# Big data Technologies

*Opdracht 2*


| | |
|---|---|
| Klas | EIN3VBDa |
| Namen | Vincent Beltman & Mike Holtkamp |
| Versie | 1.0 |
| Leraar | Evert Duipmans |

# Inhoudsopgave

# 1 Opdracht

## 1.1 Achtergrond

Het bedrijf Thuiseten wil een database hebben waarmee recepten en klantgegevens kunnen worden opgeslagen en opgehaald. De volgende dingen moeten apart ingevoerd kunnen worden:

- Een klant moet een beoordeling kunnen geven aan een recept;
- Een klant moet zich kunnen registreren;
- Een klant moet een recept kunnen liken;
- Een klant moet een recept kunnen toevoegen;
- Een klant moet een reactie op een recept kunnen geven, maar ook op een andere reactie;

Op de volgende manieren moet kunnen worden gezocht:

- Een klant moet een recept kunnen vinden door ingrediënten in te voeren;
- Een klant moet een recept op naam kunnen vinden;
- Een klant moet zijn eigen of die van iemand anders kunnen vinden;
- Een klant moet een lijst krijgen met recepten die hij wel leuk zou kunnen vinden;
- Een klant moet de beste recepten kunnen zien;

Verder moeten er indexen geplaatst worden. En er moet nagedacht worden over eventuele uitbreidingen aan het systeem. Het systeem hoeft geen updates, als in het wijzigen van namen en ingrediënten, uit te voeren.

# 2 Database ontwerp

De gehele database bestaat uit twee collecties. De user-collectie en de recepten-collectie. De usercollectie ziet er als volgt uit:

| Attribuut | | Voorbeeld | Verantwoording |
|---|---|---|---|
| **_id** | | Vincent1995 | In _id wordt een username opgeslagen. Hiervoor hebben wij gekozen, omdat de username al uniek moet zijn per user. Hierdoor kunnen we een user gemakkelijk ophalen aan de hand van zijn name in plaats van een id.<br>Ook is dit een voordeel, omdat Mongo automatisch een index maakt voor de _id velden. Hierdoor wordt het zoeken op username snel. |
| **likes** | **commentID** | 54fc59831822ad7f85691953 | Deze verwijst naar de reactie die de user heeft geliked. We slaan dit op in user, zodat het mogelijk is om, bij het zichtbaar maken van de reacties, te zien welke reacties hij al geliked heeft.<br>Dit is beter dan het opslaan van usernames in een reactie, omdat het userobject nog vrij klein is en dat van een recept niet. Zo kun je een recept sneller op halen. |
| | **like** | true | Dit geeft aan of de like positief of negatief was. We slaan dit op in een variabele in plaats van een aparte lijst. |
| **recipes** | | [54ff266ea4c1adc39e8fbdda, 54f88f063674acb7bfabf910] | Dit is een lijst van de recepten die een user heeft toegevoegd. Op deze manier is het gemakkelijk om de recepten van een gebruiker op te halen. |

De recepten collectie ziet er als volgt uit.

