Restructuration du Projet OnePick : Simplification et Amélioration de la Gestion du Code

Lors de ma prise en main du projet OnePick, plusieurs défis se sont immédiatement présentés, principalement liés à la structure des fichiers. Le projet, développé sans une architecture bien définie dès le départ, souffrait d'un manque d'organisation qui compliquait la compréhension globale du code. Par exemple, les pages du projet étaient disséminées sans logique apparente, ce qui rendait difficile leur identification et leur gestion. De même, les services, bien que regroupés dans un dossier dédié, étaient parfois dispersés dans d'autres emplacements inattendus, provoquant confusion et frustration. Ce désordre structurel augmentait non seulement le temps d'apprentissage, mais aussi la difficulté à naviguer et à travailler efficacement sur le projet.

Ces problèmes initiaux ont mis en évidence la nécessité d'une refonte structurelle pour améliorer l'expérience de développement et faciliter l'intégration des nouveaux arrivants. C'est dans ce contexte que j'ai entrepris de réorganiser le projet, en mettant en place une structure plus cohérente et intuitive. Cette initiative vise non seulement à résoudre les problèmes immédiats, mais aussi à poser les bases d'une gestion plus efficace et évolutive du code à long terme.

Problèmes de l'Ancienne Structure

L'ancienne structure de l'application OnePick présentait plusieurs lacunes qui rendaient le développement complexe et peu intuitif. Le projet avait été développé sans une structure clairement définie, ce qui a entraîné un flux de travail désorganisé.

Par exemple, les pages du projet n'étaient pas regroupées dans un dossier spécifique, rendant difficile l'identification et la localisation des différentes parties de l'application. Cela créait une confusion sur l'attribution des fichiers, amenant à se demander si un fichier appartenait à une page ou à un composant. Cette désorganisation s'étendait également aux services, dont certains étaient regroupés dans un dossier dédié, tandis que d'autres se trouvaient éparpillés à l'extérieur, suscitant des interrogations sur leur emplacement et leur rôle.

Ces problèmes de structure augmentaient non seulement la frustration des développeurs mais aussi leur incompréhension générale du projet. Un manque de cohérence dans l'organisation des fichiers rendait la maintenance difficile, et chaque nouvelle tâche ou ajout de fonctionnalité devenait un défi, car il était ardu de trouver les éléments nécessaires ou de comprendre leur interconnexion. En somme, l'ancienne structure ne facilitait pas le travail en équipe, et encore moins l'intégration de nouveaux arrivants.

Avec la nouvelle structure que j'ai mise en place, ces obstacles ont été largement atténués. L'objectif principal était d'améliorer l'expérience de développement en rendant la structure plus logique, intuitive, et facile à naviguer. Cependant, il est important de noter que de nombreux petits problèmes similaires subsistent dans l'application, et ils devront être abordés progressivement pour assurer une organisation optimale.

Nouvelle Structure

Structure du Répertoire `core`

La réorganisation de l'application OnePick se concentre sur une architecture modulaire et claire, visant à améliorer la gestion du code et la maintenabilité du projet. Le répertoire `core` est au cœur de cette nouvelle

structure, assurant le bon fonctionnement interne de l'application. Ce répertoire est divisé en trois sous-dossiers principaux : `adapters`, `models`, et `ports`. Chacun de ces sous-dossiers joue un rôle crucial dans l'architecture et contribue à la robustesse et à la flexibilité de l'application.

`adapters`

Le dossier `adapters` établit la connexion entre l'application et ses sources de données externes. Il contient des composants chargés de communiquer avec des services externes, des API, et d'autres systèmes. Les adaptateurs veillent à ce que les données provenant de ces sources soient correctement intégrées et utilisées dans l'application, tout en isolant les détails d'implémentation externe.

- **Fonctions Principales :** Les adaptateurs gèrent les appels API, les transformations de données, et assurent une intégration fluide avec des sources externes telles que les bases de données ou les services web.
- **Exemples :** Un adaptateur peut être responsable de l'envoi de requêtes au backend, de la récupération des données, ou de la conversion des données en un format compatible avec l'application.

`models`

Le dossier `models` contient les définitions des données utilisées dans l'application. Il regroupe les classes ou interfaces qui décrivent la structure des données manipulées, ainsi que les règles de validation et les relations entre les différentes entités. Ce dossier est fondamental pour maintenir la cohérence des données et faciliter leur gestion au sein de l'application. Il inclut également les `builders`, qui sont responsables de la construction des objets de manière efficace et cohérente.

- **Fonctions Principales :** Définir les structures de données, gérer les règles de validation, et faciliter la gestion des relations entre entités.
- **Exemples :** Les modèles définissent les objets de l'application, comme les utilisateurs ou les posts, ainsi que les méthodes associées pour manipuler ces objets.

