Google App Engine

Introduction à la programmation
Cloud Computing

J. Saraydaryan

CPE - Lyon

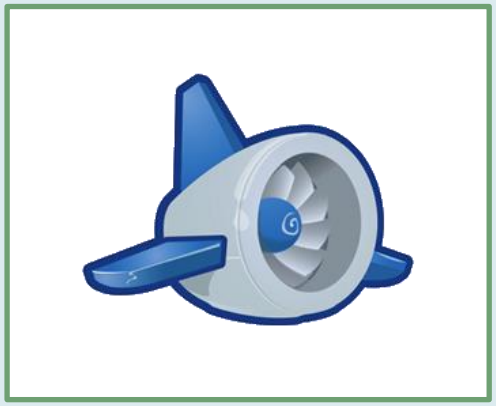
Introduction

Google App Engine

- **Qu'est ce que le Google Apps Engine**

  Plateforme de conception et d'hébergement d'applications

  Web basé sur une infrastructure de Google (serveurs)

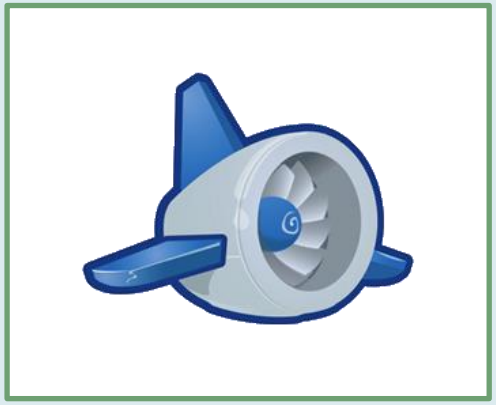


  - Historique
    - 2008 Avril : lancement Google App Engine
    - 2008 4 versions ! (amélioration des fonctionalités)
    - 2009 3 versions
      - Support de nouveaux services
        - Planification de taches automatiques (task Queue)
        - Support du protocol XMPP (messagerie instantanée)
        - Support de réception de mail
      - Amélioration des outils de développement
        - Support du langage Java
        - Plugin pour Eclipse et intégration d'application GWT
        - Python Google App Engine Launcher disponible sur windows
    - 2011 Support officiel par Google (plus en version beta)

Google App Engine

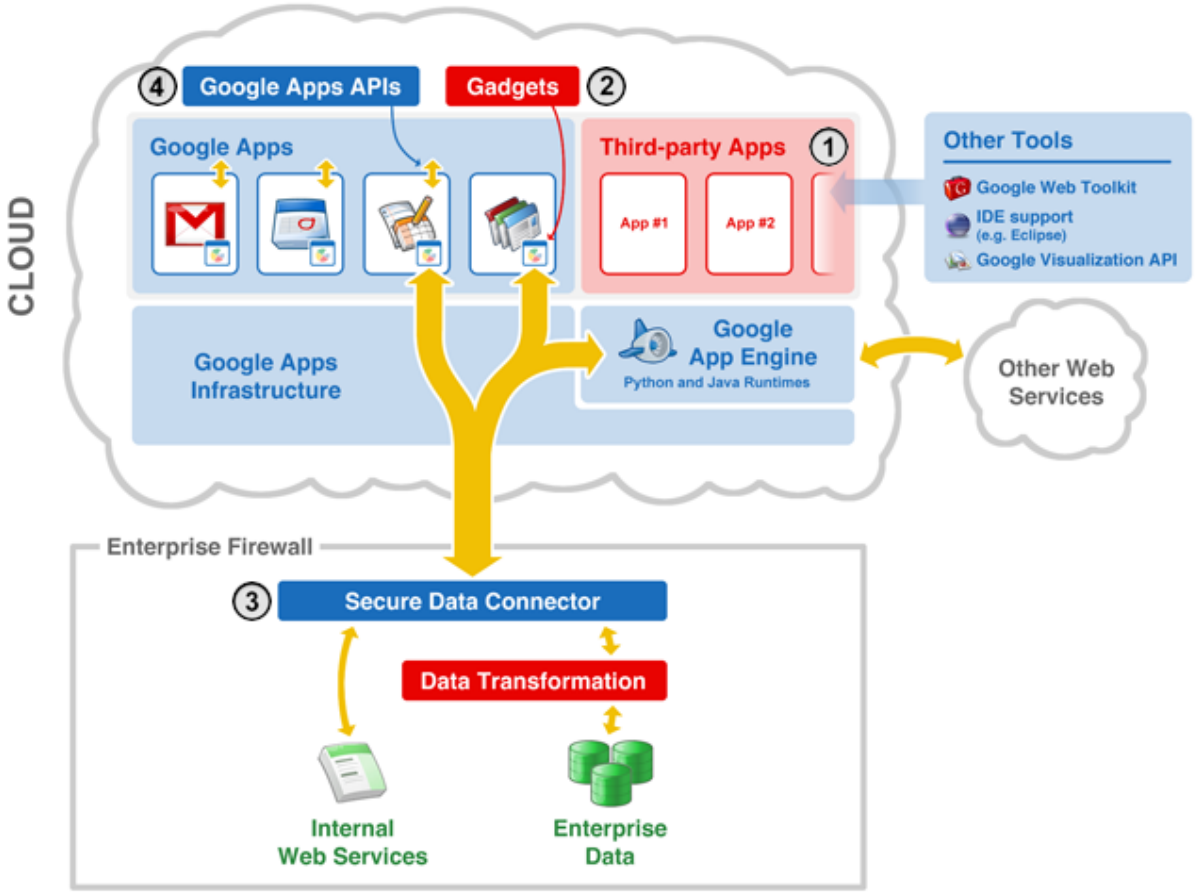
- **Qu'est ce que le Google Apps Engine**
  - Propriétés
    - Diffusion Web dynamique (prise en charge des technos courantes)
    - Stockage permanent (requète, tri et transaction)
    - Equilibrage des charges et évolutivité automatiques

    → Google App Engine adapte les ressources utilisées à vos besoins
    - Exécution de vos applications dans des Sandbox
      - Pas de chevauchement d'espace de travail (sécurité)
      - Contraintes supplémentaires d'utilisation des ressources
    - Files d'attente de tâches (exécution en dehors des requètes web)
    - Planification de taches