| Attribuut | | Voorbeeld | Verantwoording |
|---|---|---|---|
| **_id** | | 54f88f063674acb7bfabf910 | Een recept heeft wel een naam, maar we hebben ervoor gekozen om dit niet als primary key te gebruiken. Er kunnen namelijk twee kipsalades toegevoegd worden en het zou dan lelijk zijn om er kipsalade2 van te moeten maken. |
| **name** | | Kipsalade | De naam van het recept. |
| **personCount** | | 5 | Het aantal personen waarvoor dit recept bedoeld is. |
| **courses** | | [Toetje] | Een lijst met gangen waar dit recept voor bedoeld is. |
| **difficulty** | | 5 | De moeilijkheidsgraad van het recept |
| **preparationTime** | | 20 | Het aantal minuten om het recept te kunnen bereiden. We hadden hier ook seconden kunnen opslaan, maar als je dan een erwtensoep van anderhalf uur wilt opslaan, wordt het onoverzichtelijk en moeilijker om in te voeren. Recepten worden ook meestal afgebeeld in minuten. |
| **types** | | [Frans, Engels] | De keuken waar dit recept vandaan komt. Dit is een lijst, omdat het mogelijk is dat een recept bijvoorbeeld Chinees-Indisch is. |
| **ingredients** | **name** | Kip | De naam van het ingrediënt. |
| | **amount** | 3 | De benodigdheden. |
| | **unit** | stuks | De eenheid. Bijvoorbeeld gram of theelepels. |
| **description** | | Een salade met kip mayonaise en als geheim ingrediënt: ananas. | Een beschrijving van wat het recept inhoud. |
| **procedures** | | [Stap1, Stap2, Stap3] | Een lijst met stappen. We hebben ervoor gekozen om dit gescheiden te houden van de beschrijving, omdat het op deze manier gemakkelijk is om het gescheiden af te beelden. |
| **reviews** | **username** | Vincent1995 | De naam van de gebruiker die de review gegeven heeft. We slaan reviews niet op in de user, omdat de user alleen gebruik maakt van de reviews als deze een recept ophaalt. Dan kunnen we dus beter de reviews in de recepten opslaan. |
| | **review** | 4 | Het aantal sterren dat deze user geeft aan dit recept. |

| | | | |
|---|---|---|---|
| **comments** | **_id** | 54fc541218225a705103e521 | Het ID van deze reactie. |
| | **username** | Vincent1995 | De eigenaar van deze reactie. |
| | **body** | Wat een geweldig recept | De body van deze reactie. |
| | **path** | | * |
| | **depth** | 0 | * |
| | **likes** | 1 | Het aantal likes voor deze reactie. De likes worden hier ook opgeslagen, zodat de database bij het ophalen van de likes niet alle users langs moet gaan. De likes worden hier echter wel als een getal opgeslagen en niet als een lijst. De usernames hebben we hier toch niet nodig. |
| | **dislikes** | 0 | Zie likes |

* Wij hebben ervoor gekozen om de reacties niet in een boomstructuur op te slaan. Het nadeel van het opslaan in een boomstructuur, is namelijk dat het zoeken dan veel moeilijker en intensiever wordt. Daarom hebben we ervoor gekozen om de reacties in één grote lijst op te slaan. Hierdoor is het zoeken wel gemakkelijk. Ons eerste idee was om alleen het ID van de bovenliggende reactie op te slaan in een reactie. Het probleem hiervan is, dat je dan moeilijk kan vinden wie de parent is van de parent van deze reactie. Ons tweede idee was om een lijst te maken met alle parentID's. Maar ook dit is weer lastig zoeken. Uiteindelijk vonden we een idee uit het boek MongoDB in action. Hierbij worden alle parentID's van een reactie opgeslagen in één string, met als separator een dubbele punt. Ook wel het pad genoemd. Aan de hand van regex of andere Mongo functies is het gemakkelijk om te zoeken naar een subtree en zijn parents. Tijdens het opbouwen van de boom kun je alle sub-reacties ophalen door alle paths langs te gaan en te kijken of een kind bij deze reactie hoort. En als je dan alle kinderen hebt en voor die kinderen hetzelfde doet, heb je een boom opgebouwd.

Verder geeft de depth aan hoe diep de reactie zit. De depth is gelijk aan het aantal parentID's in het pad. Aan de hand van de depth is het makkelijker om de boomstructuur uit te printen.

De reden dat de comments en beoordeling bij het recept worden bijgehouden heeft te maken met dat wij verwachten dat de recepten lijdend zullen zijn. Gebruikers zullen in een dergelijk systeem een recept kunnen zoeken met de daar bij behorende reacties. Als dit bij de gebruiker wordt opgeslagen zullen eerst alle gebruikers langs gelopen moeten worden voordat de informatie getoond kan worden

# 3 Mogelijke (Toekomstige) verbeteringen

Bij het maken van een database schema is er altijd ruimte voor discussie/verbeteringen. Om het behapbaar houden moeten er keuzes gemaakt worden welke situaties we wel en niet moeten oplossen. In deze paragraaf staan een aantal mogelijke verbeteringen/uitbreidingen voor in de toekomst.

