Lecture 4: persistence with mongodb
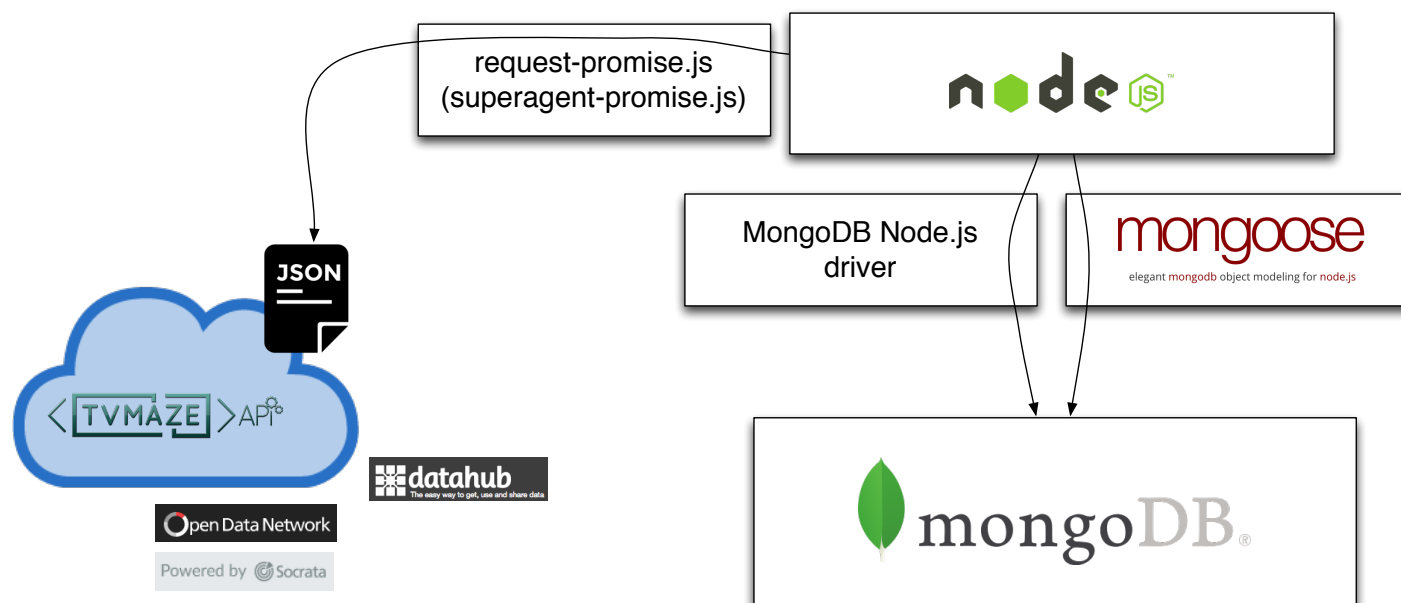
Olivier Liechti
TWEB

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Webcasts

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

1 use one of the "promisified" npm modules
to fetch JSON data from the Web

request-promise.js
(superagent-promise.js)

MongoDB Node.js
driver

mongoose
elegant mongodb object modeling for node.js

JSON

TVMAZE API

datahub
The easy way to get, use and share data

Open Data Network
Powered by Socrata

mongoDB

2 store the JSON documents in
MongoDB collections and run queries

Tasks

1. Prepare the environment
   1.1. Start a MongoDB server in a Docker container
   1.2. Start a MongoDB shell in another Docker container
   1.3. Run basic commands (create a document, find all documents)

2. Get data from the Web
   2.1. Find JSON data source
   2.2. Pick a npm module to submit HTTP requests
   2.3. Fetch data

3. Access MongoDB with the mongo driver
   3.1. Handle the connection with the MongoDB server
   3.2. Explore the driver API
   3.3. Interact with collections and documents

4. Integrate all the pieces with Promises
   3.1. Callbacks vs promises in the APIs
   3.2. Promise chains
   3.3. Passing data and returning results

5. Access MongoDB with Mongoose
   4.1. Select and study a template
   4.2. Discover jquery datatables
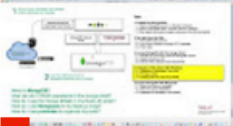   4.3. Integrate the template in the project

What is **MongoDB**?
How can do I CRUD operations in the mongo shell?
How do I use the mongo **driver** in my Node.JS script?
How do I use **Mongoose** in my Node.js script?
How do I use **promises** to organize my code?

