

# Web Frameworks Node.js / Express.js / MongoDB

Elektronica – ICT

Sven Mariën

(sven.marien01@ap.be)

# Inhoud

## **1. MEAN Stack**

### 2. Node.JS

1. Nieuw “project”

2. Nodemon

3. CommonJS

### 3. Express.JS

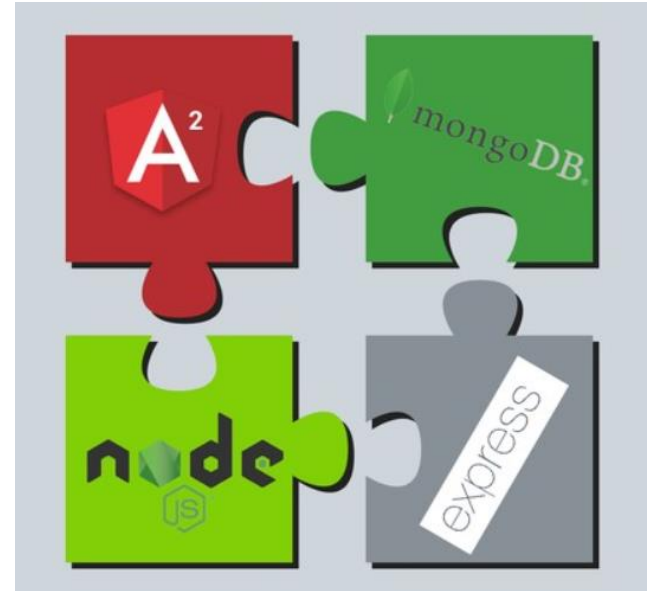
1. Klassieke webserver (files)

2. Web API services

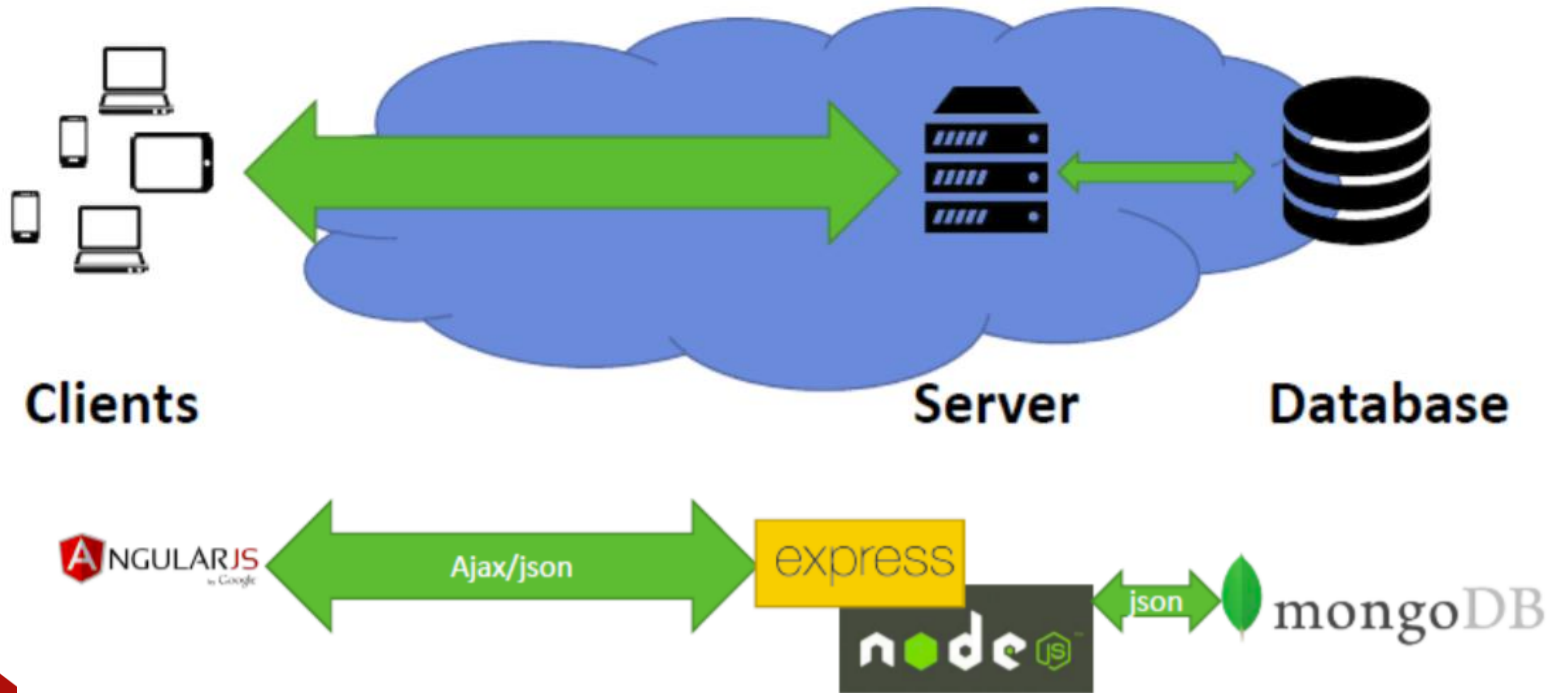
### 4. MongoDB

# “MEAN Stack”

- “Stack” is de verzameling van software waaruit het project is samengesteld.
- MEAN Stack:
  - **M**ongo DB : Databank (niet-relatieveel / NoSQL)
  - **E**xpress : Server-side framework voor Node
  - **A**ngular : Client-side framework
  - **N**ode.JS: Server-side javascript runtime
- De MEAN Stack applicatie is dus volledig gebaseerd op javascript technologieën.
- Tussen de software lagen wordt **JSON** data uitgewisseld



