SUPRSS

Projet à réaliser individuellement.

1 - Contexte du projet

Afin de diversifier son offre de services numériques et de fournir des outils performants à ses employés et potentiels clients, la société "InfoFlux Pro" vous mandate personnellement pour développer leur propre outil de lecture et de gestion de flux RSS nommé "SUPRSS". Cet outil se veut une alternative complète et intuitive aux solutions existantes (Feedly, Inoreader, NewsBlur, etc.), souvent limitées ou payantes.

Voici quelques flux RSS exemples :

- https://www.youtube.com/feeds/videos.xml?channel_id=<ID Chaine Youtube> (https://www.youtube.com/feeds/videos.xml?channel_id=UCH6ppHEvV3_WIXEwmhv9HEg),
- https://www.lemonde.fr/rss/une.xml,
- https://news.ycombinator.com/rss,
- https://rssviewer.app/ (qui permet de visualiser et obtenirs des flux RSS aléatoires), etc.

Vous devrez concevoir la charte graphique du projet, ainsi que l'application web.

2 - Description du projet

2.1 - Généralités

Le but principal de cette application est de permettre aux utilisateurs (employés ou clients de l'entreprise) de s'abonner, de consulter et d'organiser divers flux RSS provenant de différentes sources (sites d'actualités, blogs techniques, revues scientifiques, podcasts, etc.).

Chaque utilisateur possédera son propre compte et pourra gérer ses abonnements personnels. De plus, il aura la possibilité de créer des collections de flux partagées, permettant à plusieurs membres de consulter des articles communs et d'échanger sur ces derniers.

Chaque membre d'une collection partagée pourra ajouter des flux, classer des articles, et éventuellement commenter des éléments, avec des permissions définies pour chaque action.

2.2 - Fonctionnalités de l'application à implémenter

2.2.1 - Connexion

Un utilisateur pourra se connecter à l'application via un compte créé spécifiquement ou en utilisant un service d'OAuth2 (Google, Microsoft, GitHub, etc.).

2.2.2 - Création et gestion d'une collection de flux partagée

Tout utilisateur peut créer une collection de flux et inviter d'autres membres à la rejoindre. Une page dédiée à cette collection sera alors créée. Elle comportera l'ensemble des articles agrégés à partir des flux RSS sauvegardés au sein du groupe, avec les différentes permissions visibles par utilisateur (Créateur de la collection, droits d'ajout de flux, droits de lecture, droits de commentaire).

Chaque collection (individuelle ou partagée) doit avoir une barre de recherche qui lui est propre, permettant de rechercher les articles en fonction de leur titre, de leur source, de mots-clés ou de leur contenu.

2.2.3 - Gestion d'un flux et de ses articles

Tout utilisateur peut ajouter un flux RSS (au sein d'une collection personnelle ou partagée). Chaque flux sera constitué des éléments suivants :

- Nom du flux / Titre : Le nom affiché du flux (ex: "Le Monde - Actualités").
- URL du flux RSS : L'URL du fichier XML du flux RSS.
- Description (optionnel) : Une brève description du flux.
- Catégories / Tags : Possibilité d'assigner une ou plusieurs catégories ou tags pour organiser les flux.
- Fréquence de mise à jour : Possibilité de définir une fréquence pour la récupération des nouveaux articles (ex: toutes les heures, toutes les 6 heures, quotidiennement).
- Statut du flux : Actif/Inactif.
- S'il s'agit d'une collection partagée, les membres du groupe ayant la possibilité de modifier ou supprimer le flux. Tout utilisateur membre du groupe aura par défaut la permission de consulter les articles du flux.
- Articles : Chaque article agrégé du flux devra afficher :
    - Titre de l'article
    - Lien vers l'article original
    - Date de publication
    - Nom de l'auteur (si disponible)
    - Extrait ou résumé du contenu
    - Possibilité de marquer l'article comme "lu" ou "non lu".
    - Possibilité de "favoriser" un article.

Un flux RSS étant par nature volatile, vous devrez stocker en base de données chaque nouvel article afin de garantir leur disponibilité et pérénité.

2.2.4 - Catégorisation et filtrage des articles

Cet onglet permettra à l'utilisateur de visualiser et filtrer les articles en fonctions de différents critères qui devront explicitement être précisés :

- Filtrage par source (nom du flux).
- Filtrage par catégories / tags.
- Filtrage par statut (lus / non lus).
- Filtrage par favoris.
- Recherche plein texte au sein de tous les articles.

2.2.5 - Paramètres utilisateurs

Une gestion d'utilisateur standard est à prévoir : Changement de mot de passe, ajout d'OAuth2 (Google, Microsoft, GitHub, etc.), etc. En supplément, l'utilisateur pourra définir des préférences de lecture (ex: mode sombre, taille de police par défaut).

2.2.6 - Export

Afin d'anticiper la migration ou le partage de données, si un utilisateur le souhaite et malgré un avertissement, il devra pouvoir exporter l'ensemble des flux RSS auxquels il est abonné (et dont il est le créateur s'il s'agit de collections partagées) dans un format interprétable (OPML, JSON ou CSV).

L'ensemble des données comprises dans ce fichier seront en brut et donc lisibles.

2.2.7 - Import

Un utilisateur devra pouvoir importer, via un fichier interprétable (OPML, JSON ou CSV), des listes de flux RSS. Il sera automatiquement attribué comme créateur de ces derniers.

2.2.8 - Messagerie instantanée et commentaires

Une zone de messagerie instantanée interne à une collection de flux partagée devra permettre d'effectuer des discussions de groupe. De plus, il devra être possible de commenter spécifiquement chaque article au sein de ces collections partagées. Les différents membres pourront ainsi demander et obtenir des explications ou échanger sur des sujets précis.

