Manuel de développement du programme UN AN UN SCRIPT

> Ce manuel en un seul fichier est censé fournir une information rapide sur le développement du programme UN AN UN SCRIPT. Il doit être produit en PDF à chaque changement, et ce PDF doit pouvoir être chargé depuis n'importe quelle grande partie du site.

[liste des travaux de l'auteur] #listedestravauxdelauteur "Liste des travaux de l'auteur"

Généralités

Description du programme

Ce programme accompagne un auteur sur une année dans :

- son approche de la narration (aspect pédagogique)
- l'écriture d'un premier scénario ou manuscrit (aspect pratique)

Fonctionnement général

Le programme est conçu sur une année, avec des tâches quotidiennes à exécuter par l'auteur. L'auteur travaille en autonomie presque complète, donc ces tâches sont censées lui donner toute information nécessaire pour mener à bien son apprentissage et son récit.

Bureau de l'auteur

Chaque auteur inscrit au programme possède un bureau, son centre de travail névralgique d'où il peut accomplir toutes les actions et voir où il en est de son travail.

Bases de données

Il existe trois types de base de données dans le programme UN AN UN SCRIPT :

1. **unan_cold**.

Ce sont les données “froides”, i.e. la définition des travaux, des jours-programmes, etc. Ce sont en quelque sorte des *données fixes*. 2. **unan_hot**.

Ce sont les données “chaudes”, i.e. la définition des programmes qui ont déjà eu lieu ou ont lieu, les données sur les projets, etc. Ce sont des données dynamiques qui peuvent être modifiées à tout moment et, surtout, qui concernent les auteurs impliqués dans le programme. 3. **user/program.db**.

Ce sont les bases de données propre à chaque auteur, qui consigne ses

travaux, ses réponses aux questionnaires, etc., i.e. toutes les informations concernant le programme concerné.

Aide pour l'auteur

Toute l'aide pour l'auteur doit se trouver dans un unique fichier PDF qui peut être chargé dans une autre page.

Ce fichier se trouve à l'adresse :

```
    ./objet/unan/aide/manuel_utilisateur/manuel_utilisateur
.pdf
```

Il est composé en Latex mais on pourra imaginer le faire en Markdown aussi.

Synopsis du parcours d'un auteur

Démarrage du programme UN AN UN SCRIPT

Un programme UN AN UN SCRIPT se démarre lorsque l'auteur s'inscrit au programme, c'est-à-dire au moment précis où il revient du site PayPal après le paiement de son inscription.

> Note : Le montant de ce paiement est défini dans le fichier `./objet/unan/lib/required/unan/class.rb` dans la méthode `tarif`.

Vue et module `on_ok`

Il passe alors par la vue `./objet/unan/paiement/on_ok.erb` et le module `./objet/unan/paiement/on_ok.rb`.

Module signup_user.rb

on_ok.rb charge le module `./objet/unan/lib/module/signup_user.rb` et appelle dans ce module la méthode `User#signup_program_uaus`.

signup_program_uaus

Cette méthode :

- Crée le programme UN AN UN SCRIPT de l'auteur
- Crée le projet de l'auteur
- Crée ses tables dans la base de données propre au programme (chaque programme possède sa propre base de données, même lorsqu'il est suivi par le même auteur).
- Instancie le premier "jour-programme" de l'auteur, ce qui va avoir pour conséquence de définir son travail (cf. program.start_pday ci-dessous).
- Envoie les messages de confirmation d'inscription, de premières explications, etc. à l'auteur et d'avertissement à l'administration.

Changement des jours-programme (P-Days)

Tous les jours-programme, donc tous les jours pour un rythme de 5 et plus ou moins un jour réel pour les autres rythmes, le jour-programme des auteurs change.

Ce changement est produit par le cronjob horaire qui travaille toutes les heures.

`Program#start_pday` est la méthode appelée pour démarrer un nouveau jour de travail.

> Noter que cette cette méthode qui est appelée au tout premier démarrage du programme (après le paiement).

Les Préférences de l'auteur

Convention de nommage

Par convention, tous les noms de préférence sont enregistrées dans la table `variables` avec **un nom commençant par "pref_"**.

L'identifiant de cette préférence, utilisé par et dans les méthodes fait abstraction de ce "pref_".

