

Table des matières

1	Explications et planning	2
2	Présentation du site	2
2.1	Fonctionnement général	3
2.2	Graphisme	4
3	Considérations techniques	4
3.1	Bases de données	4
3.2	Authentification	5
3.2.1	Version 1 du site : pseudo-identification	5
3.2.2	Version 2 du site : vraie identification	5
3.2.3	Aspect technique	6
3.3	Templates	6
3.4	Divers	6
4	Fonctionnement détaillé	6
4.1	CSS	6
4.2	Page de bienvenue	7
4.3	Menu	7
4.4	Connexion/déconnexion	7
4.5	Créer un compte client	7
4.6	Éditer/modifier son profil	8
4.7	Lister les produits du magasin	8
4.8	Gérer son panier	9
4.9	Gérer les utilisateurs	10
4.10	Ajouter un produit	10
4.11	Ajouter un administrateur	10
4.12	Récapitulatif	10
4.13	Service	11
4.14	Validation des entités	11
4.15	Mail	11
5	Travail à rendre	11

1 Explications et planning

Ce devoir est à faire en binôme exclusivement.

La date de remise du projet est :

- mardi 1 avril 2025 à 18h15
- le site de dépôt se fermera automatiquement et pas de retard autorisé

Le projet sera déposé sur la plate-forme UPDAGO, sous forme d'un fichier archive au format *tar* compressé avec l'utilitaire *gzip*. Un seul des deux membres (le premier dans l'alphabet) du binôme déposera l'archive ; l'autre membre déposera un fichier texte (peu importe le contenu) nommé *binome_nom1_nom2.txt* (où vous remplacez *nom1* et *nom2* de manière adéquate, par exemple *binome_subrenat_zrour.txt*).

Un devoir sur table aura lieu après la remise du projet.

Le nom de l'archive sera IMPÉRATIVEMENT composé de vos noms de famille en minuscules dans l'ordre lexicographique, d'un underscore, du mot "projet", par exemple *subrenat_zrour_projet*, suivi des extensions classiques (i.e. ".tar.gz").

Le désarchivage devra créer, dans le répertoire courant, un répertoire s'appelant : *PROJET*.

Ces directives sont à respecter SCRUPULEUSEMENT (à la minuscule/majuscule près). Un script-shell est à votre disposition pour vérifier votre archive ; il peut être exécuté sous Linux ou *git-bash*.

Les langages utilisés sont obligatoirement :

- HTML (soyez strict dans votre syntaxe) et Twig
- CSS (non ou peu utilisé en fait dans ce projet)
- JavaScript (non ou peu utilisé en fait dans ce projet)
- PHP version de 8.2 ou plus
- SQL avec le SGBD SQLite (comme vous utiliserez Doctrine, il n'y aura pas de requête SQL explicite à écrire)
- le framework Symfony avec la version 7.2 impérativement

On vous demande d'utiliser *git* pour avoir une trace des différentes versions de votre logiciel. Le dépôt *git* concerne uniquement le code du site web. Vous pouvez avoir un repository distant sur la plate-forme de votre choix (et c'est vivement conseillé).

Le répertoire *.git* devra être dans l'archive rendue. Il permettra de voir les contributions de chaque membre du binôme. Les répertoires *var* et *vendor* ne doivent pas être suivis par *git* (ce qui le cas par défaut si vous ne corrompez pas les fichiers *.gitignore*)

Pour insister :

- la version de Symfony est impérative¹
- une archive ne passant pas avec succès le vérificateur sera considérée comme inexploitable et sera pas corrigée² ; mais vous pouvez demander de l'aide pour la faire.
- plus généralement, si vous ne respectez pas les directives vous serez sanctionnés.

Il est interdit d'utiliser des programmes de génération de sites.

Vous n'êtes pas autorisés à utiliser des bibliothèques ou des composants qui ne sont pas de votre cru, hormis les bibliothèques habituelles et bien entendu hormis Symfony. En cas de doute, demandez l'autorisation.

