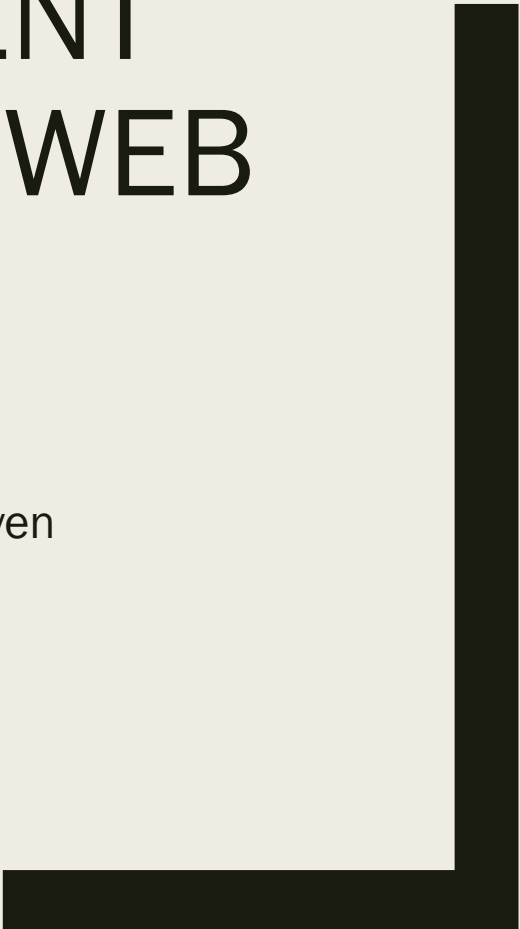




DÉVELOPPEMENT D'APPLICATIONS WEB

COURS 10: MONGODB

Enseignante: NGUYEN Thi Minh Tuyen



Plan du cours

1. Asynchronous
2. MongoDB
3. NodeJS et MongoDB
4. Mongoose

Plan du cours

1. Asynchronous
2. MongoDB
3. NodeJS et MongoDB
4. Mongoose

asynchronous

- **Problème:** Quand deux ou plusieurs événements peuvent se produire à des moments imprévisibles, et ces événements dépendent l'un de l'autre d'une manière ou d'une autre.
- **Solutions possible:**
 - *Désactiver les boutons jusqu'au chargement du JSON*
 - *OU: ne pas afficher les boutons avant le chargement du JSON*
 - *OU: ne pas afficher l'interface utilisateur du tout avant la fin du JSON*
- **Autre solution:** asynchronous



async/await

```
async function loadJson() {  
    const response = await fetch('albums.json');  
    const json = await response.json();  
    console.log(json);  
}  
loadJson();
```

Fonction async [1]

Une fonction marquée `async` a les qualités suivantes:

- Il se comportera plus ou moins comme une fonction normale si vous ne mettez pas l'expression `await` dedans.
- Une expression `await` est de forme:
`wait promise`

Fonction async [2]

Une fonction marquée `async` a les qualités suivantes:

- S'il y a une expression `await`, l'exécution de la fonction mettra en pause jusqu'à `Promise` dans l'expression `await` est résolue.
- **Remarque:** le navigateur n'est pas bloqué; il continuera à exécuter JavaScript lorsque la fonction asynchrone est suspendue.
- - Ensuite, lorsque `Promise` est résolue, l'exécution de la fonction continue.
- - L'expression `wait` correspond à la valeur résolue de `Promise`.

Exemple

```
function onJsonReady(json) {  
    console.log(json);  
}  
function onResponse(response) {  
    return response.json();  
}  
fetch('albums.json')  
    .then(onResponse)  
    .then(onJsonReady);  
  
async function loadJson() {  
    const response = await fetch('albums.json');  
    const json = await response.json();  
    console.log(json);  
}  
loadJson();
```


Plan du cours

1. Asynchronous
2. **MongoDB**
3. NodeJS et MongoDB
4. Mongoose

Définitions de base de données

Une base de données (DB) est une collection structurée de données.

- Le fichier JSON peut être considéré comme une base de données.
- Un système de gestion de base de données (SGBD) est un logiciel qui gère le stockage, la récupération et la mise à jour des données.
 - *Exemples: MongoDB, MySQL, PostgreSQL, etc.*
 - *Habituellement, quand les gens disent "base de données", ils veulent dire données qui est géré via un SGBD.*

Pourquoi utiliser une base de données/SGBD? [1]

Pourquoi utiliser un SGBD au lieu d'enregistrer dans un fichier JSON?

- **Rapide:** permet de rechercher/filtrer une base de données rapidement par rapport à un fichier
- **Evolutif:** peut gérer de très grandes tailles de données
- **Fiable:** mécanismes en place pour des transactions sécurisées, des sauvegardes, etc.
- **Fonctionnalités intégrées:** peut rechercher, filtrer des données, combiner des données de plusieurs sources
- **Abstract:** fournit une couche d'abstraction entre les données stockées et applications)
- Peut changer l'emplacement et la façon dont les données sont stockées sans avoir besoin pour changer le code qui se connecte à la base de données.

Pourquoi utiliser une base de données / SGBD? [2]

Pourquoi utiliser un SGBD au lieu d'enregistrer dans un fichier JSON?

- Aussi: certains services comme Heroku ne sauvegarderont pas les fichiers de façon permanente, donc utiliser `fs` ou `fs-extra` ne fonctionnera pas

NoSQL – MongoDB

NoSQL

- L'utilisation de bases de données SQL fournit un stockage fiable pour les premières applications Web
- Dirigé vers de nouvelles bases de données correspondant au modèle d'objet d'application Web
 - *Connues collectivement sous le nom de bases de données NoSQL*

MongoDB - Base de données NoSQL la plus importante

- Modèle de données: stocke les collections contenant des documents (objets JSON)
- A un langage de requête expressif
- Peut utiliser des indices pour des recherches rapides
- Essaie de gérer l'évolutivité, la fiabilité, etc.