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

11 Bootcamp 4.1: Intro aux webcasts "MongoDB"
by oliechti
2:48

12 Bootcamp 4.2: prise en main de MongoDB
by oliechti
20:30

13 Bootcamp 4.3 (a): identification de la source de données JSON
by oliechti
5:40

14 Bootcamp 4.3 (b): utilisation de request-promise pour interroger l'API REST
by oliechti
15:12

15 Bootcamp 4.4: utilisation du driver node.js MongoDB
by oliechti
21:01

16 Bootcamp 4.4: implémentation de la chaîne de promesses
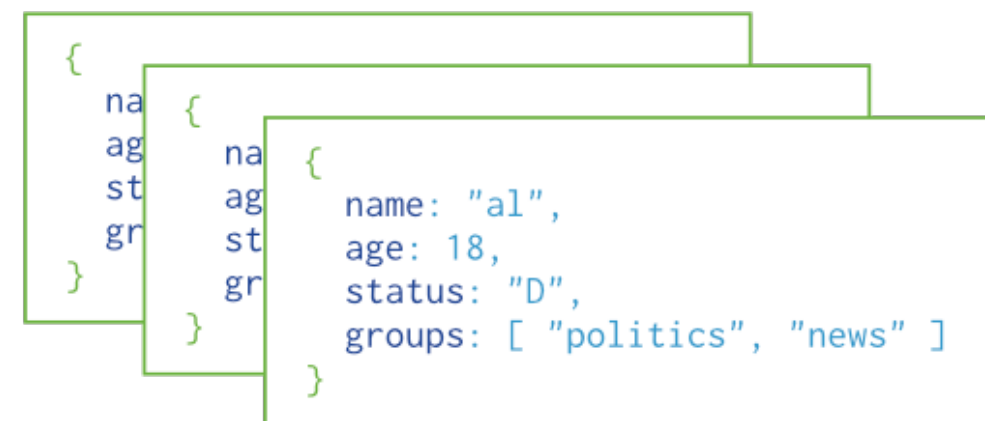by oliechti
38:46

Document-oriented NoSQL Database

MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- MongoDB is one of the most popular **NoSQL** databases (and one of the first to have been categorized as such).

- It is a **schema-less document-oriented** database:

  - The data store is made of several **collections**.

  - Every collection contains a set of **documents**, which you can think of as JSON objects.

  - The **structure of documents is not defined** *a priori* and is not enforced. This means that a collection can contain documents that have different fields.

```
{
  name: "sue",          ←——— field: value
  age: 26,              ←——— field: value
  status: "A",          ←——— field: value
  groups: [ "news", "sports" ]  ←——— field: value
}
```

```
{
  na
  ag
  st
  gr
}

    {
      na
      ag
      st
      gr
    }

        {
          name: "al",
          age: 18,
          status: "D",
          groups: [ "politics", "news" ]
        }
```

Collection

http://docs.mongodb.org/manual/core/crud-introduction/

Data modeling

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- Creating a data model with MongoDB does not have to follow the rules that apply for relational databases. Often, they should not.