# “Software Layers”



# Inhoud

1. MEAN Stack

**2. Node.JS**

1. Nieuw “project”

2. Nodemon

3. CommonJS

3. Express.JS

1. Klassieke webserver (files)

2. Web API services

4. MongoDB



# Node.JS

Node.js® is a JavaScript runtime built on **Chrome's V8 JavaScript engine**. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, **npm**, is the largest ecosystem of open source libraries in the world.

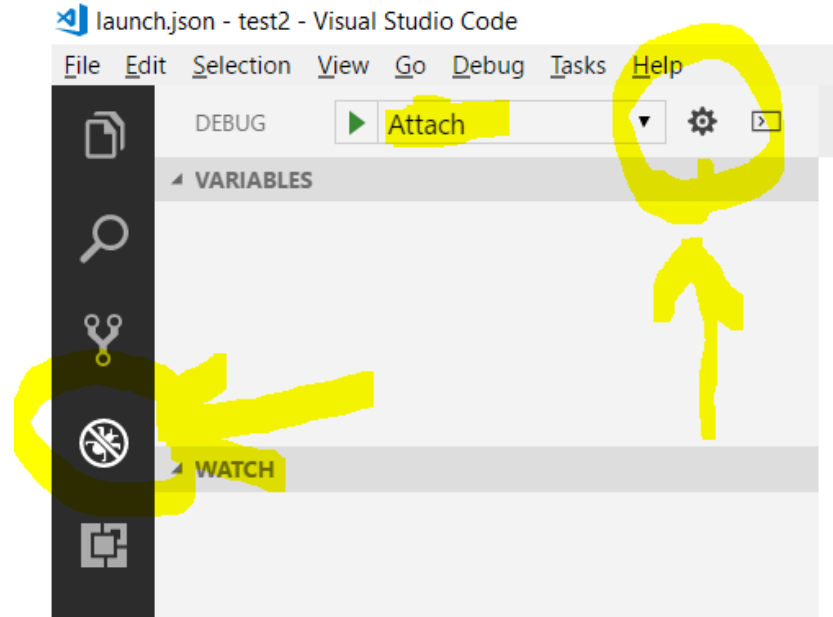
- Node.JS
  - is reeds geïnstalleerd (met de installatie van NPM)
  - Is gebaseerd op de javascript engine die oorspronkelijk is ontwikkeld voor een browser
  - Maar aan de server zijde zijn er andere noden.
  - En dus werd hiervoor voorzien met een aantal standaard “libraries” (toegang tot bestandssysteem, netwerk,...)
  - Node heeft **deels** ondersteuning voor de ES2015/ES6 standaard.(zie <http://node.green>)
    - Sommige functionaliteiten worden ondersteund “by default”
    - Andere moeten expliciet worden geactiveerd indien je ze wil gebruiken

# Node project “from scratch”

- Maak een nieuwe map aan (**mkdir xxx**)
- Start VSCode vanuit die map (**cd xxx** en **code .**)
- Maak een start bestand: bv. **App.js**
- Voer **npm init** uit -> **package.json** wordt aangemaakt
  - => hier je npm packages (dependencies) toevoegen
- Met **console.log(“...”)** kan je eenvoudig logging toevoegen.
- Debugging vanuit VSCode -> **F5** key (menu: Debug, Start Debugging)

# Nodemon

- Nodemon detecteert wijzigingen aan de broncode, her-compileert en herstart de Node applicatie
- Installeer: **npm install -g nodemon**
- Opstarten: **nodemon --inspect**
- Voeg in VSCode een Debug configuratie toe:
  - Ga links naar debug rubriek
  - Open/maak het Launch.json bestand
  - Voeg een configuratie toe (Node.js attach)
  - zet hierin **“restart”:true**
- Hiermee kan je vanuit VSCode debuggen en **terwijl de code aanpassen**





# CommonJS

- Node maakt nog gebruik van de **CommonJS** specificatie voor het werken met modules
  - De ES6 import/export syntax (zoals in typescript) zal op termijn ook mogelijk worden.
- Gebruik **module.exports** om naar andere modules te exporteren, bv.
  - `module.exports.hello = "Hallo";`
  - `module.exports.som = function(a,b) { return a+b };`
  - `module.exports.verschil = (a,b) => { return a - b }`
- Gebruik **require(...)** om een andere module in te laden, bv.
  - Eigen modules (relatief path)
    - `var helper = require("./helper.js")`
      - `console.log(helper.hello);`
      - `console.log(helper.som(5,4));`
  - Npm en standaard modules (zonder path)
    - `var mathhelper = require("mathjs")`

# Eenvoudige webserver met Node.js

- Volg bv. deze stappen (of google even):  
<https://www.digitalocean.com/community/tutorials/how-to-create-a-web-server-in-node-js-with-the-http-module>
- De interne keuken van een webserver zien we meer in detail in Cloud API (sem.2) maar een belangrijk deel is het routing gebeuren.
- Routing zorgt ervoor dat voor elke URL de juiste resource zal worden teruggestuurd naar de client.
- Het kan dus met node.js maar het is iets eenvoudiger mbv. de express npm module

