Les Webservices REST sous Android avec App Engine, Restlet et Objectify

par Feanorin

Date de publication : 12 mai 2011

Dernière mise à jour :

Suite à de nombreuses demandes, ce tutoriel va aborder un sujet important qui est l'interaction entre une application **Android** et un serveur **Google App Engine**, en utilisant une communication via **Restlet**.

I - Présentation

Notre application permettra d'ajouter et voir des utilisateurs depuis l'interface **Android**. Voici un ensemble complet des technologies que l'on va utiliser pour ce projet :

- **SDK Android** ;
- **Restlet** ;
- **Google App Engine** ;
- **Objectify**.

Vous allez comprendre au fur et à mesure l'utilité de chaque technologie dans notre application.
J'ai découpé ce tutoriel en 3 parties :

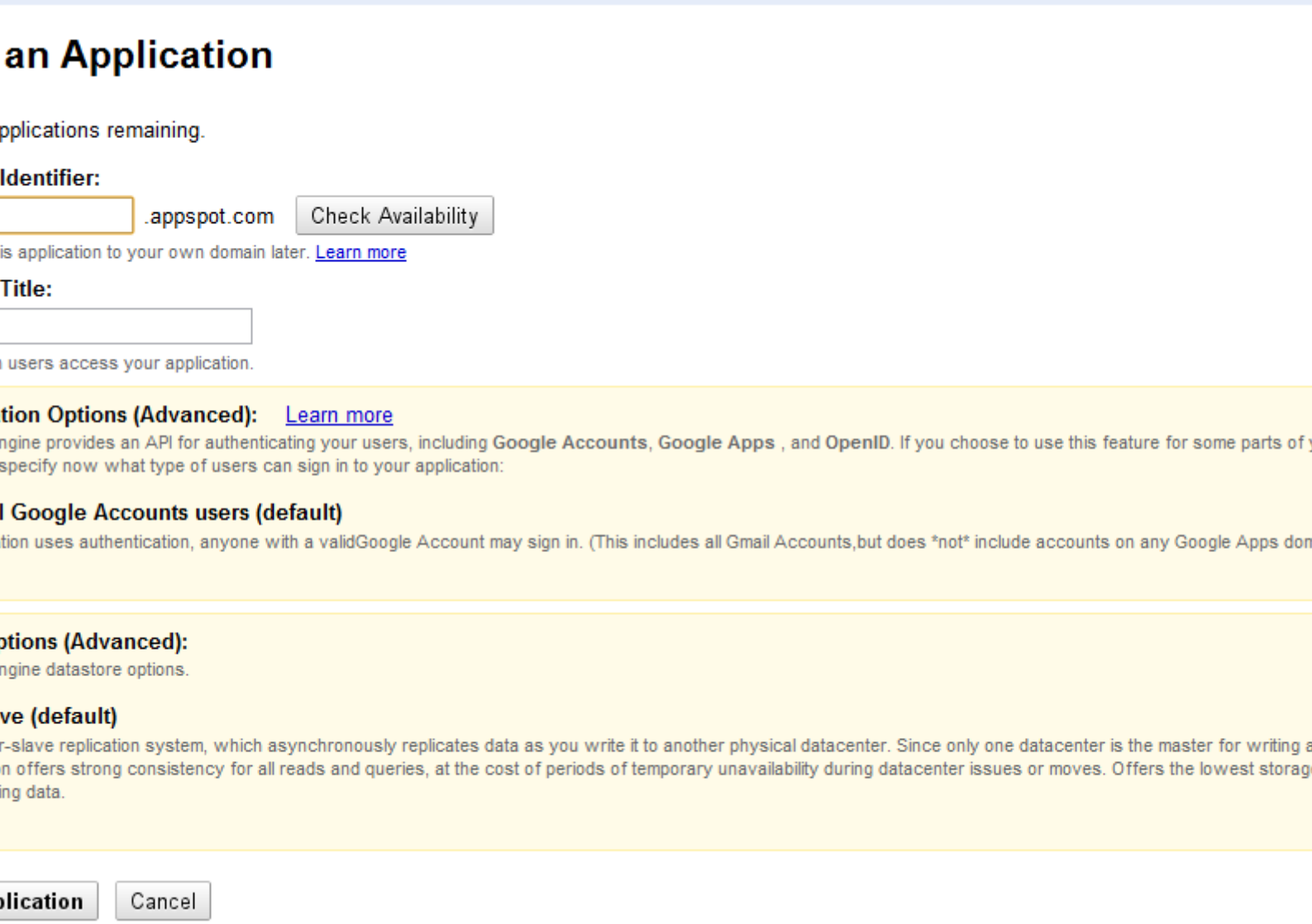
- la création du serveur et sa mise en place ;
- la création des **Webservices** côté serveur à l'aide de **Restlet** ;
- l'application **Android** pour le côté client.

II - Le côté serveur

Tout d'abord nous allons créer notre serveur **App Engine**. Nous nous occuperons ensuite de l'application web que nous déploierons sur le serveur **App Engine**. Enfin, nous mettrons en place les **Webservices Restlet**.