- Consider theses questions: is this a **composition** relationship (**containment**)? Is this "**aggregate**" of documents often used at the same time (i.e. can we reduce chattiness)? Would embedding lead to "a lot" of data **duplication**?
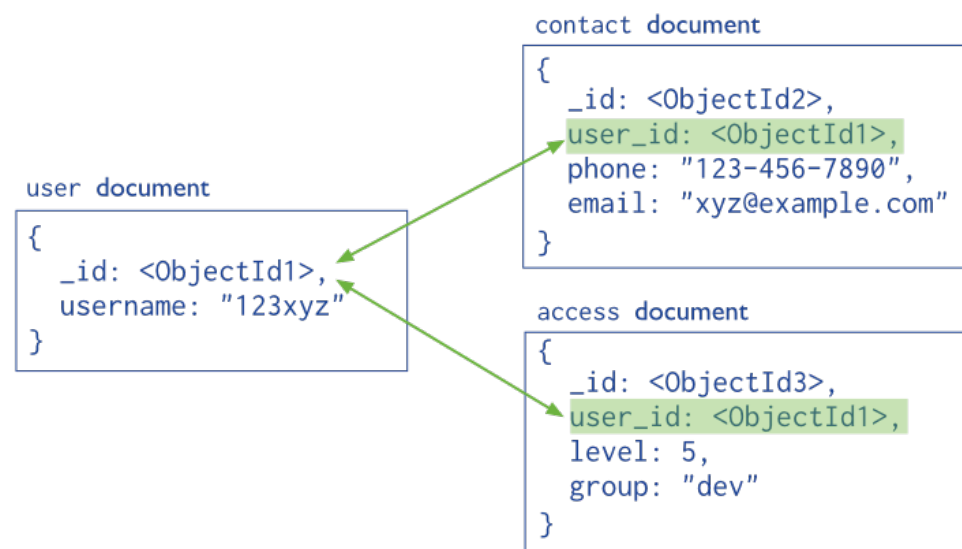
Normalized data model
(references)

Embedded data model
(sub-documents)

https://docs.mongodb.org/manual/core/data-modeling-introduction/

One-to-one relationships

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

2 documents (requires 2 queries
to get all of the person data)

```
{
    _id: "joe",
    name: "Joe Bookreader"
}

{
    patron_id: "joe",
    street: "123 Fake Street",
    city: "Faketon",
    state: "MA",
    zip: "12345"
}
```

```
{
    _id: "joe",
    name: "Joe Bookreader",
    address: {
                street: "123 Fake Street",
                city: "Faketon",
                state: "MA",
                zip: "12345"
            }
}
```

1 single, aggregate document
(in this case, it is a better choice)

Normalized data model
(references)

Embedded data model
(sub-documents)

https://docs.mongodb.org/manual/tutorial/model-embedded-one-to-one-relationships-between-documents/

One-to-many relationships

```
{
    _id: "joe",
    name: "Joe Bookreader"
}


{
    patron_id: "joe",
    street: "123 Fake Street",
    city: "Faketon",
    state: "MA",
    zip: "12345"
}


{
    patron_id: "joe",
    street: "1 Some Other Street",
    city: "Boston",
    state: "MA",
    zip: "12345"
}
```

MongoDB document can have an arbitrary structure, including arrays

```
{
    _id: "joe",
    name: "Joe Bookreader",
    addresses: [
        {
            street: "123 Fake Street",
            city: "Faketon",
            state: "MA",
            zip: "12345"
        },
        {
            street: "1 Some Other Street",
            city: "Boston",
            state: "MA",
            zip: "12345"
        }
    ]
}
```

https://docs.mongodb.org/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/

One-to-many relationships

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

ok if if have few books per publisher

```
{
    title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher: {
            name: "O'Reilly Media",
            founded: 1980,
            location: "CA"
        }
}

{
    title: "50 Tips and Tricks for MongoDB Developer",
    author: "Kristina Chodorow",
    published_date: ISODate("2011-05-06"),
    pages: 68,
    language: "English",
    publisher: {
            name: "O'Reilly Media",
            founded: 1980,
            location: "CA"
        }
}
```

```
{
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA",
    books: [12346789, 234567890, ...]
}

{
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}

{
    _id: 234567890,
    title: "50 Tips and Tricks for MongoDB Developer",
    author: "Kristina Chodorow",
    published_date: ISODate("2011-05-06"),
    pages: 68,
    language: "English"
}
```

```
{
    _id: "oreilly",
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA"
}

{
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher_id: "oreilly"
}

{
    _id: 234567890,
    title: "50 Tips and Tricks for MongoDB Developer",
    author: "Kristina Chodorow",
    published_date: ISODate("2011-05-06"),
    pages: 68,
    language: "English",
    publisher_id: "oreilly"
}
```

duplication

better if you have many books per publisher

https://docs.mongodb.org/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/

Insert data in MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- To insert data in MongoDB, you simply have to provide a **JSON document** (with an **arbitrary structure**).

