
strapi
DESIGN SYSTEM

Table des matières

Introduction

Strapi est une plateforme révolutionnaire qui a changé la façon dont les développeurs interagissent avec les systèmes de gestion de contenu (CMS). Avec une flexibilité remarquable et des options de personnalisation étendues, Strapi s'impose rapidement comme un incontournable dans le monde en évolution rapide des applications web et mobiles. Ce document vise à offrir une vue exhaustive de Strapi, ses avantages, fonctionnalités, et guides de déploiement.

Qu'est-ce que Strapi (Headless CMS)

Histoire et genèse de Strapi

Strapi émerge dans un contexte où l'agilité et l'adaptabilité sont devenues primordiales dans le développement web. En réponse aux limitations des CMS traditionnels, Strapi propose une architecture "headless" révolutionnaire, qui sépare le back-end gestionnaire de contenu de la couche front-end. Cette séparation offre une liberté sans précédent aux développeurs, qui peuvent désormais concevoir l'interface utilisateur selon leurs propres spécifications, en utilisant les frameworks ou les bibliothèques de leur choix, tels que React, Vue.js ou Angular.

Les principes d'un CMS headless

Un CMS headless met l'accent sur une architecture back-end robuste qui s'expose à travers des API, déconnectée de toute front-end prédéterminée. Cela donne naissance à une flexibilité opérationnelle propice à l'intégration sur diverses plateformes comme les sites web, les applications mobiles, les objets connectés (IoT) et même les plateformes de réalité augmentée ou virtuelle. L'approche headless est particulièrement bien adaptée aux stratégies de contenu omnicanal, où l'expérience utilisateur continue doit transiter sur plusieurs interfaces et dispositifs.

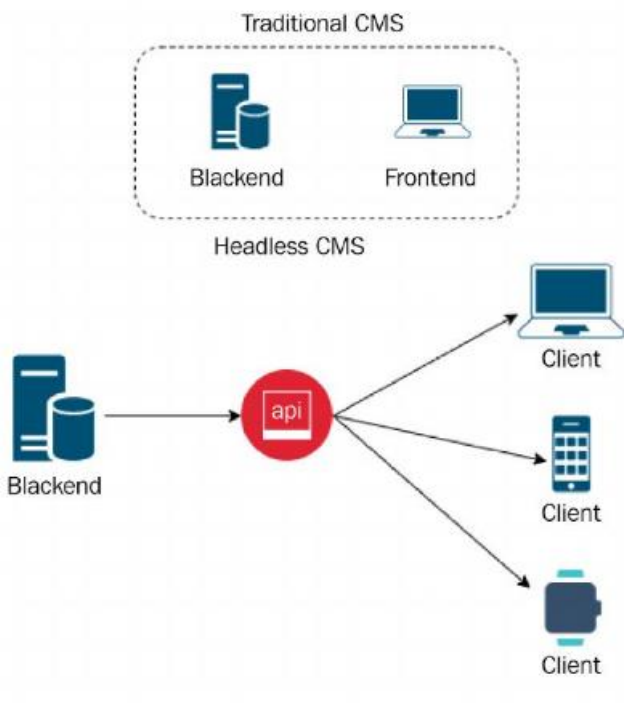


Figure 1.1: Traditional CMS versus headless CMS

[1]

L'architecture et l'écosystème de Strapi

Bâti sur Node.js, un environnement d'exécution JavaScript côté serveur, Strapi se distingue par sa rapidité et sa flexibilité. En offrant un support natif pour REST et GraphQL, Strapi libère les développeurs des contraintes d'interfaçage et leur permet de manipuler des contenus de manière riche et variée. L'écosystème de Strapi, enrichi par des plugins et une communauté engagée, permet également d'étendre les fonctionnalités de base pour répondre à des besoins spécifiques et complexes.

Les avantages de Strapi

Flexibilité et personnalisation

La notoriété de Strapi s'ancre dans sa capacité exceptionnelle à s'adapter aux processus de développement existants, tout en permettant une personnalisation poussée de son interface d'administration. Les utilisateurs peuvent modifier les fonctionnalités de base, intégrer des plugins tiers ou développer les leurs, optimisant ainsi Strapi pour répondre exactement aux requêtes de leurs projets.

[1] Designing Web APIs with Strapi, p.4

Performance optimisée

Avec son architecture dépourvue de tête (headless), Strapi est conçu pour servir efficacement le contenu dans sa forme la plus pure, ce qui se traduit par de meilleures performances d'application grâce à des temps de chargement réduits et une expérience utilisateur améliorée. Cette efficacité se ressent tant au niveau de la rapidité de réponse de l'API que de la souplesse avec laquelle le contenu est adapté et réparti sur différentes plateformes.

