OStack Manager

Index

Introduction

The following document is intended to describe all features of OStack Manager which offer you an administration dashboard of Openstack for Android mobile devices.

Firstly, in the next sections we will introduce you a brief description of the App. Then we are going to show you all the functionalities that are covered within OStack Manager and the architecture of the application including its code documentation.

Finally, we will show you as getting started with OStack Maneger so that you can use it properly.

What is OStack Manager?

OStack Manger is a native android application that provides a management dashboard for Openstack, allowing a user to manage computing resources to which it has access. OStack Maneger is very similar to the web module “Horizon” but simplified and adapted to a mobile environment.

List of feature

Below is a list of all functionalities that has been implemented in OStack Manager:

- User authentication. Currently the App does not do a distinction between admin and non-admin users. All users are treated as a non-administrator user, so the additional functionalities of an admin user are not implemented in the application.
- Selecting a project in which the user has permission, also known with the nomenclature “Tenant”.
- List instances.
- List images.
- List volumes.
- Project overview with its current limits and usage of the project´s resources.
- Create instances.
- Delete instances.
- Start instances.
- Stop instances.
- Create and delete volumes.

The used technologies and tools

The technologies that were used for creating the OStack Manager are:

- Android SDK for Eclipse.
- Java
- Openstack API Rest according to the official documentation described in http://docs.openstack.org/api/openstack-block-storage/2.0/content/.
- Achartengine Library. It is an open source library for android that provides us different tools for painting statistical graphics.

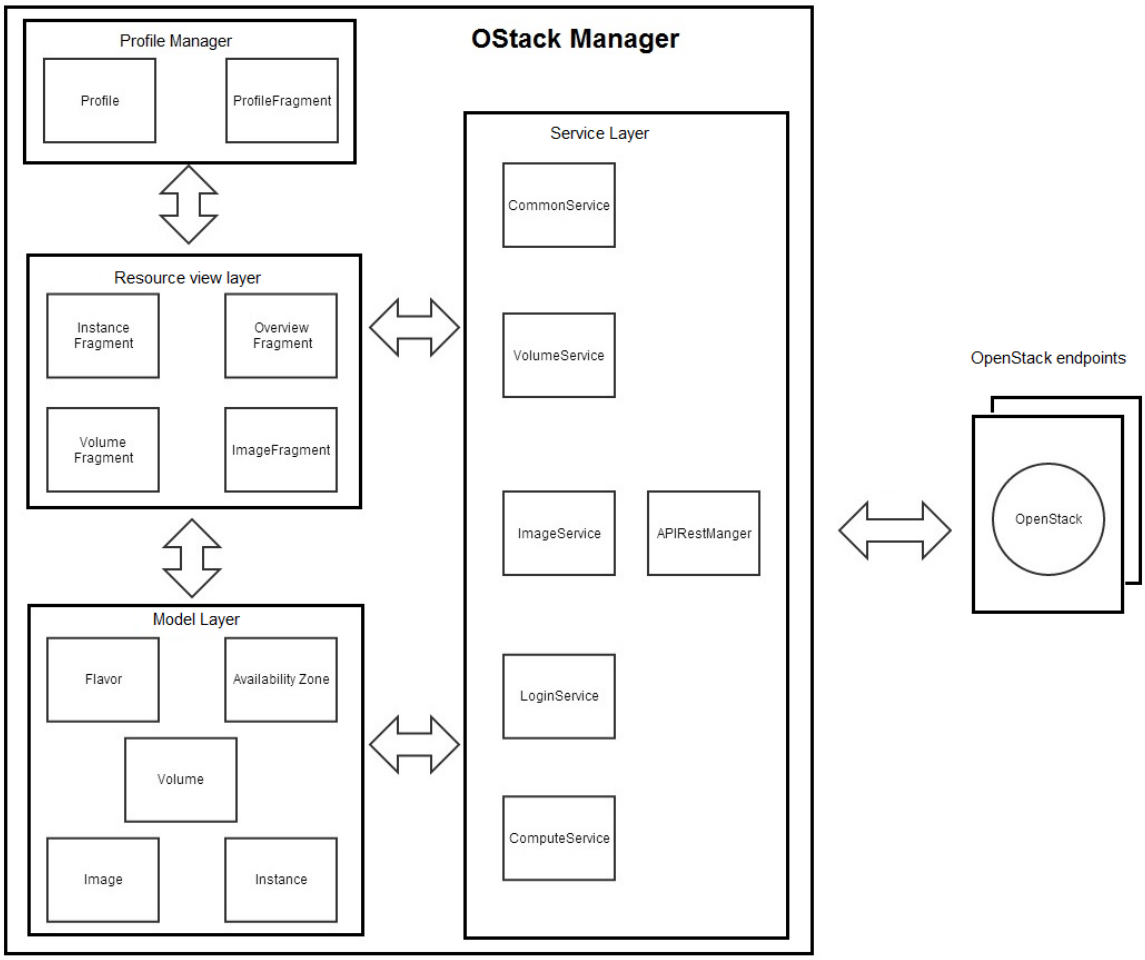
Application global architecture

In the following section, we will explain you the high level architecture of OStack Manager.

