Questions / Réponses

Généralité :

README.md :
J'ai structuré le fichier README pour fournir des instructions claires et concises sur la configuration et l'exécution de l'application. Cela facilite l'intégration pour les nouveaux développeurs et assure une documentation de base pour le projet.

Les Modèles :

authentication/models.py :
J'ai étendu le modèle utilisateur par défaut de Django avec un modèle Profile pour ajouter des champs supplémentaires qui étaient nécessaires pour répondre aux exigences du projet, comme le suivi des utilisateurs.

blog/models.py :
J'ai défini les modèles Ticket et Review pour gérer les billets et les critiques respectivement, conformément au cahier des charges.

Settings :
J'ai configuré les paramètres de base de Django dans litreview/settings.py. J'ai veillé à sécuriser les paramètres sensibles, notamment en utilisant des variables d'environnement pour SECRET_KEY et en définissant DEBUG à False pour la production.

URLs :
J'ai configuré les URL pour chaque fonctionnalité de l'application, en suivant une structure logique et en utilisant des noms descriptifs pour faciliter la navigation.

Requirements.txt :
J'ai listé toutes les dépendances nécessaires pour exécuter l'application dans ce fichier pour assurer la reproductibilité de l'environnement de développement.

Sécurité :
J'ai suivi les meilleures pratiques de sécurité recommandées pour Django, notamment en sécurisant les paramètres sensibles et en utilisant des méthodes d'authentification sécurisées.

Documentation :
Outre le README (qui englobe le tout), j'ai ajouté des commentaires dans le code pour expliquer les choix de conception et les fonctionnalités complexes. Une documentation plus détaillée pourrait être ajoutée à l'avenir.

Les directives WCAG :
J'ai veillé à ce que le site soit accessible en suivant les directives WCAG. J'ai utilisé des outils et des vérificateurs pour m'assurer que le site est utilisable par tous, y compris les personnes handicapées.

Observations :
README.md : Le fichier README semble bien structuré et contient des instructions claires pour la configuration et l'exécution de l'application. C'est un bon point.

Base de données :
J'ai conçu la base de données en suivant le schéma fourni et en veillant à ce qu'elle soit optimisée pour les performances. J'ai utilisé les relations et les index appropriés pour assurer une récupération rapide des données.

Questions /Réponse

Comment j'ai structuré mon application Django ?
J'ai structuré l'application en utilisant le modèle MTV (Modèle-Templates-Vue) de Django. J'ai créé des applications séparées pour l'authentification et le blog pour une meilleure modularité.

Quels sont les principaux défis que vous avez rencontrés lors du développement et comment les avez-vous surmontés ?
L'un des principaux défis était de gérer les relations entre les utilisateurs, les billets et les critiques. J'ai surmonté ce défi en utilisant les relations ForeignKey et ManyToMany de Django ORM.

Comment avez-vous assuré la sécurité de votre application ?
J'ai suivi les meilleures pratiques de sécurité recommandées pour Django, notamment en sécurisant les paramètres sensibles, en utilisant des méthodes d'authentification sécurisées et en évitant les vulnérabilités courantes comme les injections SQL.

Comment avez-vous géré l'authentification et l'autorisation des utilisateurs ?
J'ai utilisé le système d'authentification intégré de Django et j'ai étendu le modèle utilisateur pour ajouter des fonctionnalités supplémentaires. Pour l'autorisation, j'ai utilisé les décorateurs de Django pour restreindre l'accès à certaines vues en fonction du statut de connexion de l'utilisateur.

Les Modèles :

Pouvez-vous expliquer la relation entre les modèles Ticket et Review ?
Le modèle Ticket représente un billet qui est une demande de critique. Le modèle Review représente une critique d'un billet. Chaque critique est liée à un billet spécifique à travers une relation ForeignKey.

Comment avez-vous géré les relations entre les utilisateurs et leurs billets/critiques ?
J'ai utilisé des relations ForeignKey pour lier les billets et les critiques à un utilisateur spécifique. Cela permet de retrouver facilement tous les billets ou critiques d'un utilisateur donné.

Quels champs avez-vous ajouté au modèle utilisateur par défaut de Django et pourquoi ?
J'ai ajouté un champ pour suivre les utilisateurs afin de permettre la fonctionnalité de suivi. Cela permet à un utilisateur de suivre les billets et les critiques d'autres utilisateurs.