MongoDB [1]

MongoDB: un SGBD open source populaire

- une base de données orientée document par opposition à une base de données relationnelle

Base de données relationnelle:

Name	School	Employer	Occupation
Lori	null	Self	Entrepreneur
Malia	Harvard	null	null

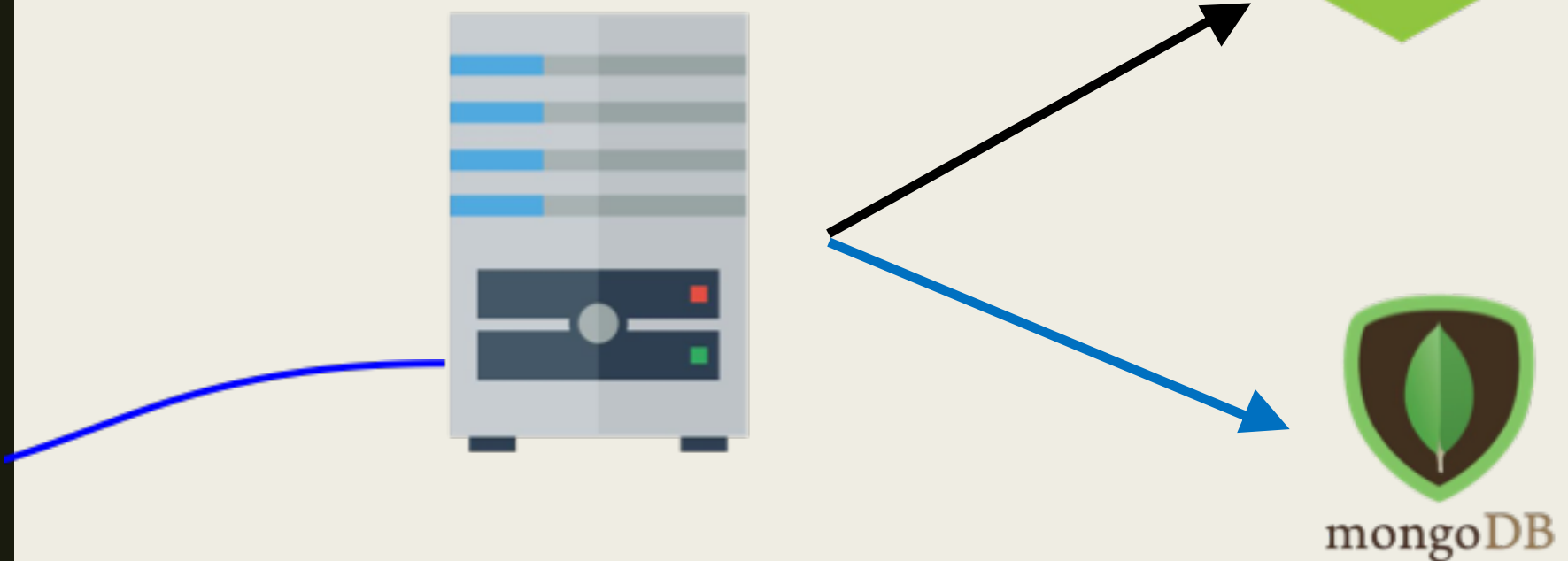
Base de données orientée document:

```
{  
  name: "Lori",  
  employer: "Self",  
  occupation: "Entrepreneur"  
}  
{  
  name: "Malia",  
  school: "Harvard"  
}
```

Les bases de données relationnelles ont des schémas fixes; les bases de données orientées document ont des schémas flexibles

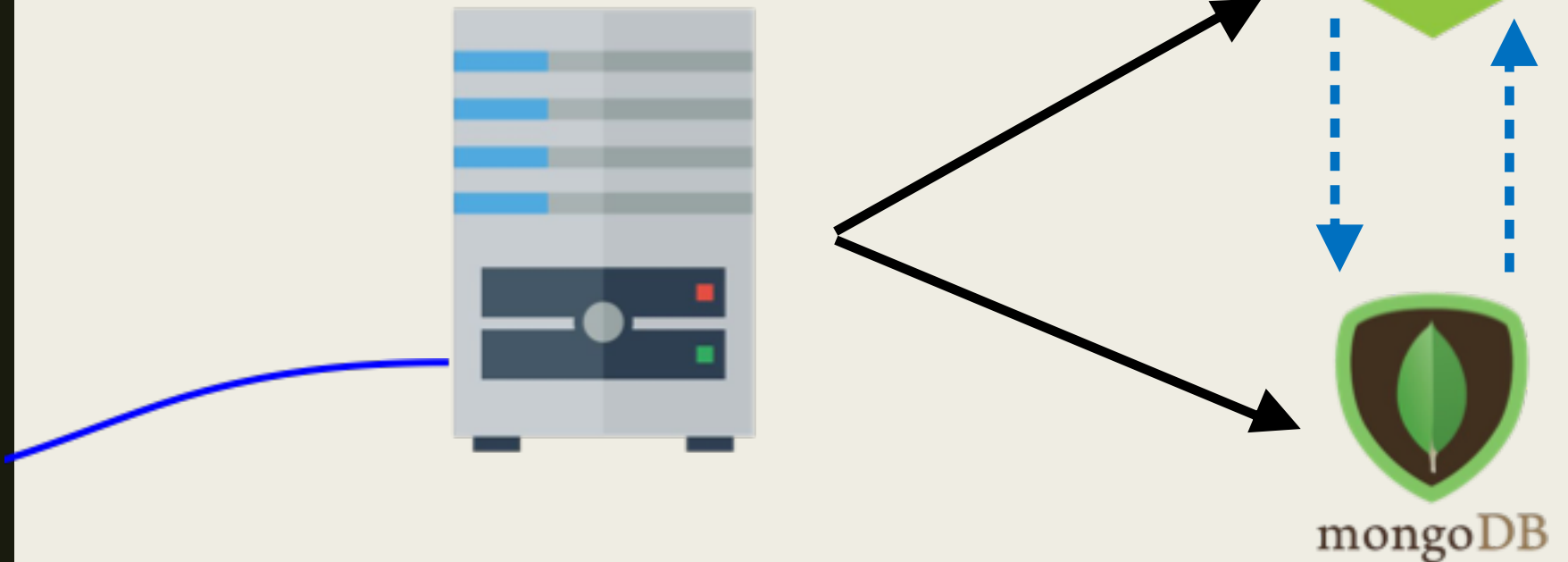
MongoDB [2]

- MongoDB est un autre logiciel exécuté sur l'ordinateur, à côté de notre programme de serveur NodeJS.
- Il est également appelé **serveur MongoDB**.



