Ajout de Polytech Nantes

- Création d'un flux général pour une instance Castopod
- Connexion à Nextcloud
- Insertion d'image flexible
- Sondage

Création d'un flux général pour une instance Castopod

Dans la version actuelle de Castopod, il est possible de récupérer le flux RSS d'un épisode pour pouvoir utiliser ce protocole pour la gestion d'outils d'abonnement. Mais il n'est pas encore possible de créer un flux global rassemblant les flux RSS de chaque épisode dans une instance de Castopod. Il faut donc que chaque utilisateur récupère le flux RSS d'un épisode, et qu'il gère de son côté la récupération de multiples flux pour pouvoir être avertie de la sortie d'un nouvel épisode.

C'est dans cette problématique que Jet Asso nous a demandé de faire en sorte d'avoir un flux global RSS contenant l'ensemble des flux de chaque épisode. Et après quelques recherches, nous avons pu trouver un outil adapté pour cette demande : FreshRSS.

FreshRSS est un agrégateur de flux RSS open source et auto-hébergé qui vous permet de suivre facilement vos flux RSS préférés à partir d'une seule interface. Nous allons vous montrer comment installer FreshRSS à l'aide de Docker et comment l'utiliser pour suivre les flux RSS du Castopod de l'association Jet.

Installation de Fresh RSS

Prérequis

Assurez-vous que Docker est installé sur votre système. Si ce n'est pas le cas, vous pouvez suivre les instructions d'installation de Docker Desktop disponibles sur le site de Docker.

Installation avec Docker

Ouvrez une fenêtre de terminal, allez dans un dossier où les différentes données de FreshRSS seront stockés, et exécutez la commande suivante pour lancer le conteneur FreshRSS :

```
docker run -d --restart unless-stopped --log-opt max-size=10m \
  -p 8080:80 \
  -e TZ=Europe/Paris \
  -e 'CRON_MIN=1,31' \
  -v freshrss_data:/var/www/FreshRSS/data \
  -v freshrss_extensions:/var/www/FreshRSS/extensions \
```

```
--name freshrss \
freshrss/freshrss
```

Dans cette commande, nous spécifions :

- Le port sur lequel FreshRSS sera accessible (8080 dans cet exemple).
- Le fuseau horaire (Europe/Paris).
- La configuration de la tâche cron pour mettre à jour les flux toutes les 30 minutes (à la 1ère et à la 31ème minute de chaque heure).
- Les volumes pour stocker les données de FreshRSS et les extensions.

Une fois la commande exécutée, vous pouvez accéder à FreshRSS en ouvrant votre navigateur et en saisissant `http://localhost:port`.

Configuration initiale

La première fois que vous accédez à FreshRSS, vous devez effectuer une configuration initiale. Suivez les instructions à l'écran pour configurer votre compte utilisateur et vos préférences.

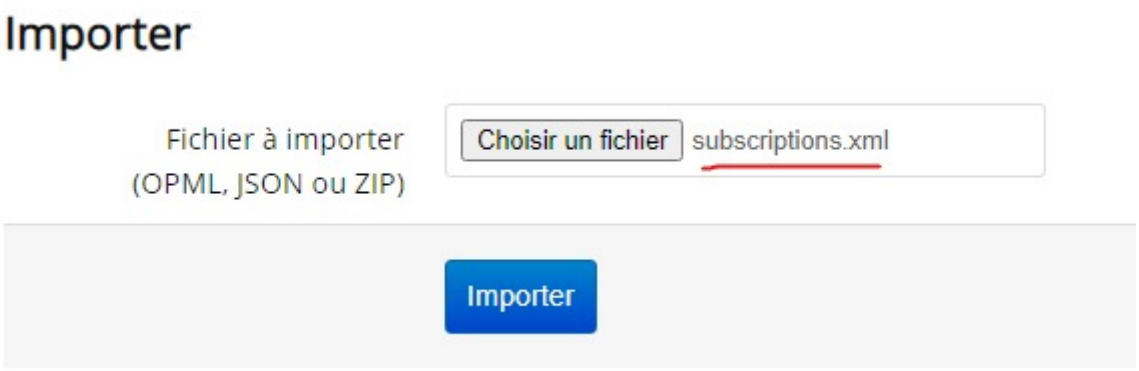
Utilisation de FreshRSS

Abonnement à l'ensemble des flux de Castopod

Pour suivre les flux RSS de l'association Jet, vous devez vous abonner aux flux RSS correspondants. Vous pouvez télécharger le fichier `subscriptions.xml` contenant tous les flux RSS de Castopod en cliquant **ici**.