2.1 - Créer le serveur App Engine

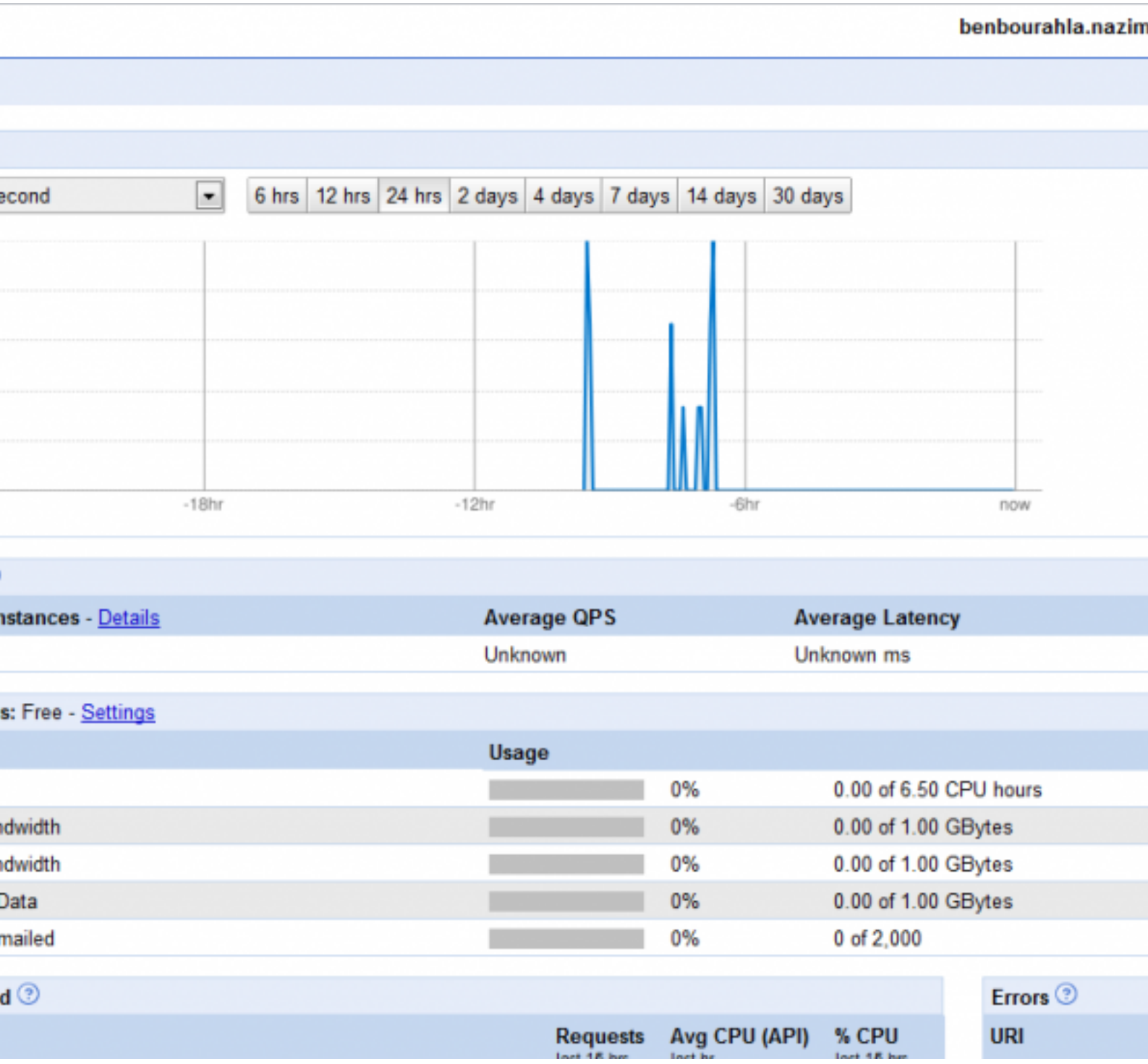
Pour créer le serveur **App Engine**, il suffit de vous connecter sur **https://appengine.google.com/** à l'aide de votre compte Google. Vous disposez de 10 applications gratuites.
Une fois sur l'URL, vous arriverez sur cette page :



Il suffit ensuite de remplir les champs (l'identifiant de l'application est unique donc vous ne pourrez pas utiliser le même que moi pour vos tests !).
Donc pour mon exemple je choisis :

- **android-gae-restlet** comme identifiant ;
- **Tutos Android - Restlet GAE** comme nom de l'application ;
- puis cliquez sur Create Application.

Maintenant quand vous retournez sur votre page **App Engine**, vous verrez ce projet dans la liste et vous pourrez accéder au tableau de bord de votre application ci-dessous :