Communauté et support

Strapi bénéficie d'une communauté vibrante et en plein essor, composée de développeurs passionnés, de contributeurs et d'entreprises adoptant la plateforme. La force de cette communauté tient à son partage dynamique de connaissances, à sa contribution régulière à l'évolution de la plateforme et à la fourniture d'un support étendu. Cela se traduit par une documentation fournie, des tutoriels détaillés et des forums d'échanges où les utilisateurs peuvent obtenir de l'aide sur des aspects techniques ou des conseils sur les meilleures pratiques de déploiement.

Compatibilité avec les technologies modernes

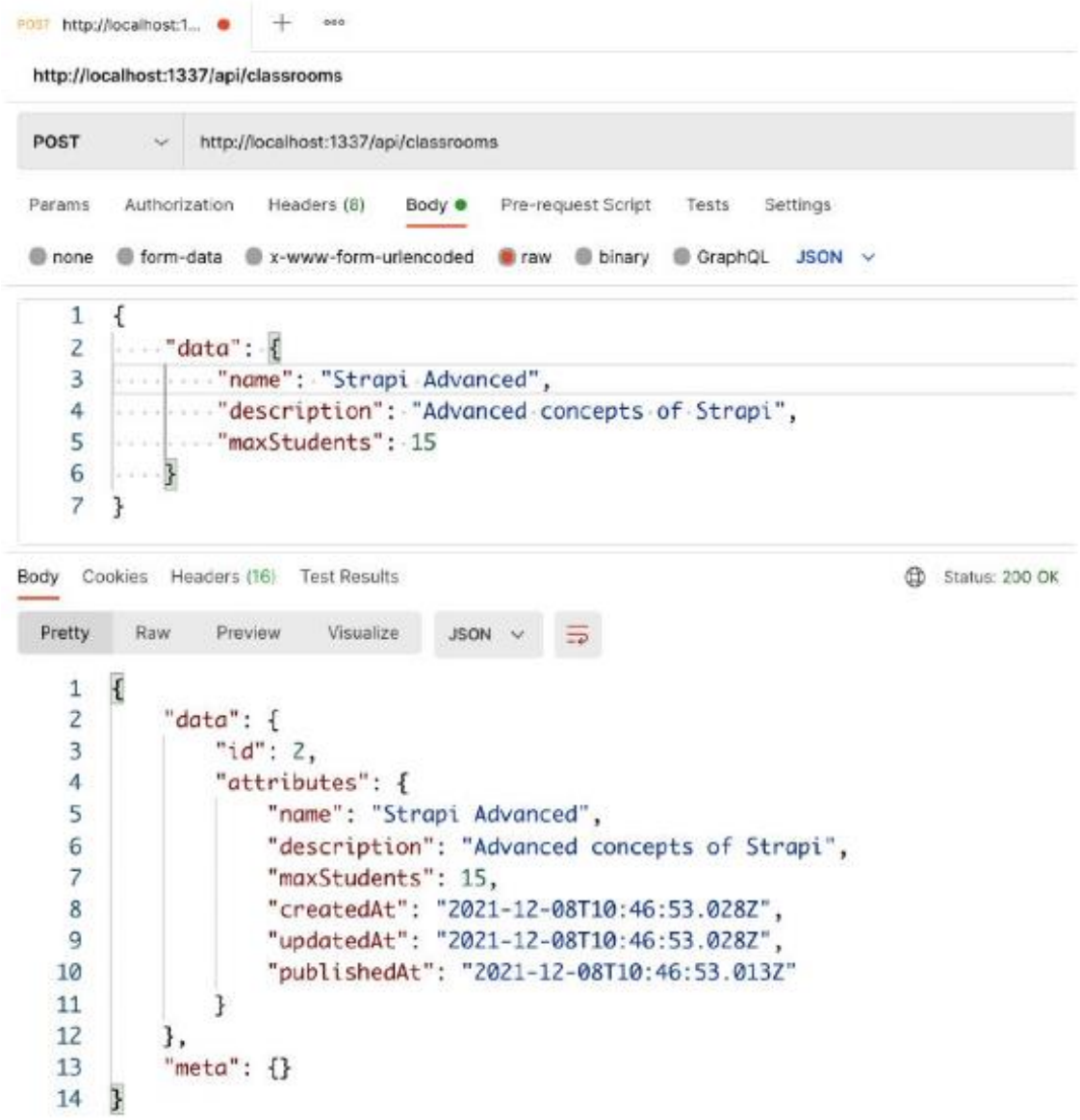
Strapi est expressément conçu pour fonctionner de concert avec l'éventail des technologies front-end et back-end actuelles. Que ce soit pour une utilisation avec des bibliothèques populaires comme React, des frameworks progressifs tels que Vue.js et Angular, ou pour le développement d'applications mobiles via des plateformes comme React Native ou Flutter, Strapi offre une compatibilité transparente, permettant une intégration efficace dans l'écosystème de développement moderne.