# Inhoud

1. MEAN Stack
2. Node.JS
  1. Nieuw “project”
  2. Nodemon
  3. CommonJS
- 3. Express.JS**
  1. Klassieke webserver (files)
  2. Web API services
4. MongoDB

# Express.js

- Express is een **server side web framework** gebaseerd op Node.js
- <https://expressjs.com/>
- Hulpmiddel om sneller en eenvoudiger een webserver / website te ontwikkelen
  - Vooral naar routing (aan server zijde uiteraard)
- is gewoon te installeren als NPM package
- Er bestaat ook een cli (express-generator) om een volledig (start) web site project te maken (= buiten de scope van deze cursus)

# Express “basis setup”

- Aanmaken van een Express Http Server instantie
- Opstarten van de server, de server luistert op een bepaalde poort.
- Vervolgens kan je een (aantal) routes gaan definiëren
- Deze kunnen wildcards (\*) bevatten
  - bv. Alle bestanden die eindigen met **.html**
- De lijst wordt van boven naar onder afgelopen tot de eerste match is gevonden

\_\_ **dirname** is de naam van het huidige path

```
let express = require('express')
let app = express();
```

```
app.listen(8000, () => {
  console.log("server up and running")
});
```

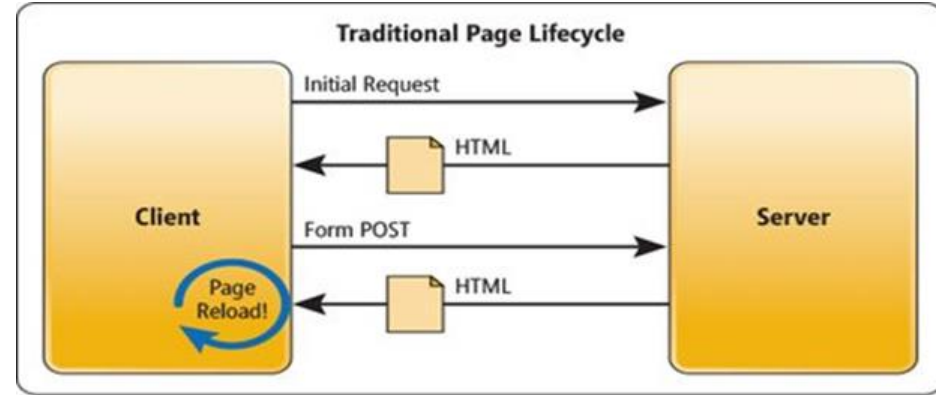
```
app.get('/', function (req, res) {
  res.send('GET request to homepage');
});
```

```
app.get('/*.html', (req, res) => {
  res.sendFile(__dirname + '\\views\\' + req.path)
});
```

```
app.get("*", (req, res) => {
  res.sendFile(__dirname + '\\public\\' + req.path.replace('/', '\\'))
})
```

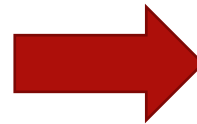
# “Klassieke” website met Express

- Veelal op basis van verschillende (html) bestanden
- Daarnaast ook “static files” (css, js, images, ...)
- Met express kan je dus vrij eenvoudig routes definiëren om de bestanden beschikbaar te maken voor de browser
- Met **app.use** kan een volledige mapstructuur beschikbaar worden gemaakt



```
app.get('/*.html', (req, res) => {
  res.sendFile(__dirname + '\\views\\' + req.path)
});

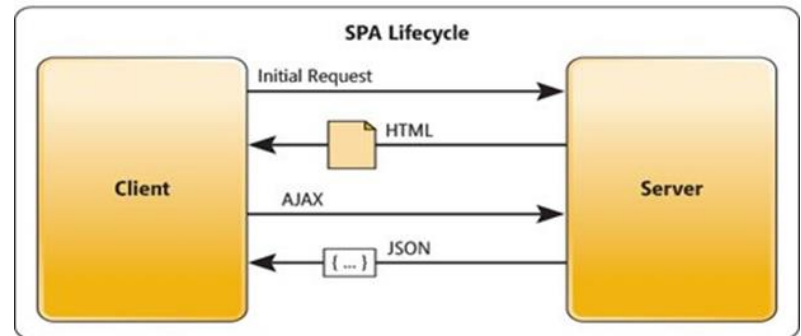
app.get("*", (req, res) => {
  res.sendFile(__dirname + '\\public\\' + req.path.replace('/', '\\'))
})
```



```
app.use(express.static("public"));
app.use(express.static("views"));
```

# Web API met Express

- Voor onze Angular SPA willen we daarnaast ook nog (JSON) 'data' kunnen opvragen.
- Dit kan via "webservices" of een Web API
- Tegenwoordig maakt men veelal gebruik van **REST** API
- REST: **RE**presentational **S**tate **T**ransfer
  - => Uniforme interface
  - => steeds onder api/xxx
- bv. <http://www.xxx.be/api/People> (lijst van alle personen)
- bv. <http://www.xxx.be/api/People/3> (enkel persoon met ID=3)
- De HTTP VERB bepaalt de actie
  - GET: ophalen
  - POST: aanmaken
  - PUT: update:
  - DELETE: verwijderen