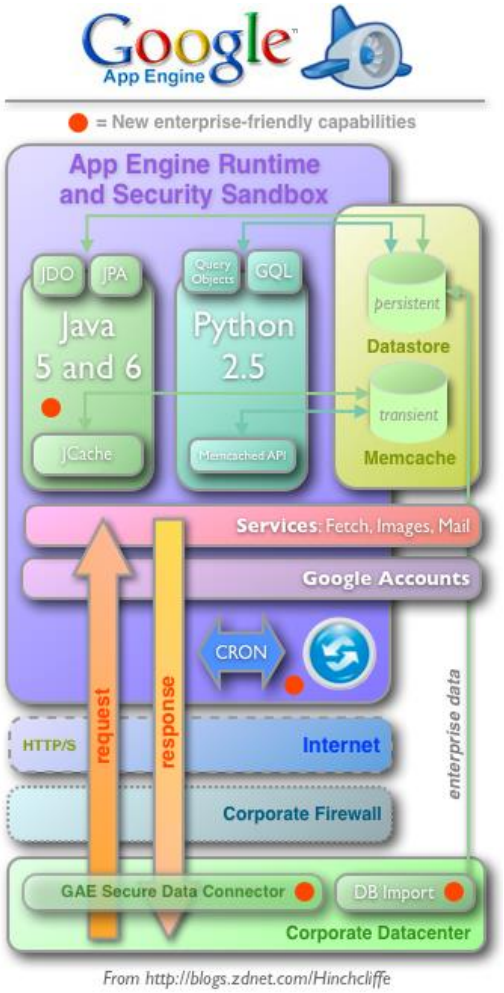
Google App Engine

- **Architecture fonctionnelle 1**

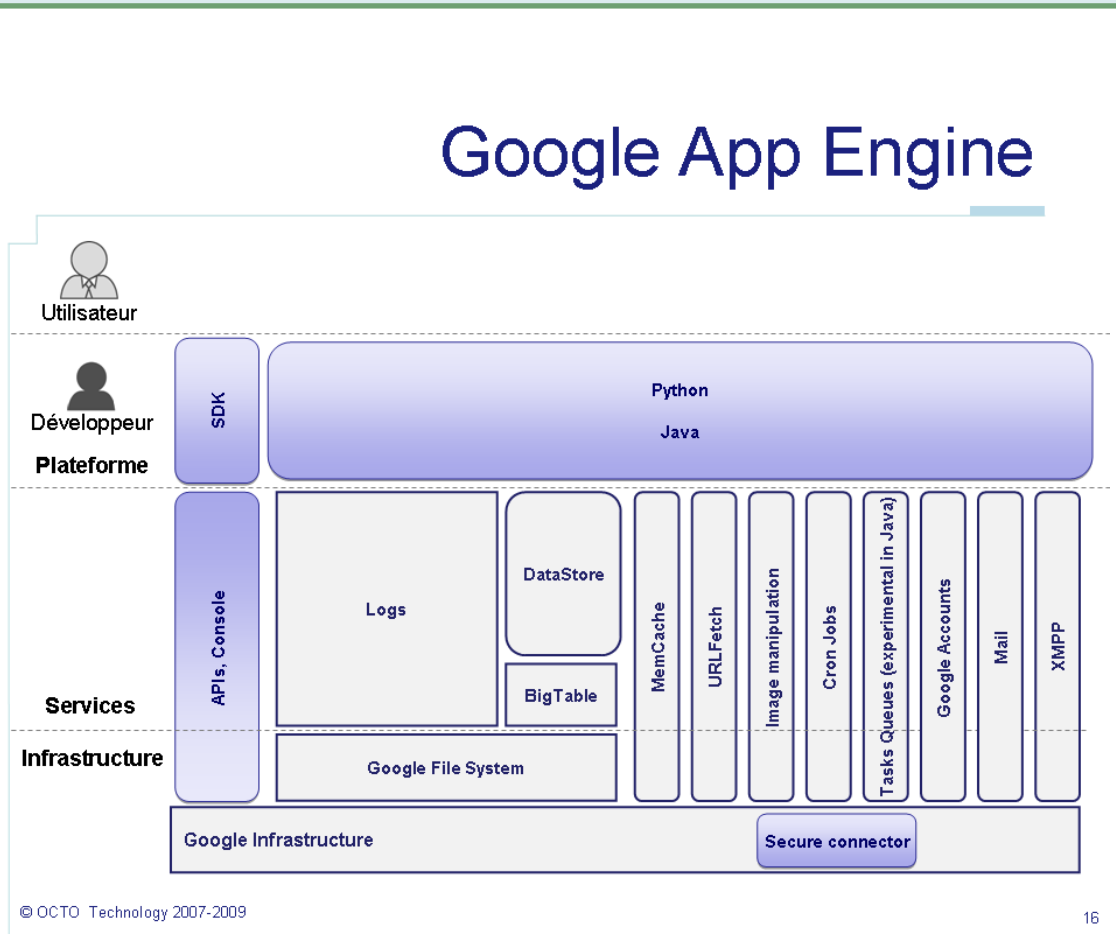


Google App Engine

- **Architecture fonctionnelle 2**

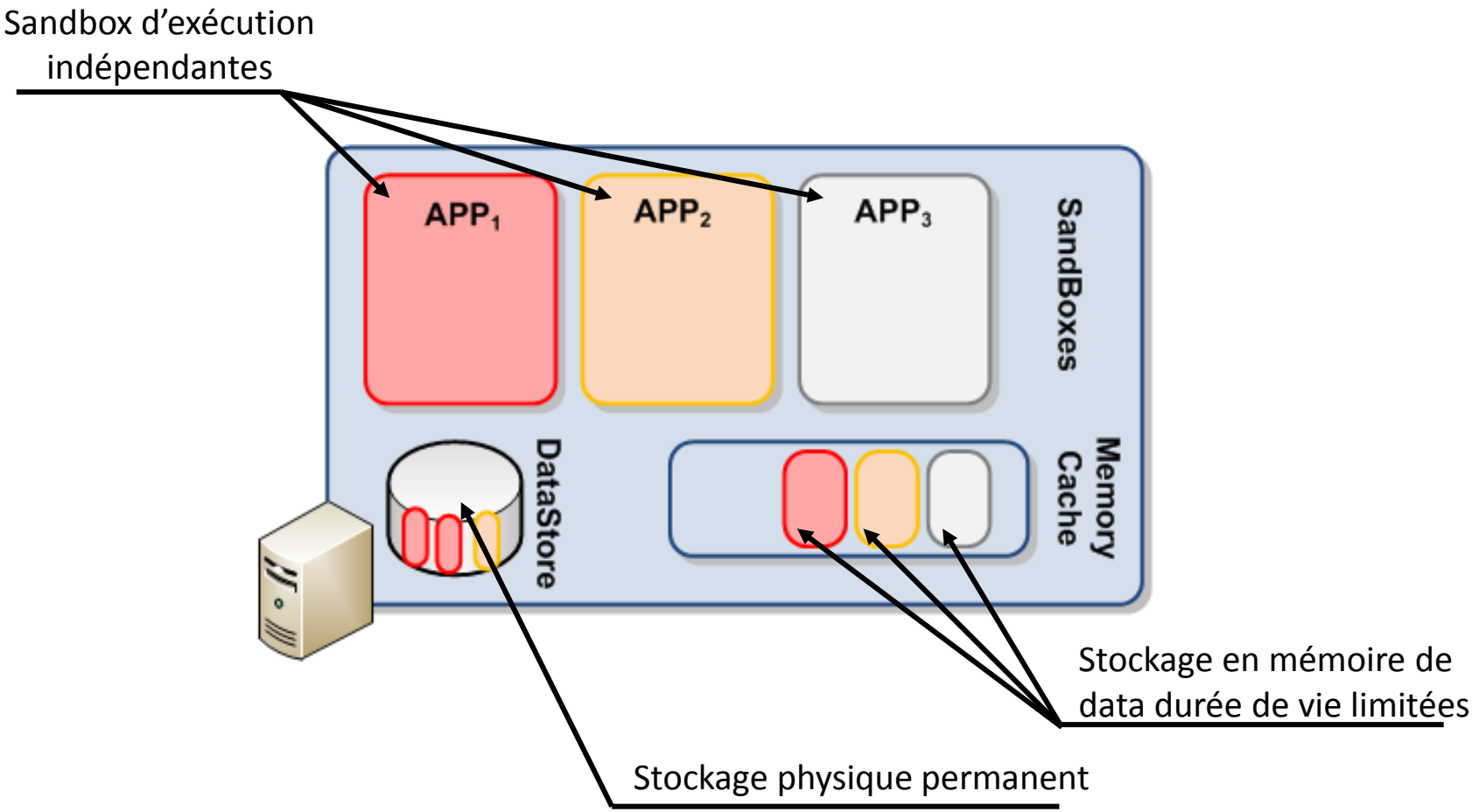


From http://blogs.zdnet.com/Hinchcliffe



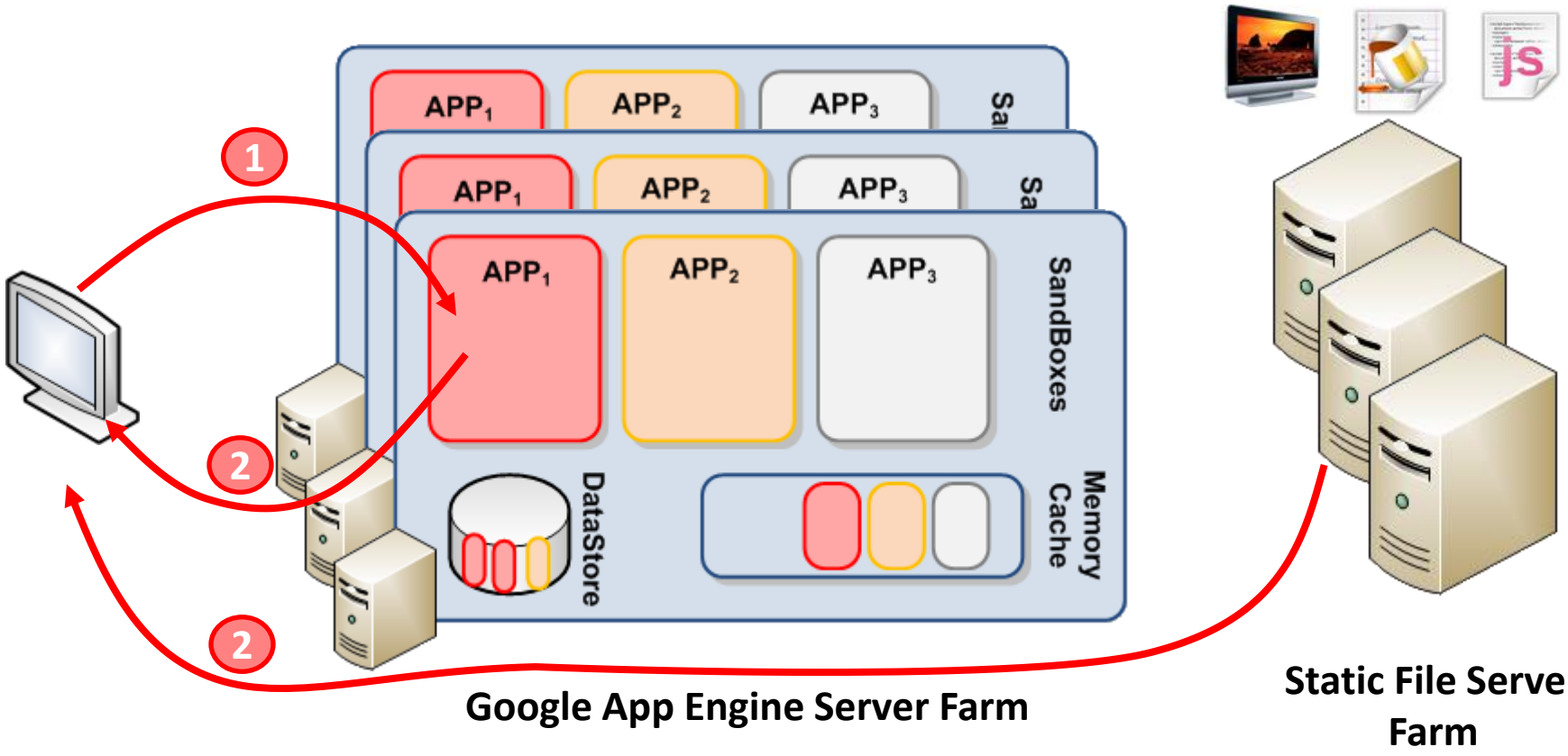
Google App Engine

- **Exemple d'usage**



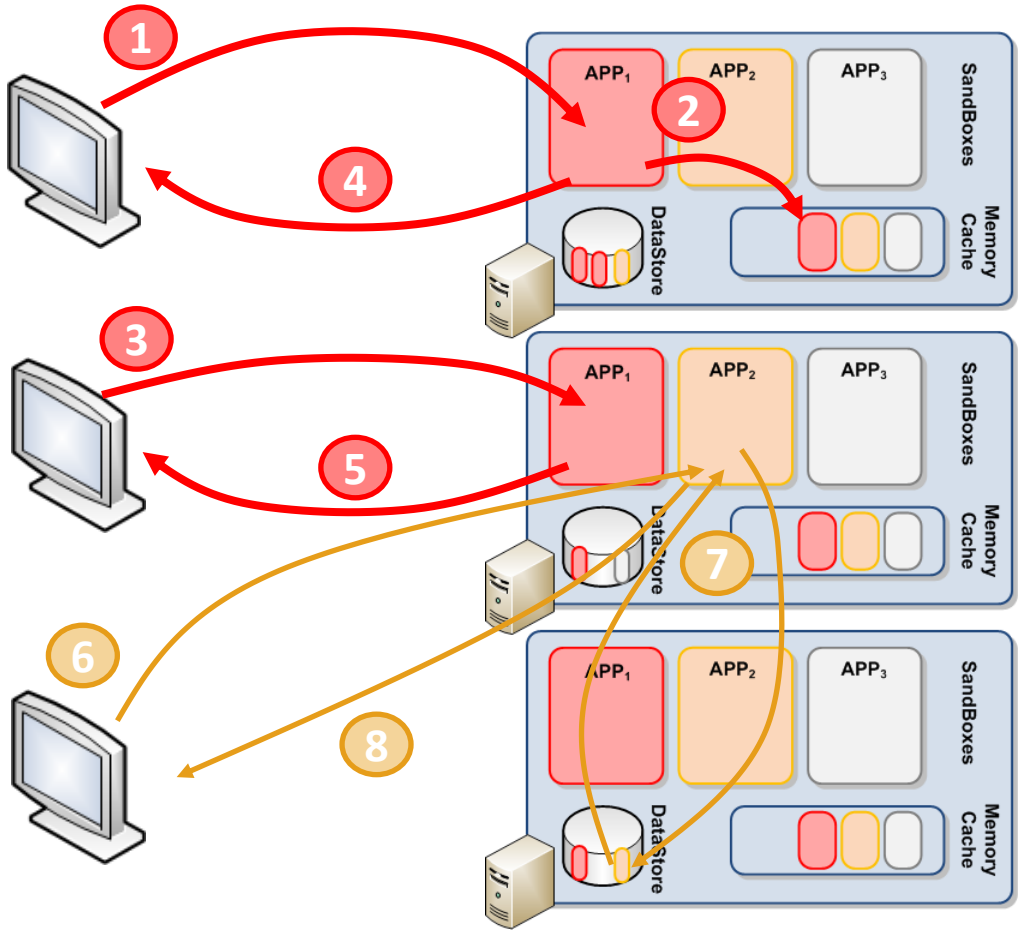
Google App Engine

- **Exemple d'usage**

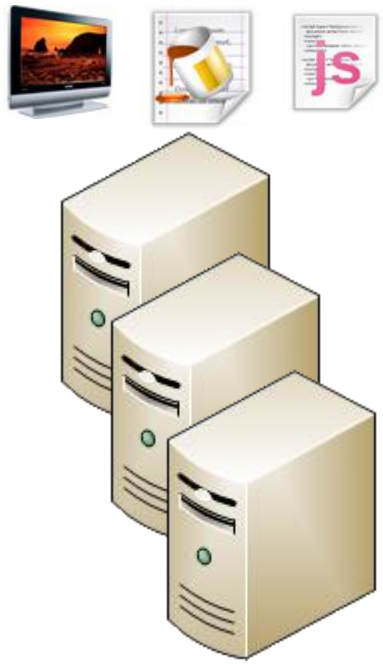


Google App Engine

- **Exemple d'usage**



Google App Engine Server Farm



Static File Server Farm

Google App Engine

- **Les services Goole App Engine**
  - Datastore
    - Persistance des données
  - Transaction
    - gestion de la concurrence des modifications dans le DataStore
  - Memory Cache
    - Persistance temporaire
  - URL Fetch Service
    - Redirection de requete http
  - Mail Service
    - Envoi et réception d'email
  - Channel
    - Web socket pour Google App Engine
  - Support XMPP
    - Messagerie instantanée (Google talk)
  - Image Processing service
    - Service de transformation d'images

Google App Engine

- **Quotas**

	Free	Paid	Premier
Price		$9/app/month	$500/account/month
Dynamic scaling	✓	✓	✓
Java Runtime	✓	✓	✓
Python Runtime	✓	✓	✓
Go Runtime	✓	✓	✓
Usage based pricing		✓	✓
Infinitely scalable		✓	✓
SLA		✓	✓
Operational support			✓
Tools			
Google Plugin for Eclipse	✓	✓	✓
Code upload/download	✓	✓	✓
Graph History	✓	✓	✓
Request Logs	✓	✓	✓
Developer Access Control	✓	✓	✓

- **Quotas**

	Free quota per app per day	Pricing if you exceed your free quota
Hosting	Free quota per app per day	Price
On-demand Frontend Instances	28 free instance hours	$0.08 / hour
Reserved Frontend Instances		$0.05 / hour
High Replication Datastore	1G	$0.24 / G / month
Outgoing Bandwidth	1G	$0.12 / G
Incoming Bandwidth	1G	Free

