

# Mongoose

FORMATEUR : MAHDI KELLOUCH

# MONGODB

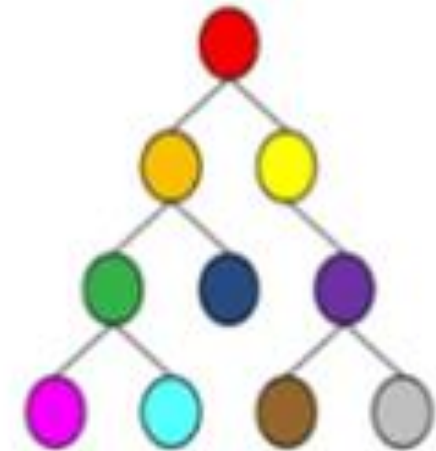
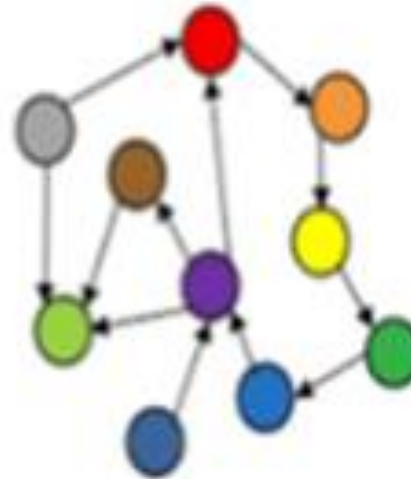
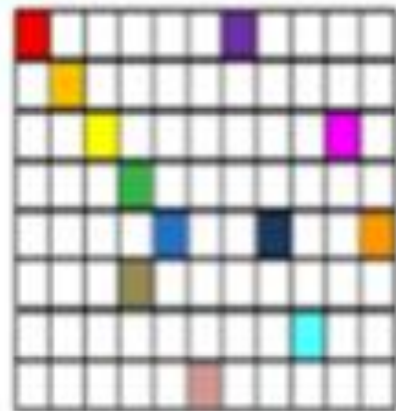
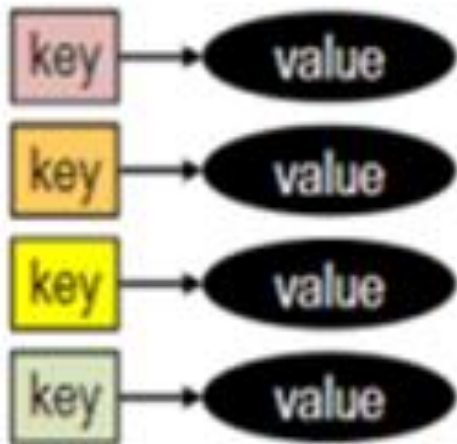
CRÉATION APPLICATION CLOUD NATIVE

# Les base de données NoSQL

- ▶ NoSQL est un système de base de données qui est dit "non relationnel".
- ▶ L'appellation "NoSQL" est une abréviation de "Not Only SQL".
- ▶ Il s'agit d'une base de données non-structurée, c'est-à-dire qu'elle ne suit pas de schéma de construction fixe.
- ▶ Aujourd'hui les bases NoSQL sont majoritairement utilisées par les applications web qui gèrent de grandes quantités de données en temps réel (Big Data).

# Les types de BD noSQL

- ▶ Il existe quatre types de bases de données NoSQL :
  - ▶ Bases de données orientées **Clé-valeur**
  - ▶ Bases de données orientées **Colonnes**
  - ▶ Bases de données orientées **Document**
  - ▶ Bases de données **Graph-Based**