RESTful et GraphQL

Les fondamentaux de REST API avec Strapi

Strapi donne aux développeurs la liberté et le contrôle complets sur la déclaration, la sécurisation et l'exploitation des API REST. En supprimant les complexités inhérentes à la création d'API, Strapi rend le processus plus intuitif et fait gagner beaucoup de temps, tout en conservant la puissance et la flexibilité nécessaires pour construire des structures de données élaborées.

Nous avons accès à un requêteur d'API Rest & GraphQL forunit par Strapi comme cela pour tester les différents endpoints :



GraphQL dans Strapi

Strapi démystifie l'intégration de GraphQL dans les processus de développement en rendant la création d'API efficaces et performantes aussi simple que possible. Par le biais d'un puissant éditeur de requêtes et de mutations intégré, les développeurs peuvent exploiter toute la puissance de GraphQL pour minimiser le sur-échange de données et rationaliser les interactions client-serveur. Pour ce qui est du GraphQL, seul un endpoint pour toutes ressources est présent !

Avantages de l'utilisation de GraphQL contre REST

Les requêtes GraphQL sont intrinsèquement optimisées, car elles permettent aux clients de demander précisément les données dont ils ont besoin, sans les surcharges souvent associées aux standards REST. Cette efficacité réduit le poids des échanges de données sur le réseau, favorisant ainsi des performances supérieures et une meilleure réactivité de l'application. Parallèlement,

GraphQL facilite la maintenance et l'évolution des API en raison de sa nature fortement typée et auto-documentée.

Routes, controllers, services et models

Définition et rôles

Strapi décompose la logique applicative en éléments modulaires tels que les routes, les controllers, les services et les models. Les routes définissent les chemins d'accès, les controllers orchestrent la logique métier, les services gèrent les règles et opérations liées aux données, tandis que les models déterminent la structure des données. Cette modularité crée un système où chaque composant peut être personnalisé ou étendu pour répondre aux exigences spécifiques d'une application, sans toutefois perturber le fonctionnement cohérent et sécurisé de l'ensemble du système.

Voici la structure du dossier racine de Strapi :

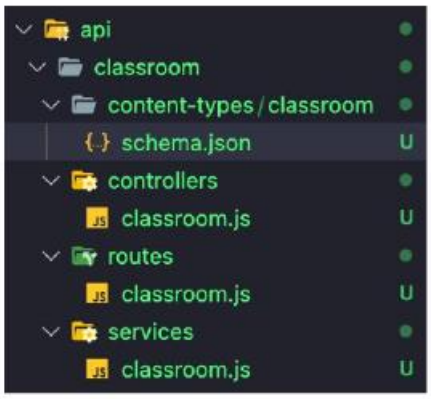


Figure 2.15: The classroom content-type folder structure

[2]

Routes

Les routes dans Strapi sont des déclarations qui correspondent aux URL demandées par les clients et qui déterminent quelle fonction (dans un contrôleur) sera exécutée lorsqu'une certaine URL est visitée. Chaque route est associée à une méthode HTTP (comme GET, POST, PUT, DELETE) et à une URL spécifique. Les routes sont définies dans des fichiers de configuration au sein de Strapi, habituellement dans le dossier `src/api/[api-name]/routes`.

La configuration d'une route inclut le chemin, la méthode HTTP, le contrôleur et la méthode du contrôleur à exécuter.

Controllers

Les contrôleurs sont des fichiers qui contiennent des fonctions logiques utilisées pour répondre aux requêtes entrantes. Ces fonctions reçoivent les données de la requête (comme les paramètres

[2] Designing WEB APIs with Strapi p.31

d'URL, les paramètres de requête et le corps de la requête), interagissent avec les services pour accéder ou modifier les données, et renvoient des réponses au client.

Les contrôleurs dans Strapi sont généralement situés dans le dossier `src/api/[api-name]/controllers`.

Services

Les services sont l'endroit où la logique business est défini. Ce sont des classes ou des ensembles de fonctions qui encapsulent la logique business de l'application, indépendamment du protocole de communication utilisé pour les interfaces utilisateur. Les services sont appelés par les contrôleurs et peuvent inclure des opérations comme la création, la lecture, la mise à jour, ou la suppression de données (CRUD), ainsi que d'autres opérations métier complexes.

Les services dans Strapi se trouvent dans le dossier `src/api/[api-name]/services`.

Content-types

Les Content-Types, ou types de contenu, sont des modèles de données définis par les utilisateurs qui décrivent la structure des données à gérer par l'API. Ils sont définis via le Strapi admin panel ou en créant des fichiers de configuration JSON ou JavaScript dans le dossier `src/api/[api-name]/content-types`.

Chaque Content-Type définit les champs et les types de données que ces champs peuvent contenir, ainsi que des validations et des relations avec d'autres Content-Types.

