OZALID L-1.1.2

Traces de l'Editeur Ozalid

Responsables : Urbilog & IRIS/Silex

Rédacteurs : Urbilog (Michel Hoël), IRIS/Silex (Anh Hoang Lê)

Lecteurs/Correcteurs : IRIS/Silex (Alain Mille, Amélie Cordier, Anh Hoang Lê)

Date : Février 2013

Résumé

Ce livrable décrit le prélèvement des traces et leur envoi au serveur de Traces.

Contenu

Traces de l'Editeur Ozalid 1
Responsables : Urbilog & IRIS/Silex 1
Résumé 1
1. Introduction 3
2. La trace 3
1. Les types de trace 3
2. La collection de traces 3
3. Modèle trace 3
4. Les différents types de message 4
5. Stockage local de la trace 5
1. Initialisation de l'éditeur Ozalid 5
2. Traitement envoi à l'initialisation de l'éditeur 6
3. Un nouveau paquet à stocker dans LS 6
4. Ajout d'une trace dans paquet 6
5. Demande d'envoi du paquet courant 6
6. Stockage sur serveur Ozalid 7
7. Récupération par le serveur de Trace 7

1. Introduction

Le but des Traces Ozalid est d'offrir une navigation assistée en analysant en temps réel les informations les actions utilisateurs avec l'Editeur Ozalid.

2. La trace

1. Les types de trace

Vu l'évolutivité de l'application, il n'est pas souhaitable de tracer toutes les actions de l'application.

Le plus intéressant est de tracer plutôt les actions directes, via les fonctions de l'Editeur, et indirectes notamment les mots/lignes/blocs corrigés /validés/enrichis.

2. La collection de traces

La collection de traces est à mettre à jour au fur et à mesure de l'activité via l'éditeur.
Chaque instance d'un modèle trace correspond à la trace d'un événement.
Le format des échanges avec le système de Traces est JSON.

3. Modèle trace

Le format d'un « paquet trace » est le suivant:

```
{
  "hT" : {
      "dT1":<date début trace>",
      "dT2" : <date fin trace>,
      "dT3" : <date réception serveur>,
      "lock": "<true|false>",
      "ready": "<true|false>"
      },
  "pT" :
      {
        "dT" : "<delta de temps en millisecondes>",
        "mT" :
            {
                  <objet message>
            }
      },
      …
      {
            …
      }
}
```

Donc une en-tête **hT** définissant la datation des traces :

- **dT1** : Date GTM de la première trace du paquet, au format YYYYMMDDHHMMSSMS
- **dT2** : Date GTM de l'envoi du paquet, au format YYYYMMDDHHMMSSMS
- **dT3** : Date GTM de réception du paquet, au format

YYYYMMDDHHMMSSMS, sera mis à jour par serveur Ozalid.

- **lock** : Etat verrouillé ou non du paquet (« true » ou « false »), c'est-à-dire que l'application ne peut plus ajouter des traces dans ce paquet.
- **ready** : Etat prêt à être envoyé (« true » ou « false »), est mis à jour lorsque qu'une trace de modification ou de validation du contenu à corriger est dans le paquet, cela mérite d'être envoyé à tout moment. Sinon il sera envoyé à l'initialisation d'un éditeur.

Puis l'ensemble des traces du paquet (**pT**).

Le format d'une trace est le suivant:

Avec :

- **dT** : delta de temps entre la prise en compte de la trace et le temps **dT1**, la date de la première trace du paquet.
- **mT** : le message de la trace.

4. Les différents types de message

Le format d'un message d'une ligne de trace est le suivant:

```
"mT" :
    {
        "id" :  [prefixe]_[type],
        [info_1] :  [val_info_1],
        …,
        [info_n] :  [val_info_n]

    }
```

Le "**prefixe**" dépend de l'objet spécifique de l'application :

- **oze** : l'application elle-même, l'Editeur ozalid,
- **ozem** : annonce une modalité de l'action,
- **ozec** : annonce une correction.
- **ozev** : annonce d'une validation.

Le "**type**" dépend du préfixe:

- **oze_[code]** : annonce le chargement d'une nouvelle vue de l'éditeur
  - **oze_idg** : Identifiant utilisateur + Session + OS + User Agent + Version Editeur + id instance Editeur
  - **oze_view** : Affichage d'une vue particulière
  - **oze_context** : Récupération d'un contexte d'informations
  - **oze_subview** : Affichage d'une « sous-vue » particulière
- **ozem_[code]** : annonce une modalité de l'action
  - **ozem_k** : annonce un événement par clavier
  - **ozem_m** : annonce un événement via la souris
  - **ozem_t** : annonce un événement par toucher (« touch »)
  - **ozem_fn** : trace dans une fonction de l'éditeur
- **ozec_[code]** : annonce une correction
  - **ozec_w** : correction d'un mot -> id du mot corrigé, texte du mot avant correction, après correction

    - **ozec_l** : correction d'une ligne -> id de la ligne validée, texte de la ligne avant correction et après correction
    - **ozec_b** : correction d'un bloc, structure
- **ozev_[code]** : annonce une validation
    - **ozev_w** : validation d'un mot -> id du mot validé, texte du mot

Les propriétés particulière à chaque élément **`[prefixe]_[type]`**:

- **ozem_k**: annonce un événement par clavier:
    - **key :**
        - **tab** : déplacement par tabulation
        - **stab** : déplacement par tabulation inversée (Shift Tab)
        - **left** : déplacement flèche vers la gauche
        - **right**: déplacement flèche vers la droite
        - **up**: déplacement flèche vers le haut
        - **down**: déplacement flèche vers le bas
        - **sc**: déplacement par raccourci clavier (Shortcut)
    - **value :** si **key : sc**, valeur du raccourci
    - **elt :** élément concerné (target)