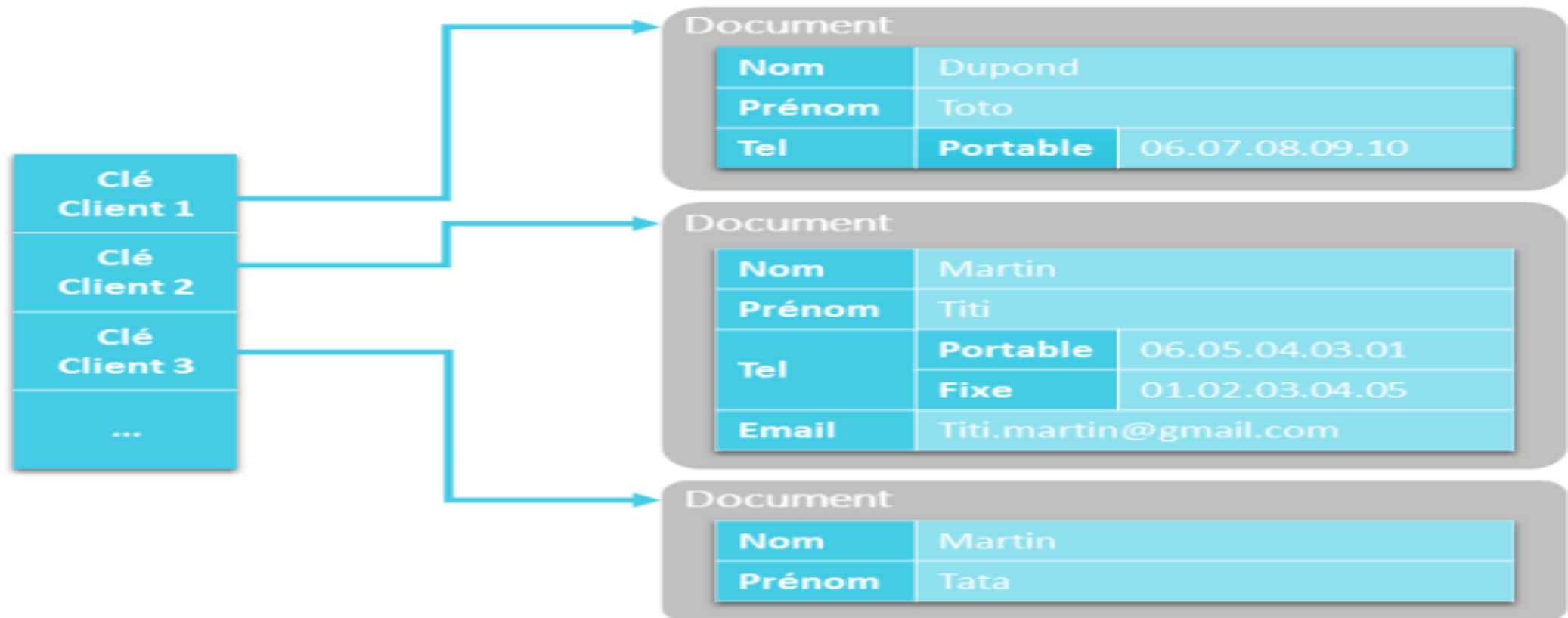
# Les bases NoSQL documents

- ▶ Une base de données NoSQL orienté document stocke et extrait les données sous forme d'une paire clé/valeur.
- ▶ La partie valeur est stockée sous forme de document.
- ▶ Le document est stocké aux formats JSON ou XML.

# Les bases NoSQL documents

Un exemple de stockage de « fiches clients » dans une base NoSQL orientée document