# REST API met Express

- Express maakt het eenvoudig om JSON data terug te geven

```
var ex = require('express');
var app = ex();

let people = [{
  name: "McEnroe",
  firstName : "John"
}, {
  name : "Smith",
  firstName : "Will"
}]

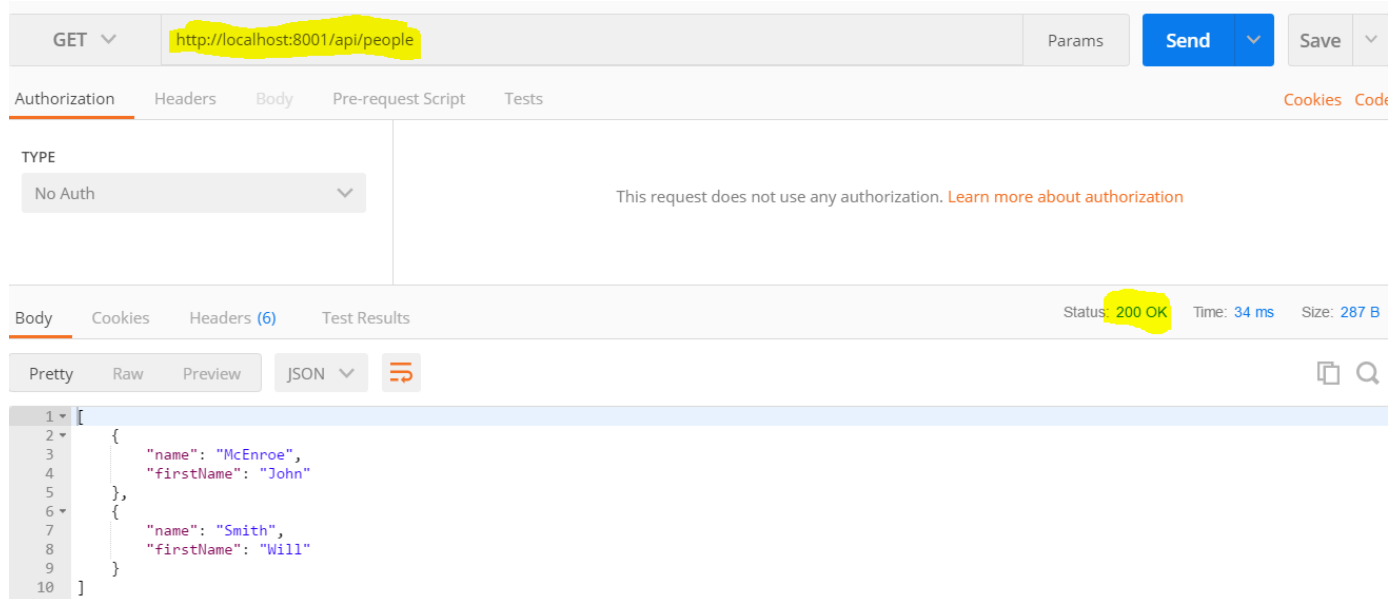
app.get('/api/people', (req, res) => {
  res.json(people)
})

app.listen(8001, () => {console.log("server is running")});
```



# REST API testen

- Er zijn verschillende tools waarmee je een REST API goed kan testen
  - Bv. Postman (<https://www.getpostman.com/> )



# Gebruik van Express Router

- Voor meer complexe routing is het beter om een Router te gebruiken, deze handelt alle requests af onder een bepaalde URL

```
var ex = require('express');
var app = ex();

let people = [{
  id: 1,
  name: "McEnroe",
  firstName: "John"
}, {
  id: 2,
  name: "Smith",
  firstName: "Will"
}]

var router = ex.Router();

router.route("/people")
  .get((req, res) => {
    res.json(people)
  })

app.use('/api', router);    //alles onder '/api' wordt afgehandeld door deze router

app.listen(8001, () => { console.log("server is running") });
```

# Ophalen van slechts 1 object

- Via de REST url kan de ID worden meegestuurd om slechts 1 object op te vragen
  - bv. <http://www.xxx.be/api/people/1>
- Via de router kunnen we een extra route instellen met id als parameter

```
var ex = require('express');
var app = ex();

let people = [{
  id: 1,
  name: "McEnroe",
  firstName: "John"
}, {
  id: 2,
  name: "Smith",
  firstName: "Will"
}]

var router = ex.Router();

router.route("/people")
  .get((req, res) => {
    res.json(people)
  })

router.route("/people/:id")
  .get((req, res) => {
    res.json(people.filter(person => person.id === req.params.id));
  })

app.use('/api', router); //alles onder '/api' wordt afgehandeld door deze router

app.listen(8001, () => { console.log("server is running") });
```

GET ▼ http://localhost:8001/api/people/1

Authorization Headers Body Pre-request Script

TYPE

No Auth ▼

Body Cookies Headers (6) Test Results

Pretty Raw Preview JSON ▼ ≡

```
1 [
2   {
3     "id": 1,
4     "name": "McEnroe",
5     "firstName": "John"
6   }
7 ]
```

