SAÉ 1.03
Installation d'un poste de développement

Contexte

Vous travaillez dans une ESN[1] qui a besoin de produire de la documentation technique ainsi que de la documentation utilisateur sur les logiciels que vous développez pour vos clients.

Les clients sont livrés régulièrement d'une archive contenant le code source de leur produit en cours de développement, d'une documentation technique et d'une documentation utilisateur du produit. Tous ces documents sont des PDF.

Attendus

Environnement Dockerisé

Le produit du client est fictif et son développement ne fait donc pas partie des attendus.

Vous allez devoir travailler sur la partie de la chaîne concernant les outils de génération de documentation.

Cet environnement sera **Dockerisé**, **paramétrable** et **documenté**.

Les différents conteneurs devront s'échanger leur données de manière **automatisée** (sans intervention de l'utilisateur) par l'usage de **volumes managés**.

Même si vos postes de travail disposent de certains des outils listés plus loin, vous ne devez pas en faire usage et supposer que cette chaîne de traitement pourra être installée et utilisée, à l'avenir, sur un ordinateur qui ne disposera que de Docker. Vous ne devez donc utiliser que des conteneurs faisant tourner ces outils. C'est la philosophie Docker et ce sera donc aussi la vôtre pour cette SAÉ.

[1] **E**ntreprise de **S**ervice du **N**umérique

Langages et outils

Pour chaque étape, vous devrez respecter les langages et les outils demandés.

Vous aurez à votre disposition :

- PHP
- Bash
- Filtres Unix
- Regex (ça devrait réellement bien vous aider)
- Images Docker : **`sae103-php`**, **`sae103-html2pdf`**, **`clock`**

Attention, l'image **`sae103-html2pdf`** n'est pas la même que **`html2pdf`** du TP Docker !

Les images mises à votre disposition sont sur le dépôt Docker de l'IUT ainsi que sur le dépôt public de Docker. Si vous souhaitez travailler depuis l'extérieur, comme vous ferez implicitement appel au dépôt public, vous devez donc préfixer vos noms d'image par **`bigpapoo/`**, par exemple une image **`sae103-php`** à l'IUT devient **`bigpapoo/sae103-php`** chez vous.

Toutes les images disposent de **`bash`** et des filtres Unix. Pour les autres outils, seules les images dédiées disposent des outils en question (i.e : seule **`sae103-php`** dispose de PHP etc.).

Pour la SAÉ, utilisez toujours les images avec un tag **`latest`**. Il est possible que ces images évoluent un tout petit peu durant la période de SAÉ, mais vous serez tenus au courant si ça arrive et en utilisant **`latest`** vous êtes sûrs d'avoir la dernière version.

> Attention, si vous avez déjà fait un **`docker image pull`** d'une image, même en utilisant le tag **`latest`**, Docker ne fera pas automatiquement la mise à jour si une nouvelle version **`latest`** est mise à disposition. C'est à vous de forcer les choses si besoin en ajoutant une option **`--no-cache`** au lancement d'un **`docker image pull`**. Encore une fois, vous serez avertis si ça arrive.

Par défaut les images Docker fournies pour cette SAÉ sont prévues pour lancer un processus par défaut immédiatement et quitter ensuite, ce qui fait que vos conteneurs sont éphémères. Cependant, vous allez être amenés à vouloir lancer des traitements manuellement dans certains conteneurs. L'astuce consiste donc à créer des conteneurs en mode interactif et en demandant le lancement d'un **`bash`** au lieu de laisser le processus par défaut s'exécuter. Pour ce faire, ajoutez simplement **`bash`** en bout de commande de lancement du conteneur.

Documentation

Rien ne devra être laissé sans explication ni à l'appréciation du développeur qui aura la charge de mettre en œuvre la chaîne d'outils. Vous devrez prévoir le nécessaire (par script et/ou en documentant) pour le guider de A à Z dans le processus.

L'idée est d'avoir un environnement reproductible aisément pour que la chaîne d'opérations puisse être facilement redéployée plusieurs fois par an, sur n'importe quel poste d'un développeur de l'équipe, et sans nécessiter de connaissance des outils ni de formation préalable à ces outils.

