# manual-tecnico Documentation

***Release 1.0***

**Gonzalo Guzzardi, Matias Carballo, Juan Patricio Marshall**

**Dec 06, 2016**

CONTENTS

Contents:

CHAPTER

ONE

# ACTIVITY

## 1.1 MainScreenActivity

public class **`MainScreenActivity`** extends AppCompatActivity

Actividad principal de la aplicación. Contiene 4 Fragments correspondientes a las principales ventanas de la aplicación: Perfil, Contactos, Notificaciones, Chats. Para manipular dichos Fragments, utiliza la clase *ViewPagerAdapter*.

### 1.1.1 Fields

#### mToolbar

private Toolbar **`mToolbar`**

#### mTabLayout

private TabLayout **`mTabLayout`**

Referencia al layout que contiene y establece como se muestran los tabs.

#### mViewPager

private ViewPager **`mViewPager`**

ViewPager que maneja el paginado de los distintos Fragments que componen a la Activity.

### 1.1.2 Methods

#### setupTabIcons

private void **`setupTabIcons`**()

Establece las asociaciones entre íconos y Tabs.

#### setupViewPager

private void **`setupViewPager`**(ViewPager *viewPager*)

Crea todos los Fragments a utilizar en la MainScreenActivity y los agrega a un *ViewPagerAdapter*, para luego asignarselo al viewPager pasado como parámetro.

**Parameters**

- **viewPager** – Referencia al viewPager utiliado para manejar el paginado de los Fragments. A esta ViewPager se le seteará el Adapter *ViewPagerAdapter* específico de la aplicación.

## 1.2 SingleFragmentActivity

public class **SingleFragmentActivity** extends AppCompatActivity

Actividad inicial que contiene un único Fragment para manejar su contenido visible. SingleFragmentActivity es una clase abstract

### 1.2.1 Methods

#### createFragment

protected abstract Fragment **createFragment**()

Metodo que deben sobreescribir todas las clases que heredan de SingleFragmentActivity, para definir el Fragment que se utilizará.

**Returns** nuevo Fragment que se utilizará para manejar la interfaz de usuario de la activity.

#### onCreate

protected void **onCreate**(Bundle *savedInstanceState*)

Inicializa la activity y crea un Fragment mediante el método abstracto createFragment(). Luego configura dicho Fragment para ser utilizado como la interfaz de usuario de esta Activity.

**Parameters**

- **savedInstanceState** –

## 1.3 LogInActivity

public class **LogInActivity** extends *SingleFragmentActivity*

Actividad en la que inicia la aplicación, en la cual el usuario puede loguearse para ingresar a la aplicación, o crearse una nueva cuenta. Crearse una nueva cuenta iniciará una nueva *SignUpActivity*.

### 1.3.1 Methods

#### createFragment

protected Fragment **createFragment**()

Crea el Fragment que se utilizará en la Activity.

**Returns** *LogInFragment* en forma de Fragment

## 1.4 SignUpActivity

public class **SignUpActivity** extends *SingleFragmentActivity*
Actividad con la interfaz para crear una nueva cuenta en la aplicación

### 1.4.1 Methods

#### createFragment

protected Fragment **createFragment** ( )
Crea el Fragment que se utilizará en la Activity.

**Returns** *SignUpFragment* en forma de Fragment

## 1.5 AddContactActivity

public class **AddContactActivity** extends *SingleFragmentActivity*
Actividad con la interfaz para agregar un nuevo contacto

### 1.5.1 Methods

#### createFragment

protected Fragment **createFragment** ( )
Crea el Fragment que se utilizará en la Activity.

**Returns** *AddContactFragment* en forma de Fragment

## 1.6 SkillsActivity

public class **SkillsActivity** extends *SingleFragmentActivity*
Actividad con la interfaz para mostrar y editar las destrezas

### 1.6.1 Methods

#### createFragment

protected Fragment **createFragment** ( )
Crea el Fragment que se utilizará en la Activity.

**Returns** *SkillsFragment* en forma de Fragment

## 1.7 AddSkillActivity

public class **AddSkillActivity** extends *SingleFragmentActivity*
Actividad con la interfaz para agregar una nueva destreza

### 1.7.1 Methods

#### createFragment

protected Fragment **createFragment**()
Crea el Fragment que se utilizará en la Activity.