- **Quotas**

APIs		
Datastore API	50k free read/write/small	$0.10/100k write ops $0.07/100k read ops $0.01/100k small ops
Blobstore API	5G	$0.13 / G / month
Email API	100 recipients	$0.01 / 100 recipients
XMPP API	10k stanzas	$0.10 / 100k stanza
Channel API	100 channels opened	$0.01 / 100 channels opened
Image Manipulation API	✔	✔
Memcache API	✔	✔
Users API	✔	✔
Task Queue	✔	✔
Files API	✔	✔
URL Fetch API	✔	✔
Cron	✔	✔
SNI SSL Certificates	No free quota	$9.00 / sets of five SNI certificate slots per month
SSL Virtual IPs	No free quota	$39.00 / Virtual IP per month
Pagespeed Bandwidth	No free quota	$0.39 / G

- **Quotas**

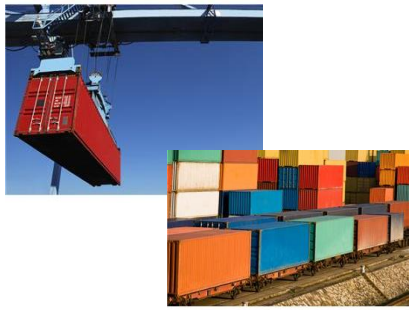
Service	Quota gratuit / jour	Maximum du quota payant / jour
Nombre de requêtes	1 300 000	43 000 000
Bande passante entrante	1 GB	1 046 GB
Bande passante sortante	1 GB	1 046 GB
Temps CPU	6.5 heures	1 729 heures
Nombre d'appels à la base de données	10 000 000	140 000 000
Taille des données	1 GB	Pas de maximum

Mise en oeuvre

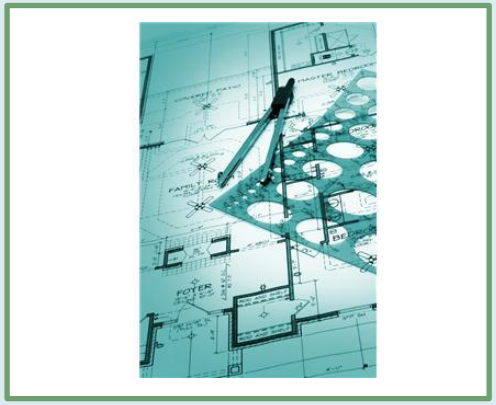
- **Sandbox**
  - Définition

    Isoler l'environnement d'execution de chaque App du système sous-jacent et des autres Apps.

  - Conséquences
    - Une App ne peut pas déclencher/démarrer de nouveaux processus ou threads (utilisation des TaskQueues)
    - Une App ne peut pas établir des connexions réseaux arbitraires (utilisation de URL Fetch/Mail)
    - Une App peut simplement lire les fichiers systèmes, lire son propre code et ces ressources (utilisation du Datastore pour stocker les données)
    - Une App ne sait rien à propos des autres applications ou processus démarré sur le même serveur.



- **Environnement de Développement**
  - Type
    - JAVA
    - PYTHON
  - SDK App Engine
    - Librairie et boite à outils de développement
    - Application de serveur web pour test
      - → *Reproduire les services APP Engine sur votre machine*
    - Outil de transfert de votre application vers APP Engine
    - Dev JAVA- Plugin Google
  - Console d'administration
    - Management du datastore
    - Management de l'application (version, activation)
    - Visualisation des instances de votre application
    - Configuration des services



Google App Engine

- **Environnement de Développement**

Total number of instances	Average QPS*	Average Latency*	Average Memory
10 total (10 Resident)	0.000	Unknown ms	7.0 MBytes

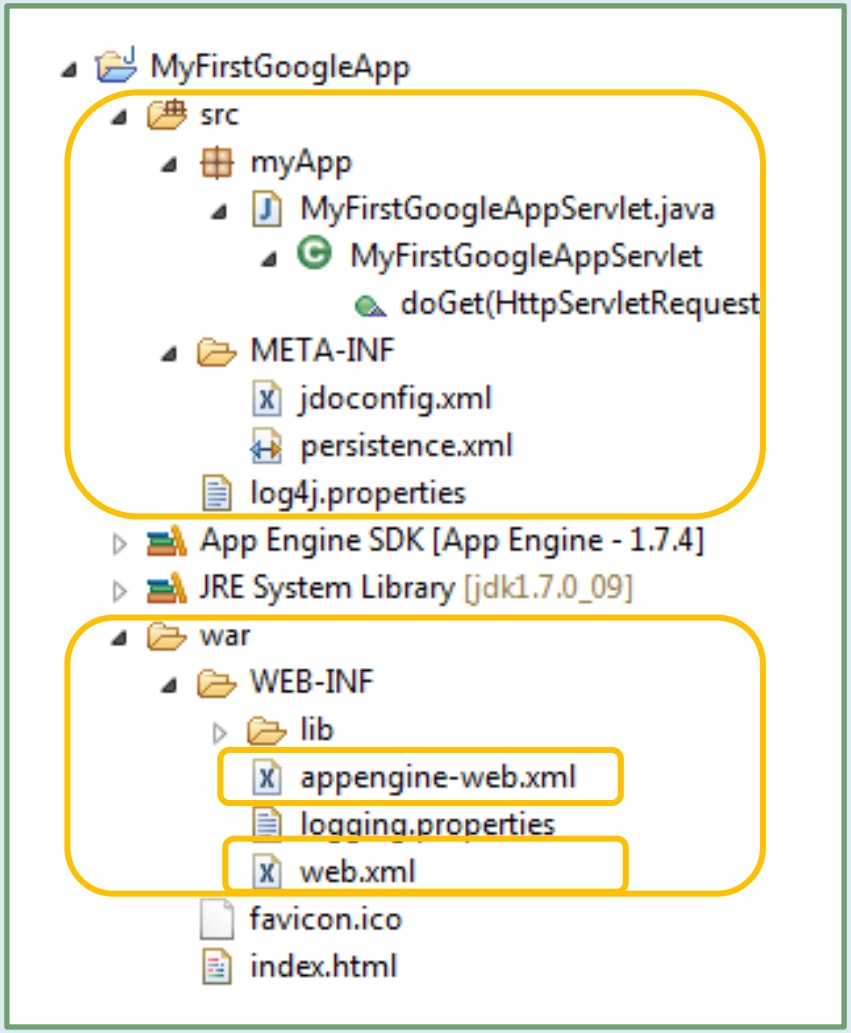
Instances

QPS*	Latency*	Requests	Errors	Age	Memory	Availability	Shutdown
0.000	0.0 ms	1	0	0:30:23	7.0 MBytes	Resident	Shutdown
0.000	0.0 ms	1	0	0:30:22	6.9 MBytes	Resident	Shutdown
0.000	0.0 ms	1	0	0:30:21	6.9 MBytes	Resident	Shutdown
0.000	0.0 ms	1	0	0:30:20	7.0 MBytes	Resident	Shutdown
0.000	0.0 ms	1	0	0:38:52	6.9 MBytes	Resident	Shutdown
0.000	0.0 ms	1	0	0:38:51	6.9 MBytes	Resident	Shutdown
0.000	0.0 ms	1	0	0:38:49	6.9 MBytes	Resident	Shutdown
0.000	0.0 ms	1	0	0:38:36	7.0 MBytes	Resident	Shutdown
0.000	0.0 ms	1	0	0:30:34	6.9 MBytes	Resident	Shutdown
0.000	0.0 ms	1	0	0:30:33	7.0 MBytes	Resident	Shutdown

* QPS and latency values are an average over the last minute.

Google App Engine

- **Anatomie d'un programme Google App (Eclipse Google plugin)**



- src/myApp → fichier application métier (Servlet and other)
- src/META-INF
  - Jdoconfige.xml
    Configuration du service JDO
  - persistence.xml
    Configuration du service JPA
- War
  - WEB-INF
    - → *Appengine-web.xml (Configuration Web App (id,version) definitions des fichiers statiques*
    - → *web.xml (management des URL)*

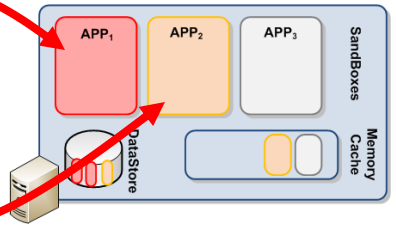
- **Anatomie d'un programme Google App : Nom de domaine**

Nom de domaine

Nom de domaine gratuit fournir par Google

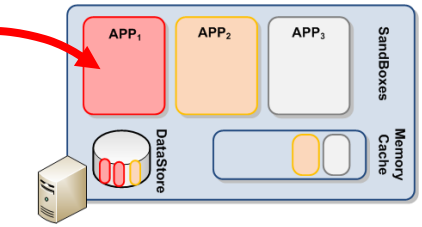
Dfn dans appengine-web.xml → app-id1.appspot.com

app-id2.appspot.com



Sous-domaine

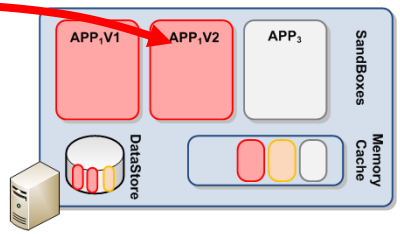
subdomain app-id1.appspot.com



Version différente

version-id lastest.app-id1.appspot.com

Dfn dans appengine-web.xml



- **Anatomie d'un programme Google App :**
  - Contraintes
    - Création de theads interdits
      - → *Pas de java.lang.Thread/ java.lang.ThreadGroup*
    - Possibilité d'interagir avec le thread courant (Thread.currentThread().dumpStack())
    - Interaction avec les fichiers ressources en lecture uniquement (java.io.Reader ok, java.io.Write ko)
    - Récupération des fichiers ressources directement dans le code Class.getResources(), ServletContext.getResources()

- **A vous de Jouer !**

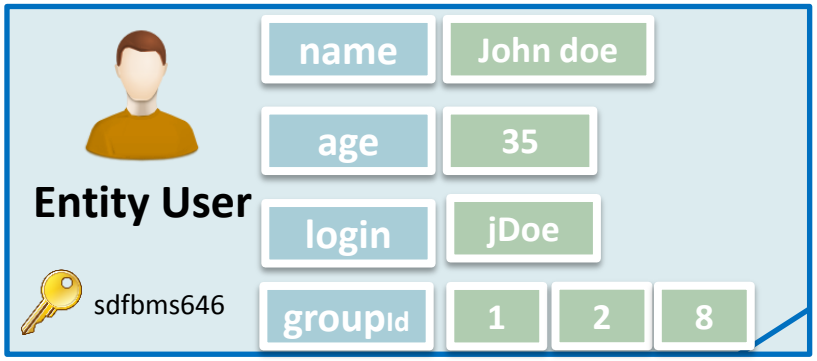
- Configuration de mon environnement
- Ma première Google App

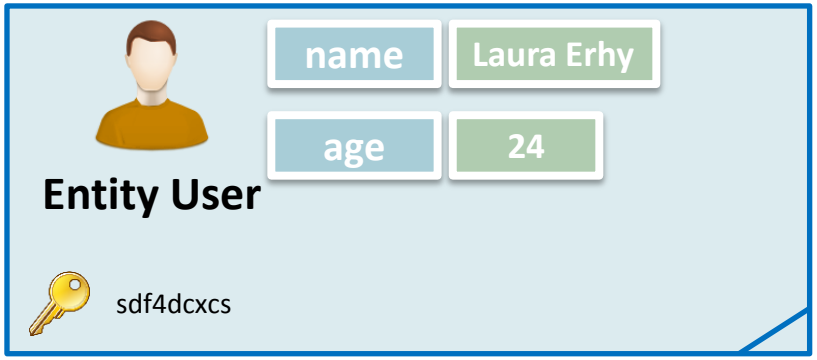
https://developers.google.com/appengine/docs/java/tools/eclipse

Service DataStore

- **DataStore**
  - Base de données répartie
    - Modèle différent des base de données relationnelles
    - Notion d'*Entity* (e.g ligne d'enregistrement)
    - Notion de *Property* (colonne d'une table de base de données)
    - 1 *Entity* peut avoir plusieurs *Properties*
    - 1 *Properties* peut avoir 1-n valeurs
    - 2 *Entity* de même types peuvent avoir des *Properties* différentes
    - 1 *Entity* possède une clé unique (une clé n'est pas un *Property*)
    - La clé d'une *Entité* est définie à sa création et ne peut être modifée