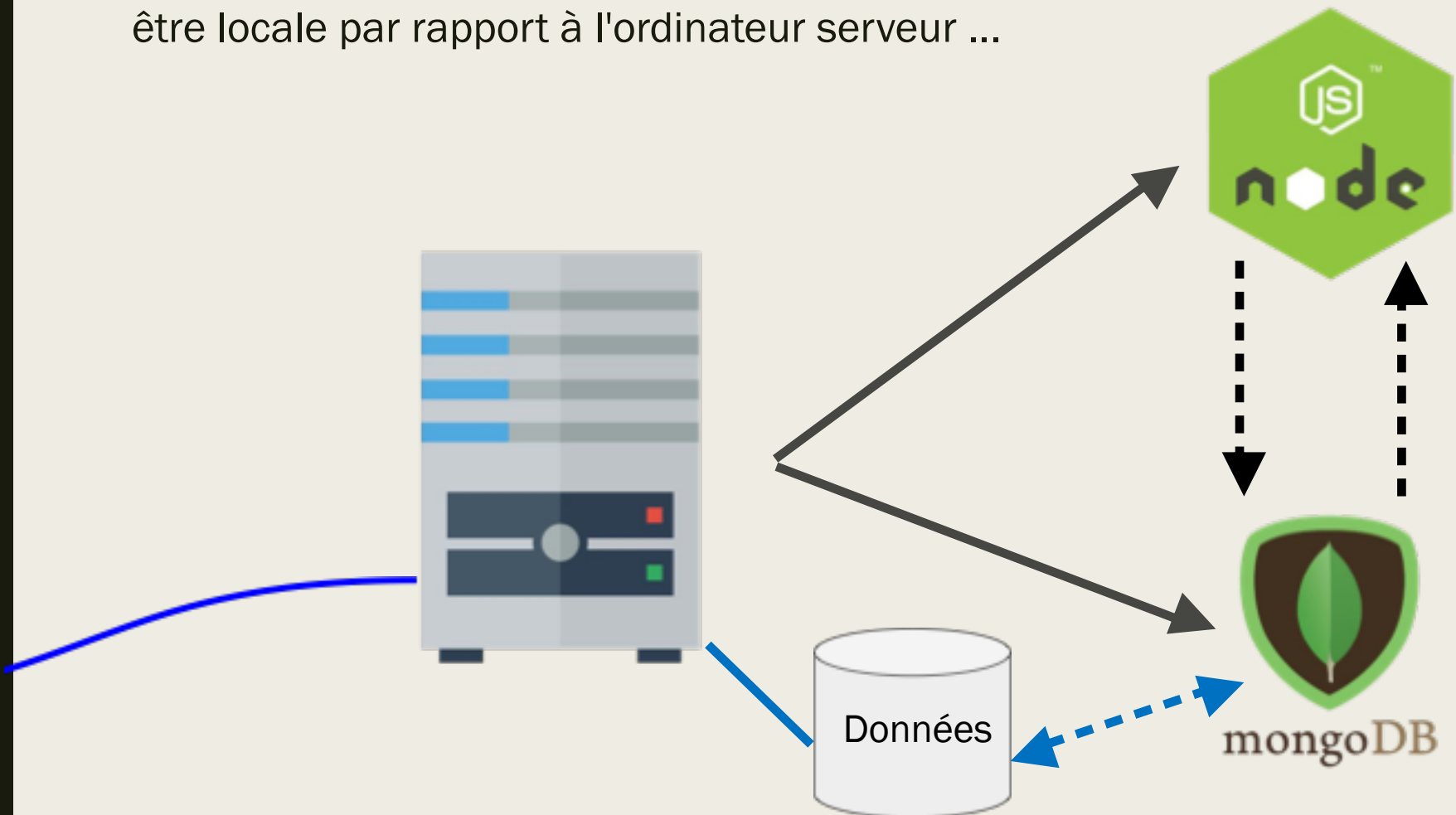
MongoDB [3]

- Il existe des bibliothèques MongoDB que nous pouvons utiliser dans NodeJS pour communiquer avec le serveur MongoDB, qui lit et écrit des données dans la base de données qu'il gère.



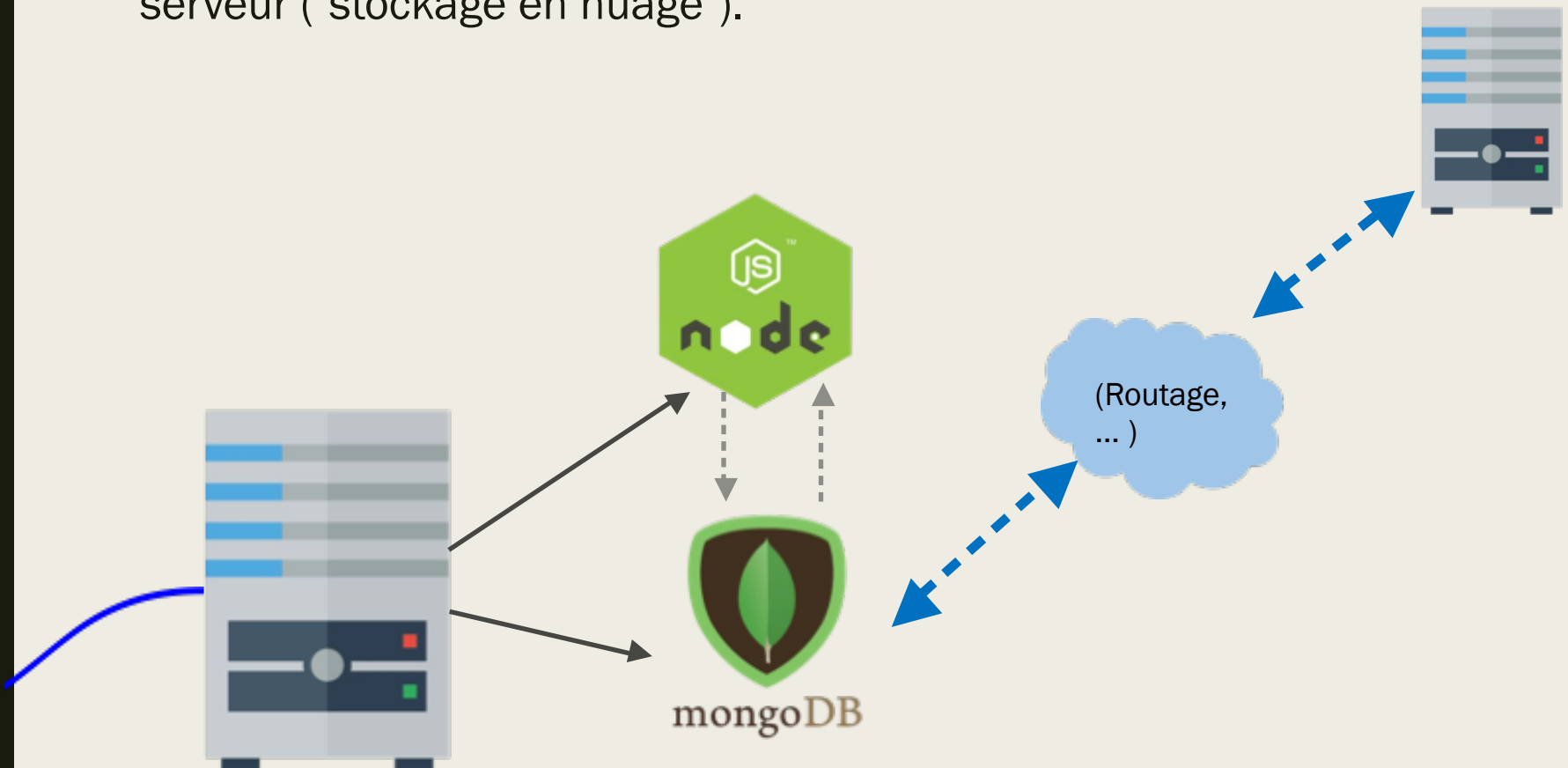
MongoDB [4]