The global architecture is comprised by four main modules:

- Profile manager
- Resource view Layer
- Model Layer
- Service Layer

Below is a global architecture diagram with the modules mentioned above.



The resource view layer is the View Layer and it allows to display the data of Open Stack's resources to users. This layer also provides all UI (User Interface) functionalities.

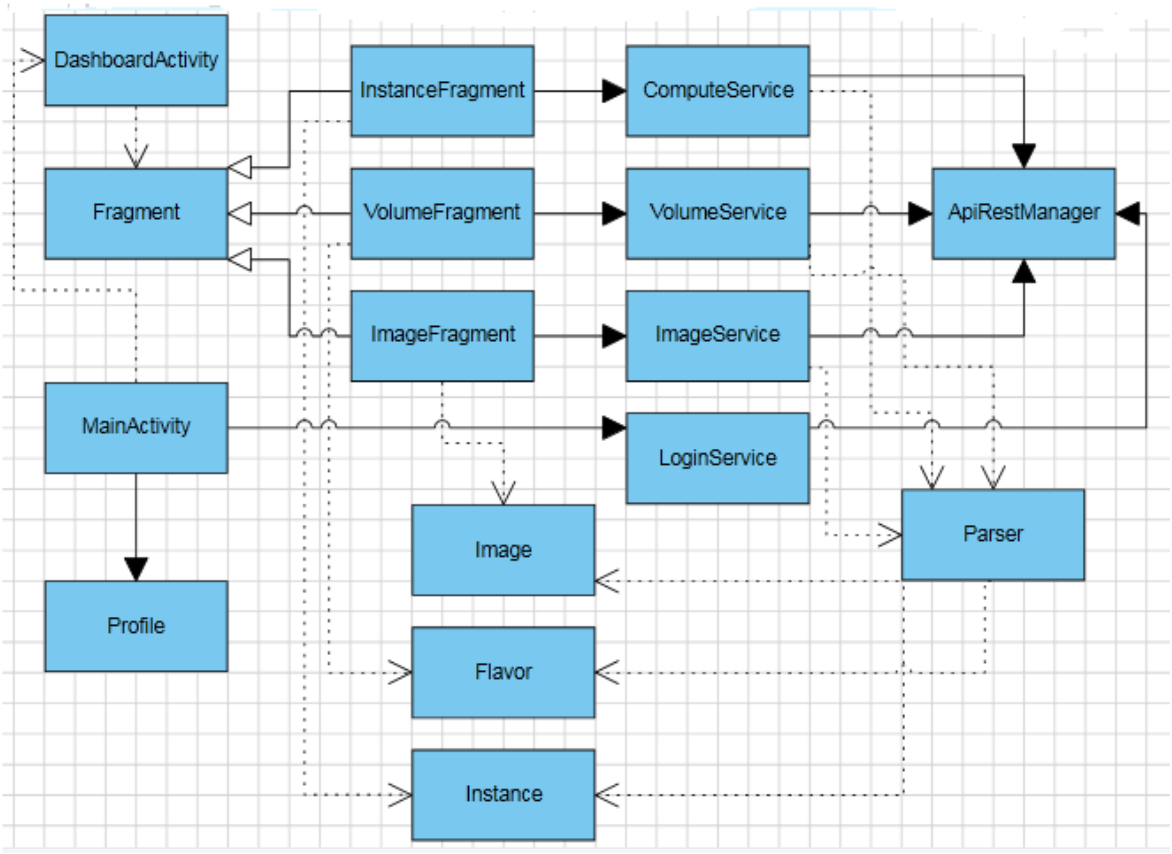
The model layer provides us a representation of the states for every resource at given moment.

The service layer is responsible for the communication with the endpoint where the OpenStack has been installed. This communication is accomplished through the API Rest provided by Openstack.

Finally, there is a module that manages the connection profile of the users. A connection profile allows to a user to connect with an endpoint of Openstack.

Class Diagram

Here we provider a class diagram with the classes more important of OStack Maneger.



MainActivity: It managers the user connection profile, allows to create, edit and remove connection profiles. This class is also responsible for performing the user login process.

DasboardActivity: provides us an activities dashboard. It contains the instance, volume and image fragments.