Par exemple, on utilise dans le code la préférence `bureau_after_login` mais dans la table `variables` la donnée est enregistrée au nom de `pref_bureau_after_login`.

Note : Cela permet simplement de reconnaitre ce type de variable dans la table et de, par exemple, pouvoir toutes les relever d'un coup lorsque ce sont les préférences qu'on doit régler.

Méthodes `preference` et `preference=`

Les méthodes `User#preference` et `User#preferences=` permettent d'enregistrer les préférences de l'user :

```
user.preference= <pref id>, <pref value>

ou

user.preference= <pref id> => <pref_value>
```

Pour récupérer la valeur :

```
user.preference(<:pref id>[, valeur défaut])
```

Par exemple :

```
user.preference(:bureau_after_login)
# => true s'il faut rejoindre le bureau "Un an un script" a
près
# l'identification.
```

Relève de toutes les préférences

On peut relever toutes les préférences d'un coup à l'aide de la méthode `User#preferences`. On peut faire ensuite appel à la méthode `User#preference` pour obtenir une valeur, de façon tout à fait normale.

Note : En fait, faire appel à la méthode `User#preferences` évite simplement de faire des appels trop nombre d'affilée à la table `variables`, lorsque plusieurs préférences doivent être utilisées.

Ajout d'une nouvelle préférence

Pour ajouter une nouvelle préférence programme UN AN UN SCRIPT, il faut :

- lui définir un nom original,
- ajouter un checkbox et une explication pour cette préférence dans la vue `./objet/unan/bureau/panneau/preferences.erb` ,
- l'ajouter à la liste des préférences de l'user dans le fichier `./objet/unan/bureau/panneau/preferences.rb` (méthode `user_preferences`)

Variables programme de l'auteur

Listing complet des variables courantes

Cette liste doit tenir à jour la liste complète des variables dans la table `variables` d'un user inscrit au programme UN AN UN SCRIPT.

Noter que les variables sont classées dans leur ordre d'importance et donc qu'on trouve pêle-mêle des préférences et des variables normales.

```
pref_rythme

    {Fixnum} Le rythme courant de l'auteur, de 1 à 9
    <- user.rythme
    <- user.preference(:rythme)

current_pday

    {Fixnum 1-start} Le jour-programme courant de l'auteur.
    <- user.get_var :current_pday
    <- user.pday

pref_daily_summary

    {Boolean} True si l'auteur veut recevoir des récapilati
fs
    journaliers même lorsqu'il n'a pas de nouveau travail.
    Défault : false

pref_sharing

    {Fixnum} Niveau de partage du projet de l'auteur
    <- user.preference(:sharing)

total_points_projet

    {Fixnum} Total des points courants sur le projet couran
t.
    C'est cette valeur qui est utilisée pour connaitre le g
rade
    de l'user dans Unan::Program::DATA_POINTS

total_points

    {Fixnum} Total des points de l'auteur, tous programmes
    confondus.
```

Indice du jour-programme (p-day) courant

On obtient le jour-programme courant de l'auteur par :

```
    <user>.program.current_pday
```

> Note : On pourrait faire un raccourci `user.current_pday` mais je préfère que le jour-programme courant soit toujours associé au programme de l'auteur.

Helpers de liens

Liens vers les panneaux principaux

Utiliser ces liens pour rejoindre les panneaux principaux du bureau :

```
<%= bureau.lien_etat %>         # L'état des lieux
<%= bureau.lien_travail %>      # Vers le travail à faire
<%= bureau.lien_quiz %>         # Vers les questionnaires,
checklist, etc.
<%= bureau.lien_messages %>     # Vers les messages forum
<%= bureau.lien_pages_cours %>  # Vers les pages de cours à
 lire
```

On peut ajouter en **premier argument** le titre à donner au lien :

```
<%= bureau.lien_travail "votre travail du jour" %>
```

On peut définir en **second argument** les options à utiliser, c'est-à-dire les attributs à ajouter à la balise :

```
    <%= bureau.lien_travail nil, {target: `_new', class:`mo
nlienspecial'} %>
```

Note : Ces liens sont définis dans le fichier `./objet/unan/lib/required/Bureau/helper.rb`

Programme de l'auteur

Tout auteur (user) inscrit au programme UN AN UN SCRIPT possède une

instance `Unan::Program` consignée dans la table `programs` de `unan_hot.db`.