**Returns** *AddSkillFragment* en forma de Fragment

## 1.8 JobsActivity

public class **JobsActivity** extends *SingleFragmentActivity*
Actividad con la interfaz para mostrar y editar las experiencias laborales

### 1.8.1 Methods

#### createFragment

protected Fragment **createFragment**()
Crea el Fragment que se utilizará en la Activity.

**Returns** *JobsFragment* en forma de Fragment

## 1.9 AddJobActivity

public class **AddJobActivity** extends *SingleFragmentActivity*
Actividad con la interfaz para agregar una nueva experiencia laboral

### 1.9.1 Methods

#### createFragment

protected Fragment **createFragment**()
Crea el Fragment que se utilizará en la Activity.

**Returns** *AddJobFragment* en forma de Fragment

CHAPTER

TWO

# FRAGMENTS

## 2.1 LogInFragment

public class **LogInFragment** extends Fragment implements LoaderManager.LoaderCallbacks<Cursor>
Establece la interfaz de usuario para loguearse a la aplicación o crearse cuenta de no poseer una.

### 2.1.1 Fields

#### mAuthTask

private UserLoginTask **mAuthTask**
Task asincrónica para intentar loguearse al servidor sin detener la UI thread.

#### mEmailAutocompleteText

private AutoCompleteTextView **mEmailAutocompleteText**
Referencia al campo EditText que contiene la forma para ingresar el email.

#### mPasswordEditText

private EditText **mPasswordEditText**
Referencia al campo EditText que contiene la forma para ingresar el password

#### mSignInButton

private Button **mSignInButton**
Referencia al botón de Sign Up utilizado para iniciar una *SignUpActivity*.

#### mSignUpButton

private Button **mSignUpButton**
Referencia al botón de Log In utilizado para intentar acceder a la aplicación.

### 2.1.2 Methods

#### onCreateView

#### joinApplication

private void **joinApplication**()
Inicia la Activity: *MainScreenActivity*.

#### onResume

public void **onResume**()
Invoca a *resetFields* para limpiar los campos de los formularios.

#### resetFields

private void **resetFields**()
Limpia los campos de los formularios.

#### OpenSignUpActivity

private void **OpenSignUpActivity**()
Inicia la Activity: *SignUpActivity*.

#### populateAutoComplete

private void **populateAutoComplete**()
Sugiere emails conocidos para ingresar en el campo *mEmailAutocompleteText*.

#### attemptLogin

private void **attemptLogin**()
Se fija que todos los campos ingresados tengan información válida. De ser así, inicia una tarea asincrónica para intentar conectarse al servidor con el email y password ingresados.

#### showProgress

private void **showProgress**(boolean *show*)
Si el parámetro ingresado es true, esconde los elementos de UI de logueo y muestra la animación de cargado. Si es true, esconde la animación de cargado y regresa la aplicación a su estado ordinario.

**Parameters**

- **show** – Determina si se muestra la animación de cargando, o si se esconde.

## 2.2 SignUpFragment

public class **SignUpFragment** extends Fragment implements LoaderManager.LoaderCallbacks<Cursor>
Establece la interfaz de usuario para crearse una nueva cuenta en la aplicación.

### 2.2.1 Fields

#### mSignUpTask

private UserSignUpTask **mSignUpTask**
Task asincrónica para conectarse al servidor y crearse una nueva cuenta sin detener la UI thread.

#### mEmailAutocompleteText

private AutoCompleteTextView **mEmailAutocompleteText**
Referencia al campo EditText que contiene la forma para ingresar el email.

#### mPasswordEditText

private EditText **mPasswordEditText**
Referencia al campo EditText que contiene la forma para ingresar el password

#### mFirstNameEditText

private EditText **mFirstNameEditText**
Referencia al campo EditText que contiene la forma para ingresar el nombre del usuario.

#### mLastNameEditText

private EditText **mLastNameEditText**
Referencia al campo EditText que contiene la forma para ingresar el apellido del usuario.

#### mSignUpButton

private Button **mSignUpButton**
Referencia al botón de Sign up, utilizado para confirmar los campos ingresados y crearse una nueva cuenta.

### 2.2.2 Methods

#### onCreateView

#### attemptSignUp

private void **attemptSignUp**()
Se fija que todos los campos ingresados tengan información válida. De ser así, inicia una tarea asincrónica para intentar conectarse al servidor y crear la cuenta. Si la cuenta se crea con éxito, la aplicación regresa a la Activity: *LogInActivity*.