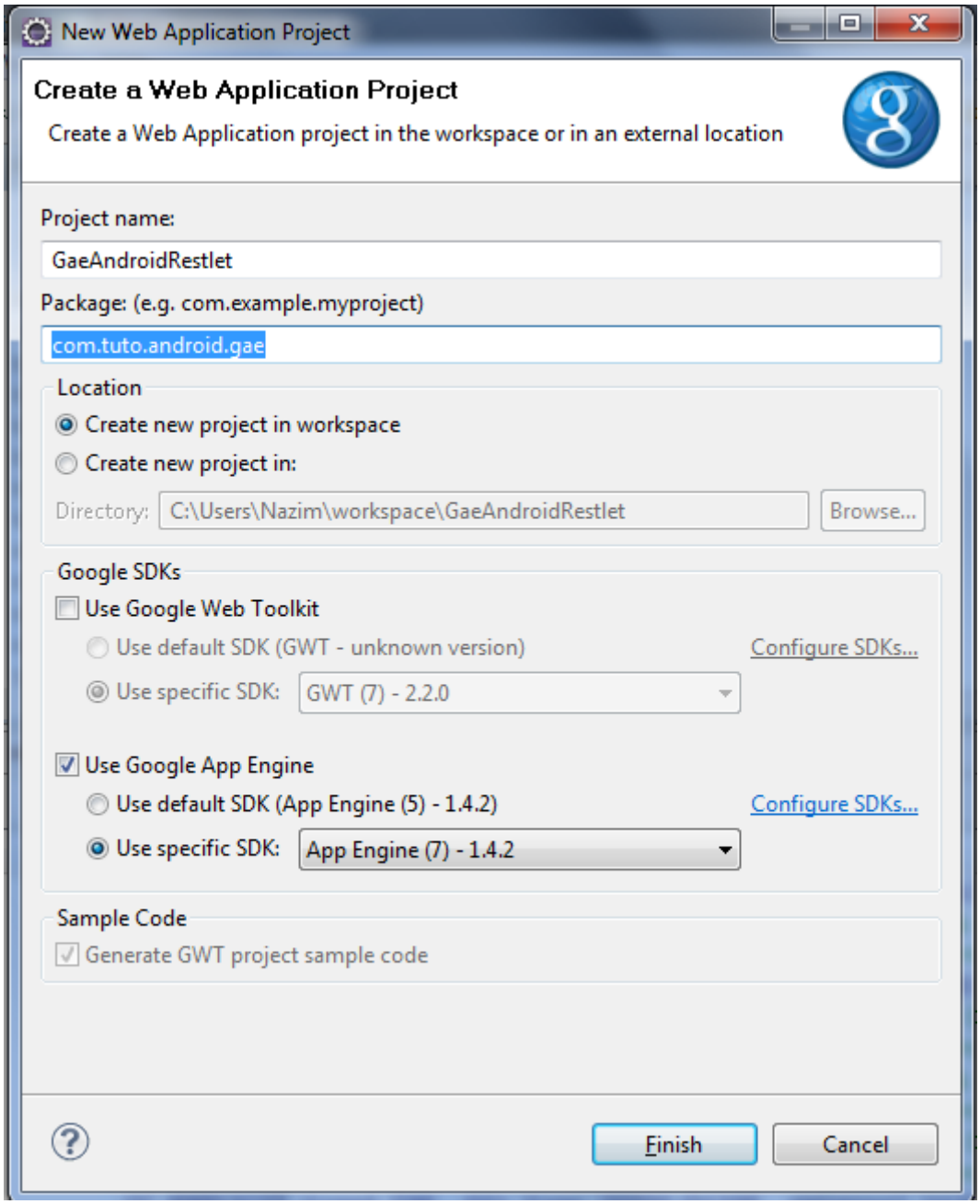


Voici une explication des plus importants onglets disponibles :

- **DashBoard** : graphique et statistiques sur l'utilisation des ressources ;
- **Logs** : équivaut un peu à system.out / err, ce sont les logs de votre application ;
- **Datastore Viewer** : pour voir le contenu de votre base de données ;
- **Application Settings** : paramétrage de votre application ;
- **Permissions** : pour gérer les permissions sur votre application ;
- **Version** : pour gérer les différentes versions de votre application.

2.2 - Créer l'application web

Maintenant vous pouvez ouvrir Eclipse (vous devez auparavant installer les **plugins Google** (**http://code.google.com/intl/fr-FR/eclipse/**) et télécharger le **SDK App Engine** (**http://code.google.com/intl/fr-FR/appengine/downloads.html**)). Puis, vous serez prêt à commencer. Dans Eclipse, faites un nouveau "**Web Application Project**", que l'on nommera "**GaeAndroidRestlet**" avec pour nom de package "**com.tuto.android.gae**". Désélectionnez le **SDK GWT** et sélectionnez la version du **SDK App Engine** (1.4.2 dans mon cas). Voici la fenêtre que vous devriez obtenir :



Cliquez ensuite sur Finish et le projet est créé.

2.3 - Configurer l'application

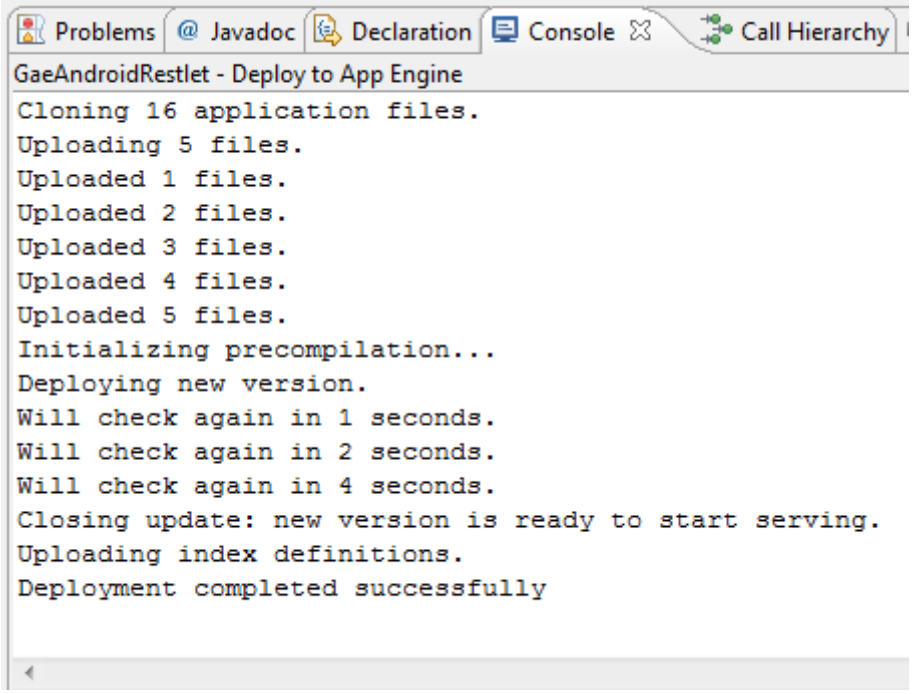
Dans le dossier war/WEB-INF de votre projet se trouve un fichier "**appengine-web.xml**" il faut rajouter l'id de votre application entre la balise <application>, donc dans notre cas :

Maintenant quand vous retournez sur votre page App Engine, vous verrez ce projet dans la liste et vous pourrez accéder au tableau de bord de votre application ci-dessous :

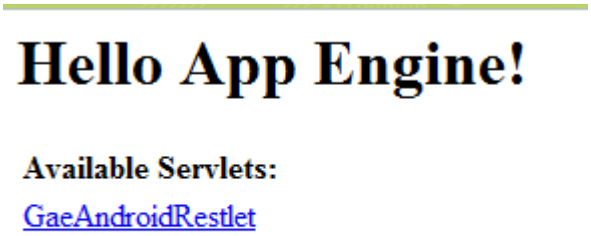
```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
    <application>android-gae-restlet</application>
    <version>1</version>

    <!-- Configure java.util.logging -->
    <system-properties>
        <property name="java.util.logging.config.file" value="WEB-INF/logging.properties"/>
    </system-properties>

</appengine-web-app>
```

Déployons une première fois pour voir. Il suffit pour cela de cliquer sur l'icône App Engine dans votre Eclipse (icône de moteur d'avion).

