Mini Projet

Réalisé par : SENECHAL Morgan

Professeur : BOUBCHIR Larbi

Module : ADIF82

Table des matières

Introduction

Apache CouchDB a été développée par Damien Katz, un ancien développeur d'IBM sur Lotus Notes, et est née en 2005 pour répondre aux défis uniques de la gestion de données dans un environnement web en constante évolution. Conçue comme une base de données NoSQL orientée document, CouchDB excelle dans la réplication et la synchronisation des données sur des systèmes distribués, répondant ainsi à un besoin croissant dans un monde de plus en plus interconnecté. Inspirée par les concepts de bases de données distribuées et adaptée aux exigences d'un web moderne, elle se distingue par sa robustesse, sa facilité d'utilisation et son efficacité dans la gestion des données non structurées, même dans des contextes de connectivité intermittente ou peu fiable.

Dans le paysage dynamique des bases de données, Apache CouchDB se positionne comme une solution innovante et adaptée. En stockant des données sous forme de documents JSON (JavaScript Object Notation), CouchDB offre une grande flexibilité et facilité d'utilisation. Sa conception, axée sur la réplication et la synchronisation aisée des données, la rend cruciale dans des scénarios exigeant une cohérence des données en temps réel. Cette singularité, comparée à d'autres systèmes NoSQL, souligne son rôle essentiel dans la gestion de données modernes.

L'une des caractéristiques distinctives de CouchDB est sa capacité à synchroniser des données entre plusieurs instances. Cette fonctionnalité est cruciale pour les applications web et mobiles où la disponibilité et la réplication des données sont essentielles, offrant une expérience utilisateur fluide même avec une connectivité limitée ou intermittente. La réplication dans CouchDB, qui peut être continue ou sur demande, est particulièrement adaptée pour les applications distribuées et les systèmes nécessitant une haute disponibilité et une continuité des opérations.

Conçue avec une architecture RESTful, CouchDB permet une interaction facile via HTTP, facilitant l'accès et la manipulation de données pour les développeurs. CouchDB se distingue également par sa robustesse et sa tolérance aux pannes, garantissant la sécurité et l'intégrité des données même dans des situations difficiles. Sa nature open-source, soutenue par une communauté active, offre une flexibilité supplémentaire, permettant aux développeurs d'adapter et d'étendre la base de données selon leurs besoins spécifiques.

La scalabilité est une autre force de CouchDB, capable de gérer de grandes quantités de données et de maintenir des performances optimales lors de la montée en charge. Des entreprises et projets de renom, tels que BBC, Credit Suisse, et Meebo, ont adopté CouchDB pour sa facilité de gestion de données à grande échelle, témoignant de sa fiabilité dans des environnements exigeants.

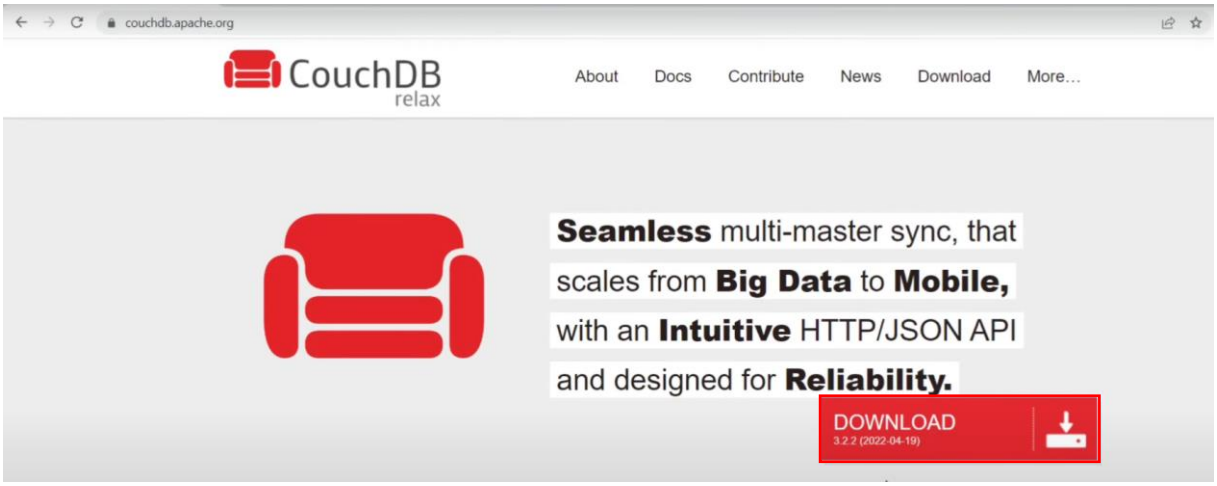
Avec des fonctionnalités comme la réplication incrémentielle, les vues MapReduce pour les requêtes, et une interface web intuitive pour l'administration, CouchDB se positionne comme une option puissante pour le stockage et la gestion de données non structurées. Son interface utilisateur est conçue pour être intuitive, rendant la gestion de la base de données accessible même pour ceux qui ne sont pas des experts en bases de données. De plus, sa capacité à s'intégrer aisément avec d'autres technologies et systèmes la rend particulièrement attractive dans des environnements de développement complexes.

Enfin, CouchDB ne cesse d'évoluer, avec l'introduction de nouvelles fonctionnalités et des améliorations continues. Ces innovations montrent l'engagement de la communauté CouchDB à rester à la pointe des technologies de bases de données, assurant ainsi que CouchDB reste une solution pertinente et puissante pour les défis du stockage de données à l'ère moderne.

Installation Windows

Téléchargé CouchDB

Lien de téléchargement : https://couchdb.apache.org/



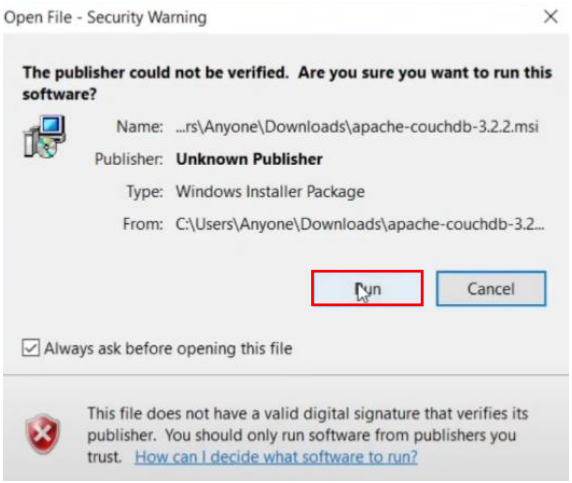
Cliqué sur **DOWNLOAD**

Cliqué sur Download **CouchDB**

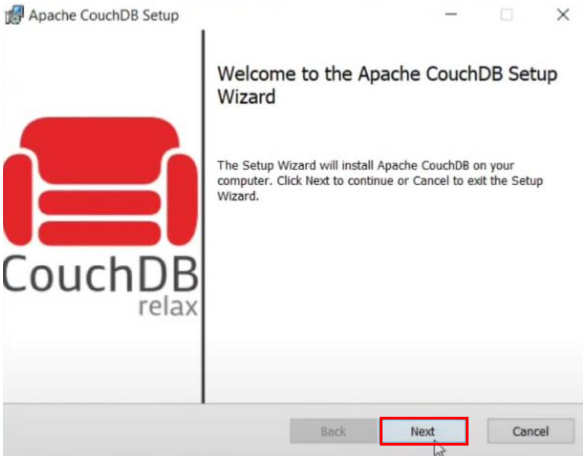
Installateur .MSI

apache-couchdb-3.3.3-2

Lancé l'installateur .msi

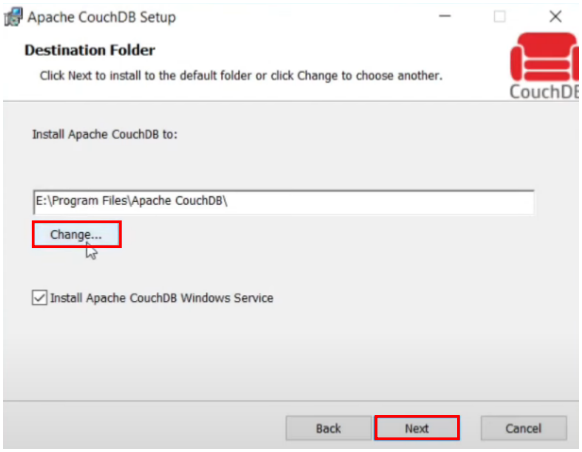


Cliqué sur **Run**

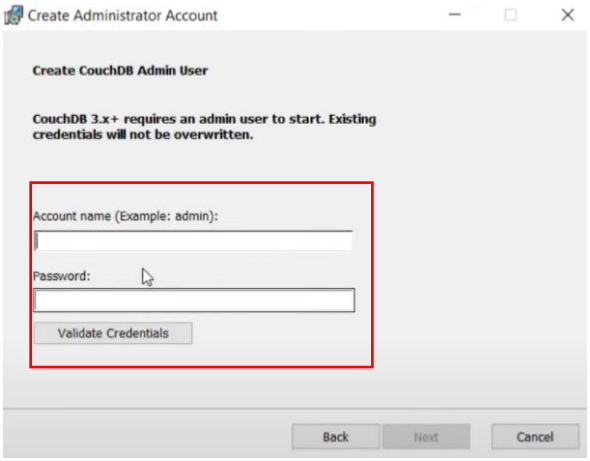


Cliqué sur **Next**

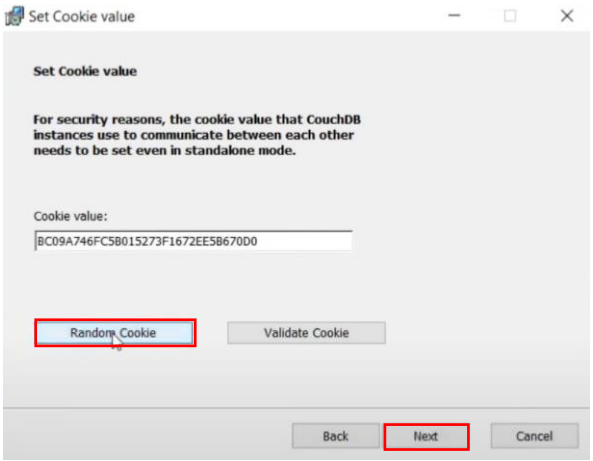
Accepté les **conditions d'utilisation de la licence** et cliqué sur **Next**



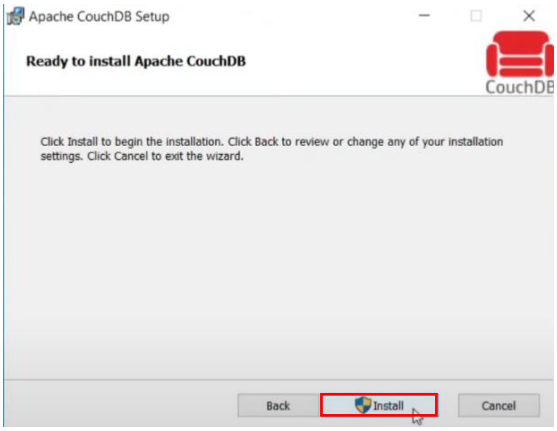
Cliquez sur **Change** pour modifier le chemin d'installation ou laissez le chemin indiqué par défaut. Une fois cela fait, cliquez sur **Next**.



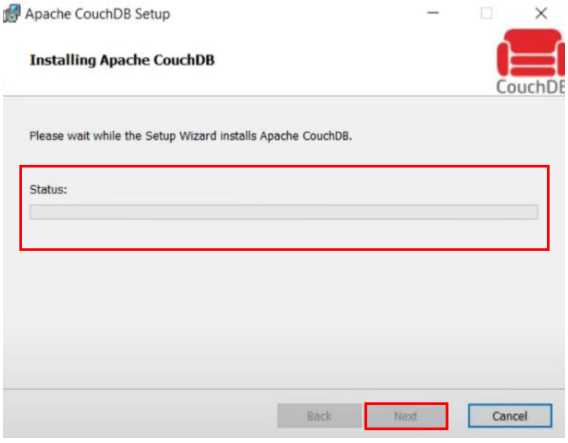
Créez un nom **d'utilisateur admin** et un **mot de passe**. Une fois cela fait, cliquez sur **Validate Credentials**, puis cliquez sur **Next**.



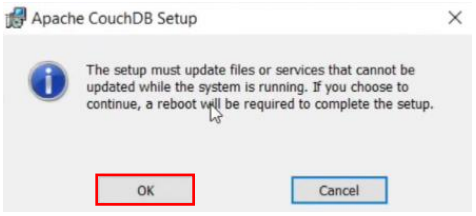
Cliquez sur **Random Cookie**, puis cliquez sur **Next.**



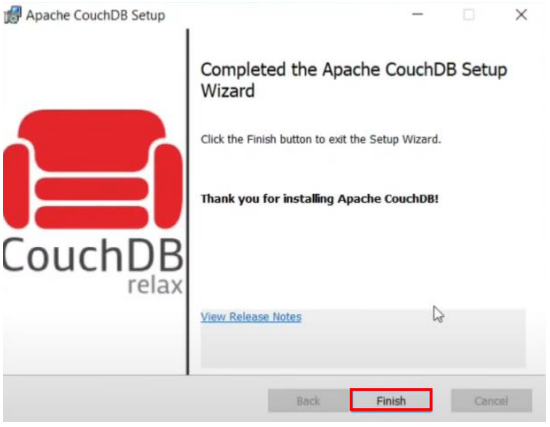
Cliquez sur **Install.**



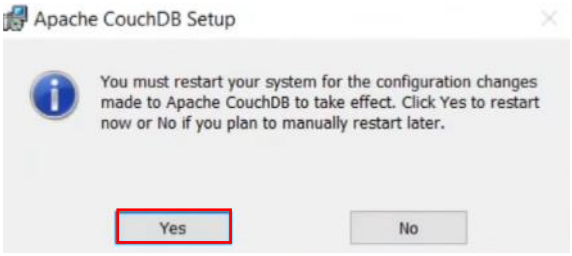
Acceptez les **autorisations de modification**, puis attendez la fin de l'installation.



Cliquez sur **OK.**

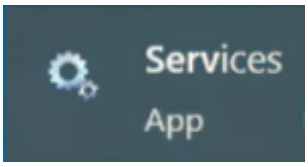


Cliquez sur **Finish.**

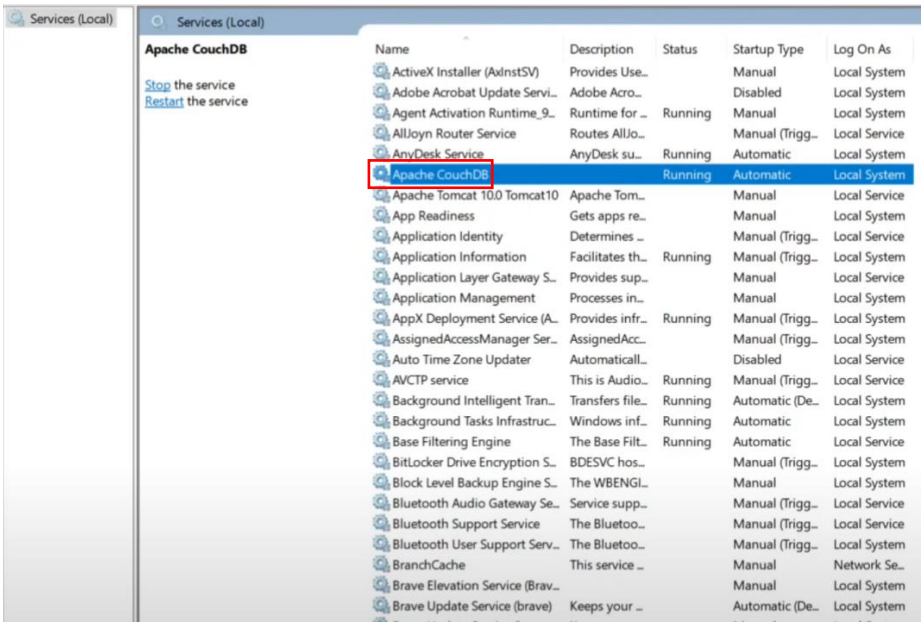


Cliquez sur **Yes**. Cela va redémarrer votre système.

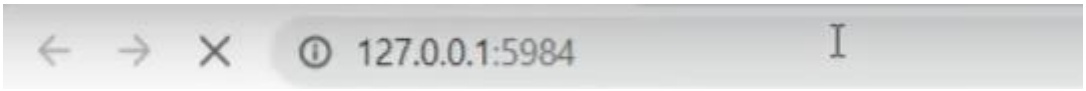
Vérification de l'installation



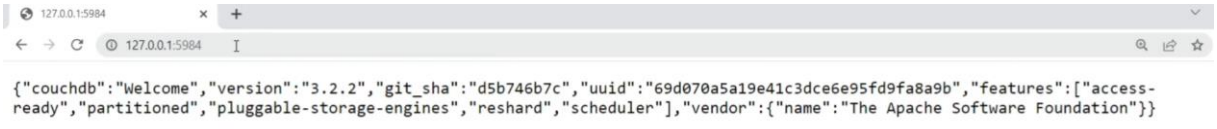
Lancez l'application **Services App**.



Une fois lancé, **Apache CouchDB** devrait apparaître.

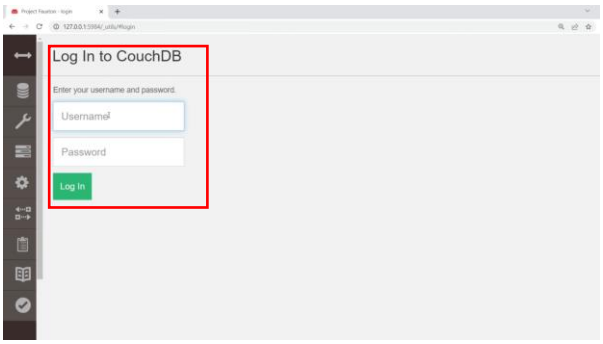


Lancez votre navigateur web, puis tapez votre **adresse localhost** suivie des **numéros par défaut de CouchDB**.

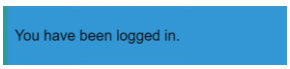


Une fois cette étape réalisée, vous devez voir s'afficher ce message indiquant que **CouchDB est correctement installé**.

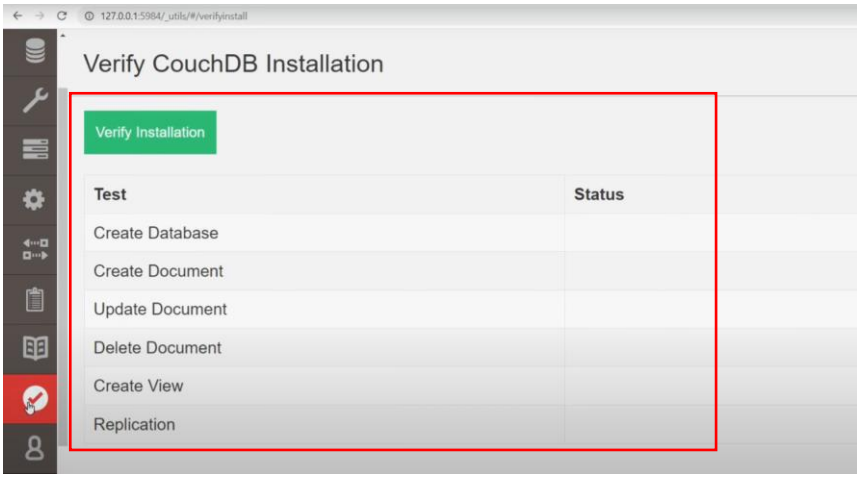
Rendez-vous maintenant sur ce lien.



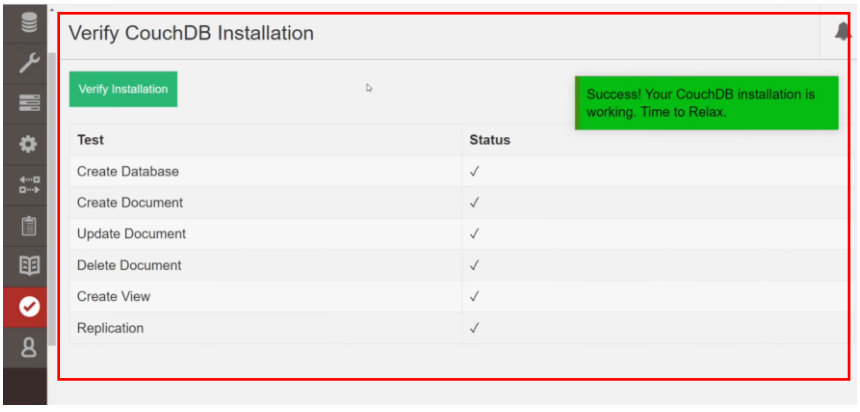
Vous arrivez sur la plateforme d'utilisation de CouchDB. Entrez le **nom d'utilisateur admin** et le **mot de passe** créé lors de l'installation précédente de CouchDB, puis cliquez sur **Log In**.



Une notification devrait apparaître pour vous informer que vous vous êtes bien connecté.



Cliquez sur le bouton V dans la barre d'outils, puis cliquez sur Verify Installation.



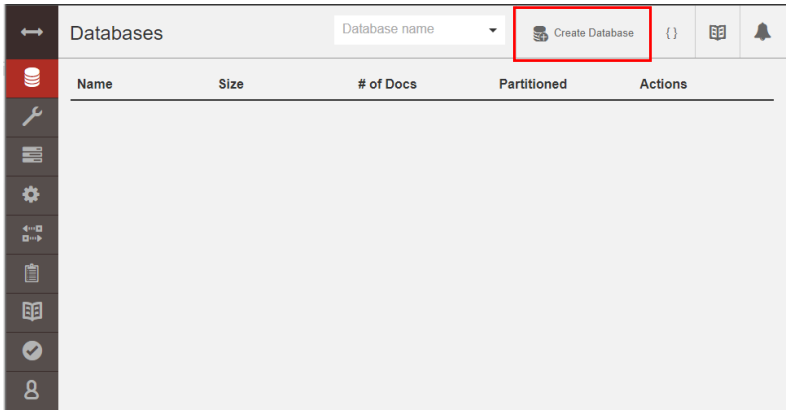
Si l'installation a été réussie, une notification devrait apparaître pour vous indiquer que l'installation de CouchDB s'est réalisée avec succès.

L'installation est terminée, CouchDB est prêt à être utilisé.