#### onResume

public void **onResume**()
Invoca a *resetFields* para limpiar los campos de los formularios.

#### resetFields

private void **resetFields**()
Limpia los campos de los formularios.

#### OpenSignUpActivity

private void **OpenSignUpActivity**()
Inicia la Activity: *SignUpActivity*.

#### populateAutoComplete

private void **populateAutoComplete**()
Sugiere emails conocidos para ingresar en el campo *mEmailAutocompleteText*.

#### showProgress

private void **showProgress**(boolean *show*)
Si el parámetro ingresado es true, esconde los elementos de UI de logueo y muestra la animación de cargado. Si es true, esconde la animación de cargado y regresa la aplicación a su estado ordinario.

**Parameters**

- **show** – Determina si se muestra la animación de cargando, o si se esconde.

## 2.3 ProfileFragment

public class **ProfileFragment** extends Fragment
Provee la interfaz para visualizar el perfil del usuario.

### 2.3.1 Methods

#### onCreateView

#### attemptSignUp

private void **attemptSignUp**()
Se fija que todos los campos ingresados tengan información válida. De ser así, inicia una tarea asincrónica para intentar conectarse al servidor y crear la cuenta. Si la cuenta se crea con éxito, la aplicación regresa a la Activity: *LogInActivity*.

#### onResume

public void **onResume**()
Actualiza los campos del perfil a través del método *updateFields*.

#### updateFields

private void **updateFields**()
Actualiza los valores de los componentes de la vista, utilizando los datos contenidos en el Singleto *InformationHolder*.

#### chooseProfilePicture

private void **chooseProfilePicture**()
Inicia un Intent para seleccionar una foto del dispositivo con cualquier aplicación capaz.

#### onActivityResult

public void **onActivityResult**(int *requestCode*, int *resultCode*, Intent *data*)
Si el resultado obtenido corresponde al de haber seleccionado una imagen satisfactoriamente. Procede a agregar la imagen como foto de perfil.

#### setUpResumeField

private void **setUpResumeField**(View *v*)
Prepara el campo del Resumen para que pueda ser editable como corresponde.

#### setUpNameField

private void **setUpNameField**(View *v*)
Prepara el campo del Nombre para que pueda ser editable como corresponde.

#### setUpSaveChangesButton

private void **setUpSaveChangesButton**(View *v*)
Inicializa el botón de guardar los cambios y le agrega los listeners correspondientes.

## 2.4 SkillsFragment

public class **SkillsFragment** extends Fragment
Utiliza un RecyclerView para enlistar en forma eficiente las destrezas, permitiendo además, la edición de las mismas.

### 2.4.1 Fields

#### mSkillsRecycleView

private RecyclerView **mSkillsRecycleView**
Referencia al RecyclerView que maneja la vista de las destrzas

#### mSkillsAdapter

private *SkillsAdapter* **mSkillsAdapter**
Adapter utilizado para manejar la vista de las destrzas en el RecyclerView. Ver *SkillsAdapter*

#### mAddSkillButton

private Button **mAddSkillButton**
Referencia al botón utilizado para agregar destrezas.

### 2.4.2 Methods

#### updateUI

private void **updateUI**()
Actualiza la vista de las destrezas. Para eso pide la información a *InformationHandler*, luego crea un *SkillsAdapter* y lo asigna al RecyclerView.

#### onCreateView

public View **onCreateView**(LayoutInflater *inflater*, ViewGroup *container*, Bundle *savedInstanceState*)
Infla el Fragment con su layout correspondiente e inicializa las referencias y componentes.

#### onResume

public void **onResume**()
Al resumir el Fragment se actualiza la vista de las destrezas invocando a UpdateUI().

## 2.5 AddSkillFragment

public class **AddSkillFragment** extends Fragment
Otorga al usuario una interfaz para agregar un skill a su perfil.

### 2.5.1 Fields

#### mTitleEditText

#### mCategoryEditText

private EditText **mCategoryEditText**
Referencia al EditText correspondiente a la forma para ingresar la categoría de la destreza a agregar.

#### mDescriptionEditText

private EditText **mDescriptionEditText**
Referencia al EditText correspondiente a la forma para ingresar la descripción de la destreza a agregar.

#### mAddSkillButton

private Button **mAddSkillButton**
Referencia al botón que confirma los campos ingresados e intenta agregat la nueva destreza al usuario.