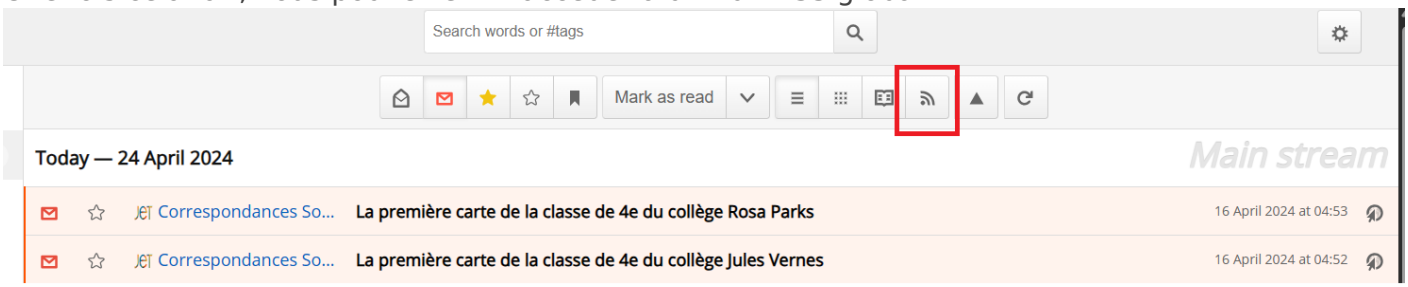
Pour importer le fichier subscriptions.xml dans FreshRSS, suivez ces étapes :

1. Cliquez sur l'icône "Gestion des abonnements" dans le menu latéral gauche.
2. Cliquez sur l'onglet "Importer / Exporter".
3. Dans la section "Importer", cliquez sur le bouton "Parcourir" et sélectionnez le fichier subscriptions.xml que vous avez téléchargé précédemment.
4. Cliquez sur le bouton "Importer" pour importer les flux RSS.



Une fois que vous avez importé les flux RSS, vous pouvez actualiser le flux, ce qui permettra de charger tous les flux que vous venez d'importer et de les afficher sur la page d'accueil de FreshRSS.

Une fois cela fait, vous pouvez enfin accéder à un flux RSS global :



Et c'est ce lien que vous pourrez partager à vos utilisateurs pour qu'ils puissent accéder aux flux RSS global que vous avez configuré.

Ajout d'un nouveau flux RSS

Si un nouveau podcast est ajouté à Castopod, celui-ci ne sera pas présent dans le fichier `subscriptions.xml` et vous devrez l'ajouter à la main. Pour ce faire :