Remplissez les données et validez l'opération pour que la phase de déploiement puisse commencer. Un message vous montrera que le déploiement à réussi :



Pour tester il suffit de vous rendre à l'adresse suivante : iddevotreapplication.appspot.com, donc dans notre cas **http://android-gae-restlet.appspot.com/** et vous obtiendrez le résultat suivant :



2.4 - Création du modèle

Maintenant que la structure de notre projet est initialisée, créez une classe User qui nous servira à stocker les données en base. Pour indication la base de données de **Google App Engine** est **Big Table**, c'est une base **NoSql**.
Nous allons utiliser **Objectify**, qui est un framework permettant de simplifier la persistance des données et leurs manipulations dans **Google App Engine**, il rajout une couche d'abstraction facilitant énormément les différentes interactions.
Nous allons créer notre modèle qui sera stocké en base. Pour cela nous créons un package model dans lequel nous ajoutons une classe User contenant les différents champs.

- Le modèle doit absolument implémenter **Serializable**.
- Vous devez générer un numéro de serial version.
- Vous devez avoir un champ annoté de **@Id** qui sera l'id de votre base, (nécessaire pour **Objectify**).
- Vous devez avoir un constructeur vide.
- Vous devez avoir un getter / setter pour chaque champs.
- Rien de bien compliqué jusqu'à présent.

```
package com.tuto.android.gae.model;

import java.io.Serializable;

import javax.persistence.Id;

public class User implements Serializable {

  private static final long serialVersionUID = 7390103290165670089L;
  @Id
  private Long id;
  private String firstname;
  private String lastname;
  private String mail;
  private String phone;

  public User() {

  }

  public Long getId() {
   return id;
  }

  public void setId(Long id) {
   this.id = id;
  }

  public void setMail(String mail) {
   this.mail = mail;
  }

  public String getMail() {
   return mail;
  }

  public String getFirstname() {
   return firstname;
  }

  public void setFirstname(String firstname) {
   this.firstname = firstname;
  }

  public String getLastname() {
   return lastname;
  }
```

```
  public void setLastname(String lastname) {
   this.lastname = lastname;
  }

  public String getPhone() {
   return phone;
  }

  public void setPhone(String phone) {
   this.phone = phone;
  }

}
```

2.5 - Création des Webservices Restlet

Nous allons créer nos **Webservices Restlet**. Pour cela il faut commencer par télécharger les **jar** de Restlet que je fournis **ici** et rajouter les à votre projet. Une fois cette étape faite nous allons créer nos **webservices**.
La première étape pour créer des webservices restlet côté serveur est de modifier le **web.xml** pour rajouter les lignes suivantes :

```
<context-param>
    <param-name>org.restlet.application</param-name>
    <param-value>com.tuto.android.gae.server.RestletDispatch</param-value>
</context-param>
```

Cette première ligne sert à initialiser **restlet** et ce qu'on appelle un **Dispatcher**. Ce dernier recevra les appels Reslet et s'occupera de les rediriger vers la bonne classe pour être traiter par le serveur.
Puis nous allons déclarer les **Servlets** utiles pour nos **webservices**. On déclare un servlet qu'on nommera **RestletServlet** et qui correspond à un servlet restlet et on le map avec tout les liens qui finissent par **/rest/***. Donc tous les liens contenant **/rest** seront redirigés vers notre RestletDispatch.

```
<servlet>
      <servlet-name>RestletServlet</servlet-name>
      <servlet-class>
         org.restlet.ext.servlet.ServerServlet
      </servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>RestletServlet</servlet-name>
      <url-pattern>/rest/*</url-pattern>
   </servlet-mapping>
```

Voilà la version finale de **web.Xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">

    <context-param>
        <param-name>org.restlet.application</param-name>
        <param-value>com.tuto.android.gae.server.RestletDispatch</param-value>
    </context-param>

    <servlet>
        <servlet-name>GaeAndroidRestlet</servlet-name>
        <servlet-class>com.tuto.android.gae.GaeAndroidRestletServlet</servlet-class>
    </servlet>
    <servlet-mapping>
```

```
        <servlet-name>GaeAndroidRestlet</servlet-name>
        <url-pattern>/gaeandroidrestlet</url-pattern>
    </servlet-mapping>

    <servlet>
      <servlet-name>RestletServlet</servlet-name>
      <servlet-class>
         org.restlet.ext.servlet.ServerServlet
      </servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>RestletServlet</servlet-name>
      <url-pattern>/rest/*</url-pattern>
   </servlet-mapping>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

Maintenant nous allons créer un package server, dans lequel nous allons créer tous nos classes indispensables à notre serveur dont notre restlet dispatch qui ressemblera à ceci.

```
package com.tuto.android.gae.server;

import java.io.File;
import org.restlet.Application;
import org.restlet.Restlet;
import org.restlet.data.ChallengeScheme;
import org.restlet.data.LocalReference;
import org.restlet.data.Protocol;
import org.restlet.resource.Directory;
import org.restlet.routing.Router;
import org.restlet.security.Guard;

@SuppressWarnings("deprecation")
public class RestletDispatch extends Application
{
  @Override
  public synchronized Restlet createInboundRoot()
  {
   final Router router = new Router(getContext());
   router.attach("/user", UserController.class);
   return router;
  }
}
```

- Donc un **Dispatcher** est une classe qui hérite **Application** et qui implémente la méthode **createInboundRoot**.
- On crée un nouveau **Router** avec le **context** actuel de l'application.
- On indique que les liens contenant **/user** seront redirigés vers la classe UserController.
- Pas besoin d'indiquer **/rest** qui est déjà indiqué dans le web.xml.
- Si vous devez passer des arguments à la méthode get, il faut les passer dans l'URL : par exemple **/mail/{mail}/pass/{pass}** pour passer l'adresse mail et le mot de passe et il suffit de mettre cette valeur à la place de **/user** dans votre **router.attach**.
- Pour récupérer ces valeurs passées en paramètre il suffit d'utiliser :

```
Request req = super.getRequest();
mail = req.getAttributes().get("mail").toString();
```

- Vous pouvez avoir plusieurs router.attach.