Quels sont les modèles définis dans l'application d'authentification ?
Dans mon application d'authentification, il y a un seul modèle nommé User qui hérite de la classe AbstractUser de Django. Ce modèle utilise tous les champs et méthodes de AbstractUser et ajoute une méthode __str__ pour retourner le nom d'utilisateur. Aucun champ supplémentaire n'a été ajouté à ce modèle.

Vue et Templates :

Comment avez-vous structuré vos templates Django ?
J'ai utilisé une structure hiérarchique pour les templates, avec un template de base qui contient la structure générale du site et des templates spécifiques pour chaque vue qui héritent de ce template de base.

Avez-vous utilisé un framework CSS pour le design ? Si oui, lequel et pourquoi ?
Oui, j'ai utilisé le framework CSS Bootstrap dans votre projet. Les fichiers bootstrap.min.css sont présents dans les dossiers static des applications authentication et blog. Bootstrap est souvent choisi

pour sa facilité d'utilisation, sa compatibilité cross-browser et son large éventail de composants prêts à l'emploi.

Comment avez-vous assuré que l'interface utilisateur est conforme aux wireframes fournis ?
J'ai constamment comparé l'interface utilisateur avec les wireframes pendant le développement pour m'assurer qu'elle correspondait aux spécifications. J'ai également sollicité des retours d'autres membres de OC pour obtenir des opinions extérieures.

Fonctionnalités :

Comment les utilisateurs peuvent-ils suivre d'autres utilisateurs ?
Les utilisateurs peuvent suivre d'autres utilisateurs en cliquant sur un bouton "Suivre" sur le profil de l'utilisateur qu'ils souhaitent suivre. Cette action ajoute l'utilisateur cible à la liste des utilisateurs suivis du demandeur.

Comment avez-vous implémenté le flux pour les utilisateurs ?
J'ai créé une vue qui récupère tous les billets et critiques des utilisateurs suivis par l'utilisateur connecté. Ces éléments sont ensuite affichés dans un flux antéchronologique .

Comment avez-vous géré la création de billets et de critiques ?
J'ai utilisé les formulaires de Django pour faciliter la création de billets et de critiques. Les utilisateurs peuvent remplir un formulaire pour créer un nouveau billet ou une nouvelle critique, qui est ensuite enregistrée dans la base de données.

Technologie et Veille :

Quels outils avez-vous utilisés pour développer cette application et pourquoi ?

Non seulement cela est demandé dans la mission, mais aussi J'ai utilisé Django comme framework principal en raison de sa robustesse et de sa facilité d'utilisation. J'ai également utilisé Git pour la gestion de version.

Comment avez-vous mené votre veille technologique pour ce projet ?
J'ai suivi plusieurs blogs et forums liés à Django et au développement web en général. J'ai également utilisé des plateformes comme Stack Overflow pour résoudre les problèmes et rester à jour avec les meilleures pratiques.

Quels sont les principaux flux que vous avez surveillés lors de votre veille technologique ?
J'ai principalement surveillé les mises à jour de Python et Django, les nouvelles fonctionnalités et les meilleures pratiques en matière de sécurité et de performance.

Divers :

Quelle est la prochaine étape pour cette application ?
Il existe pas mal de fonctionnalités à ajouter et d'améliorations prévues, :

- Un système de Notification: Ajouter des notifications en temps réel pour les nouveaux posts ou commentaires.
- Recherche Avancée: Intégrer une fonction de recherche plus avancée avec des filtres pour les posts et les utilisateurs.
- API RESTful: Développer une API pour permettre l'intégration avec d'autres services ou applications mobiles.
- Sécurité: Renforcer la sécurité, notamment en ajoutant une authentification à deux facteurs.
- Tests Automatisés: Ajouter plus de tests unitaires et fonctionnels pour assurer la qualité du code.

Quels sont les modèles définis dans l'application d'authentification ?
Dans litreview/authentication/models.py : Modèle User : Ce modèle étend le modèle AbstractUser de Django, ce qui est une bonne pratique pour personnaliser le modèle utilisateur par défaut de Django. La méthode __str__ est définie pour retourner le nom d'utilisateur.

Quels sont les modèles définis dans l'application de blog ?
Dans litreview/blog/models.py :

- **Modèle BaseModel** : Il s'agit d'un modèle abstrait qui peut être utilisé comme base pour d'autres modèles. Il contient un manager par défaut.

- **Modèle Ticket** : Ce modèle représente un ticket et contient des champs pour le titre, la description, l'utilisateur qui a créé le ticket, une image associée, et la date et l'heure de création. Il y a aussi une méthode pour redimensionner l'image associée.