- La base de données gérée par MongoDB Server peut être locale par rapport à l'ordinateur serveur ...

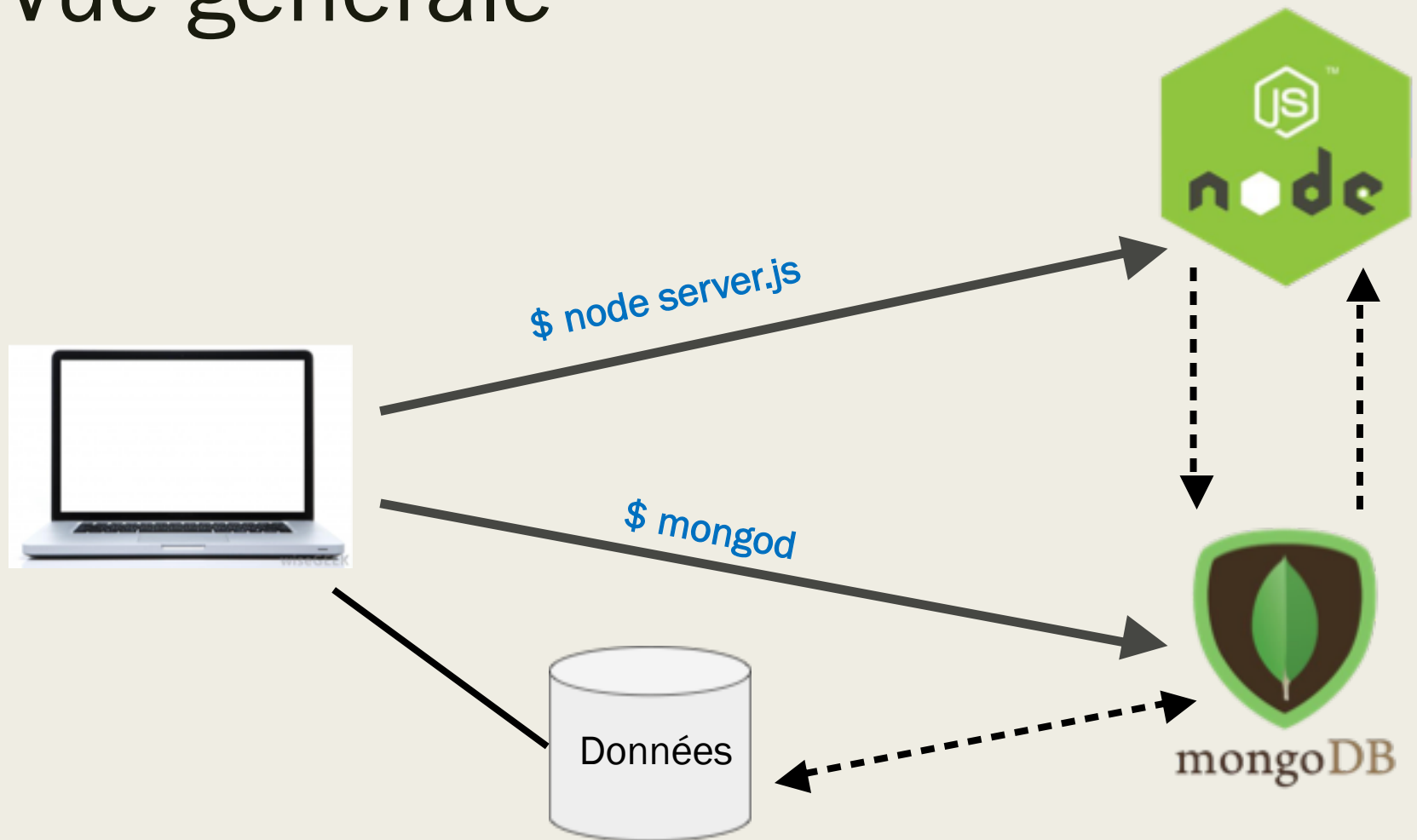


MongoDB [5]

- Ou il pourrait être stocké sur un autre ordinateur serveur ("stockage en nuage").