# Ophalen van objecten met Url query string

- Via de REST Url kunnen eveneens query parameters worden doorgegeven
  - Bv. <http://www.xxx.be/api/people?firstName=Jos&Age=32>
  - In Express kunnen deze parameters worden opgevraagd via de query property van het request object
  - We krijgen dan een object met alle ingestelde parameters als property
  - Hiermee kan dan vervolgens gefilterd worden in de data

```
var router = ex.Router();
```

```
router.route("/people")  
  .get((req, res) => {  
    var query = req.query;
```

```
http://localhost:3000/api  
router.route('/people')  
get((req, res) => {  
  //Select  
  let query = req.query;  
  res.json(people);
```

```
Object {firstName: "Jos", Age: "32"}
```

```
Age: "32"
```

```
firstName: "Jos"
```

```
__proto__: Object {__defineGetter__: , __de
```

# GET/POST/PUT/DELETE

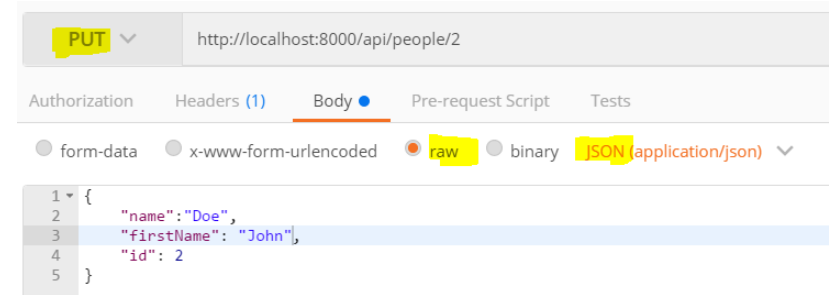
- Via de router is het eveneens mogelijk om de andere HTTP VERBS te voorzien:

```
// http://localhost:3000/api/people          GET => alle personen
// http://localhost:3000/api/people?firstName=bert&lastName=DeLoo GET => bepaalde personen
// http://localhost:3000/api/people          POST => een persoon toevoegen
// http://localhost:3000/api/people?firstName=bert&lastName=DeLoo DELETE
apiRouter.route('/people')
  .get((req, res) => {
    //Select
    let query = req.query;
    //...
  })
  .post((req, res) => {
    //Insert
  })
  .delete((req, res) => {
    //Delete
  })

// http://localhost:3000/api/people/id      GET => een bepaalde persoon
// http://localhost:3000/api/people/id      PUT => een persoon aanpassen
// http://localhost:3000/api/people/id      DELETE => een persoon verwijderen
apiRouter.route('/people/:id')
  .get((req, res) => {
    //Select
    let id = req.params.id
    //...
  })
  .put((req, res) => {
    //Update
  })
  .delete((req, res) => {
    //Delete
  })
```

# POST/PUT

- Bij het gebruik van de POST (insert data) en PUT (update data) verbs dient er uiteraard data te worden meegestuurd met de HTTP request.
- Dit gebeurt via de **body** van de request, dewelke als **raw / JSON** dient te worden doorgestuurd.
- Aan de zijde van express moeten we dan de **body-parser** NPM package installeren en instellen zodat deze wordt gebruikt door express.



```
var bodyParser = require('body-parser');
// support encoded bodies
app.use(bodyParser.json());
```

```
router.route("/people/:id")
  .get((req, res) => {
    res.json(people.filter(person => person.id === +req.params.id)[0]);
  })
  .put((req, res) => {
    let person = people.filter(person => person.id === +req.params.id)[0];
    person.name = req.body.name;
    person.firstName = req.body.firstName;
    //Send some response so that status code 200 is returned
    res.json(person);
  })
```

# Inhoud

1. MEAN Stack
2. Node.JS
  1. Nieuw “project”
  2. Nodemon
  3. CommonJS
3. Express.JS
  1. Klassieke webserver (files)
  2. Web API services
- 4. MongoDB**

# MongoDB



- MongoDB is een noSQL databank
- Je kan zelfs werken zonder “schema”, dit wordt automatisch bepaald door de “data”
- Is “document” gebaseerd, alle gegevens voor 1 “record” staan in 1 document.
- Documenten worden verzameld in een “collection”.
- Documenten hebben het JSON formaat (name/value pairs)

```
{
  _id: "12244",
  name: "McEnroe",
  firstName: "John"
}, {
  _id: "43432",
  name: "Smith",
  firstName: "Will"
}
```

document

document

“people”  
collection



# De gegevens bepalen het “schema”

```
{  
  id: 1,  
  name: "McEnroe",  
  firstName: "John"  
}, {  
  id: 2,  
  name: "Smith",  
  firstName: "Will"  
}
```

```
{  
  id: 1,  
  name: "McEnroe",  
  firstName: "John"  
}, {  
  id: 2,  
  name: "Smith",  
  firstName: "Will",  
  age: 45  
}
```

```
{  
  id: 1,  
  name: "McEnroe",  
  firstName: "John",  
  address: {  
    street: "An der Helling 2",  
    state: "Wiesbaden",  
    country : "Germany"  
  }  
}, {  
  id: 2,  
  name: "Smith",  
  firstName: "Will",  
  address: {  
    street: "Route 66",  
    state: "Kansas",  
    country: "USA"  
  }  
}
```