Création d'une base de données

Create Database

Créer une base de données sur CouchDB est très simple. Pour cela, il suffit de se rendre dans la rubrique 'Data Base' puis de sélectionner **Create Database**:

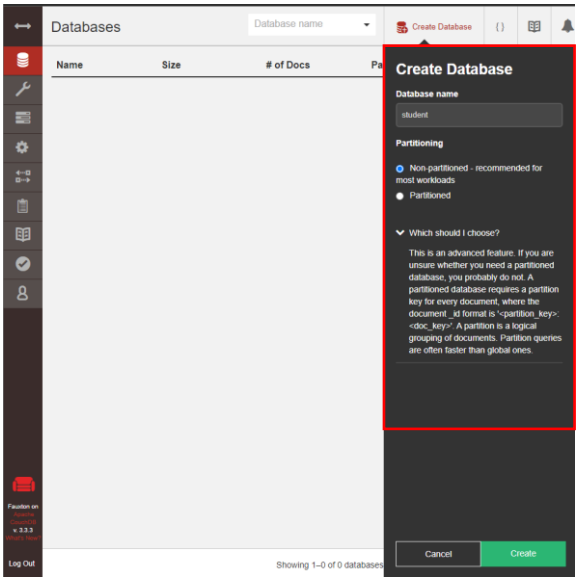


Une fois Create Database sélectionner, nous devons entrer les informations relatives de la base de données que nous voulons crée :

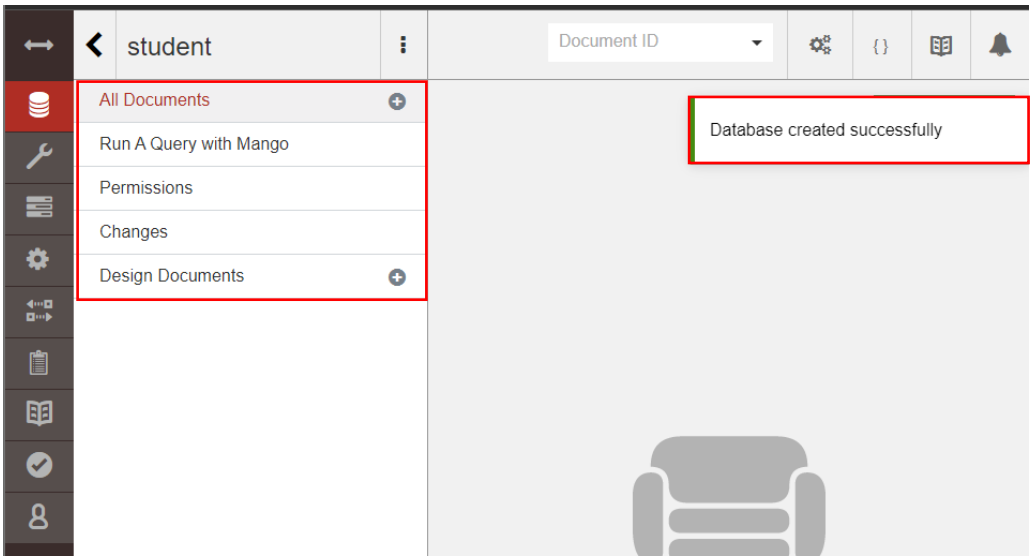
Database name : Le nom que nous voulons donner à notre base de données

Partitioning :

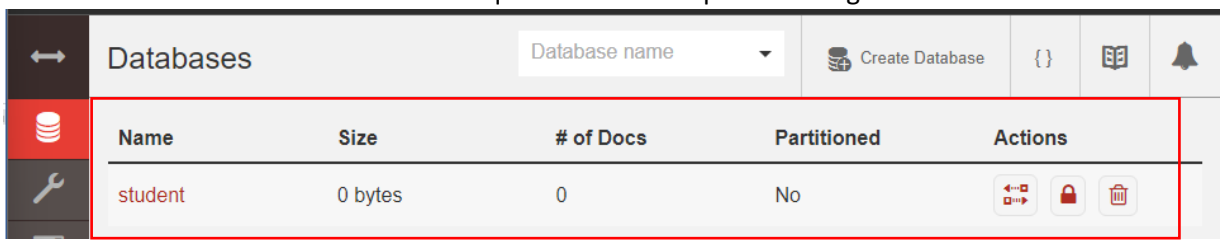
- **Base de Données Non-Partitionnée (Non-Partitioned Database) :** C'est la configuration par défaut dans CouchDB pour les cas d'usage généraux. Idéale pour les charges de travail standard, elle est recommandée s'il n'y pas des exigences spécifiques de performance pour de grandes quantités de données ou pour des requêtes parallèles massives. Elle est plus simple à gérer car elle ne requiert pas de structuration particulière des identifiants de documents. Cependant, pour des bases de données volumineuses, les requêtes peuvent devenir moins performantes, car elles doivent potentiellement examiner tous les documents.
- **Base de Données Partitionnée (Partitioned Database) :** Conçue pour optimiser les performances sur de larges ensembles de données en regroupant des documents selon une clé de partition spécifiée dans l'identifiant du document (e.g., partition_key:doc_key). Cela facilite des requêtes rapides et efficaces au sein d'une partition donnée, rendant ce type de base de données idéal pour des charges de travail parallèles importantes. La mise en œuvre de partitions requiert une conception préalable pour déterminer la meilleure façon de segmenter logiquement les données, ce qui peut complexifier la gestion initiale de la base de données.



Une fois les différents paramètres sélectionnés concernant la création de la base de données, cliquez sur **Create** pour valider les paramètres et créer la base de données. Une fois validée, une notification indiquera que la base de données a bien été créée.



Si nous revenons en arrière en cliquant sur la rubrique 'Data Base', nous pouvons constater la création de notre base de données ainsi que ses différents paramètres généraux :



Voici la correspondance des différentes informations de notre base de données :

Name : C'est le nom de votre base de données. Dans ce cas, elle s'appelle "student".

Size : La taille physique sur le disque de la base de données. Ici, elle est indiquée comme "0 bytes", ce qui signifie qu'aucune donnée n'a encore été stockée ou que la base de données vient d'être créée et ne contient pas encore de documents.

# of Docs : Le nombre de documents contenus dans la base de données. Actuellement, il y a "0" documents, indiquant que la base de données est vide.

Partitioned : Indique si la base de données est partitionnée ou non. "No" signifie que cette base de données n'utilise pas le partitionnement.

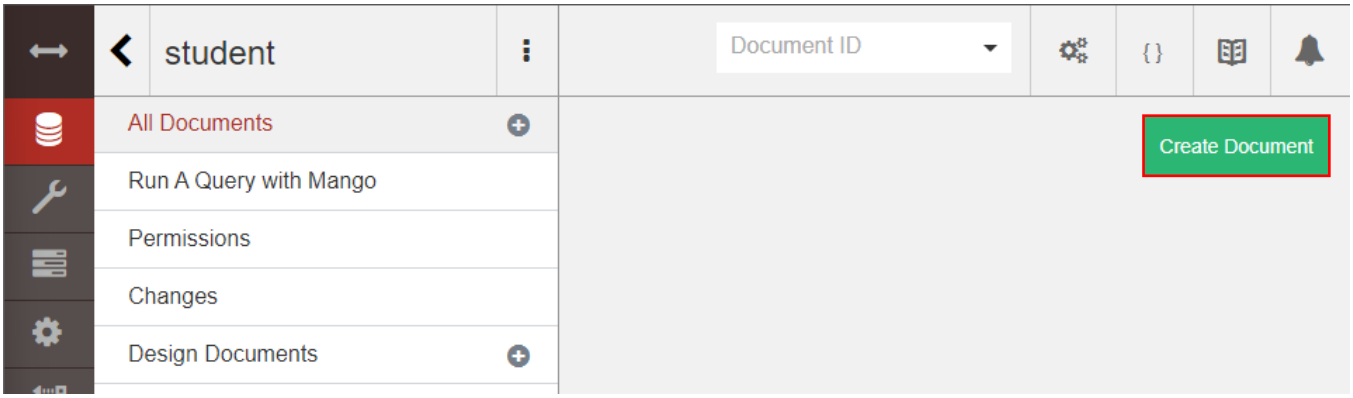
Actions : Il s'agit des actions que vous pouvez effectuer directement depuis cette vue. Les icônes représentent généralement les actions suivantes :

- **L'icône avec les flèches :** Elle représenter une fonction pour "répliquer" la base de données, ce qui permet de copier ou de synchroniser les données avec une autre instance de CouchDB.
- **L'icône de cadenas :** Elle est généralement utilisée pour gérer les "permissions" sur la base de données, pour contrôler qui a le droit de lire ou d'écrire dans la base de données.
- **L'icône de poubelle :** Cela permet de "supprimer" la base de données.

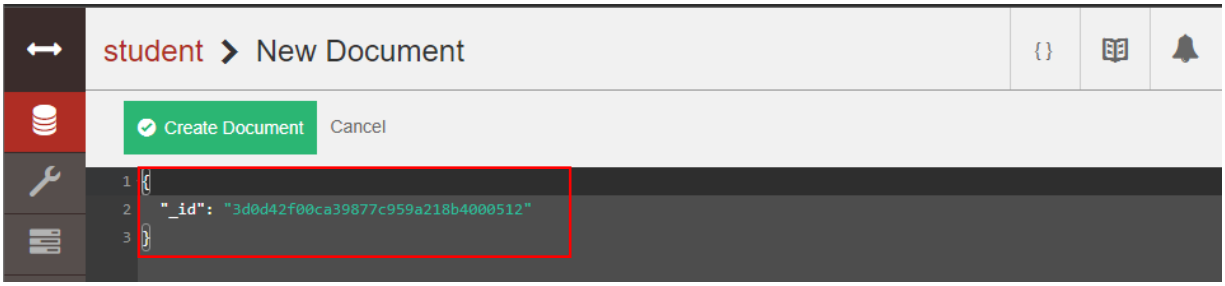
Create Document

All Documents : C'est l'endroit où nous pouvons voir tous les documents stockés dans notre base de données. Chaque document est une unité de données qui peut contenir divers champs JSON. Cette section nous permet de créer de nouveaux documents, de les modifier ou de les supprimer.

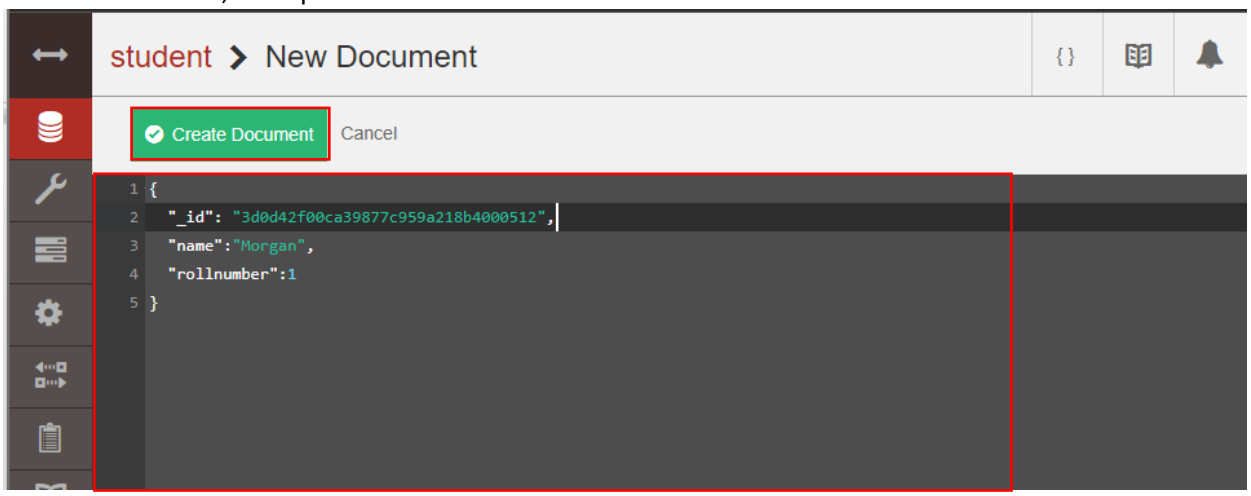
Pour créer un nouveau document, il suffit de cliquer sur **'Create Document'** :



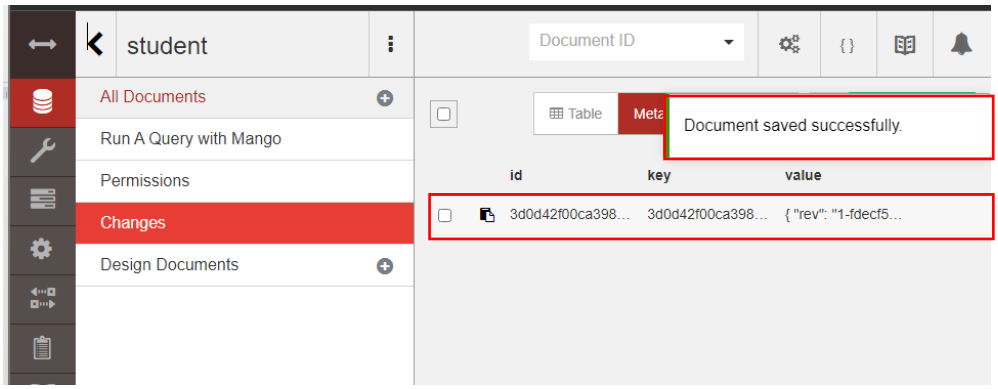
Une fois cela fait, nous arrivons sur l'interface d'édition de notre document au format JSON. Nous y trouvons un « _id » créé automatiquement, qui correspond à l'identifiant unique de chaque document.



Sur cette interface, nous pouvons créer différentes colonnes :

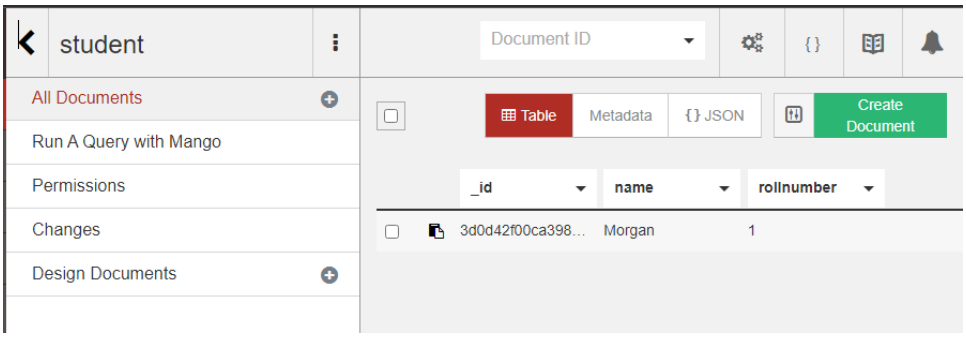


Par exemple, nous avons le champ « name » contenant l'attribut « Morgan », suivi du deuxième champ « rollnumber » avec la valeur 1. Une fois cela fait, il suffit de cliquer sur 'Create Document' pour le créer. Après validation, une notification apparaît pour nous indiquer que le document a bien été sauvegardé.

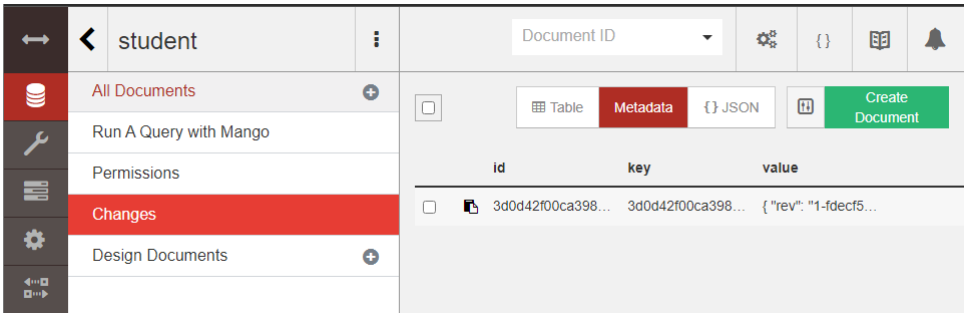


Nous pouvons naviguer dans notre document pour le distinguer sous différentes formes : Table, Métadonnées, JSON :

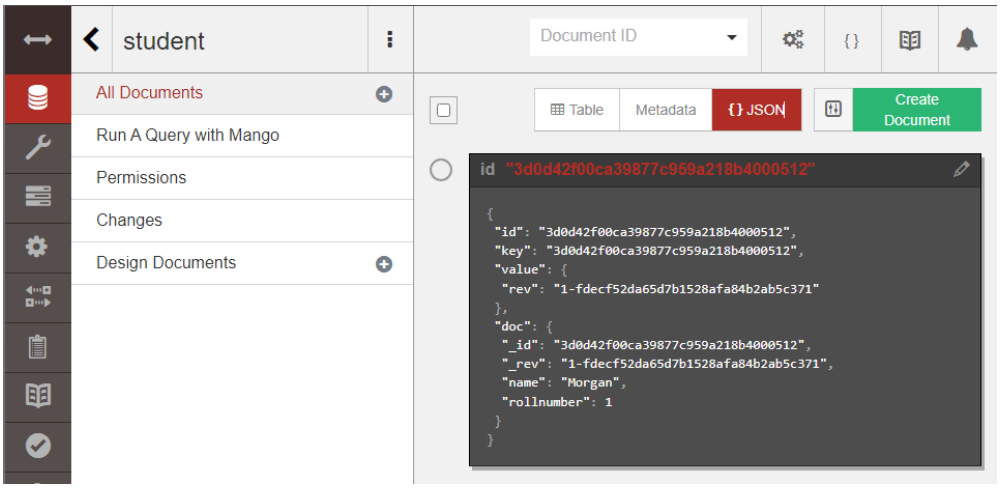
Table :



Metadata :



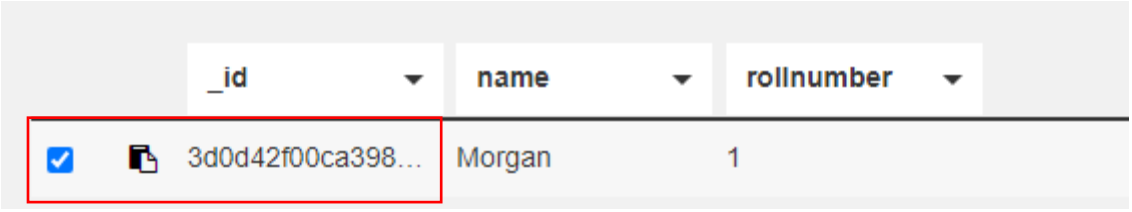
JSON :



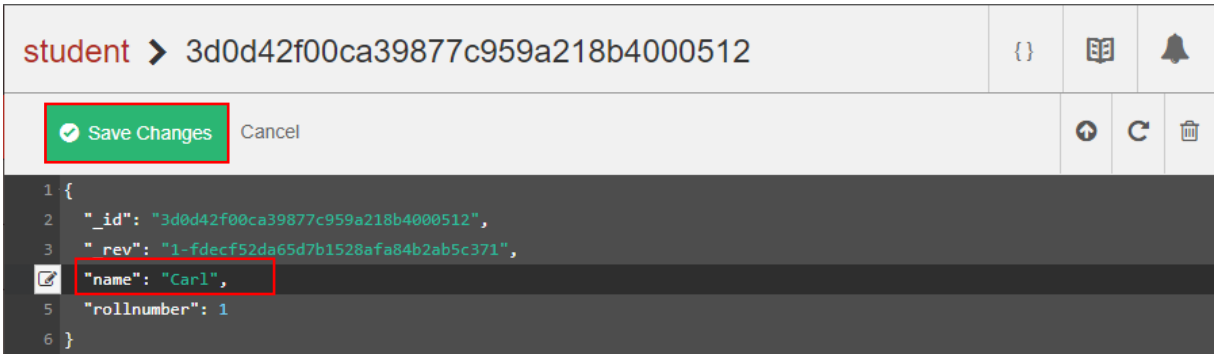
Exploration des fonctionnalités de CouchDB

Update Operation

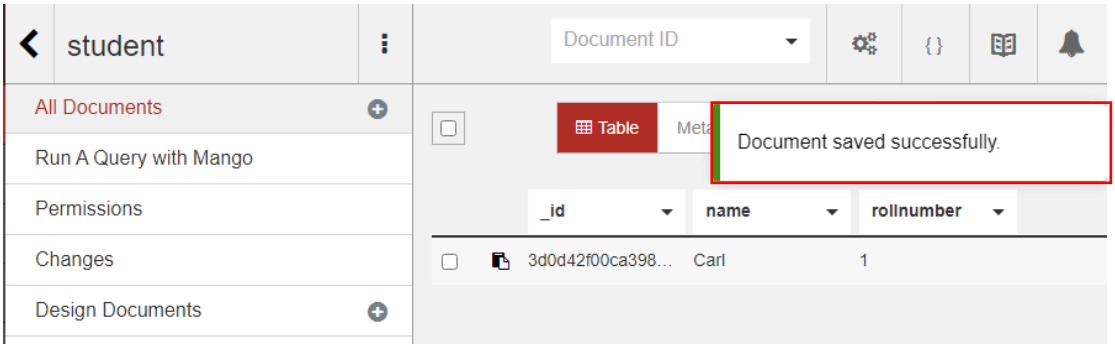
Nous avons la possibilité de modifier le document créé pour y ajouter, supprimer ou modifier les informations insérées. Pour cela, il suffit de sélectionner le document, puis de cliquer sur l'identifiant du document.



Une fois cela fait, nous arrivons sur l'interface d'édition de notre document. Nous pouvons donc ajouter, supprimer ou modifier des informations. Dans notre exemple, nous allons modifier le nom « Morgan » en « Carl », puis cliquer sur « save changes » pour appliquer les modifications apportées.

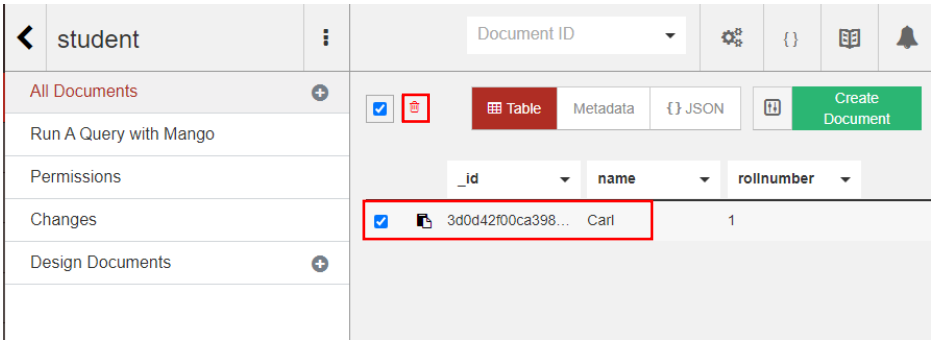


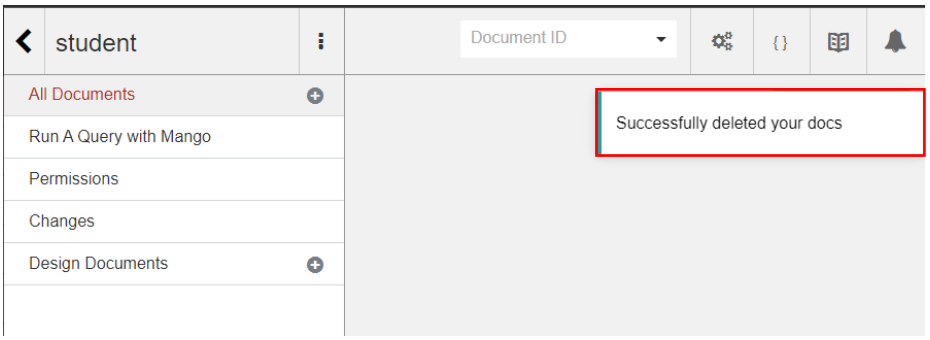
Une fois les modifications apportées, une notification apparaît pour nous indiquer que les changements ont bien été sauvegardés dans le document.



