con la interfaz de la aplicación.

Sin embargo tampoco es imposible su uso en este tipo de escenarios ya que podremos utilizar por ejemplo mensajes broadcast o notificaciones para comunicar eventos al hilo principal (como ya hemos mencionado), como por ejemplo la necesidad de actualizar controles de la interfaz o simplemente para comunicar la finalización de la tarea ejecutada.

En nuestra aplicación hemos utilizado varias tareas asincrónicas:

- **AsyncLogin.java:** recibe como parámetros el par usuario y contraseña (como Strings), inicia sesión en Guaraní y devuelve una instancia de Alumno con los datos ya seteados.
- **AsyncDesloguear.java:** no recibe nada como parámetro. Se encarga de cerrar sesión en Guaraní. No devuelve nada.
- **AsyncInscribirse.java:** recibe como parámetros una instancia de Object, al cual lo tratamos como un arreglo, que contiene el alumno, la mesa y el tipo de inscripción (regular/libre), y con esos datos hace una petición post a Guaraní para inscribirse a una mesa. Devuelve como resultado un booleano, el cual indica verdadero cuando se pudo inscribir correctamente a la mesa, o falso si ha ocurrido algún error.
- **AsyncDesinscribirse.java:** recibe como parámetro el par código de la carrera y código de la materia (como un String) y con ellos hace una petición a Guaraní para desinscribirse de una mesa de examen. Devuelve como resultado un booleano, que indica verdadero o falso según el éxito que tuvo al realizar la transacción.

Las Preferencias en nuestra Aplicación

La aplicación que hemos desarrollado no utiliza una Base de Datos. Por eso, como forma de hacer persistir los datos, utilizamos las Preferencias que nos ofrece la plataforma Android.
En los componentes de nuestra App podremos acceder a las preferencias guardadas a través del manager de preferencias **PreferenceManager**. Las preferencias son manipuladas a través de un objeto SharedPreferences.
Nosotros, programáticamente guardamos preferencias sin la intervención del usuario. Para ello tenemos que obtener un objeto **editor** de preferencias. Luego realizaremos los cambios que necesitemos y luego comprometeremos los cambios a través de commit(). Por ejemplo: si queremos guardar un String "Valor" cuya clave es "clave_para_el_valor", podemos codificar lo siguiente:

```
SharedPreferences.Editor editor = sharedPref.edit();
editor.putString("clave_para_el_valor", "Valor");
editor.commit();
```

Ese par clave/valor va a persistir por más que nuestro dispositivo se reinicie o se apague. Para obtener este par, debemos hacer lo siguiente:

```
SharedPreferences preferences  =
context.getSharedPreferences("loginPrefs", MODE_PRIVATE);
String valor =
preferences.getString("clave_para_el_valor", "");
```

Con estas líneas de código hemos almacenado las mesas de examen, las inscripciones, los datos de la cuenta del alumno y otras variables que necesitábamos persistir para el buen funcionamiento de la aplicación.

Uso de Fragments en nuestra Aplicación

Se puede definir a un fragment como porciones modulares y cohesivas de interfaz con su propio ciclo de vida. Utilizamos fragments cuando queremos aprovechar mejor las dimensiones de la pantalla. Los fragments, siempre son hospedados por actividades. Cada fragment especificará su propio layout pudiendo este a su vez ser responsivo. Nosotros hemos definido el layout fragment_listado_mesas.xml, el cual justamente contendrá un listado de mesas e inscripciones, junto al spinner que permite seleccionar la Carrera. Dicho fragment está contenido dentro de activity_alumno.xml y se mapea en tiempo de ejecución con la clase ListadoMesasFragment.java, en la cual se definen los métodos del ciclo de vida.
El ciclo de vida de un fragment está fuertemente relacionado al ciclo de vida de la actividad que lo hospeda, pero recibe sus propios callbacks en la medida que va cambiando de estado, condicionado por el estado de la actividad y por la naturaleza de su interfaz y la interacción del usuario con ella.

Mejoras Futuras

La aplicación que hemos desarrollado podría expandirse para incorporar nuevas funcionalidades o mejorar las que ya tiene, entre ellas:

- Desarrollar un servidor que simule al servidor Guaraní para poder realizar peticiones, ya sea de login/logout, inscripción/desinscripción etc.
- Actualmente el servicio de sincronización con el servidor Guarani es configurable, pero no para el usuario. La idea es permite al usuario cambiar los horarios de sincronización con el servidor guarani.
- El servicio en background efectúa las consultas de mesas usando username y password definidas en preferencias. Puede quedar inconsistente el password almacenado con el registrado en GUARANÍ y esto puede generar un error. La solución sería la siguiente:
    - El servicio en background envíe una notificación al usuario indicando que el password es incorrecto.