- The documents in the collection do not have to all have the same structure (this is why we talk about a **schemaless** database).

```
Collection                    Document
   ↓                             ↓
db.users.insert(
                  {
                      name: "sue",
                      age: 26,
                   status: "A",
                   groups: [ "news", "sports" ]
                  }
                )
```

A document is always inserted in a specific collection.

MongoDB will assign a unique ID in the **_id** field for the new document.

```
Document
{
   name: "sue",
   age: 26,
   status: "A",
   groups: [ "news", "sports" ]
}
```

insert →

```
Collection
{ name: "al", age: 18, ... }
{ name: "lee", age: 28, ... }
{ name: "jan", age: 21, ... }
{ name: "kai", age: 38, ... }
{ name: "sam", age: 18, ... }
{ name: "mel", age: 38, ... }
{ name: "ryan", age: 31, ... }
{ name: "sue", age: 26, ... }
users
```

https://docs.mongodb.org/manual/core/crud-introduction/

Insert data in MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- This is one example. You can also insert multiple documents at the same time, either by passing an array of documents or by performing a bulk operation. If you are dealing with many documents, this is important for performance reasons.

```
db.inventory.find()
```

```
db.inventory.insert(
    {
        item: "ABC1",
        details: {
            model: "14Q3",
            manufacturer: "XYZ Company"
        },
        stock: [ { size: "S", qty: 25 }, { size: "M", qty: 50 } ],
        category: "clothing"
    }
)
```

```
{ "_id" : ObjectId("53d98f133bb604791249ca99"), "item" : "AB
```

https://docs.mongodb.org/manual/tutorial/insert-documents/

Update and delete data in MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- When you update or delete documents, you specify which documents are concerned by the operation.

- You do that by specifying update or remove criteria. Specifying "{}" means that you want to apply the operation on all documents of the collection.

```
db.users.update(                    ←——— collection
   { age: { $gt: 18 } },            ←——— update criteria
   { $set: { status: "A" } },       ←——— update action
   { multi: true }                  ←——— update option
)
```

```
db.users.remove(                    ←——— collection
   { status: "D" }                  ←——— remove criteria
)
```

https://docs.mongodb.org/manual/core/crud-introduction/

Update data in MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- By default, update will modify only the first document that matches the selection criteria. You can specify the "multi" options if you want to update all documents that match the criteria.

selection criteria

```
db.inventory.update(
    { item: "MNO2" },
    {
        $set: {
            category: "apparel",
            details: { model: "14Q3", manufacturer: "XYZ Company" }
        },
        $currentDate: { lastModified: true }
    }
)
```

```
db.inventory.update(
    { category: "clothing" },
    {
        $set: { category: "apparel" },
        $currentDate: { lastModified: true }
    },
    { multi: true }
)
```

update operators

lastModified will be set to the current date

https://docs.mongodb.org/manual/tutorial/modify-documents/

Update data in MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Update Operators

Fields

Name	Description
$inc	Increments the value of the field by the specified amount.
$mul	Multiplies the value of the field by the specified amount.
$rename	Renames a field.
$setOnInsert	Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
$set	Sets the value of a field in a document.
$unset	Removes the specified field from a document.
$min	Only updates the field if the specified value is less than the existing field value.
$max	Only updates the field if the specified value is greater than the existing field value.
$currentDate	Sets the value of a field to current date, either as a Date or a Timestamp.

Array

Operators

Name	Description
$	Acts as a placeholder to update the first element that matches the query condition in an update.
$addToSet	Adds elements to an array only if they do not already exist in the set.
$pop	Removes the first or last item of an array.
$pullAll	Removes all matching values from an array.
$pull	Removes all array elements that match a specified query.
$pushAll	*Deprecated*. Adds several items to an array.
$push	Adds an item to an array.

https://docs.mongodb.org/manual/reference/operator/update/

Query MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- MongoDB is one of the most popular **NoSQL** databases (and one of the first to have been categorized as such).