InstanceFragment: It manages the Openstack´s instances (Servers). This class is responsible for getting, creating, starting, stopping and removing instances.

VolumeFragemnt: It manages the Openstack´s volumes (Flavors) allowing you to list, create and remove volumes.

ImageFragment: This fragment only lists the available images that are used for creating instances.

ComputeService: It is responsible for managing the call requests to the APIs of the “nova” module. These APIs allow us to obtain all resources referred to the instances management.

VolumeService: handles the APIs that is used for the volume management.

ImageService: handles the APIs that is used for the image management.

LoginService: manages those issues related to the user login process.

ApiRestManager: it accomplishes the connection with Openstack´s APIs through Rest services using the http protocol. This class implements some basic verbs of a service Rest as get, post and delete.

Parser: It translates a “Json” object to a particular model object. The parser class is used when a response is returned by some Openstack´s API. Note that the resulted responses of any API are received through “Json” object.

Image: Represents an image model object.

Instance: Represents an instance object.

Flavor: Represents a volume or flavor object.

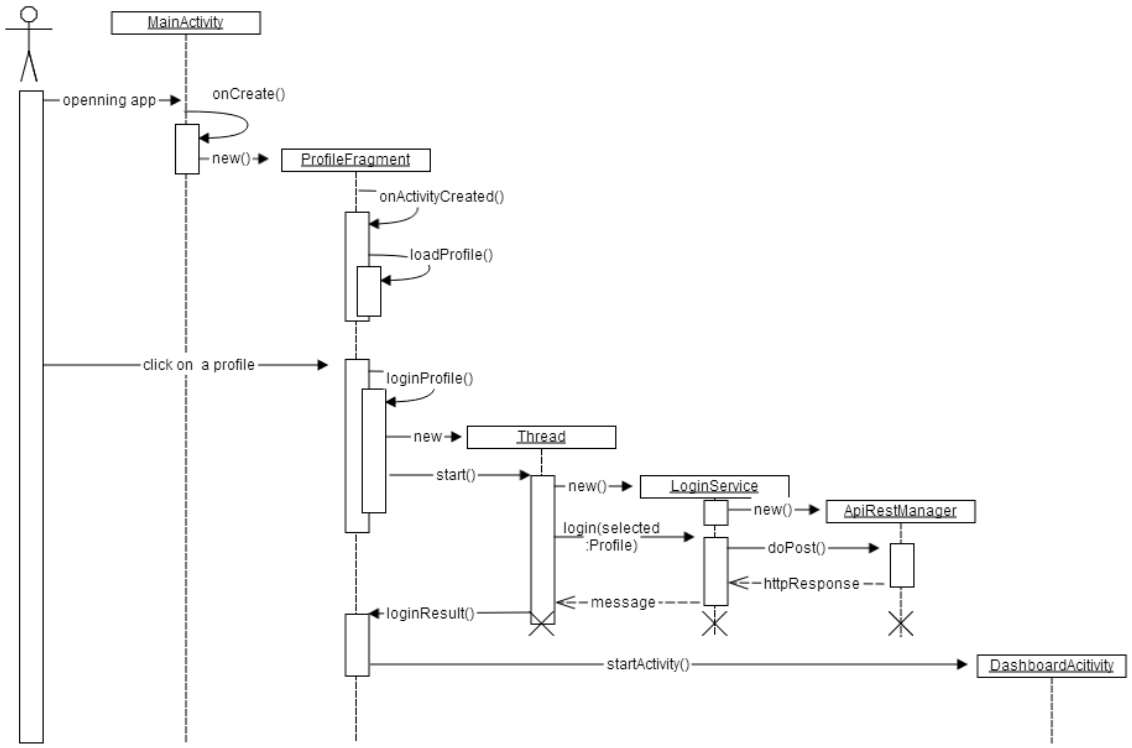
Profile: Models a user connection profile object.

Implementation

In the following pages, you will be given a description of the code used for building OStack Manager to the effect that it can be interpreted by those who want to further improvements in the future.
For a better understanding by the reader, we will detail the code used within the processes of user login, instance listing, instance creation and deletion.
Although the application provides more functionality, after reading these processes you will be able to understand the other features and also create new features, because these mechanisms are repeated for all other functions.

User login process

Let´s start to explain as the login process is accomplished and what are its steps that it has.
Below is a sequence diagram that shows the main interactions between objects and classes.



The main activity is launched when the user opens the app. It loads a fragment called "*ProfileFragment*" into the method *onCreate().*

```
@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.activity_main);

	if (savedInstanceState == null) {
		getSupportFragmentManager()
		.beginTransaction()
		.add(R.id.container, new ProfileFragment(),
		PROFILE_FRAGMENT).commit();
	}
}
```

onCreate() method of the MainActivity class.