- Se lleva al usuario a una activity donde pueda ingresar su nueva password.
- Se verifica que nuevo password sea correcto.
- Se actualizan las preferencias.
- En ningún momento se eliminan los datos del alumno almacenados.

Manual de Usuario

Inicio de Sesión

Al iniciar la aplicación, se mostrará la interfaz de Login, la cual solicita que se ingrese el usuario y la contraseña que utilizamos en SIU GUARANÍ.

Podemos tildar la opción "Recordarme" para no tener que volver a escribir el usuario y la contraseña cada vez que nos deslogueamos. Al presionar "Continuar", nos estaremos logueando en dicho sitio. Una vez logueados, se nos mostrará la pantalla principal de la aplicación, la cual mostrará un listado de las mesas actuales y las inscripciones actuales en caso de que estén abiertas las inscripciones a examen en GUARANÍ, caso contrario no se mostrará nada:

Pantalla principal: Listado de Mesas e Inscripciones

Podremos filtrar las mesas/inscripciones según la carrera. Además, podemos tildar qué es lo que deseamos ver: mesas, inscripciones o ambos. Si una mesa está coloreada de verde, significa que estamos inscriptos a dicha mesa, si está coloreada de gris y posee una fecha, significa que estamos habilitados para inscribirnos a dicha mesa, y por último, si está coloreada de gris pero sin fecha, significa que no podremos inscribirnos a dicha mesa porque no tenemos las correlativas.

Inscripción a una Mesa

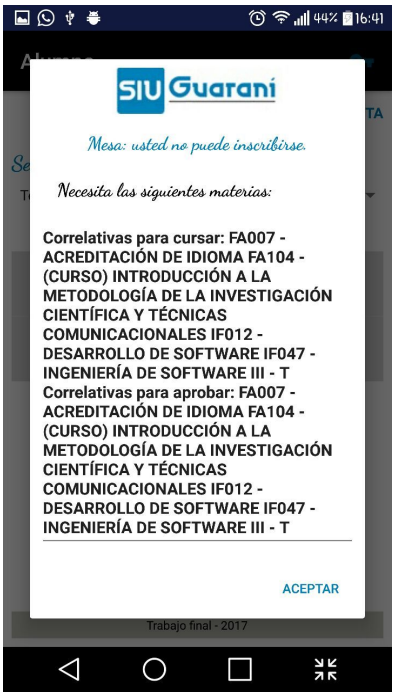
Al presionar sobre una mesa coloreada de gris y con una fecha, se nos abrirá un diálogo que mostrará el detalle de dicha mesa. Al presionar en el botón "Inscribir" se solicitará la confirmación pertinente, y si aceptamos, ya estaremos inscritos a dicha mesa (siempre y cuando esté dentro de los rangos de tiempo adecuados).

Desinscripción a una Mesa

Al presionar sobre una fila que está coloreada de verde, se nos abrirá el diálogo de desinscripción a dicha mesa. Mostrará los detalles de inscripción, y si presionamos el botón "Desinscribir" se nos solicitará la confirmación pertinente, y si aceptamos, ya estaremos desinscriptos de dicha mesa. Recordar que sólo podemos desinscribirnos hasta 48 hs antes del examen.

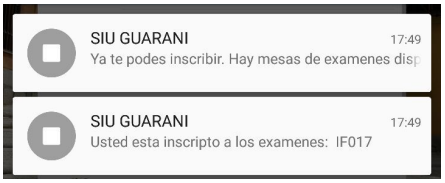
Faltan Correlativas para inscribirnos a una Mesa

Al presionar sobre una fila que está coloreada de gris y sin fecha, se nos abrirá una ventana de diálogo que nos indicará que no te puedes inscribir en la mesa porque te faltan las correlativas, y mostrará en un cuadro de texto dichas materias.



Notificación

La aplicación te avisará automáticamente a través de una notificación en el área de notificaciones de tu dispositivo cuando haya nuevas mesas de examen. Al presionar sobre ella, se abrirá la pantalla principal de la aplicación y mostrará el listado de mesas/inscripciones.