Interaction entre eux

Dans un flux typique de Strapi :

1. Route : Une requête HTTP est reçue et la route correspondante est identifiée.

2. Controller : La fonction spécifiée dans le contrôleur associé à la route est invoquée.

3. Service : La fonction du contrôleur peut ensuite appeler une ou plusieurs fonctions de service pour accomplir la logique business nécessaire, comme l'interaction avec la base de données.

4. Content-Type : Les services interagissent avec les Content-Types pour effectuer des opérations sur les données (grâce à l'ORM de Strapi).

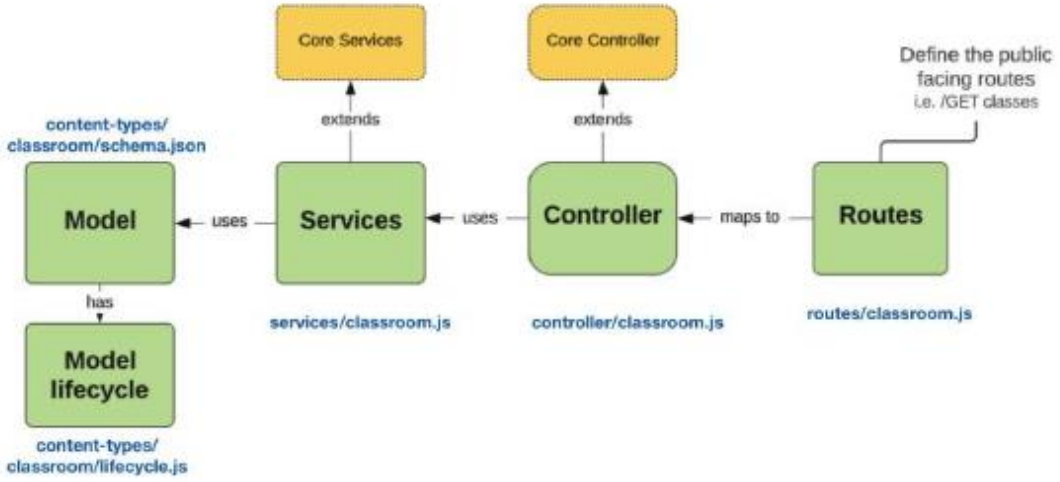


Figure 2.17: An overview of the Strapi core components [3]

Strapi et Content-Type

Interface et navigation

L'interface utilisateur de Strapi est conçue pour être à la fois simple et intuitive, permettant une navigation aisée et une gestion efficace du contenu.

Gestion des contenus

Strapi dispose d'une suite robuste d'outils pour la gestion de contenu, y compris des fonctionnalités avancées de tri, de filtration et de recherche, ainsi que des options de personnalisation des vues. Il offre également des paramètres de rôles et de permissions, permettant ainsi une granularité du contrôle d'accès, essentielle pour sécuriser les contenus et les opérations au sein d'équipes de toutes tailles.

Création de Content-Types

Strapi donne aux développeurs l'aisance de créer des "Content-Types" (types de contenu) personnalisés avec une flexibilité et une simplicité inégalée. La plateforme permet d'ajouter, de configurer et de gérer des champs de données variées qui peuvent correspondre à des types primitifs comme des chaînes de caractères, des nombres, des booléens, mais aussi à des types de données complexes comme des fichiers, des relations ou des types composés. Ces capacités élargissent considérablement la façon dont les développeurs peuvent modéliser les informations dans leurs applications, permettant ainsi de concevoir des architectures de données qui répondent précisément aux besoins des utilisateurs finaux.

[3] Designing Web APIs with Strapi p.34

Collection types

Utilité : Les types de collection représentent généralement des ensembles d'enregistrements structurés et sont utilisés pour gérer des données répétitives. Ces enregistrements sont créés, lus, mis à jour et supprimés via l'API de Strapi et peuvent être liés à d'autres enregistrements ou types de contenu.

Quand choisir : Optez pour un type de collection lorsque vous avez besoin de stocker et de gérer un ensemble d'entités identiques ou similaires, comme des articles de blog, des produits, des utilisateurs ou des commentaires.

Single types

Utilité : Les types uniques sont conçus pour gérer des enregistrements uniques qui représentent des configurations globales, des pages uniques, ou des structures de données qui n'ont pas besoin d'être répétées.

Quand choisir : Utilisez un type unique lorsque vous avez besoin de créer un contenu qui n'aura pas d'instances multiples, comme une page d'accueil, une page « À propos » ou des paramètres de site.

Component types

Utilité : Les composants sont des blocs de contenu réutilisables qui peuvent être insérés dans des collections ou des types uniques. Ils sont utiles pour créer des structures de données modulaires et réutilisables.