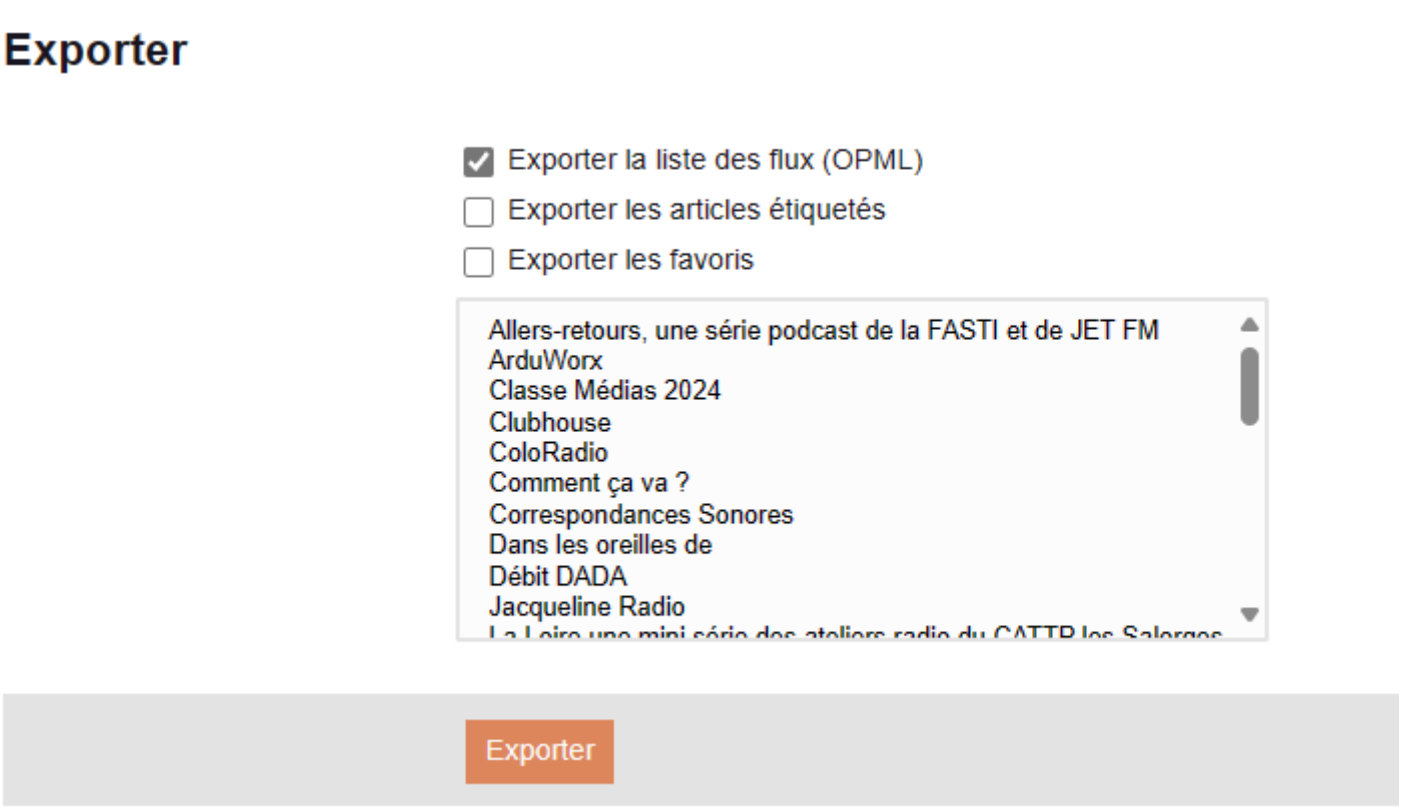
1. Cliquer sur le petit `+` à côté de "Gestion des abonnements"
2. Copier le lien du flux RSS du podcast dans "Ajouter un flux"
3. Choissisez la catégorie
4. Cliquer sur "Ajouter"
5. Dans la page qui s'ouvre, vous pouvez cliquer sur "Valider" directement sauf si vous voulez modifier certains paramètres

Et voilà ! Vous avez importé l'ensemble des flux RSS présents sur le Castopod de l'association Jet (au 22/04/2024).

Ajout d'un utilisateur

Une fois que le compte administrateur est correctement abonné à tous les flux RSS du Castopod de l'association, il est possible de créer de nouveaux comptes pour les personnes intéressées. Pour ce faire, cliquez sur l'icône en forme d'écrou en haut à droite, puis sélectionnez "Gestion des utilisateurs". Vous pouvez alors créer un nouvel utilisateur en remplissant les champs requis.

Pour que le nouvel utilisateur puisse également s'abonner à tous les flux RSS, le compte administrateur peut lui partager son fichier d'abonnement. Pour ce faire, l'administrateur doit retourner dans l'onglet "Importer / Exporter" de la gestion des abonnements et exporter sa liste d'abonnements en décochant les options "Exporter les articles étiquetés" et "Exporter les favoris".



Le nouvel utilisateur n'a alors plus qu'à importer le fichier généré par l'administrateur en suivant la même méthode que celle présentée précédemment dans la section "Utilisation". Il pourra ainsi s'abonner facilement à tous les flux RSS partagés par l'administrateur.

Extensions

Nous vous recommandons de vous pencher sur les extensions disponibles pour FreshRSS, que vous pouvez retrouver sur **cette page**. Chaque extension que vous téléchargez pourra être mis dans le dossier *freshrss_extensions* du dossier utilisé pour le déploiement du conteneur.