Pour commencer nous allons créer un UserControllerInterface, qui sera l'api exposé au client et qui fera le lien entre le client et le serveur.

```
package com.tuto.android.gae.server;

import org.restlet.resource.Get;
import org.restlet.resource.Put;

import com.tuto.android.gae.model.User;

public interface UserControllerInterface {
  @Put
  void create(User user);

  @Get
  Container getAllUsers();

}
```

Donc on a juste une méthode get et une autre put, rien de bien compliqué. Vous remarquerez la présence d'une classe Container. Pourquoi cette classe, car si jamais vous voulez retourner une List (ArrayList) par exemple, Reslet vous ne retournera pas forcément un ArrayList formatté comme vous le souhaitez donc pour éviter ce problème on crée une classe à nous et on retourne cet objet et comme vous pouvez le voir cette classe contient juste un ArrayList Ce qui donnera :

```
package com.tuto.android.gae.server;

import java.util.ArrayList;
import java.util.List;

import com.tuto.android.gae.model.User;

public class Container
{
  public List user_list;

  public List getUser_list() {
   return user_list;
  }

  public void setUser_list(List user_list) {
   this.user_list = user_list;
  }

  public Container()
  {
   user_list = new ArrayList();
  }

  public Container(List user_list)
  {
   this.user_list = user_list;
  }

}
```

Passons à notre **UserController**, qui s'occupera de récupérer la liste des utilisateurs ainsi que de stocker les données grâce à **objectify** dont voici le **jar à inclure**. (**Objectify** est un framework utilisé pour faciliter la persistance et la récupération des données depuis un serveur **app engine**). Vous pouvez utiliser un autre framework si vous le souhaitez. Cette technologie n'est pas une obligation.
Donc dans notre **UserController** nous allons implémenter notre **create** ainsi que notre **getAllUser**.

- Notre **UserController** doit absolument hériter **ServerResource** et implémenter notre interface **UserControllerInterface**.
- Elle doit posséder un constructeur vide.
- Pour notre **create**, il suffit d'enregistrer notre modèle et obtenir une instance d'objectify service, puis on crée notre objet user et on le stocke, facile.

- Dans une méthode **@Put** (donc notre **create**) on peut passer qu'**un seul argument** à la méthode Restlet donc.
- Pour notre **getAllUser**, comme précédemment on fait un register et on récupère une instance (cette partie est obligatoire pour chaque utilisation d'**objectify**). On récupère tous les utilisateurs à l'aide d'une query (vous pouvez faire des filtres sur la query pour préciser ce que vous souhaitez) puis on crée notre container et on le retourne.

```
package com.tuto.android.gae.server;

import java.util.ArrayList;
import java.util.List;

import org.restlet.resource.ServerResource;

import com.googlecode.objectify.Objectify;
import com.googlecode.objectify.ObjectifyService;
import com.googlecode.objectify.Query;
import com.tuto.android.gae.model.User;

public class UserController extends ServerResource implements
 UserControllerInterface {

  public UserController() {
  }

  @Override
  public void create(User user) {
   ObjectifyService.register(User.class);
   Objectify ofy = ObjectifyService.begin();

   User tp = new User();
   tp.setFirstname(user.getFirstname());
   tp.setLastname(user.getLastname());
   tp.setMail(user.getMail());
   tp.setPhone(user.getPhone());
   ofy.put(tp);
  }

  @Override
  public Container getAllUsers() {
   Container content = null;
   List users = new ArrayList();
   ObjectifyService.register(User.class);
   Objectify ofy = ObjectifyService.begin();

   Query q = ofy.query(User.class);

   for (User u : q)
   users.add(u);

   content = new Container();
   content.setUser_list(users);

   return content;
  }
}
```

Voici notre partie serveur terminée, il faut juste la **déployer** comme expliqué précédemment. Voici le **code source** du serveur si vous le souhaitez.

III - Le côté client

Nous allons créer un projet Android qu'on nommera **GaeAndroidRestletAndroid**, avec un **SDK 2.1**, avec comme nom d'application **App Engine Client** et comme package **com.tuto.android.gae** et comme activité de base **GaeHomeScreen**.

Nous allons ajouter tout d'abord les libs restlet.
Cette application nous permettra soit :

- d'ajouter un utilisateur ;
- récupérer la liste de tous les utilisateurs.

Il faut créer notre première vue qui contient deux boutons (ajout utilisateur et récupération de la liste des utilisateurs).
Donc le fichier **XML (main.xml)**.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/add_user"
    android:id="@+id/addUser"
    />

<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/get_users"
     android:id="@+id/getUsers"
    />

</LinearLayout>
```

Et le fichier java correspondant.