### 2.5.2 Methods

#### addSkill

private void **addSkill**()
Recolecta los datos ingresados en las formas mostradas e inicia una tarea asincrónica para conectarse al servidor y agregar la nueva destreza.

#### onCreateView

public View **onCreateView**(LayoutInflater *inflater*, ViewGroup *container*, Bundle *savedInstanceState*)
Infla el Fragment con su layout correspondiente e inicializa las referencias y componentes.

## 2.6 JobsFragment

public class **JobsFragment** extends Fragment
Utiliza un RecyclerView para enlistar en forma eficiente las experiencias laborales, permitiendo además, la edición de las mismas.

### 2.6.1 Fields

#### mJobsRecycleView

private RecyclerView **mJobsRecycleView**
Referencia al RecyclerView que maneja la vista de las experiencias laborales

#### mJobsAdapter

private *JobsAdapter* **mJobsAdapter**
Adapter utilizado para manejar la vista de las experiencias laborales en el RecyclerView. Ver *JobsAdapter*

#### mAddJobButton

private Button **mAddJobButton**
Referencia al botón utilizado para agregar experiencias laborales

### 2.6.2 Methods

#### updateUI

private void **updateUI**()
Actualiza la vista de las experiencias laborales. Para eso pide la información a *InformationHandler*, luego crea un *JobsAdapter* y lo asigna al RecyclerView.

#### onCreateView

public View **onCreateView**(LayoutInflater *inflater*, ViewGroup *container*, Bundle *savedInstanceState*)
Infla el Fragment con su layout correspondiente e inicializa las referencias y componentes.

#### onResume

public void **onResume**()
Al resumir el Fragment se actualiza la vista de experiencias laborales.

## 2.7 AddJobFragment

public class **AddJobFragment** extends Fragment
Otorga al usuario una interfaz para agregar una experiencia laboral a su perfil.

### 2.7.1 Fields

#### mTitleEditText

#### mCategoryEditText

private EditText **mCategoryEditText**
Referencia al EditText correspondiente a la forma para ingresar la categoría de la experiencia laboral a agregar.

#### mDescriptionEditText

private EditText **mDescriptionEditText**
Referencia al EditText correspondiente a la forma para ingresar la descripción de la experiencia laboral a agregar.

#### mAddJobButton

private Button **mAddJobButton**
Referencia al botón que confirma los campos ingresados e intenta agregat la nueva experiencia laboral al usuario.

### 2.7.2 Methods

#### addJob

private void **addSkill**()
Recolecta los datos ingresados en las formas mostradas e inicia una tarea asincrónica para conectarse al servidor y agregar la nueva experiencia laboral.

#### onCreateView

public View **onCreateView**(LayoutInflater *inflater*, ViewGroup *container*, Bundle *savedInstanceState*)
Infla el Fragment con su layout correspondiente e inicializa las referencias y componentes.

## 2.8 ContactsFragment

public class **ContactsFragment** extends Fragment
Utiliza un RecyclerView para enlistar en forma eficiente los contactos del usuario, permitiendo además, la edición de los mismos.

### 2.8.1 Fields

#### mContactsRecycleView

private RecyclerView **mContactsRecycleView**
Referencia al RecyclerView que maneja la vista de los contactos

#### mContactsAdapter

private *ContactsAdapter* **mContactsAdapter**
Adapter utilizado para manejar la vista de los contactos en el RecyclerView. Ver *ContactsAdapter*

#### mSearchButton

private Button **mSearchButton**
Referencia al botón utilizado para buscar contactos.

### 2.8.2 Methods

#### updateUI

private void **startSearchContactActivity**()
Inicia la Activity: *AddContactActivity*.

#### updateUI

private void **updateUI**()
Actualiza la vista de los contactos en el RecyclerView. Para eso pide la información a *InformationHandler*, luego crea un *ContactsAdapter* y lo asigna al RecyclerView.

#### onCreateView

public View **onCreateView**(LayoutInflater *inflater*, ViewGroup *container*, Bundle *savedInstanceState*)
Infla el Fragment con su layout correspondiente e inicializa las referencias y componentes.

#### onResume

public void **onResume**()
Al resumir el Fragment se actualiza la vista de los contactos invocando a UpdateUI().

## 2.9 AddContactFragment

public class **AddContactFragment** extends Fragment
Otorga al usuario una interfaz para agregar una experiencia laboral a su perfil.