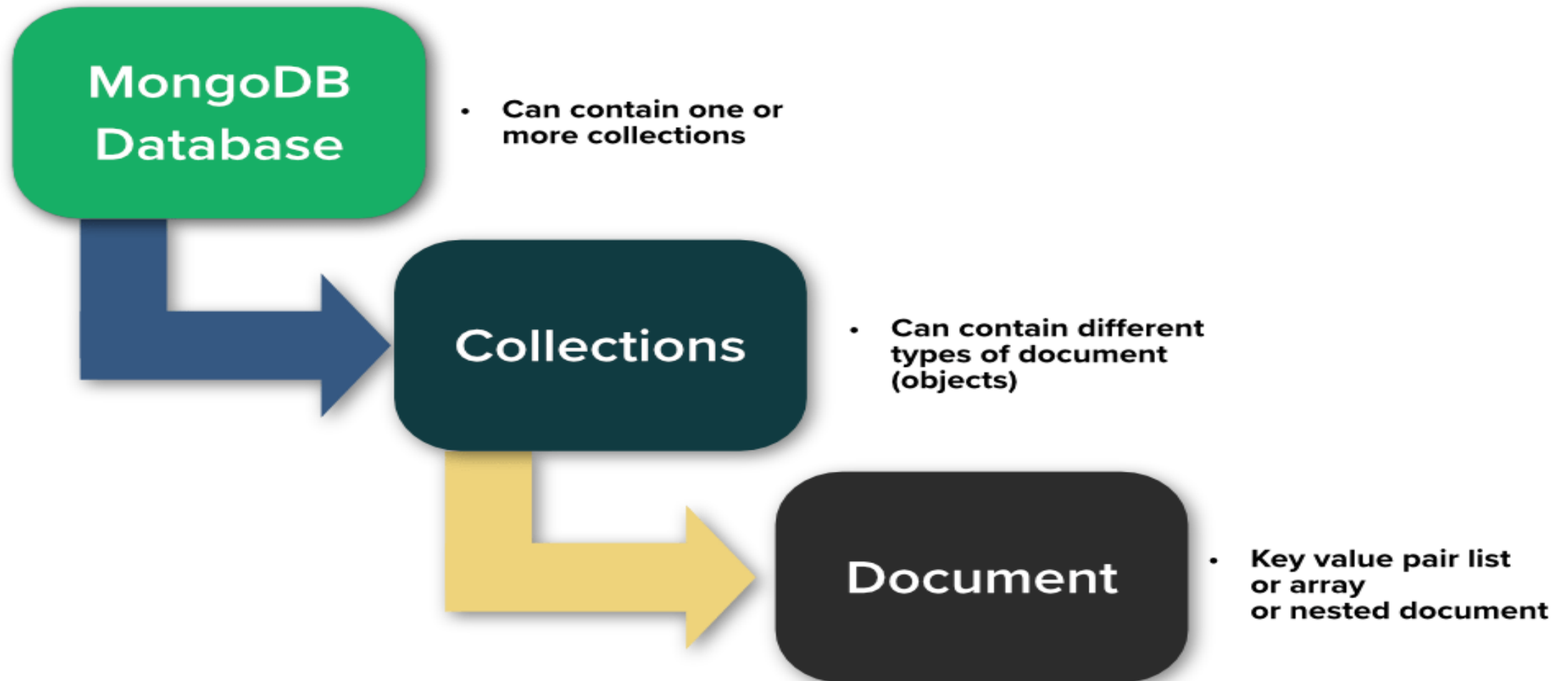
*illustradata.com*



# Presentation de MongoDB

- ▶ MongoDB est un système de gestion de base de données NoSQL.
- ▶ Open source orientée document.
- ▶ MongoDB utilise des documents de type JSON pour stocker toutes les données.
- ▶ Chaque base de données contient une ou plusieurs collections, et chaque collection contient zéro ou plusieurs documents.
- ▶ Une base de données peut contenir plusieurs collections, mais une collection ne peut pas couvrir plusieurs bases de données.
- ▶ De même, une collection peut contenir plusieurs documents, mais un document ne peut pas couvrir plusieurs collections.