```
package com.tuto.android.gae;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class GaeHomeScreen extends Activity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);

   Button addUser = (Button) findViewById(R.id.addUser);
   addUser.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View arg0) {
     Intent i = new Intent(GaeHomeScreen.this, AddUserActivity.class);
     startActivity(i);
    }
   });

   Button getUsers = (Button) findViewById(R.id.getUsers);
   getUsers.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View arg0) {
     Intent i = new Intent(GaeHomeScreen.this, GetAllUserActivity.class);
```

```
      startActivity(i);
     }
    });

   }

}
```

Il faut créer un package model, dans ce dernier il faut mettre toutes les classes indispensables à la communication restlet.

3.1 - Container

- C'est le même que côté serveur.
- Il doit être pareil, car il est utilisé par Restlet pour convertir le JSON reçu depuis le serveur

```
package com.tuto.android.gae.model;

import java.util.ArrayList;
import java.util.List;

import com.tuto.android.gae.model.User;

public class Container
{
  public List user_list;

  public List getUser_list() {
   return user_list;
  }

  public void setUser_list(List user_list) {
   this.user_list = user_list;
  }

  public Container()
  {
   user_list = new ArrayList();
  }

  public Container(List user_list)
  {
   this.user_list = user_list;
  }

}
```

3.2 - EngineConfiguration

- Cette classe nous sert à configurer notre application pour savoir sur quel serveur elle se connecte.
- Cette classe suit le design pattern **singleton**.
- On stocke l'adresse vers le lien de notre serveur.
- On instancie un Json converter.
- Vous pouvez reprendre cette classe pour vos autres projets il suffit juste de changer le **gae_path**.

```
package com.tuto.android.gae.model;

import org.restlet.engine.Engine;
import org.restlet.ext.httpclient.HttpClientHelper;
import org.restlet.ext.jackson.JacksonConverter;

public class EngineConfiguration
```

```
{
    private static EngineConfiguration  ourInstance = new EngineConfiguration();

    public final static String  gae_path = "http://android-gae-restlet.appspot.com/";

    public static EngineConfiguration getInstance()
    {
        return ourInstance;
    }

    public EngineConfiguration()
    {
        Engine.getInstance().getRegisteredConverters().add(new JacksonConverter());
        Engine.getInstance().getRegisteredClients().add(new HttpClientHelper(null));
    }

    public static EngineConfiguration getOurInstance()
    {
        return ourInstance;
    }

    public static void setOurInstance(EngineConfiguration ourInstance)
    {
        EngineConfiguration.ourInstance = ourInstance;
    }
}
```

3.3 - User

- Doit être exactement pareil que coté serveur pour la conversion JSON.
- Il suffit juste d'enlever l'annotation **@id**.
- Sinon le reste est pareil.

```
package com.tuto.android.gae.model;

import java.io.Serializable;

public class User implements Serializable {

  private static final long serialVersionUID = 7390103290165670089L;
  private Long id;
  private String firstname;
  private String lastname;
  private String mail;
  private String phone;

  public User() {

  }

  public Long getId() {
   return id;
  }

  public void setId(Long id) {
   this.id = id;
  }

  public void setMail(String mail) {
   this.mail = mail;
  }

  public String getMail() {
   return mail;
  }

  public String getFirstname() {
```

```
    return firstname;
  }

  public void setFirstname(String firstname) {
   this.firstname = firstname;
  }

  public String getLastname() {
   return lastname;
  }

  public void setLastname(String lastname) {
   this.lastname = lastname;
  }

  public String getPhone() {
   return phone;
  }

  public void setPhone(String phone) {
  this.phone = phone;
  }

}
```

3.4 - UserControllerInterface

- Même interface que côté serveur.
- De ce fait, les deux parties exposent des méthodes identiques pour la communication entre les deux.

```
package com.tuto.android.gae.model;

import org.restlet.resource.Get;
import org.restlet.resource.Put;

public interface UserControllerInterface {
  @Put
  void create(User user);

  @Get
  Container getAllUsers();
}
```

3.5 - UserController

- C'est la partie importante coté client pour les webserices.
- À la différence de coté serveur, elle ne doit pas implémenter le UserController interface.
- On Crée un client ressource dans lequel on instancie le lien vers notre webservice qu'on a défini dans le serveur (cf web.xml et RestletDispatcher).
- On instancie le EngineConfiguration dans le constructeur.
- Il faut toujours faire un wrap de l'interface avant d'appeler la méthode souhaitée.
- Le reste parle de lui-même.

```
package com.tuto.android.gae.model;

import java.util.List;

import org.restlet.resource.ClientResource;

import android.util.Log;

public class UserController {
  public final ClientResource cr = new ClientResource(
```

```
  EngineConfiguration.gae_path + "/rest/user");

  public UserController() {
   EngineConfiguration.getInstance();
  }

  public void create(User user) throws Exception {
   final UserControllerInterface uci = cr
   .wrap(UserControllerInterface.class);

   try {
    uci.create(user);
    Log.i("UserController", "Creation success !");
   } catch (Exception e) {
    Log.i("UserController", "Creation failed !");
    throw e;
   }
  }

  public List getAllUsers() {
   final UserControllerInterface uci = cr
   .wrap(UserControllerInterface.class);
   Container content = uci.getAllUsers();
   return content.getUser_list();
  }
}
```

3.6 - Création des vues

3.6.1 - Nouvel Utilisateur

Nous allons commencer par créer la vue pour rajouter un utilisateur (donc fichier XML plus Java).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:hint="Nom" android:id="@+id/userName" />

    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:hint="Prenom" android:id="@+id/userFirstName" />

    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:hint="Email" android:id="@+id/userMail" />

    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:hint="Telephone"
        android:id="@+id/userPhone" />

    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Ajouter cet utilisateur"
        android:id="@+id/addUserNow" />

</LinearLayout>
```