### 2.9.1 Fields

#### mUsernameEditText

private EditText **mUsernameEditText**
Referencia al EditText correspondiente a la forma para buscar un contacto.

#### mAddContactButton

private Button **mAddContactButton**
Referencia al botón que envía una solicitud de amistad al contacto seleccionado, para esperar ser aceptado.

### 2.9.2 Methods

#### addContact

private void **addContact**()
Inicia una tarea asincrónica para conectarse al servidor y agregar al contacto seleccionado.

#### isUsernameValid

private boolean **isUsernameValid**(String *username*)
Devuelve true si el usuario ingresado por parámetro existe en la base de datos del servidor. False en caso contrario.

**Parameters**

- **username** – nombre de usuario del contacto a agregar

#### onCreateView

public View **onCreateView** (LayoutInflater *inflater*, ViewGroup *container*, Bundle *savedInstanceState*)
Infla el Fragment con su layout correspondiente e inicializa las referencias y componentes.

## 2.10 NotificationsFragment

public class **NotificationsFragment** extends Fragment
Utiliza un RecyclerView para enlistar en forma eficiente las destrezas, permitiendo además, la edición de las mismas.

### 2.10.1 Fields

#### mNotificationsRecycleView

private RecyclerView **mNotificationsRecycleView**
Referencia al RecyclerView que maneja la vista de las notificaciones.

#### mNotificationsAdapter

private NotificationsAdapter **mNotificationsAdapter**
Adapter utilizado para manejar la vista de las notificaciones en el RecyclerView. Ver *NotificationsAdapter*.

#### mAddingFriend

private boolean **mAddingFriend** = false
Define si se verán o no los títulos de los tabs. Por defecto, los títulos no se verán.

### 2.10.2 Methods

#### createAddFriendDialog

private void **createAddFriendDialog** (String *friendUsername*, int *notificationIndex*)
Crea un Dialog en respuesta a una notificación de solicitud de amistad, en el cual se puede aceptar o rechazar al contacto solicitante.

**Parameters**

- **friendUsername** – Nombre de usuario del usuario que envío la solicitud.
- **notificationIndex** – índice correspondiente a la lista de notificaciones.

#### updateUI

private void **updateUI** ()
Actualiza la vista de las notificaciones. Para eso pide la información a *InformationHandler*, luego crea un *NotificationsAdapter* y lo asigna al RecyclerView.

### onCreate

public View **onCreateView**(LayoutInflater *inflater*, ViewGroup *container*, Bundle *savedInstanceState*)
Infla el Fragment con su layout correspondiente e inicializa las referencias y componentes.

### onResume

public void **onResume**()
Al resumir el Fragment se actualiza la vista de las notificaciones invocando a UpdateUI().

CHAPTER

# THREE

# SINGLETONS

## 3.1 ServerHandler

public class **`ServerHandler`**

Permite realizar operaciones HTTP a través de una API Rest, utilizando un token de authenticación. Es importante mencionar, que estas operaciones NO deben realizarse en la UI thread de Android, sino que deben ser desplazadas a otra thread. En esta aplicación, se utilizan AsyncTasks para ejecutar las operaciones del ServerHandler en la Background thread.

### 3.1.1 Fields

#### mServerIP

private String **`mServerIP`**

IP del server con puerto incluido, el cual utilizarña para conectarse al servidor. Ejemplo: 192.168.0.19:8000

#### mUsername

private String **`mUsername`**

Nombre de usuario con el cual se ha establecido la conexión

#### mConnectionToken

private String **`mConnectionToken`**

Token provista para el usuario actual.

### 3.1.2 Methods

#### get

public static *ServerHandler* **`get`**(Context *context*)

Devuelve la instancia actual del Singleton ServerHandler.

### setConnectionToken

public void **setConnectionToken** (String *connectionToken*)
Permite utilizar el token provisto en los headers de los mensajes HTTP realizados.

**Parameters**

- **connectionToken** – token de authenticación provisto

### GET

public String **GET** (String *urlSpec*)
Realiza un GET request a la dirección especificada por el parámetro urlSpec.

**Parameters**

- **urlSpec** – URL a en la que se establecerá la conexión

**Returns** Respuesta del server

### POST

public String **POST** (String *urlSpec*, String *parameters*)
Realiza un POST request a la dirección especificada por el parámetro urlSpec.