The loaded fragment calls to its *loadProfile()* method that retrieves all connection profiles created above by the user.

```
public void loadProfile() {
	ProfileDataSource profileDataSource = new
	ProfileDataSource(getActivity());
	List<Profile> profiles = profileDataSource.getAllProfile();
	adpater = new ProfilesAdapter(profiles);
	lsV_profiles.setAdapter(adpater);
}
```

loadProfile() method of the ProfileFragment class.

The user connections profiles are obtained from an embedded database where they are stored when the user creates them.

Once the connection profiles are loaded, the user clicks on one of them and a thread is automatically started to log in into the OStackManager. The thread initializes the service "*LoginService*" and proceeds to call the *login()* method passing it as parameter the current connection profile selected.

```
/**
 * It allows to enter a openstack's session on background
 * @param selectedProfile
 */
public void loginProfile(final Profile selectedProfile) {
	final ProgressDialog dialog= ProgressDialog
		.show(mContext, null,
		mContext.getString(R.string.logging),true,false);

	Thread thread = new Thread( new Runnable() {
		@Override
		public void run() {
			LoginService loginService = new
			LoginService(mContext);
			String result =
			loginService.login(selectedProfile);
			loginResult(result,dialog);
		}
	});
	thread.start();
}
```

loginProfile() method of the ProfileFragment calss.

Then, the login service builds a call to the API of Openstack that it allows to do the user authentication process. For the connection with API mentioned above, the login service uses the REST services provided by *ApiRestManager* class. For this case, we do a post request through the *doPost()* method and it return us a http

response with a session token code and the corresponded API endpoints of the Openstack´s modules.

```
* This method allows get a token for a connection profile.
*
* @param profile
* @return String that describes the request result.
*/
public String login(Profile profile) {
	String response = "";
	ApiRestManager.APIServices.clear();
	try {
		JSONObject payload = new JSONObject();
		JSONObject auth = new JSONObject();
		JSONObject passwordCredentials = new JSONObject();
		passwordCredentials.put("username",
		profile.getUsername());
		passwordCredentials.put("password",
		profile.getPassword());

		auth.put("passwordCredentials", passwordCredentials);
		auth.put("tenantName", profile.getTenantName());

		payload.put("auth", auth);

		StringEntity entity = new
		StringEntity(payload.toString());

		ArrayList<BasicNameValuePair> headers = new
		ArrayList<>();
		headers.add(new BasicNameValuePair("Content-Type",
				"application/json"));

		HttpResponse httpResponse =
		apiRestManager.doPost(profile.getEndpoint()
				+ "/v2.0/tokens", entity, headers);

		final int statusCode =
		httpResponse.getStatusLine().getStatusCode();

		if (statusCode == 200) { // Ok
			JSONObject result = new JSONObject(response);
			setSessionData(result, profile);
			return ApiRestManager.OK;
		}
		if (statusCode == 401) { // Invaild User / password
			return mContext.getString(R.string.invalidUser);
		}
```

```
            if (statusCode == 404) { // Invalid endpoint.
                  return mContext.getString(R.string.invalidEndpoint);
            }
      } catch (Exception ex) {
            Log.e("Debug", "Exception onLogin()", ex);
      }
      // If an error has triggered (Exception, timeout, ...) return null
      return null;
}
```

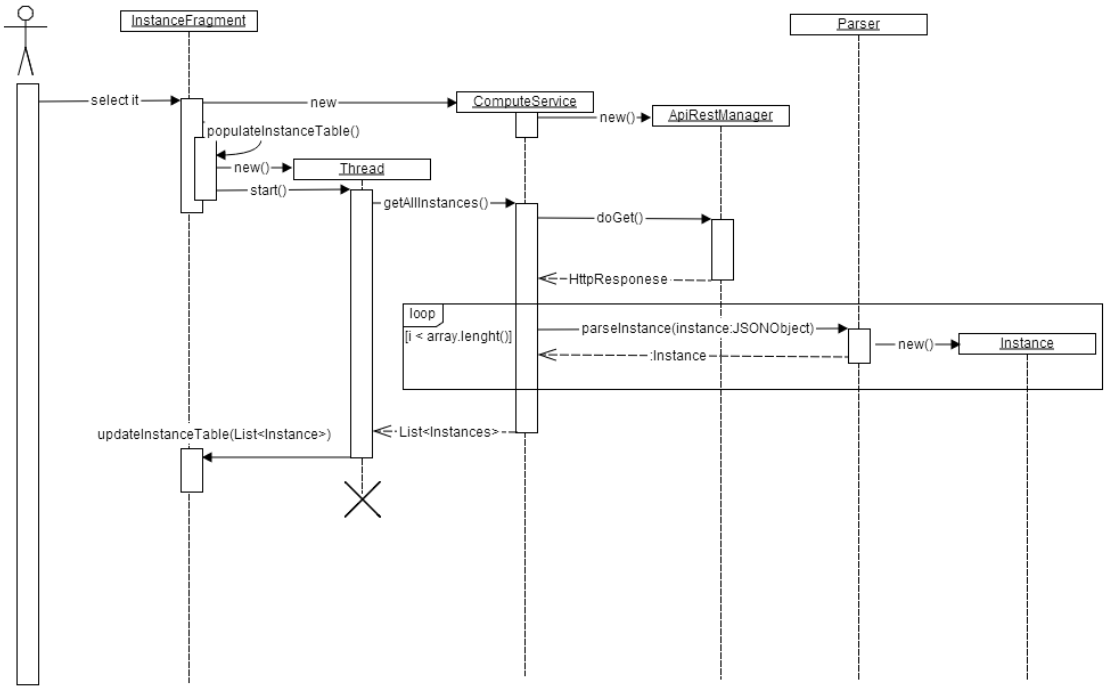
login() method of the LoginService class