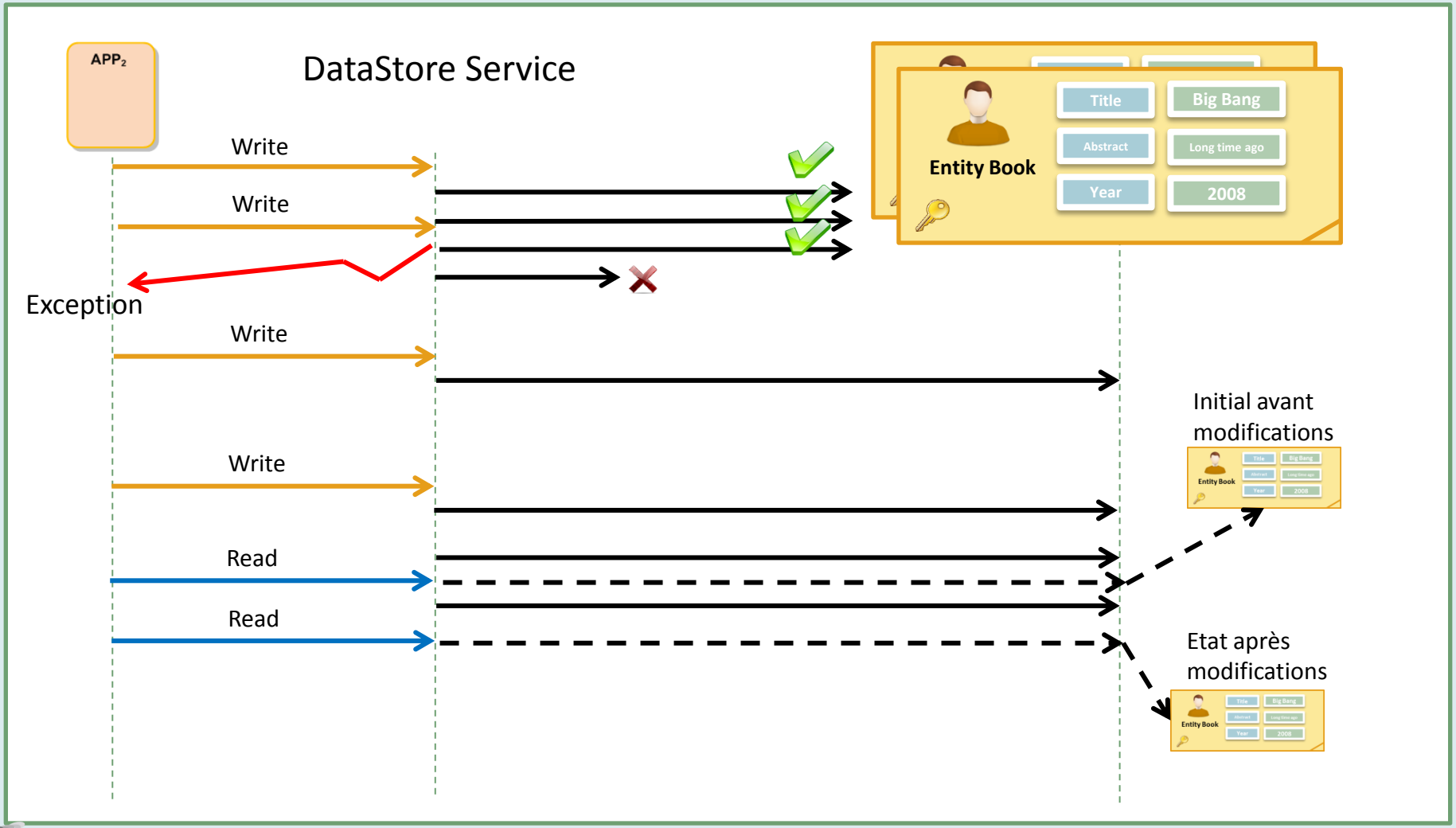
- **DataStore Entity et Property**





- **Transaction**
  - Transaction atomique: Modification d'une Entity
    - Attente que l'ensemble des modifications en cours ont été qualifiées soit success soit failed
    - Optimistic Concurrency Control
      - → *Exception si modification en cours reprise à gérer niveau code*
    - Modification bloquante
  - La lecture d'*Entity* est toujours possible (sur le dernier état stable)
  - Modification de plusieurs Entity dans une même requête possibles
    - → *Besoin de rassembler les Entities susceptibles d'être modifiées au sein d'Entity Group*
  - Création de requête → création automatique d'index (crée au déploiement de la Google App)
    - → *Toute modification d'Entity/property entraine une mise à jour de ces indexes*

- **Transaction**



- **Utilisation du DataStore**
  - Requête
    - Optimisation pour grand volume (création de table d' index de réponse)
    - Contrainte de requêtes
      - → *Pas de jointure*
      - → *Filtrage tri exige que la propriété existe*
      - → *Filtre d'inégalité (<, <=, >=, >, !=) uniquement autorisés sur une seule propriété*
      - → *Et d'autres…..*

https://developers.google.com/appengine/docs/java/datastore/queries?hl=fr

- **Utilisation du DataStore (1/2)**

Appel du service DataStore

Création d'une entité de type « Employee », clé générée automatiquement

Ajout des propriétés à notre entité

Ajout de l'entité au datastore

Récupération d'une entité à partir de ça clé

```
import java.util.Date;
import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
import com.google.appengine.api.datastore.Entity;

// ...
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();

Entity employee = new Entity("Employee");
employee.setProperty("firstName", "Antonio");
employee.setProperty("lastName", "Salieri");
Date hireDate = new Date();
employee.setProperty("hireDate", hireDate);
employee.setProperty("attendedHrTraining", true);

datastore.put(employee);

Key employeeKey = employee.getKey();
Entity employee = datastore.get(employeeKey);
```

- **Utilisation du DataStore (2/2)**

Création d'une requête sur les entités de type « Person » avec 2 Filtres

Préparation de la requête →création auto d'une table d'index de réponse au sein du datastore

Parcours des Entité répondant aux critères de notre requête

```
// Recupération du service Datastore
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();

// Utilisation Query afin de rassembler les éléments a appeler/filter
Query q = new Query("Person");
Filter lastNameFilter =  new FilterPredicate("lastName",
                                    FilterOperator.EQUAL, lastNameParam);
Filter heightMaxFilter =  new FilterPredicate("height",
                FilterOperator.LESS_THAN_OR_EQUAL,  maxHeightParam);
Filter heightRangeFilter =  CompositeFilterOperator.and(lastNameFilter,
 heightMaxFilter);
q.setFilter(heightRangeFilter);
// Récupération du résultat de la requête à l'aide de PreparedQuery
PreparedQuery pq = datastore.prepare(q);

for (Entity result : pq.asIterable()) {
  String firstName = (String) result.getProperty("firstName");
  String lastName = (String) result.getProperty("lastName");
  Long height = (Long) result.getProperty("height");
}
```

- **Utilisation du DataStore: Types des propriétés 1/2**

Type de valeur	Type Java	Ordre de tri	Remarques
Booléen	boolean ou java.lang.Boolean	false < true	
Chaîne d'octets courte	com.google.appengine.api.datastore.ShortBlob	Ordre des octets	Jusqu'à 500 octets. Une valeur supérieure à 500 octets génère une exception JDOFatalUserException.
Chaîne d'octets longue	com.google.appengine.api.datastore.Blob	*n/a*	Jusqu'à 1 méga-octet, non indexé
Catégorie	com.google.appengine.api.datastore.Category	Unicode	
Date et heure	java.util.Date	Chronologique	
Adresse e-mail	com.google.appengine.api.datastore.Email	Unicode	
Nombre à virgule flottante	float, java.lang.Float, double,java.lang.Double	Numérique	Double précision 64 bits, IEEE 754
Point géographique	com.google.appengine.api.datastore.GeoPt	Par latitude, puis longitude	
Utilisateur de Google Accounts	com.google.appengine.api.users.User	Adresse e-mail dans l'ordre Unicode	
Entier	short, java.lang.Short, int,java.lang.Integer, long, java.lang.Long	Numérique	Stocké en tant qu'entier long, puis converti dans le type du champ. Les valeurs hors plage provoquent un dépassement.

- **Utilisation du DataStore: Types des propriétés 2/2**

Type de valeur	Type Java	Ordre de tri	Remarques
Clé, blobstore	com.google.appengine.api.blobstore.BlobKey	Ordre des octets	
Clé, magasin de données	com.google.appengine.api.datastore.Key, ou l'objet référencé (en tant qu'enfant)	Par éléments de chemin d'accès (kind, identifiant ou nom...)	
Lien	com.google.appengine.api.datastore.Link	Unicode	
Identificateur de messagerie	com.google.appengine.api.datastore.IMHandle	Unicode	
Null	null	*n/a*	
Adresse postale	com.google.appengine.api.datastore.PostalAddress	Unicode	
Note	com.google.appengine.api.datastore.Rating	Numérique	
Numéro de téléphone	com.google.appengine.api.datastore.PhoneNumber	Unicode	
Chaîne de texte courte	java.lang.String	Unicode	Jusqu'à 500 caractères Unicode. Une valeur supérieure à 500 caractères génère une exception JDOFatalUserException.
Chaîne de texte longue	com.google.appengine.api.datastore.Text	*n/a*	Jusqu'à 1 méga-octet, non indexé

- **A vous de Jouer !**

❑ 1 Page Formulaire d'enregistrement d'utilisateur (nom, prenom, age, login, pwd)

❑ 1 Page Login (login + PWD), une fois loggué afficher les propriétés de l'utilisateur

Service AppCache

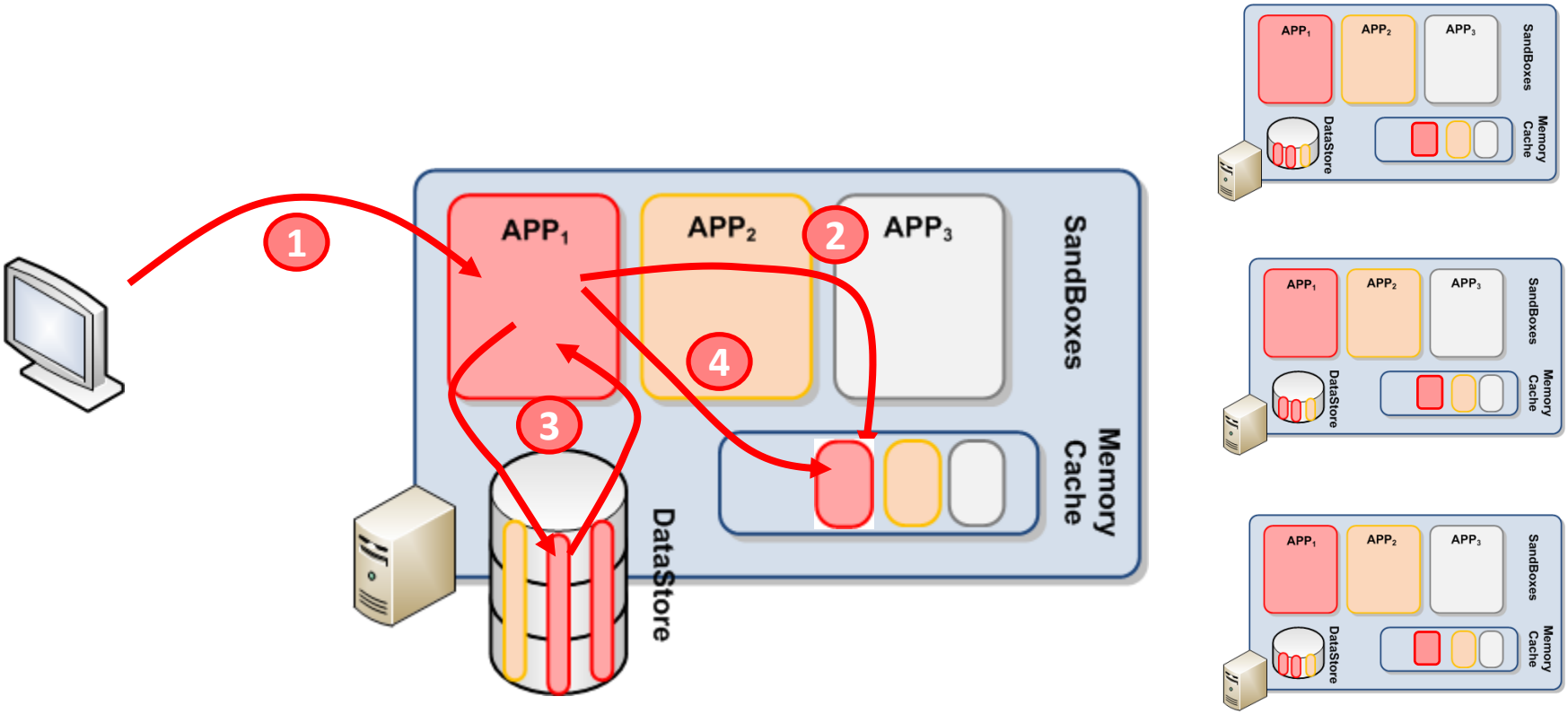
- **Memory Cache**
  - Stockage temporaire de clé/valeurs
  - Rapide car stocké en mémoire
  - Distribué sur les différentes instances
  - Pas de persistance, si arrêt du serveur