# “Relaties”, 2 mogelijkheden

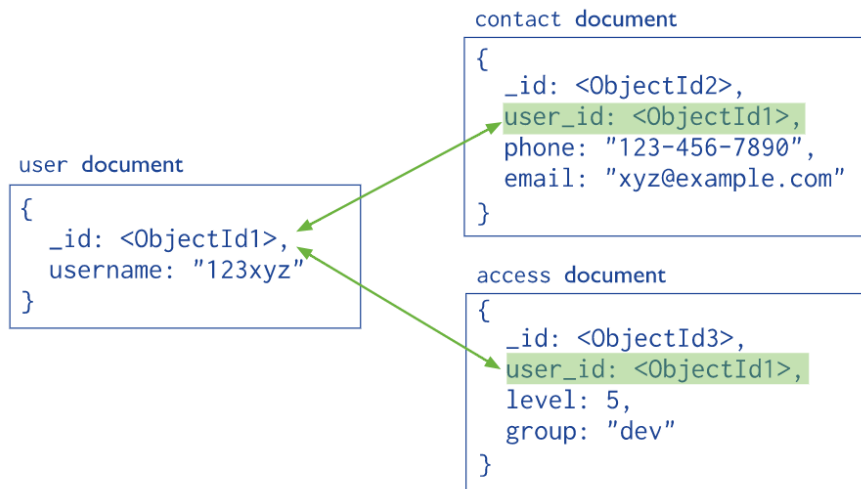
## Embedded “subdocuments”

- Gedenormaliseerd
- Minder “queries” (geen “joins”)
- Betere “read performance”
- Update : 1 “write operation”



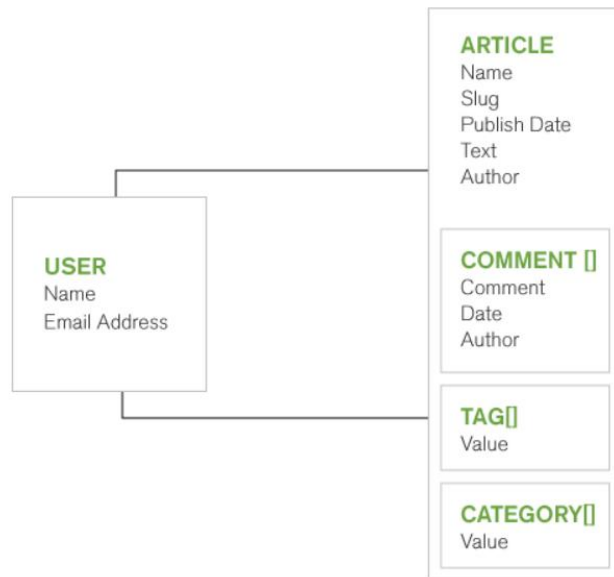
## Referenties tussen documenten

- Genormaliseerd
- Geen duplicatie van gegevens
- Kleinere documenten
- Voor “many-to-many” relaties



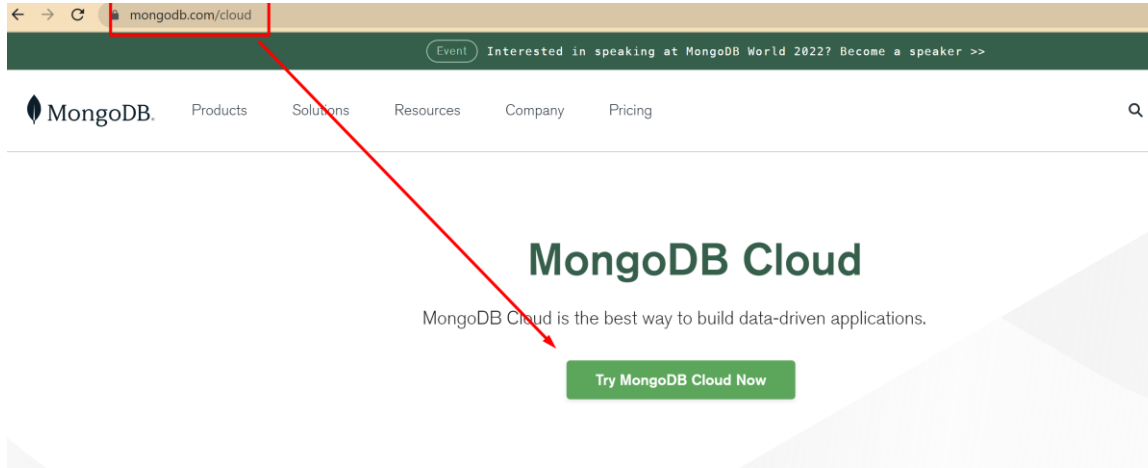
# Hoe “document based modeleren” ?

- Voorbeeld (Blog): links - klassiek genormaliseerd, rechts - document based
- Afhankelijk van hoe de toepassing omgaat met de data
  - Bij het ophalen van artikels zullen veelal ook de “comments”, “tags”,.. worden opgehaald
  - Bij het verwijderen van een artikel moeten de “comments”,.. ook verwijderd worden.
  - Over hoeveel “data” gaat het, enz...




# MongoDB setup (cloud versie)

- <https://cloud.mongodb.com/>



**Get started free**  
No credit card required

 Sign up with Google


or

Your Company (optional)

Your Work Email

First Name

Last Name

Password 

8 characters minimum

☐ I agree to the [terms of service](#) and [privacy policy](#).