Gestion du nombre de points de l'auteur

Pour encourager l'auteur, un nombre de points lui est accordé chaque fois qu'il achève un travail.

Les points sont conservés dans les instances `works` du programme. On peut donc obtenir les points d'un travail en particulier par :

```
    iwork = Unan::Program::Work::get(program_id, work_id)
    nombre_points = iwork.points

    Ou, si c'est le programme courant de l'auteur :

    nombre_points = user.work(work_id).points
```

Changement de jour-programme de l'auteur

Avec le nouveau fonctionnement, il suffit de changer le `current_pday` du programme de l'auteur pour le faire changer de programme. Tout le reste s'effectue tout seul puisque les travaux sont calculés à la volée lorsqu'il vient sur son bureau. Donc passer l'user au jour suivant revient simplement à faire :

```
    - Prendre le rythme du programme de l'auteur
    - Calculer s'il y a un changement de rythme
    - Passer l'auteur au jour-programme suivant si nécessai
re
```

Ce changement se fait par le Cron-job horaire

Rythme de l'auteur

Le rythme auquel on suit le programme UN AN UN SCRIPT est une donnée

fondamentale du programme.

Ce rythme a une valeur de **5** quand il est *moyen*, c'est-à-dire qu'un jour-réel correspond à un *jour-programme*.

```
RYTHME = 5
=>  UN JOUR RÉEL = UN JOUR-PROGRAMME
```

Cette valeur s'obtient par différents moyens :

```
user.rythme

user.program.rythme
```

Calendrier du programme

À chaque programme correspond à calendrier (`Unan::Program::Cal`) propre au programme de l'user qui possède le programme.

On instancie ce calendrier seulement avec l'instance programme :

```
program = Unan::Program::new(<prog id>)
ou
program = user.program

calend  = Unan::Program::Cal::new(program)

ou

program.cal # TODO: Voir si on peut l'avoir comme ça
```

Donc pour obtenir le calendrier depuis l'user :

```
calendrier = user.program.cal
```

P-Days absolus

- Principes générateurs des PDays
- Organisation d'un P-Day

Les `p-days` absolus, instance de class `Unan::Program::AbsPDay`, définissent précisément le travail à faire ou amorcer un jour précis du calendrier, quel que soit le rythme du programme.

Ces p-days absolus sont définis dans la table :

```
    unan_cold.absolute_pdays
```

… qu'on peut obtenir par :

```
    Unan::table_absolute_pdays
```

Principes générateurs des P-Days

- **Un p-day est absolu**

  Il est immuable et définit le plus justement ce que doit être la journée de travail de l'auteur.

  En revanche, il peut être constitué de parties dynamique en fonction des points de l'auteur, de son support final (roman, film, etc.) ou de toute autre donnée de l'auteur qui peut influencer son travail.

- Quel que soit le rythme de travail choisi, il y aura toujours 366 p-days dans un cycle complet de programme.

  Simplement, plus le rythme de travail est lent et plus il y aura de jours réels dans un p-day. Inversement, plus le rythme de travail est important et plus il y aurait de jours réels dans un p-day.

- Ces 366 PDays définissent la composition de la version 2 d'un scénario/manuscrit et la lecture complète de Narration.

- Un p-day définit tout le travail à accomplir

  C'est le "point central" du programme, le point d'où on envoie vers les pages de cours, vers les questionnaires, etc.

Il est constitué de `travaux` (instances de classe `Unan::Program::AbsWork`), travaux qui renvoient à ces pages de cours, ces questionnaires, etc.

- Organisation d'un P-Day

Organisation d'un P-Day

Le P-Day est constitué de tâches (des “works” ou instances de classe `Unan::Program::AbsWork`) à accomplir :

```
P-Day --> Work  --> Page cours
      --> Work  --> Page cours
      --> Work  --> Questionnaire
      --> Work  --> Action à accomplir
```

Travaux de l'auteur

L'instance `work`

- L'instance `Unan::Program::Work`
- Obtenir l'instance `Work`
- Propriétés de l'instance `Unan::Program::Work`
- Suivi du travail
- Programmation future d'un `work`

L'instance `Unan::Program::Work`

Les travaux de l'auteur sont des instances :

```
    Unan::Program::Work
```

Ils sont consignés dans la table :

```
    DATABASE
      ./database/data/unan/user/<user-id>/programme<program
-id>.db
    TABLE
      works
```

Noter un point important : ces “works”, la plupart du temps, ne sont créés que lorsque le travail est marqué démarré par l'auteur. Plus précisément, pour créer le work, il faut que :

- une “tâche” doit être marquée “démarrée” par l'auteur,
- une “page de cours” doit être marquée "vue" par l'auteur,
- un “quiz” soit rempli par l'auteur,
- (les choses ne sont pas encore définis pour les messages de forum).