Sources

Plus loin, vous aurez besoin d'un fichier de configuration. Vous le créerez avec ce contenu et vous le nommerez **`config`** :

```
CLIENT=Les Génies du Ponant
PRODUIT=Machine à courber les bananes
VERSION=2.7.4
```

Rendu client

Vous devrez générer une unique archive compressée en **`tar.gz`** contenant tous les PDF.

Voici la syntaxe pour créer une archive **`tar.gz`**, qui est un format incontournable dans le monde Unix/Linux[2], et non pas **`ZIP`** qui est issu du monde WIndows :

> **`tar czvf`** ***`nom_archive.tar.gz nom_dossier_a_archiver`***

Avec ces options (**`c`** pour **`c`**reate), le fichier d'archive est écrasé s'il existe déjà. Il y a des options pour ajouter à une archive déjà existante (pas nécessaire ici, voir le **`man`** si besoin).

Voici la syntaxe pour lister le contenu d'une archive **`tar.gz`** :

> **`tar tzvf`** ***`nom_archive.tar.gz nom_dossier_a_archiver`***

Le **`t`** est pour… une signification obscure que seuls les barbus connaissent… lis**`t`** ?

[2] **`tar`** signifie **`t`**ape **`ar`**chive, une archive sur bande (magnétique), oui ça remonte à très loin !

Voici enfin la syntaxe pour extraire le contenu d'une archive **tar.gz** :

> **tar xzvf *nom_archive.tar.gz nom_dossier_a_archiver***

Le **x** est pour e**x**tract.

Vous aurez noté que les options ne sont pas préfixées d'un **-** (tiret), c'est une exception à la plupart des autres commandes Unix. Cependant, il existe des versions de **tar** qui acceptent aussi l'usage du préfixe **-** (tiret) classique. L'usage est classiquement de ne pas utiliser le tiret.

L'option **v** est pour **v**erbose, vous pouvez l'omettre si vous voulez moins de logs durant l'exécution de la commande **tar**.

L'option **z** indique la compression, sans elle votre archive n'est pas compressée. Il s'agit du format **gzip** produit par la commande éponyme.

Enfin, l'option **f** pour **f**ile, permet d'indiquer qu'il faut créer l'archive sous forme d'un fichier dont le nom est donné dans l'argument suivant ce **f**.

Vous pourrez aussi trouver parfois l'extension **.tar.gz** raccourcie en **.tgz**.

Documentations technique et utilisateur

Vous allez générer des documentations **PDF** en passant d'abord par du **HTML**. Vous aurez d'ailleurs ces deux formats à rendre.

Vous allez générer du HTML par script s'exécutant dans un conteneur Docker, puis vous allez, dans un second temps, générer des PDF à partir de ces pages HTML. La génération du PDF se fera par un conteneur Docker dédié dont l'image vous est fournie.

Vous devez donc mettre en œuvre un formatage propre et élégant avec du CSS.

Que ce soit la documentation technique ou la documentation utilisateur, vous devez faire figurer ces informations sur chacune :

Page de couverture

La page de couverture de la documentation technique doit faire figurer

- Le nom du client (**config**)
- Le nom du produit développé (**config**)
- La version du produit (calculée par incrément).
- La date de génération de la documentation (calculée)

Les données indiquées **config** proviendront d'un fichier de configuration qui vous est fourni. Les données calculées le seront à la volée au moment de la génération de la documentation.

Versions

Note spéciale à propos du numéro de version d'un produit :

- Un numéro de version est de la forme : **`MAJOR.minor.build`**. Par exemple 2.7.14.

  Le **`MAJOR`** indique un numéro de version majeure : évolution importante du produit, prise en compte de nouvelles fonctionnalités, de changements d'interface utilisateur, etc.

  Le **`minor`** indique un numéro de version mineure : correction de bugs importants, pas de changement impactant l'utilisation ou les fonctionnalités du produit.

  Le **`build`** indique une version apportant de légères modifications : bugs légers et non fonctionnels comme des correctifs esthétiques par exemple.

- Le numéro de version doit être conservé pour que l'incrément puisse être calculé à la génération de documentation suivante. Le numéro de version sera conservé dans le fichier de configuration.
- Vous devrez prévoir un script permettant de modifier au choix (par une option passée à la commande) le **`MAJOR`**, le **`minor`** ou le **`build`**.
- Incrémenter le **`build`** conserve le **`MAJOR`** et le **`minor`**.
- Incrémenter le **`minor`** met à **`0`** le **`build`**.
- Incrémenter le **`MAJOR`** met à **`0`** le **`minor`** et le **`build`**.