Et le fichier java.

```
package com.tuto.android.gae;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

import com.tuto.android.gae.model.User;
import com.tuto.android.gae.model.UserController;

public class AddUserActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.add_user);

    Button btn = (Button) findViewById(R.id.addUserNow);
    btn.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {
            addUser();
        }
    });
    }

    final void addUser()
    {
    final Thread checkUpdate = new Thread() {
        public void run() {
        EditText name = (EditText) findViewById(R.id.userName);
        EditText firstname = (EditText) findViewById(R.id.userFirstName);
        EditText mail = (EditText) findViewById(R.id.userMail);
        EditText phone = (EditText) findViewById(R.id.userPhone);
        User u = new User();
        u.setFirstname(firstname.getText().toString());
        u.setLastname(name.getText().toString());
        u.setMail(mail.getText().toString());
        u.setPhone(phone.getText().toString());
        final UserController uc = new UserController();
        try {
            uc.create(u);
        } catch (Exception e) {
            return;
        }
        final Intent intent = new Intent(AddUserActivity.this,
            GaeHomeScreen.class);
        startActivity(intent);
        }
    };
    checkUpdate.start();
    }
}
```

3.6.2 - Récupération des Utilisateurs

Puis on crée la vue de récupération des utilisateurs.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<ListView
                android:id="@+id/listLists"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:layout_marginTop="10px"
                />

</LinearLayout>
```

Et le fichier java.

```
package com.tuto.android.gae;

import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import com.tuto.android.gae.model.User;
import com.tuto.android.gae.model.UserController;

public class GetAllUserActivity extends Activity {
 private List lists = null;
 private List listsName = null;
 private ListView listLists;

 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.get_users);

  listLists = (ListView) findViewById(R.id.listLists);

  getUsers();

 }

 final void getUsers() {

  UserController list = new UserController();
  listsName = new ArrayList();
  try {
   lists = list.getAllUsers();
  } catch (Exception e) {
   e.printStackTrace();
  }
  if (lists != null) {
   for (User u : lists) {
    if (u != null)
     listsName.add(u.getFirstname() + " " + u.getLastname());
   }

   listLists.setAdapter(new ArrayAdapter(getBaseContext(),
   android.R.layout.simple_list_item_1, listsName));

   listLists.setTextFilterEnabled(true);

  }
 }
}
```

3.6.3 - Autres éléments nécessaires

Sans oublier le strings.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, GaeHomeScreen!</string>
    <string name="app_name">App Engine Client</string>
    <string name="add_user">Ajouter un utilisateur</string>
    <string name="get_users">Voir la liste des utilisateurs</string>
</resources>
```

Et le AndroidManifest.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tuto.android.gae" android:versionCode="1"
    android:versionName="1.0">

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".GaeHomeScreen" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".AddUserActivity" />
        <activity android:name=".GetAllUserActivity" />
    </application>