**Parameters**

- **urlSpec** – URL a en la que se establecerá la conexión
- **parameters** – parametros del POST en formato json

**Returns** Respuesta del server

### PUT

public String **PUT** (String *urlSpec*, String *parameters*)
Realiza un PUT request a la dirección especificada por el parámetro urlSpec.

**Parameters**

- **urlSpec** – URL a en la que se establecerá la conexión
- **parameters** – parametros del PUT en formato json

**Returns** Respuesta del server

### DELETE

public String **DELETE** (String *urlSpec*, String *parameters*)
Realiza un DELETE request a la dirección especificada por el parámetro urlSpec.

**Parameters**

- **urlSpec** – URL a en la que se establecerá la conexión
- **parameters** – parametros del DELETE en formato json

**Returns** Respuesta del server

## 3.2 InformationHolder

public class **InformationHolder**

Permite realizar operaciones HTTP a través de una API Rest, utilizando un token de authenticación. Es importante mencionar, que estas operaciones NO deben realizarse en la UI thread de Android, sino que deben ser desplazadas a otra thread. En esta aplicación, se utilizan AsyncTasks para ejecutar las operaciones del InformationHolder en la Background thread.

### 3.2.1 Fields

#### mName

private String **mName**

Nombre del usuario actualmente conectado.

#### mResume

private String **mResume**

Resumen del usuario actual

#### mMail

private String **mMail**

Email del usuario actual

#### mProfilePicture

private Bitmap **mProfilePicture**

Bitmap correspondiente a la foto de perfil del usuario actual.

#### mContacts

private List<*Contact*> **mContacts**

Lista de contactos del usuario actual

#### mJobs

private List<*Job*> **mJobs**

Lista de experiencias laborales del usuario actual

#### mSkills

private List<*Skill*> **mSkills**

Lista de destrezas del usuario actual

#### mNotifications

private List<*Notification*> **mNotifications**
Lista de notificaciones del usuario actual

### 3.2.2 Methods

Getters y Setters de los atributos anteriores.

#### get

public static *InformationHolder* **get**()
Devuelve la instancia actual del Singleton InformationHolder.

CHAPTER

FOUR

# ADAPTERS

## 4.1 ViewPagerAdapter

public class **ViewPagerAdapter** extends FragmentPagerAdapter
Adaptador definido internamente por *MainScreenActivity* para controlar los tabs.

### 4.1.1 Fields

#### mFragmentList

private final List<Fragment> **mFragmentList**
Lista constante de Fragments que serán manipulados

#### mFragmentTitleList

private final List<String> **mFragmentTitleList**
Lista constante con los títulos de los Fragments.

#### mShowTittle

private boolean **mShowTittle**
Determina si el usuario verá el título de cada tab o si solo verá los íconos.

### 4.1.2 Constructor

#### ViewPagerAdapter

public **ViewPagerAdapter**(FragmentManager *manager*)
Contructor, inicializa el Adapter y setea por defecto que no se vean los nombre de los tabs.

**Parameters**

- **manager** –

### 4.1.3 Methods

#### addFragment

public void **addFragment** (Fragment *fragment*, String *title*)

Agrega un Fragment con el título pasado como segundo parámetro, para ser manipulado por el ViewPagerAdapter.

**Parameters**

- **fragment** – Fragment a manipular
- **tittle** – tituló asociado al Fragment

#### getItem

public Fragment **getItem** (int *position*)

Devuelve el Fragment en la posicion pasada como parámetro

**Parameters**

- **position** – índice del Fragment en la lista.

#### getCount

public int **getCount** ()

Devuelve la cantidad de Fragments que están siendo manipulados por la ViewPagerAdapter

#### getPageTitle

public CharSequence **getPageTitle** (int *position*)

Devuelve el título del Fragment correspondiente al índice pasado por parámetro

**Parameters**

- **position** – índice del Fragment en la lista.

## 4.2 SkillsAdapter

private class **SkillsAdapter** extends RecyclerView.Adapter<SkillsViewHolder>

Adaptador definido internamente por *SkillsFragment* para controlar cada destreza en forma de un *SkillsViewHolder*

### 4.2.1 Fields

#### mSkills

private List<*Skill*> **mSkills**

Lista de *Skill*, que contienen la información acerca de las destrezas del usuario.

### 4.2.2 Constructor

#### SkillsAdapter

public **SkillsAdapter**(List<*Skill*> *skills*)
Inicializa la lista de *Skill* a manipular.