Obtenir l'instance `Work`

Pour obtenir une instance d'un travail en particulier et donc lire ses données ou la manipuler :

```
    Unan::Program::Work::get(<program>, <work-id>)

    <program> est une instance Unan::Program qu'on peut obt
enir par :
    user.program
```

Si c'est le programme courant de l'auteur, on peut l'obtenir par :

```
    user.work(<work-id>)
```

Propriétés de l'instance `Unan::Program::Work`

Cette instance possède ces propriétés :

```
    id              {Fixnum} Identifiant propre au travail
    program_id      {Fixnum} Identifiant du programme auquel
appartient le
                             work
    abs_work_id     {Fixnum} Identifiant du travail absolu co
rrespondant
    indice_pday     {Fixnum} L'indice du jour-programme de ce
 travail.
                             Noter qu'il correspond au jour-p
rogramme du
                             travail absolu, même si le trava
il a été
                             démarré plus tard.
    type_w          {Fixnum(2)} Type de travail (pour raccour
ci)
    status          {Fixnum} Statut actuel du travail, de 0 à
 9
    options         {String} Options sur 64 octets du travail
.
                             Cf. ci-dessous.
    points          {Fixnum} Nombre de points gagnés pour ce
travail.
    ended_at        {Fixnum} Timestamp de la fin du travail
    created_at
    updated_at
```

Pour les points, cf. Gestion du nombre de points de l'auteur.

Pour les particularités de `created_at`, cf. Programmation dans le futur d'un travail ci-dessous.

Suivi du travail

Les deux propriétés permettant de suivre un travail sont :

```
    status          {Fixnum} Nombre de 0 à 9
    options         {String} Chaine de 64 "bits"
```

Valeurs du status

```
    0   Travail créé/instancié (inutilisé maintenant que le
 travail
        est seulement créé lorsque l'auteur le marque démar
ré/vu)
    1   Travail marqué démarré par l'auteur, c'est-à-dire q
u'il l'a
        vu et pris en compte. Il se marque démarré depuis l
e bureau
        Noter que ça ne s'applique qu'aux "vrais" travaux,
pas aux
        pages de cours ou autres quiz.
        Interroger la méthode `started?` pour savoir si le
travail a
        été démarré.

    9   Travail terminé
        Interroger la méthode `completed?` pour savoir si l
e travail
        est fini.
```

Programmation future d'un `work`

CETTE PARTIE DOIT DEVENIR OBSOLÈTE AVEC LE NOUVEAU FONCTIONNEMENT PAR “RECHERCHE ABSOLUE” (cf. Liste des travaux de l'auteur). MAIS COMMENT LA REMPLACER ? Un `work` peut être programmé à l'avance. C'est le cas par exemple lorsqu'un questionnaire n'a pas été rempli correctement et qu'il doit être re-proposé plus tard à l'auteur. Dans ce cas, on utilise la propriété `created_at` et `updated_at` synchronisé pour définir que le travail doit être commencé plus tard.

```
    iwork.set( created_at: <date dans le future>, updated_a
t: <date future> )
    # rappel : date est un fixnum correspondant au nombre d
e secondes
```

Liste des travaux de l'auteur

NOTE : La liste des travaux ne fonctionne plus par liste d'identifiants consignant les travaux courants de l'auteur. Cela rendait la gestion des changements de jours infornale.