Votre script de génération des documentations doit accepter un unique paramètre dont les valeurs possibles ont la signification suivante :

- **`--major`** : création d'une nouvelle version majeure
- **`--minor`** : création d'une nouvelle version mineure
- **`--build`** : création d'une nouvelle version build

En fonction de la valeur du paramètre, vous devez calculer le nouveau numéro de version en vous basant sur la valeur actuelle (lue dans le fichier **`config`**) et stocker ce nouveau numéro dans le fichier **`config`** pour servir de base de calcul au prochain lancement de la commande.

Les documentations doivent faire usage de ce numéro dans la page d'accueil ainsi que dans le nom de fichier généré et dans l'archive qui sera créée en fin de chaîne (voir le paragraphe **Archive finale**).

Documentation technique

Vous devrez construire une documentation à partir des commentaires trouvés dans les codes sources.

Vos codes sources seront des fichiers C.

Vous devrez fournir un jeu d'essai complet avec au moins 3 fichiers sources en **`.c`**.

Vous ne serez pas notés sur le contenu du code C utilisé comme exemple.

Voici les parties qu'un code source peut contenir et qui, quand elles apparaissent, devront faire être intégrées de la documentation technique générée.

En **rouge** ce qui permet de localiser les éléments à extraire pour construire la documentation. La documentation doit faire apparaître l'intégralité de l'information, c'est-à-dire le commentaire mais également l'élément associé au commentaire, bien entendu, sinon le commentaire manquera de contexte à la lecture de la documentation.

Vous allez voir que les commentaires sont généralement encadrés par **`/**...*/`**. Il s'agit d'extraire ces commentaires pour faire une présentation propre et agréable à lire. Cela signifie que les ouvertures et fermetures de commentaires, qui sont ici des tags spécifiques au langage C, devront être omises au profit d'un formatage en CSS. Vous avez toute liberté dans votre présentation mais vous serez aussi notés sur votre effort de présentation de l'information.

En-tête de code

Un en-tête qui doit apparaître dès la ligne 1 du code. Sans cet en-tête, le code n'est pas analysé et aucune documentation n'est générée :

```
/**
 * Un texte qui décrit ce que fait le fichier source.
 * Par exemple ça peut être une série de fonction
 * utilitaires pour le produit, ou encore le programme
 * complet d'un jeu de Puissance 4 ou d'un Sudoku !
 */
```

Defines

Les **`#define`** définissent des constantes. Exemple :

```
#define NB_NOTES  8   /** Nombre de notes du bulletin */
```

Tous les **`#define`** et leur commentaire associé doivent être groupés dans la documentation dans un chapitre **DEFINES**. Le mot clé **`#define`** n'est pas à afficher, seuls le nom, la valeur et le commentaire sont à afficher.

Structures

Les structures définissent de nouveaux types. Exemple :

```
typedef struct {
   char nom[SZ_NOM + 1];   /** Nom de l'étudiant·e */
   char groupe_td;         /** Lettre indiquant son groupe TD */
   int num_tp;             /** Numéro de son groupe TP (1 ou 2) */
} str_etu;  /** Structure d'un·e étudiant·e */
```

Toutes les structures et leur commentaire associé doivent être groupés dans la documentation dans un chapitre **STRUCTURES**. Elles doivent apparaître intégralement avec chaque attribut suivi de son commentaire. Le commentaire associé à la structure doit être placé avant le détail de la structure.

Variables globales

Les variables globales sont donc extérieures à toute fonction. Exemple :

```
int notes[NB_NOTES];   /** Liste des notes d'un étudiant */
```

Toutes les variables globales et leur commentaire associé doivent être groupés dans la documentation dans un chapitre **GLOBALES**.