Il vous est demandé un travail précis. Il est inutile de faire plus que ce qui est demandé. Dans le meilleur des cas le surplus sera ignoré, et dans le pire des cas il sera sanctionné.

2 Présentation du site

Si vous n'avez pas lu la section précédente, il est vivement recommandé de le faire.

1. peu importe que vous pensiez que c'est un mauvais choix (ou que vous pensiez quelque chose de moins poli) : un client vous paie (très peu dans votre cas) pour un produit et vous lui donnez un autre produit cela ne pourra que mal se passer.

2. cf. note précédente

Nous présentons ici les possibilités du site sans beaucoup de considérations techniques (bien qu'il y en ait quelques-unes).

2.1 Fonctionnement général

Le but est de créer un site Web classique permettant de manipuler une base de données. Le projet sera la gestion de ventes en ligne.

Évidemment, vous être libre de choisir les produits vendus tant que vous restez dans la légalité et ne portez pas atteinte aux bonnes moeurs.

Il y a quatre types d'utilisateurs connectés :

- *non authentifié* : il accède au site sans mot de passe et ne peut que s'authentifier ou créer un compte,
- *authentifié client* : il a accès à la mise à jour de son profil et à la gestion de ses commandes.
- *authentifié administrateur* : il a les droits d'un client, et il gère en plus les clients.
- *authentifié super-administrateur* : il gère uniquement les administrateurs (et rien d'autre).

Le fait qu'un internaute soit un anonyme, un client ou un (super-)administrateur est précisé dans la base de données (cf. section [3.2 Authentification](#)).

À un utilisateur authentifiable sont associés :

- un login,
- un mot de passe (cf. section [3.2](#)),
- un nom, un prénom et une date de naissance,
- le fait qu'il soit administrateur ou non (cf. section [3.2](#)),
- zéro ou un pays (cf. table ci-dessous) d'appartenance.

À un produit sont associés :

- un libellé,
- un prix unitaire (en euros avec un nombre réel),
- une quantité en stock.

À un pays sont associés :

- un nom,
- un code sur 2 caractères.

Les produits sont vendus par potentiellement plusieurs pays, et inversement un pays peut vendre plusieurs produits (mais pour simplifier, le prix du produit est le même dans tous les pays et est stocké dans la table des produits).

Un panier, associé à un client, contient la liste des produits en instance de commande, avec la quantité désirée pour chacun. Un panier est stocké dans la base de données (et non dans des cookies). Il faut une seule table pour stocker les contenus des paniers de tous les clients

Un visiteur non authentifié peut :

- se connecter,
- créer un compte.

Un client authentifié non administrateur peut :

- se déconnecter
- éditer et modifier son profil,
- lister le contenu du magasin et ajouter des produits à son panier,
- gérer son panier (acheter ou vider).

Un administrateur peut, en plus des possibilités d'un client :

- gérer les utilisateurs
- ajouter un produit dans la base

Un super-administrateur peut uniquement :

- se déconnecter
- gérer les administrateurs

Donc le menu (cf. section suivante et section 4.3) doit s'adapter au profil de l'utilisateur.

2.2 Graphisme

On demande très peu de style de présentation CSS. Le site sera moche, et *osef*.

Toutes les pages du site auront :

- un bandeau supérieur fixe (i.e. presque toujours le même quelle que soit la page affichée) qui est une image. Pour être exact l'image dépendra du fait qu'on soit connecté en mode authentifié ou non, et du fait qu'on soit administrateur ou non (version 2 uniquement). Prenez une image qui ne prend pas trop d'espace disque (inférieur à 50 Ko).
- un menu listant les actions possibles (qui dépendent du profil de l'utilisateur).
- le corps de la page permet d'afficher les informations et de faire interagir l'utilisateur.
- un bandeau de bas de page fixe (i.e. toujours le même quelle que soit la page affichée) qui est une image (de même pas trop volumineuse).