Delete Operation

Maintenant, si nous voulons supprimer notre document, il nous suffit de sélectionner notre document puis de cliquer sur l'icône delete. Une notification nous indiquera que le document a bien été supprimé.

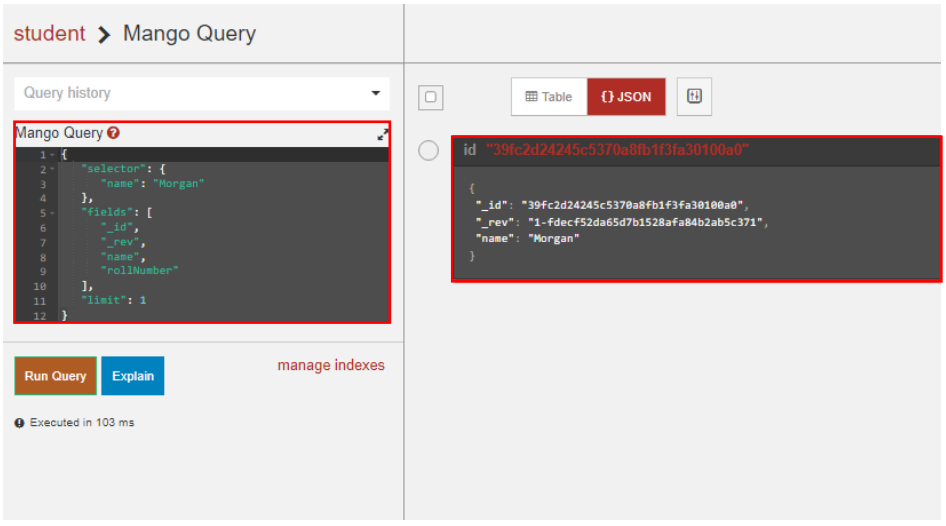




Run A Query with Mango

Mango est le système de requêtes de CouchDB basé sur JSON, qui nous permet de définir des requêtes pour filtrer et trier les documents dans notre base de données. Cela nous facilite la recherche de documents selon des critères spécifiques sans nécessiter l'écriture de fonctions de vue complexes.

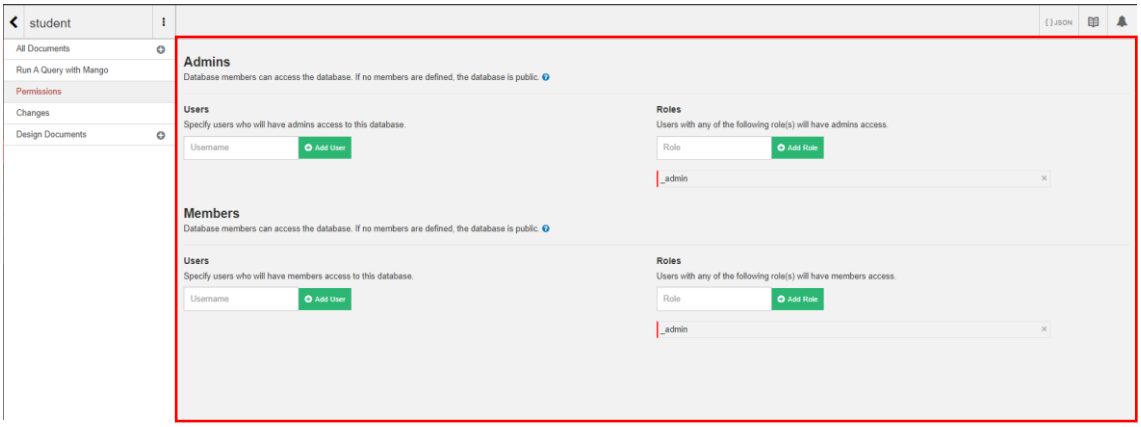
Voici un exemple :



Cette requête demande à CouchDB de sélectionner le document dont le champ "name" est égal à "Morgan". Elle spécifie également de retourner uniquement les champs "_id", "_rev", "name", et "rollNumber", et limite le résultat à un seul document.

Permissions

C'est ici que nous pouvons gérer les autorisations de la base de données, en définissant qui peut lire ou écrire dans la base. Cela constitue un élément crucial pour la sécurité et le contrôle d'accès.



Admins :

- **Database Admins :** Peuvent créer, modifier et supprimer des documents de base de données, y compris le document de design et le document de sécurité. Ils ont également le droit de répliquer la base de données.
- **Server Admins :** Ont tous les droits sur tous les aspects du serveur CouchDB, y compris la création de nouvelles bases de données et la visualisation de toutes les bases de données du serveur.

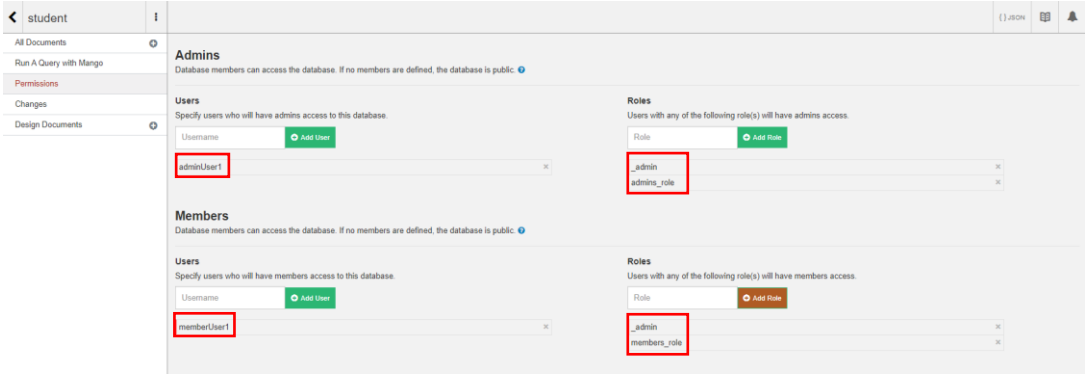
Members :

- **Database Members :** Peuvent lire et écrire les documents de la base de données, à l'exception du document de design et du document de sécurité.

Les permissions de lecture et d'écriture ne s'appliquent pas aux documents de design, qui sont contrôlés par les permissions d'admin.

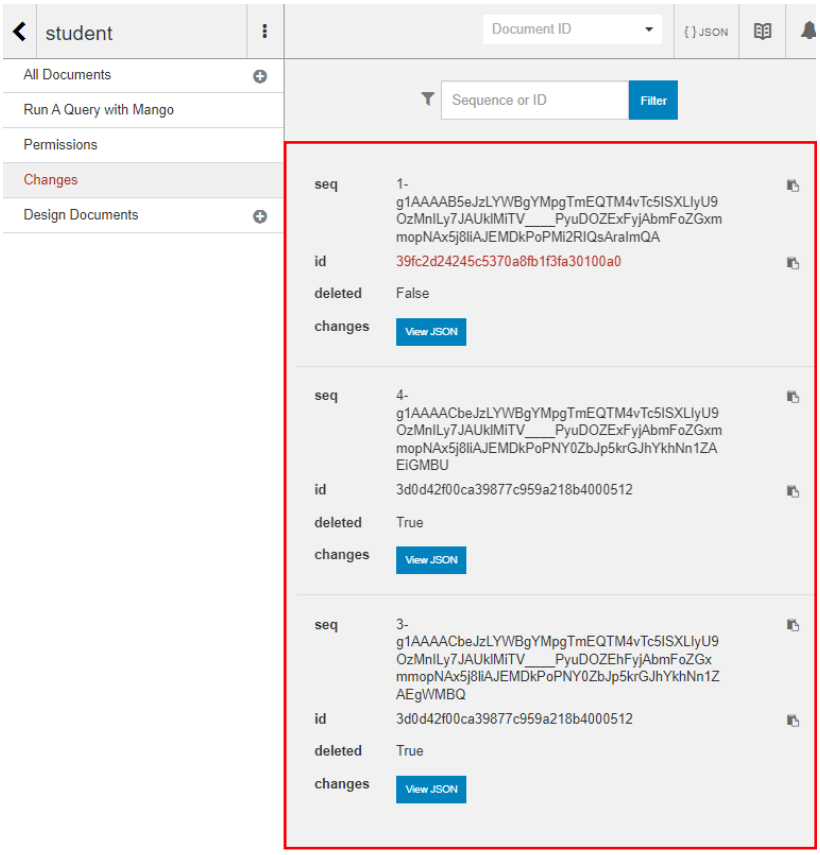
Les permissions dans CouchDB peuvent également être définies à l'aide de rôles, qui sont des étiquettes que vous pouvez attribuer aux utilisateurs et qui peuvent ensuite être vérifiées dans les fonctions de validation des mises à jour pour autoriser différentes actions.

Voici un exemple :



Changes

Cette section offre un aperçu de tous les changements survenus dans la base de données. Chaque ajout, modification ou suppression de document est enregistré et visible ici.



seq : C'est le numéro de séquence de la modification. CouchDB utilise une séquence monotone croissante pour identifier chaque modification. Cela peut être utilisé pour suivre l'ordre des changements et pour synchroniser de manière incrémentale avec la base de données.

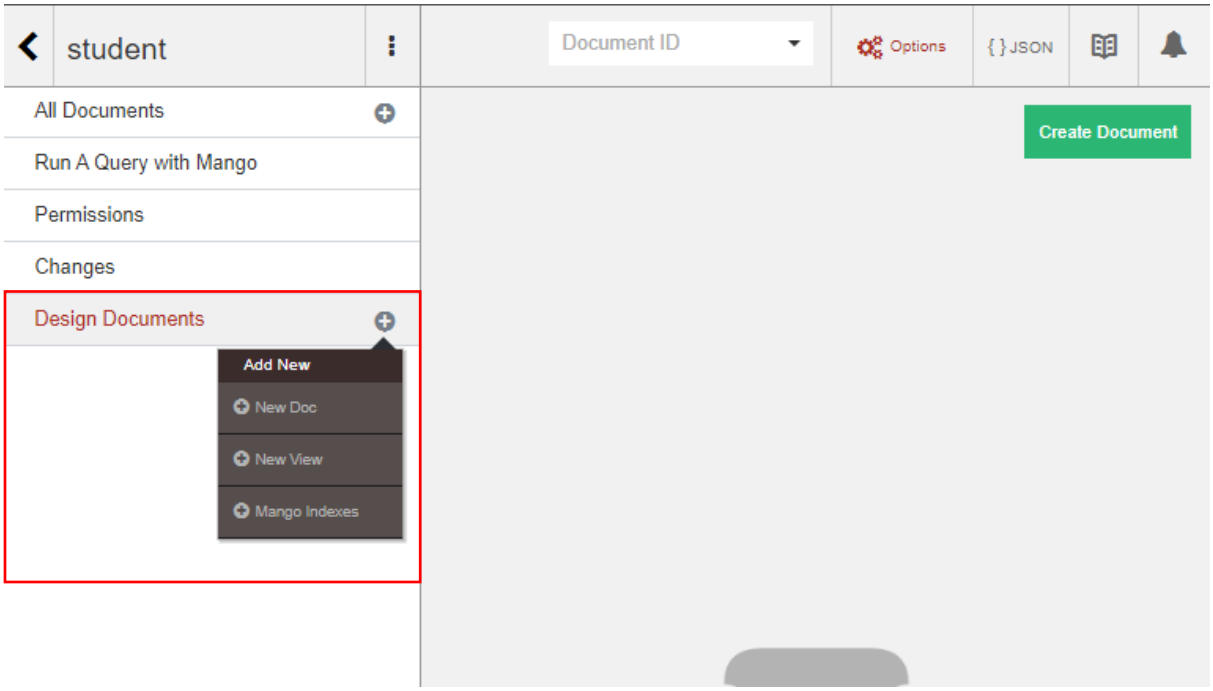
id : L'identifiant unique du document qui a été modifié.

deleted : Un booléen qui indique si le document a été supprimé. True signifie que le document a été supprimé, False signifie qu'il n'a pas été supprimé.

changes : Il s'agit d'une liste des changements de révision du document. Chaque entrée contient une "révision" (rev) qui est un identifiant unique pour la version du document après la modification. Vous pouvez cliquer sur "View JSON" pour voir les détails de la révision.

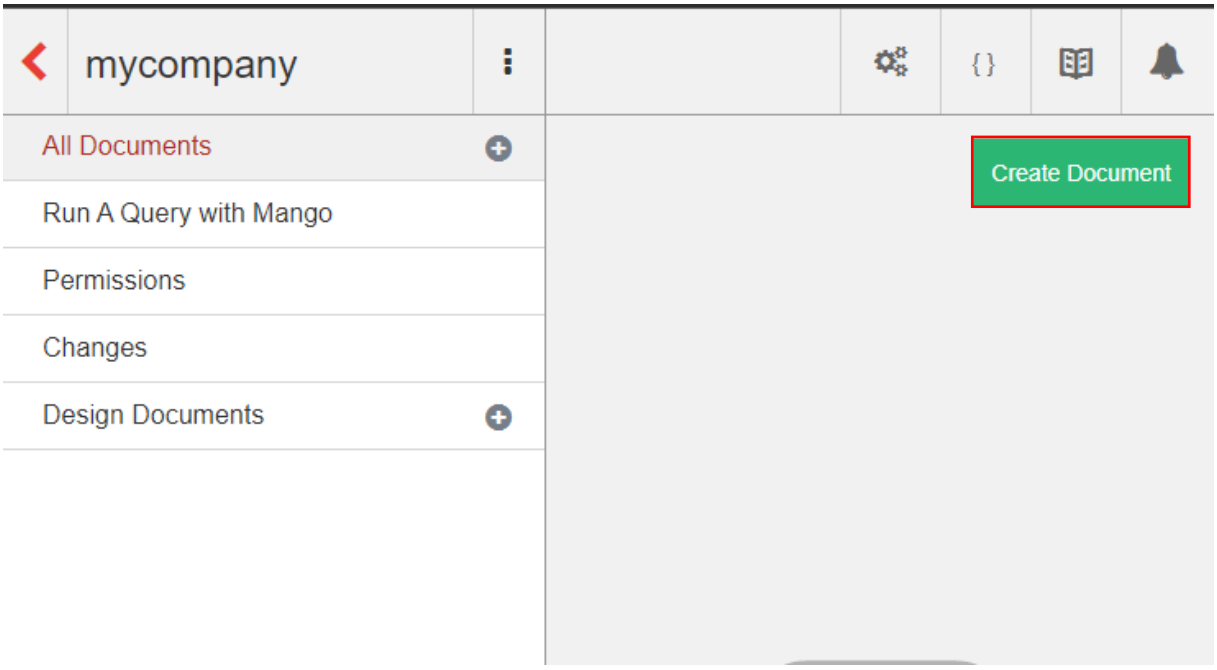
Design Documents

Les documents de conception sont des éléments spéciaux au sein de la base de données qui incluent des définitions de vues, des fonctions de validation de documents, des mécanismes d'indexation, entre autres, lesquels déterminent le comportement global de la base. En particulier, les vues sont utilisées pour formuler des requêtes complexes et créer des index destinés à optimiser la performance des interrogations.

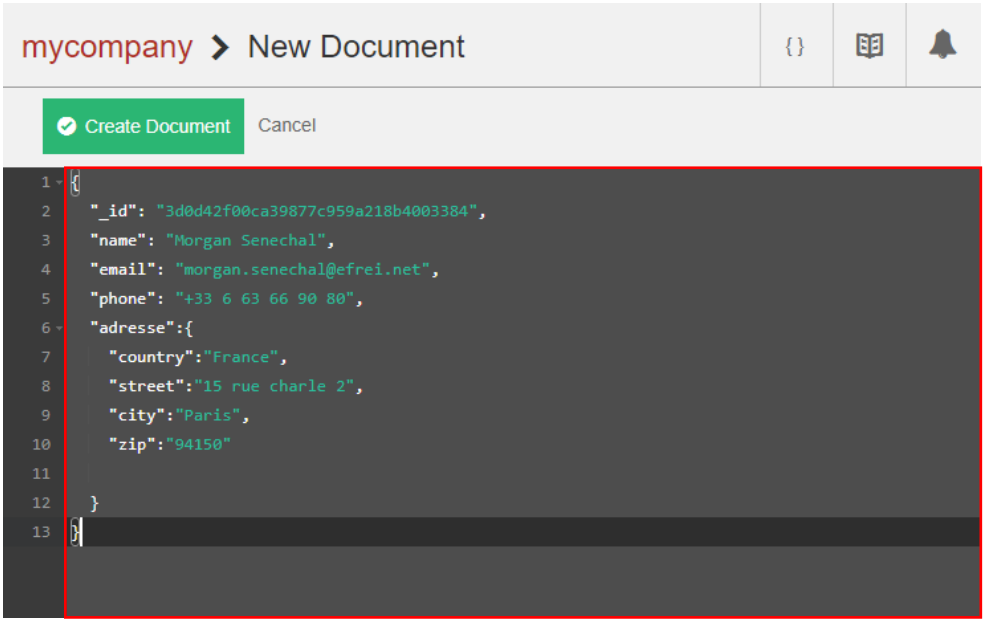


New doc

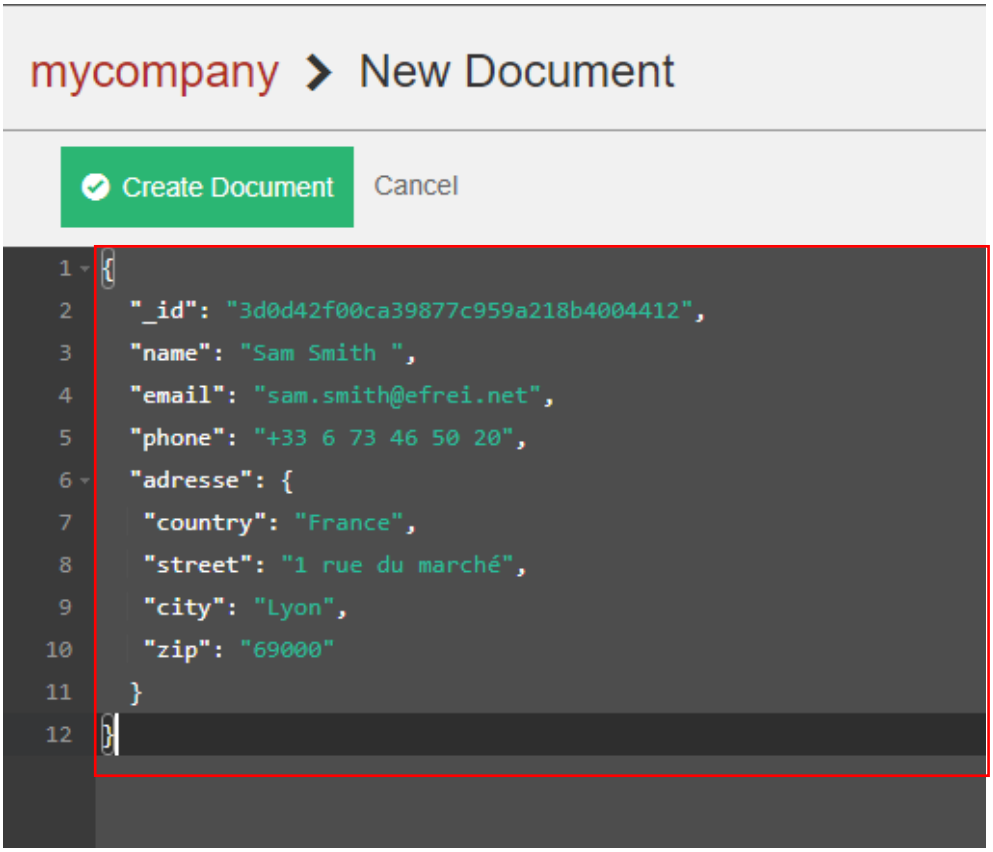
Création d'une nouvelle base de données : mycompany.



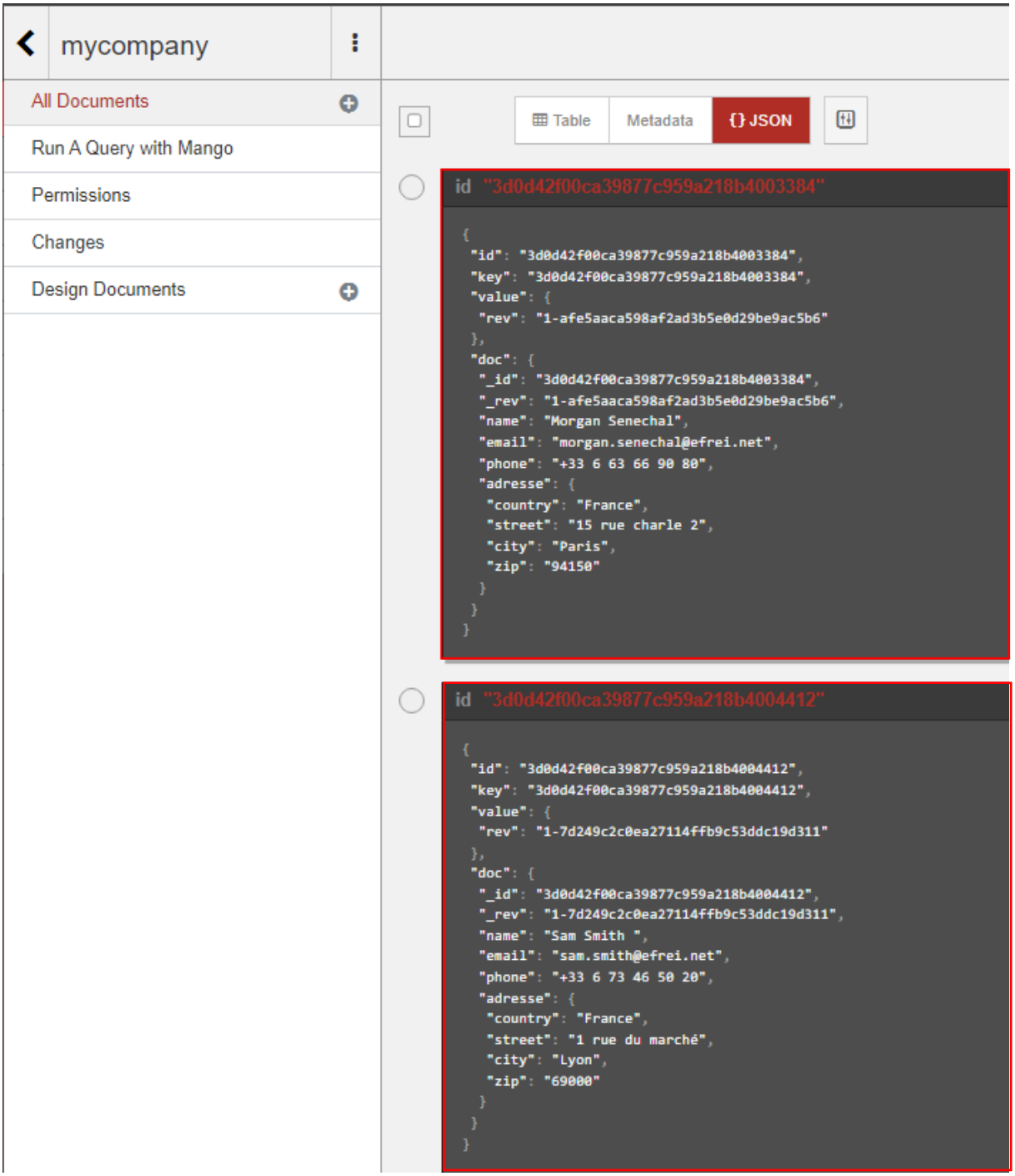
Création d'un nouveau document dans la base de données "mycompany" :



Création d'un deuxième document similaire dans la base de données "mycompany" :



Nous pouvons maintenant voir que nos deux documents ont été créés :



New View

Dans CouchDB, une "View" (vue) est essentiellement une requête sauvegardée qui vous permet de filtrer et d'afficher les documents de la base de données d'une manière spécifique. Les vues sont un composant des "Design Documents" et utilisent des fonctions MapReduce pour produire des résultats.