# Présentation MongoDB



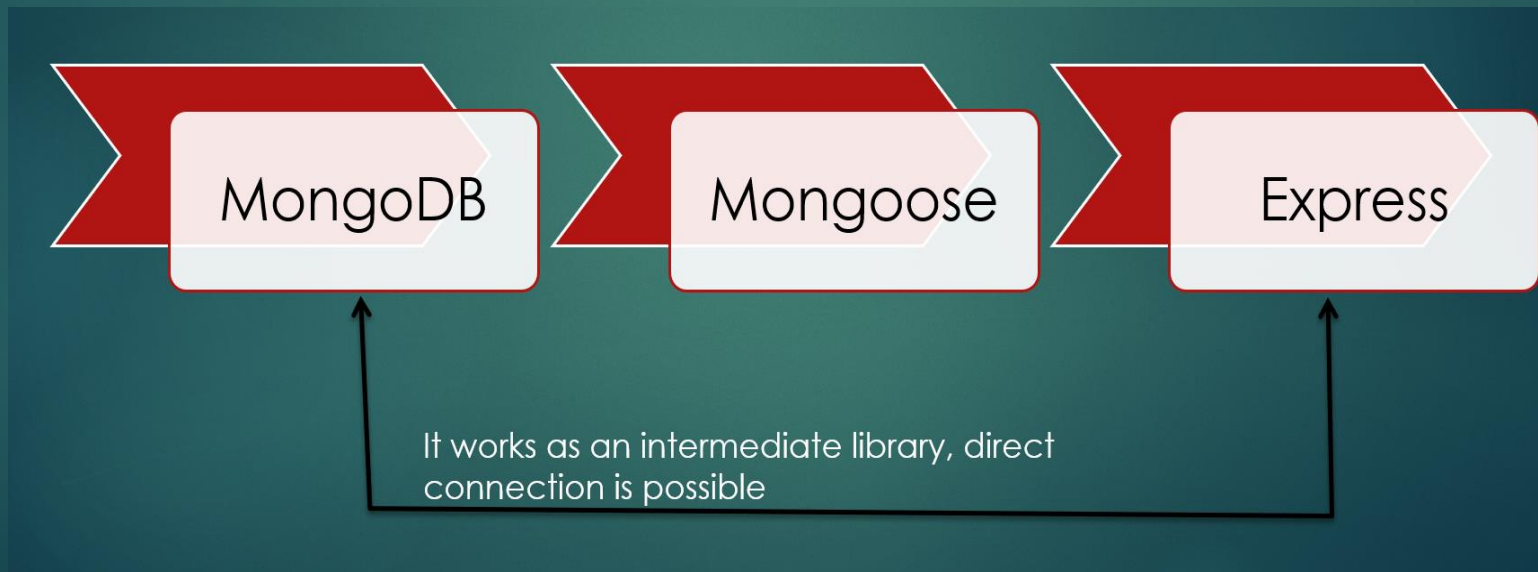


# Mongoose

CRÉATION APPLICATION CLOUD NATIVE

# Définition

- ▶ Mongoose, qui est un ODM (Object-Data Modelling) pour MongoDB et nodeJS
- ▶ Est une bibliothèque NPM (Node Package Manager).
- ▶ Connecte les collections MongoDB à une application Node.js.
- ▶ Il est utilisé pour traduire les objets dans le code et la représentation de ces objets dans MongoDB.



# Installation

Pour utiliser Mongoose, nous devons installer le paquetage mongoose en tapant la commande suivante dans le terminal ou l'invite de commande :

```
npm install mongoose
```



# Connection a la base de données

CRÉATION D'UNE APPLICATION CLOUD NATIVE

# Les étapes de connexion (1)

1. Créez un nouveau fichier app.js pour démarrer notre serveur Express.js.
2. Chargez mongoose et express en ajoutant le code suivant à app.js.

```
import express from 'express'
import mongoose from 'mongoose'

const app = express();
app.use(express.json());

app.listen(3000, (err) => {
  if(!err)
    console.log('Server started at 3000')
  else
    console.log('Echec start server at 3000')
});
```

# Les étapes de connexion (3)

3. Ensuite, On utilise la fonction `mongoose.connect()` pour se connecter à une instance locale de MongoDB.

```
mongoose.connect('mongodb://localhost:27017/dbbase',  
  {useNewUrlParser: true})
```

Nous passons le paramètre `useNewUrlParser : true`, etc. pour éviter le `DeprecationWarning`.