La première page du site (route "/") propose un message de bienvenue et indique si l'on est un visiteur anonyme, un client ou un (super-)administrateur. Notamment le formulaire de connexion ne s'affiche pas ici.

Les actions possibles, pour un utilisateur, apparaissent soit dans le menu (sous forme de liens), soit dans le corps de la page (validation d'un formulaire ou liens). Un clic sur une action fait apparaître les informations, formulaires, ... dans le corps du site. Une action découle donc toujours directement de l'intervention de l'utilisateur.

Beaucoup de choses sont simplifiées (inutile d'essayer de réaliser un vrai site). Notamment faites des interfaces simples même si elles ne sont pas très ergonomiques.

3 Considérations techniques

3.1 Bases de données

Le SGBD SQLite est imposé. Le fichier contenant la base (extension *.db* obligatoire, avec un nom de votre choix) doit se trouver dans un répertoire nommé *bd_commerce* à la racine de votre site.

Tous les noms de vos tables seront préfixés par *"l3_"*, mais dans les noms des classes de Doctrine, ce préfixe ne doit pas apparaître.

La base de données se déduit directement des explications qui précèdent (ou qui suivent dans la section sur les explications détaillées).

Vous devez modéliser les relations entre les entités via Doctrine.

Pour la relation entre les pays et les produits, on impose une relation *ManyToMany* dans laquelle le nom de la table de jointure et les noms des deux clés étrangères doivent apparaître explicitement..

Le hashage des mots de passe sera géré par le composant *security* (cf. section 3.2).

Attention les doublons ne sont pas autorisés pour le couple (nom, prénom) dans la table des utilisateurs ; de même le login doit être unique (il faut des contraintes dans le SGBD),

Créez au moins les 4 utilisateurs suivants :

- *sadmin* (mot de passe *nimdas*) qui est un super-administrateur du site,
- *gilles* (mot de passe *sellig*), qui est un administrateur,
- *rita* (mot de passe *atir*), qui est une cliente,
- *boumediene* (mot de passe *eneidemuob*), qui est un client.

Alimentez votre base avec suffisamment de données pour faire des tests représentatifs. Le fichier `.db` devra être présent dans l'archive, mais ne doit pas être versionné dans `git`. Le fichier `.db` devra pouvoir être réinitialisé via les *fixtures*.

Dans la base, un champ non renseigné (si c'est autorisé) devra avoir la valeur `NULL` et non pas contenir une chaîne vide.

3.2 Authentification

La gestion de l'authentification, avec le composant `security` de Symfony, peut être faite dans un second temps (i.e. vers la fin du projet).

- Avantage : cela vous permettra d'avancer tout de suite sur le projet, au prix d'une petite config cependant.
- Inconvénient : il faudra repasser sur le site pour gérer les droits.
- Avertissement : avant d'installer la sécurité, faites une sauvegarde de votre site ! Avec une mauvaise configuration, il est possible de rendre le site non opérationnel.

Si vous activez immédiatement l'authentification (connexion, rôles), vous gagnerez du temps sur la totalité du projet, mais cela signifie vous devez maîtriser l'authentification presque dès le début (alors que c'est le dernier TP).

Quel que soit votre choix, pour créer la table des utilisateurs, il faut impérativement utiliser la commande `make:user` (cf. ci-dessous) au lieu de `make:entity`.

Pour compléter la table des utilisateurs avec d'autres données, vous pouvez utiliser la commande `make:entity`.

Pour les autres tables, vous utilisez la commande habituelle `make:entity`.

Des indications sur la commande `make:user` qui va poser quelques questions :

- choix du nom de la table : vous êtes libre
- veut-on stocker les données en base de données : répondre "oui"
- sur quel champ se fait l'authentification : choisir le login de l'utilisateur
- veut-on hasher les mots de passe : la réponse est "oui"

3.2.1 Version 1 du site : pseudo-identification

Si vous voulez que le site fonctionne sans authentification, il faut tout de même que l'on sache qui est authentifié, ce qui bien sûr est contradictoire.