Finally a message is passed to the profile fragment and if it is correct, *DahsboardActivity* is started and it is already for providing all the OStack Manager´s features to the user.

Instance listing process

In the following section, we going to explain you as the instances of Openstack are listed.

Below is a sequence diagram that shows the main interactions between objects and classes.



When the user wants to know what instances has it into Openstack, It selects the instance options on dashboard menu and "*InstanceFragment"* is launched.

The instance fragment calls to *populateInstanceTable()* method for listing all available instances. This method starts a thread that is the responsible for obtaining all instances through *ComputeService* class calling its *getAllInstances()* method.

```
/**
* Populate the Instances table
*/
private void populateInstancesTable() {
      prB_loadInstance.setVisibility(View.VISIBLE);
      new Thread(new Runnable() {
            @Override
            public void run() {
                  ArrayList<Instance> result =
                  computeService.getAllInstances();
                  updateInstanceTable(result);
            }
      }).start();
}
```

populateInstancesTable() method of the InstanceFragment class

Then, the compute service builds a call to the API of Openstack that it allows to get all instances. For the connection with API mentioned above, the login service uses the REST services provided by *ApiRestManager* class. For this case, we do a "GET" request through the method "doGet()" and it return us a http response with an JSON object that contains an array of instances.

```
public ArrayList<Instance> getAllInstances() {
      ArrayList<Instance> instances = new ArrayList<>();
      String response;

      ArrayList<BasicNameValuePair> headers = new ArrayList<>();
      headers.add(new BasicNameValuePair("X-Auth-Token",
      UserAuthDataPreferences.getInstance(mContext)
      .getTokenId()));

      try {
            HttpResponse httpResponse =
            apiRestManager.doGet(ApiRestManager
            .APIServices.get(ApiRestManager.COMPUTE_SERVICE)
            .getPublicUrl() + "/servers/detail", headers, null);

            final int statusCode = httpResponse.getStatusLine()
                  .getStatusCode();

            if (statusCode == 200) { // Ok
                  response = EntityUtils
                        .toString(httpResponse.getEntity());

                  JSONArray servers = new JSONObject(response)
                        .getJSONArray("servers");

                  for (int i = 0; i < servers.length(); i++) {
                        Instance ins = Parser
                        .parseInstance(servers.getJSONObject(i));
                        getImageAndFlavorData(ins,
                        servers.getJSONObject(i));

                        instances.add(ins);
                        return instances;
                  }
            }

      . . .
}
```

getAllInstances() method of the ComputeService class.

Once the JSON object is received, it is parsed for obtaining our own Instances object through the *Parse* class calling its *parseInstance().*