Voici les composants clés d'une vue tels que montrés dans votre image :

Design Document : Un design document est un document spécial qui contient les vues et d'autres éléments tels que des fonctions de validation.

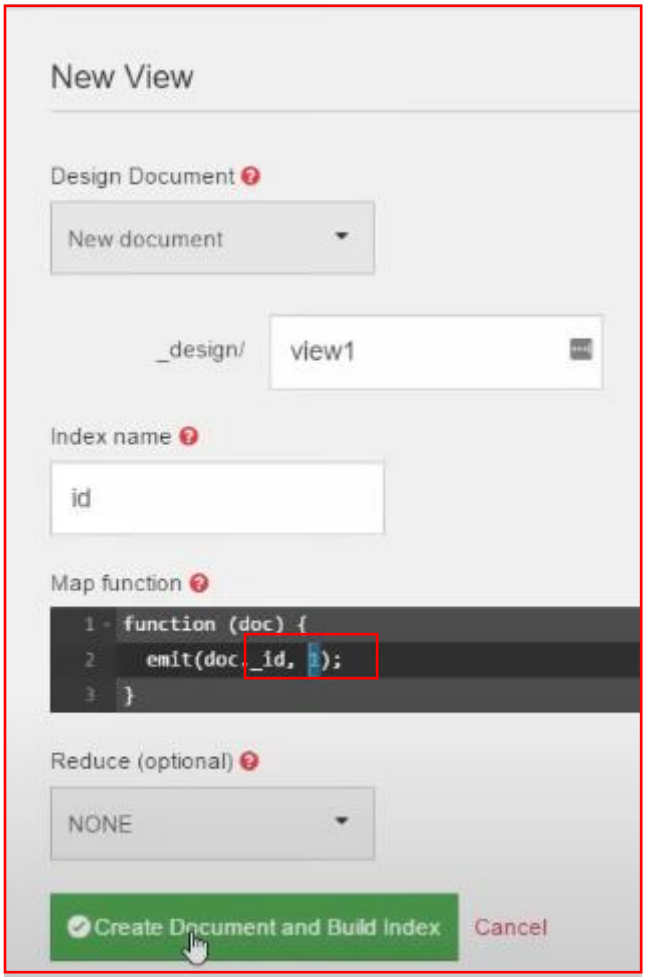
Index name: Le nom de l'index créé par la vue.

Map function: La fonction de map est une fonction JavaScript qui est exécutée sur chaque document de la base de données. Elle détermine quelles données apparaissent dans la vue. Dans l'exemple, la fonction emit(doc._id, 1); émet l'ID de chaque document avec une valeur de 1. C'est un modèle simple pour énumérer tous les documents.

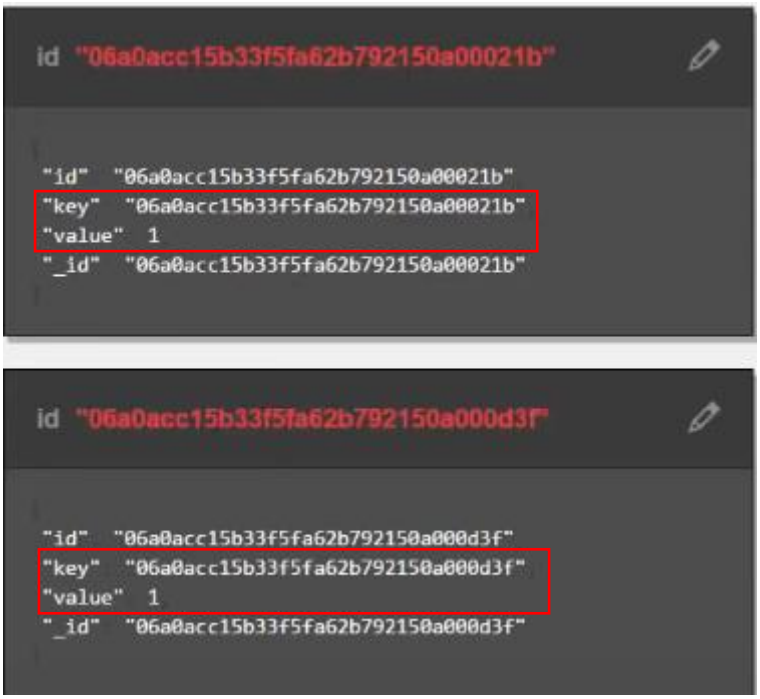
Reduce function (optional): Une fonction reduce est utilisée pour agréger ou résumer les résultats de la fonction de map. Par exemple, elle pourrait compter le nombre de documents qui correspondent à un certain critère.

Dans notre cas, nous allons créer 3 view différentes :

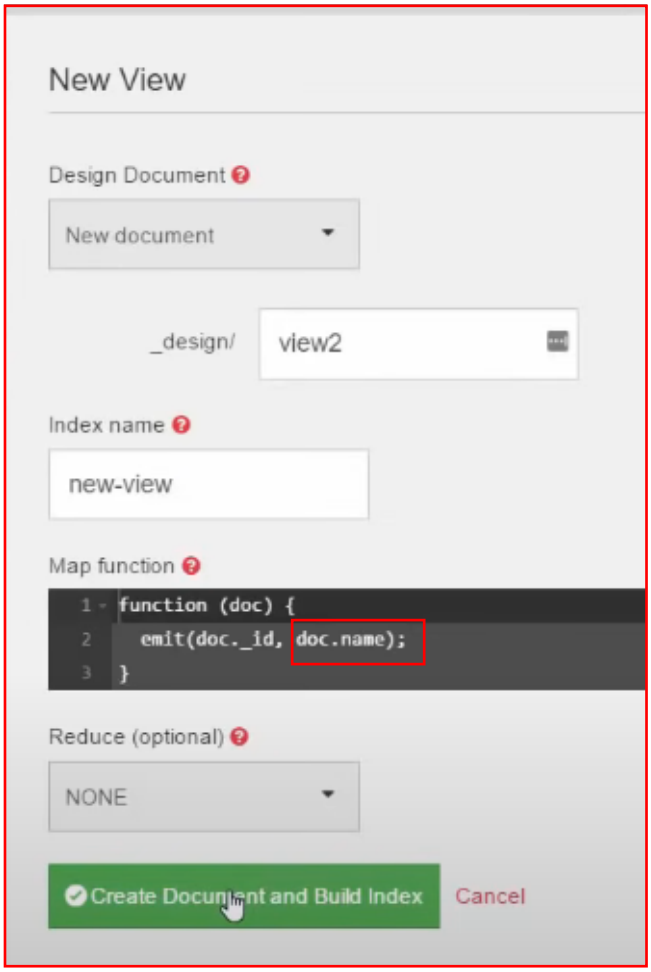
View 1 :



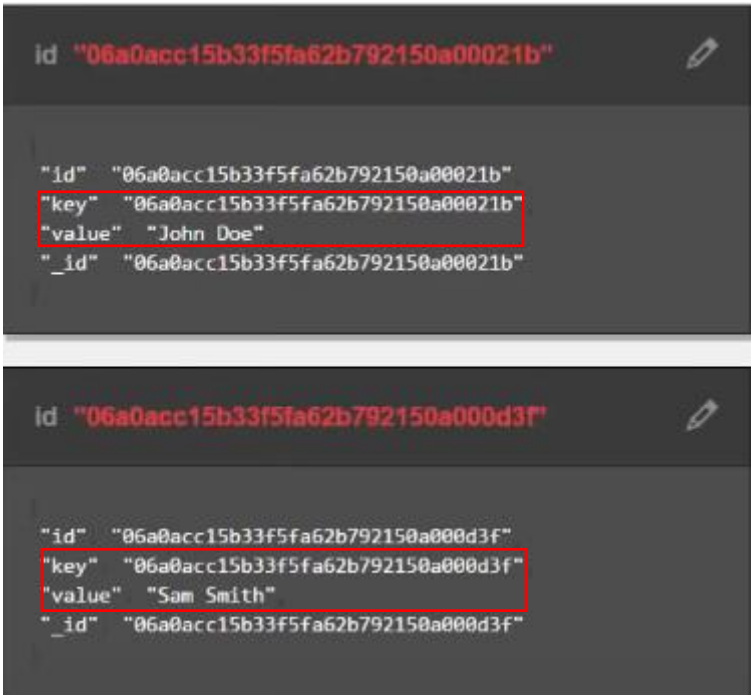
Nous pouvons voir que dans cette View 1, la paire Key, Value est bien présente. Dans cette vue, nous avons défini la valeur (Value) par l'entier 1.



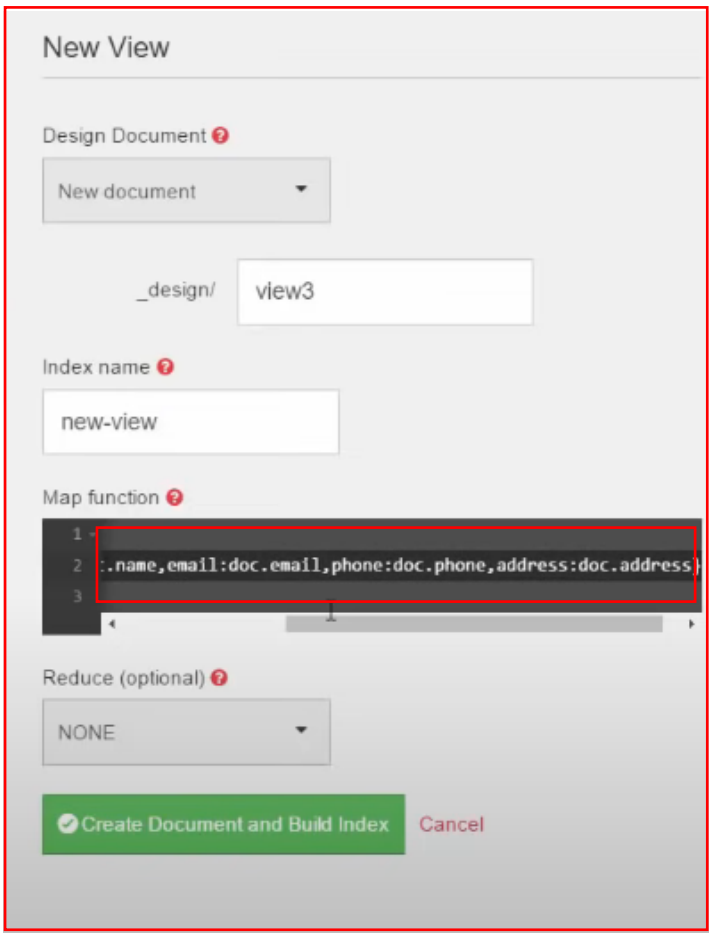
View 2 :



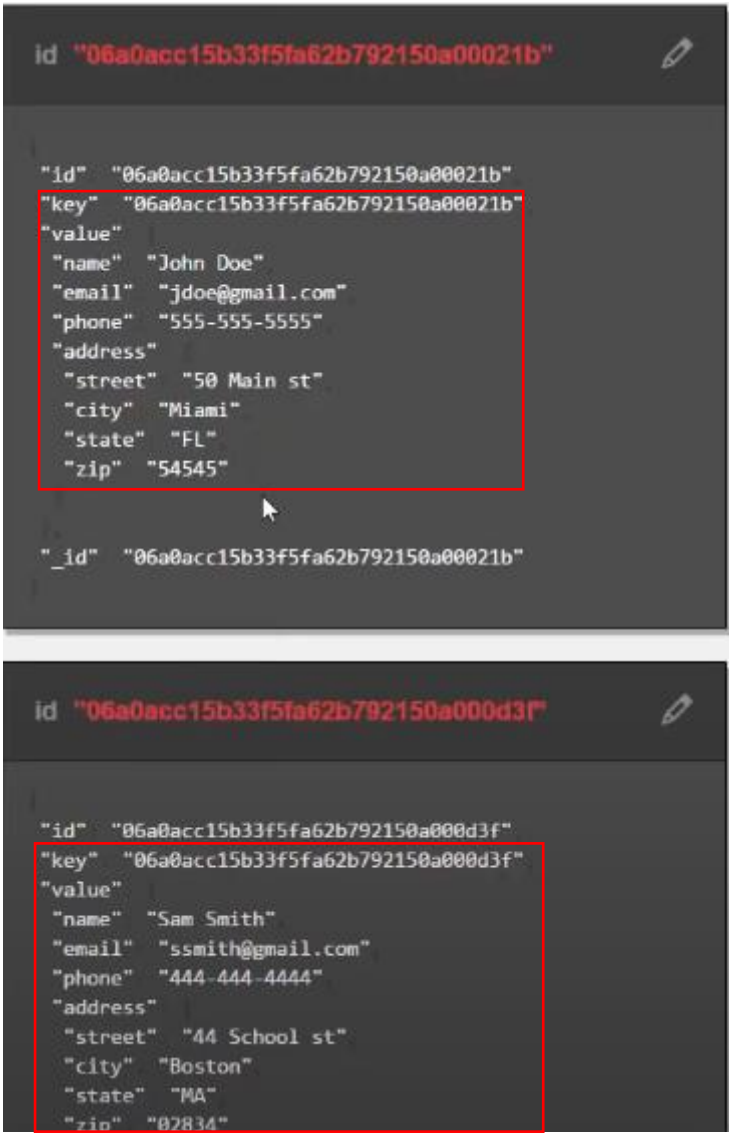
Nous pouvons maintenant voir la valeur de notre vue avec les paires Key, Value. Ici, nous avons défini la valeur (Value) comme étant le nom (name).



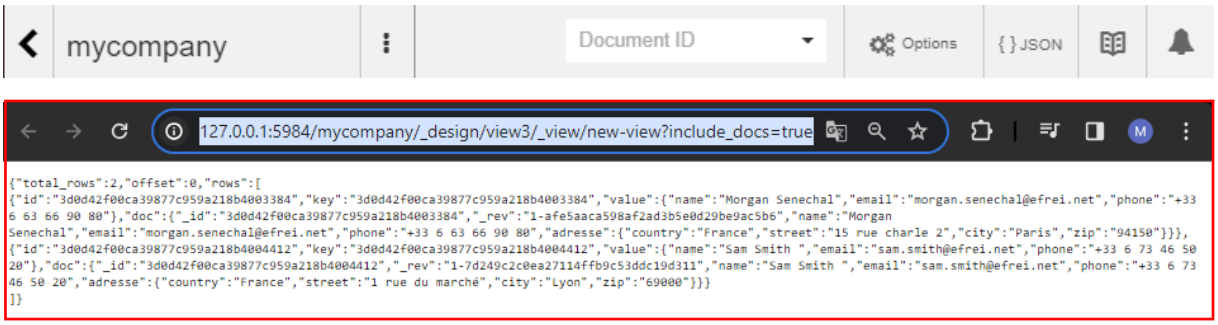
View 3 :



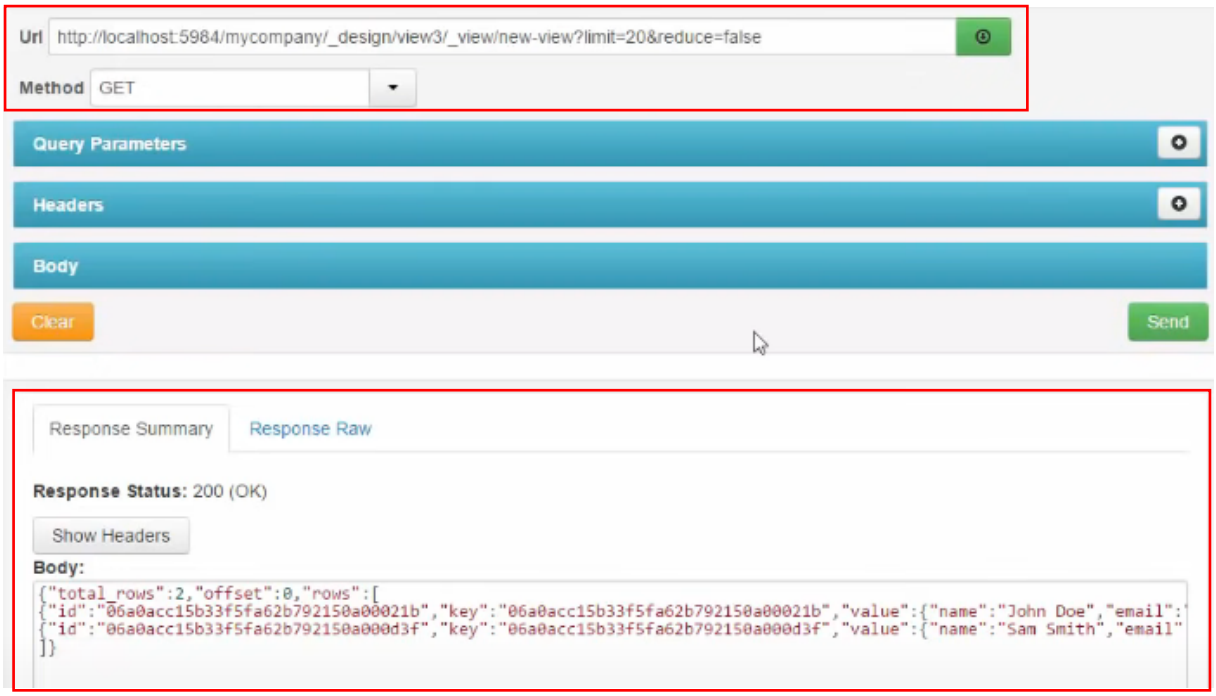
Nous pouvons maintenant voir la valeur de notre View 3 avec les paires Key, Value. Ici, nous avons défini la valeur (Value) comme un objet :



Maintenant, nous pouvons interroger nos vues depuis une URL API. Pour cela, il suffit de cliquer sur {} JSON, ce qui nous redirigera vers l'URL API où notre vue est contenue :



Nous pouvons tester cette API avec un outil REST tel que RESTEasy :

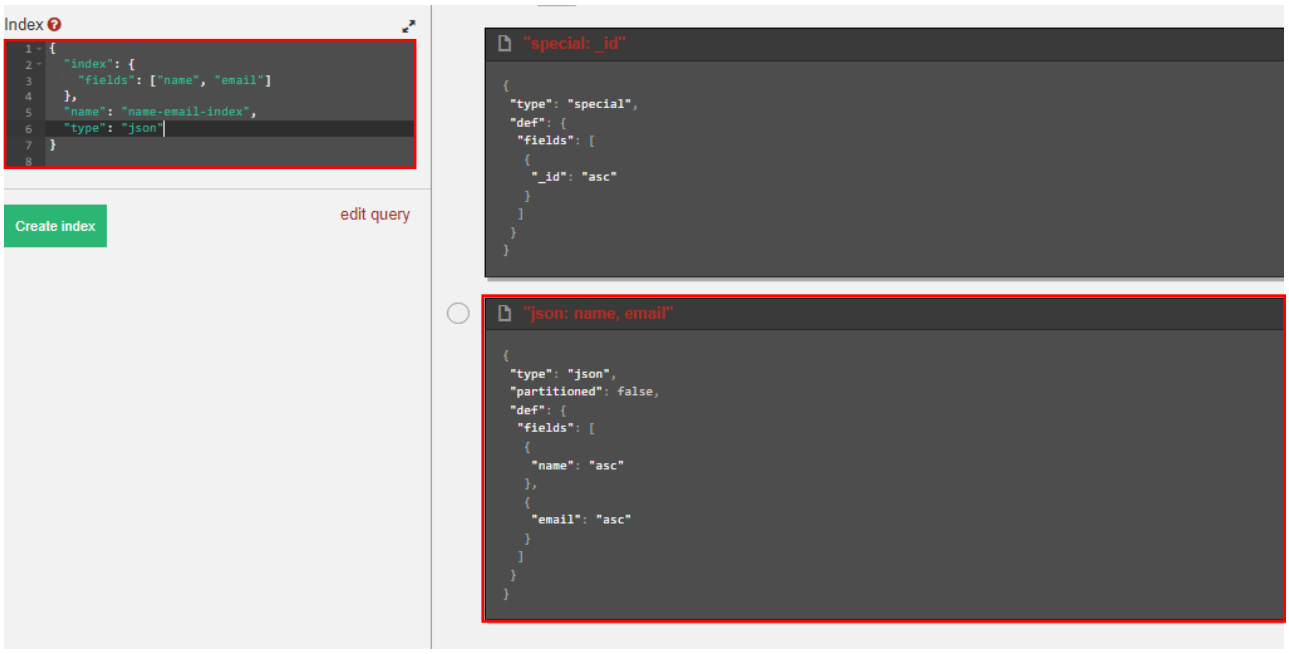


Nous pouvons constater que l'API fonctionne, et que la vue contenant nos données a bien été obtenue grâce à la méthode GET.

Mongo Indexes

Un index Mango dans CouchDB est utilisé pour améliorer la performance des requêtes en définissant des chemins d'accès aux documents qui seront souvent recherchés.