    → perte de données

  - Bonne pratique:
    - Vérifier si valeur est disponible dans Memory Cache
    - Si non présent recherche de la valeur désirée (calcul ou requête DataStore)
    - Mise à jour du cache avec la nouvelle valeur

Google App Engine

- **Memory Cache**



- **Utilisation de l'APP Cache**
  - 2 modes d'utilisation
    - JCache Interface
    - → API de haut niveau pour l'utilisation de l'App Cache
    - Low Level API
    - → Fournir des services de stockage à la mémoire synchrone et asynchrone, plus riche que Jcache
  - Propriétés
    - Stockage rapide des données
    - Utilisation pour stockage d'informations temporaires
    - Spécification de la durée d'expiration des données possible
    - 1Mo de stockage maximum en APP Cache
    - Utilisation des librairies ***net.sf.jsr107cache***

- **Utilisation de l'App Cache: JCache**

Définition de propriétés (expiration et politique)

Récupération/Création du Cache

Ajout d'une valeur au cache

Récupération d'une valeur du cache

```
Cache cache=null;
Map props = new HashMap();
props.put(GCacheFactory.EXPIRATION_DELTA, 3600);
props.put(MemcacheService.SetPolicy.ADD_ONLY_IF_NOT_PRESENT, true);
try {
  // Récupération du Cache
  CacheFactory cacheFactory = CacheManager.getInstance().getCacheFactory();
  // création/récupération du cache suivant des propriétés spécifiques
  cache = cacheFactory.createCache(props);

  // Si aucune propriété n'est spécifiée,
  //créer/récupérer un cache comme ci-dessous
  //cache = cacheFactory.createCache(Collections.emptyMap());
 } catch (CacheException e) {
    // Traitement en cas d'erreur sur la récupération/configuration du cache
 }
String key="objKey1";      // Définir la clé de la valeur à stocker
String value="myValue1"    // Définir l'objet à stocker

// Mise en cache de l'objet à l'aide d'une clé
cache.put(key, value);
// Récupération de l'objet stocké
value = (String)cache.get(key);
```

- **Utilisation de l'App Cache: Low Level API**

Récupération/Création d'un Cache synchrone

Récupération/mise à jour synchrone d'une valeur

Récupération/Création d'un Cache asynchrone

Récupération mise à jour synchrone d'une valeur

```
String key="objKey1";      // Définir la clé de la valeur à stocker
String value="myValue1"    // Définir l'objet à stocker
// Méthode de cache synchrone
MemcacheService syncCache = MemcacheServiceFactory.getMemcacheService();
syncCache.setErrorHandler(ErrorHandlers.getConsistentLogAndContinue(Level.INFO));
value = (String ) syncCache.get(key); // Lecture depuis le cache
if (value == null) {
  // récupération de la valeur et exécution de son code ….
  syncCache.put(key, value); // Mise à jour du cache
}
// Méthode de cache asynchrone
AsyncMemcacheService asyncCache =
        MemcacheServiceFactory.getAsyncMemcacheService();
asyncCache.setErrorHandler(ErrorHandlers.getConsistentLogAndContinue(Level.INFO));
Future<Object> futureValue = asyncCache.get(key); // Lecture depuis le cache
// ... Executation de tache en parallèle de la recherche dans le cache
value = (String) futureValue.get();
if (value == null) {
  // récupération de la valeur et exécution de son code ….
  // Mise à jour du cache asynchrone
  // Retourne Future<Void> qui peut être utilisé pour bloquer jusqu'à ce la fin de l'action
  asyncCache.put(key, value);
}
```

- **A vous de Jouer !**

- 1 Page Formulaire d'enregistrement d'utilisateur (nom, prenom, age, login, pwd)
- 1 Page Login (login + PWD), une fois loggué afficher les propriétés de l'utilisateur
- Ajouter un message de bienvenue dans la page login à récupérer une première fois dans le dataStore (ajout dans l'app cache) et à récupérer depuis l'app cache pour les autres fois

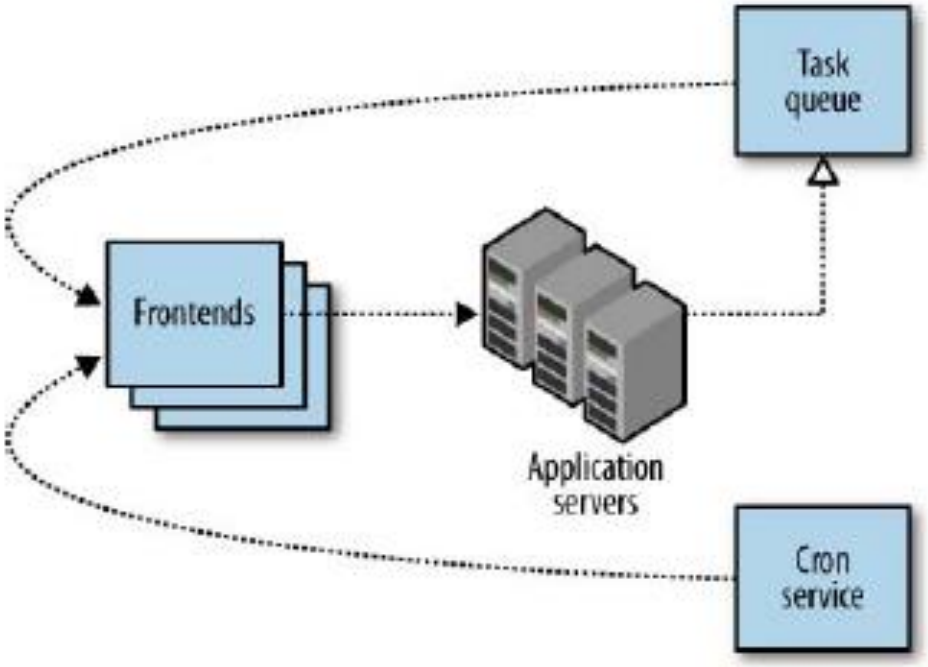
Service Task Queue - Cron

- **Tâches planifiées et files d'attente de tâches**
  - Définition

    Effectuer des tâches en dehors des réponses aux requêtes WEB.

  - Propriétés
    - Planification possible (CRON like)
    - Tentative de réexecution automatiques en cas d'echec (datastore)
    - Les tâches planifiées ne support pas de multiple tentatives
    - Une tâche spécifie une URL à appeler ainsi que des paramètres
    - Deadline d'exécution de 30s pour chaque tache
    - Possibilité de définir un « rate queue » spécifiant la rapidité à laquelle les tâches vont consommer les ressources

- **Tâches planifiées et files d'attente de tâches**



- **Utilisation de Task Queue**
  - Configuration des task queue dans un fichier WEB-INF/queue.xml
  - Si pas de fichier queue.xml une queue par default existe
  - Utilisation de point d'ancrage (Anchror)
    - Exécution des tâches sur une URL données
    - Passage de paramètres dans l'URL ou à part (POST)
    - Configuration supplémentaires possible dans l'entête HTTP (Nom de la queue, Nom de la tâche, nombre de tentative)

- **Utilisation de Task Queue: Configuration queue.xml**

```
<queue-entries>
 <queue>
  <name>default</name>
  <rate>1/s</rate>
 </queue>
 <queue>
  <name>mail-queue</name>
  <rate>2000/d</rate>
  <bucket-size>10</bucket-size>
 </queue>
 <queue>
  <name>background-processing</name>
  <rate>5/s</rate>
 </queue>
</queue-entries>
```

Nom des task queues

Fréquences d'exécution

Taille du sceau de l'algorithme saut/jeton
(optimisation du tirage des tâches des files d'attentes)

- **Utilisation de Task Queue**

Récupération de la task queue par défaut

Création d'une tâche avec des paramètres

Définition des propriétés de tâche

Tentative de suppression de la tache «task»

Suppression des tâches de la queue

```
String key="name";
String value="idoe";
Queue queue = QueueFactory.getDefaultQueue();
// Ajout d'une tache simple
TaskOptions task=TaskOptions.Builder.withUrl("/worker").param(key, value);
 queue.add(task);
// Ajout d'une tache simple avec des paramètres de configuration
Map<String, String> headers=new HashMap<String, String>();
headers.put("X-AppEngine-TaskName","task2");
headers.put("X-AppEngine-TaskRetryCount","4");
TaskOptions task2=TaskOptions.Builder.withUrl("/worker2").headers(headers);
queue.add(task2);
// Ajout d'une tache en spécifiant la méthode utilisée
TaskOptions
task3=TaskOptions.Builder.withUrl("/worker?a=b&c=d").method(Method.GET);
queue.add(task3);
//...
queue.deleteTask("task");
//...
queue.purge();
```

- **Utilisation de Panification des tâches: configuration cron.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<cronentries>
  <cron>
    <url>/recache</url>
    <description>Repopulate the cache every 2
minutes</description>
    <schedule>every 2 minutes</schedule>
  </cron>
  <cron>
    <url>/weeklyreport</url>
    <description>Mail out a weekly
report</description>
    <schedule>every monday 08:30</schedule>
    <timezone>America/New_York</timezone>
  </cron>
</cronentries>
```

Url de la tâche

Fréquence d'utilisation

Time Zone à utiliser

- **Utilisation de task Queue: sécurisation des URL**

web.xml