`ports`

Le dossier `ports` est dédié aux interfaces qui définissent les points d'entrée et de sortie de l'application. Il spécifie les méthodes par lesquelles les services internes communiquent avec le reste de l'application, ainsi que les interfaces que les adaptateurs doivent implémenter. Cette séparation assure une meilleure organisation du code et une meilleure extensibilité.

- **Fonctions Principales :** Définir les interfaces pour la communication entre les services et les autres composants de l'application.
- **Exemples :** Les ports définissent les méthodes que les services doivent exposer et les interfaces que les adaptateurs doivent suivre pour intégrer les données.

Évolution Future - Core

La structure du dossier `core` est conçue pour être évolutive. À l'avenir, il est prévu d'ajouter des sous-dossiers supplémentaires, tels que `guards` et `interceptors`, pour enrichir l'architecture avec des fonctionnalités supplémentaires comme la gestion de la sécurité et des requêtes. Ces ajouts permettront d'améliorer encore la flexibilité et la robustesse de l'application.

Structure du Répertoire `shared`

Le répertoire `shared` joue un rôle crucial dans l'architecture modulaire de l'application OnePick. Il regroupe les ressources réutilisables qui sont partagées entre différents composants de l'application et qui peuvent également être utilisées dans d'autres projets. Cette organisation vise à promouvoir la réutilisabilité et à éviter la duplication de code, ce qui facilite la maintenance et l'évolution du projet. Le répertoire `shared` est divisé en plusieurs sous-dossiers : `services`, `pipes`, `directives`, et `components`. Chacun de ces sous-dossiers contribue à fournir des fonctionnalités communes à travers l'application.

`services`

Le dossier `services` contient des classes et des services qui fournissent des fonctionnalités communes et partagées entre différents composants de l'application. Ces services gèrent des opérations comme la gestion des données, la communication avec des APIs, ou encore la gestion des états applicatifs.

- **Fonctions Principales :** Fournir des services communs comme la gestion des requêtes HTTP, la gestion des états globaux, et d'autres fonctionnalités utilitaires utilisées à travers l'application.
- **Exemples :** Un service de gestion des utilisateurs, un service de configuration, ou un service de communication avec une API externe.

`pipes`

Le dossier `pipes` regroupe les pipes personnalisés qui transforment les données à afficher dans l'application. Les pipes permettent de formater, filtrer, ou transformer les données de manière à ce qu'elles soient présentées de manière cohérente et appropriée dans l'interface utilisateur.

- **Fonctions Principales :** Transformer et formater les données avant qu'elles ne soient affichées à l'utilisateur.
- **Exemples :** Un pipe pour formater les dates, un pipe pour convertir des valeurs numériques en pourcentages, ou un pipe pour filtrer des listes d'éléments.

`directives`

Le dossier `directives` contient des directives personnalisées qui modifient le comportement ou l'apparence des éléments du DOM. Les directives permettent d'ajouter des comportements spécifiques aux éléments HTML ou de modifier leur présentation de manière dynamique.

- **Fonctions Principales :** Appliquer des comportements ou des styles spéciaux aux éléments du DOM en fonction de conditions ou d'interactions spécifiques.
- **Exemples :** Une directive pour ajouter des effets visuels lors du survol d'un élément, ou une directive pour gérer les comportements de validation des formulaires.

`components`

Le dossier `components` regroupe les composants réutilisables à travers l'application. Ces composants sont des blocs de construction de l'interface utilisateur, conçus pour être utilisés dans différentes parties de l'application, ce qui facilite la création d'interfaces cohérentes et maintenables.

- **Fonctions Principales :** Fournir des composants d'interface utilisateur qui peuvent être réutilisés dans différentes vues de l'application.
- **Exemples :** Des boutons, des cartes, des formulaires, ou des dialogues modaux.

Évolution Future - Shared

Le répertoire `shared` est conçu pour évoluer avec les besoins du projet. À l'avenir, d'autres types de ressources ou fonctionnalités réutilisables pourraient être ajoutés pour enrichir encore le répertoire et améliorer la réutilisabilité du code.

Structure du Répertoire `views`

Le répertoire `views` est au cœur de l'interface utilisateur de l'application OnePick. Il regroupe toutes les vues, c'est-à-dire les pages et les modals, qui constituent les différents écrans de l'application. Cette structure vise à organiser les éléments de manière cohérente, facilitant ainsi la navigation et la maintenance du code. Le répertoire `views` est divisé en deux sous-dossiers principaux : `pages` et `modals`. Chacun de ces sous-dossiers joue un rôle spécifique dans la présentation des contenus aux utilisateurs.