Sources

- Github de FreshRSS
- Site web de FreshRSS

Connexion à Nextcloud

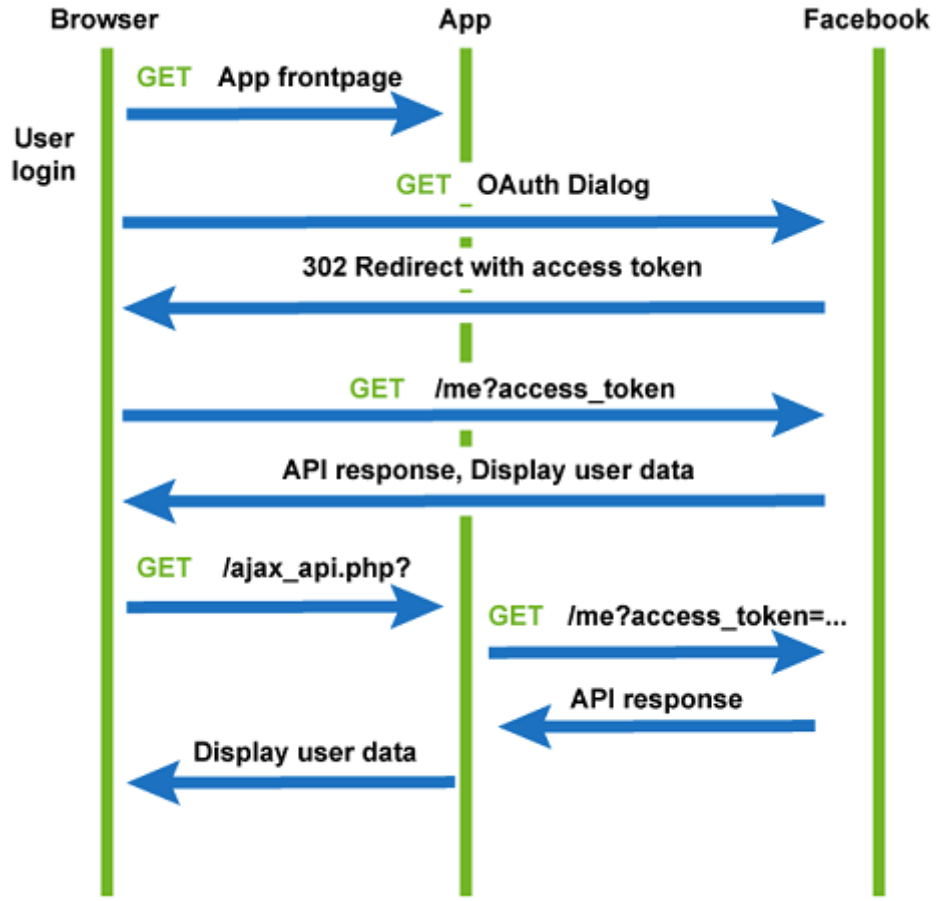
Connexion à Nextcloud via Castopod

1. Objectifs :

Permettre à l'utilisateur de se lié à Castopod son compte Nextcloud, offrant ainsi une expérience de connexion simplifiée et sécurisée.

2. Explication :

Pour réaliser cette fonctionnalité, nous avons mis en place un système d'authentification basé sur le protocole **OAuth**, qui implique un échange sécurisé de clés entre le client (Castopod) et le serveur (Nextcloud).



Lorsque l'utilisateur choisit de se connecter via Nextcloud, un contrôleur spécifique est déclenché pour initier le processus. Ce contrôleur effectue une requête GET afin d'afficher la page de connexion de Nextcloud, en incluant des paramètres tels que l'identifiant du client (idClient) et de serveur (idServeur), préalablement configurés dans les paramètres d'administration de Nextcloud.

Une fois que l'utilisateur autorise l'accès sur la page de connexion de Nextcloud, Nextcloud renvoie à Castopod un access_token. Ce token est utilisé par Castopod pour obtenir une clé API ainsi qu'un refresh_token. Ce dernier peut être utilisé ultérieurement pour obtenir une nouvelle clé API lorsque celle en cours expirera.

Pour implémenter cette fonctionnalité, plusieurs éléments ont été ajoutés au projet :

- **Contrôleur d'authentification (AuthController) :** Ce contrôleur est responsable de l'interaction avec le protocole OAuth. Il gère les requêtes et les réponses nécessaires à l'authentification via Nextcloud.

- **Entité OAuth :** Cette entité contient les informations requises pour le protocole OAuth, telles que les clés d'identification du client et du serveur, ainsi que des fonctions pour l'utilisation du protocole.

- **Modèle OAuth (OAuthModel) :** Ce modèle permet d'interagir avec la table de la base de données pour stocker et récupérer les informations relatives à l'authentification OAuth.

- **Utilisation de fonctions externes :** Pour garantir l'implémentation, notre projet s'appuie sur des fonctions fournies par un projet Git de Bahuma20. Ces fonctions offrent des interfaces PHP CodeIgniter pour la gestion du protocole OAuth.