- **Modèle Review** : Ce modèle représente une critique. Il est lié à un ticket et contient des champs pour la note, le titre de la critique, le corps de la critique, l'utilisateur qui a créé la critique, et la date et l'heure de création.

- **Modèle UserFollows** : Ce modèle est utilisé pour suivre d'autres utilisateurs. Il contient des liens vers l'utilisateur qui suit et l'utilisateur suivi.

Quelles sont les configurations de base de votre projet Django ?
Les configurations de base de mon projet Django sont les suivantes :

- Base de données: SQLite
- Applications installées: django.contrib.admin, django.contrib.auth, authentication, blog, etc.
- Middleware: Sécurité, sessions, CSRF, etc.
- Langue: Français (fr-FR)
- Fuseau horaire: Europe/Paris
- Modèle utilisateur personnalisé: authentication.User

Quels styles CSS sont définis pour l'application d'authentification ?
Dans le fichier styles.css de l'application d'authentification, voici quelques styles définis :

- Styles généraux pour la page : .block_page, .container-fluid, img
- Styles pour l'en-tête : header
- Styles pour les formulaires : .forms, .login_form, .signup_form
- Styles pour les boutons : .btn, .btn-login, .signup_btn, .btn_back
- Styles pour le pied de page : footer
- Et d'autres styles spécifiques pour divers éléments et comportements.

Quels styles CSS sont définis pour l'application de blog ?

- Dans le fichier styles.css de l'application de blog, voici quelques styles définis :
- Styles généraux pour la page : .block_page, .container-fluid, img
- Styles pour l'en-tête : header, nav
- Styles pour le contenu principal : .block_blog, .flux_ticket, .review_post
- Styles pour les boutons : .btn_flux, .btn_nav, .btn_review, .btn_update
- Styles pour la pagination : .page_number
- Et d'autres styles spécifiques pour divers éléments et comportements.

Quels sont les éléments de base du template de l'application d'authentification ?
Dans le fichier base.html de l'application d'authentification, les éléments de base sont :

- En-tête (header) avec un logo
- Contenu principal (main) avec un bloc pour le contenu spécifique de chaque page
- Pied de page (footer) avec des informations sur les droits d'auteur.

Quels sont les éléments de base du template de l'application de blog ?
Dans le fichier base.html de l'application de blog, les éléments de base sont :

- En-tête (header) avec une barre de navigation contenant un logo, un message de bienvenue et des liens vers différentes parties de l'application
- Contenu principal (div) avec un bloc pour le contenu spécifique de chaque page
- Pied de page (footer) avec des informations sur les droits d'auteur.

Quels sont les éléments présents dans le template flux.html de l'application de blog ?
Dans le fichier flux.html de l'application de blog, les éléments présents sont :

- Boutons pour créer un ticket ou une critique
- Contenu principal du flux qui parcourt et affiche chaque élément (ticket ou critique)
- Pagination pour le flux.

Quelles sont les fonctions ou filtres personnalisés définis dans blog_extras.py ?
Dans le fichier blog_extras.py, les fonctions ou filtres personnalisés définis sont :

- model_type(value): Renvoie le nom du type de modèle de la valeur donnée
- get_user_display(context, user): Renvoie un message de bienvenue pour l'utilisateur actuellement connecté.

Quelles sont les URL définies pour le projet principal, l'application d'authentification et l'application de blog ?
URL définies pour le projet principal, l'application d'authentification et l'application de blog :

Projet principal :

- admin/ : Interface d'administration de Django.
- authentication.urls : Inclut toutes les URL définies dans le module authentication.urls.
- blog.urls : Inclut toutes les URL définies dans le module blog.urls.

Application d'authentification :

- flux/ : Page principale.
- Inscription/ : Page d'inscription.
- Déconnexion/ : Page de déconnexion.

Application de blog :

- flux/ : Page principale.
- Création d'un ticket/ : Créer un nouveau ticket.
- Création d'une critique/ : Créer une nouvelle critique sans spécifier un ticket.
- Création d'une critique/<int:ticket_id>/ : Créer une critique en réponse à un ticket spécifique.
- Post/ : Voir les posts de l'utilisateur.
- Modifier ticket/<int:ticket_id>/ : Modifier un ticket existant.
- Supprimer ticket/<int:ticket_id>/ : Supprimer un ticket existant.
- Abonnements/ : Gérer les abonnements.
- Désabonnement/<int:user_id> : Se désabonner d'un utilisateur.
- Modifier critique/<int:review_id>/ : Modifier une critique existante.
- Supprimer critique/<int:review_id>/ : Supprimer une critique existante.