Maintenant, on consulte simplement les travaux à faire dans l'absolu de cette façon :

```
    Soit le jour-programme courant PD
    * On relève tous les travaux absolus de tout type à fai
re jusqu'à
      ce PD (ce qui peut être très "long" vers la fin, mais
 il n'y
      aura que 366 jours de toute façon)
    * On relève tous les travaux finis et non finis de l'au
teur
      (donc ses "works")
    * On retire de la liste des travaux absolus tous les tr
avaux
      qui ont été achevés par l'auteur.
    * Il ne reste alors dans la liste des travaux absolus q
ue les
      travaux courants, dont certains peuvent avoir été dém
arrés
      (un "work" auteur existe, alors) et d'autres non déma
rrés (dans
      ce cas, ils ne possèdent pas de "work" auteur)
```

Class Unan::Program::CurPDay

Une classe est spécialement dédiée à la gestion des travaux au jour courant :

```
    Unan::Program::CurPDay
```

Elle est définie dans :

```
    ./objet/unan/lib/required/cur_pday/inst.rb
```

Dans le bureau central de l'auteur, on peut obtenir l'instance de cette classe par :

```
    bureau.current_pday
```

Pour obtenir les travaux non achevés de type `task` :

```
bureau.current_pday.undone(:task)
# => {Array} de {Hash} contenant les données des
# travaux absolus + les données du work si le travail
# a été démarré
```

Les méthodes principales sont :

```
done(type)        # Liste des data de travaux accomplis
                  # Data des Works, pas des AbsWorks
undone(type)      # Liste des data de travaux inachevés
                  # Data des AbsWorks + quelques autres
started(type)     # Liste des data de travaux démarrés
                  # mais non achevés
                  # Data des Works pas des AbsWorks
encours(type)     # idem
```

Ces méthodes retourne un `Array` de `Hash` qui sont les données enregistrées soit du travail absolu (`Unan::Program::AbsWork`) soit du travail propre à l'auteur (`Unan::Program::Work`) auxquelles sont ajoutées quelques autres données comme :

```
work_id       L'ID du work de l'auteur, if any
indice_pday   L'indice du jour-programme
```

Types de travaux

L'auteur peut avoir ces types de travaux :

- fiche de cours à lire,
- travail à faire (un document, une recherche, etc.),
- questionnaire à remplir,
- réponse ou message à passer sur le forum.

```
  WORKS désigne n'importe lequel de ces travaux dans le
 programme.

  Les types, au niveau du programme, sont :

      TASK      Une tâche à accomplir
      QUIZ      Un questionnaire à répondre
      PAGE      Une page de cours (Narration) à lire.
                Noter qu'il ne s'agit pas de l'id de la
 page dans
                Narration mais d'un identifiant propre
à Unan,
                puisqu'il s'agit ici d'un work.
```

Pour le détail sur les listes de travaux, cf. la [liste des travaux de l'auteur][].

Un work s'instancie à l'aide de :

```
    Unan::Program::Work::new(<user>.program, <work id>)
    # (ou get)

    L'instance Work possède une propriété `item_id` qui déf
init
    l'ID de l'item en fonction du type de work :

        Si le type est          Alors item_id est
        ------------------------------------------------
---------
        task                    l'ID d'une tâche
        quiz                    l'ID d'un questionnaire
        page                    l'ID d'une page de cours
        forum                   l'ID d'un travail de forum


    abs_work_id         ID de l abswork du travail
    indice_pday         Indice du jour-programme du travail
```

Noter que l `indice_pday` n'est pas le jour-programme auquel le travail a été commencé, mais le jour absolu pour lequel le travail a été programmé, même s'il est démarré seulement le lendemain ou surlendemain de ce jour.

Les Questionnaires (Quiz)

Classe absolue (`Unan::Quiz`) et auteur (`User::UQuiz`)

Il faut comprendre qu'il y a deux classes pour les quiz :

La classe qui contient les données absolues :

```
Unan::Quiz
```

La classe qui contient les résultats de l'auteur au quiz :

```
User::UQuiz
```

Dans les deux cas, il faut requérir la librairie `quiz` .

Requérir la librairie `quiz`

```
site.require_objet 'unan'
Unan::require_module 'quiz'
```

Questionnaires de l'auteur

On peut récupérer les questionnaires de l'auteur à l'aide de la propriété :

```
user.quizes
# => Hash avec en clé l'ID qui UQuiz et en valeur
# l'instance User::UQuiz.
```

Cette méthode peut recevoir un filtre pour ne retourner que les instances de quiz voulues.

```
user.quizes(
  created_after:
  created_before:
  max_points:            Ne doit pas avoir plus de points q
ue ça
  min_points:            Doit avoir au moins ce nombre de p
oints
  quiz_id:           {Fixnum} ID du questionnaire Unan::Qui
z absolu
  )
```