Quand choisir : Sélectionnez des composants lorsque vous souhaitez utiliser le même bloc de champs dans différents types de contenu ou lorsque vous avez besoin de modularité, comme pour les sections répétables d'une page web ou une liste de témoignages.

Dynamic zones

Utilité : Les zones dynamiques permettent de combiner différents composants dans un ordre personnalisable au sein d'un type de collection ou unique. Cela offre une grande flexibilité pour la création de pages ou de sections de contenu qui nécessitent une disposition variable.

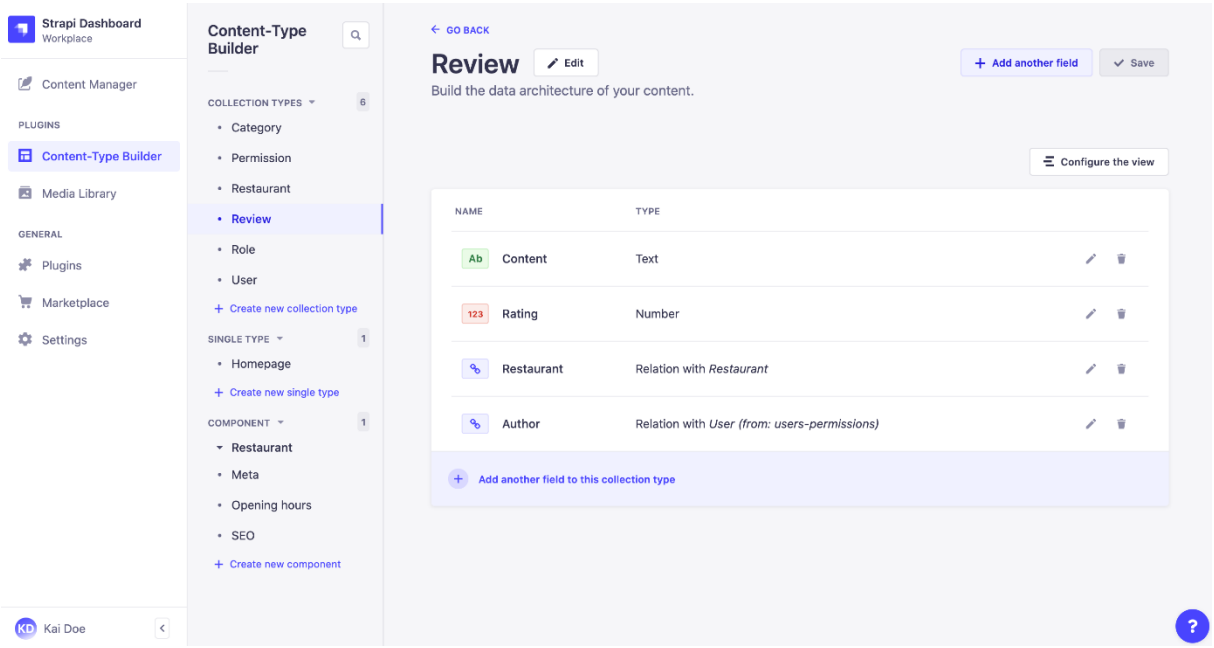
Quand choisir : Optez pour des zones dynamiques quand vous avez besoin de créer un contenu qui peut changer dynamiquement en termes de structure et de composants, comme des pages de contenu riche avec une mise en page personnalisée.

Relation fields

Utilité : Les champs de relation définissent des liens entre différents types de contenu, ce qui permet de créer des relations complexes entre les données, telles que les relations un-à-un, un-à-plusieurs et plusieurs-à-plusieurs.

Quand choisir : Implémentez des champs de relation chaque fois que vous avez besoin de connecter des enregistrements à travers différents types de contenu, par exemple, pour lier des articles à leur(s) auteur(s), ou pour associer des produits à des catégories ou des tags.

Voici une vue de l'interface de Strapi :



Gestion des champs et types de données

L'utilisation de champs et de types de données personnalisés contribue à la précision et l'intégrité du contenu stocké, permettant ainsi une meilleure qualité de données.

Voici une liste des différents types de données que Strapi gère pour la base de données :

1. String (chaîne de caractères) : Type de données pour stocker de courts textes, comme des titres, des noms ou de petites descriptions. Il est limité en taille et il est parfait pour des informations courtes et précises.
2. Text (texte) : Semblable à la chaîne de caractères, mais utilisé pour le stockage de textes plus conséquents. Il n'a pas de limite de taille définie, ce qui le rend idéal pour des paragraphes entiers ou des articles.
3. Rich Text (texte riche) : Ce champ offre un éditeur WYSIWYG (What You See Is What You Get) pour la saisie de contenu. Il permet de formater le texte avec des options telles que le gras, l'italique, les listes à puces, et d'ajouter des hyperliens ou des images. Parfait pour le contenu de blog ou les articles qui requièrent une mise en forme avancée.
4. Number (nombre) : Utilisé pour stocker des chiffres, qui peuvent être entiers ou flottants. Ce type est adapté pour tout élément quantifiable, tel que l'âge, le prix, ou toute autre mesure numérique.