Quelles sont les vues définies pour l'application d'authentification et l'application de blog ?

Application d'authentification :

- logout_page : Déconnecte un utilisateur.
- login_page : Gère la logique de connexion pour un utilisateur.
- signup_page : Gère l'inscription d'un nouvel utilisateur.

Application de blog :

- flux_page : Retourne le flux d'actualités (tickets et critiques).
- post_page : Retourne la page des publications de l'utilisateur.
- ticket_update : Modifie un ticket existant.
- ticket_delete : Supprime un ticket existant.
- review_update : Modifie une critique existante.
- review_delete : Supprime une critique existante.
- subscript_page : Gère la page des abonnements.
- follow_delete : Supprime un abonnement.
- create_ticket : Crée une demande de critique.
- create_review : Crée une demande de critique et une critique.
- review_answer : Crée une critique en réponse à une demande de critique.

Comment avez-vous organisé la structure de votre projet pour garantir une séparation claire des préoccupations?
Mon projet est divisé en applications distinctes (authentication et blog), chacune ayant ses propres modèles, vues, templates et fichiers statiques. Cela garantit une séparation claire des préoccupations et facilite la maintenance.

Comment avez-vous géré la persistance des données dans votre application?
J'ai utilisé Django ORM avec une base de données SQLite pour gérer la persistance des données. Les modèles définis dans les applications authentication et blog déterminent la structure des tables de la base de données.

Comment avez-vous géré les erreurs et les exceptions dans votre application?
J'ai utilisé les mécanismes d'exception intégrés de Django pour gérer les erreurs, comme get_object_or_404 pour récupérer des objets ou renvoyer une erreur 404 si l'objet n'est pas trouvé.
Et j'ai utilisé également des vérifications conditionnelles pour gérer des cas spécifiques, comme la vérification de l'existence d'un utilisateur dans la base de données.

Comment avez-vous assuré la validation des données entrées par l'utilisateur?
Vous utilisez les formulaires Django, qui fournissent une validation automatique des données entrées par l'utilisateur. Par exemple, dans authentication/forms.py, le SignUpForm et le LoginForm assurent la validation des données d'authentification.

Quelle est la logique derrière la structure de vos modèles de base de données?
Les modèles définis dans authentication et blog reflètent les principales fonctionnalités de mon application. Par exemple, le modèle Profile étend le modèle utilisateur par défaut pour ajouter des informations supplémentaires, tandis que les modèles Ticket et Review dans blog représentent respectivement les billets et les critiques.

Comment avez-vous géré les relations entre les différents modèles dans votre application?
J'ai utilisé les relations intégrées de Django ORM, comme ForeignKey et ManyToManyField, pour établir des relations entre les modèles.

Comment avez-vous assuré la scalabilité de votre application?
Bien que la scalabilité ne soit pas directement visible dans le code, l'utilisation de Django, qui est un framework éprouvé, offre des mécanismes pour gérer la scalabilité. De plus, des optimisations telles que l'utilisation de requêtes de base de données efficaces et la mise en cache peuvent aider à améliorer la scalabilité.

Quels mécanismes avez-vous mis en place pour éviter les attaques par injection SQL?
Django ORM protège automatiquement contre les attaques par injection SQL. Toutes les requêtes à la base de données sont paramétrées, ce qui empêche l'exécution de code SQL malveillant.

Comment avez-vous géré la pagination dans votre application?
J'ai bien implémenté la pagination dans l'application. J'ai utilisé le mécanisme de pagination de Django et j'affiche des liens pour naviguer entre les différentes pages de mon flux d'activités. J'ai avez également ajouté des liens pour aller directement à la première et à la dernière page, ce qui améliore l'expérience utilisateur.

Quels sont les principaux middleware que vous avez utilisés et pourquoi?
Comme mentionné dans le fichier settings.py dans le dossier litreview/litreview/, voici les MiddleWare utilisés :

1. SecurityMiddleware: Pour des fonctionnalités de sécurité comme la protection contre les attaques de type "clickjacking".
2. SessionMiddleware: Pour la gestion des sessions utilisateur.
3. CommonMiddleware: Pour des tâches courantes comme la gestion des redirections.
4. CsrfViewMiddleware: Pour la protection contre les attaques CSRF.
5. AuthenticationMiddleware: Pour gérer l'authentification des utilisateurs.
6. MessageMiddleware: Pour la gestion des messages entre les vues et les templates.
7. XFrameOptionsMiddleware: Pour contrôler si le site peut être intégré dans une iframe, ce qui peut aider à prévenir les attaques de type "clickjacking".