**Get started free**

Already have an account? [Sign in](#).

# Aanmaken van een Cluster / databank

Access our products from your MongoDB account

## DEVELOP & DEPLOY



Access your database from the Cloud and manage your data

[Visit MongoDB Atlas](#)

## EDUCATION & TRAINING



Take free courses to become a certified expert on MongoDB

[Visit MongoDB University](#)

## GET HELP



View active support cases and speak to the Support team

[Visit MongoDB Support](#)

## CONNECT & DISCUSS



Meet and seek guidance from other MongoDB developers

[Visit MongoDB Community](#)

## LEAVE FEEDBACK



## Create a database

Choose your cloud provider, region, and specs.

[Build a Database](#)


Once your database is up and running, live migrate an existing MongoDB database into Atlas with our [Live Migration Service](#).

# Selecteer de Shared (Free) version

Deploy a cloud database

Experience the best of MongoDB on AWS, Azure, and Google Cloud. Choose a deployment option to get started.

PREVIEW


 **Serverless**

For serverless applications that aren't critical with variable traffic. Minimal configuration required.

- ✓ Pay only for the operations you run
- ✓ Resources scale seamlessly to meet your workload
- ✓ Always-on security and backups

Create

Starting at  
**\$0.30/1M reads**

 **Dedicated**


For production applications with sophisticated workload requirements. Advanced configuration controls.

- ✓ Network isolation and fine-grained access controls
- ✓ On-demand performance advice
- ✓ Multi-region and multi-cloud options available

Create

Starting at  
**\$0.08/hr\***  
\*estimated cost \$56.94/month

FREE

 **Shared**

For learning and exploring MongoDB in a cloud environment. Basic configuration options.

- ✓ No credit card required to start
- ✓ Explore with sample datasets
- ✓ Upgrade to dedicated clusters for full functionality

Create

Starting at  
**FREE**

# Kies een Provider

**PREVIEW** Serverless    Dedicated    **FREE Shared**

For learning and exploring MongoDB in a sandbox environment. Basic configuration controls.  
No credit card required to start. Upgrade to dedicated clusters for full functionality.  
Explore with sample datasets. Limit of one free cluster per project.

**Cloud Provider & Region**    GCP, Belgium (europ

aws    Google Cloud    Azure

★ Recommended region ⓘ    Paid tier region ⓘ

**NORTH AMERICA / SOUTH AMERICA**    **EUROPE / MIDDLE EAST / AFRICA**    **ASIA PACIFIC**

🇧🇷 Sao Paulo (southamerica-east1) ★    🇧🇪 Belgium (europa-west1) ★    🇮🇩 Jakarta (asia-southeast2)

🇺🇸 Iowa (us-central1) ★    🇳🇱 Netherlands (europa-west4) ★    🇰🇷 Seoul (asia-northeast3)

**Cluster Tier**    M0 Sandbox (Shared RAM, 512 MB Storage)    Encrypted

**Additional Settings**    MongoDB 4.4, No Backup

**Cluster Name**    WebFrameworksCluster

One time only: once your cluster is created, you won't be able to change its name.

Cluster names can only contain ASCII letters, numbers, and hyphens.

**FREE**    Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.    [Back](#)    **Create Cluster**

# Aanmaken van een user account

The screenshot shows the 'Security Quickstart' page in the MongoDB Atlas interface. The 'Username and Password' option is selected and highlighted with a green border. Below it, a text box explains that a database user will be created with 'read and write to any database' privileges. A red box highlights the 'Create User' form, which includes fields for 'Username' (containing 'Sven') and 'Password', and a 'Create User' button. An arrow points from the 'Create User' button to the 'Authentication Type' column in the table below, which shows 'Password' as the selected type for the user 'Sven'. An 'EDIT' button is also visible next to the table entry.

Atlas Realm Charts

## Security Quickstart

To access data stored in Atlas, you'll need to create users and set up network security controls. [Learn more about security setup](#)

✓ How would you like to authenticate your connection?

Your first user will have permission to read and write any data in your project.

Username and Password Certificate

Create a database user using a username and password. Users will be given the *read and write to any database privilege* by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password. You can manage existing users via the [Database Access Page](#).

Username Password

Enter username Enter password Create User

Username Authentication Type

Sven Password EDIT



# Toegang instellen

- Enkel vanop ingestelde adres(sen) zal toegang tot de db mogelijk zijn.

✓ Where would you like to connect from?

Enable access for any network(s) that need to read and write data to your cluster.

**My Local Environment**  
Use this to add network IP addresses to the IP Access List. This can be modified at any time.

**Cloud Environment**  
Use this to configure network access between Atlas and your cloud or on-premise environment. Specifically, set up IP Access Lists, Network Peering, and Private Endpoints.

ADVANCED

Add entries to your IP Access List

Only an IP address you add to your Access List will be able to connect to your project's clusters. You can manage existing IP entries via the [Network Access Page](#).

IP Address	Description	
<input type="text" value="Enter IP Address"/>	<input type="text" value="Enter description"/>	<input type="button" value="Add Entry"/>
		<input type="button" value="Add My Current IP Address"/>

IP Access List	Description	
193.191.187.200/32	My IP Address	<input type="button" value="REMOVE"/>

# Verbinding maken adhv. Compass Client

S/S ORG - 2021-12-08 > PROJECT 0

## Database Deployments

Find a database deployment...