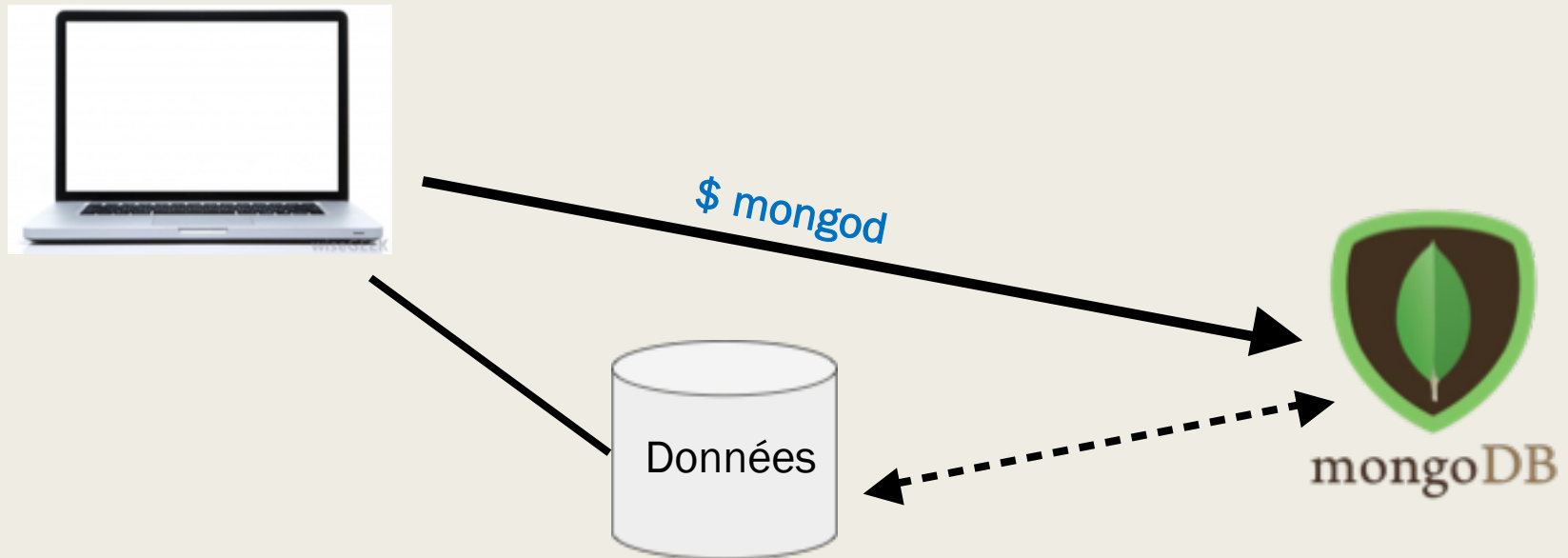
Vue générale



Pour le développement, nous aurons 2 processus en cours d'exécution:

- Node exécutera le programme du serveur principal sur le port 3000
- Mongod exécutera le serveur de base de données sur un port 27017

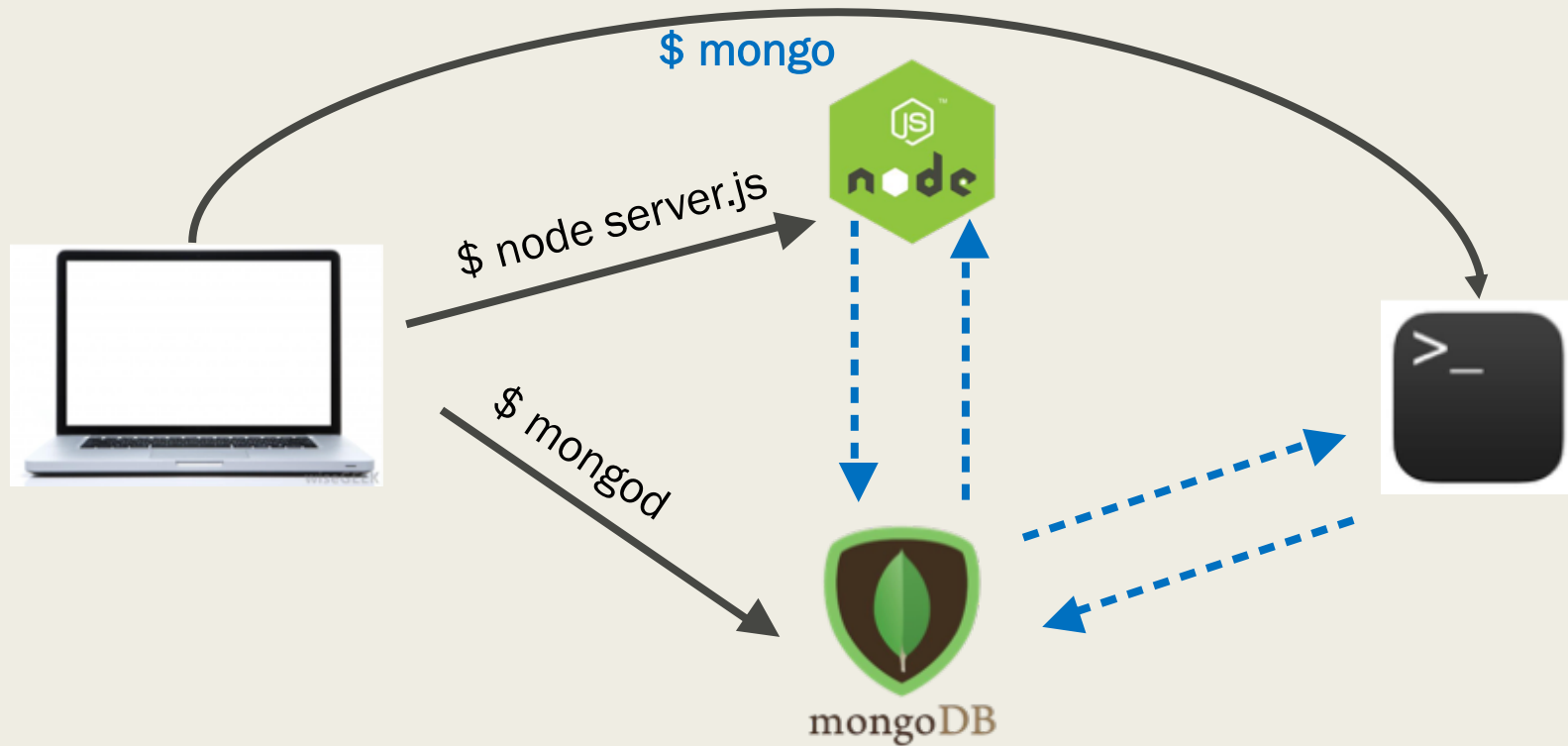
Vue générale [1]



Le serveur Mongod sera lié au port 27017 par défaut

- Le processus Mongod écoute les messages envoyés à manipuler la base de données: insérer, rechercher, supprimer, etc.