Pour être clair : comme on ne gère pas les rôles, n'importe quel utilisateur est à la fois client, administrateur et super-administrateur.

On utilisera une variable globale (cf. ci-dessous), en dur dans le code, pour savoir qui est connecté. Autrement dit, pour changer l'utilisateur connecté, il faudra modifier le code du site.

Cette variable contiendra tout simplement l'identifiant (clé primaire de la table ci-dessus) de l'utilisateur. Et on supposera que sa valeur est valide sans qu'il soit nécessaire de le vérifier.

On insiste sur le fait que pour changer d'utilisateur il faut donc mettre à jour ce paramètre et donc modifier le code du site³.

Dans cette version on ne gère pas la connexion et la déconnexion, ni les autorisations (rôles).

3.2.2 Version 2 du site : vraie identification

On utilise pleinement l'entité des utilisateurs générée par `make:user`. Par rapport à la version précédente il y a donc en plus le mot de passe et les rôles.

Il faut donc vérifier finement les droits d'accès aux différentes parties du site.

En outre il faut gérer les utilisateurs anonymes, la connexion et la déconnexion.

Note : la variable globale (si vous êtes passé par la version 1) peut être laissée dans le code, mais ne sera plus utilisée.

3. Est-il utile de rappeler qu'il n'est pas demandé de construire un site opérationnel (bis repetita).

3.2.3 Aspect technique

Pour créer un paramètre global accessible dans les contrôleurs (et uniquement ceux-ci), il faut modifier le fichier *config/services.yaml* et ajouter une ligne dans la section *parameters*.

Voici le contenu du fichier pour créer une variable *moi* contenant un prénom :

```
parameters:
    moi: 'Jarod'
```

Pour récupérer cette variable dans un contrôleur, le code est :

```
$prenom = $this->getParameter('moi');
```

Donc cette constante *moi* n'est récupérable que par les contrôleurs et non par Twig : ce sera aux contrôleurs de passer les arguments nécessaires aux vues si nécessaire.

Pour information, si on veut créer des paramètres globaux accessibles par les templates Twig, il faut modifier le fichier *config/package/twig.yaml*. Mais ce n'est pas utile pour ce projet.

3.3 Templates

Les templates Twig doivent former un héritage à 3 niveaux.

Le premier niveau correspond à une page très générale (i.e. a priori valable pour n'importe quel site), tel que le template fourni d'office par Symfony (*base.html.twig*) ; mais on vous demande une version personnelle (et a priori plus simple) de ce template.

Le deuxième niveau est propre à votre site de vente : entête, pied de page, menu, ...

Le troisième niveau correspond aux vues associées aux actions.

Les templates de premier et deuxième niveaux ne seront pas à la racine du répertoire *templates* mais dans un sous-répertoire dédié. Dans ce sous-répertoire il pourra y avoir d'autres templates (comme par exemple le code du pied de page).

3.4 Divers

Toute duplication de code est à éviter.

Un formulaire mal rempli devra être réaffiché avec la mémoire des saisies précédentes de l'utilisateur.

Comme on impose d'utiliser les formulaires de Symfony, ce mécanisme est géré d'office.

On insiste sur le fait que vous ne devez pas écrire des formulaires à la main (sauf celui pour s'authentifier).

Votre site doit pouvoir être installé dans n'importe quel répertoire d'un serveur. Vous ne devez pas mettre d'URL absolue en dur dans le code : utilisez les mécanismes de Symfony. De même il est peu probable qu'il y ait des chemins relatifs en dur : si dans un chemin il y a le motif *“..”*, votre code est vraisemblablement incorrect.

4 Fonctionnement détaillé

Attention toutes les actions ne sont peut-être pas décrites ici : on ne décrit que celles qui apparaissent dans le menu, plus quelques autres.