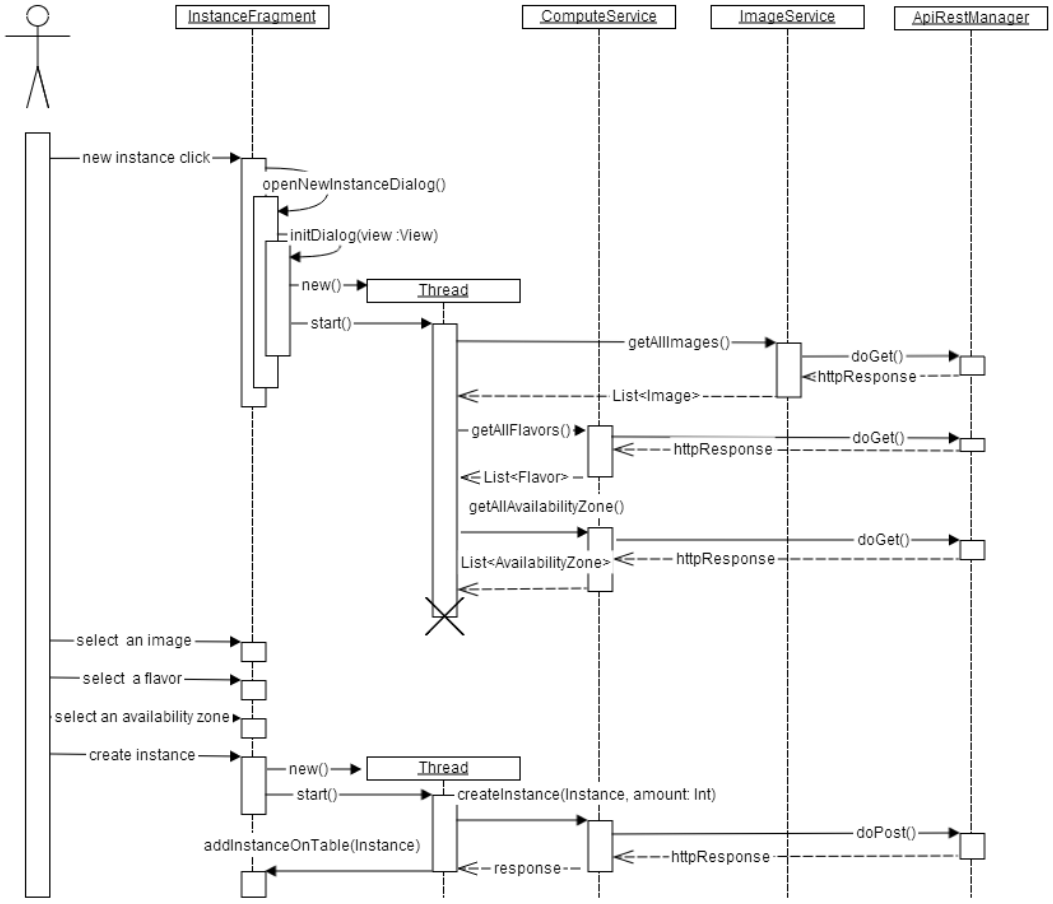
```
public static Instance parseInstance(JSONObject server) throws
JSONException {

	Instance instance = new Instance();
	instance.setId(server.getString("id"));
	instance.setName(server.getString("name"));
	instance.setStatus(server.getString("status"));
	instance.setUpdated(server.getString("updated"));
	instance.setCreated(server.getString("created"));
	instance.setAvailabilityZone(server.getString("OS-EXT-
	AZ:availability_zone"));

	//Set the security groups
	String securityGroup = "";
	JSONArray securityGroups = server.getJSONArray("security_groups");
		for (int j = 0; j < securityGroups.length(); j++) {
			securityGroup += securityGroups
				.getJSONObject(j)
				.getString("name") + ", ";

		}
	instance.setSecurityGroupName(securityGroup);

	//Set the key name
	instance.setKeyName(server.isNull("key_name") ? "" :
	server.getString("key_name"));

	//Set private addresses
	String privateAddresses = "";
	JSONObject addresses = server.getJSONObject("addresses");
	JSONArray privateAdrressesArray = !addresses.isNull("private")?
	addresses.getJSONArray("private"):null;
	if(privateAdrressesArray != null){
		for (int i = 0; i < privateAdrressesArray.length(); i++) {
			privateAddresses += privateAdrressesArray
			.getJSONObject(i)
			.getString("addr") + "; ";
		}
	}
	instance.setPrivateIp(privateAddresses);
	return instance;
}
```

parseInstance() method of the Parse class

Finally an array of instances is returned to the *InstanceFragment* for updating the user interface and showing the instance table.

Instance creation process

Other important process is the instance creation which is detailed below.



When the user clicks on new instance button, the *openNewInstanceDialog()* method is called. This method opens a dialog and gets all current images, flavors and availability zone of Openstack. Then the user selects an image, a flavor and an availability zone for the instance that wishes to create and clicks on create button. Once the above button is clicked, a thread is started for creating the selected instance. The creation process is very similar to the other process

explained in the previous sections. First, we do a call to the *createIsntance()* method of the *ComputeSevice* class passing it as parameter the instance and number of instance to generate.