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

</manifest>
```

Je mets à votre disposition le code de la **partie android**.

IV - Résultat

Voici les différents écrans que vous obtiendrez :

- écran d'accueil ;


10:59
App Engine Client
Ajouter un utilisateur
Voir la liste des utilisateurs

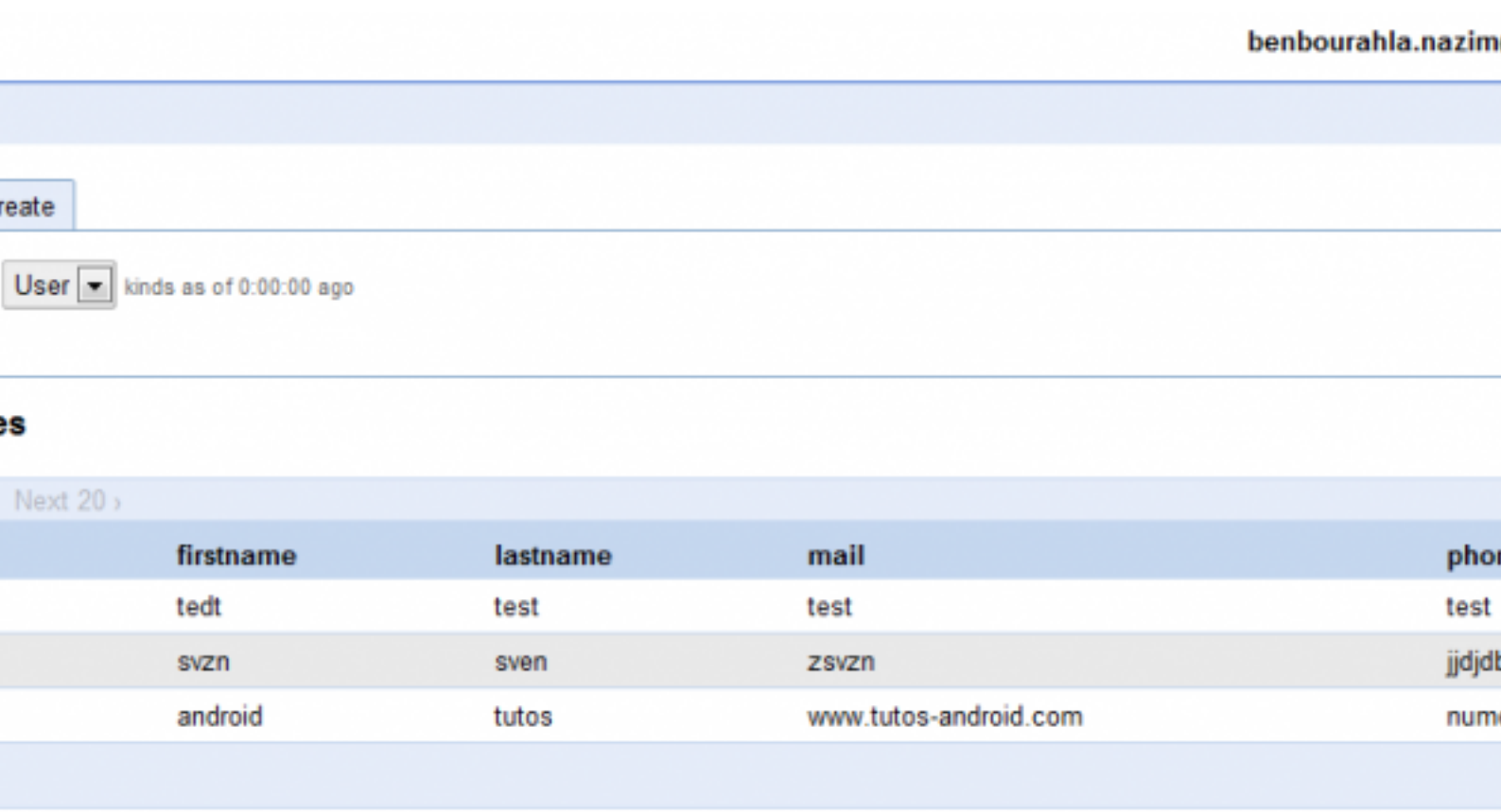
- écran d'ajout d'un utilisateur ;

- liste des utilisateurs.

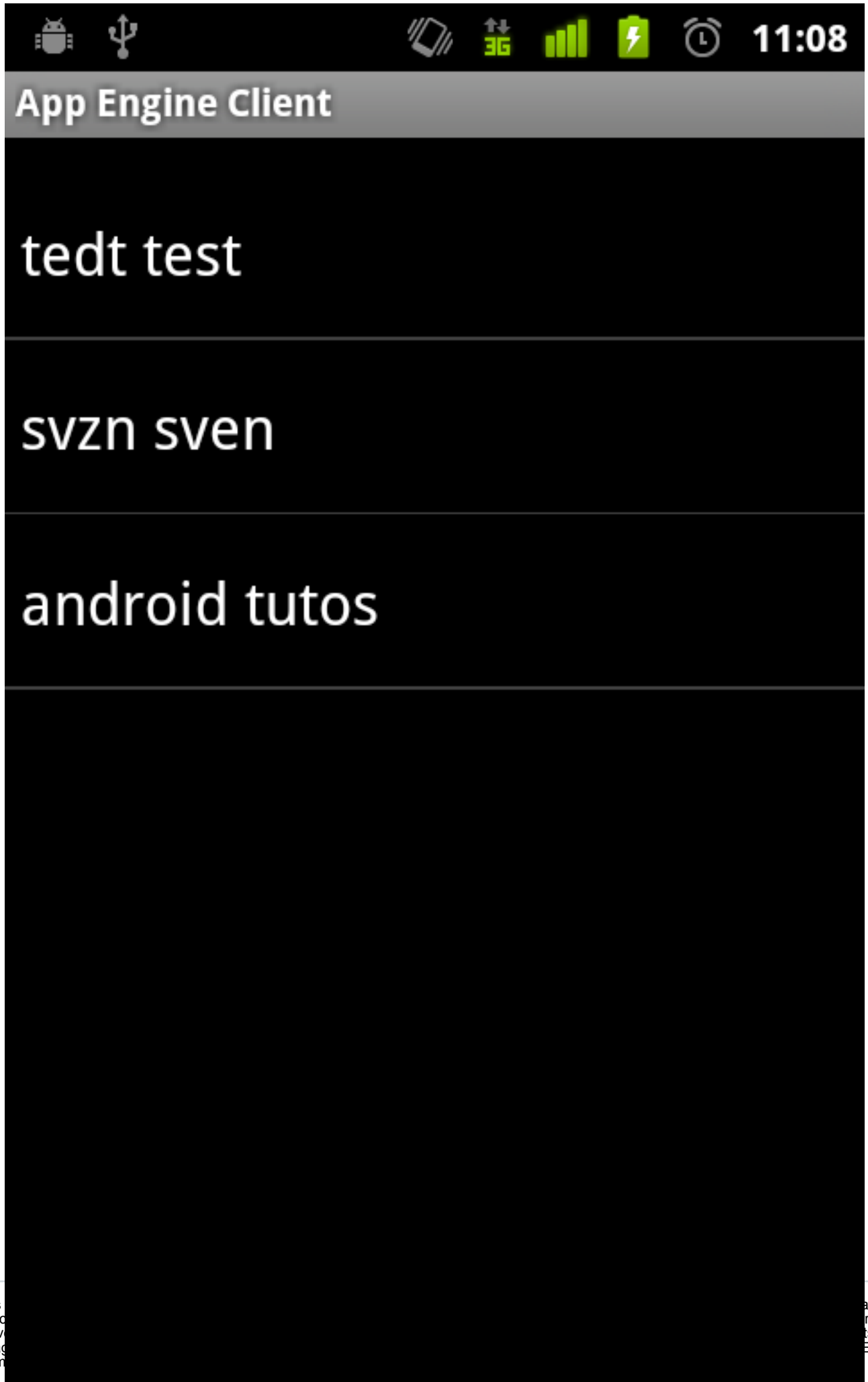
Nous allons tester notre application, pour cela il suffit de la lancer, on va ajouter un nouvel utilisateur.


11:03
App Engine Client
tutos
android
www.tutos-android.com
numero de telephone
Ajouter cet utilisateur

Maintenant pour vérifier que l'ajout c'est bien passé, on va se connecter à notre compte app engine puis à notre application et on va voir le contenu du DataStore Viewer (ce qui devrait donner) :



On remarque bien qu'un champ en plus est apparu avec l'utilisateur qu'on vient d'ajouter. Dernier test, on va afficher la liste des utilisateurs :



Donc l'utilisateur qu'on vient d'ajouter s'affiche bien dans la liste.

V - Conclusion

Voilà c'est ici que s'arrête ce tutoriel, en espérant qu'il vous à aidé à avancer et qu'il à répondu à toutes les questions que vous vous posez. N'hésitez pas à commenter ou à me contacter si une partie ne vous semble pas très claire.

VI - Remerciements

Je tiens à remercier tout particulièrement **Feanorin** qui a mis ce tutoriel au format Developpez.com.
Merci également à **Mahefasoa** d'avoir pris le temps de le relire et de le corriger.

VII - Liens

Tutoriel origine sur Tutos-Android