5. Integer (entier) : Une sous-catégorie de Number, spécifiquement pour les nombres entiers (sans décimales), tels que le comptage d'articles, l'année de publication, etc.
6. Float (nombre à virgule flottante) : Une autre sous-catégorie de Number, pour les nombres avec des décimales, comme le poids, les dimensions, ou les taux d'intérêt.
7. Boolean (booléen) : Type de donnée qui peut être vrai ou faux. Idéal pour des états binaires, comme la publication (publié / non-publié), ou des options de oui/non.
8. Date : Utilisé pour stocker des dates et des heures. Vous pouvez spécifier simplement une date, ou une date avec l'heure.
9. Enumeration (énumération) : Enum est un champ qui permet de choisir parmi un ensemble de valeurs prédéfinies. Utile pour une liste déroulante ou un choix de boutons radio, par exemple, le statut d'une commande (en attente, expédiée, livrée).
10. JSON : Permet de stocker un objet ou un tableau JSON. C'est idéal pour des données structurées complexes qui ne cadrent pas bien avec les autres types de champs.
11. Media (média) : Permet d'ajouter des fichiers médias comme des images, des vidéos, des fichiers audios ou des documents. Strapi fournit une gestion intégrée des médias avec la possibilité de charger, de stocker, et de récupérer des médias.
12. Relation (relation) : Ce type de champ est utilisé pour créer des relations entre différentes entités dans votre projet Strapi. Il peut s'agir d'une relation un-à-un, un-à-plusieurs, ou plusieurs-à-plusieurs, et sert à lier des contenus entre eux, comme des articles à des auteurs ou des produits à des catégories.
13. UID (Identifiant Unique) : Un champ qui permet de créer un identifiant unique et convivial pour les URL ou les slugs. Il est généralement utilisé pour les titres d'articles ou de pages afin de générer une URL conviviale pour le référencement.
14. Component (composant) : Les composants sont des structures de données réutilisables qui peuvent être incluses dans d'autres types de contenu. Ils sont similaires aux blocs dans d'autres CMS et sont utiles pour créer des sections de mise en page répétables.
15. Dynamic Zone (zone dynamique) : Une zone qui permet d'ajouter une combinaison flexible de composants et autres types de champs à l'intérieur d'un même champ. Cela offre une flexibilité maximale pour composer du contenu personnalisé pour des cas d'utilisation spécifiques.

Types de relations et configurations

Strapi propose une gestion avancée des relations entre les "Content-Types", permettant ainsi de modéliser des structures de données complexes et interconnectées. Que ce soit pour des associations simples comme "un à un", "un à plusieurs", ou pour des relations plus complexes comme "plusieurs à plusieurs", Strapi rend la création et la gestion de ces liens entre les données intuitives et puissantes. L'API de Strapi, ainsi que le panel administratif, fournissent les outils nécessaires pour gérer efficacement ces relations, qu'il s'agisse de l'ajout, la mise à jour ou la récupération de données interdépendantes.

Personnaliser le backend de Strapi (Controller)

Le custom controller

Les controllers dans Strapi jouent un rôle crucial en agissant comme intermédiaires entre la base de données et les vues (dans le contexte d'un headless CMS, les "vues" réfèrent souvent aux consommateurs de l'API). Le but principal d'un controller est de traiter les requêtes entrantes, interagir avec le modèle de données pour lire ou écrire des informations, et enfin, renvoyer la réponse appropriée au client. Personnaliser les controllers est un moyen efficace de personnaliser la logique métier et les réponses de l'API selon les besoins spécifiques d'une application.

Leur création

Strapi permet de créer des controllers personnalisés pour surcharger les comportements par défaut ou en ajouter de nouveaux. La personnalisation peut inclure l'ajout de validations, la modification de la logique de traitement des données ou l'implémentation de nouvelles opérations métier. Pour personnaliser un controller, on peut l'étendre ou le surcharger dans le dossier `/api/[model]/controllers`. Cette approche permet de mieux contrôler le flux de données et d'assurer que les opérations sur les données sont gérées correctement.

Un exemple de custom controller

Un exemple de personnalisation pourrait être la modification du controller de création d'une entité pour inclure des étapes de validation ou de transformation des données avant qu'elles soient persistées dans la base de données. Un autre exemple courant est la personnalisation de la logique de récupération des données, où un développeur pourrait vouloir modifier le comportement par défaut de l'API pour inclure des relations ou appliquer des filtres personnalisés.

Utiliser les hooks

Qu’est-ce qu’un hook ?