Vue générale [3]



Deux manières de communiquer avec le serveur MongoDB:

- Bibliothèques NodeJS
- Outil de ligne de commande mongo

Concepts MongoDB

Base de données:

- Un conteneur de collections MongoDB

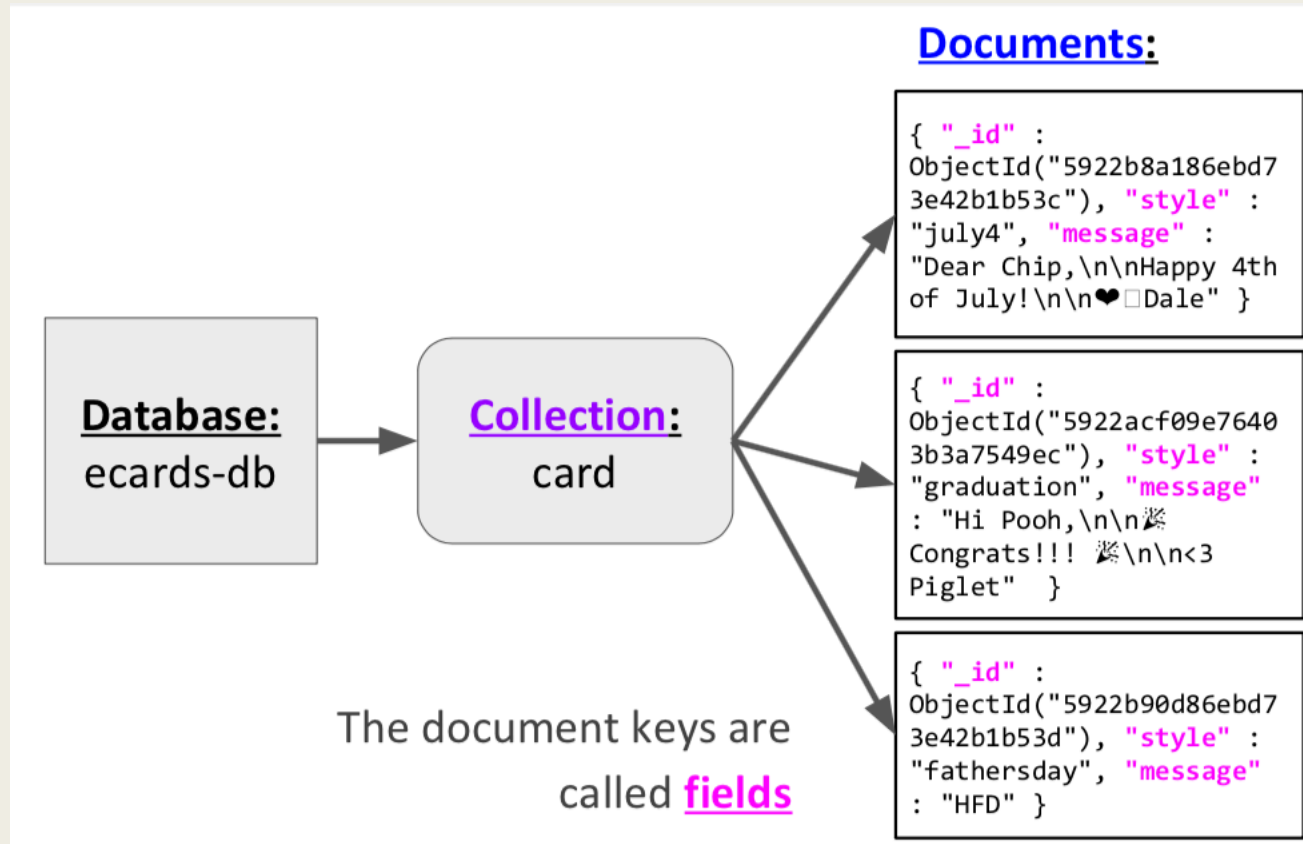
Collection:

- Un groupe de documents MongoDB.
- (**Table** dans une base de données relationnelle)

Document:

- Un objet de type JSON qui représente une instance d'une collection (**Row** dans une base de données relationnelle)
- Également utilisé plus généralement pour désigner tout ensemble de paires clé-valeur.

Exemple MongoDB



mongod: processus de base de données

- Lorsque vous installez MongoDB, il est fourni avec le programme de ligne de commande `mongod`. Cela lance le processus de gestion de la base de données MongoDB et le lie au port 27017:

```
$ mongod
```



`$ mongod`



mongo: interface de ligne de commande

- Vous pouvez vous connecter au serveur MongoDB via le shell mongo:

```
$ mongo
```



\$ mongod



Commandes de shell mongo [1]

> show dbs

- Affiche les bases de données sur le serveur MongoDB

> use databaseName

- Change la base de données actuelle vers databaseName
- databaseName ne doit pas déjà exister
- Il sera créé pour la première fois que vous écrivez des données dans databaseName

> show collections

- Affiche les collections pour la base de données actuelle

Commandes de shell mongo [2]

> `db.collection`

- Variable référençant à la collection `collection`

> `db.collection.find(query)`

- Imprime les résultats de `collection` correspondant à la requête
- La requête est un document MongoDB (c'est-à-dire un objet JSON)
 - *Pour obtenir tous ce qui dans la collection, utilisez `db.collection.find()`*
 - *Pour obtenir tout ce qui correspond à `x = foo` dans la collection, utilisez `db.collection.find ({x: 'foo'})`*

Commandes de shell mongo [3]

> `db.collection.findOne(query)`

- Imprime le premier résultat de `collection` correspondant à la requête

> `db.collection.insertOne(document)`

- Ajoute `document` à la collection `collection`.
- `document` peut avoir n'importe quelle structure.

> `db.test.insertOne({ name: 'dan' })`

> `db.test.find()`

```
{ "_id" : ObjectId("5922c0463fa5b27818795950"), "name" :  
  "dan" }
```

- MongoDB ajoutera automatiquement un `_id` unique à chaque document d'une collection.

Commandes de shell mongo [4]

> `db.collection.deleteOne(query)`

- Supprime le premier résultat de `collection` correspondant à la requête

> `db.collection.deleteMany(query)`

- Supprimer plusieurs documents de `collection`.
- Pour supprimer tous les documents, utilisez `db.collection.deleteMany()`

> `db.collection.drop()`

- Supprime `collection` de la base de données

mongo shell

Quand devriez-vous utiliser le shell de mongo?