4.1 CSS

On veut une feuille CSS commune pour tout le site avec une seule règle (par exemple une couleur de fond pour la page). On entend par *“tout le site”* une feuille utilisée par toutes les vues (celles déjà écrites ou les futures) : on se place ici au premier niveau de la hiérarchie des templates.

On veut également une feuille CSS utilisée (*a priori*) uniquement par le site de vente (par exemple pour encadrer les titres) : on se place ici au deuxième niveau de la hiérarchie des templates. Cette feuille de style doit compléter la première (celle précisée dans le template de niveau 1).

Enfin chaque vue du site aura possibilité de rajouter sa propre feuille CSS (par exemple pour colorer les titres) : vous choisirez la page de bienvenue pour profiter de cette possibilité. Cette feuille de style devra compléter celle précisée dans le template de niveau 2.

Sinon il n'y a pas d'autre CSS à faire. Et si vous en rajoutez, il ne faut pas que cela perturbe la navigation ou cochoonne le code.

Exemple de problème : l'image de bas de page se retrouve en plein milieu de l'écran parce que je n'ai pas la même résolution d'écran que vous et je n'ai pas accès aux liens en dessous.

4.2 Page de bienvenue

Une page est dédiée à l'accueil du site et sa vue contient uniquement un message poli de bienvenue avec la nature du visiteur (anonyme, client, administrateur ou super-administrateur) et son pays s'il est connu. Le formulaire de connexion n'est pas sur cette page.

Plus exactement la vue hérite du template de niveau 2 (et donc de celui de niveau 1) et les éléments de ces deux templates seront donc affichés.

L'URL associée à cette page est “/”.

4.3 Menu

La division *menu* affiche deux informations :

- le nombre d'articles dans le panier du client (rien si le visiteur est anonyme ou super-administrateur)
- le menu proprement dit qui propose une liste de liens.

Définition du nombre d'articles : le nombre total d'objets. Par exemple si dans le panier il y a 5 vélos et 3 voitures, le nombre attendu est 8.

Comme il a été dit, chaque type d'utilisateur a des actions possibles bien précises. Donc le menu s'adapte à la nature et aux droits de l'internaute (uniquement pour la version 2 du site).

Si on est en mode administrateur, il y a un lien supplémentaire vers :

<https://fr.wikipedia.org/wiki/Pong>

4.4 Connexion/déconnexion

Ces actions concernent uniquement la version 2 du site.

Seul un utilisateur non authentifié peut se connecter via un formulaire classique.

La déconnexion redirige l'internaute vers la page d'accueil avec un message flash.

L'habitude est d'écrire le formulaire en dur (sans utiliser le composant *form*). Le formulaire peut être généré automatiquement par le composant *security*.

4.5 Créer un compte client

Cette action concerne un internaute non authentifié.

On arrive sur un formulaire classique (géré par les outils de Symfony). Pensez à permettre de choisir un pays d'appartenance.

Attention le compte créé est obligatoirement un compte client.

Si le formulaire est correct, la base de données est mise à jour et on se retrouve sur la page de bienvenue (avec un message flash) ; sinon le formulaire réapparaît avec un message indiquant le dysfonctionnement (champ vide, nom déjà utilisé, ...). Créer un compte n'authentifie pas l'utilisateur.

Rappel : il est imposé d'utiliser les formulaires proposés par Symfony, et ce sera la cas pour tous les autres formulaires.

4.6 Éditer/modifier son profil

Cette action concerne un internaute authentifié.

Cette entrée du menu conduit à une page similaire à la création d'un compte, si ce n'est que les champs sont pré-remplis avec les données de l'utilisateur.

Si le client quitte la page en cliquant sur un autre lien du menu, rien ne se passe.

S'il modifie les champs et valide, alors son profil est enregistré dans la base de données si les données sont correctes.

On fait exactement les mêmes vérifications que pour l'ajout d'un compte.

Si le formulaire n'est pas valide, on le réaffiche avec les messages d'erreurs adéquats.