Les hooks sont des fonctions qui peuvent être utilisées pour personnaliser et étendre la fonctionnalité de Strapi à différentes étapes du cycle de vie des requêtes et des modèles. Ils permettent aux développeurs d'exécuter du code personnalisé lorsqu'un événement spécifique survient dans l'application, par exemple avant ou après la création, la mise à jour, la suppression d'une entité, ou même du démarrage de l'application.

Type de Hooks

Il existe deux types principaux de hooks dans Strapi : les hooks de modèle et les hooks de lifecycle. Les hooks de modèle permettent d'effectuer des tâches spécifiques liées à la manipulation des données, tandis que les hooks de lifecycle sont utilisés pour exécuter du code à certains moments du cycle de vie des requêtes. Pour utiliser un hook, vous devez le définir dans la configuration du modèle ou dans les fichiers de configuration globale.

Exemple concret d’un hook

Un exemple concret d'utilisation de hook pourrait être la création d'un hook de lifecycle `beforeCreate` sur un modèle utilisateur pour hacher un mot de passe avant qu'il ne soit stocké dans la base de données. Un autre exemple serait l'utilisation d'un hook de modèle pour envoyer un email de notification lorsqu'un nouvel article est publié sur le site.

Comment faire des requêtes à l’API (filtering, sorting, populate etc…)

Les requêtes API

Faire des requêtes à l'API est fondamental pour interagir avec un headless CMS tel que Strapi. Ces requêtes permettent de récupérer des données, d'effectuer des mises à jour, de supprimer des ressources ou de créer de nouvelles entrées. Strapi rend cette interaction intuitive grâce à une API REST ou GraphQL qui supporte des opérations avancées telles que le filtrage, le tri ou le peuplement de relations entre données (populate).

Filtrage et tri des données

Le filtrage est crucial pour obtenir des sous-ensembles de données basés sur des critères spécifiques. Dans une requête Strapi, le filtrage se fait via des paramètres dans l'URL qui spécifient les champs et les conditions à appliquer. Le tri est également un aspect important de l'interaction avec l'API, permettant d'ordonner les résultats selon un ou plusieurs champs. Les développeurs peuvent facilement implémenter ces opérations via des paramètres d'URL standardisés

Utilisation de populate

Dans un CMS headless, la gestion des relations entre les différents types de contenu est essentielle. Strapi offre la fonction ‘populate’ pour inclure des relations dans les résultats de requête, ce qui rend les données liées directement accessibles. Cela est souvent utilisé pour récupérer des informations connexes, par exemple, afficher un article avec les informations de l'auteur ou lister des produits avec leurs catégories respectives.

Authentification et autorisation

Qu’est ce que l’authentification et l’autorisation

L'authentification et l'autorisation sont deux composants fondamentaux de la sécurité d'une application web. L'authentification, dans Strapi, se réfère au processus de vérification de l'identité d'un utilisateur, tandis que l'autorisation concerne les permissions accordées à cet utilisateur pour accéder à certaines fonctionnalités ou données. Strapi fournit des fonctionnalités robustes pour gérer les deux, permettant aux développeurs d'implémenter des systèmes sécurisés facilement.

Comment cela fonctionne sous le capot

Pour l'authentification, Strapi propose des routes intégrées qui permettent la gestion des utilisateurs tels que la création de compte, la connexion, et la réinitialisation de mot de passe. Ces fonctionnalités peuvent être étendues ou personnalisées selon les besoins spécifiques du projet. Les tokens JWT (JSON Web Tokens) sont utilisés pour maintenir l'état de connexion, offrant une méthode sécurisée pour vérifier les utilisateurs à travers les requêtes.

Gestion avec les rôles et permissions

Quant à l'autorisation, Strapi offre un système de gestion des rôles et des permissions flexibles. Les développeurs peuvent définir des rôles spécifiques et associer des permissions à ces rôles, contrôlant ainsi l'accès aux différentes opérations de l'API. Cela peut aller de la permission de lire uniquement certains types de contenu à la permission de modifier ou de supprimer des données. Les permissions peuvent être ajustées dans le panneau d'administration de Strapi, permettant une gestion fine des droits d'accès.

Les politiques dans Strapi

Qu’est-ce qu’une politique ?

Les politiques dans Strapi sont des fonctions qui sont exécutées avant que le controller ne traite la requête. Elles sont utilisées pour définir des règles de sécurité ou des conditions avant l'exécution de certaines actions. Cela permet aux développeurs de contrôler et de sécuriser l'accès aux différentes routes de l'API en s'assurant que certaines conditions sont remplies avant de poursuivre le traitement des requêtes.

La création d’une politique

Pour créer une politique personnalisée, il suffit de déclarer une nouvelle fonction dans le dossier `/api/[model]/policies` et d'indiquer cette politique dans la configuration des routes. Une politique peut valider des paramètres de demande, vérifier des en-têtes ou exécuter toute autre logique nécessaire avant que le contrôleur associé ne soit appelé. Les politiques sont un outil puissant pour imposer la segmentation des utilisateurs, les limitations des taux d'accès, ou la validation des requêtes.