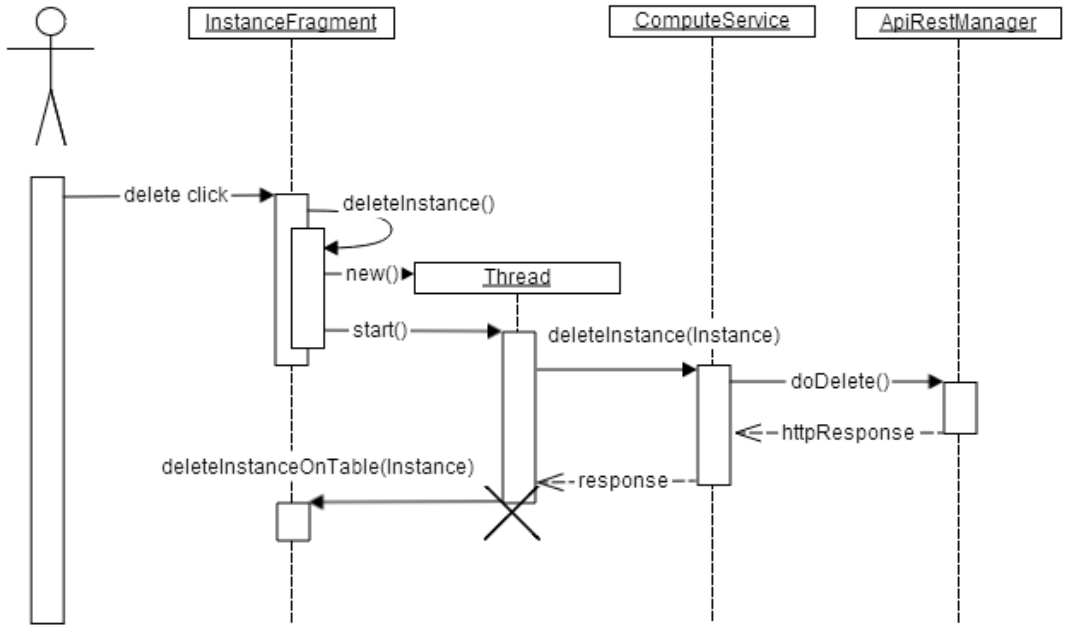
```
public String createInstance(Instance instance, int instanceCount){
	String response;
	ArrayList<BasicNameValuePair> headers = new ArrayList<>();
	headers.add(new BasicNameValuePair("X-Auth-Token",
	UserAuthDataPreferences.getInstance(mContext).getTokenId()));
	headers.add(new BasicNameValuePair("Content-Type",
			"application/json"));
	try{
		JSONObject payload = new JSONObject();
		JSONObject server = new JSONObject();
		server.put("name", instance.getName());
		server.put("imageRef", instance.getImage().getId());
		server.put("flavorRef",instance.getFlavor().getId());
		server.put("availability_zone",
			instance.getAvailabilityZone());
		server.put("max_count", instanceCount);
		server.put("min_count", instanceCount);
		payload.put("server", server);

		StringEntity entity = new StringEntity(payload.toString());
		HttpResponse httpResponse =
			apiRestManager.doPost(ApiRestManager
			.APIServices.get(ApiRestManager.COMPUTE_SERVICE)
			.getPublicUrl() + "/servers", entity,
			headers);

		final int statusCode =
			httpResponse.getStatusLine().getStatusCode();

		if(statusCode == 202) {
			response =
				EntityUtils.toString(httpResponse.getEntity());

			String instanceId = new JSONObject(response)
			.getJSONObject("server")
			.getString("id");
			instance.setId(instanceId);
			return ApiRestManager.OK;
		}
	. . .
}
```

createInstance() method of the computeService class

Then, the compute service builds a call to the API of Openstack that it allows to create instances. To connect to the API, the login service uses the REST services provided by *ApiRestManager* class. For this case, we do a “POST” request through the method “doGet()” and it return us a http response with the final result of the process. Finally, a response is passed to the Instance fragment and if it is correct, the UI is updated through the *addInstanceOnTable()* method.

Instance deletion process

Finally, we explain you the instance deletion process.



Once the user clicks on delete button, the *deleteInstance()* method is called. This starts a thread for deleting the selected instance. The thread does a call to the *deleteIsntance()* method of the *ComputeSevice* class passing it as parameter the instance to delete.