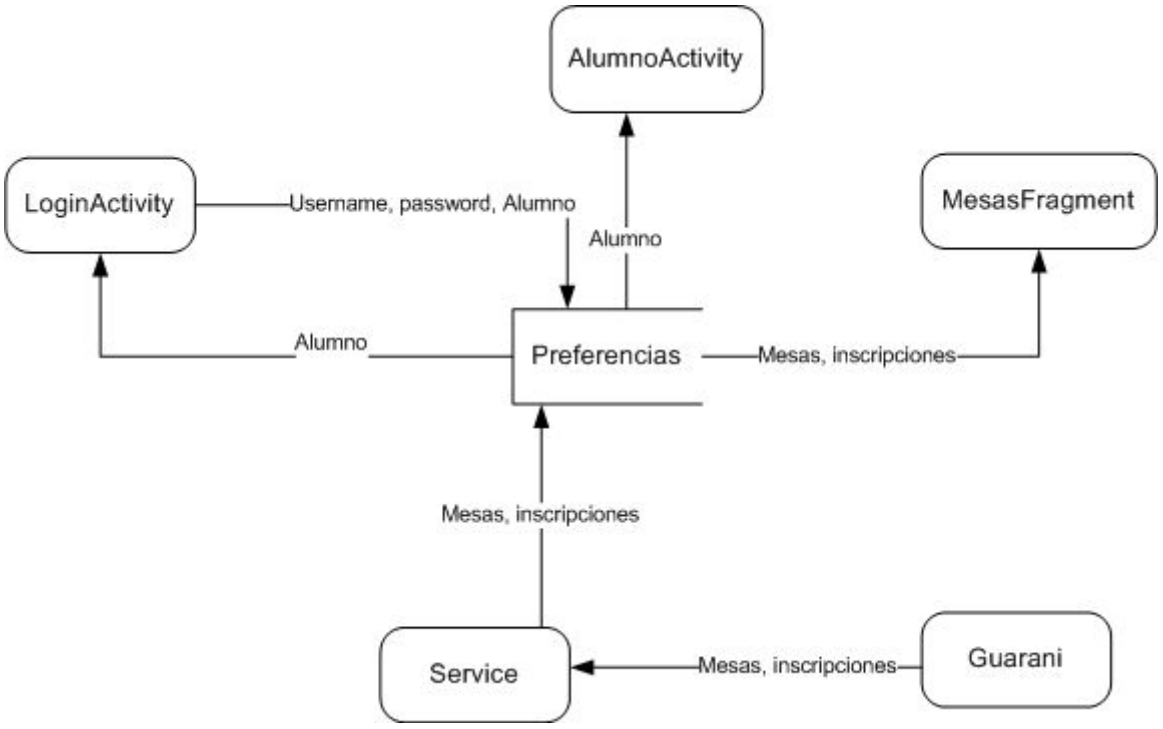


Cerrar Sesión

Al presionar en el ícono que posee una llave en el menú de la aplicación, se cerrará la sesión, tanto en la aplicación como en GUARANÍ. Hay que tener en cuenta que al cerrar sesión dejaremos de recibir notificaciones de mesas de examen.

Documentación Técnica

Flujo de datos



En este diagrama mostramos el flujo de información que se almacena en Preferencias. Podemos destacamos que es el Servicio quien recupera mesas e inscripciones de Guarani, guardandolas en preferencias para que las activities la consuman.