```
<security-constraint>
    <web-resource-collection>
      <url-pattern>/tasks/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>admin</role-name>
    </auth-constraint>
 </security-constraint>
```

Définition d'un domaine

Définition des autorisations allouées au domaine

- **A vous de Jouer !**

❑ Réutiliser vos formulaires d'enregistrement en déléguant la sauvegarde de vos utilisateurs à une tâches

❑ Utiliser une queue que vous aurez défini dans le fichier queue.xml avec un fréquence de 1 tâche par seconde

- **A vous de Jouer !**

❑ Utiliser une tache planifiée CRON permettant de supprimer tous les utilisateurs toutes les minutes

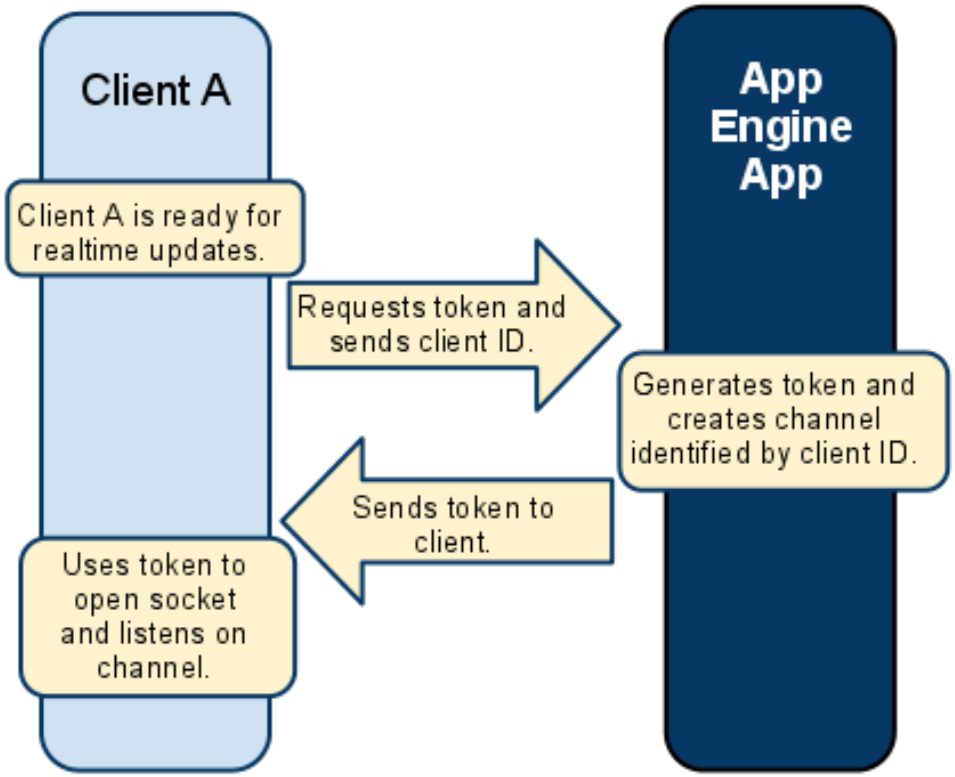
Service Channel

- **Channel Service**
  - Définition

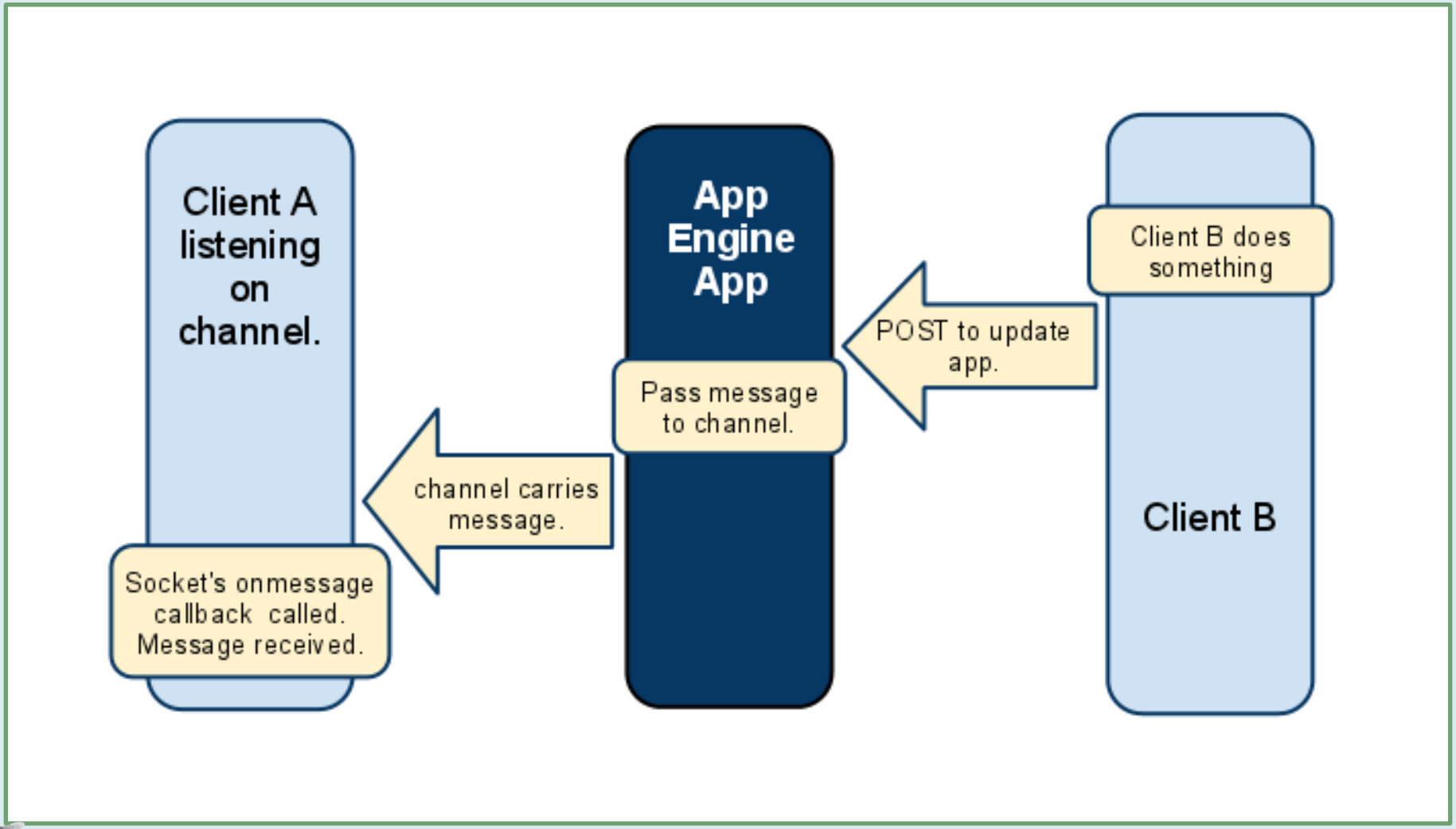
    Service permettant de créer des connections persistances entre les applications et les serveurs Google, permettant aux applications d'envoyer des messages et d'en recevoir des serveur (push).

  - Propriétés
    - Crée une communication « socket like » entre server et client javascript
    - Fourni une API javascript permettant de communiquer avec les channels
    - Les serveurs reçoivent des mises à jour des données via HTTP
    - Les clients javascript reçoivent des mises à jour des serveurs.

- **Channel Service**



- **Channel Service**



- **Channel Service : html side**

```
<!DOCTYPE html>
<html>
  <head>
      <script src='/_ah/channel/jsapi'></script>
      <script src="js/jquery-1.11.1.min.js"></script>
  </head>
  <body onload="">
    <h1>Channel GAE Sample</h1>
    <input type="text" id="loginId">
    <button onclick="getToken()">OpenChannel</button>
    <h2>Talk</h2>
    <input type="text" id="txtId">
    <button onclick="sendMessage()"> send</button>
    <p id="historyId"></p>

    <script src='js/code.js'></script>
  </body>
</html>
```

- **Channel Service : javascript side (1/2)**

```
var token = '{{ token }}';
   getToken=function(){
   $.post(
       "/token",
       { login: $("#loginId").val(), CMD:"GET_TOKEN"},
         function(m){
         //retreive the token generated associated with the created Channel
           token=m.token;
           openChannel();
         } );
       };
     openChannel = function() {
    //retrieve the communication channel associated with the given token
       var channel = new goog.appengine.Channel(token);
       //define function handler associated with the channel
       var handler = {
         'onopen': onOpened,
         'onmessage': onMessage,
         'onerror': function(err)
                     {   alert("Error Occurred"+err.data);},
         'onclose': function() {alert("channel Closed"); }
       };
       //open the channel with associated function handler
       var socket = channel.open(handler);
       socket.onopen = onOpened;
       socket.onmessage = onMessage;
     }
```

- **Channel Service : javascript side (2/2)**

```
…
    onOpened = function() {
        alert("Opened");
      };

      onMessage = function(m) {
          $("#historyId").append(m.data);

      }
    sendMessage=function(){
        var message={};
        message.token=token;
        message.CMD="SEND_MSG";
        message.message=$("#txtId").val();
        //Send message to the servlet at the URL /token with message to send
        $.post(
            "/token",
            message,
            function(m){

              } );
        //remove text from input box
         $("#txtId").val("");
        };
```

- **Channel Service : Server Servlet side (1/3)**

```
public class TokenChannelGeneratorServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;
        private ConcurrentHashMap<String, String> loginTokenMap;

        public TokenChannelGeneratorServlet() {
                loginTokenMap = new ConcurrentHashMap<String, String>();
        }
        @Override
        protected void doPost(HttpServletRequest req, HttpServletResponse resp)
                        throws ServletException, IOException {
                String currentOrder = req.getParameter("CMD");
                if ("GET_TOKEN".equals(currentOrder)) {
                        String currentLogin = req.getParameter("login");
                        registerChannelSendToken(currentLogin, resp);
                } else if ("SEND_MSG".equals(currentOrder)) {
                        String currentMessage = req.getParameter("message");
                        String token = req.getParameter("token");
                        sendMessageToAllChannel(currentMessage, token);
                }

        }
```

- **Channel Service : Server Servlet side (2/3)**

```
private void registerChannelSendToken(String currentLogin,
                                      HttpServletResponse resp) throws IOException {
                String token = "";
                if (!loginTokenMap.containsKey(currentLogin)) {
                                      String uuid = UUID.randomUUID().toString();
                                      // Call the Channel service
                                      ChannelService channelService = ChannelServiceFactory     .getChannelService();
                                      // Generate the Channel associated with the unique identifier
                                      token = channelService.createChannel(uuid);
                                      loginTokenMap.put(currentLogin, token);
                } else {
                                      token = loginTokenMap.get(currentLogin);
                }
                resp.setContentType("application/json");
                JSONObject jsonToSend;
                try {
                                      jsonToSend = new JSONObject("{'token':" + token + "}");
                                      PrintWriter out = resp.getWriter();
                                      out.write(jsonToSend.toString());
                                      out.flush();
                } catch (JSONException e) {
                                      e.printStackTrace();
                }

        }
```

- **Channel Service : Server Servlet side (3/3)**

```
private void sendMessageToAllChannel(String currentMessage,
                                     String originToken) {
                // Get channel service
                ChannelService channelService = ChannelServiceFactory.getChannelService();
                // Browse all channel token
                for (String token : this.loginTokenMap.values()) {
                        // send a message to the current channel token
                        channelService.sendMessage(new ChannelMessage(token,
                        getLoginFromToken(originToken) + ": " + currentMessage    + "<br>"));
                }
        }

        private String getLoginFromToken(String originToken) {
                for (String s : this.loginTokenMap.keySet()) {
                        if (originToken.equals(loginTokenMap.get(s))) {
                                return s;
                        }
                }
                return "unknown";
        }

}
```

- **Channel Service : 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">
<servlet>
          <servlet-name>TokenServlet</servlet-name>
          <servlet-class>simple.TokenChannelGeneratorServlet</servlet-class>
</servlet>
<servlet-mapping>
          <servlet-name>TokenServlet</servlet-name>
          <url-pattern>/token</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>channel.html</welcome-file>
</welcome-file-list>

</web-app>
```

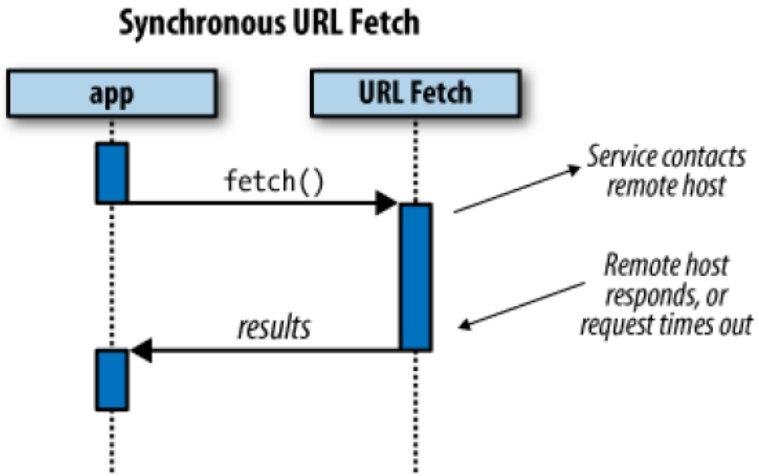
Service URL Fetch

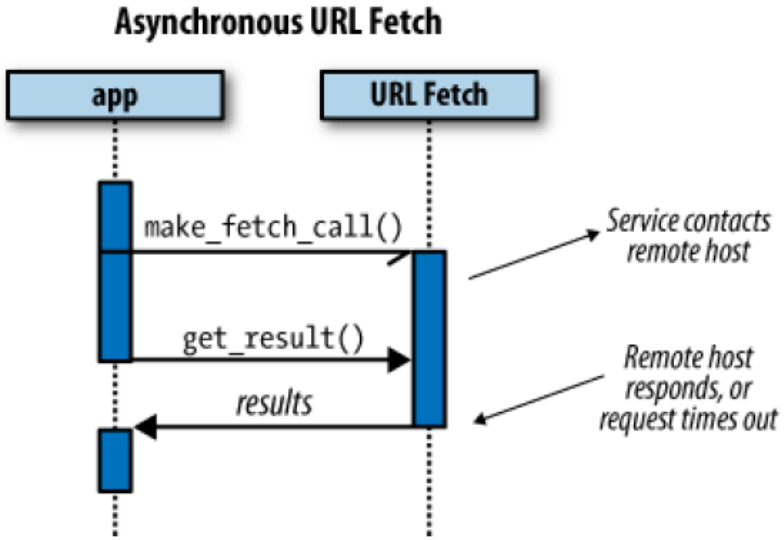
- **URL Fetch Service**
  - Définition

    Service permettant d'appeler des URL distantes extérieures à l'App.