# Les étapes de connexion (3)

4. L'ajout du code suivant permet de vérifier que la connexion est effectuée avec succès :

```
mongoose.connect('mongodb://localhost:27017/dbbase', {useNewUrlParser: true})
  .then(() => {
    console.log('Connection Reussie')
  })
  .catch((err) => {
    console.log('Echec Connection')
  })
```

# Le code complet

```
import express from 'express'
import mongoose from 'mongoose'

const app = express();
app.use(express.json());

mongoose.connect('mongodb://localhost:27017/dbbase', {useNewUrlParser: true})
  .then(() => {
    console.log('Connection Reussie')
  })
  .catch((err) => {
    console.log('Echec Connection')
  })

app.listen(3000, (err) => {
  if(!err)
    console.log('Server started at 3000')
  else
    console.log('Echec start server at 3000')
});
```





# Création des schémas et modèle

CRÉATIONS D'UNE APPLICATION CLOUD NATIVE

# Les schemas

- ▶ Tout dans Mongoose commence par un schéma.
- ▶ Chaque schéma correspond à une collection MongoDB et définit la forme des documents de cette collection.

# Les schemas – Exemple 1

## Exemple 1 – Creation d'un schema sans validation

```
import mongoose from 'mongoose'

const UserSchema = new mongoose.Schema({
  nom:String,
  prenom:String,
  age:Number
})
```

# schémas types

Les schémas types autorisés sont :

- ❑ String
- ❑ Number
- ❑ Date
- ❑ Boolean
- ❑ Mixed
- ❑ ObjectId
- ❑ Array

# Les schemas – Exemple 2

## Exemple 2 – sans validation

```
const blog = new Schema({  
  title: String,  
  slug: String,  
  published: Boolean,  
  content: String,  
  tags: [String],  
  comments: [{  
    user: String,  
    content: String,  
    votes: Number  
  }]  
});
```

# Validation de mongoose

- ▶ Il existe deux types de validation dans mongoose :
  - ▶ Validation intégrée : required, unique, default, min, max, minlength
  - ▶ Validation personnalisée

# Les schemas – Exemple 3

## Exemple 3 – Avec Validation:

```
import mongoose from 'mongoose'

const UserSchema = new mongoose.Schema({
  nom:{type:String, required:[true, "Nom est obligatoire"]},
  prenom:{type:String, required:true, unique:true, minLength:4},
  age:{type:Number, min:20, max:40}
})

export default mongoose.model('user', UserSchema)
```

# Les schemas – Exemple 4

## Exemple 4 – avec validation :

```
const blog = new Schema({
  title: { type: String, required: true },
  author: { type: String, required: true, minlength: 20},
  published: Boolean,
  content: {
    type: String,
    required: true,
    minlength: 250
  }
});
```



# Les schemas – Exemple 5

## Exemple 5 – avec validation:

```
import mongoose from 'mongoose';
const { Schema } = mongoose;
const blogSchema = new Schema({
  title: String,
  author: String,
  body: String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs: Number
  }
});
```

# Création d'un modèle

- ▶ Pour utiliser notre définition de schéma, nous devons convertir notre schéma en un modèle avec lequel nous pouvons travailler.
- ▶ Pour ce faire, nous le passons dans :

```
mongoose.model(modelName, schema)
```

## Exemple :

```
const user = mongoose.model('User', UserSchema, "users")
```

# Exemple complet

```
import mongoose from 'mongoose'

const UserSchema = new mongoose.Schema({
  nom:{type:String, required:[true, "Nom est obligatoire"]},
  prenom:{type:String, require:true, unique:true, minLength:4},
  age:{type:Number, min:20, max:40}
})

export default mongoose.model('user', UserSchema)
```

# Insérer des données

CRÉATION D'UNE APPLICATION CLOUD NATIVE

# Insérer des données

- ▶ Pour insérer un seul document dans MongoDB, on utilise la méthode `save()` sur l'instance du document :
  - ▶ Une fonction `CallBack (err, document)` est un argument facultatif de la méthode `save()`.
  - ▶ L'insertion se fait de manière asynchrone et toute opération dépendant du document inséré doit se faire dans la fonction `CallBack` pour être correcte.
- ▶ Pour insérer plusieurs documents dans `mongodb` on utilise la méthode `insertMany`.