Si le formulaire est valide, on est redirigé, systématiquement avec un message flash, sur :

- dans le cas d'un super-administrateur, la page d'accueil
- sinon la page qui liste les produits

NB : un utilisateur peut changer son mot de passe, son nom, son prénom, sa date de naissance et même son login, mais ne peut pas changer son statut (administrateur ou non).

Pour simplifier le code :

- à l'affichage du formulaire, le champ de saisie du mot de passe est vide. Et il faut systématiquement resaisir une valeur
- et on ne demande pas l'ancienne valeur du mot de passe

Est-il utile de rappeler qu'il faut utiliser les formulaires proposés par Symfony ?

4.7 Lister les produits du magasin

Cette action concerne un internaute authentifié client (et donc aussi administrateur, mais pas super-administrateur).

Il s'agit de lister ici tous les produits de la base les uns en dessous des autres.

Pour chacun on a les éléments suivants à l'écran sur une ligne :

- libellé
- prix unitaire
- quantité n en stock
- une liste déroulante avec les nombres de $-m$ à n permettant de choisir la quantité que l'on veut commander ou décommander (m étant le nombre de produits dans le panier, et n le nombre de produits encore en stock)⁴. L'entrée présélectionnée doit être "0". Attention, si la quantité dans le panier est nulle et si la quantité en stock est également nulle, le produit apparaît dans la page mais sans menu déroulant.
- un bouton *modifier* qui permet de modifier le produit et son nombre d'occurrences dans le panier. S'il n'y a pas de menu déroulant, alors il n'y a pas de bouton de modification.

Il y a donc autant de formulaires qu'il y a de produits dans la base de données (ou moins si des produits ne sont pas en stock).

Comme chaque formulaire est envoyé à la même page, il faut envoyer deux informations :

- le produit concerné (un champ caché ou un paramètre *GET* par exemple)
- la quantité (le menu déroulant)
- mais pas l'utilisateur concerné car c'est obligatoirement la personne connectée.

4. Notez qu'on ne gère pas le fait que plusieurs accès concurrents pourraient conduire à ce que le nombre de produits commandés soit plus grand que celui en stock.

Comme il n’y a pas de JavaScript, modifier une quantité revient à envoyer un formulaire. Après validation, on reste sur la même page, avec toute la liste déroulante réinitialisée pour éventuellement compléter la commande.

Le fait d’ajouter ou enlever des produits dans le panier, diminue ou augmente leurs quantités dans la base de données (comme s’ils avaient été réellement commandés ou restitués).

Tous ces formulaires sont gérés par le mécanisme fourni par Symfony. Attention tous les champs (la liste déroulante) ne sont pas directement liés à une colonne de la base de données.

On considère qu’ils ne peuvent pas être mal remplis ; mais si le contrôleur gérant le formulaire détecte une erreur, une exception pourra être levée en guise de traitement.

Cf. fichier *liste_deroulante_bis.html* pour un exemple de présentation de cette page.

La gestion du panier est expliquée ci-dessous.

4.8 Gérer son panier

Cette action concerne un internaute authentifié client (et donc aussi administrateur, mais pas super-administrateur).

Cette page affiche la liste des produits, avec la quantité commandée et le prix résultant pour chaque article (cf. fichier *panier.html*).

Notons que si le même produit a été ajouté en plusieurs fois, il ne doit apparaître qu’une seule fois dans la liste ; et également dans la partie de la base de données qui stocke les paniers.

Il n’y a techniquement qu’une seule table dans la base pour stocker les paniers de tous les clients : on peut voir cette table comme une table de jointure entre les utilisateurs et les produits.

Au bas de la page apparaissent :

- le prix total des commandes,
- un bouton/liens “Commander”,
- un bouton/liens “Vider”.

Il est également possible de supprimer un produit du panier (par exemple via un bouton/liens à côté de chaque produit).

Cette page ne contient que des boutons/liens et non pas des formulaires.