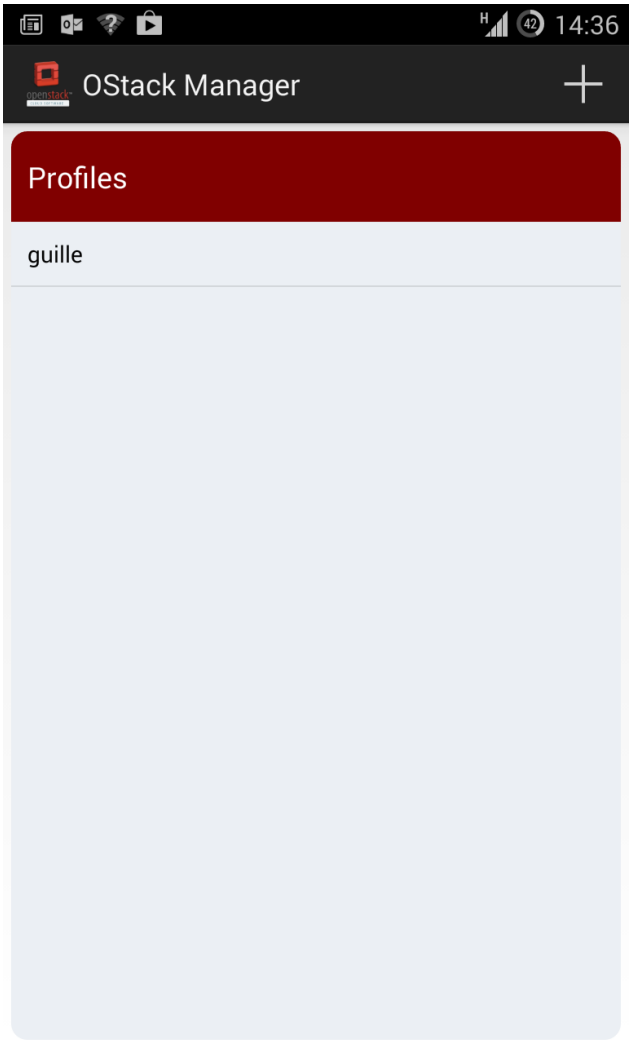
Then, the compute service builds a call to the API of Openstack that it allows to create instances. To connect to the API, the login service uses the REST services provided by *ApiRestManager* class. For this case, we do a “DELETE” request through the method “doDelete()” and it return us a http response with the final result of the process. Finally, a response is passed to the Instance fragment and if it is correct, the UI is updated through the *deleteInstanceOnTable()* method.

Getting Started

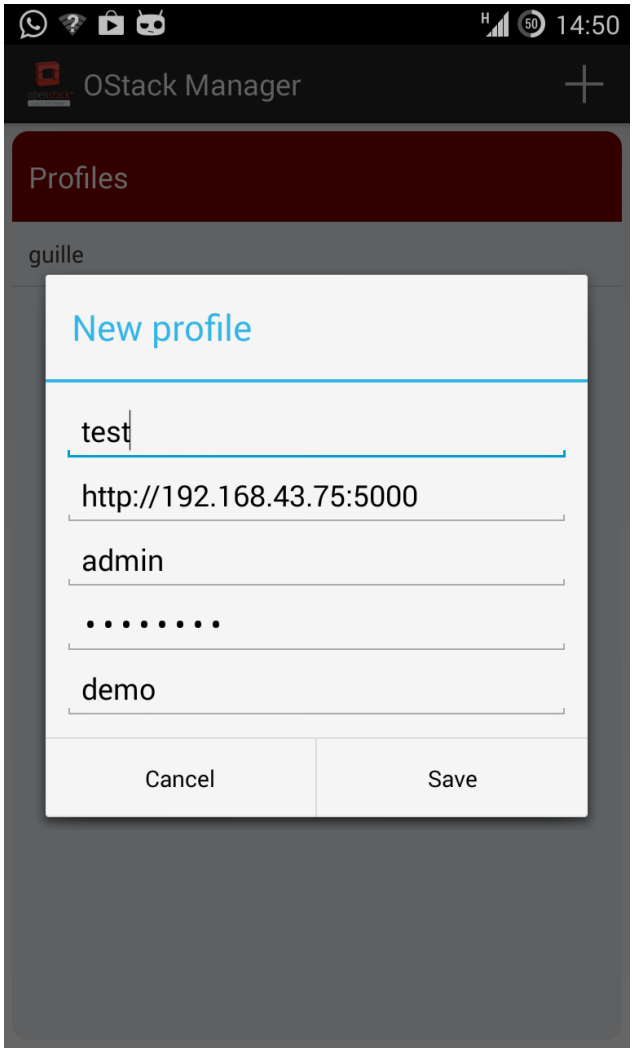
This section describes as getting started with OStack Manager. Before you can begin to use the app, you need to know the following data:

- URL of the Openstack endpoint where is the keystone is listening (Example: http://<keystone endpoint Dir>:5000).
- User name.
- Password.
- Project name (tenant name).

Then, you should create a connection profile with the data mentioned above. To create a new profile, click the add button in the action bar.



Note that it is the first screen visualized when the app is started.



Once you have saved the new profile, you can now log in and begin to use all functionalities of OStack Manager. To log in with the new created profile, look for it into profile table and click it.