Exemple :



Cet index créera un index sur les champs "name" et "email", ce qui permettra des recherches rapides et efficaces lorsque vous interrogez la base de données pour ces champs.

Index

```
{
  "index": {
    "fields": ["adresse.city", "adresse.zip"]
  },
  "name": "city-zip-index",
  "type": "json"
}
```

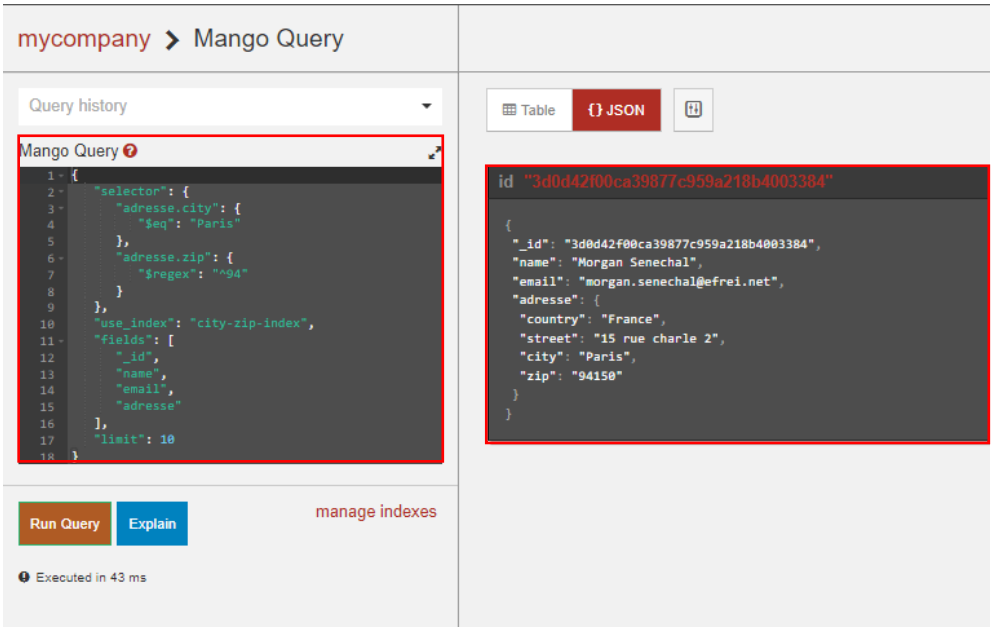
"json: adresse.city, adresse.zip"

```
{
 "type": "json",
 "partitioned": false,
 "def": {
  "fields": [
   {
    "adresse.city": "asc"
   },
   {
    "adresse.zip": "asc"
   }
  ]
 }
}
```

Cet index permet d'effectuer des requêtes ciblées sur les documents où la ville et le code postal correspondent à certains critères. Par exemple, si nous souhaitons trouver tous les utilisateurs situés à Paris avec un code postal spécifique, nous pouvons utiliser une requête exploitant cet index pour accélérer la recherche.

Testons cette index :

Recherche tous les documents où la ville est "Paris" et le code postal commence par "94"



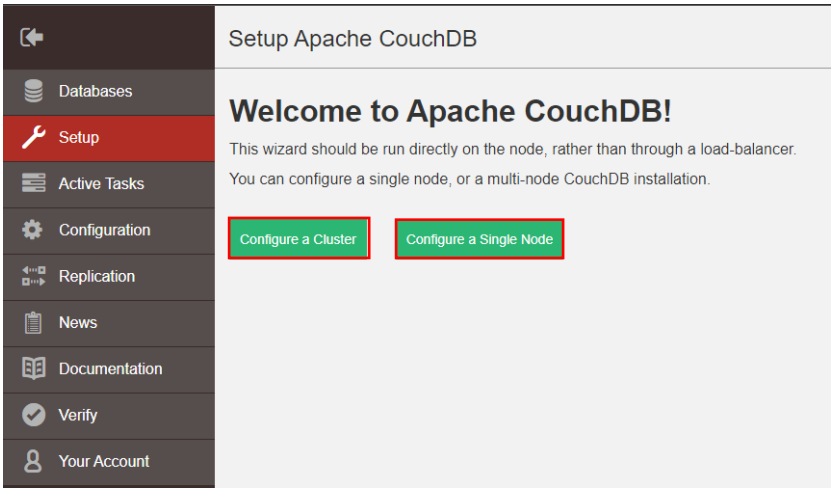
"use_index": "city-zip-index" indique à CouchDB d'utiliser l'index spécifique que nous avons créé précédemment pour cette requête.

Setup

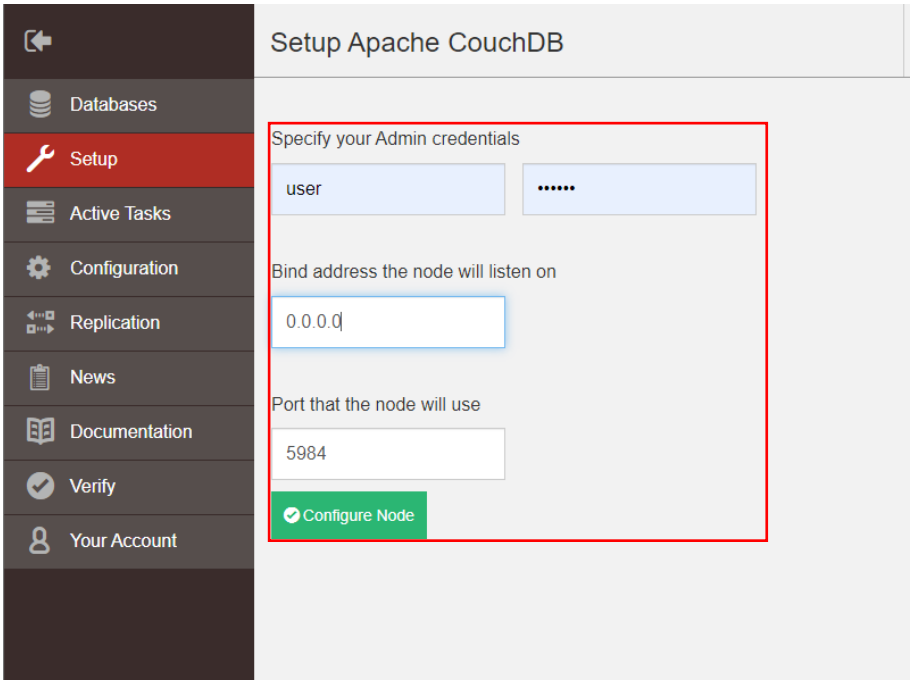
Voici les différentes configurations auquel nous pouvons opter :

Configure a Cluster: Cette option est utilisée pour configurer plusieurs instances de CouchDB pour qu'elles fonctionnent ensemble comme un seul cluster. Dans un cluster, les données peuvent être répliquées et équilibrées entre différents nœuds (machines ou instances), ce qui offre une meilleure résilience et une meilleure montée en charge. Si une instance échoue, les autres peuvent continuer à servir les données. Cela est particulièrement utile pour les applications qui nécessitent une haute disponibilité et une capacité à gérer de grands volumes de données ou de trafic.

Configure a Single Node: Cette option est destinée à la configuration d'une seule instance de CouchDB qui fonctionnera de manière autonome. C'est le choix approprié pour les scénarios où une seule base de données est suffisante, comme les environnements de développement, les petites applications ou lorsque la haute disponibilité n'est pas une exigence critique.



Dans cet exemple, nous allons faire le choix d'une configuration : nœud unique.



Voici les différents paramètres que nous devons saisir :

Specify your Admin credentials: Ici, nous devons entrer les identifiants pour le compte administrateur de CouchDB

Bind address the node will listen on: Cela définit l'adresse IP sur laquelle CouchDB acceptera les connexions entrantes. L'adresse 0.0.0.0 signifie que CouchDB écoutera sur toutes les interfaces réseau disponibles de la machine. Cela inclut l'accès local et l'accès depuis d'autres machines sur le réseau, selon la configuration de votre pare-feu et de votre réseau.

Port that the node will use: Ce paramètre spécifie le port TCP sur lequel CouchDB écoutera les connexions entrantes. Le port par défaut pour CouchDB est 5984.

Une fois les différents paramètres entrés, il suffit de sélectionner Configure Node pour crée notre configuration. Une notification apparaitra pour nous indiquer que notre Setup à bien été enregistré.

Active Tasks

Active Tasks sert à afficher les tâches actuellement en cours d'exécution dans l'instance de CouchDB.

Voici les différents tasks que nous pouvons retrouver :

All Tasks: Cette option affiche toutes les tâches actives en cours sur le serveur, quelle que soit leur nature. Cela inclut la réplication, l'indexation, la compaction, et d'autres tâches de maintenance ou opérationnelles.

Replication: Lorsque nous sélectionnons cette option, nous ne verrons que les tâches de réplication.

Database Compaction : Cette option filtre et affiche les tâches de compaction de la base de données. La compaction est le processus par lequel CouchDB nettoie les anciennes révisions de documents qui ne sont plus nécessaires et qui occupent de l'espace disque. Ce processus est important pour maintenir de bonnes performances et optimiser l'utilisation de l'espace de stockage.

Indexer : Ici, nous verrons les tâches liées à l'indexation des documents. CouchDB crée des index pour accélérer les requêtes. Quand de nouveaux documents sont ajoutés ou modifiés, ces index doivent être mis à jour, ce qui est une opération qui peut être suivie dans cette catégorie.

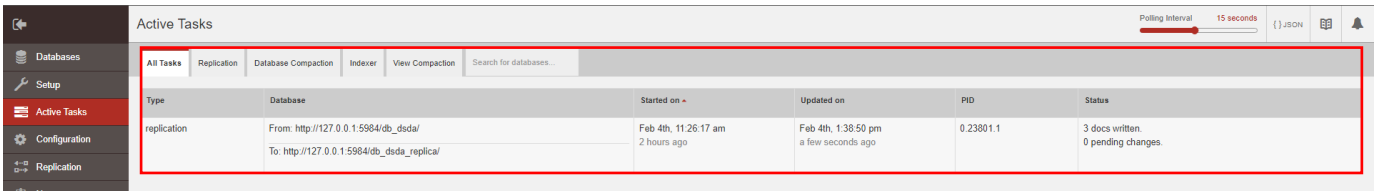
View Compaction : Similaire à la compaction de la base de données, la compaction des vues réduit l'espace disque utilisé par les vues, qui sont des représentations indexées des données créées par les fonctions map et reduce dans CouchDB. En compactant une vue, nous enlevons les données obsolètes et optimisons la performance des requêtes basées sur ces vues.

Search for data bases...: Cela n'est pas une catégorie de tâche, mais une fonctionnalité de recherche qui vous permet de filtrer et de trouver rapidement une base de données spécifique parmi celles listées dans l'interface, pour ensuite examiner ses tâches actives.

Chacune de ces catégories nous aide à surveiller des aspects spécifiques des opérations en cours sur notre serveur CouchDB. Elles sont particulièrement utiles pour l'administration du système, la surveillance des performances et le dépannage.

Exemple :

Nous voyons une tâche de réplication. Elle indique que la base de données "dsdal" est en train d'être répliquée vers "dsda_replica". Les informations affichées incluent :



Le type de tâche : dans ce cas, une réplication.

La base de données source et cible.

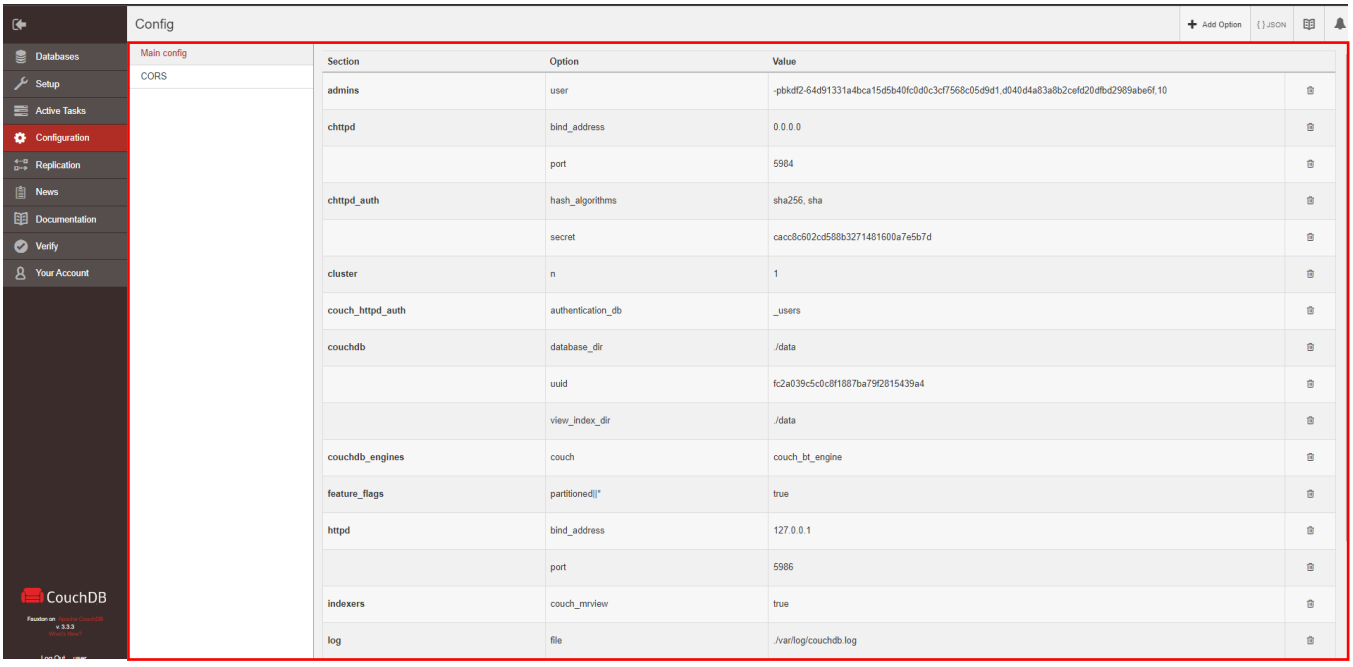
La date de début de la tâche et la dernière mise à jour.

Le PID (Process Identifier) qui est l'identifiant du processus de cette tâche spécifique.

Le statut de la tâche, qui comprend le nombre de documents écrits et le nombre de changements en attente.

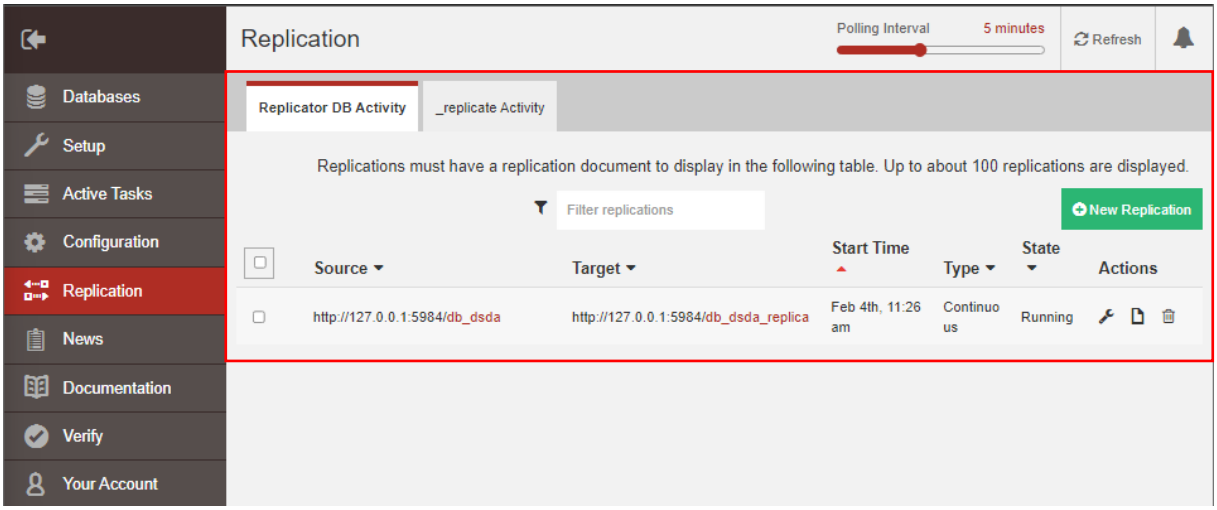
Configuration

La rubrique de configuration dans CouchDB vous permet de gérer différents paramètres qui influencent le comportement du serveur CouchDB.



Réplication

Dans CouchDB, la réplication est un processus qui permet de copier et de synchroniser des données entre deux bases de données, potentiellement situées sur différents serveurs ou instances de CouchDB. C'est un élément clé de la conception de CouchDB, permettant une haute disponibilité des données et une tolérance aux pannes, ainsi que le travail hors ligne suivi d'une synchronisation.



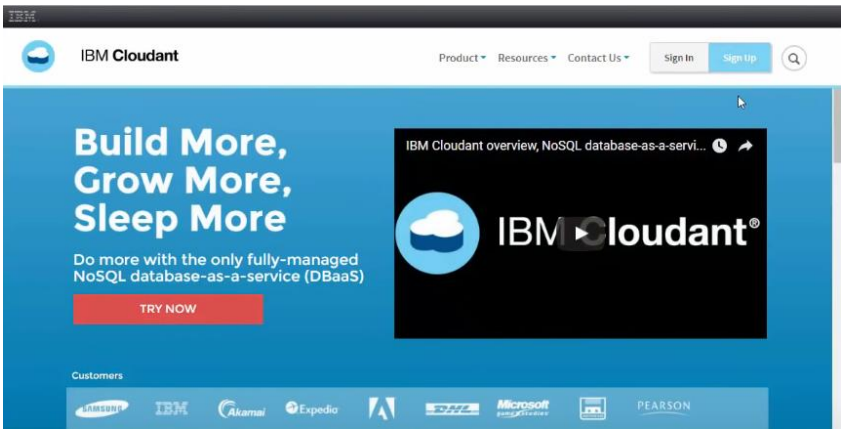
Ici, nous avons accès options :

Replicator DB Activity : Cette activité concerne les réplications définies par des documents de réplication qui sont stockés dans une base de données spéciale appelée _replicator. Quand un document de réplication est ajouté à cette base de données, CouchDB planifie et exécute la réplication selon les paramètres spécifiés dans le document. Les avantages de l'utilisation de la base de données _replicator incluent la persistance des tâches de réplication (elles ne sont pas perdues après un redémarrage) et la possibilité de les gérer via l'interface utilisateur ou l'API HTTP de CouchDB.

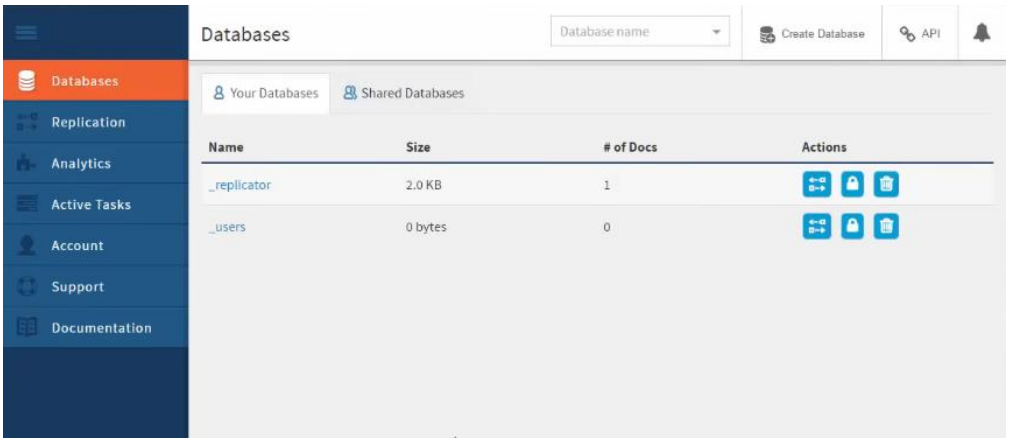
_replicate Activity : Cela fait référence à une opération de réplication déclenchée par une requête à l'API _replicate de CouchDB. C'est une manière plus immédiate et souvent temporaire de démarrer une réplication, car les informations ne sont pas stockées dans la base de données _replicator et la réplication n'est pas automatiquement redémarrée si le serveur CouchDB est redémarré.

Exemple :

Nous allons réaliser un cas de réplication en utilisant IBM Cloudant. Pour cela, nous allons nous enregistrer sur IBM Cloudant :

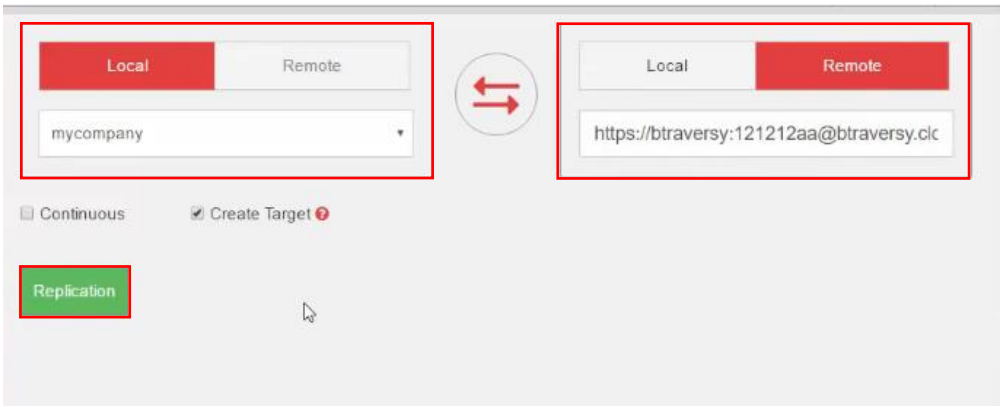


Une fois fait, nous voici sur l'interface d'IBM Cloudant :

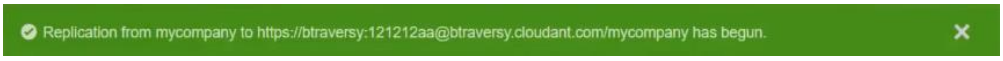


Nous pouvons voir que l'interface de ce service est très similaire à celle de CouchDB. Maintenant, nous allons réaliser la réplication depuis l'outil CouchDB. Pour cela, il nous suffit de nous rendre dans l'onglet Replication, puis de sélectionner notre base de données mycompany en local, puis d'entrer nos informations distantes de notre base de données IBM Cloudant.