Fonctions

Les fonctions ont plusieurs caractéristiques. Voici un exemple complet. Il peut y avoir certains informations absentes selon la nature de la fonction :

```
/**
 * \brief Ajoute étudiant·e (Résumé court du rôle de la fonction)
 * \detail Un descriptif détaillé de ce que fait la fonction
 * entrant dans les détails de ses caractéristiques,
 * des conditions particulières d'utilisation,
 * faisant apparaître éventuellement :
 * - un formatage sommaire
 * - sous forme de listes
 *
 * et de sauts de lignes
 * \return int Numéro d'identité de l'étudiant·e ajouté·e
 * \param char* nom Nom de l'étudiant·e.
 * \param char groupe_td Son groupe TD
 * \param int num_tp Son numéro de groupe TP
 */
int ajoute_etu(char *nom, char groupe_td, int num_tp) {
...
```

Les caractéristiques de chaque fonction ainsi que son nom doivent être groupées dans un chapitre **FONCTIONS**.

Documentation utilisateur

Vous allez faire d'une pierre deux coups : la documentation utilisateur que vous allez générer sera tout simplement celle de votre chaîne de production créée dans cette SAÉ !

Vous allez la créer au format Markdown et vous allez créer un script pour transformer un document Markdown en HTML. Il existe des outils qui font cela mais on vous demande d'en créer un vous-mêmes. Vous disposez de toutes les connaissances nécessaires pour y arriver, à l'aide de PHP et éventuellement de filtres Shell.

Voici les tags Markdown que vous devez supporter.

Votre documentation de SAÉ doit utiliser tous ces tags au moins une fois.

Titres

4 niveaux de titres : **#**, **##**, **###** et **####**, suivi d'un espace et d'un texte sur la même ligne, qui constitue le titre.

Un titre de niveau 1 est centré, les autres sont cadré à gauche.

Vous jouerez sur la taille et la graisse de la police de caractères.

Listes

Toute suite de lignes commençant par un tiret (**-**) suivi d'un espace. Si une ligne est trouvée sans tiret au départ, elle est ajoutée à la suite de la ligne précédente et ainsi de suite jusqu'à rencontrer une autre ligne débutant par un tiret+espace, ce qui constitue une nouvelle ligne de la liste, ou une ligne totalement vide (ou contenant uniquement des espaces), ce qui constitue alors la fin de la liste.

Tableaux

Une ligne commençant par un **|** (pipe) et suivi d'un certain nombre quelconque de tiret (**-**) et d'autres **|** (pipes). Le nombre d'intervalles **|---|...** indique le nombre de colonnes de détail.

Des lignes de détail dont les cellules sont aussi encadrées par des **|** (pipes).

Les lignes dont le nombre de cellules est inférieur aux colonnes de l'entête sont complétées par des champs vides.

Les cellules surnuméraires aux colonnes de l'entête sont simplement ignorées.

Le tableau se termine quand une ligne vide (ou avec uniquement des espaces) est rencontrée.

Exemple :

```
|Titre col1|Titre col2|Titre col3|
|-|-|-|
| 1 | Toto | toto@gmail.com |
| 2 | Lulu | lulu@hotmail.com |
```

Code

Un bloc de texte encadré par des ` (backticks ou accents graves) doit être écrit dans une fonte à chasse fixe. Ce bloc peut être au milieu d'une ligne normale. Exemple :

Utilisez la commande `./gendoc` pour générer la documentation.

doit afficher ceci :

Utilisez la commande **./gendoc** pour lancer la génération de la documentation.

Un bloc de texte encadré par des ``` (3 backticks) doit être écrit dans une fonte à chasse fixe et dans un paragraphe isolé.

Les espaces doivent obligatoirement être préservés ! En HTML, plusieurs espaces consécutifs sont considérés comme un seul espace par le navigateur. Vous devez donc convertir les espaces consécutifs en ** ** pour préserver leur nombre dans ces blocs de code.

Ces blocs sont généralement utilisés pour afficher du code ou des commandes à taper. Exemple :

````
```
   int main() {
   printf("Hello World!\n");
   return EXIT_SUCCESS;
}
```
````

doit afficher ceci :

```
int main() {
   printf("Hello World!\n");
   return EXIT_SUCCESS;
}
```

URL

Une URL est constituée d'un texte affiché et d'un lien vers un site Web. Les deux peuvent être identiques mais on pourra aussi préférer présenter un nom en clair dans le texte plutôt qu'un lien long et incompréhensible.

Le format est le suivant : **(Texte à afficher)[Lien]**

Les liens doivent donc être cliquables.