- **ozem_m**: annonce un événement par souris:
    - **evt :**
        - **click** : click souris
        - **dblClick** : double click souris
    - **elt :** élément concerné (target)

- **ozem_t**: annonce un événement par toucher (touch):
    - **evt :**
        - **touch** : touch simple
        - **tap** : tap
        - **dbltap** : double click souris
        - **touchmove** : avec 1, 2, 3 … doigts, sens (haut, bas, gauche, droite..)
        - **gesture** : geste complexe
    - **elt :** élément concerné (target)

- **ozem_fn**: annonce tout autre événement
    - **<clé> : <valeur>**

Aujourd'hui il est difficile de donner une liste exhaustive des informations du message. Un dictionnaire sera mis à jour et disponible au fur et à mesure de l'avancement des IHMs.

5. Stockage local de la trace

Les traces locales avant envoi au serveur sont stockées en **LocalStorage** du navigateur. Cette zone est permanente pour un domaine particulier et permet ainsi de conserver les traces même si elles n'ont pu être envoyées au serveur.

Voici comment sera gérée la mise à jour des traces locales.

1. Initialisation de l'éditeur Ozalid

L'éditeur Ozalid s'initialise lorsque l'on ouvre un nouvel onglet dans un navigateur avec l'URL de l'éditeur.

Plusieurs onglets sur un navigateur peuvent être ouverts sur l'application Ozalid, il faut donc prévoir à chaque initialisation de créer un « id éditeur » unique.

Cet « id éditeur » va servir de base à la clé de stockage dans le LocalStorage, que l'on appellera **LS** maintenant, endroit où seront stockés les « paquets traces ».

2. *Traitement envoi à l'initialisation de l'éditeur*

Il peut rester des « paquets traces » dans le LS. Les causes sont multiples, par exemple :

- Fermeture de l'onglet ou du navigateur sans s'être déconnecté.
- Le travail hors-ligne (dans le futur), donc pas possibilité d'envoyer les traces.

Donc la première mission de l'éditeur est d'essayer d'envoyer ces paquets restés dans LS.

La procédure est la suivante :

- D'abord il faut verrouiller (voir plus haut **lock**) le paquet si ce n'est déjà fait.
- Si on en ligne, il est donc possible d'envoyer le/les paquets :
  - Pour chaque paquet, mettre à jour **dT2** (voir plus haut).
  - Envoyer les paquets (requête Ajax) au serveur Ozalid.
  - Attendre réponse :
    - Si **Success** : supprimer les paquets de LS.
    - Si **Error** : recommencer (3 fois max).
    - Si pas réussi au bout de 3 fois abandonner, cette procédure sera refaite plus tard car les paquets sont toujours en LS.

3. *Un nouveau paquet à stocker dans LS*

L'éditeur peut commencer à tracer. Il crée un nouveau paquet :

- Clé = « id éditeur » (Voir plus haut), cette clé sert de clé courante.
- Contenu est un json, par défaut :

```
{
  "hT" : {
        "dT1": "<date début trace>",
        "dT2" : "",
        "dT3" : "",
        "lock": "false",
        "ready": "false"
        },
  "pT" : {}
}
```

4. *Ajout d'une trace dans paquet*

Si le paquet courant est verrouillé, on crée un nouveau paquet. Ce nouveau paquet aura comme clé : <id éditeur>_<i>, <i > étant un index s'incrémentant au fur et à mesure. Cette nouvelle clé sert de clé courante.

Puis dans le paquet courant on insert la trace dans **pT.**

Si la trace et de type correction ou validation (**ozec**, **ozev**), le status **ready** du paquet est mis à **true**.

5. *Demande d'envoi du paquet courant*

La demande peut être provoquée par :

- l'utilisateur : sur bouton « Envoyer trace », qui devient opérationnel si le status **ready** du paquet courant est à **true.**
- Sur changement de page de l'éditeur, on reprend donc à la procédure d'initialisation de l'éditeur.

- Lorsque le paquet statué **ready** contient un nombre de traces supérieur à une limite définie.
- Au bout d'un délai à partir de la création du paquet supérieur une limite définie également et statué **ready**.

6. Stockage sur serveur Ozalid

Côté serveur Ozalid, les paquets réceptionnés sont enrichis de la date de réception.
L'information **dT3** (Voir plus haut) du json reçu est mise à jour.
Ce json modifié mis en fichier sur un répertoire spécifique de dépôt de traces, le nom du fichier ressemblera à :

<répertoire dépôt trace>/**oze_YYYYMMDDHHMMSSMS.json**

YYYYMMDDHHMMSSMS étant la valeur de **dT3**

7. Récupération par le serveur de Trace

Un service destiné au serveur de Trace de LIRIS permettra de récupérer les informations des fichiers du dépôt de traces.

Si on nomme maintenant les paquets stockés sur répertoire de dépôt traces, p1, p2, p3… pn, le protocole est le suivant :

- Demande serveur Trace : envoyer paquet (pi, 10), à partir du paquet pi les 10 suivants.
- Le service répond en envoyant les paquets pi+1, pi+2…
- Le serveur de Trace après réception envoie un accusé de réception : ACK, pi+1, pi+2….
- Le service répond OK.
- Les paquets pi+1, pi+2… sont mis dans une archive sur un répertoire de sauvegarde des traces.