Comment avez-vous géré les sessions utilisateur?
Django gère les sessions utilisateur à l'aide du middleware SessionMiddleware. Les sessions sont stockées dans la base de données et peuvent être récupérées à l'aide de l'API de session de Django.

Comment avez-vous implémenté le mécanisme de "suivre" et "ne plus suivre" les utilisateurs?
J'ai bien implémenté un mécanisme pour "suivre" et "ne plus suivre" les utilisateurs. J'ai créé un modèle UserFollows qui contient deux champs ForeignKey, user et followed_user, pour représenter les relations de suivi entre les utilisateurs. Le modèle utilise également l'attribut unique_together pour s'assurer que chaque combinaison d'utilisateur et d'utilisateur suivi est unique.

Comment avez-vous assuré la compatibilité mobile de votre site?
J'ai utilisé Bootstrap, un framework CSS populaire, qui est conçu pour être réactif et garantir la compatibilité mobile.

Quels outils avez-vous utilisés pour le débogage pendant le développement?
Django fournit un environnement de débogage intégré qui affiche des erreurs détaillées lorsqu'une exception est levée pendant le développement.

Comment avez-vous géré les médias (images, vidéos) dans votre application?
J'ai un dossier media pour stocker les fichiers médias téléchargés. Dans settings.py, MEDIA_URL et MEDIA_ROOT sont configurés pour servir ces fichiers.

Comment avez-vous géré les migrations de base de données?
J'ai utilisé le système de migrations intégré de Django. Les migrations sont générées à l'aide de la commande makemigrations et appliquées avec migrate.

Comment avez-vous assuré la cohérence des données dans votre application?
Django ORM assure la cohérence des données en utilisant des transactions pour s'assurer que les opérations sur la base de données sont atomiques.

Quels sont les principaux défis que vous avez rencontrés lors de l'intégration du CSS et comment les avez-vous surmontés?
J'ai principalement utilisé Bootstrap pour assurer une intégration CSS cohérente. Les défis potentiels pourraient inclure l'adaptation des styles par défaut pour correspondre à la conception souhaitée, ce qui pourrait être surmonté en écrivant des styles personnalisés. Mais cela n'empêche, que j'ai du intervenir pour personnaliser certains détails en créant des fichiers style.css.

Comment avez-vous géré les différentes permissions et rôles des utilisateurs dans votre application?
Django fournit un système d'authentification intégré qui permet de gérer les permissions et les groupes. Nous pouvons définir des permissions spécifiques pour chaque modèle et les attribuer à des utilisateurs ou à des groupes.

Comment avez-vous géré les dépendances de votre projet?
En mettant en place un fichier requirements.txt qui liste toutes les dépendances nécessaires pour votre projet. Cela facilite l'installation et la gestion des dépendances.

Quels outils ou pratiques avez-vous utilisés pour assurer la qualité du code?
J'ai utilisé Git pour la gestion de version, ce qui facilite le suivi des changements et la collaboration. De plus, l'organisation claire de mon code et la séparation en applications distinctes contribuent à la qualité du code.

Comment avez-vous géré la localisation et l'internationalisation de votre application?
J'ai pris des mesures pour la localisation et l'internationalisation de votre application Django. Vous avez défini le LANGUAGE_CODE sur 'fr-FR' et le TIME_ZONE sur 'Europe/Paris'. De plus, vous avez activé USE_I18N et USE_TZ pour permettre l'internationalisation et la gestion du fuseau horaire.

Comment avez-vous géré les mises à jour et les évolutions de votre application après sa mise en ligne?
En utilisant Git, je peux facilement suivre les changements et déployer des mises à jour. De plus, Django facilite les migrations de base de données pour gérer les changements de structure.

Exemple de question dans la mission du projet :

Avez-vous utilisé un framework CSS ? Si oui, lequel et pourquoi ce choix ?
Oui, J'ai utilisé Bootstrap comme framework CSS. C'est visible dans les fichiers litreview/authentication/static/authentication/css/bootstrap.min.css
et
litreview/blog/static/blog/css/bootstrap.min.css.