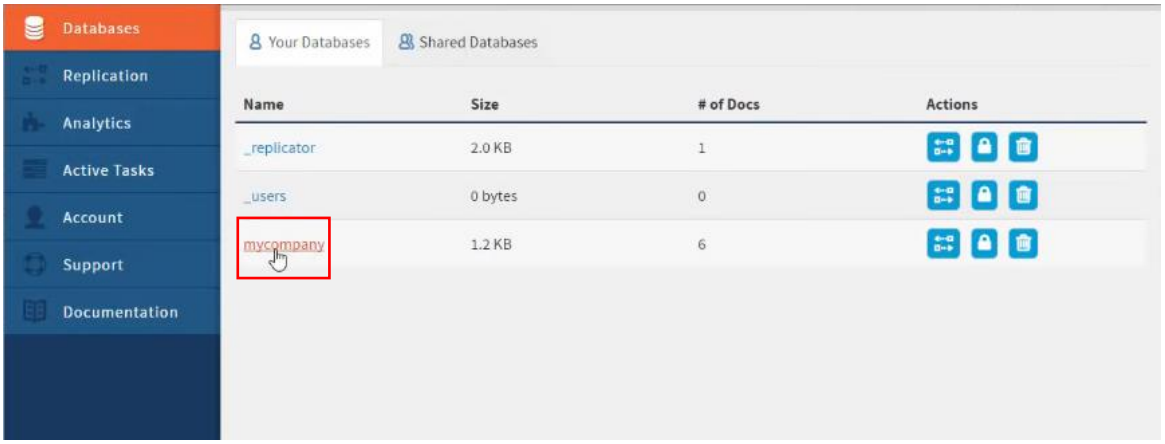
Une fois les informations entrées, il nous suffit de sélectionner "Réplication" pour démarrer le processus.



Une notification apparaît nous indiquant que la réplication a bien réussi.

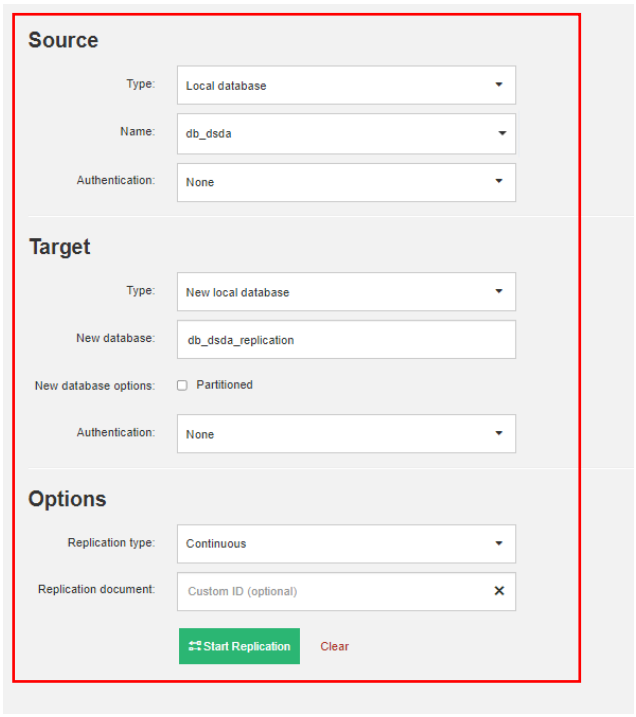


Maintenant, si nous nous rendons sur IBM Cloudant, nous pouvons retrouver notre base de données mycompany répliquée :



Maintenant, nous allons réaliser une réplication locale pour avoir une copie de notre base de données ayant la même localisation. Nous allons donc dans Replication, puis sur Replicator DB Activity, puis sélectionnons "Nouvelle Réplication" :

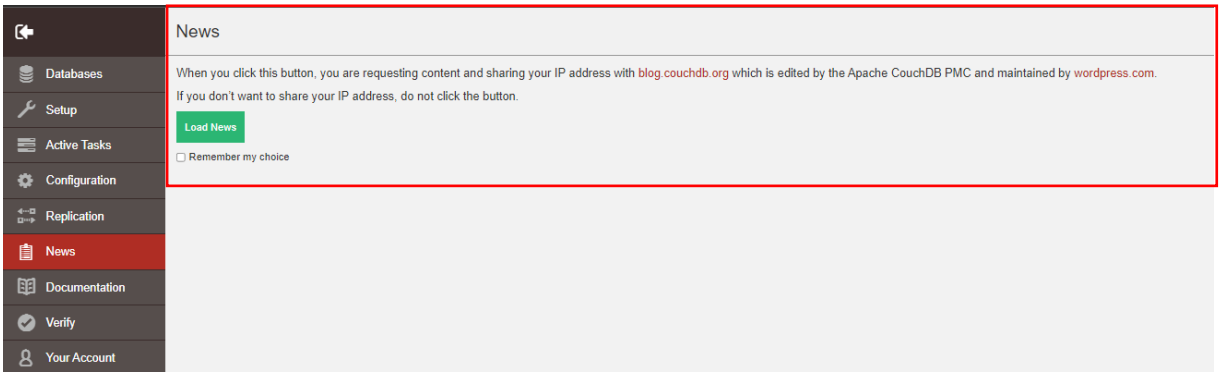
Nous sélectionnons les différents paramètres de réplication souhaités, puis sélectionnons "Démarrer la réplication" :



Nous pouvons voir que la réplication a bien réussi :

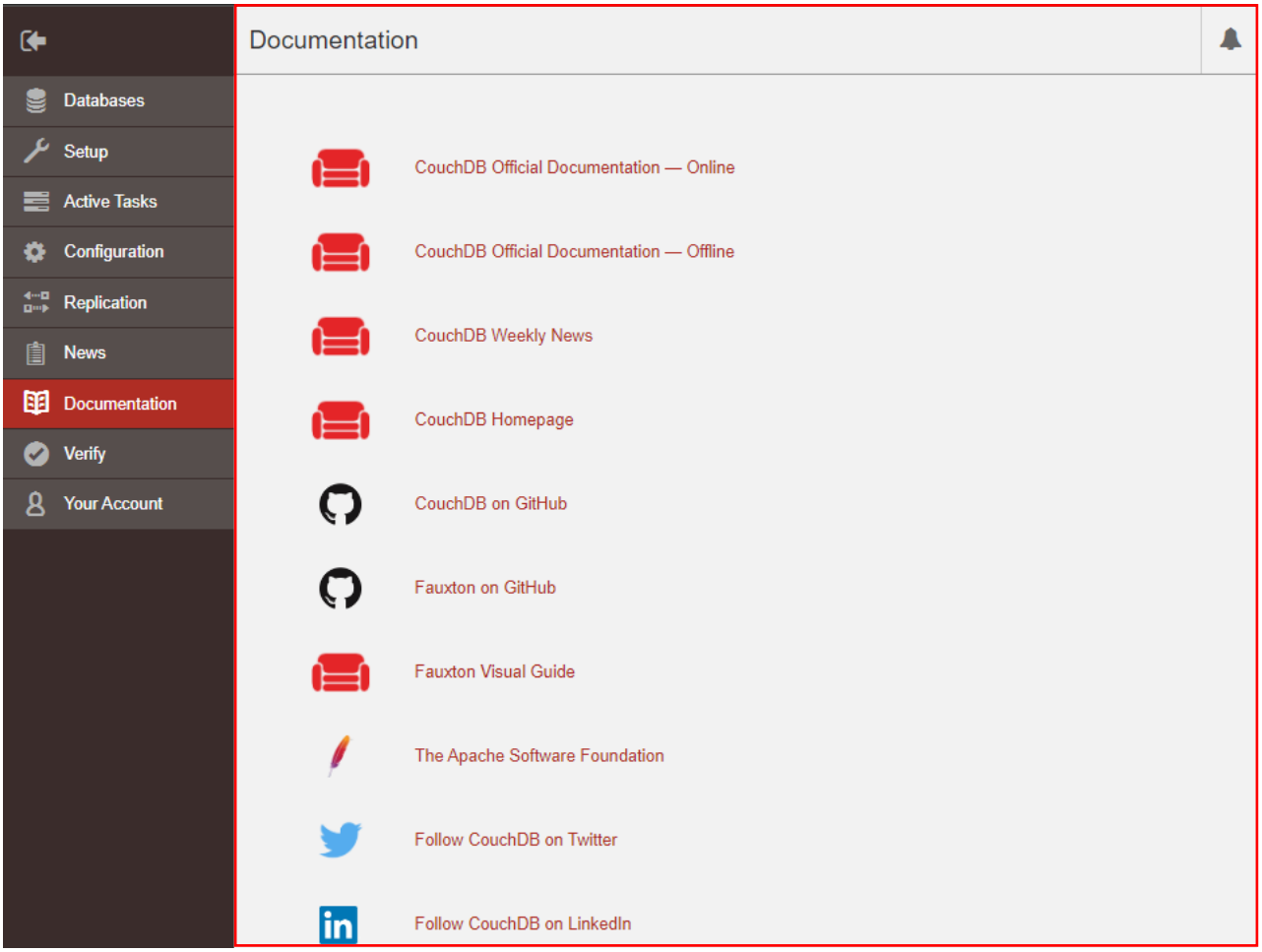
News

La section "News" dans l'interface utilisateur de CouchDB est destinée à fournir aux utilisateurs des actualités et des mises à jour liées à CouchDB. En cliquant sur "Load News", l'interface utilisateur de CouchDB fera une requête pour récupérer les dernières nouvelles et les informations depuis le blog de CouchDB.



Documentation

La rubrique "Documentation" dans l'interface utilisateur de CouchDB sert de centre de ressources pour les utilisateurs et les développeurs. Elle fournit des liens vers divers types de documents et de ressources pour aider les utilisateurs à comprendre et à utiliser CouchDB.



Utilisation de Curl

Curl est un outil en ligne de commande utilisé pour transférer des données avec des URL. Il supporte une variété de protocoles, dont HTTP, HTTPS, FTP... curl est largement utilisé pour interagir avec des serveurs web, effectuer des requêtes, soumettre des données de formulaire, télécharger des fichiers, et réaliser d'autres opérations liées au transfert de données sur le web.

Dans le contexte de CouchDB, qui est une base de données NoSQL orientée document accessible via HTTP, curl est particulièrement utile pour plusieurs raisons :

Interface RESTful : CouchDB expose une API RESTful qui permet d'interagir avec la base de données en utilisant des requêtes HTTP standards. curl est un outil idéal pour effectuer ces requêtes directement depuis la ligne de commande, ce qui facilite la création, la lecture, la mise à jour, et la suppression de documents, ainsi que la gestion de la base de données sans nécessiter une interface graphique ou un client spécialisé.

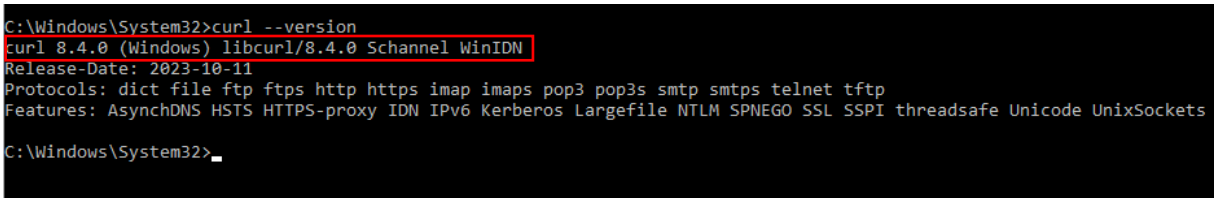
Développement et débogage : curl est souvent utilisé par les développeurs pour tester et déboguer leurs applications CouchDB. Il permet d'envoyer rapidement des requêtes personnalisées à CouchDB et de voir immédiatement les réponses, ce qui aide à comprendre comment l'application interagit avec la base de données.

Automatisation : Les commandes curl peuvent être intégrées dans des scripts pour automatiser certaines tâches liées à la gestion de CouchDB, comme la sauvegarde de données, la réplication de bases de données, ou l'exécution de vues et requêtes.

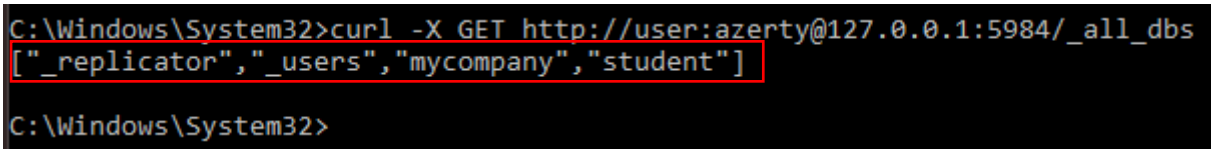
Simplicité et accessibilité : curl est disponible sur la plupart des systèmes d'exploitation et est déjà installé par défaut sur de nombreux d'entre eux, ce qui le rend facilement accessible aux développeurs et aux administrateurs système.

Exemple d'utilisation avec l'invite de commande :

Après avoir installé curl, vérifions que celui-ci est bien installé :



Nous pouvons voir que curl 8.4.0 est bien installé sur notre machine. Maintenant, nous allons récupérer l'ensemble des bases de données que nous avons sur notre profil CouchDB :



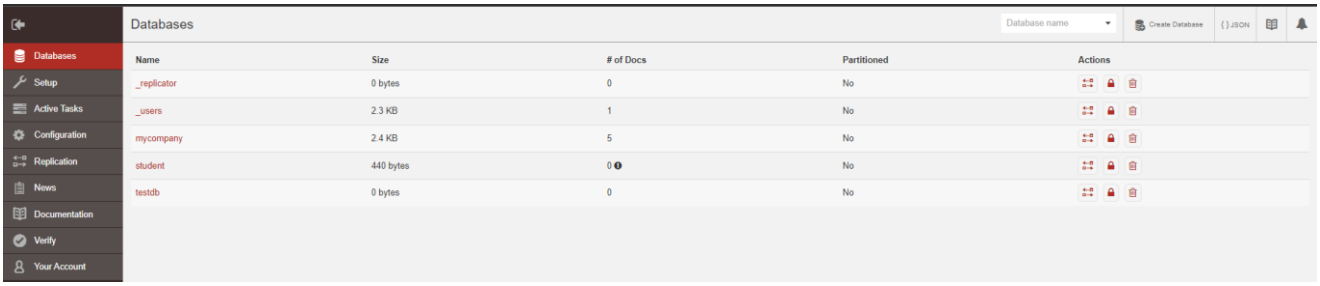
Ici, nous pouvons voir que curl a récupéré l'ensemble des bases de données de notre profil, telles que "mycompany". Maintenant, nous allons insérer une nouvelle base de données : "testdb" :

```
C:\Windows\System32>curl -X PUT http://user:azerty@127.0.0.1:5984/testdb
{"ok":true}
```

Ici, nous pouvons voir que la base de données a bien été créée grâce à la méthode PUT. Si nous récupérons à nouveau l'ensemble de nos bases de données avec la méthode GET, nous pouvons voir que "testdb" a bien été ajouté :

```
C:\Windows\System32>curl -X GET http://user:azerty@127.0.0.1:5984/_all_dbs
["_replicator","_users","mycompany","student","testdb"]
```

Nous retrouvons bien "testdb". Nous pouvons aussi le vérifier en regardant directement dans l'outil CouchDB :



"testdb" a bien été ajouté. Maintenant, pour nous simplifier la tâche, nous allons définir une variable "SERVER" qui prendra le lien de notre API. Cela nous facilitera l'utilisation et allégera les commandes à écrire :

```
C:\Windows\System32>SET SERVER=http://user:azerty@127.0.0.1:5984

C:\Windows\System32>ECHO %SERVER%
http://user:azerty@127.0.0.1:5984

C:\Windows\System32>curl -X GET %SERVER%/_all_dbs
["_replicator","_users","mycompany","student","testdb"]

C:\Windows\System32>
```

La variable SERVER a bien été créée et comporte le lien API de nos bases de données CouchDB. Maintenant, nous allons récupérer les informations de la "view3" créée précédemment :

```
C:\Windows\System32>curl -X GET %SERVER%/mycompany/_design/view3/_view/new-view
{"total_rows":2,"offset":0,"rows":[
{"id":"3d0d42f00ca39877c959a218b4003384","key":"3d0d42f00ca39877c959a218b4003384","value":{"name":"Morgan Senechal","email":"morgan.senechal@efrei.net","phone":"+33 6 63 66 90 80"}},
{"id":"3d0d42f00ca39877c959a218b4004412","key":"3d0d42f00ca39877c959a218b4004412","value":{"name":"Sam Smith ","email":"sam.smith@efrei.net","phone":"+33 6 73 46 50 20"}}
]}
```

Comme nous pouvons le voir, les informations de la "view3" contenues dans "mycompany" ont bien été récupérées. Maintenant, nous allons utiliser curl pour créer un nouveau document. Comme nous l'avons vu précédemment, lorsqu'on crée un document sur l'outil CouchDB, un ID est automatiquement généré. Pour pouvoir récupérer un ID automatiquement généré, nous allons demander à CouchDB de nous générer cet ID en utilisant la commande "uuids".

```
C:\Windows\System32>curl -X GET %SERVER%/_uuids
{"uuids":["3d0d42f00ca39877c959a218b400595e"]}

C:\Windows\System32>
```

CouchDB vient de nous générer cet ID.

Maintenant, nous allons copier cet ID puis créer notre nouveau document dans "mycompany" :

```
C:\Windows\System32>curl -X PUT %SERVER%/mycompany/3d0d42f00ca39877c959a218b400595e -d "{\"name\":\"Ryan Seb\",\"email\":\"ryan.seb@efrei.net\"}"
{"ok":true,"id":"3d0d42f00ca39877c959a218b400595e","rev":"1-d68e07c9349caef8f757a52eb56975b6"}
```

Le nouveau document a bien été créé avec les informations "name" et "email". Maintenant, vérifions que nos informations ont bien été ajoutées dans la "view3" :

```
C:\Windows\System32>curl -X GET %SERVER%/mycompany/_design/view3/_view/new-view
{"total_rows":3,"offset":0,"rows":[
{"id":"3d0d42f00ca39877c959a218b4003384","key":"3d0d42f00ca39877c959a218b4003384","value":{"name":"Morgan Senechal","email":"morgan.senechal@efrei.net","phone":"+33 6 63 66 90 80"}},
{"id":"3d0d42f00ca39877c959a218b4004412","key":"3d0d42f00ca39877c959a218b4004412","value":{"name":"Sam Smith ","email":"sam.smith@efrei.net","phone":"+33 6 73 46 50 20"}},
{"id":"3d0d42f00ca39877c959a218b400595e","key":"3d0d42f00ca39877c959a218b400595e","value":{"name":"Ryan Seb","email":"ryan.seb@efrei.net"}}
]}
```

Nous pouvons retrouver les nouvelles informations ajoutées précédemment dans la "view3". Nous pouvons aussi vérifier l'ajout de ce nouveau document directement dans l'outil CouchDB :

Nous retrouvons bien le nouveau document inséré.

Python avec CouchDB

L'utilisation de Python avec CouchDB présente plusieurs avantages, notamment en raison de la simplicité d'utilisation de Python et des fonctionnalités de CouchDB :

Facilité d'intégration : Python dispose de bibliothèques comme CouchDB-Python et Cloudant qui facilitent l'interaction avec CouchDB. Ces bibliothèques permettent une intégration transparente et offrent des méthodes simples pour créer, lire, mettre à jour et supprimer des documents dans une base de données CouchDB.

Manipulation aisée des documents JSON : CouchDB utilise le format JSON pour le stockage des documents. Python, avec son support natif du JSON via le module json et des bibliothèques tierces comme requests pour les opérations HTTP, rend la manipulation de ces documents et l'interaction avec CouchDB particulièrement aisée et intuitive.

Scalabilité et flexibilité : CouchDB est conçu pour être hautement scalable et répliquer les données facilement d'une instance à une autre. L'utilisation de Python pour écrire des scripts ou des applications qui interagissent avec CouchDB permet de tirer parti de cette scalabilité et flexibilité, notamment pour des applications distribuées ou des architectures de microservices.

Développement rapide : Python est réputé pour sa syntaxe claire et concise, ce qui accélère le développement des applications. Lorsque vous travaillez avec CouchDB, cette rapidité est un atout, permettant de mettre en place des prototypes ou des applications complètes en moins de temps.

Traitement de données : La combinaison de Python et CouchDB est puissante pour le traitement de données. Python offre des bibliothèques avancées pour l'analyse de données (comme Pandas) et la visualisation (comme Matplotlib et Seaborn). Lorsque ces capacités sont associées à la capacité de CouchDB à stocker et gérer de grandes quantités de données non structurées, vous disposez d'un ensemble d'outils robuste pour le traitement et l'analyse de données.

Développement d'applications web et de services REST : Python, avec des frameworks comme Flask ou Django, est largement utilisé pour le développement web. L'utilisation de CouchDB comme backend pour stocker des données rend le développement d'applications web et de services RESTful à la fois simple et efficace, profitant de la nature sans schéma de CouchDB pour une évolutivité des applications facilitée.

Dans cet exemple, nous allons utiliser Anaconda3 et l'IDE Spider. Pour installer CouchDB sur Anaconda3, nous devons installer le package "couchdb" :

L'installation du package "couchdb" a été réussie.

Maintenant, nous pouvons nous rendre sur Spyder et commencer à utiliser Python avec CouchDB :

```python
# Importation du module CouchDB et utilisation de l'alias 'cdb' pour y faire référence.
import couchdb as cdb

# Définition des variables pour le nom d'utilisateur et le mot de passe pour l'authentification CouchDB.
username = 'user'
password = 'azerty'

# Création d'un objet 'Server' pour interagir avec le serveur CouchDB local.
# Les informations d'authentification sont incluses dans l'URL de connexion.
couchDb = cdb.Server("http://%s:%s@localhost:5984" % (username, password))

#Vérifie les informations connexion
couchDb.resource.credentials

# Définition du nom de la base de données à utiliser ou à créer.
dbname = 'db_dsda'

# Vérification si la base de données spécifiée par 'dbname' existe sur le serveur CouchDB.
if dbname in couchDb:
    # Si la base de données existe, on se connecte à cette base de données.
    db = couchDb[dbname]
else:
    # Si la base de données n'existe pas, on crée une nouvelle base de données avec ce nom.
    db = couchDb.create(dbname)

# Récupération et affichage des informations de la base de données connectée ou créée.
# Cela inclut des données telles que le nombre de documents, la taille de la base de données, etc.
db.info()
```

Ce script Python est utilisé pour se connecter à un serveur CouchDB local. Il vérifie l'existence d'une base de données spécifique, la crée si elle n'existe pas, puis récupère et affiche des informations sur cette base de données.