- Ajout de données de test
- Suppression des données de test

Plan du cours

1. Asynchronous
2. MongoDB
3. **NodeJS et MongoDB**
4. Mongoose

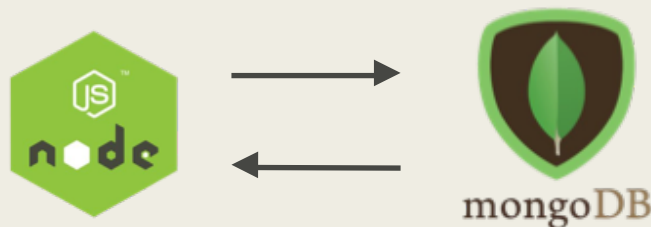
Pilote NodeJS

- Pour lire et écrire dans la base de données MongoDB à partir de Node, nous utiliserons la bibliothèque `mongodb`.

- Installer via npm:

```
$ npm install --save mongodb
```

- Sur le site Web de MongoDB, cette bibliothèque s'appelle [Pilote MongoDB NodeJS](#).



Objets mongodb

La bibliothèque de Node mongodb fournit des objets permettant de manipuler la base de données, les collections et les documents:

- **Db**: base de données; peut obtenir des collections en utilisant cet objet
- **Collection**: peut obtenir/insérer/supprimer des documents de cette collection via des appels tels que `insertOne`, `find`, etc.
- Les documents ne sont pas des classes spéciales; ce ne sont que des objets JavaScript.

Obtenir un objet db

- Vous pouvez obtenir une référence à l'objet de base de données à l'aide de la fonction `MongoClient.connect (url, callback)`:
 - *url* est la chaîne de connexion du serveur MongoDB.
 - *callback* est la fonction appelée lors de la connexion
 - **database** parameter: l'objet db

- Example:

```
const MongoClient = require('mongodb').MongoClient;
const DATABASE_NAME = 'engine-dict';
const MONGO_URL =
    'mongodb://localhost:27017/${DATABASE_NAME}';
let db = null;
MongoClient.connect(MONGO_URL, function(err, database){
    db=database;
});
```

Chaîne de connexion

```
const DATABASE_NAME = 'engine-dict';  
const MONGO_URL =  
    'mongodb://localhost:27017/${DATABASE_NAME}';
```

- L'URL est celle d'un serveur MongoDB: elle commence par `mongodb://` et pas `http://`
- Le serveur MongoDB s'exécute sur notre machine locale: utilise `localhost`.
- La fin de la chaîne de connexion spécifie le nom de la base de données que nous utilisons.
- Si une base de données de ce nom n'existe pas déjà, elle sera créée pour la première fois ce que nous lui écrivons.

Callbacks et Promises [1]

Chaque méthode MongoDB asynchrone a deux versions:

- Callback
- Promise

La version de callback de `MongoClient.connect` :

```
let db = null;
MongoClient.connect(MONGO_URL, function(err, database){
    db=database;
});
```

Callbacks et Promises [2]

■ La version Promise :

```
let db = null;  
function onConnected(err, database){  
    db=database;  
}
```

```
MongoClient.connect(MONGO_URL).then(onConnected);
```

Callbacks et Promises [3]

- La version Promise + async/await:

```
let db = null;
async function main(){
    const client = await MongoClient.connect(MONGO_URL);
    db = await client.db(DATABASE_NAME);
}
main();
```

Utiliser une collection

```
const coll = db.collection (collectionName);
```

- Obtient l'objet de collection nommé `collectionName` et le stocke dans `coll`
- Il n'est pas nécessaire de créer la collection avant de l'utiliser
- Il sera créé pour la première fois que nous lui écrivons - Cette fonction est synchrone

```
async function main(){  
  const client = await MongoClient.connect(MONGO_URL);  
  db = await client.db(DATABASE_NAME);  
  collection = await db.collection('words');  
}
```

`collection.insertOne(doc, Callback)`

- Ajoute un élément à `collection`
- `doc` est un objet JavaScript représentant les paires clé-valeur à ajouter à la collection.
- Le callback se déclenche lorsqu'il a fini d'insérer
 - *Le premier paramètre est un objet d'erreur*
 - *Le deuxième paramètre est un objet de résultat, où `result.insertedId` contiendra l'identifiant de l'objet créé.*

Version callback

```
async function insertWord(word, definition){
  const doc = {
    word:word,
    definition:definition
  };
  collection.insertOne(doc, function(err, result){
    console.log('Document id: ${result.insertId}');
  });
}
```

Eviter d'utiliser celui-ci dans votre projet

collection.insertOne(Promise)

```
const result = await collection.insertOne (doc);
```

- Ajoute un élément à la collection
- `doc` est un objet JavaScript représentant les paires clé-valeur à ajouter à la collection.
- Retourne une promesse qui se résout en un objet de résultat lorsque l'insertion est terminée
 - *`result.insertedId` contiendra l'id de l'objet créé*

Version promise

```
async function insertWordAsync(word, definition){  
  const doc = {  
    word:word,  
    definition:definition  
  };  
  const result = await collection.insertOne(doc);  
}
```

- Nous utiliserons les versions Promise + async/wait de toutes les fonctions asynchrones de MongoDB.

collection.findOne

```
const doc = await collection.findOne(query);
```

- Trouve le premier élément de la collection qui correspond à `query`
- `query` est un objet JS représentant les champs à rechercher
- Renvoie une promesse qui se résout en un objet de document lorsque `findOne` est terminée
 - *`doc` sera l'objet JS, vous pourrez donc accéder à un champ via `doc.fieldName`. Par exemple: `doc._id`*
 - *Si rien n'est trouvé, `doc` sera `null`*

collection.findOne

```
async function printWord(word){
  const query = {
    word: word
  };
  const response = await collection.findOne(query);
  console.log(
    `Word: ${response.word},
    definition: ${response.definition}`
  );
}
```

collection.find()

```
const cursor = await collection.find(query);
```

- Renvoie un Cursor indiquant la première entrée d'un ensemble de documents correspondant à query
- Vous pouvez utiliser hasNext et à côté d'itérer dans la liste:

```
async function printAllWordsCursor(){  
  const cursor = await collection.find();  
  while (await cursor.hasNext()){  
    const result = await cursor.next();  
    console.log(`Word: ${result.word}  
                , definition:${result.definition}`);  
  }  
}
```

collection.find().toArray()

```
const cursor = wait collection.find(query);  
const liste = wait cursor.toArray();
```