Textes

Tout ce qui n'entre pas dans une des autres catégories de tags.

Le texte doit être formaté dans des paragraphes. Un paragraphe se termine par une ligne vide (ou remplie uniquement d'espaces).

Formatages spéciaux

Dans les sections de texte et les tableaux, vous autoriserez le formatage des valeurs en gras (**`<b>...</b>`**) et/ou[3] en italique (**`<i>...</i>`**) à l'exclusion de tout autre type de formatage qui devra être ignoré. Cela signifie que toute autre ouverture ou fermeture de tag HTML (**`<xxx>`** ou **`</xxx>`**) sera simplement ignorée.

Dans les autres tags, aucun formatage n'est autorisé, vous devez donc opérer ce même nettoyage.

Archive finale

En fin de traitement, une archive finale au format **`tgz`** (**`tar + gzip`**) doit être produite, comme indiqué au paragraphe Rendu. Elle sera livrée au client et elle doit contenir :

- La documentation technique au formats PDF et HTML
- La documentation utilisateur au formats PDF et HTML
- Tous les codes sources **`.c`** ayant servi à la génération de la documentation technique

L'archive doit porter le nom du client (en minuscules et en remplaçant les espaces par des underscores) extrait du fichier **`config`** et doit avec un suffixe **`-<version>`** reflétant le numéro de version, ainsi qu'une extension **`.tgz`**.

Etapes

La SAÉ se déroule en 6 semaines et 2 périodes de livrables.

Vous déposerez vos livrables sur Moodle.

Semaines 1 et 2

Travaux, outils et livrables :

- Création de 3 fichiers C contenant toutes les parties commentées qui vous serviront de jeu de test. Vous pouvez soit récupérer du code d'un projet existant récupéré sur Github par exemple, soit un projet personnel.

  Quelle que soit la source des codes C utilisés, les commentaires doivent obligatoirement être en français et doivent cadrer avec le contexte (avec le code

[3] tags combinables entre eux.

source). Si vous utilisez du code d'un projet open source sur Github, inspirez-vous des commentaires déjà existants pour créer/traduire les vôtres.

Livrables : 3 fichiers en C nommés **`src1.c`**, **`src2.c`** et **`src3.c`**, commentés de manière complète, en utilisant dans chacun tous les types de commentaires possibles et cela plusieurs fois (au moins 2 de chaque).

- Création manuelle d'un modèle de documentation technique au format HTML faisant apparaître tous les chapitres possibles. Vous pouvez vous inspirer d'un des codes sources précédents pour remplir ce modèle.

  La raison d'être de ce modèle est de vous faciliter la construction automatique des réels documents techniques lors de la phase suivante (semaines 3 à 6)

  Livrable : un fichier **`DOC_TECHNIQUE.html`**.

- Création manuelle d'un modèle de documentation utilisateur au format Markdown faisant apparaître tous les types de tags possibles. Les textes insérés sont sans importance.

  La raison d'être de ce modèle est encore une fois de vous faciliter l'étape suivante (semaines 3 à 6)

  Livrable : un fichier **`DOC_UTILISATEUR.md`**.

- Une liste des tâches réalisées en semaines 1+2 et la répartition du travail individuel (en pourcentage) dans votre équipe.

  Livrable : un fichier **`TACHES_PERIODE_1.txt`**.

Semaines 3 à 6

Toutes les images Docker qui vous sont fournies ont un répertoire de travail par défaut qui est **`/work`**.

Comme tous les conteneurs vont s'échanger des données (c'est un chaînage de traitements), il faut donc que chacun d'eux monte ce volume (qu'ils vont donc se partager) sur son **`/work`** afin que ce que produit l'un puisse ensuite être visible et consommé par le suivant.

Ce volume doit être créé avant tout. Nommez-le **`sae103`**.

Travaux, outils et livrables :

- Générateur de documentation technique par l'analyse d'un lot de codes sources (fichiers **`.c`**) et la production d'une page HTML contenant la documentation technique formatée suivant les principes détaillés en introduction (paragraphe **Documentation technique**).

  Le générateur doit être écrit en PHP.

Les fichiers à traiter doivent être placés, par le développeur, au même endroit que le script par une commande de copie de l'hôte vers le conteneur.

Le script doit traiter tous les fichiers **`.c`** qu'il trouve.

