1 Explications et planning

Ce devoir est à faire en binôme exclusivement.

La date de remise du projet est :

— vendredi 16 avril 2021 à 12h00

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).

Les enseignants se réservent le droit de convoquer les étudiants à un oral pour des précisions sur le travail.

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.

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 7.1 à 7.4 (évitez la version 8 qui nous posera des problèmes)
- 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 5.2 impérativement

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

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

2.1 Fonctionnement

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

L'authentification n'est pas gérée dans ce projet. Cependant un paramètre global indiquera si on est un visiteur authentifié ou non. Autrement dit, pour changer l'utilisateur connecté, il faudra modifier le code du site.

Il y a trois 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 quasi-totalité du site,
— *authentifié administrateur* : il gère uniquement les clients.

Le fait qu'un utilisateur soit un visiteur ou un administrateur est précisé dans la base de données.

À un utilisateur sont associés (cf. table fournie) :
— un login,
— un mot de passe,
— un nom, un prénom et une date de naissance,
— le fait qu'il soit administrateur ou non.

À un produit sont associés :
— un libellé,
— un prix unitaire (en euros avec un nombre réel),
— une quantité en stock.

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).

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 uniquement :
- se déconnecter
- gérer les utilisateurs
- ajouter un produit dans la base

Donc le menu (cf. section suivante) 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.
— 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.

La première page du site propose un message de bienvenue et indique si l'on est un visiteur anonyme, un client ou un administrateur.

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 *sqlite* à la racine de votre site.

Tous les noms de vos tables seront préfixés par *"im2021_"*, 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 entres les entités via Doctrine.

Le hashage des mots de passe se fera en utilisant la fonction *sha1*. Attention, on ne stocke dans la base que les mots de passe hashés.
Note : comme il n'y a pas de gestion de l'authentification, ce mot de passe de sera pas utilisé.

Attention les doublons ne sont pas autorisés pour le login dans la table des utilisateurs.

Créez au moins les 3 utilisateurs suivants (cf. table fournie) :

- *admin* (mot de passe *nimda*) qui est un administrateur du site,
- *gilles* (mot de passe *sellig*),
- *rita* (mot de passe *atir*),

La table *utilisateurs* vous est fournie. Elle vous sert de modèle pour créer l'entité correspondante et n'avez pas le droit de modifier sa structure ou de changer les noms des colonnes ou de la table. Autrement dit l'entité correspondante doit générer un code SQL reflétant celui fourni (notamment pour les noms, les types, les valeurs par défaut, les commentaires).
Une mention particulière pour le nom de la clé primaire qui n'est pas le nom attendu par défaut par Doctrine.

Alimentez votre base avec suffisamment de données pour faire des tests représentatifs.

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

3.2.1 Principe

Donc l'authentification est codée en dur dans le programme par un paramètre global précisé dans un fichier de configuration (cf. ci-après).

Ce paramètre contient uniquement la clé primaire qui caractérise l'internaute connecté ou une valeur caractéristique (0, "null", ... par exemple) pour un internaute anonyme.

On insiste sur le fait que pour changer d'utilisateur il faut donc mettre à jour ce paramètre et donc modifier le code du site [1].