Notez qu’il n’est pas possible (i.e. qu’il n’est pas demandé d’implémenter), dans cette page, de modifier la quantité commandée d’un produit.

Si vous voulez modifier la quantité d’un produit, il faut :

- retourner sur la page listant les produits,
- commander le surplus ou le “surmoins” (et non pas la nouvelle quantité),
- retourner sur la page gérant le panier.

Bref il s’agit d’un contre-exemple de bonne ergonomie^{5 6}.

L’action de supprimer un produit (liens à droite de chaque ligne) le fait disparaître du panier et remet également la quantité en stock à jour.

Le fait de vider le panier, outre le fait de faire disparaître tous les produits du panier, remet à jour toutes les quantités en stock.

Le fait de “commander” supprime tous les produits du panier, sans remettre à jour les quantités en stock puisque les produits vont être expédiés^{7 8}.

Et dans tous les cas on revient sur la même vue, qui affichera donc un panier remis à jour.

Attention, on insiste sur le fait que les paniers sont stockés en base de données, et non pas dans des cookies, pour ne pas être perdus si on change d’ordinateur.

5. Est-il utile de rappeler qu’il n’est pas demandé de construire un site opérationnel.

6. cf. l’introduction si vous décidez d’implémenter, directement dans la page de gestion du panier, la possibilité de modifier la quantité commandée.

7. On ne vous demande pas de gérer l’expédition des produits.

8. On ne vous demande pas de gérer la phase de paiement qui est donc tout simplement ignorée ; il est clair que votre entreprise ne sera pas économiquement viable.

4.9 Gérer les utilisateurs

Cette action n'est accessible que par un administrateur non super-administrateur.

On ne gère que la suppression des utilisateurs.

Le résultat débouche sur une vue qui affiche tous les utilisateurs les uns en dessous des autres :

- login,
- mot de passe (hashé⁹).
- nom et prénom,
- date de naissance,
- statut (client, admin ou super-admin)
- pays d'appartenance

Il est possible de supprimer un utilisateur (attention à vider proprement son panier s'il a des commandes en cours) avec par exemple un bouton à côté de chacun.

Les super-administrateurs apparaissent mais sans possibilité de les supprimer. D'ailleurs l'action de suppression devra vérifier qu'on n'essaie pas de supprimer un super-administrateur (par modification directe de l'URL).

Il ne doit pas être possible de supprimer l'utilisateur loggué. Comme pour la suppression des super-administrateurs, même si l'interface ne le permet pas, l'action devra vérifier qu'on ne supprime pas l'utilisateur courant.

Cette page ne contient que des boutons/liens et non pas des formulaires.

4.10 Ajouter un produit

Cette action n'est accessible que par un administrateur non super-administrateur.

On arrive sur un formulaire classique.

Si le formulaire est correct, la base de données est mise à jour et on se retrouve sur la page de bienvenue (avec un message flash) ; sinon le formulaire réapparaît avec un message indiquant le dysfonctionnement (champ vide, stock négatif, ...).

On ne demande pas de gérer les liens entre les produits et des pays ; le produit créé ne sera donc lié à aucun pays.

Il est imposé d'utiliser les formulaires proposés par Symfony.

On ne demande pas de gérer la suppression d'un produit.

4.11 Ajouter un administrateur

Cette action n'est accessible que par un super-administrateur.

On arrive sur un formulaire classique.

Il est imposé d'utiliser les formulaires proposés par Symfony.

4.12 Récapitulatif

Un client non authentifié a uniquement les entrées suivantes dans son menu :

- connexion
- aller à la page d'accueil
- créer un compte

Un client authentifié a uniquement les entrées suivantes dans son menu :

- déconnexion

9. il ne peut en être autrement

- aller à la page d'accueil
- gérer le profil
- lister les produits
- gérer le panier

Un administrateur a en outre les entrées suivantes :

- gérer les clients
- créer un produit
- un lien vers un site extérieur

Un super-administrateur a uniquement les entrées suivantes dans son menu :

- déconnexion
- aller à la page d'accueil
- gérer son profil
- créer un administrateur

Et un internaute cherchant à accéder à une page interdite en modifiant l'URL soit être rejeté. Notamment il ne doit pas pouvoir modifier le profil d'un autre utilisateur.