  - Propriétés
    - Support du protocole HTTP et HTTPs (vérification du certificat du serveur distant non assuré)
    - Les autres protocoles ne sont pas supportés (e.g FTP)
    - Support des actions HTTP Get-Post-Put-Head-Delete
    - Support uniquement les connections standards TCP vers ports standards (HTTP 80,HTTPs 443)
    - Appel Synchrone et Asynchrone possible

- **URL Fetch Service**





- **Utilisation de URL Fetch**
  - Communication via URL (openStream)
  - Communication via HttpURLConnection
  - ATTENTION, URL Fetch → pas de connexion HTTP persistante

- **Utilisation de URL Fetch: URL**

Définition d'une URL, ouverture d'un flux de lecteur sur cette URL

Traitement du flux de retour

```
import java.net.MalformedURLException;
import java.net.URL;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
// …
      try {
         URL url = new URL("http://www.example.com/atom.xml");
         BufferedReader reader = new BufferedReader(new
                                 InputStreamReader(url.openStream()));
         String line;

         while ((line = reader.readLine()) != null) {
            // Traitement des données reçues
         }
         reader.close();

      } catch (MalformedURLException e) {
         // Gestion d'exceptions d'ouverture de flux
      } catch (IOException e) {
         // Gestion d'exceptions de lecture de flux
      }
```

- **Utilisation de URL Fetch: HttpURLConnection**

Définition d'une URL, ouverture d'un connexion sur l'URL, sélection de la méthode de communication

Ecriture de la « Charge utilise » de la connexion HTTP

Gestion du flux de retour

```
String message = URLEncoder.encode("my message", "UTF-8");
try {
    URL url = new URL("http://www.example.com/comment");
    HttpURLConnection connection = (HttpURLConnection) url.openConnection();
    connection.setDoOutput(true);
    connection.setRequestMethod("POST");

    OutputStreamWriter writer = new
            OutputStreamWriter(connection.getOutputStream());
    writer.write("message=" + message);
    writer.close();

    if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {
       // Connexion réussie, possibilité de récupérer le flux de retour
    } else {
       // HTTP error code.
    }
} catch (MalformedURLException e) {
   // Gestion d'exceptions d'ouverture de flux
} catch (IOException e) {
   // Gestion d'exceptions de lecture de flux
}
```

- **A vous de Jouer !**

- Créer une Servlet 1 permettant de renvoyer une message formaté (<key1>=<valeur1>; <key2>=<valeur2>)
- Créer une Servlet 2 utilisant les valeurs fournies par l'url de Servlet 1 et permettant d'afficher dans un tableau le message formaté

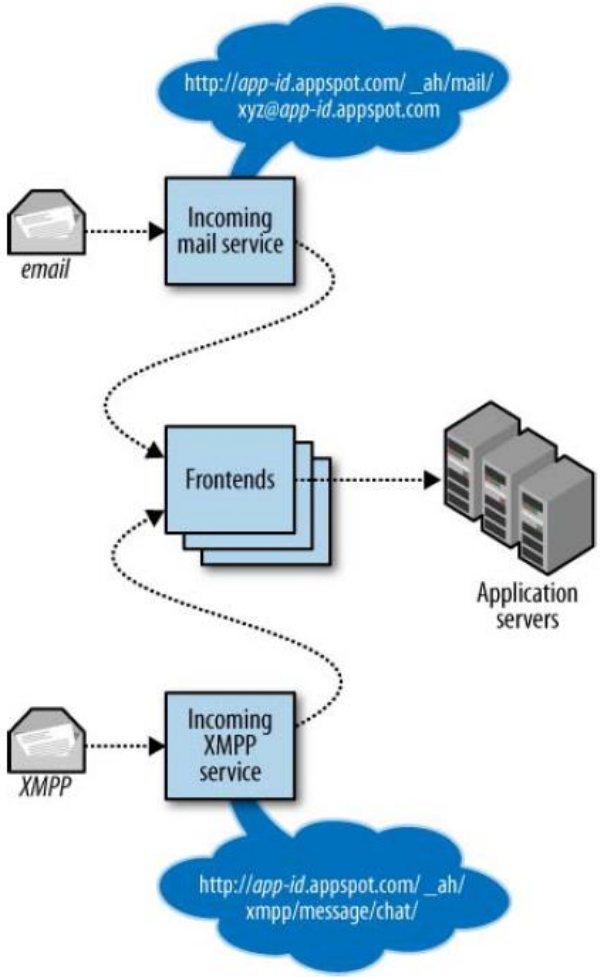
Service Email - Chat

- **Messagerie Instantanée et Mail**
  - Définition

    Délégation d'envoi/réception de message/email permettant à l'App. de communiquer avec l'extérieur

  - Propriétés
    - Les message reçus (messages/emails) sont perçus comme des POST actions par l'APP, similaire à une HTTP request
    - Communication uniquement avec le protocole XMPP pour la messagerie instantanée
    - Envoi et réception d'email avec pièces jointes possible
    - Pour communiquer en chat (XMPP) le client doit se connecter à son application de chat (gtalk)
    - Réception des email à l'adresse app-id@appspotmail.com ou anything@app-id.appspotmail.com

- **Messagerie Instantanée et Mail**



- **Utilisation simple de JavaMail: envoi d'email**

Utilisation de javax.mail classique

```
import java.io.IOException;
import java.util.Properties;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

- **Utilisation simple de JavaMail: envoi d'email**

Création/récupération d'une session eMail avec les propriétés souhaitées

Création d'un message respectant les spécifications de la session

Création du contenu et des attributs du message

Envoi du message suivant les spécification de la session

```
public class UserMailSender extends HttpServlet{
    private static final long serialVersionUID = 1L;
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                        throws ServletException, IOException {
        Properties props = new Properties();
        Session session = Session.getDefaultInstance(props, null);
        String msgBody = "YO! Welcome to the JavaMail Service provided by
                                Google App Engine !";
        try {
            Message msg = new MimeMessage(session);
            msg.setFrom(new InternetAddress("app@yahoo.fr", "Example.com
                                Admin"));
            msg.addRecipient(Message.RecipientType.TO,
                        new InternetAddress("jDoe@cpe.fr", "Mr. User"));
            msg.setSubject("Your Example.com account has been activated");
            msg.setText(msgBody);
            Transport.send(msg);
        } catch (AddressException e) {e.printStackTrace();
        } catch (MessagingException e) {e.printStackTrace();   }
    }
}
```

- **Utilisation simple de JavaMail: réception d'email (Configuration)**

appengine-web.xml

```
<inbound-services>
  <service>mail</service>
</inbound-services>
```

Ajout du support d'un service

web.xml

```
<servlet>
  <servlet-name>mailhandler</servlet-name>
  <servlet-class>MailHandlerServlet</servlet-class>
</servlet>
```

Servlet gérant les emails

```
<servlet-mapping>
  <servlet-name>mailhandler</servlet-name>
  <url-pattern>/_ah/mail/*</url-pattern>
</servlet-mapping>
```

Envoi de l'ensemble des mails reçu vers la servlet gérant les emails

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/_ah/mail/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
</security-constraint>
```

Restriction de l'accès à l'URL gérant les emails

- **Utilisation simple de JavaMail: réception d'email (Traitement)**

Création/récupération de la session

Récupération du message (reponse HTTP)