Dans notre cas, la base de données a été créée. Nous pouvons constater sa création dans l'outil CouchDB :

Nous pouvons aussi constater des informations concernant cette base de données retournées sur la console Python grâce à la commande "db.info()" :

```
{'instance_start_time': '1707038061',
 'db_name': 'db_dsda',
 'purge_seq': '0-g1AAAABXeJzLYWBgYMpgTmEQTM4vTc5ISXLIyU9OzMnILy7JAUnlsQBJhgYg9R8IshIZ8KhNZEiqhyjKAgBm5Rxs',
 'update_seq': '0-
g1AAAACbeJzLYWBgYMpgTmEQTM4vTc5ISXLIyU9OzMnILy7JAUnlsQBJhgYg9R8IsjKYExlygQLsxkaWlmaJKdj04TEtkSGpHsWYNPNkQyNzC2wasgAPJzB6',
 'sizes': {'file': 16700, 'external': 0, 'active': 0},
 'props': {},
 'doc_del_count': 0,
 'doc_count': 0,
 'disk_format_version': 8,
 'compact_running': False,
 'cluster': {'q': 2, 'n': 1, 'w': 1, 'r': 1}}
```

Maintenant, nous allons créer 2 nouveaux documents dans notre nouvelle base de données :

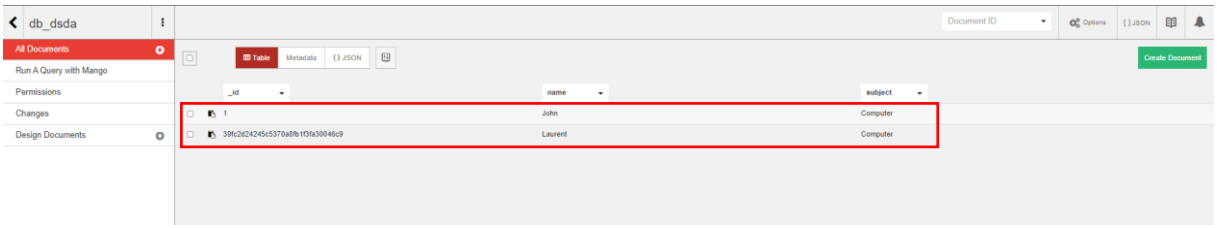
```
# Crée un document avec les informations sur Laurent et son sujet d'étude.
doc1 = {'name': 'Laurent', 'subject': 'Computer'}
# Crée un second document avec un ID spécifié, les informations sur John et son sujet d'étude.
doc2 = {'_id': '1', 'name': 'John', 'subject': 'Computer'}

# Enregistre le premier document dans la base de données CouchDB et récupère l'ID et la révision du document.
docId1, docRev1 = db.save(doc1)
# Enregistre le second document dans la base de données CouchDB et récupère l'ID et la révision du document.
# L'ID est explicitement défini ici, donc le document aura l'ID '1' dans la base de données.
docId2, docRev2 = db.save(doc2)

# Récupère le document correspondant à 'docId1' depuis la base de données CouchDB.
docFromDb = db.get(docId1)
# Affiche le document récupéré. Cette ligne retourne le document pour que vous puissiez le voir.
docFromDb
```

Créez et enregistrez deux documents, puis récupérez l'un des documents en utilisant son ID et stockez-le dans "docFromDb", prêt à être utilisé ou affiché.

Voici le résultat de l'affichage du document 1 sur la console :

```
In [23]: docFromDb
Out[23]: <Document '39fc2d24245c5370a8fb1f3fa30046c9'@'1-54aa6f2c315240b073ea0df938eba75a' {'name': 'Laurent', 'subject':
'Computer'}>
```

Nous pouvons également nous assurer que les deux documents ont bien été créés en regardant directement dans l'outil CouchDB :



De plus, grâce à "db.info()", nous pouvons maintenant voir que notre base de données contient 2 documents :

```
In [32]: db.info()
Out[32]:
{'instance_start_time': '1707038061',
 'db_name': 'db_dsda',
 'purge_seq': '0-g1AAAABXeJzLYWBgYMpgTmEQTM4vTc5ISXLIyU9OzMnILy7JAUnlsQBJhgYg9R8IshIZ8KhNZEiqhyjKAgBm5Rxs',
 'update_seq': '36-
g1AAAACbeJzLYWBgYMpgTmEQTM4vTc5ISXLIyU9OzMnILy7JAUnlsQBJhgYg9R8IsjKYE6VygQLsxkaWlmaJKdj04TEtkSGpHmoMF9iYNPNkQyNzC2wasgAbJTCe',
 'sizes': {'file': 164235, 'external': 101, 'active': 3492},
 'props': {},
 'doc_del_count': 13,
 'doc_count': 2,
 'disk_format_version': 8,
 'compact_running': False,
 'cluster': {'q': 2, 'n': 1, 'w': 1, 'r': 1}}
```

Maintenant, nous allons mettre à jour l'un de nos documents :

```
# Met à jour la valeur de la clé 'name' dans le dictionnaire doc1 pour qu'elle soit 'Morgan'.
doc1['name'] = 'Morgan'
# Enregistre le document modifié dans la base de données CouchDB. Cela va soit créer un nouveau document,
# soit mettre à jour le document existant si 'doc1' avait déjà un champ '_id'.
# La fonction 'save' retourne l'identifiant unique du document et la nouvelle révision du document après la sauvegarde.
docId1, docRev1 = db.save(doc1)

# Récupère le document de la base de données CouchDB en utilisant l'identifiant unique retourné précédemment lors de la sauvegarde.
# Cela permet d'obtenir la version la plus récente du document après la modification.
docFromDb = db.get(docId1)

# Affiche le contenu du document récupéré de la base de données.
# Cela devrait montrer le document avec le nom mis à jour à 'Morgan'.
docFromDb
```

Met à jour le champ 'name' du document doc1 pour le nom 'Morgan', sauvegarde ce document mis à jour dans la base de données CouchDB, récupère le document en utilisant son ID unique, puis affiche ce document récupéré.

Nous pouvons constater la mise à jour de l'information depuis la console Python et l'outil CouchDB :

Console Python :

```
In [36]: docFromDb
Out[36]: <Document '39fc2d24245c5370a8fb1f3fa3008394'@'2-64bb3dc9cb16d40e261ea9c276d6d90f' {'name': 'Morgan',
'subject': 'Computer'}>
```

Outils CouchDB :

Maintenant, nous allons créer un nouveau document vide :

```
# Crée un nouveau document vide (sans données).
doc3 = {}

# Enregistre le document vide dans la base de données CouchDB.
docId3, docRev3 = db.save(doc3)
```

Lorsque nous créons un document vide, CouchDB attribue automatiquement un _id et _rev. Le document 3 étant vide, nous allons utiliser la fonction "copy()" pour copier les informations du document 1 vers le document 3 :

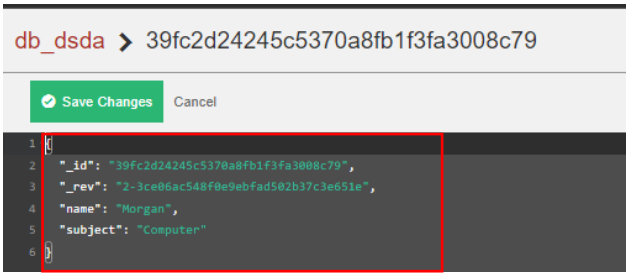
```
#Copy les informations du document 1 vers le document 3
db.copy(doc1, doc3)
```

Nous pouvons voir que l'opération de "copy()" a bien réussi en consultant la console Python et l'outil CouchDB :

Console Python :

```
In [39]: db.copy(doc1, doc3)
Out[39]: '2-3ce06ac548f0e9ebfad502b37c3e651e'
```

CouchDB :



Maintenant, nous allons afficher tous les documents de notre base de données :

```
#Affiche tous les documents de la base de données
for row in db.view("_all_docs"):
    print(row)
```

Depuis la console Python, nous retrouvons bien nos 3 documents créés précédemment.

```
In [40]: for row in db.view("_all_docs"):
    ...:     print(row)
<Row id='1', key='1', value={'rev': '9-5e788a5f421eb36970f5db5c1e723173'}>
<Row id='39fc2d24245c5370a8fb1f3fa3008394', key='39fc2d24245c5370a8fb1f3fa3008394', value={'rev':
'2-64bb3dc9cb16d40e261ea9c276d6d90f'}>
<Row id='39fc2d24245c5370a8fb1f3fa3008c79', key='39fc2d24245c5370a8fb1f3fa3008c79', value={'rev':
'2-3ce06ac548f0e9ebfad502b37c3e651e'}>
```

"Note : Nous pouvons choisir d'afficher uniquement les clés de nos documents en utilisant la fonction .keys attachée à row : row.keys

Maintenant, nous allons supprimer le document 3 :"

```
# Récupère le document identifié par 'docId3' de la base de données CouchDB.
# 'docId3' est l'identifiant unique du document que nous voulons récupérer.
doc3FromDb = db.get(docId3)

# Supprime le document récupéré de la base de données CouchDB.
# La méthode 'delete' prend un document et le supprime de la base de données en utilisant l'ID et la révision du document.
# La suppression est réussie seulement si la révision actuelle du document correspond à celle passée à 'delete'.
db.delete(doc3FromDb)

# Récupère et affiche des informations sur la base de données actuelle.
db.info()
```

Nous pouvons constater la suppression du document 3 en utilisant db.info() et en regardant le nombre total de documents présents dans notre base de données :

Console Python :

```
In [43]: db.info()
Out[43]:
{'instance_start_time': '1707038061',
 'db_name': 'db_dsda',
 'purge_seq': '0-g1AAAABXeJzLYWBgYMpgTmEQTM4vTc5ISXLIyU9OzMnILy7JAUnlsQBJhgYg9R8IshIZ8KhNZEiqhyjKAgBm5Rxs',
 'update_seq': '40-
g1AAAACbeJzLYWBgYMpgTmEQTM4vTc5ISXLIyU9OzMnILy7JAUnlsQBJhgYg9R8IsjKYE6VzgQLsxkaWlmaJKdj04TEtkSGpHmoML9iYNPNkQyNz
C2wasgAb9DCi',
 'sizes': {'file': 180622, 'external': 102, 'active': 4181},
 'props': {},
 'doc_del_count': 14,
 'doc_count': 2,
 'disk_format_version': 8,
 'compact_running': False,
 'cluster': {'q': 2, 'n': 1, 'w': 1, 'r': 1}}
```

Outils CouchDB :

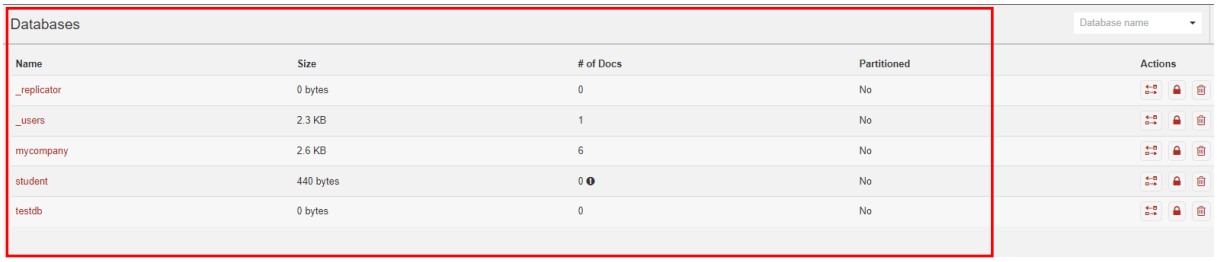
Maintenant, nous allons supprimer notre base de données :

```
#Suprimmer la base de données
couchDb.delete(dbname)

db.info()
```

Nous pouvons constater sa suppression en observant l'erreur apparaissant sur la console Python lorsque nous souhaitons afficher les informations relatives à dbname :

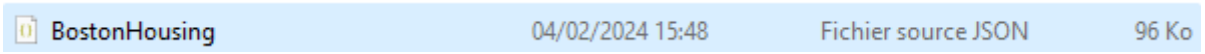
```
ResourceNotFound: ('not_found', 'Database does not exist.')
```

Nous pouvons aussi constater sa suppression sur l'outil CouchDB :



Import JSON data

Voici un exemple de fichier JSON que nous pouvons importer sur CouchDB :



Pour importer BostonHousing.json, nous allons utiliser Curl :

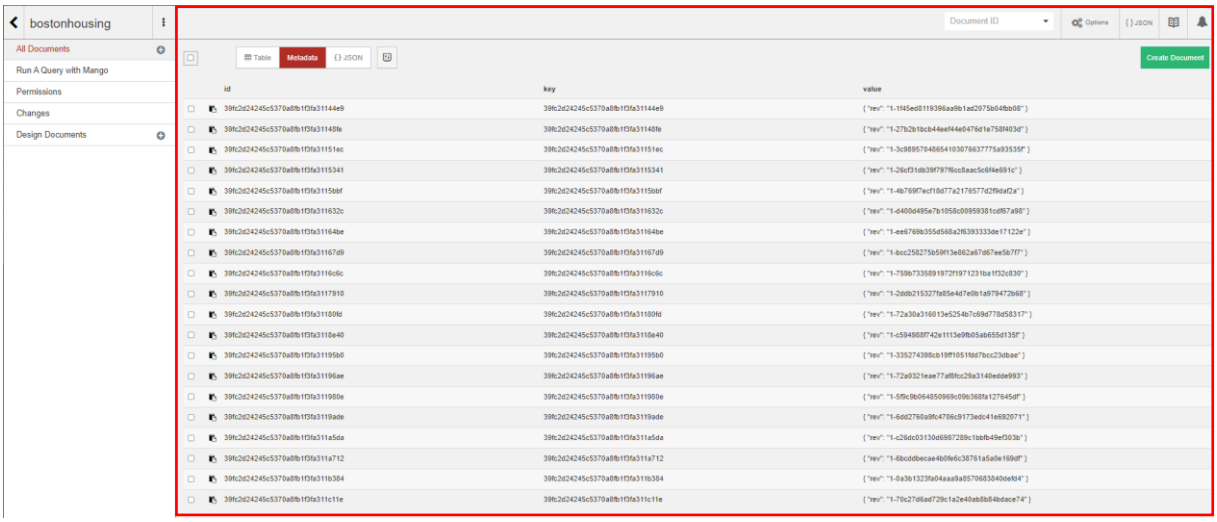
Input :

```
C:\Windows\System32>curl -X POST http://user:azerty@localhost:5984/bostonhousing/_bulk_docs -H "Content-Type: applica
tion/json" -d @D:\Bureau\NOSQL\BostonHousing.json
```

Output :

Nous pouvons retrouver ici les données présentes dans notre fichier JSON.

Nous pouvons vérifier cela sur l'interface Fauxton :

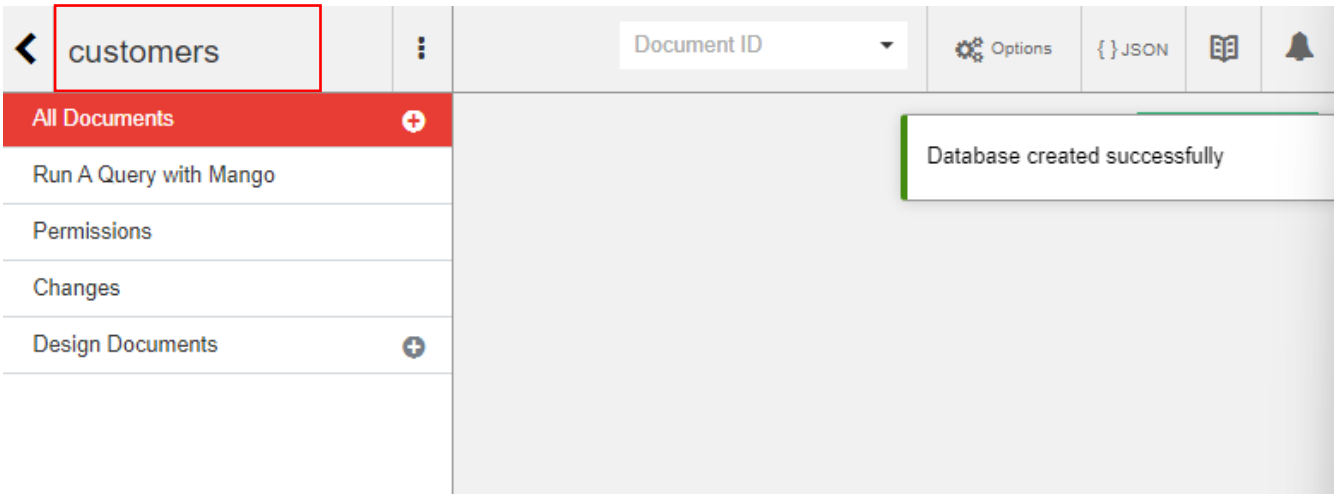


Les différents documents contenant les données de BostonHousing.json ont bien été créés. Nous allons donc pouvoir commencer à traiter les différentes fonctionnalités avancées de CouchDB.

Exemple Concret : NodeJS & CouchDB :

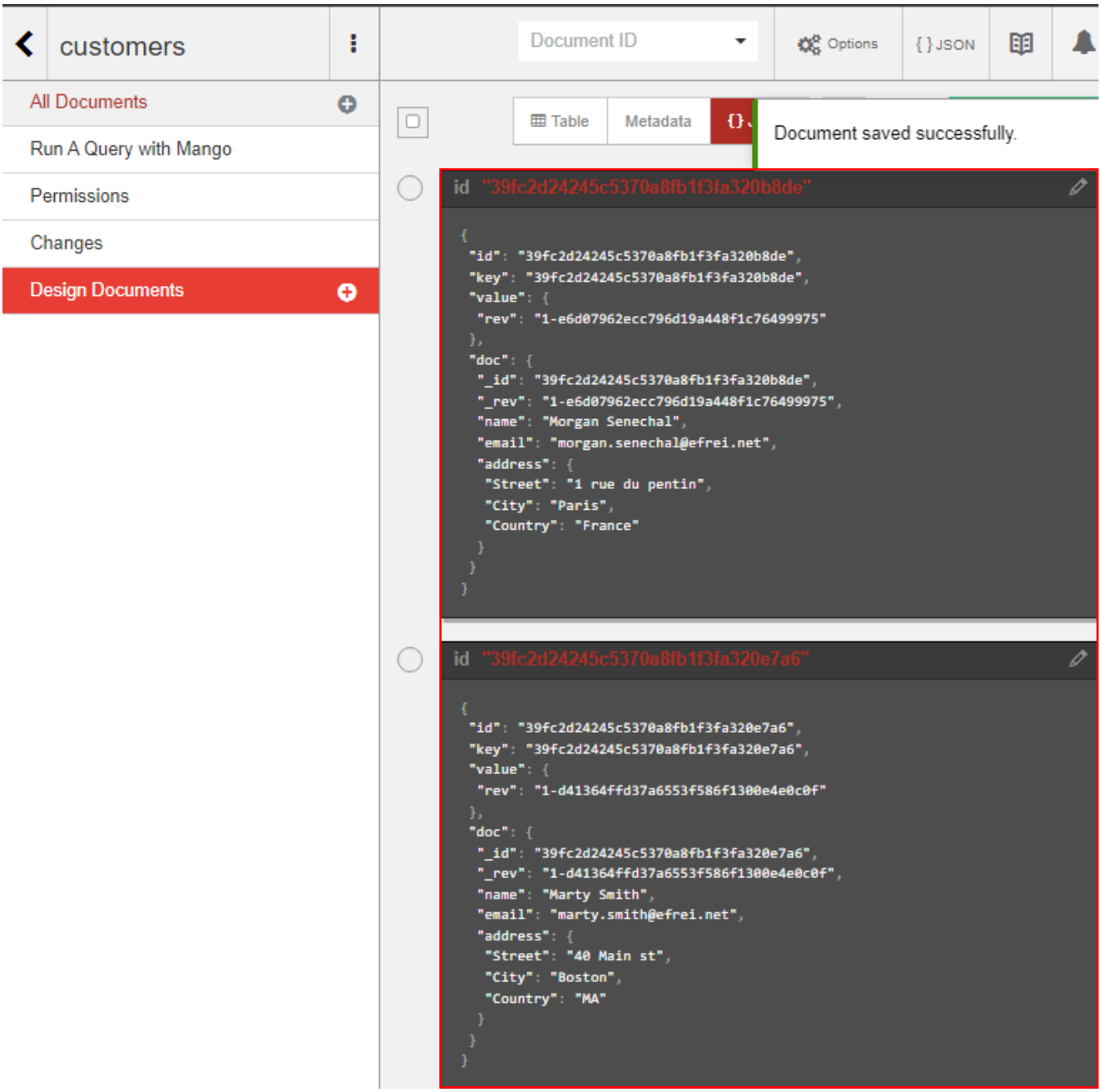
Create Database

Pour cet exemple concret avec Node.js, nous allons créer une nouvelle base de données : customers.



Create Document

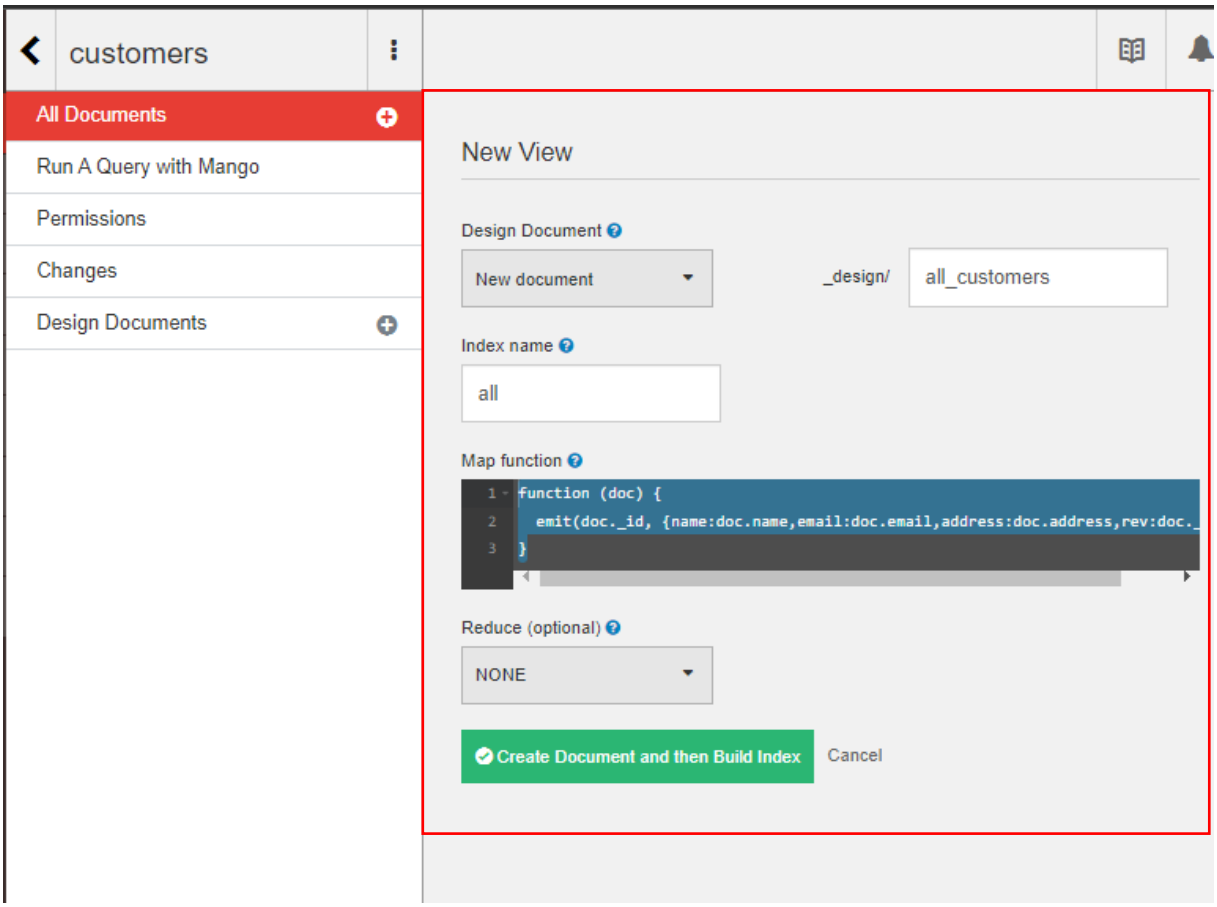
Nous allons maintenant créer de nouveaux documents dans notre base de données customers :



Create View

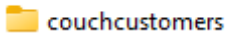
Nous allons crée ici, une nouvelle view map-reduce.

```
function (doc) {
  emit(doc._id, {name:doc.name,email:doc.email,address:doc.address,rev:doc._rev});
}
```



Create NodeJS application

Maintenant, nous allons nous atteler à la création de l'application Node.js. Pour cela, nous allons, dans un premier temps, créer un nouveau dossier : couchcustomers.