**Parameters**

- **skills** –

### 4.2.3 Methods

#### onCreateViewHolder

public NotificationsViewHolder **onCreateViewHolder**(ViewGroup *parent*, int *viewType*)
Crea el View para mostrar un *Skill* de la forma deseada y crea con el mismo, un 'SkillsViewHolder'

**Parameters**

- **parent** –
- **viewType** –

#### getItemCount

public int **getItemCount**()
Devuelve la cantidad de Skill's que se están manipulando

#### onBindViewHolder

public void **onBindViewHolder**(SkillsViewHolder *holder*, int *position*)
Rellena los valores del SkillsViewHolder para que pueda ser mostrado de la forma deseada.

**Parameters**

- **holder** – SkillsViewHolder que se está agregando
- **position** – índice en la lista de Skills, del *Skill* que está siendo agregado.

## 4.3 JobsAdapter

private class **JobsAdapter** extends RecyclerView.Adapter<JobsViewHolder>
Adaptador definido internamente por *JobsFragment* para controlar cada experiencia laboral en forma de un *JobsViewHolder*

### 4.3.1 Fields

#### mJobs

private List<*Job*> **mJobs**
Lista de *Job*, que contienen la información acerca de las experiencias laborales.

### 4.3.2 Constructor

#### JobsAdapter

public **JobsAdapter** (List<*Job*> *jobs*)
Inicializa la lista de *Job* a manipular.

**Parameters**

- **jobs** –

### 4.3.3 Methods

#### onCreateViewHolder

public JobsViewHolder **onCreateViewHolder** (ViewGroup *parent*, int *viewType*)
Crea el View para mostrar un Job de la forma deseada y crea con el mismo un 'JobsViewHolder'

**Parameters**

- **parent** –
- **viewType** –

#### getItemCount

public int **getItemCount** ()
Devuelve la cantidad de Job's que se están manipulando

#### onBindViewHolder

public void **onBindViewHolder** (JobsViewHolder *holder*, int *position*)
Rellena los valores del JobsViewHolder para que pueda ser mostrado de la forma deseada.

**Parameters**

- **holder** – JobsViewHolder que se está agregando
- **position** – índice en la lista de Jobs, del *Job* que está siendo agregado

## 4.4 ContactsAdapter

private class **ContactsAdapter** extends RecyclerView.Adapter<ContactViewHolder>
Adaptador definido internamente por *ContactsFragment* para controlar la vista de los contactos en forma de un *ContactsViewHolder*

### 4.4.1 Fields

#### mContacts

private List<*Contact*> **mContacts**
Lista de *Contact*, que contienen la información acerca de los contactos del usuario.

### 4.4.2 Constructor

#### ContactsAdapter

public **ContactsAdapter**(List<*Contact*> *contacts*)
Inicializa la lista de *Contact* a manipular.

**Parameters**

- **contacts** –

### 4.4.3 Methods

#### onCreateViewHolder

public ContactsViewHolder **onCreateViewHolder**(ViewGroup *parent*, int *viewType*)
Crea el View para mostrar un Contact de la forma deseada y crea con el mismo un 'ContactsViewHolder'

**Parameters**

- **parent** –
- **viewType** –

#### getItemCount

public int **getItemCount**()
Devuelve la cantidad de Contact's que se están manipulando

#### onBindViewHolder

public void **onBindViewHolder**(ContactsViewHolder *holder*, int *position*)
Rellena los valores del ContactsViewHolder para que pueda ser mostrado de la forma deseada.

**Parameters**

- **holder** – ContactsViewHolder que se está agregando
- **position** – índice en la lista de Contacts, del *Contact* que está siendo agregado

## 4.5 NotificationAdapter

### 4.5.1 Fields

#### mNotifications

private List<*Notification*> **mNotifications**
Lista de *Notification*, que contienen la información acerca de las notificaciones del usuario.

### 4.5.2 Constructor

#### NotificationAdapter

public **NotificationAdapter** (List<*Notification*> *notifications*)
Inicializa la lista de *Notification* a manipular.