Insertion d'image flexible

Insertion d'images flexibles dans Castopod

Objectif :

Permettre à l’utilisateur d’insérer des images dans Castopod, même si elles ne respectent pas les ratios (1:1) et la taille minimale requise (1400px), en fournissant une interface de recadrage et en ajustant automatiquement les dimensions si nécessaire.

Explication :

A la base, le formulaire de la page envoie les données de l’épisode (dont l’image) à un contrôleur (`EpisodeController`) via la route episode-create :

```
<form action="<?= route_to('episode-create', $podcast->id) ?>" method="POST" enctype="multipart/form-data" class="flex flex-col w-full max-w-xl mt-6 gap-y-8" id="episode-form">
```

```
$routes->post(
    'new',
    'EpisodeController::attemptCreate/$1',
    [
        'filter' => 'permission:podcast#.episodes.create',
    ],
);
```

Il fallait donc proposer à l’utilisateur cette interface, changer l’image du formulaire avant de l’envoyer, et gérer ce que renvoie le contrôleur.

> Pour gérer l’interface, il faut du JavaScript (JS) car c’est du code dynamique, qui va changer tout le temps, contrairement à PHP qui charge juste au début de la page. Le script Croppie permet de faire cette interface, de manière simple.

```
// Fonction pour afficher l’interface lorsqu’une image est insérée
var $uploadImage = document.getElementById('cover');
var $imagePreview = document.getElementById('imagePreview');
var croppie = new Croppie($imagePreview, {
    viewport: { width: 200, height: 200, type: 'square' },
    boundary: { width: 300, height: 300 },
});

$uploadImage.addEventListener('change', function () {
    $imagePreview.classList.remove('hidden');
    var reader = new FileReader();
    reader.onload = function (e) {
        croppie.bind({
            url: e.target.result,
        });
    };
    reader.readAsDataURL(this.files[0]);
});

// Fonction pour obtenir l'image croppée sous forme de fichier
function getCroppedImageAsFile(croppieInstance) {
    return new Promise(resolve => {
        croppieInstance.result({
            type: 'blob',
            format: 'jpeg',
            size: 'original',
        }).then(blob => {
            const file = new File([blob], 'cropped_image.jpeg', { type: 'image/jpeg' });
            resolve(file);
        });
    });
}
```

Pour gérer la taille de l’image, on utilise la fonction `resizeImage`, qui permet de changer la taille d’une image.

Une fois que l'utilisateur soumet le formulaire, nous bloquons l'envoi direct par la page. Étant donné que la nouvelle image est générée par JavaScript, l'envoi direct des données n'est pas possible. Nous sommes donc contraints de gérer l'envoi de la requête en JavaScript et de traiter la réponse. Comme cette réponse ne peut pas être un objet PHP, nous devons la gérer entièrement côté client.

```
document.getElementById('episode-form').addEventListener('submit', async function (event) {
    event.preventDefault();

    var formData = new FormData(this);

    if(document.getElementById('cover').value !== ''){
        // Récupérez l'image croppée sous forme de fichier
        const croppedFile = await getCroppedImageAsFile(croppie);

        // Redimensionnez l'image à une largeur de 1400px si nécessaire
        const resizedFile = await resizeImage(croppedFile, 1400);

        // Ajoutez le fichier redimensionné au formulaire FormData
        formData.set('cover', resizedFile);
    }

    try {
        const response = await fetch("<?= route_to('episode-create', $podcast->id) ?>", {
            method: "POST",
            body: formData,
        });
```

Auparavant, lorsque les données étaient envoyées directement depuis la page sans JavaScript, l'objet renvoyé était un `RedirectResponse`, géré en arrière-plan par un contrôleur. Cependant, dans ce scénario, cette approche n'est pas applicable. Nous devons plutôt utiliser un type d'objet compatible avec JavaScript, comme un JSON. Ce JSON peut prendre deux formes :

- En l'absence d'erreur : il contient un champ "id" qui représente l'URL de l'épisode créé. Dans ce cas, l'URL retournée est chargée.

```
const idUrl = responseData.id;
console.log(idUrl);
// Open the new episode view page
const baseUrl = '<?= base_url('/'); ?>'.slice(0, -1);
window.location.href = baseUrl + idUrl;
```