Collection — `db.users.find(` Query Criteria — `{ age: { $gt: 18 } }` `).sort(` Modifier — `{age: 1 } )`

```
{ age: 18, ...}
{ age: 28, ...}
{ age: 21, ...}
{ age: 38, ...}
{ age: 18, ...}
{ age: 38, ...}
{ age: 31, ...}
```
users

Query Criteria →

```
{ age: 28, ...}
{ age: 21, ...}
{ age: 38, ...}
{ age: 38, ...}
{ age: 31, ...}
```

Modifier →

```
{ age: 21, ...}
{ age: 28, ...}
{ age: 31, ...}
{ age: 38, ...}
{ age: 38, ...}
```
Results

We look for documents within one collection, within one database

We define the criteria for which documents should be considered, and which of their fields should be considered (projection)

We optionally sort the documents, limit the number of documents returned, etc.

https://docs.mongodb.org/manual/core/crud-introduction/

Query MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
db.inventory.find()
```

all documents

```
db.inventory.find( { type: "snacks" } )
```

documents, which have a field "type"
with a value of "snacks"

```
db.inventory.find( { type: { $in: [ 'food', 'snacks' ] } } )
```

documents, which have a field "type" with have a value
of "food" or "snacks"

```
db.inventory.find( { type: 'food', price: { $lt: 9.95 } } )
```

documents with a type field equal to "food" AND a price
field with a value less than 9.95

```
db.inventory.find(
    {
        $or: [ { qty: { $gt: 100 } }, { price: { $lt: 9.95 } } ]
    }
)
```

documents where the quantity is
more than 100 OR the price is
less than 9.95

https://docs.mongodb.org/manual/tutorial/query-documents/

Query MongoDB: arrays

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: [ 5, 8, 9 ] } )
```

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8,
```

Exact match on the entire array

```
db.inventory.find( { ratings: 5 } )
```

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8, 9 ] }
{ "_id" : 6, "type" : "food", "item" : "bbb", "ratings" : [ 5, 9 ] }
{ "_id" : 7, "type" : "food", "item" : "ccc", "ratings" : [ 9, 5, 8 ] }
```

Return documents if the array
contains a specific value

https://docs.mongodb.org/manual/tutorial/query-documents/

Query MongoDB: arrays

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: { $elemMatch: { $gt: 5, $lt: 9 } } } )
```

Documents where one element of ratings is at the same time > 5 AND < 9

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8, 9 ] }
{ "_id" : 7, "type" : "food", "item" : "ccc", "ratings" : [ 9, 5, 8 ] }
```

```
db.inventory.find( { ratings: { $gt: 5, $lt: 9 } } )
```

Documents where there is one element of ratings > 5 and one element < 9

```
{ "_id" : 5, "type" : "food", "item" : "aaa", "ratings" : [ 5, 8, 9 ] }
{ "_id" : 6, "type" : "food", "item" : "bbb", "ratings" : [ 5, 9 ] }
{ "_id" : 7, "type" : "food", "item" : "ccc", "ratings" : [ 9, 5, 8 ] }
```

https://docs.mongodb.org/manual/tutorial/query-documents/

SQL vs MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
SELECT *                   db.users.find(
FROM users                     { status: { $ne: "A" } }
WHERE status != "A"        )
```

```
SELECT *                   db.users.find( { status: "A" } ).sort( { user_id: 1 } )
FROM users
WHERE status = "A"
ORDER BY user_id ASC
```

```
SELECT *                   db.users.find(
FROM users                     { status: "A",
WHERE status = "A"               age: 50 }
AND age = 50               )
```

```
SELECT *                   db.users.find( { status: "A" } ).sort( { user_id: -1 } )
FROM users
WHERE status = "A"
ORDER BY user_id DESC
```

```
SELECT *                   db.users.find(
FROM users                     { $or: [ { status: "A" } ,
WHERE status = "A"                       { age: 50 } ] }
OR age = 50                )
```