2.3 - Déploiement

2.3.1 - Architecture

Votre application doit comporter trois briques distinctes :

- un serveur, devant être une API REST ou GraphQL et devant implémenter l'ensemble des fonctionnalités précédemment énoncées,
- un client web devant uniquement interagir avec le serveur,
- une base de données (choix libre).

Aucune logique ne doit avoir lieu sur le client qui ne sert que d'interface et redirige les différentes requêtes vers le serveur.

2.3.2 - Containérisation

Le projet doit comporter un fichier `docker-compose.yml` à la racine du projet permettant de déployer au moins 3 services Docker distincts, respectivement pour le serveur, le client web et la base de données.

L'application doit pouvoir être lancée intégralement via `docker compose` et être fonctionnelle.

3 - Le rendu

Il se fera sous la forme d'une archive au format "zip" contenant le code source, les fichiers annexes (sons, images, etc.), la documentations technique et le manuel utilisateur.

Un lien vers un dépôt Git du projet, contenant l'ensemble du projet devra être fourni. Le dépôt devra être privé durant la réalisation du projet et devra être passé en public dès le rendu sur Moodle.

Un dépôt vide, ou avec un historique de commits insuffisant ou non représentatif du travail effectué (par exemple, un seul commit contenant l'intégralité du projet), pourra entraîner une pénalité sur la note finale, voire l'ajournement du projet.

La documentation technique est à destination de professionnels du domaine et contiendra au moins les éléments suivants :

- Informations et éléments à renseigner nécessaires au fonctionnement de l'application,
- Guide de déploiement de l'application,
- Justification du choix des langages et des librairies,
- Diagrammes UML,
- Schéma de la base de données,

Le manuel utilisateur explique lui comment se servir de la solution et présente les différentes fonctionnalités.

Aucun secret (clef d'API, mot de passe, etc.) ne doit être présent dans le rendu. Un secret présent en clair entrainera un malus de points sur la notation en fonction de la criticité de ce dernier.

Tout rendu effectué en retard ne pourra pas être pris en compte.

Ce projet est noté sur 500 points avec possibilité d'obtenir 50 points en bonus :

- Documentations : 50 points (une note inférieure à 30 points sur cette partie entraînera un ajournement à ce projet)
    - Documentation technique : 30 points
    - Manuel utilisateur : 20 points
- Qualité de l’interface utilisateur : 10 points

- Qualité de l'expérience utilisateur : 10 points
- Déploiement : 50 points (une note inférieure à 30 points sur cette partie entraînera un ajournement à ce projet)
    - Architecture et abstraction : 20 points
    - Containérisation : 30 points
- Fonctionnalités : 190 points (une note inférieure à 100 points sur cette partie entraînera un ajournement à ce projet).
    - Inscription et connexion : 30 points
        - Connexion utilisateur standard (nom d'utilisateur et mot de passe) : 10 points
        - Connexion via OAuth2 (Facebook, Google, etc.) : 20 points
    - Fonctionnalités générales : 160 points
        - Création d'une collection de flux : 10 points
        - Gestion d'un flux et de ses articles : 60 points
            - Nom du flux / Titre : 5 points
            - URL du flux RSS : 5 points
            - Description (optionnel) : 5 points
            - Catégories / Tags : 5 points
            - Fréquence de mise à jour et statut : 10 points
            - Affichage des articles (titre, lien, date, auteur, extrait, lu/non lu, favoris) : 20 points
            - Permissions d'un flux par membre de la collection (lecture, modification, suppression) : 10 points
        - Catégorisation et filtrage des articles : 40 points
            - Filtrage par source : 10 points
            - Filtrage par catégories / tags : 5 points
            - Filtrage par statut (lus / non lus) : 10 points
            - Filtrage par favoris : 5 points
            - Recherche plein texte : 10 points
        - Messagerie instantanée et commentaires sur articles : 20 points
        - Export : 15 points
        - Import : 15 points
- Qualité du code : 190 points (une note inférieure à 100 points sur cette partie entraînera un ajournement à ce projet)
    - Le barème item par item est identique à celui des fonctionnalités. Pour un item non réalisé ou complètement dysfonctionnel, la note de qualité de code correspondante sera automatiquement égale à zéro.
    - Les critères appréciés ici sont essentiellement :
        - Structures de données adaptées.
        - Respect des limitations de requêtes au serveur RSS
        - Absence de duplication de code inadaptée.
        - Lisibilité du code (cela inclut la cohérence et le sens du nommage des variables et sous-programmes).
        - Facilité de maintenance.

        - Abstraction du code
- Bonus : jusqu’à 50 points
    - Exemples :
        - L’application est de qualité professionnelle (soin particulier accordé à l’UI/UX, qualité de code excellente, architecture robuste, etc.)
        - L’application est déployée en production et accessible non-localement
        - Des fonctionnalités conséquentes et pertinentes supplémentaires ont été implémentées (ex: notifications, agrégation d'articles par thème auto-appris, intégration avec des outils de veille)
- Malus : jusqu’à l’ajournement du projet
    - Quelques exemples :
        - Secret(s) en clair dans les fichiers, en fonction de la criticité :
            - Criticité mineure (par exemple : clé d’API peu importante, nom d’utilisateur de base de données) : Invalidation de la partie concernée (Qualité du code)
            - Criticité majeure (par exemple : mot de passe de base de données, clé d’API critique) : Ajournement du projet
        - Mots de passe non hachés en base de données