Exemple d’une politique

Un exemple pratique pourrait être une politique qui vérifie si un utilisateur a confirmé son adresse e-mail avant de lui permettre d'accéder à certaines routes. Une autre politique pourrait limiter le nombre de requêtes qu'un utilisateur peut faire à une partie spécifique de l'API, agissant comme un mécanisme de limitation. Les politiques sont essentielles pour renforcer la sécurité et l'intégrité des applications Strapi.

Déployer Strapi sur AWS Fargate

AWS Fargate est un moteur de calcul pour Amazon ECS et EKS qui vous permet d'exécuter des conteneurs sans avoir à gérer des serveurs ou des clusters de serveurs. Vous n'avez pas à dimensionner, provisionner ou gérer des clusters de serveurs, vous lancez simplement vos applications et Fargate s'occupe de tout le reste en ce qui concerne la gestion de l'infrastructure.

Prérequis (Compte AWS + AWS CLI + Docker installé localement)

Compte AWS : Pour déployer Strapi sur AWS, vous devrez tout d'abord avoir un compte AWS. Si vous n'avez pas encore de compte, vous pouvez en créer un sur https://aws.amazon.com/.

AWS CLI : L'interface de ligne de commande AWS (AWS CLI) est un outil unifié pour gérer vos services AWS. Vous pouvez le télécharger et l'installer en suivant les instructions sur https://aws.amazon.com/cli/. Une fois installé, configurez-le en exécutant `aws configure` pour saisir vos informations d'identification.

Docker installé localement : Docker est requis pour créer une image conteneur pour Strapi. Vous pouvez télécharger et installer Docker Desktop à partir de https://www.docker.com/products/docker-desktop.

Création de l'image docker pour Strapi

Veuillez vous référer à la documentation officielle sur le site de Strapi pour avoir un Dockerfile respectant les dernières versions de Strapi mais aussi pour qu'il soit adapté à un environnement de production !

1. Créer un fichier `Dockerfile` dans votre projet Strapi, qui spécifie comment construire l'image Docker. Exemple de contenu pour le Dockerfile :

```
FROM node:14-alpine

WORKDIR /app

COPY package.json yarn.lock ./

RUN yarn install --frozen-lockfile

COPY . .

RUN yarn build
```

2. Construisez l'image Docker en exécutant la commande suivante depuis le répertoire de votre projet Strapi :

```
docker build -t mon-tag-dimage-strapi .
```

3. Une fois l'image construite, vous pouvez la tester localement en exécutant :

```
docker run -p 1337:1337 mon-tag-dimage-strapi
```

Cela vous permet de vérifier que Strapi démarre correctement dans le conteneur.

Création d'un dépôt docker sur AWS ECR

AWS ECR (Elastic Container Registry) est un service de stockage d'images Docker. Pour y pousser l'image de Strapi :

1. Connectez-vous à AWS ECR grâce à la CLI, créez un repository :

```
aws ecr create-repository --repository-name mon-repo-strapi --region votre-region-aws
```

2. Authentifiez Docker à ECR:

```
aws ecr get-login-password --region votre-region-aws | docker login --username AWS --password-stdin votre-id-aws.dkr.ecr.votre-region-aws.amazonaws.com
```

3. Marquez votre image Docker avec les informations d'ECR :

```
docker tag mon-tag-dimage-strapi:votre-version votre-id-aws.dkr.ecr.votre-region-aws.amazonaws.com/mon-repo-strapi:votre-version
```

4. Poussez l'image vers ECR :

```
docker push votre-id-aws.dkr.ecr.votre-region-aws.amazonaws.com/mon-repo-strapi:votre-version
```

Création d'un cluster AWS Fargate

Pour créer un cluster AWS Fargate :

1. Aller à la console ECS (Elastic Container Service).
2. Cliquer sur 'Create Cluster' et choisir 'networking only' (utilisé pour Fargate).
3. Donner un nom à votre cluster et cliquer sur 'create'.
4. Créer un service Fargate qui utilise l'image de Strapi que vous avez stockée dans ECR et configurez les paramètres (Cpu, mémoire, VPC, Subnets, Security Groups, Policy Roles).
5. Lancer le service, AWS déploiera une tâche Fargate basée sur votre service et démarrera votre conteneur Strapi.

Une fois le service lancé, vous pouvez surveiller l'état du déploiement dans la console ECS et obtenir l'adresse IP ou le domaine auquel votre service Strapi est accessible.

Veuillez suivre autant la documentation officielle sur Strapi ainsi que sur le livre de référence pour effectuer cette mise en production

Bibliographie

Elshafie, K. •. (2022). *Designing Web APIs with Strapi*. Packt Publishing.