```
SELECT COUNT(*)            db.users.count()
FROM users

                           or

                           db.users.find().count()
```

```
SELECT *                   db.users.find(
FROM users                     { age: { $gt: 25 } }
WHERE age > 25             )
```

```
SELECT COUNT(user_id)      db.users.count( { user_id: { $exists: true } } )
FROM users

                           or

                           db.users.find( { user_id: { $exists: true } } ).count()
```

```
SELECT *                   db.users.find(
FROM users                     { age: { $lt: 25 } }
WHERE age < 25             )
```

```
SELECT *                   db.users.find(
FROM users                     { age: { $gt: 25, $lte: 50 } }
WHERE age > 25             )
AND   age <= 50
```

```
SELECT COUNT(*)            db.users.count( { age: { $gt: 30 } } )
FROM users
WHERE age > 30             or

                           db.users.find( { age: { $gt: 30 } } ).count()
```

```
SELECT *                   db.users.find( { user_id: /bc/ } )
FROM users
WHERE user_id like "%bc%"
```

```
SELECT DISTINCT(status)    db.users.distinct( "status" )
FROM users
```

https://docs.mongodb.org/manual/reference/sql-comparison/

Query MongoDB: projections

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- When you perform a query, you can specify which fields you are interested in (think about performance)

```
db.inventory.find( { type: 'food' } )
```

Return all fields (no second argument)

```
db.inventory.find( { type: 'food' }, { item: 1, qty: 1 } )
```

We are only interested by the item and qty fields (we will also get _id)

```
db.inventory.find( { type: 'food' }, { item: 1, qty: 1, _id:0 } )
```

We are only interested by the item and qty fields (we really don't want to get _id)

```
db.inventory.find( { type: 'food' }, { type:0 } )
```

We want all fields except type

https://docs.mongodb.org/manual/tutorial/
project-fields-from-query-results/

Accessing MongoDB from Node.js

Accessing mongoDB from Node.js

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- In the **Java ecosystem**, it is possible to interact with a RDBMS by using a JDBC driver:

  - The program loads the driver.

  - The program establishes a connection with the DB.

  - The program sends SQL queries to read and/or update the DB.

  - The program manipulates tabular result sets returned by the driver.

- With **Node.js and mongoDB**, the process is similar:

  - There is a **Node.js driver for mongoDB** (in fact, there are several).

  - A Node.js module can connect to a mongoDB server and issue queries to **manipulate collection and documents**.

Example 1: connect and insert

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect("mongodb://localhost:27017/exampleDb", function(err, db) {
  if(err) { return console.dir(err); }

  var collection = db.collection('test');

  var doc1 = {'hello':'doc1'};
  var doc2 = {'hello':'doc2'};
  var lotsOfDocs = [{'hello':'doc3'}, {'hello':'doc4'}];

  collection.insert(doc1);
  collection.insert(doc2, {w:1}, function(err, result) {});
  collection.insert(lotsOfDocs, {w:1}, function(err, result) {});

});
```

http://mongodb.github.io/node-mongodb-native/api-articles/nodekoarticle1.html

Example 2: query

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect("mongodb://localhost:27017/exampleDb", function(err, db) {
  if(err) { return console.dir(err); }

  var collection = db.collection('test');
  var docs = [{mykey:1}, {mykey:2}, {mykey:3}];
  collection.insert(docs, {w:1}, function(err, result) {

    // beware of memory consumption!
    collection.find().toArray(function(err, items) {});

    // better when many documents are returned
    var stream = collection.find({mykey:{$ne:2}}).stream();
    stream.on("data", function(item) {});
    stream.on("end", function() {});

    // special case when only one document is expected
    collection.findOne({mykey:1}, function(err, item) {});

  });
});
```

http://mongodb.github.io/node-mongodb-native/api-articles/nodekoarticle1.html

Accessing mongoDB from Node.js

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **mongojs** is a very useful npm module, which provides an alternative to the official Node.js driver. Its API is very similar to what you type on the mongo

mongojs   Star 1,099

A node.js module for mongodb, that emulates the official mongodb API. It wraps mongodb-native and is available through npm