- Cursor a également une fonction toArray() qui convertit les résultats en tableau.

```
async function printAllWords(){  
    const results = await collection.find().toArray();  
    for (const result of results){  
        console.log(`Word: ${result.word},  
                    definition: ${result.definition}`);  
    }  
}
```

collection.update

```
    await collection.update(query, newEntry);
```

- Remplace query par newEntry

```
async function updateWord(word, definition){  
    const query = {  
        word:word  
    };  
    const newEntry = {  
        word:word,  
        definition: definition  
    };  
    const response = await collection.update(query,  
                                              newEntry);  
}
```


"Upsert" avec `collection.update`

MongoDB prend également en charge "upsert", qui est

- Mettre à jour l'entrée si elle existe déjà
- Insérer l'entrée si elle n'existe pas déjà

```
const params = { upsert: true };  
await collection.update(query, newEntry, params);
```

Example

```
async function upsertWord(word, definition){
  const query = {
    word:word
  };
  const newEntry = {
    word:word,
    definition: definition
  };
  const params = {
    upsert: true
  };
  const response = await collection.update(query,
                                           newEntry, params);
}
```

collection.deleteOne/Many

```
const result = await collection.deleteOne(query);
```

- Supprime la première query correspondant à l'élément
- `result.deletedCount` donne le nombre de documents supprimés

```
const result = await collection.deleteMany(query);
```

- Supprime tous les éléments correspondant à query
- `result.deletedCount` donne le nombre de documents supprimés
- Utilisez `collection.deleteMany()` pour tout supprimer

collection.deleteOne

```
async function deleteWord(word){  
  const query = {  
    word: word  
  };  
  const response = await collection.deleteOne(query);  
  console.log(`Number deleted:  
  ${response.deletedCount}`);  
}
```

collection.deleteMany

```
async function deleteAllWords(){  
  const response = await collection.deleteMany();  
  console.log(`Number deleted:  
              ${response.deletedCount}`);  
}
```

Requêtes avancées

Pour des requêtes plus complexes, consultez:

- [Requêtes](#)

- *Sélecteurs de requêtes et opérateurs de projection*

```
db.collection('inventory').find({ qty: { $lt: 30 } });
```

- [Mise à jours](#)

- *Opérateurs de mise à jour*

```
db.collection('words').updateOne(  
  { word: searchWord },  
  { $set: { definition: newDefinition } })
```

Plan du cours

1. Asynchronous
2. MongoDB
3. NodeJS et MongoDB
4. **Mongoose**

Application du schéma

- Les blobs JSON fournit une grande flexibilité mais ne correspondent pas toujours à ce que vous voulez.
 - *Considérez: `<h1> Bonjour {{person.informalName}} </ h1>`*
 - *Bon: `typeof person.informalName == 'chaîne'` et longueur `<quelque chose`*
 - *Mauvais: le type est un objet de 1 Go, ou undefined, ou null, ou ...*
- Vouloir appliquer un schéma sur les données - peut être implémenté en tant que validateur sur des opérations de mutation
- Mongoose : langage de définition d'objet (ODL – Object Definition Language)
 - *Prenez un usage familier des ORM et mappez-le sur MongoDB*
 - *Masque efficacement l'interface de niveau inférieur de MongoDB avec quelque chose de plus convivial*

Connexion

```
var mongoose = require('mongoose');
```

1. Se connecter à l'instance MongoDB

```
mongoose.connect('mongodb://localhost/engine-  
dict', { useNewUrlParser: true });
```

2. Attendez la fin de la connexion: Mongoose exporte un EventEmitter

```
mongoose.connection.on('open', function () {  
    console.log('connecter a mongo avec mongodb');  
});  
mongoose.connection.on('error', function (err) {  
    console.log(err);  
});
```

- Peut également écouter pour la connexion, la connexion, la déconnexion, la déconnexion, etc.

Mongoose: le schéma définit des collections

Schéma assigner des noms de propriété et leurs types aux collections

- Chaîne, Nombre, Date, Tampon, Booléen,
- Tableau - par exemple. : [ObjectId]
- ObjectId – Référence à un autre objet
- Mixte – N'importe quoi

```
var userSchema = new mongoose.Schema({  
  first_name: String,  
  last_name: String,  
  emailAddresses: [String],  
  location: String });
```

Le schéma autorise les index et les valeurs par défaut secondaires

- L'indice simple

```
first_name: {type: 'String', index: true}
```

- L'indice avec application unique

```
user_name: {type: 'String', index: {unique: true} }
```

- Défauts

```
date: {type: Date, default: Date.now }
```

Indices secondaires

Performance et compromis d'espace

- Requêtes plus rapides: éliminer les analyses - la base de données ne fait que renvoyer les correspondances à partir de l'index
- Opérations de mutation plus lentes: Ajouter, supprimer, mettre à jour doit mettre à jour les index
- Utilise plus d'espace: le besoin de stocker des index et les index peuvent être plus volumineux que les données elles-mêmes

Quand utiliser ?

- Les requêtes courantes passent beaucoup de temps à analyser
- Besoin d'imposer l'unicité

Mongoose: Construire un modèle à partir d'un schéma

- Un modèle en Mongoose est un constructeur d'objets une collection peut ou non correspondre à un modèle du MVC

```
var User = mongoose.model ('User', userSchema);
```

- Créer des objets à partir d'un modèle

```
User.create({  
  first_name: 'Ian',  
  last_name: 'Malcolm'}, doneCallback  
);
```

```
function doneCallback(err, newUser) {  
  console.log('Created object with ID', newUser._id);  
}
```

Modèle utilisé pour interroger la collection

Renvoyer la collection complète de User

```
User.find(function (err, users) {  
    /*users is an array of objects*/  
});
```

■ Retour d'un objet utilisateur unique pour user_id

```
User.findOne({_id: user_id},  
    function (err, user) { /* ... */ });
```

■ Mise à jour d'un objet utilisateur pour user_id

```
User.findOne({_id: user_id},  
function (err, user) {  
    // Update user object – (Note: Object is "special")  
    user.save();  
});
```

Autres opérations de requête

Mongoose - Générateur de requête

```
var query = User.find({});
```

■ Projections

```
query.select("first_name last_name").exec(doneCallback);
```

■ Tri

```
query.sort("first_name").exec(doneCallback);
```

■ Limites

```
query.limit(50).exec(doneCallback);
```

```
query.sort("-location")  
  .select("first_name")  
  .exec(doneCallback);
```

Supprimer des objets de la collection

- Suppression d'un seul utilisateur avec l'id user_id

```
User.remove({_id: user_id}, function (err) { } );
```

- Supprimer tous les objets User

```
User.remove({}, function (err) { } );
```


Question ?