Méthodes d'instance de la classe `User::UQuiz`

`<uquiz>.reponses`

`{Hash}` des réponses données au questionnaire. Avec en clé l'ID de la question et en valeur un Hash contenant :

```
qid:        {Fixnum} ID de la question dans la table des qu
estions

type:       {String(3)} Pour que JS puisse afficher les rép
onses
points:     {Fixnum} Total des points marqués
max:        {Fixnum} Maximum de points qu'il est possible d
e marquer
            pour cette question.
value:      {Fixnum|Array de Fixnum} ID de la réponse donné
e,  ou liste
            des IDs si c'est une question à réponses multip
les.
```

Instance `Unan::Quiz` du questionnaire de référence.

`<uquiz>.points`

Le nombre de points marqués à ce questionnaire. Ou nil.

`<uquiz>.max_points`

`Fixnum` . Le nombre de points maximum qu'on peut gagner à ce questionnaire.

`<uquiz>.note_sur_vingt`

`Float` . La note sur vingt pour le questionnaire.

> Calculée avec la méthode d'`Array` `sur_vingt` qui prend en premier argument la note totale et en deuxième argument le maximum de moints :

```
[points, max_points].sur_vingt(1)
```

`<uquiz>.quiz`

Instance `Unan::Quiz` du questionnaire original.

Suivi de l'auteur

Cron-job horaire

- Description de Cron::run
- Fichier log sûr

Description de Cron::run

Toutes les heures, un `cron-job` est lancé, conséquent, qui permet de contrôler les auteurs et, principalement, de :

- leur envoyer les nouveaux travaux,
- leur signaler tout retard,
- les encourager à poursuivre,
- leur accorder des bonus en cas de bon comportement.

> Noter que ce cron-job n'est pas exclusivement réservé au programme UN AN UN SCRIPT, il sert à toutes les tâches du site. Mais il est toujours lancé en “mode sans échec” pour ne jamais s'interrompre.

Ce fichier décrit le travail du cron-job qui suit les auteurs suivant le programme UN AN UN SCRIPT.

Tous les éléments du cron-job se trouvent dans le dossier `./CRON` qui doit être placé à la racine complète de l'hébergement (pas du site, donc avant le `www`).

Le cron est lancé par la ligne de commande :

```
0 * * * * ruby ./CRON/hour_cron.rb > /dev/null
```

C'est le fichier `hour_cron.rb` qui est le fichier `main.rb` .

Ce fichier appelle le module `./lib/required.rb` .

```
./hour_cron.rb -> ./lib/required.rb
```

Le fichier `./lib/required.rb` charge des librairies de cron puis appelle la méthode générale :

```
Cron::run
```

Synopsis :

```
    * Le Cron se place sur la racine du site.
    * Il charge toutes les librairies du site, comme si on
chargeait
      la page.
    * Il appelle ensuite la méthode
      `traitement_programme_un_an_un_script` qui va se char
ger du
      traitement des programmes UN AN UN SCRIPT.
      Cf. [Traitement des programmes un an un script](#trai
tementprogrammesunanunscript)
    * Il appelle ensuite la méthode `traitement_messages_fo
rum` qui va
      se charger du traitement des messages de forum, pour
avertir des
      nouvelles publications. Cf. [Traitement des messages
du forum](#traitementmessagesforum)
```

Fichier log sûr

Pendant tout le processus, on peut utiliser la méthode :

safed_log message

… pour enregistrer un message qui le sera à tout moment.

Tester le CRON-Job en local

Si on essaie d'appeler le fichier CRON par son adresse comme un module Ruby normal, ça ne peut pas fonctionner car CGI n'aura aucune valeur. Plutôt, il faut faire :

- Éditer le fichier crontab pour modifier la fréquence

```
    $> crontab -e
    Taper "a" pour passer en édition
    Mettre par exemple */10 pour un check toutes les 10
minutes
    :wq pour écrire et quitter le crontab
```

- Attendre le moment et vérifier dans le dossier `./CRON/log/`.

Noter que cette méthode ne génèrera un rapport/mail pour l'auteur que s'il change de PDay.