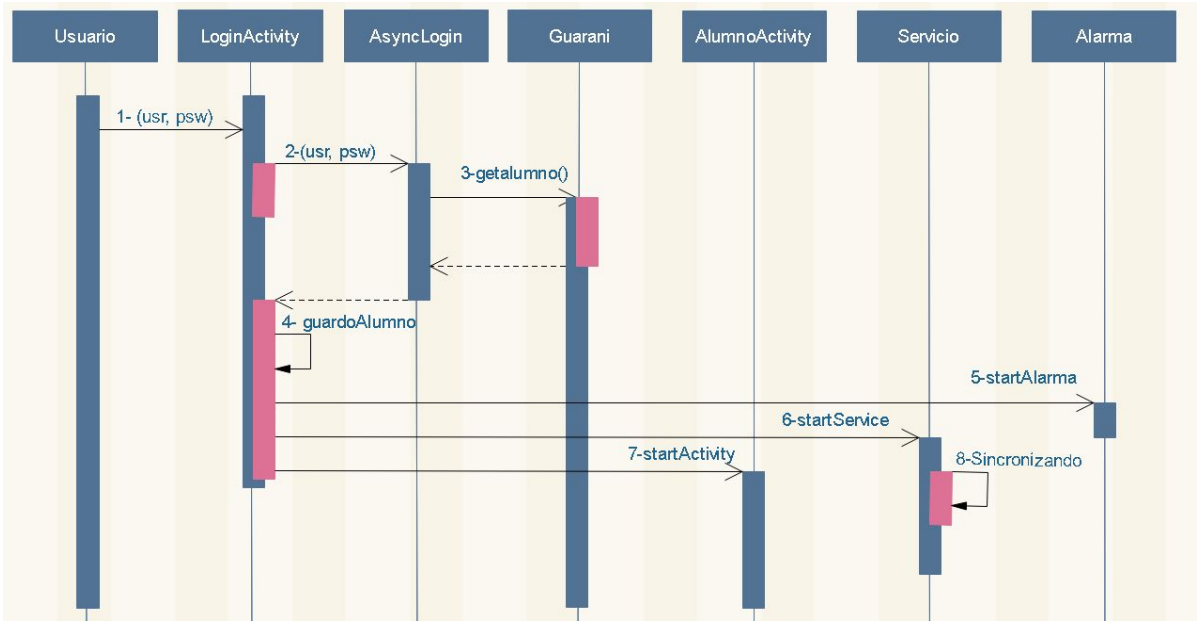
Notificaciones:
Nuestro servicio estará encargado de sincronizar periódicamente las mesas e inscripciones del alumno con el servidor Guarani. Notificara ante las siguientes situaciones:

1. Se creará una notificación y un broadcast informando que existen nuevas mesas de exámenes.
2. Se creará una notificación y un broadcast informando que se deshabilitaron mesas de examen.
3. Se creará una notificación y un broadcast informando que se actualizando las inscripciones.

Diagramas de Secuencia

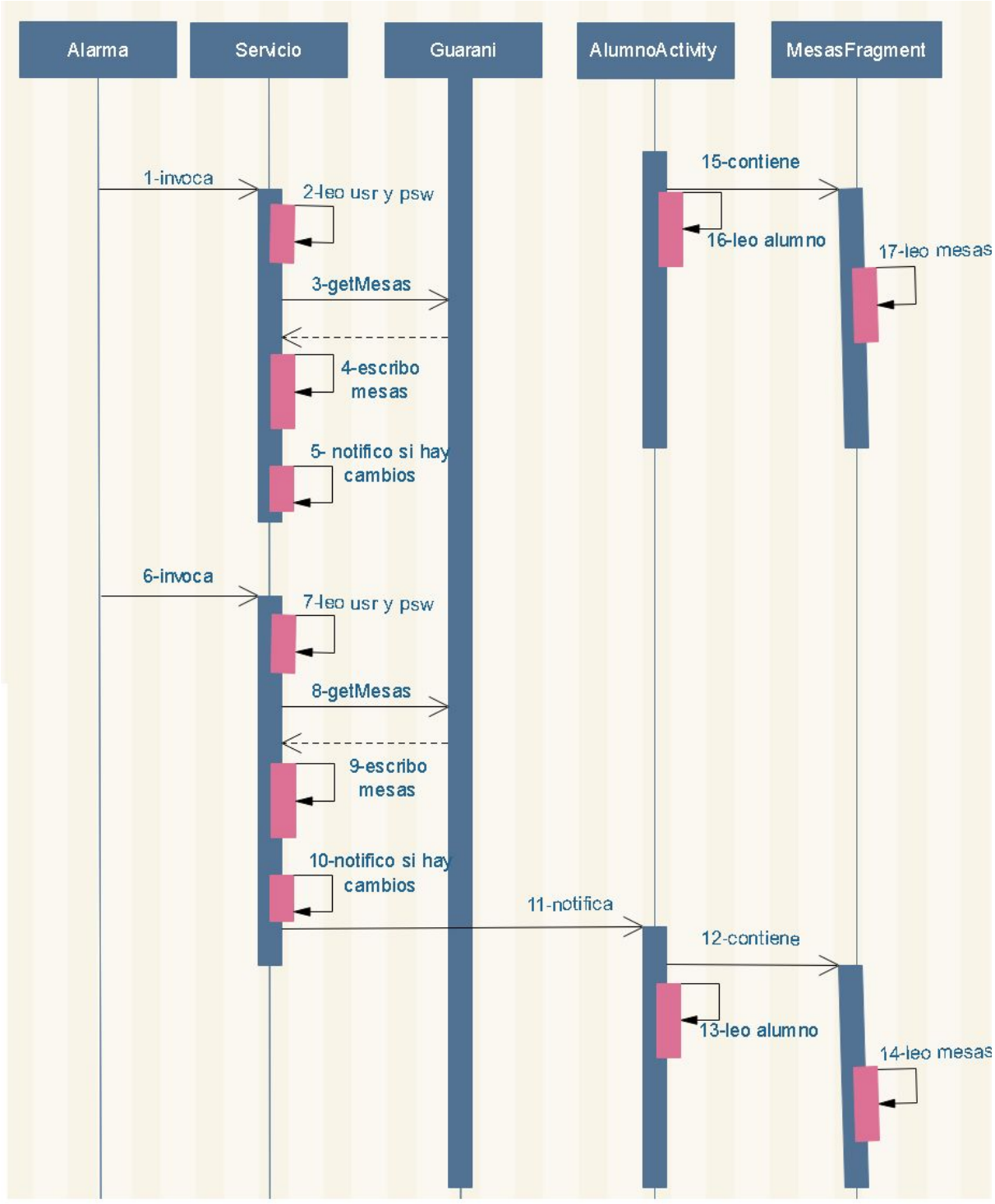
Con los siguientes diagramas pretendemos explicar el comportamiento de las principales actividades de nuestra aplicación.