4.13 Service

Créez un service (au sens de Symfony) qui contient deux méthodes :

- le constructeur qui a comme paramètre l'entity manager (fourni par injection de dépendance) et le stocke dans un membre de la classe.
- une fonction prenant un entier en paramètre. Cet entier est l'*id* d'un client. La fonction renvoie le nombre d'articles dans le panier pour le client.

Cette dernière fonction doit être utilisée dans l'action qui gère le menu.

4.14 Validation des entités

Les formulaires hydratent des entités, et les données hydratées doivent respecter certaines contraintes.

Les vérifications se feront, dans les entités, via les directives *Assert*.

Voici une liste minimale des vérifications à faire :

- une quantité en stock doit être positive
- de même pour un prix unitaire
- la longueur du mot de passe (non hashé) doit être comprise entre 3 et 30
- le mot de passe (non hashé) ne doit pas être le même que le login.
- un login d'utilisateur est unique dans la base.

4.15 Mail

Cette partie est facultative et ne sera pas prise en compte dans l'évaluation.

Dans l'affichage des produits, ajoutez un lien. Ce lien permet d'envoyer un mail avec les outils de *Symfony*. Ce mail contiendra le nombre de produits dans la base.

Faites attention de ne pas laisser un mot de passe personnel dans le code que vous rendrez.

5 Travail à rendre

Documents à rendre (tous dans l'archive) :

- un fichier texte “ls-R.txt” qui contient la liste des noms des fichiers que vous avez créés ou modifiés (avec les noms de leurs répertoires) dans le répertoire *config*; inutile de détailler les autres répertoires.
- le code du site; n’oubliez pas de laisser le répertoire *.git*, mais il ne doit pas y avoir les répertoires *var* et *vendor*.
- un rapport au format pdf, nommé “rapport.pdf” (disons 2 ou 3 pages hors page de garde et table des matières) qui contient :
 - le schéma de la base (avec les cardinalités),
 - l’organisation de votre code : la hiérarchie et la liste des classes avec des explications si nécessaire.
 - l’explication (i.e. un tutoriel) pour créer un service dans Symfony.
 - (facultatif) comment on envoie un mail avec *Symfony* .
 - des points particuliers sur ce que vous avez fait sur le framework.

N’hésitez pas à préciser ce qui ne marche pas correctement dans votre code.
 Soignez l’orthographe, la grammaire, ...

Les fichiers “rapport.pdf” et “ls-R.txt” doivent être directement dans le dossier *PROJET* (racine de votre archive). Le code du site sera dans un sous-répertoire nommé *site* qui sera donc la racine du site.

Rappelez-vous que vous avez à disposition un script-shell de vérification ... et que votre archive ne doit déclencher aucune erreur (et s’il y a des warnings, vérifiez que c’est normal).
 Pour être clair, une archive ne passant pas avec succès le validateur sera considérée comme inexploitable.

Votre site doit pouvoir s’installer sur n’importe quel serveur sans nécessiter de modification. Notamment vous ne devez pas utiliser des chemins absolus pour désigner des fichiers ou des URLs (sauf URLs extérieures au site).

Dans votre archive vous mettez la totalité du code de votre site à l’exclusion des répertoires *var* et *vendor*. Pour info, pour tester votre site, nous lancerons la commande *composer install* (ou au pire *composer update*) pour générer les répertoires manquants.

Rappel : le fichier “ls-R.txt” ne contient que la liste des fichiers que vous créez ou modifiez dans le répertoire *config*.

A minima votre site doit fonctionner sur les machines des salles de TP avec le serveur de Symfony.