**SvenCluster** **Connect** View Monitoring Browse Collections ...

**R 0**  
**W 0**  
Last 2 minutes  
100.0/s

**Connections 0**  
Last 2 minutes  
100.0

**In 0.0 B/s**  
**Out 0.0 B/s**  
Last 2 minutes  
100.0 B/s

**Data Size 0.0 B**  
Last 2 minutes  
512.0 MB

VERSION	REGION	CLUSTER TIER	TYPE	BACKUPS	LINKED REALM APP	ATLAS
4.4.10	GCP / Belgium (europe-west1)	M0 Sandbox (General)	Replica Set - 3 nodes	Inactive	None Linked	Cre...

## Connect to SvenCluster

✓ Setup connection security Choose a connection method Connect

Choose a connection method [View documentation](#)

Get your pre-formatted connection string by selecting your tool below.

**Connect with the MongoDB Shell**  
Interact with your cluster using MongoDB's interactive Javascript interface

**Connect your application**  
Connect your application to your cluster using MongoDB's native drivers

**Connect using MongoDB Compass**  
Explore, modify, and visualize your data with MongoDB's GUI

Go Back

Close

# Installeren van Compass Client

Connect to SvenCluster

✓ Setup connection security

✓ Choose a connection method

Connect

I do not have MongoDB Compass

I have MongoDB Compass

1 Select your operating system and download MongoDB Compass

Windows 64-bit (7+)

Download Compass (1.29.5)

or Copy download URL

2 Copy the connection string, then open MongoDB Compass.

mongodb+srv://Sven:<password>@svenccluster.mnrns.mongodb.net/test

You will be prompted for the password for the **Sven** user's (Database User) username.  
When entering your password, make sure that any special characters are URL encoded.

Having trouble connecting? [View our troubleshooting documentation](#)

Go Back

Close

# Connectie vanuit Compass Client

MongoDB Compass - Connect

Connect View Help

New Connection

★ Favorites

🔄 Recents

## New Connection

☆ FAVORITE

Fill in connection fields individually

Paste your connection string (SRV or Standard ⓘ)

mongodb+srv://Sven:<password>@svencuster.mnrns.mongodb.net/test

Password cannot be empty

Connect

MongoDB Compass - svencuster.mnrns.mongodb.net

Connect View Help

Local

2 DBS 7 COLLECTIONS

☆ FAVORITE

HOSTS

svencuster-shard-00-00.mnrns.mongodb.net:27017  
svencuster-shard-00-02.mnrns.mongodb.net:27017  
svencuster-shard-00-01.mnrns.mongodb.net:27017

CLUSTER

Replica Set (atlas-md98m5-shard-0)  
3 Nodes

EDITION

MongoDB 4.4.10 Enterprise

Filter your data

> admin

> local

Databases Performance

CREATE DATABASE

Database Name	Storage Size	Collections	Indexes
admin	0.0B	0	0
local	0.0B	7	0

# Vanuit javascript naar de databank

- Installeer de 'mongodb' package.
- Deze bevat een **MongoClient**
- Via de **connect** methode kan je verbinding maken met de **server** en de juiste **database** (waarvan je uiteraard zelf de naam kan kiezen)

```
const mongoClient = require('mongodb').MongoClient;
var db;

// Connection URL
const url = 'mongodb://localhost:27017';
// Database Name
const dbName = 'myPeopleDB';

//connect met MongoDB
mongoClient.connect(url, (err, client) => {
  if (err != null)
  {
    console.log(err.message);
    throw err;
  }

  db = client.db(dbName);
  console.log('connected to Mongo');
})
```

# Insert & Select

//toevoegen van gegevens

```
newUser = { 'name': 'Doe', 'firstname': 'John' };  
db.collection('persons').insert(newUser, function (err, result) {  
  |   //...  
  |}  
})
```



```
{  
  |   "_id": "5a2e39cf0f4a0c5ebcba881b",  
  |   "firstname": "John",  
  |   "name": "Doe"  
  |}
```

//Ophalen van gegevens

```
db.collection('persons').find().toArray(function (err, persons) {  
  |   //...  
  |}  
})
```

//ophalen van gegevens adhv. een 'query' object

```
var query = { firstName : 'John'};  
db.collection('persons').find(query).toArray(function (err, persons) {  
  |   //...  
  |}  
})
```

//ophalen van gegevens adhv. de \_id

```
var query = { _id : ObjectId("xxxxxxxxxx")}  
db.collection('persons').find(query).toArray(function (err, persons) {  
  |   //...  
  |}  
})
```

# Delete & Update

```
//verwijderen van gegevens adhv. de _id
var query = { _id : ObjectId("xxxxxxxxxx")}
db.collection('persons').deleteOne(query, function (err, persons) {
|   //...
})
```

```
//verwijderen van gegevens adhv. een 'query' object
var query = { firstname : "John", lastname : "McEnroe"}
db.collection('persons').deleteMany(query, function (err, persons) {
|   //...
})
```

```
//gegevens aanpassen
var key = { _id : ObjectId("xxxxxxxx")}
var updateUser = { firstname : 'Johnny', name : 'Doe' }
db.collection('persons').update(key , updateUser, function(err, result) {
|   //...
})
```