Diagrama de Secuencia: Login



En este diagrama mostramos el proceso de login. Comienza con el usuario ingresando su usuario y contraseña, correspondientes con su cuenta en Siu Guarani. Luego pedimos los datos del alumno (nombre, carreras y materias) al módulo Guarani mediante un AsyncTask.
En caso que existan errores con la autenticación se informará el error al usuario.
En este ejemplo vemos obtenemos los datos del alumno y en (4) los almacenamos en Preferencias.
En este diagrama podemos destacar que la instancia Guaraní se trata de un objeto Singleton, al cual le seteamos la cuenta de autenticación mediante un método estático.
En (5) vemos que se inicia la alarma configurada para que cada determinado tiempo, inicie a nuestro servicio en background. En (6) forzamos el inicio de dicho servicio, el cual en (6) se encargará de sincronizar las mesas e inscripciones con el servidor guarani (detallaremos este comportamiento más abajo).
Por último, (7) iniciamos la activity alumno, quien será de mostrar el listado de mesas de exámenes disponibles, permitiendo al usuario realizar las operaciones requeridas.

Diagrama de Secuencia: Servicio y notificaciones.



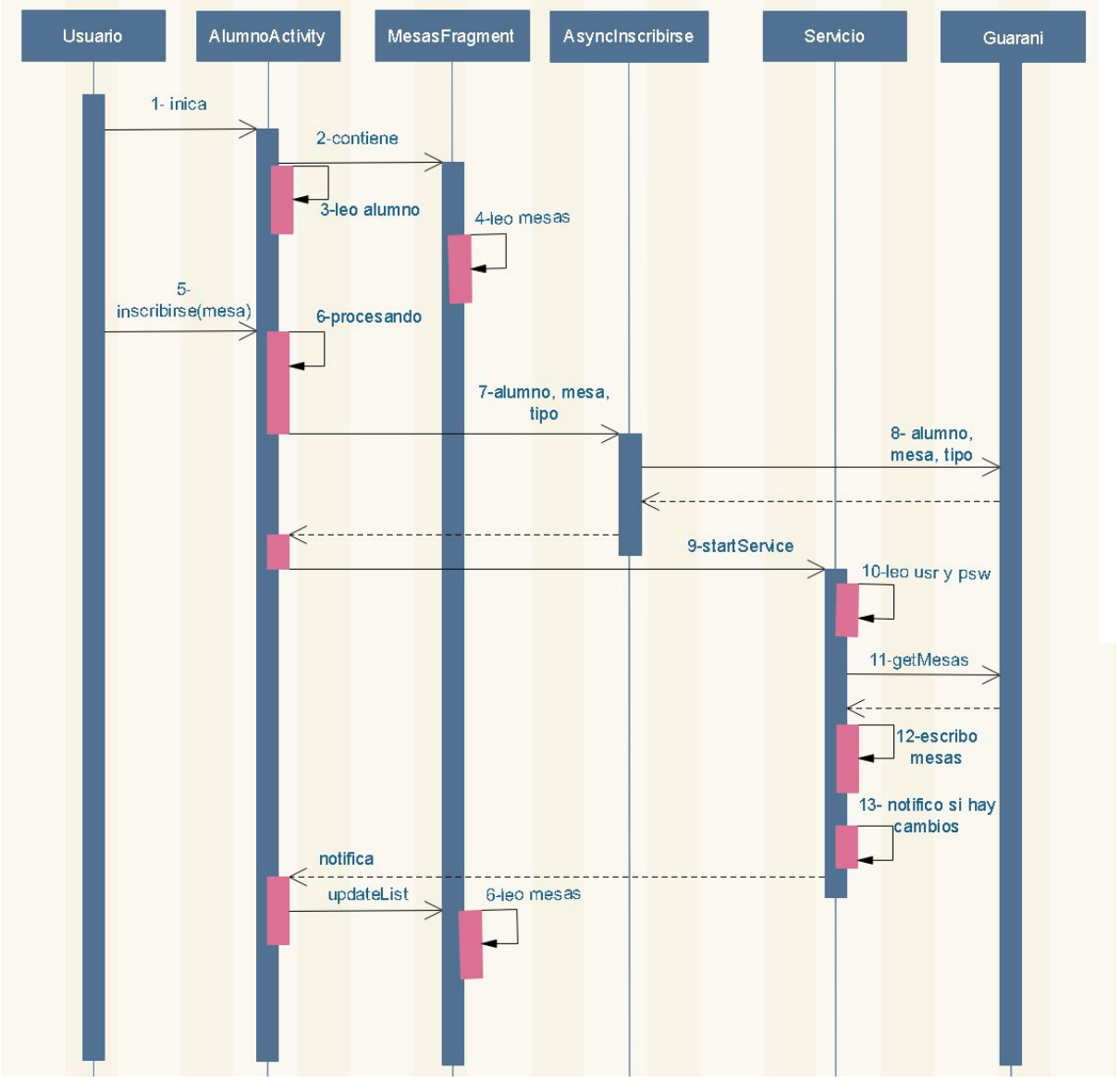
En este diagrama mostramos el funcionamiento de nuestro servicio. Cómo podemos ver, el servicio corre en background independientemente de las activities de la aplicación, es decir, sin la intervención del usuario.

Cuando se lanza la Alarma (1) se invoca a nuestro servicio el cual se encargara de pedir las mesas e inscripciones. Una vez recibidas las actualiza guardandolas en Preferencias (4) y verifica si es necesario notificar al usuario de cambios importantes. En (11) podemos ver que

se realiza una notificación por algunos de los motivos previamente mencionados, siendo la activity Alumno la encargada de atrapar la notificación para mostrar las mesas e inscripciones actualizadas al usuario.