Un unique fichier HTML doit être généré au même endroit que le script et doit s'appeler **`doc-tech-<version>.html`**. N'oubliez pas de prévoir la gestion des versions par le passage d'un paramètre à votre script, comme évoqué au paragraphe **Versions** au début du sujet.

Le générateur doit être lancé manuellement par le développeur en se connectant dans le conteneur et en lançant le script.

Image Docker à utiliser : **`sae103-php`**.

Livrable : un script autonome **`gendoc-tech.php`**.

- Générateur de documentation utilisateur par l'analyse d'un fichier Markdown (fichiers **`.md`**) et la production d'une page HTML contenant la documentation utilisateur formatée suivant les principes détaillés en introduction (paragraphe **Documentation utilisateur**).

  Le générateur doit être écrit en PHP.

  Le fichier à traiter doit être placé, par le développeur, au même endroit que le script par une commande de copie de l'hôte vers le conteneur. Il doit se nommer **`doc.md`**.

  Un fichier HTML doit être généré au même endroit que le script et doit s'appeler **`doc-user-<version>.html`**. N'oubliez pas de prévoir la gestion des versions par le passage d'un paramètre à votre script, comme évoqué au paragraphe **Versions** au début du sujet.

  Le générateur doit être lancé manuellement par le développeur en se connectant dans le conteneur et en lançant le script.

  Image Docker à utiliser : **`sae103-php`**.

  Livrable : un script autonome **`gendoc-user.php`**.

- Générateur de PDF à partir des fichiers HTML créés par **`gendoc-tech.php`** et **`gendoc-user.php`**.

  Image Docker à utiliser : **`sae103-html2pdf`**.

  Syntaxe à utiliser (dans le conteneur) pour générer un PDF à partir d'un document HTML :

```
html2pdf fichier.html fichier.pdf
```

Il faudra donc prévoir deux appels à cette commande pour générer les deux documentations.

Livrables : un exemple des deux fichiers **`doc-tech-<version>.html`** et **`doc-user-<version>.html`**.

- Générateur d'archive finale comme indiqué au paragraphe **Archive finale**.

  Une fois la génération de l'archive effectuée, sa récupération se fait par une copie du conteneur vers l'hôte

Automatisation

A ce stade vous devez avoir un chaînage de plusieurs actions faites dans plusieurs conteneurs. Pour finir proprement, il reste à automatiser tout cela.

Un petit détail peut vous bloquer car l'automatisation va se faire sans intervention humaine et donc sans conteneur en mode interactif. Or, sans conteneur interactif, impossible de copier des fichiers dans le volume **`sae103`** car il doit être monté dans un conteneur qui reste actif si on souhaite pouvoir y accéder depuis l'hôte.

L'astuce complémentaire qui va vous permettre de conserver le volume attaché à un conteneur durant tout le traitement est de créer un conteneur supplémentaire qui reste en vie sans rien faire. Pour ce faire, vous allez utiliser un conteneur **`clock`** (vous vous rappelez de lui ?) dont la vie est d'égrener le temps sans fin. Ca pourrait être n'importe quelle autre image dès lors qu'il s'agit d'une image qui ne s'arrête pas de fonctionner.

Le dernier livrable est donc un script **`bash`**, nommé **`sae103.bash`**, qui sera lancé sur l'hôte et qui effectue les actions suivantes :

- Création du volume **`sae103`**
- Lancement d'un conteneur **`clock`** en mode détaché, en lui donnant le nom **`sae103-forever`** et en lui montant le volume **`sae103`**
- Copie des fichiers **`.c`** dans le volume **`sae103`** en utilisant **`sae103-forever`** comme conteneur cible.
- Lancement de tous les autres traitements les uns après les autres, mais en mode non interactif cette fois-ci
- Récupération de l'archive finale depuis le volume **`sae103`**, en utilisant encore **`sae103-forever`** comme conteneur source cette fois-ci.
- Arrêt du conteneur **`sae103-forever`**
- Suppression du volume **`sae103`**

Liste des tâches

Il faudra aussi rendre une liste des tâches réalisées en semaines 3 à 6 et la répartition du travail individuel (en pourcentage) dans votre équipe.

Livrable : un fichier **`TACHES_PERIODE_2.txt`**.