# Exemples – Méthode save

## ► Exemple 1 :

```
const user = new userModel({nom: "alag", prenom: "Thomas", age: 44});  
user.save();
```

# Exemple complet – Méthode Save

```
var BookSchema = mongoose.Schema({name:String, price:Number, quantity:Number },
{collection : "bookstore"});

var Book = mongoose.model("Book", BookSchema, "bookstore");

var book1 = new Book({ name: « Course Mongoose", price: 10, quantity: 25,});

book1.save(function (err, book) {
    if (err) return console.error(err);
    console.log(book.name + " saved to bookstore collection.");
});
```

# Exemple - La méthode insertMany

```
userModel.insertMany(  
  [  
    { nom: « Khilal», prenom: « Maria», age: 44 },  
    { nom: « MORAR», prenom: « Nicolas», age: 44 },  
    { nom: « AZHARI», prenom: « Isabelle», age: 55 },  
  ],  
)  
.then((docs) => {  
  console.log("Donnees inserees avec succes");  
}).catch(error => {  
  console.log("Erreur d'insertion des donnees");  
});
```



# Récupérer des données

CRÉATION D'UNE APPLICATION CLOUD NATIVE

# Les méthodes de récupération des données

- ▶ Les listes de méthodes qui sont utiliser pour récupérer les données sont :
  - ▶ Model.find()
  - ▶ Model.findById()
  - ▶ Model.findOne()

# La méthode `Model.find()`

- ▶ Dans Mongoose, la fonction `Model.find()` est le principal outil d'interrogation de la base de données.
- ▶ Le premier paramètre de `Model.find()` est un objet filtre. MongoDB recherchera tous les documents qui correspondent au filtre. Si vous passez un filtre vide, MongoDB renverra tous les documents.

# La méthode Model.find() – Exemple 1

```
userModel.find({}).then((users) => {  
    console.log(users)  
}).catch((err) => {  
    console.log(err)  
})
```

# La méthode Model.find() – Exemple 2

```
userModel.find({nom: 'alag'}).then((users) => {  
    console.log(users)  
}).catch((err) => {  
    console.log(err)  
})
```

# La méthode Model.find() – Exemple 3

```
userModel.find({nom: 'alag', age: {$gt: 44}}).then((users) => {  
    console.log(users)  
}).catch((err) => {  
    console.log(err)  
})
```

# La méthode Model.find() – Exemple 4

```
userModel.find({}, "nom age").then((users) => {  
    console.log(users)  
}).catch((err) => {  
    console.log(err)  
})
```

# La méthode Model.find() – Exemple 5

```
userModel.find({}, "nom age", {limit:1}).then((users) => {  
    console.log(users)  
}).catch((err) => {  
    console.log(err)  
})
```



# La méthode findById

- La méthode findById() est utilisée pour trouver un document unique par son champ\_id.


**Exemple :**

```
app.get('/users/:id', (req, res) => {  
    userModel.findById(req.params.id).then((users) => {  
        if(!err){  
            res.send(users)  
        }  
    });  
});
```

# La méthode findOne

- La méthode findOne() est utilisée pour trouver un document selon la condition. Si plusieurs documents correspondent à la condition, elle renvoie le premier document satisfaisant la condition.

```
app.get('/users/:id', (req, res) => {  
    userModel.findOne({age: {$gt: 10}}), (err, users) => {  
        if(!err){  
            res.send(users)  
        }  
    })  
})
```



# Mettre a jour des données

CRÉATION D'UNE APPLICATION CLOUD NATIVE

# Les méthodes de modification

- ▶ Il existe deux méthodes pour modifier les documents dans une collection :
  - ▶ UpdateOne
  - ▶ UpdateMany

## Exemple :

```
userModel.updateOne({nom: 'alag'}, {age: 20}).then(docs) {  
    console.log("Updated Docs : ", docs);  
})
```

# Supprimer des données

CREATION D'UNE APPLICATION CLOUD NATIVE

# Les méthodes de suppression

► Il existe deux méthodes pour supprimer les documents dans une collection :

► deleteOne

► deleteMany

► **Exemple :**

```
userModel.deleteMany({nom: 'alag'}).then(docs) {  
    console.log("deleted Docs : ", docs);  
})
```