3.2.2 Code source

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');
```

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

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.

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

3.3 Sécurité

Beaucoup d'actions ne sont pas accessibles selon le degré d'authentification d'un utilisateur. Il faudra gérer ces cas en affichant des messages d'erreurs.
Notons que si le site n'est pas buggué, il n'y a que deux manières d'arriver dans un état incohérent :

— l'utilisateur le fait exprès en tapant directement dans la barre d'adresse une URL non autorisée,

— le serveur a détruit les données liées à l'authentification (par exemple un temps d'inaction trop long de la part de l'internaute) : dans notre cas (l'authentification étant en dur) cela ne peut pas arriver.

Donc vous n'utiliserez pas le mécanisme de sécurité [2] de Symfony, mais vous testerez, en début de chaque action des contrôleurs, si l'internaute a les droits nécessaires ; et si ce n'est pas le cas vous lèverez une exception 404.

3.4 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), telle que le template fourni d'office par Symfony (*base.html.twig*).

Le deuxième niveau est propre à votre site : 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 *template* 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).

3.5 Divers

Toute duplication de code est à éviter.

Dans la mesure du possible, un formulaire mal rempli devra être réaffiché avec la mémoire des saisies précédentes de l'utilisateur.

Lorsque vous utiliserez les formulaires de Symfony, ce mécanisme est géré d'office.

Mais vous serez peut-être amené à écrire des formulaires directement en HTML : ne gérez alors cet aspect que dans un second temps, une fois que tout le reste fonctionne.

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.

2. ce qui est bien dommage ; vous êtes encouragés par ailleurs pour l'étudier.

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.

Enfin chaque vue du site aura possibilité de rajouter sa propre feuille CSS (par exemple pour centrer les titres) : vous choisirez la page de bienvenue pour profiter de cette possibilité.

4.2 Page de bienvenue

Une page est dédiée à l'accueil du site et contient uniquement un message poli de bienvenue avec la nature du visiteur (anonyme, client ou administrateur).

4.3 Menu

La division *menu* affiche deux informations :

- le nombre de produits disponibles dans la base de données
- le menu proprement dit qui propose une liste de liens.

Comme il a été dit, chaque type d'utilisateur à des actions possibles bien précises. Donc le menu s'adapte à la nature et aux droits de l'internaute.

Si on est en mode administrateur, il y a un lien supplémentaire vers :
https://fr.wikipedia.org/wiki/Amour_et_Amnésie

4.4 Connexion

Cette action concerne un internaute non authentifié.

Comme on ne gère pas l'authentification, la page de connexion se contente d'afficher un message (par exemple "Vous pourrez bientôt vous connecter").

4.5 Créer un compte client

Cette action concerne un internaute non authentifié.

On arrive sur un formulaire classique.

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, ...).

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

4.6 Déconnexion

Comme on ne gère pas l'authentification, cette action se contente de rediriger vers la page de bienvenue, mais en ayant au préalable créé un message flash.

4.7 Éditer/modifier son profil

Cette action concerne un internaute authentifié client.

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 est valide, on est redirigé sur la page qui liste les produits (avec un message flash), sinon on réaffiche le formulaire avec les messages d'erreurs adéquats.

NB : un utilisateur peut changer son mot de passe, et même son login (attention aux doublons), mais ne peut pas changer son statut (administrateur ou non).

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

4.8 Lister les produits du magasin

Cette action concerne un internaute authentifié client.

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 :

- libellé
- prix unitaire
- quantité n en stock
- une liste déroulante avec les nombres de 0 à n permettant de choisir la quantité que l'on veut commander (n étant le nombre d'éléments en stock) [3]. Attention, si la quantité en stock est nulle, le produit apparaît dans la page mais sans menu déroulant.

Le bouton "*ajouter*", qui permet d'ajouter le(s) produit(s) commandé(s) avec son (leur) nombre(s) d'occurrences dans le panier, se situe au bas de la page (évidemment s'il y a plusieurs centaines de produits, ce peut être pénible, mais je me répète *osef*). Notez qu'il n'y a bien qu'un seul bouton *ajouter* pour toute la page, et non pas un bouton à côté de chaque article [4] (cf. fichier exemple).
Le fait d'ajouter des produits dans le panier, diminue leurs quantités dans la base de données (comme s'ils avaient été réellement commandés).

Bref il s'agit d'un seul (et gros) formulaire.
On considère qu'il ne peut pas être mal rempli ; mais si le contrôleur gérant le formulaire détecte une erreur, une exception pourra être levée en guise de traitement.

Il n'est pas imposé d'utiliser les formulaires de Symfony : le formulaire peut tout à fait être directement écrit en HTML/Twig.

Après validation, on reste sur la même page, avec toutes les listes déroulantes réinitialisées pour éventuellement compléter la commande.

La gestion du panier est expliquée ci-dessous.

4.9 Gérer son panier

Cette action concerne un internaute authentifié client.

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

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/lien "Commander",
- un bouton/lien "Vider".

3. 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.
4. comme c'est habituellement le cas sur un vrai site

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

Notez qu'il n'est pas possible (i.e. qu'il n'est pas demandé d'implémenter) de modifier la quantité commandée d'un produit.
Si vous voulez augmenter la quantité d'un produit, il faut :

— retourner sur la page listant les produits,

— commander le surplus (et non pas la nouvelle quantité),

— retourner sur la page gérant le panier.

Si vous voulez diminuer (ou augmenter) la quantité d'un produit, il faut :

— supprimer le produit du panier [5],

— retourner sur la page listant les produits,

— commander la nouvelle quantité,

— retourner sur la page gérant le panier.

Bref il s'agit d'un contre-exemple de bonne ergonomie [6] [7].

Le fait de supprimer un produit : le fait disparaître du panier et remet également la quantité en stock à jour.
Le fait de vider le panier, outre 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 [8] [9].

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 après une déconnexion.

4.10 Gérer les utilisateurs

Cette action n'est accessible que par l'administrateur.

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

— login,

— mot de passe (crypté [10]).

— nom et prénom,

— date de naissance,

— statut (admin ou non)

Il est possible de supprimer un utilisateur (attention à vider proprement son panier s'il s'agit d'un client !).

Il ne doit pas être possible de supprimer l'utilisateur loggué.

4.11 Ajouter un produit

Cette action n'est accessible que par l'administrateur.

On arrive sur un formulaire classique.

5. en espérant qu'aucune autre personne n'en profitera pour vider le stock.
6. Est-il utile de rappeler qu'il n'est pas demandé de construire un site opérationnel.
7. cf. l'introduction si vous décidez d'implémenter, directement dans la page de gestion du panier, la possibilité de modifier la quantité commandée.
8. On ne vous demande pas de gérer l'expédition des produits.
9. 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.
10. il ne peut en être autrement

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, ...).

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

Un administrateur a uniquement les entrées suivantes dans son menu :

- déconnexion
- aller à la page d'accueil
- gérer les clients
- créer un produit
- un lien vers un site extérieur

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) très simple de votre choix (du style qui calcule la somme des entiers d'un tableau ou qui inverse une chaîne de caractères).

Utilisez-le dans une page de votre choix.

4.14 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 l'outil Swift Mailer. Ce mail contiendra le nombre de produits dans la base.

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

5 Travail à rendre

Documents à rendre (tous dans l'archive) :

- un fichier SQL "bd.sql" qui contient (cf. exportation sous PHPStorm) le code complet de votre base de données, avec suffisamment d'insertions pour tester votre application. Ce fichier est en plus de celui (*.db*) contenu dans le code du site.
- 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 ; tous fichiers sources que vous avez créés doivent avoir, en fin de fichier, un commentaire avec vos noms et prénoms.

— un rapport au format pdf, nommé “rapport.pdf” (disons 3 ou 4 pages hors page de garde et table des matières) qui contient :

  — l’adresse de votre site s’il est installé sur *tpweb* ou sur une adresse publique,

  — 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.

  — des points particuliers sur ce que vous avez faits sur le framework.

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

Les fichiers “bd.sql”, “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 créés ou modifiés dans le répertoire *config*.