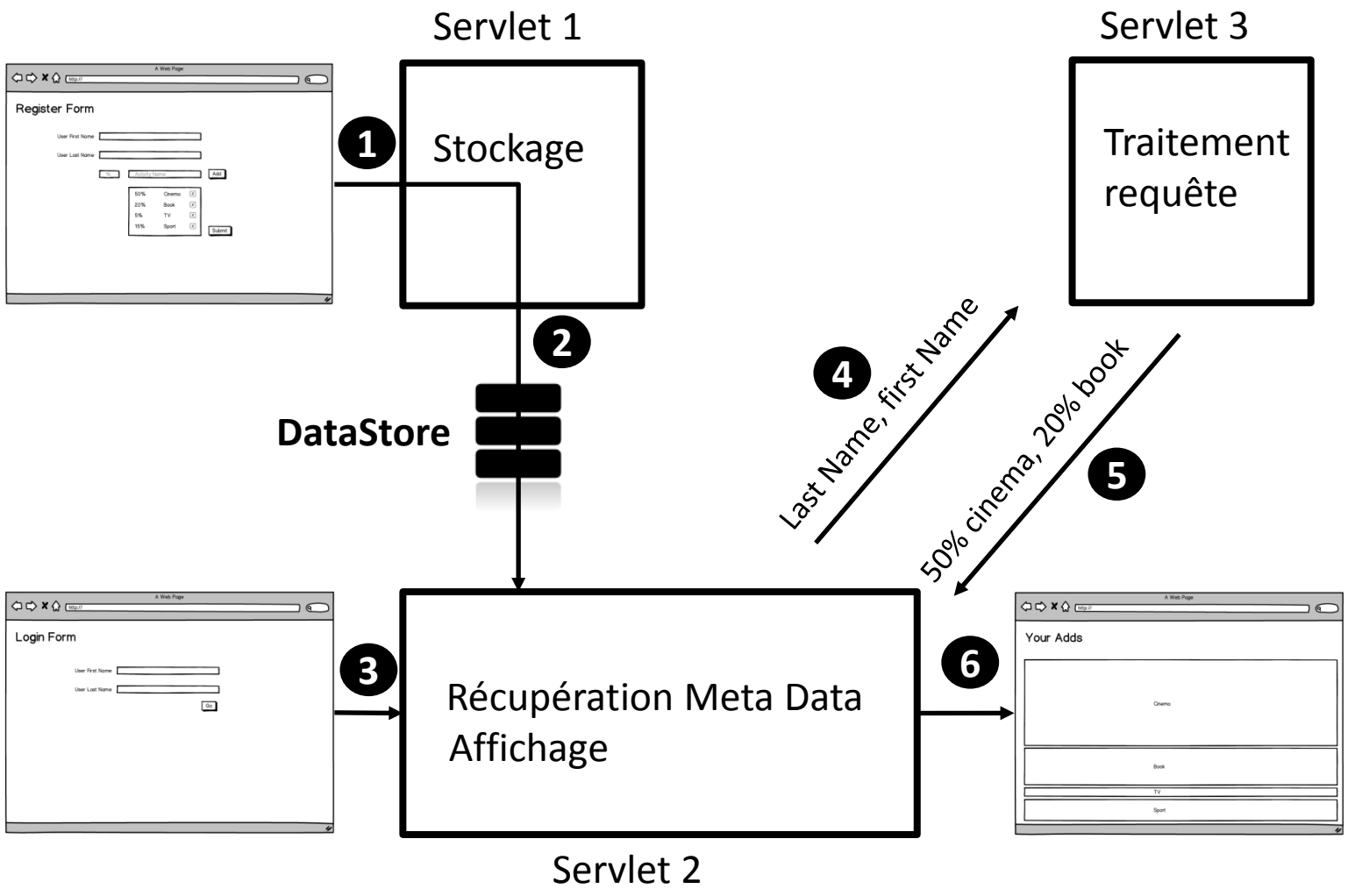
Récupération des champs du message

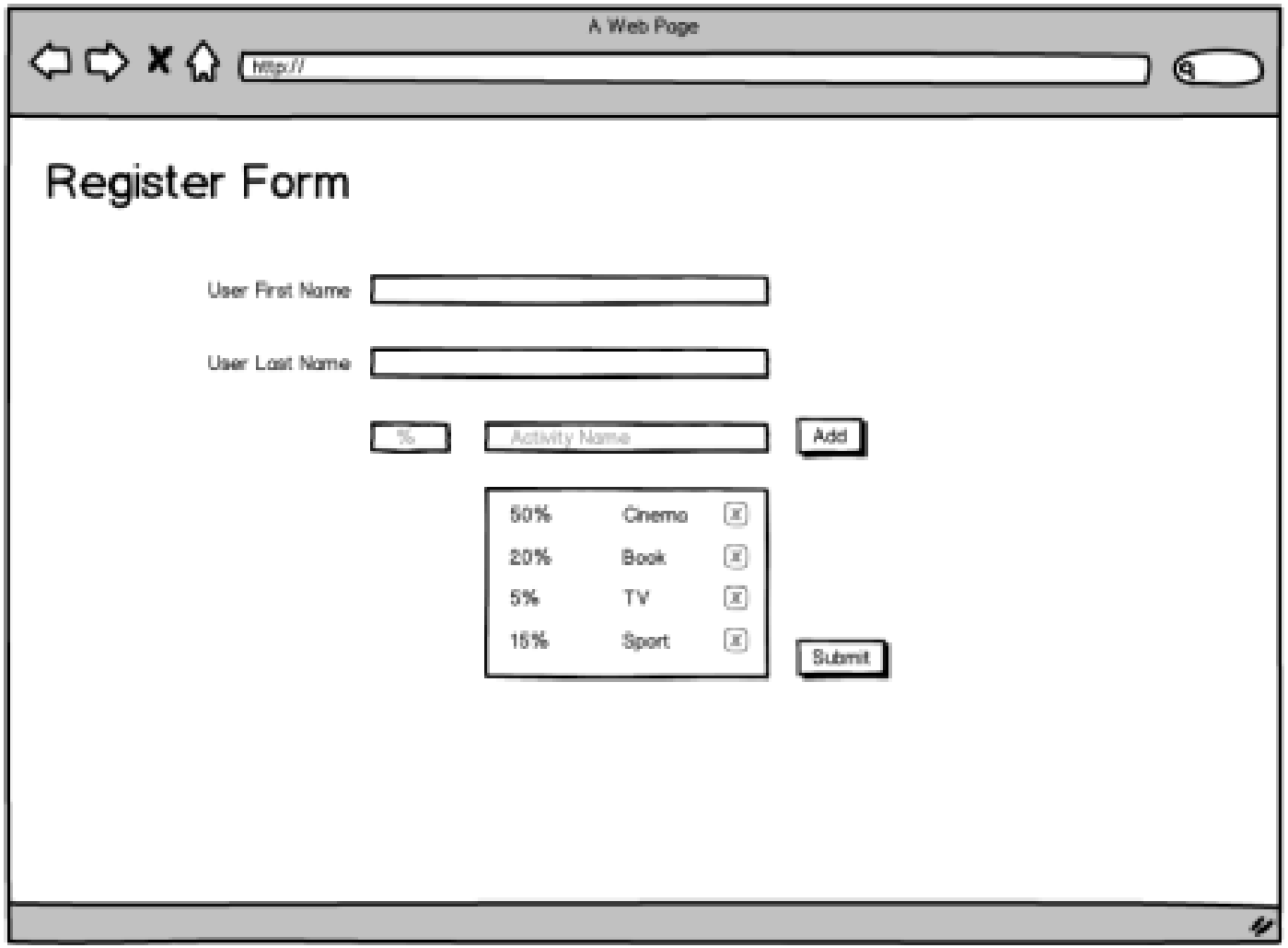
```
public class MailHandlerServlet extends HttpServlet {
  public void doPost(HttpServletRequest req,  HttpServletResponse resp)
      throws IOException {
    Properties props = new Properties();
    Session session = Session.getDefaultInstance(props, null);
    MimeMessage message;

    try {
      message = new MimeMessage(session, req.getInputStream());
      Address[] addresses = message.getFrom();
      if (message.isMimeType("text/plain")) {
        String content = (String)message.getContent();
      }
    } catch (MessagingException e) {e.printStackTrace();}
  }
}
```

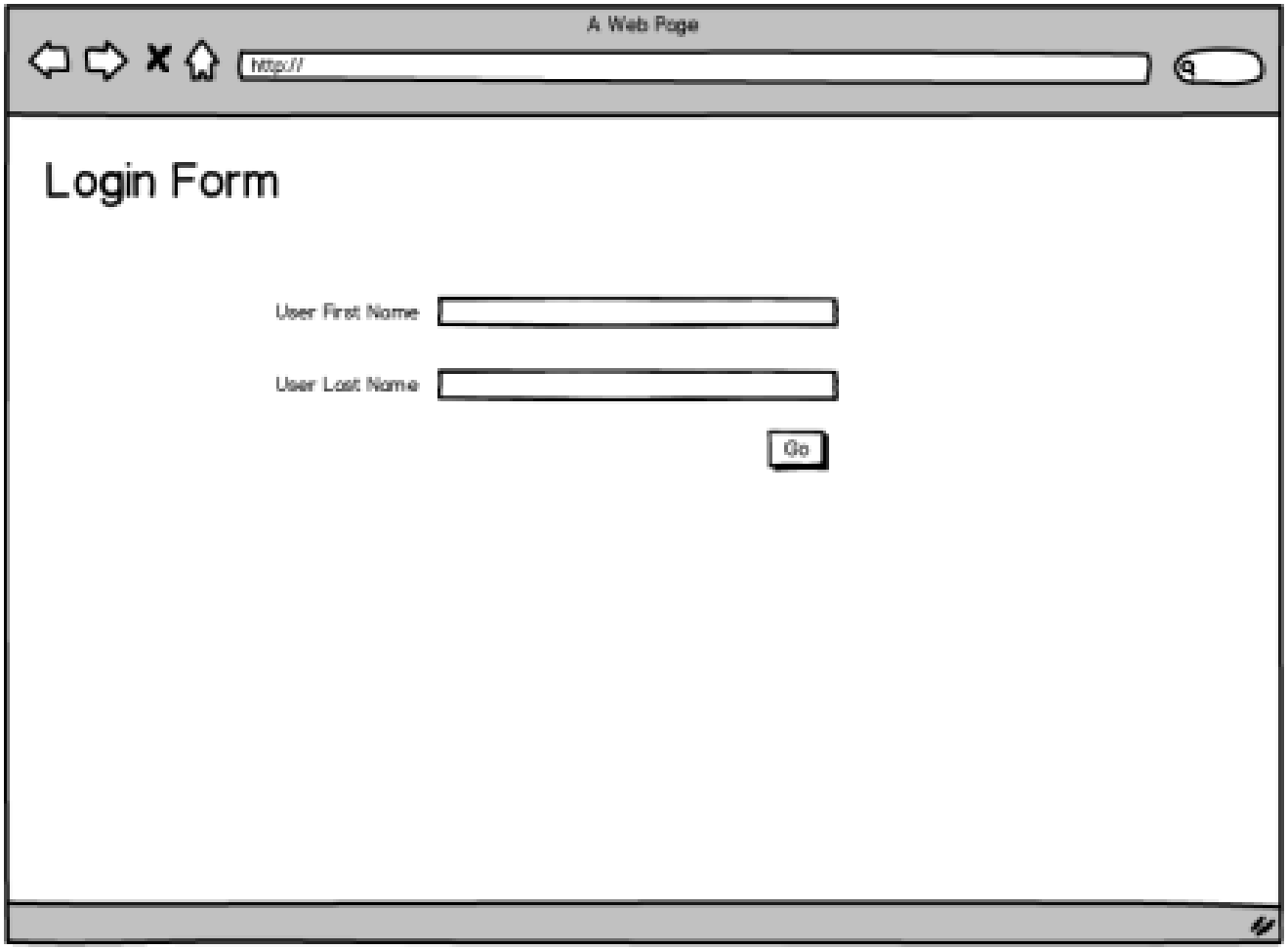
- **A vous de Jouer !**

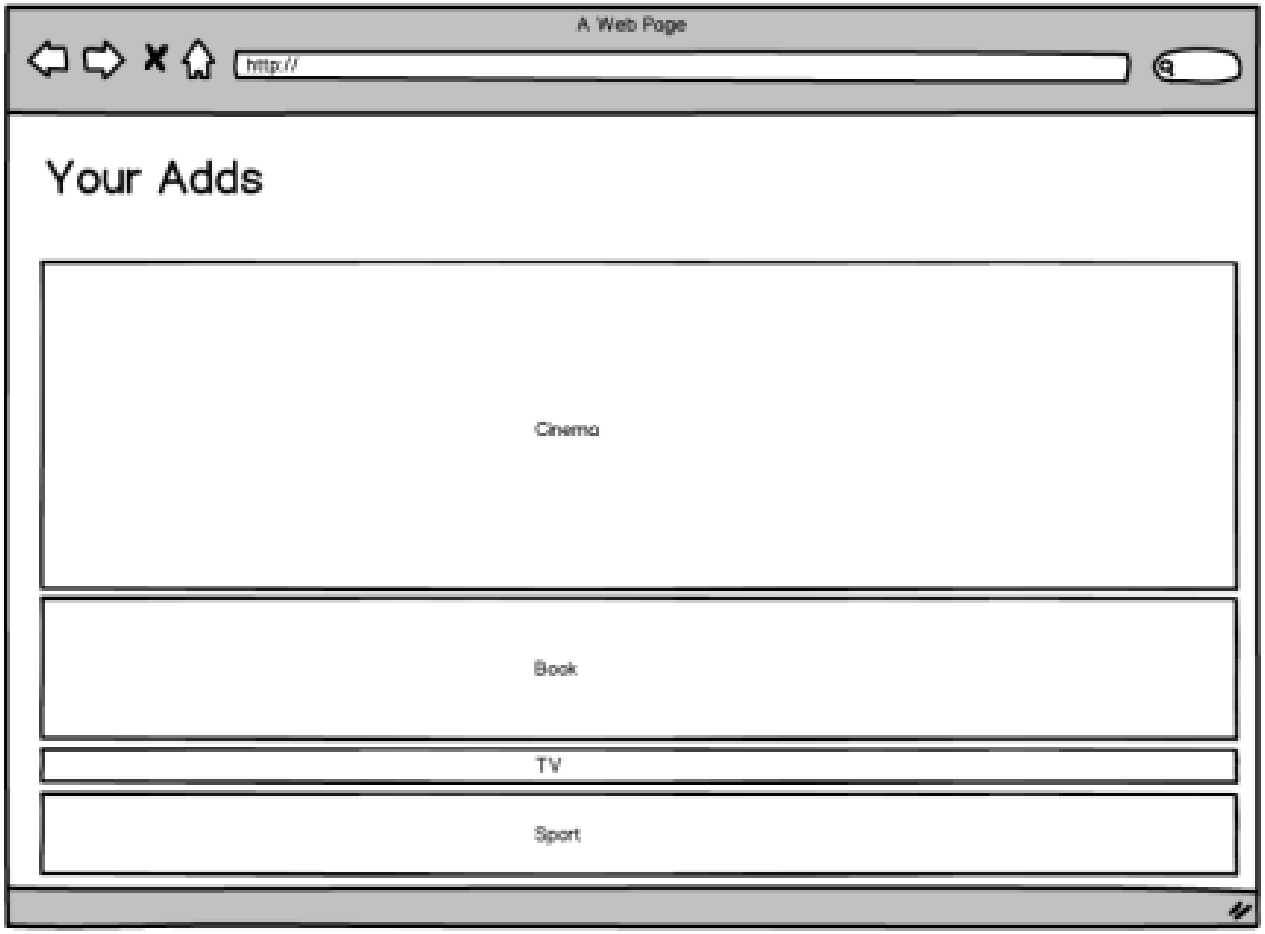
Google as a Rest

Google App Engine









Questions ?