```
npm install mongojs
```

build passing

```javascript
// simple usage for a local db
var db = mongojs('mydb', ['mycollection']);

// the db is on a remote server (the port default to mongo)
var db = mongojs('example.com/mydb', ['mycollection']);

// we can also provide some credentials
var db = mongojs('username:password@example.com/mydb', ['mycollection']);

// connect now, and worry about collections later
var db = mongojs('mydb');
var mycollection = db.collection('mycollection');
```

```javascript
// find everything
db.mycollection.find(function(err, docs) {
    // docs is an array of all the documents in mycollection
});

// find everything, but sort by name
db.mycollection.find().sort({name:1}, function(err, docs) {
    // docs is now a sorted array
});
```

Object Document Mapping with Mongoose

Mongoose: an ORM for MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- In the **Java EE ecosystem**, we have seen how the **Java Persistence API** (JPA) specifies a standard way to interact with Object-Relational Mapping (ORM) frameworks.

  - The developer **first** creates an **object-oriented domain model**, by creating Entity classes and using various annotations (@Entity, @Id, @OneToMany, @Table, etc.)

  - He **then** uses an **Entity Manager** to **C**reate, **R**ead, **U**pdate and **D**elete objects in the DB.

  - The ORM framework takes care of the details: it **generates the schema** and the **SQL queries**.

- In the Javascript ecosystem, we have similar mechanisms. **With the particular yeoman generator that we use for the project**:

  - The authors have decided not use a relational database, but rather the **mongodb** document-oriented database.

  - They have decided to use **one of the data mapping tools** available for mongodb, namely **mongoose.** Since mongodb is a document-oriented database, it is more appropriate to talk about an Object-Document Mapping tool, rather than an ORM.

Mongoose: an ODM for MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

*"Mongoose provides a **straight-forward**, schema-based solution to **modeling** your application data and includes built-in type **casting**, **validation**, **query** building, business logic hooks and more, out of the box."*

Schema

"Everything in Mongoose starts with a Schema. Each **schema maps to a MongoDB collection** and defines the shape of the documents within that collection."

Model

"**Models are fancy constructors** compiled from our Schema definitions."

Document

"Mongoose documents represent a **one-to-one mapping** to documents as stored in MongoDB. Each document is an **instance of its Model**."

http://mongoosejs.com/docs/guide.html

Example

schema

model

collection

```
var userSchema = new mongoose.Schema({
  name: {
    first: String,
    last: { type: String, trim: true }
  },
  age: { type: Number, min: 0 }
});
```

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

var johndoe = new PUser ({
  name: { first: 'John', last: '  Doe   ' },
  age: 25
});

johndoe.save(function (err) {if (err) console.log ('Error on save!')});
```

document

https://devcenter.heroku.com/articles/nodejs-mongoose

Example: query

we can chain conditions

```
Person
.find({ occupation: /host/ })
.where('name.last').equals('Ghost')
.where('age').gt(17).lt(66)
.where('likes').in(['vaporizing', 'talking'])
.limit(10)
.sort('-occupation')
.select('name occupation')
.exec(callback);
```

we are interested in only some of the fields

we only want to get at most 10 documents

http://mongoosejs.com/docs/queries.html

Yo express!

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
yo angular
```

```
// Example model

var mongoose = require('mongoose'),
  Schema = mongoose.Schema;

var ArticleSchema = new Schema({
  title: String,
  url: String,
  text: String
});

ArticleSchema.virtual('date')
  .get(function(){
    return this._id.getTimestamp();
  });

mongoose.model('Article', ArticleSchema);
```

`app/models/article.js`

Yo express!

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
yo angular
```

```
var express = require('express'),
  router = express.Router(),
  mongoose = require('mongoose'),
  Article = mongoose.model('Article');

module.exports = function (app) {
  app.use('/', router);
};

router.get('/', function (req, res, next) {
  Article.find(function (err, articles) {
    if (err) return next(err);
    res.render('index', {
      title: 'Generator-Express MVC',
      articles: articles
    });
  });
});
```

```
app/controllers/home.js
```