- En présence d'une erreur : un champ error qui contient les erreurs retournées par le contrôleur gérant l’upload de l’épisode. Dans ce cas, la page est rafraîchie avec un champ expliquant l’erreur au début de la page, à l’aide d’une variable `GET error` contenant l’erreur.

```php
<?php
  // Vérifier s'il y a une erreur, et dans ce cas l'afficher
  if (isset($_GET['error'])) {
      // Décoder la chaîne URL avant de la diviser
      $decodedError = urldecode($_GET['error']);

      // Diviser la chaîne en un tableau basé sur le caractère "+"
      $errorMessages = explode('+', $_GET['error']);

      // Supprimer les éléments vides du tableau
      $errorMessages = array_filter($errorMessages);

      // Afficher un div contenant un paragraphe pour chaque élément du tableau
      echo '<div class="flex flex-col gap-x-2 gap-y-4 md:flex-row">';
      echo '<fieldset class="w-full p-8 bg-red-200 border-3 flex flex-col items-start border-red-600 rounded-xl ">';
      foreach ($errorMessages as $errorMessage) {
          echo '<p> Erreur : ' . $errorMessage . '</p>';
      }
      echo '</fieldset>';
      echo '</div>';
  }
?>
```

Un problème rencontré en cas d'erreur est la perte des données saisies par l'utilisateur car la page est rechargée. Pour remédier à cela, nous effectuons des vérifications avant d'envoyer les données du formulaire. Nous vérifions que les champs obligatoires sont correctement remplis par l'utilisateur. Si un champ (ou plusieurs) est manquant, une erreur est affichée et la page est automatiquement ramenée à l'emplacement du champ manquant.

```
const audioFile = document.getElementById('audio_file').value;
const title = document.getElementById('title').value;
const description = document.getElementById('description').value;

// Vérifiez si les champs requis sont vides
const emptyFields = findEmptyFields([
    { id: 'audio_file', label: 'Audio File' },
    { id: 'title', label: 'Title' },
    { id: 'description', label: 'Description' }
]);
```

```
if (emptyFields.length > 0) {
    const errorMessage = `Veuillez remplir les champs obligatoires : ${emptyFields.map(field =>
field.label).join(', ')}.`;
    alert(errorMessage);

    // Faites défiler la page jusqu'au premier champ manquant
    const firstEmptyField = emptyFields[0].id;
    const element = document.getElementById(firstEmptyField);
    const elementPosition = element.getBoundingClientRect().top + window.scrollY - window.innerHeight / 2;
    window.scrollTo({ top: elementPosition, behavior: 'smooth' });
}
```

En conclusion, pendant que le script JavaScript gère l'envoi du formulaire et la réponse associée, nous désactivons le bouton d'envoi dès que l'utilisateur appuie dessus. Cette désactivation évite que l'utilisateur n'appuie plusieurs fois sur le bouton, ce qui pourrait entraîner l'envoi répété de la même requête. De plus, un indicateur visuel sous forme d'un rond de chargement apparaît pour informer l'utilisateur que le traitement est en cours.

```
// Baisser l'opacité de la page sauf pour le cercle de chargement
document.getElementById('all').style.opacity = '0.5';

// Désactiver le bouton
const submitButton = document.getElementById('submit-button');
submitButton.disabled = true;

// Afficher le cercle de chargement
const loadingCircle = document.getElementById('loading-circle');
loadingCircle.style.display = 'block';
```

Sondage

Page de Sondage sur Castopod

Introduction

La page de sondage sur Castopod constitue une fonctionnalité toujours en cours de développement au sein de l'écosystème Castopod. Initialement conçue dans le cadre d'une prise en main du développement sur Castopod, cette fonctionnalité est considérée comme secondaire, sans nécessité d'améliorations majeures étant donné qu'elle n'est pas jugée cruciale pour le client.

Fonctionnalités Principales

- **Création de Sondages:** Les utilisateurs ont la possibilité de créer des sondages directement depuis l'interface de Castopod.
- **Gestion des Sondages:** La page de sondage permet la gestion des sondages existants, y compris leur édition et leur suppression si nécessaire.
- **Participation aux Sondages:** Les auditeurs peuvent participer aux sondages en votant pour leurs choix préférés.

Architecture Technique

- **Frontend:** La page de sondage est implémentée en utilisant du HTML, CSS et JavaScript.
- **Backend:** Le backend de la fonctionnalité de sondage est développé en utilisant le framework PHP CodeIgniter, intégré à l'écosystème de Castopod. Il gère la logique métier, y compris la création, la modification et la suppression des sondages, ainsi que la gestion des votes des utilisateurs.

Limitations Connues

Étant considérée comme une fonctionnalité secondaire, la page de sondage ne bénéficie pas de ressources prioritaires pour son développement et son amélioration.