**Parameters**

- **notifications** –

### 4.5.3 Methods

#### onCreateViewHolder

public NotificationsViewHolder **onCreateViewHolder** (ViewGroup *parent*, int *viewType*)
Crea el View para mostrar un *Notification* de la forma deseada y crea con el mismo, un 'NotificationsViewHolder'

**Parameters**

- **parent** –
- **viewType** –

#### getItemCount

public int **getItemCount** ()
Devuelve la cantidad de Notification's que se están manipulando

#### onBindViewHolder

public void **onBindViewHolder** (NotificationsViewHolder *holder*, int *position*)
Rellena los valores del NotificationsViewHolder para que pueda ser mostrado de la forma deseada.

**Parameters**

- **holder** – NotificationsViewHolder que se está agregando
- **position** – índice en la lista de Notifications, del *Notification* que está siendo agregado.

CHAPTER

FIVE

# MODEL

## 5.1 User

public class **User**
Clase que contiene información del usuario

### 5.1.1 Constructor

#### User

public **User** (String *firstName*, String *lastName*, String *email*, String *password*)
Constructor. Inicializa los parametros ingresados.

**Parameters**

- **firstName** – primer nombre del usuario.
- **lastName** – apellido del usuario.
- **email** – email del usuario.
- **password** – password del usuario.

### 5.1.2 Methods

Getters y Setters de los atributos.

## 5.2 Contact

public class **Contact**
Clase que contiene información acerca de un contacto.

### 5.2.1 Contact

#### Notification

public **Contact** (String *name*, String *email*, Bitmap *pictureBitmap*)
Constructor. Inicializa los parametros ingresados.

**Parameters**

- **name** – nombre completo del contacto. Incluye apellido
- **email** – email del contacto
- **pictureBitmap** – foto de perfil del contacto

### 5.2.2 Methods

Getters y Setters de los atributos.

#### addSkill

public void **addSkill** (*Skill skillToAdd*)
Agrega una destreza ignorando duplicados.

**Parameters**

- **skillToAdd** – destreza a añadir.

#### addJob

public void **addJob** (*Job jobToAdd*)
Agrega una experiencia laboral ignorando duplicados.

**Parameters**

- **jobToAdd** – experiencia laboral a añadir.

## 5.3 Skill

public class **Skill**
Clase que representa una destreza del usuario

### 5.3.1 Constructor

#### Skill

public **Skill** (String *tittle*, String *category*, String *description*)
Constructor. Inicializa los parametros ingresados.

**Parameters**

- **tittle** – título de la destreza.
- **category** – categoría de la destreza.
- **description** – datos adicionales de la destreza.

### 5.3.2 Methods

Getters y Setters de los atributos.

## 5.4 Job

public class **Job**
Clase que representa una experiencia laboral del usuario.

### 5.4.1 Constructor

#### Job

public **Skill** (String *tittle*, String *category*, String *description*)
Constructor. Inicializa los parametros ingresados.

**Parameters**

- **tittle** – título de la experiencia laboral.
- **category** – categoría de la experiencia laboral.
- **description** – datos adicionales de la experiencia laboral.

### 5.4.2 Methods

Getters y Setters de los atributos.

## 5.5 Notification

public class **Notification**
Clase que contiene información acerca de una notificación.

### 5.5.1 Constructor

#### Notification

public **Notification** (String *content*, int *code*)
Constructor. Inicializa los parametros ingresados.

**Parameters**

- **content** – contenido de la notificación.
- **code** – código interno que identifica de que tipo de notificación se trata.

### 5.5.2 Methods

Getters y Setters de los atributos.

### generateTitle

private String **generateTitle**(int *code*)
Genera el título dependiendo dl código de la notificación.

**Parameters**

- **code** – código que identifica el tipo de notificación

CHAPTER

SIX

# UTILS

## 6.1 FieldValidator

public class **FieldValidator**
Clase que permite verificar si un campo es válido o no

### 6.1.1 Methods

#### isNameValid

public boolean **isNameValid**(String *name*)
Devuelve true si la cadena ingresada contiene solo letras del alfabeto. Devuelve false en otro caso.

**Parameters**

- **name** – cadena a verificar

#### isEmailValid

public boolean **isEmailValid**(String *email*)
Devuelve true si la cadena ingresada posee el formato de una dirección mail. Devuelve false en otro caso.

**Parameters**

- **email** – cadena a verificar

#### isPasswordValid

public boolean **isPasswordValid**(String *password*)
Devuelve true si la cadena ingresada posee 4 o más caracteres. Devuelve false en otro caso.

**Parameters**

- **password** – cadena a verificar

CHAPTER

SEVEN

# INDICES AND TABLES

- genindex
- modindex
- search