Les Pages de cours

Lien pour afficher/éditer/détruire une page de cours

Fonctionnellement, suivant le principe restfull du site, on utilise pour afficher une page de cours, pour l'éditer ou pour la détruire, ou pour modifier son texte, respectivement :

```
href="page_cours/<id>/show?in=unan"

href="page_cours/<id>/edit?in=unan_admin"

href="page_cours/<id>/destroy?in=unan_admin"

href="page_cours/<id>/edit_content?in=unan_admin"
```

Mais on préfèrera utiliser la méthode pratique utilisant les pointeurs :

```
page_cours(<hanlder>).link[ "<titre>"]
```

On obtient les liens par :

```
page_cours(<ref>).link            # lien pour afficher la p
age
                                  # read.erb
page_cours(<ref>).link(:edit)     # => lien pour éditer la
page
                                  # edit.erb
page_cours(<ref>).link(:destroy)  # => lien pour détruire l
a page
                                  # destroy.erb
```

Cf. Instance de page de cours pour le détail de cette méthode pratique.

Noter qu'on peut aussi utiliser les ID avec cette méthode, mais que c'est moins parlant. Par exemple :

```
page_cours(:introduction_au_programme).link
```

… est + parlant que :

```
page_cours(12).link
```

D'autre part, si on modifie la table des pages de cours, il suffira de changer la map des pages de cours pour corriger tous les liens d'un coup.

Instance de page de cours

Pour obtenir une instance de page de cours (`{Unan::Program::PageCours}`), on peut utiliser la méthode pratique `page_cours` .

Elle est définie dans le fichier principal des méthodes pratiques :

```
./objet/unan/lib/required/handy.rb
```

OBSOLÈTE. `page_cours` est maintenant une méthode-propriété qui retourne la page de cours définie d'après l'id de la rest-route.

Map des pages de cours

Ce que j'appelle la “map des pages de cours”, c'est la correspondance entre le handler (Symbol explicite) et l'ID de la table. C'est dans les données de la page, dans la base de données (unan_cold.pages_cours), qu'est définie cette correspondance.

Les Pointeurs de page

Dossier des pages de cours

Pour le moment, toutes les pages de cours, qu'elles soient propre au programme ou qu'elle provienne du livre ou de la collection Narration, se trouvent respectivement dans les dossiers :

```
./data/unan/ pages_cours/   unan/        Pages propres au p
rogramme
                            narration/   Pages du livre
                            cnarration/  Pages de la collec
tion
             pages_semidyn/ (même hiérarchie mais avec les
pages semi-
                             dynamiques qui seront vraiment
 chargées)
```

Enregistrement des LECTURES de la page de cours

Les lectures sont enregistrées dans la donnée `lectures` de la page (table

`pages_cours` de l'user). Pour le moment, c'est un simple Hash dont les clés sont les heures de lecture en secondes et la valeur est la même chose. Plus tard, on pourra imaginer que la valeur soit la fin de la lecture.

Construction des vues

- Emplacement des pages
- Constructeur de page

Toutes les vues sont dynamiques car elles contiennent souvent des éléments personnalisés, comme par exemple les exemples avec l'user courant.

Cependant, il y a certains traitements qui peuvent être lourds, comme par exemple le système de balisage (liens vers un exercice, vers un questionnaire, vers une autre page de cours, vers une tâche).

Faut-il imaginer des fichiers intermédiaires qui ne conserveraient que les éléments dynamiques.

```
PAGE ORIGINALE (le plus souvent ERB)

      ||
      ||      Traitement des liens-balises, etc. Ils sont r
emplacés par
      ||      des vrais liens.
      \/

PAGE SEMI-DYNAMIQUE (peut-être du contenu avec %{variable})

      ||
      ||      Traitement des quelques éléments dynamiques c
omme les
      ||      dates où les pseudos du lecteur de la page.
      \/

PAGE FINALE ENVOYÉE
```

Aucun contrôle n'est fait pour actualiser les pages semi-automatiques. Il faut le faire explicitement.

=> Un bouton-lien pour construire la page semi-dynamique.

Emplacement des pages

Les pages **originales** se trouvent dans le dossier `./data/unan/page_cours`.

Les pages **semi-dynamiques** se trouvent dans le dossier `./data/unan/page_semi_dyna`.

Les pages **finales** sont toujours envoyées à la volée.

Constructeur de page

C'est le module `./objet/unan_admin/page_cours/build.rb` qui se charge de construire la page semi-dynamique. Puis il redirige vers la page précédente, qui est certainement l'édition du contenu de la page.