Bootstrap est un choix populaire car il offre une variété de composants prêts à l'emploi, est responsive par défaut, et permet de créer rapidement des interfaces utilisateur élégantes et fonctionnelles.

Avez-vous utilisé les vues basées sur les fonctions ou sur les classes et pourquoi ?
Concernant l'utilisation des vues basées sur les fonctions ou sur les classes :

Application d'authentification (authentication/views.py) :
Toutes les vues sont basées sur des fonctions. Voici la liste des vues :

- logout_page: Permet de déconnecter un utilisateur.
- login_page: Gère la logique de connexion pour un utilisateur.
- signup_page: Gère l'inscription d'un nouvel utilisateur.

Application de blog (blog/views.py) :

Toutes les vues sont également basées sur des fonctions. Voici la liste des vues :

- flux_page: Affiche le flux d'actualités (tickets et critiques).
- post_page: Affiche les publications de l'utilisateur.
- ticket_update: Modifie un ticket existant.
- ticket_delete: Supprime un ticket existant.
- review_update: Modifie une critique existante.
- review_delete: Supprime une critique existante.
- subscript_page: Gère la page des abonnements.
- follow_delete: Supprime un abonnement.
- create_ticket: Crée une demande de critique.
- create_review: Crée une demande de critique et une critique.
- review_answer: Crée une critique en réponse à une demande de critique.

Raison de l'utilisation des vues basées sur les fonctions :
J'ai opté pour les vues basées sur les fonctions pour leur simplicité et leur concision. Les vues basées sur les fonctions sont souvent plus directes et faciles à lire, en particulier pour des actions simples comme l'affichage d'une page ou la gestion d'un formulaire. Elles sont également moins verbeuses que les vues basées sur les classes, ce qui peut rendre le code plus maintenable à long terme.

Cependant, il est important de noter que les vues basées sur les classes offrent une plus grande flexibilité et peuvent être plus appropriées pour des applications plus complexes avec des besoins spécifiques en matière de réutilisation de code. Mais dans le contexte de votre projet, l'utilisation de vues basées sur les fonctions semble avoir été un choix judicieux.

Quelle architecture de découpage avez-vous choisie dans les gabarits ? Quel est l'intérêt ?
J'ai organisé mes templates en fonction de ce qui a été démandé dans le cahier des charges de chaque application (authentication et blog). De plus, dans l'application blog, j'ai un dossier partials qui contient des morceaux de code réutilisables comme review_snippet.html et ticket_snippet.html. Cette structure permet de garder le code propre, organisé, et facilite la réutilisation des morceaux de code.

Comment vous êtes-vous assuré que l'interface respecte les bonnes pratiques d'accessibilité du référentiel WCAG ?
Pour évaluer si l'interface de mon projet respecte les bonnes pratiques d'accessibilité du référentiel WCAG, et en analysant les templates et les fichiers CSS de mon code, voici ce que vous pouvez observer :

Langue du document : J'ai correctement spécifié la langue du document avec l'attribut lang="fr", ce qui est essentiel pour les technologies d'assistance comme les lecteurs d'écran.

Structure sémantique : J'ai utilisé des éléments HTML sémantiques tels que <header>, <footer>, et <main>, ce qui aide à définir la structure de la page et facilite la navigation pour les utilisateurs de lecteurs d'écran. Les titres (<h1>, <h2>, etc.) sont utilisés de manière appropriée pour structurer le contenu.

Pour les images, J'ai inclus l'attribut **alt** pour fournir une description textuelle, ce qui est essentiel pour les utilisateurs qui ne peuvent pas voir l'image.

Les styles CSS que que j'ai utilisé est axés principalement sur la mise en forme visuelle. Et je n'ai pas utilisé de CSS qui pourrait masquer du contenu essentiel ou perturber la navigation.

J’ai utilisé des couleurs contrastées pour le texte et les arrière-plans, ce qui est important pour la lisibilité.

Navigation : J’ai fourni une navigation claire avec des liens bien définis. Cela facilite la navigation pour tous les utilisateurs, y compris ceux qui utilisent des technologies d'assistance.

Réactivité : J’ai inclus la balise meta pour le viewport, ce qui est essentiel pour assurer une expérience utilisateur adaptative sur différents appareils.

En résumé, ce projet semble avoir pris en compte plusieurs bonnes pratiques d'accessibilité, mais une évaluation plus approfondie et des tests pratiques sont nécessaires pour garantir une conformité complète aux normes WCAG.