`pages`

Le dossier `pages` contient les pages principales de l'application, chacune représentant un écran ou une section majeure que l'utilisateur peut visiter. Chaque page est généralement associée à une route dans l'application, permettant une navigation claire et structurée.

- **Fonctions Principales :** Regrouper les écrans principaux de l'application, chacun correspondant à une fonctionnalité ou à une section spécifique.
- **Exemples :** Une page d'accueil, une page de profil utilisateur, une page de gestion des posts, ou une page de paramètres.

`modals`

Le dossier `modals` contient les fenêtres modales de l'application, utilisées pour afficher des contenus ou des actions supplémentaires sans quitter la page actuelle. Les modals sont souvent utilisées pour des interactions contextuelles, comme des confirmations, des formulaires rapides, ou des informations détaillées.

- **Fonctions Principales :** Fournir des fenêtres contextuelles qui permettent des interactions rapides ou des informations supplémentaires sans perturber le flux principal de l'utilisateur.
- **Exemples :** Un modal de confirmation de suppression, un modal pour éditer un post, ou un modal d'alerte.

Évolution Future - Views

Le répertoire `views` est conçu pour s'adapter aux futures évolutions de l'application. À mesure que de nouvelles fonctionnalités sont ajoutées, de nouvelles pages ou modals pourront être créées et organisées au sein de cette structure. Cette flexibilité assure que l'interface utilisateur peut croître avec les besoins de l'application tout en maintenant une organisation claire et compréhensible.

Transition pour les Développeurs Frontend Futurs

En tant que futur développeur frontend travaillant sur le projet OnePick, il est essentiel de comprendre que cette réorganisation a été conçue pour faciliter votre intégration et accélérer votre prise en main du projet. La nouvelle structure, bien qu'elle puisse paraître différente de ce que vous avez pu rencontrer auparavant, a été pensée pour améliorer la lisibilité, la maintenabilité, et l'efficacité du développement.

Objectifs de la Nouvelle Structure

La réorganisation de l'application vise à réduire la complexité et à clarifier les responsabilités de chaque partie du code. En vous familiarisant avec les répertoires `core`, `shared`, et `views`, vous aurez une compréhension claire des rôles de chaque composant et de la manière dont ils interagissent. Cette structure vous permettra de naviguer facilement dans le code et d'identifier rapidement les éléments à modifier ou à étendre.

Stratégies pour une Transition Efficace

1. **Prendre le temps d'explorer la structure** : Avant de plonger dans le code, parcourez les différents répertoires et examinez les fichiers pour comprendre leur rôle. Cela vous aidera à voir comment les éléments sont liés les uns aux autres.

2. **Se référer à la documentation** : Des commentaires et des documents explicatifs ont été ajoutés pour clarifier les choix architecturaux. Utilisez-les comme point de référence pour comprendre les intentions derrière chaque dossier et chaque fichier.

3. **Adopter les bonnes pratiques** : La nouvelle structure encourage l'utilisation de composants réutilisables et bien encapsulés, ainsi que l'application de modèles de conception clairs. Adoptez ces pratiques dès le début pour maintenir la cohérence du code.

4. **Communication et collaboration** : Même si vous travaillez seul au début, n'hésitez pas à documenter vos changements et à partager vos idées sur l'amélioration continue de la structure. Cette documentation sera précieuse pour les futurs membres de l'équipe.

En vous familiarisant avec cette structure dès le départ, vous serez mieux préparé à contribuer efficacement au projet OnePick. La clarté et l'organisation du code faciliteront non seulement votre travail quotidien, mais aussi celui de tous ceux qui suivront.

Conclusion

La restructuration de l'application OnePick marque une étape cruciale dans l'évolution du projet, en mettant l'accent sur la simplicité, la clarté, et la maintenabilité du code. En organisant les différents composants de l'application au sein des répertoires core, shared, et views, nous avons jeté les bases d'une architecture plus robuste et adaptable, qui facilitera non seulement le travail des développeurs actuels mais aussi l'intégration des futurs membres de l'équipe.

Cette nouvelle structure, bien que conçue pour répondre aux besoins actuels, est également pensée pour évoluer avec le projet. Elle est le fruit d'une réflexion approfondie sur les défis auxquels nous avons été confrontés, et vise à prévenir les problèmes similaires à l'avenir.

Les futurs développeurs frontend pourront ainsi s'appuyer sur une base solide pour contribuer efficacement au projet, en comprenant rapidement le rôle de chaque composant et en adoptant les bonnes pratiques encouragées par cette réorganisation.

En somme, cette restructuration n'est pas seulement une réponse aux problèmes du passé, mais un investissement dans l'avenir du projet OnePick, garantissant une expérience de développement plus fluide et plus satisfaisante pour tous les membres de l'équipe.