Destacamos que el servicio se encarga de leer username y password de preferencias y setearlos al módulo guarani mediante un método estático. Si bien AsyncLogin también realiza esta acción, el servicio puede ser ejecutado independientemente de la aplicación, por lo que la instancia de Guaraní puede que sea creada cuando el servicio la solicita, requiriendo de una cuenta de autenticación.

Diagrama de Secuencia: Inscripción



En este diagrama mostramos el proceso de inscripción a una mesa de examen. Aclarar que previo a la activity alumno, se efectúa el correspondiente login y se obtiene la instancia única de Guarani.

En (5) podemos ver al usuario solicitando inscribirse a una mesa del listado. Luego de la confirmación, realizamos la inscripción con nuestro modulo Guarani mediante un AsyncTask, ya que no se permiten realizar peticiones http en el hilo principal de ejecución (activity).

Llegado al caso que guaraní retorne operación exitosa, necesitamos actualizar el listado de mesas, para determinar cuales de ellas está inscripto. En (9), forzamos a nuestro servicio a iniciar, para que sincronice las mesas e inscripciones con SIU Guarani.

El servicio actualiza en Preferencias las mesas e inscripciones (12) y notifica a la aplicación (13) para que los recupere y actualice la vista.

Reporte de vulnerabilidades de Siu Guaraní

- Podemos obtener el DNI de todos los alumnos de la facultad de ingeniería. (Por ejemplo: sacarlos del listado de padrones en las elecciones) y en base al DNI podemos bloquear la cuenta de todos los alumnos.
- Al momento de inscribirse en una mesa de examen, se tiene en cuenta el legajo del alumno que se envia por POST. Esto permite, por ejemplo: generar una inscripción con el legajo de otra persona.
- Podría ser un problema, el hecho de que guaraní permita múltiple sesiones de un mismo usuario en varios dispositivos al mismo tiempo (posible ataque de denegación de servicio).