Une fois le dossier crée, nous allons ouvrir l'invite de commande en tant qu'administrateur puis ce rendre dans l'espace ou se contient le nouveau fichier crée :

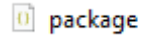
```
C:\>cd Projects/couchcustomers

C:\Projects\couchcustomers>
```

Après être entré dans le bon espace, nous allons créer un package.json :

```
C:\Projects\couchcustomers>npm init
```

```
About to write to C:\Projects\couchcustomers\package.json:

{
  "name": "couchcustomers",
  "version": "1.0.0",
  "description": "Simple customer list",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this ok? (yes)

C:\Projects\couchcustomers>
```

Une fois le package.json créé, nous pouvons le retrouver dans notre dossier créé précédemment :



Maintenant, nous allons installer les différentes dépendances :

```
C:\Projects\couchcustomers>npm install express body-parser ejs node-couchdb --save
```

```
    | +-- assert-plus@0.2.0
    | +-- jsprim@1.3.1
    | | +-- extsprintf@1.0.2
    | | +-- json-schema@0.2.3
    | | `-- verror@1.3.6
    | `-- sshpk@1.10.1
    |   +-- asn1@0.2.3
    |   +-- assert-plus@1.0.0
    |   +-- bcrypt-pbkdf@1.0.0
    |   +-- dashdash@1.14.0
    |   | `-- assert-plus@1.0.0
    |   +-- ecc-jsbn@0.1.1
    |   +-- getpass@0.1.6
    |   | `-- assert-plus@1.0.0
    |   +-- jodid25519@1.0.2
    |   +-- jsbn@0.1.0
    |   `-- tweetnacl@0.14.3
    +-- is-typedarray@1.0.0
    +-- isstream@0.1.2
    +-- json-stringify-safe@5.0.1
    +-- node-uuid@1.4.7
    +-- oauth-sign@0.8.2
    +-- qs@6.1.0
    +-- stringstream@0.0.5
    +-- tough-cookie@2.2.2
    `-- tunnel-agent@0.4.3

npm WARN couchcustomers@1.0.0 No repository field.

C:\Projects\couchcustomers>
```

Maintenant que les dépendances sont installées, nous allons créer notre application app.js :

```
package-lock.json   JS app.js
JS app.js > ...
1   const express = require('express');
2   const bodyParser = require ('body-parser');
3   const path = require ('path');
4   const NodeCouchDb = require('node-couchdb');
5
6
7   const app = express();
8
9   app.set('view engine', 'ejs');
10  app.set('views', path.join(__dirname,'view'));
11
12  app.use(bodyParser.json());
13  app.use(bodyParser.urlencoded({extended:false}));
14
15  app.get('/', function(req,res){
16      res.send('Working...');
17
18  });
19
20  app.listen(3000,function(){
21      console.log('Server Started On Port 3000');
22  });
23
24
```

Ce code est un script de base pour une application web utilisant Node.js avec plusieurs modules comme Express, Body-Parser, Path, et **Node-CouchDb**.

Maintenant que le script app.js est opérationnel, nous pouvons démarrer notre application Node.js :

```
C:\Projects\couchcustomers>node app
Server Started On Port 3000
```

Nous pouvons voir que le serveur est lancé :

Maintenant, nous allons ajouter les fonctionnalités nécessaires pour créer un formulaire permettant d'ajouter et de supprimer des customers.

Voici notre index.ejs :

```ejs
<!-- Formulaire pour ajouter un nouveau client -->
<h3>Add Customer</h3>
<form method="post" action="/customer/add">
    <!-- Champ de saisie pour le nom du client -->
    <input type="text" name="name" placeholder="Add Name">
    <!-- Champ de saisie pour l'email du client (doit être type="email") -->
    <input type="test" name="email" placeholder="Add Email">
    <!-- Bouton pour soumettre le formulaire -->
    <input type="submit" value="Submit">
</form>

<!-- Liste des clients existants -->
<h3>Customers</h3>
<ul>
    <!-- Début de la boucle pour chaque client (correction nécessaire dans la syntaxe) -->
    <% customers.forEach(function(customer) { %>
    <li>
        <!-- Affiche le nom du client -->
        <h4><%= customer.value.name %></h4>
        <!-- Affiche l'email du client -->
        <%= customer.value.email %>
        <!-- Formulaire pour supprimer un client spécifique -->
        <form method="post" action="/customer/delete/<%= customer.key %>">
            <!-- Champ caché contenant la révision du document (pour la gestion des conflits) -->
            <input type="hidden" value="<%= customer.value.rev %>" name="rev">
            <!-- Bouton pour soumettre le formulaire de suppression -->
            <input type="submit" value="X">
        </form>
    </li>
    <% }); %>
</ul>
```

Ce code permet de créer une interface utilisateur pour une fonctionnalité de gestion de clients, comprenant l'ajout et la suppression de clients.

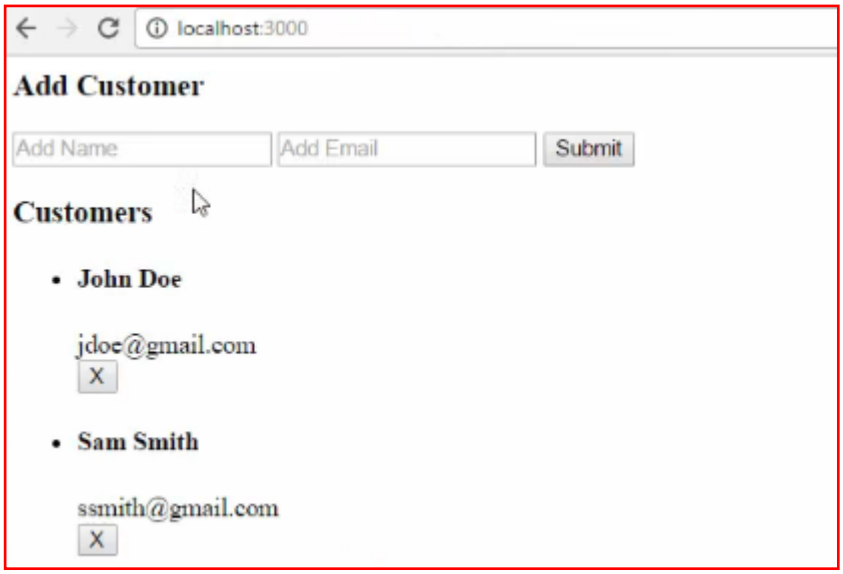
Voici notre app.js :

```
// Importation des modules nécessaires
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const NodeCouchDb = require('node-couchdb');

// Configuration de l'instance CouchDB avec authentification
const couch = new NodeCouchDb({
    auth: {
        user: 'user',
        password: 'azerty'
    }
});

// Nom de la base de données et URL de la vue CouchDB
const dbName = 'customers';
const viewUrl = '_design/all_customers/_view/all';

// Affichage des bases de données CouchDB disponibles
couch.listDatabases().then(function(dbs) {
    console.log(dbs);
});

// Initialisation de l'application Express
const app = express();

// Configuration du moteur de template EJS et du répertoire des vues
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Utilisation de bodyParser pour parser les corps des requêtes JSON et URL-encoded
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: false}));
```

```
// Route GET racine pour afficher la liste des clients
app.get('/', function(req, res) {
    couch.get(dbName, viewUrl).then(
        function(data, headers, status) {
            console.log(data.data.row); // Devrait être data.data.rows pour correspondre à la structure de données
            // Rendu de la vue avec la liste des clients
            res.render('index', {
                customers: data.data.rows
            });
        },
        function(err) {
            // Gestion des erreurs
            console.error('Error:', err);
            res.status(500).send('Internal Server Error');
        }
    );
});

// Route POST pour ajouter un client
app.post('/customer/add', function(req, res) {
    const name = req.body.name;
    const email = req.body.email;

    // Génération d'un ID unique pour le nouveau client
    couch.uniqid().then(function(ids) {
        const id = ids[0]; // Correction pour récupérer le premier ID généré
        // Insertion du nouveau client dans CouchDB
        couch.insert(dbName, { // Correction pour utiliser la variable dbName
            _id: id,
            name: name,
            email: email
        }).then(
            function() {
                // Redirection vers la route racine après ajout
                res.redirect('/');
            },
            function(err) {
                // Gestion des erreurs lors de l'insertion
                res.send(err);
            }
        );
    });
});
```

```
// Route POST pour supprimer un client
app.post('/customer/delete/:id', function(req, res) {
    const id = req.params.id;
    const rev = req.body.rev;

    // Suppression du client dans CouchDB
    couch.del(dbName, id, rev).then(
        function(data, headers, status) {
            // Redirection vers la route racine après suppression
            res.redirect('/');
        },
        function(err) {
            // Gestion des erreurs lors de la suppression
            res.send(err);
        }
    );
});

// Démarrage du serveur sur le port 3000
app.listen(3000, function() {
    console.log('Server Started On Port 3000');
});
```

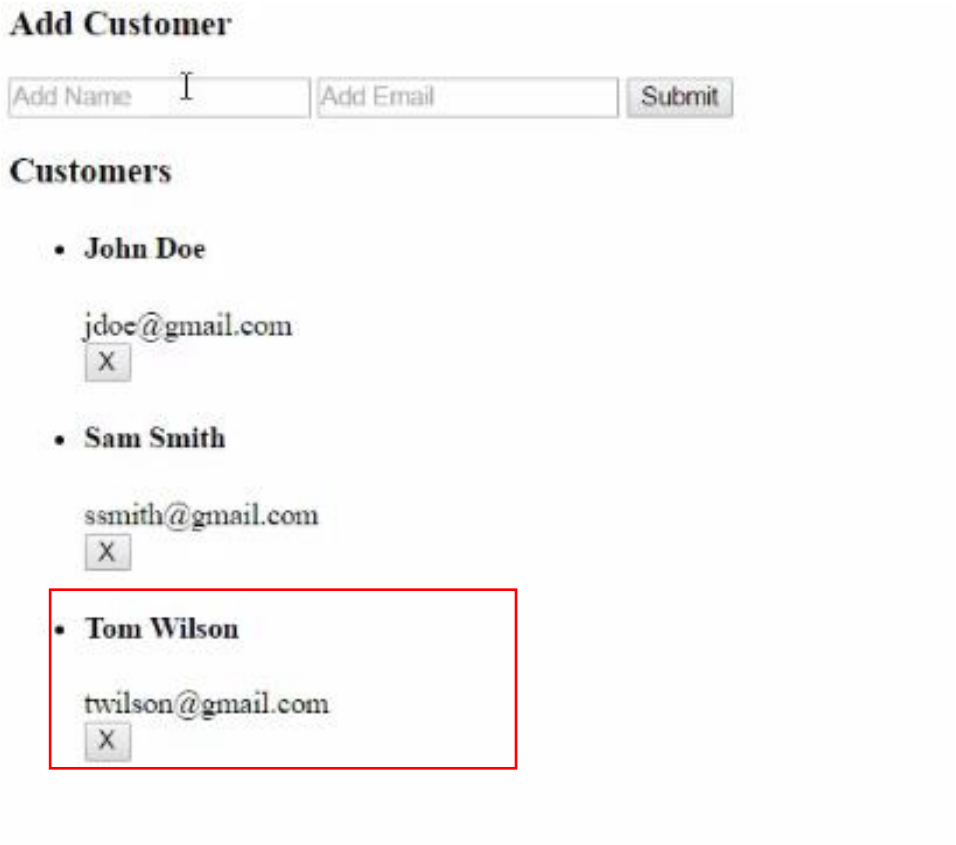
Ce code nous permet d'interagir avec notre base de données. Maintenant, si nous relançons notre application avec node app, nous obtenons un exemple d'application web qui interagit avec notre base de données CouchDB : customers.



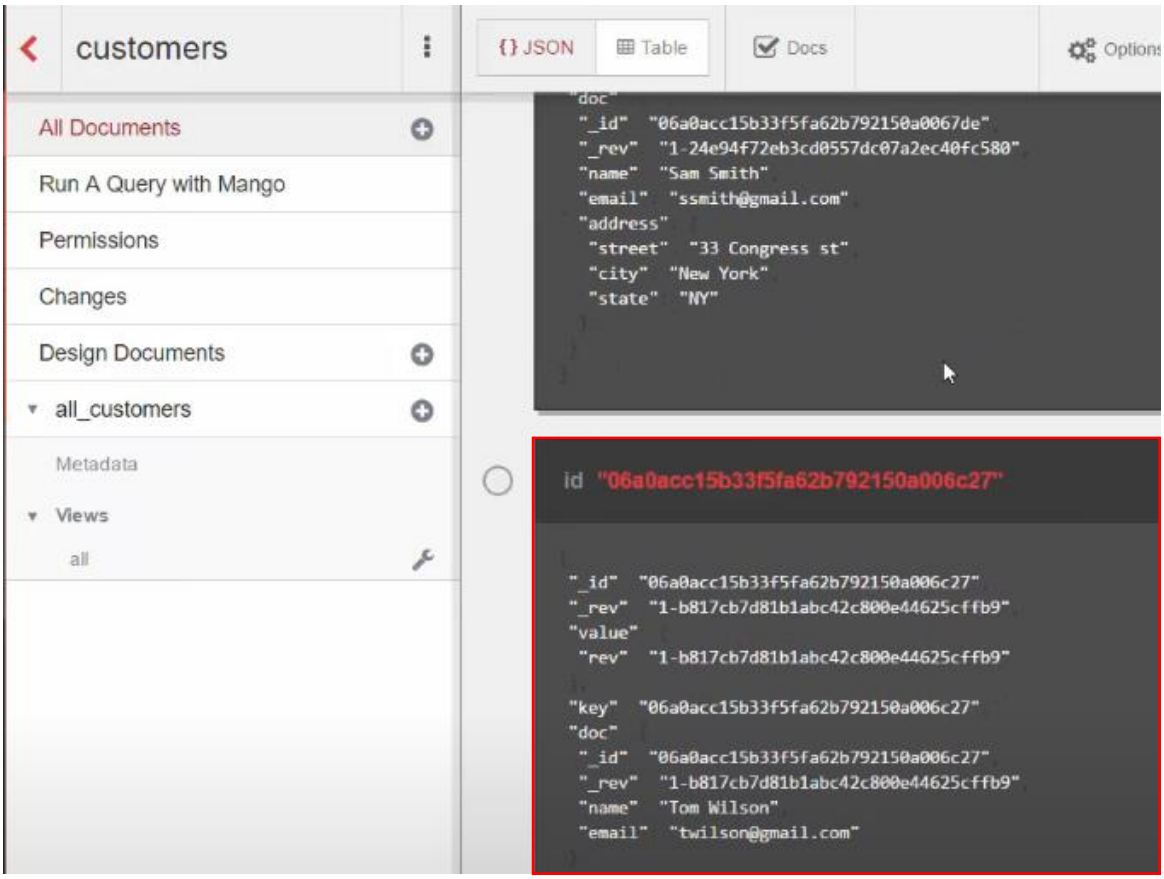
Grâce à ce formulaire, nous pouvons ajouter de nouveau customers :

Une fois ajouté, nous pouvons retrouver le nouveau customer ajouté sur notre application web mais aussi dans notre base de données couchDB :

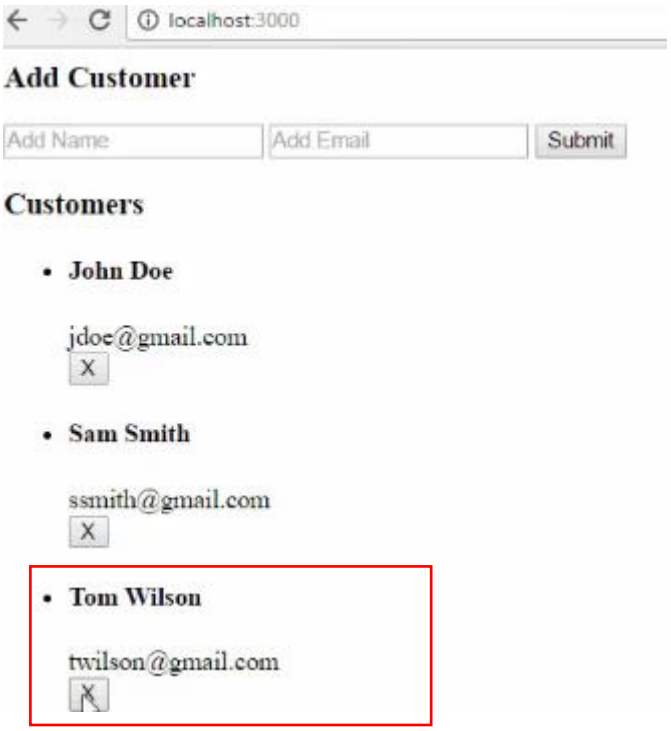
Application web :



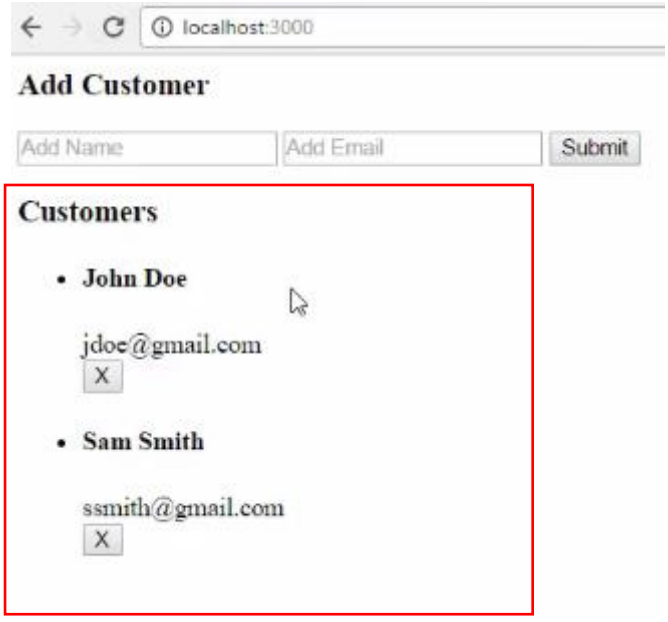
CouchDB :



Nous avons aussi le choix de supprimer les customers de notre application et base de données en cliquant simplement sur la croix :



Une fois cliqué, notre customer est supprimer de notre application web se qui veux dire que son document associé dans couchDB à bien été supprimé :



Conclusion

CouchDB se présente comme une avancée notable dans le monde des bases de données NoSQL, se distinguant grâce à sa flexibilité, sa capacité à monter en charge et sa robustesse. En tant que système de gestion de base de données orienté document, CouchDB offre une gestion des données plus intuitive et adaptable, comparée aux systèmes traditionnels de bases de données relationnelles. Sa facilité de réplication et de synchronisation en fait un choix idéal pour les applications web modernes et les configurations distribuées.

Lorsqu'on le compare à MongoDB, un autre système de base de données orienté document, CouchDB se démarque par sa capacité de réplication native et sa facilité de synchronisation. Bien que MongoDB soit souvent privilégié pour sa performance en écriture et ses fonctionnalités de requête avancées, CouchDB excelle dans la gestion de la concurrence et montre une plus grande robustesse dans les environnements distribués.

Face à Cassandra, reconnue pour sa haute disponibilité et sa capacité à gérer d'importants volumes de données, CouchDB se distingue par sa simplicité d'utilisation et son modèle de données plus flexible. Cassandra est plus adaptée aux applications nécessitant une grande scalabilité et une écriture rapide, tandis que CouchDB convient mieux aux applications requérant une forte cohérence des données et une réplication aisée.

Redis, connu comme un magasin de structure de données en mémoire offrant une vitesse de lecture et d'écriture extrêmement rapide, est idéal pour des cas d'utilisation tels que la mise en cache ou les files d'attente de messages. CouchDB, avec son stockage sur disque et sa nature orientée document, est plus adapté aux applications nécessitant une persistance durable des données et une structure complexe.

En comparaison avec Neo4J, une base de données orientée graphe, CouchDB propose une approche différente dans la modélisation des données. Alors que Neo4J excelle dans la représentation des relations complexes entre les données, CouchDB est plus adapté aux applications où les données sont structurées de manière moins interconnectée et où la réplication et la synchronisation sont cruciales.

CouchDB, grâce à son modèle de données JSON et à ses vues basées sur JavaScript, offre une grande souplesse dans la gestion et l'interrogation des données. Sa tolérance aux pannes et son modèle de cohérence éventuelle assurent la disponibilité et la résilience des données, même dans des conditions difficiles. L'interface RESTful de CouchDB facilite aussi l'intégration avec d'autres applications et services, augmentant ainsi son adaptabilité.

Bien que CouchDB ait des limites, notamment en matière de complexité des requêtes et de gestion des transactions de grand volume, elle demeure un choix robuste et fiable pour les développeurs et les entreprises en quête d'une solution efficace pour la gestion de données distribuées et décentralisées.

Ainsi, CouchDB est une solution de base de données particulièrement adaptée à l'ère numérique, offrant une combinaison unique de flexibilité, de robustesse et d'efficacité pour le développement d'applications web et mobiles modernes, se positionnant avantageusement face à des alternatives telles que MongoDB, Cassandra, Redis et Neo4J dans divers scénarios d'utilisation.