## 3.1 Reactie tijdstip van creatie

Het is mogelijk om als een uitbreiding ook de reactie tijd weer te geven. Al is dit geen veld in de database, je kunt aan de hand van het object id achterhalen wat de tijd van creatie was. Het object id wordt namelijk gecreëerd aan de hand van de tijd van creatie.

## 3.2 User tijdstip van creatie

Uiteraard is het bij de user niet mogelijk om het tijdstip van creatie te achterhalen aan de hand van het object id. Dit komt omdat een user geen object id heeft, maar een username als de waarde van _id. Met andere woorden, als uitbreiding zal er dus een extra veld aangemaakt moeten worden voor het tijdstip van creatie.

## 3.3 Indexen op comments

Het is mogelijk om indexen te plaatsen op de ids en de paths van de reacties. Dit maakt het zoeken op reacties namelijk veel sneller. Wij hebben ervoor gekozen om dit niet te implementeren, omdat wij helemaal niet zoeken op reacties. Als uitbreiding kunnen er dus indexen geplaatst worden op de ids en de paths van de comments als je het mogelijk wil maken om gemakkelijk op comments te kunnen zoeken.

## 3.4 Gerelateerde recepten / Zoeken verbeteren

Wanneer een gebruiker een bepaald recept zoekt kan het voor de gebruiker interessant zijn om een aantal recepten in dezelfde categorie te zien. Dit zou bijvoorbeeld kunnen op ingrediënten, type keuken en moeilijkheidsgraad. Wanneer dit geïmplementeerd word is het dan ook eenvoudig om een geavanceerdere zoek actie mogelijk te maken.

## 3.5 Hoeveelheid

Op basis van de product en hoeveelheid die de gebruiker in huis heeft een recept zoeken. Dit is vrij ingewikkeld omdat er verschillende eenheden van een product gebruikt worden in recepten. Een oplossing hiervoor zou kunnen zijn alles om te rekenen naar een uniforme eenheid.

## 3.6 Inloggen

Op het moment kan een klant wel registreren. Aangezien het niet de opdracht was om ook te kunnen inloggen, is dit een leuke uitbreiding om toe te voegen.

# 4 Uitvoering

## 4.1 Het systeem

Het systeem bestaat uit aantal delen:

- Een invoer-apl. In deze class worden alle invoerquery's uitgevoerd. Met andere woorden deze apl moet voor het zoeken worden uitgevoerd, want anders is er geen test data om op te zoeken.
- Een zoek-apl. Deze class voert alle zoekquery's uit. De invoer-apl moet eerst worden uitgevoerd.
- Een Main-class. Deze zorgt ervoor dat de invoer-apl en de zoek-apl worden uitgevoerd.
- Een databasehelper. Hierin wordt de database connectie aangemaakt en worden query's uitgevoerd. De opbouw van de query's wordt zoveel mogelijk gedaan in de invoer-apl en de zoek-apl.
- Recepten helper. Deze class zorgt er voor dat er eenvoudig een recept document aangemaakt kan worden. We hebben hier voor gekozen omdat anders in de invoer apl heel veel dubbele code zou staan om er voor te zorgen dat de database gevuld word met recepten.

## 4.2 Query's

### 4.2.1 Invoer query's

**AddReview**: Deze methode voegt een beoordeling toe aan een recept. Hierbij worden de gebruikersnaam (user._id) en de review als int opgeslagen in een recept via een update.

**AddUser**: Deze methode voegt een user toe aan de user collectie. Hiervoor is alleen de gebruikersnaam nodig. De gebruikersnaam moet uniek zijn.

**AddLikeWithUpdate**: Deze methode voegt een like toe aan de user die geliked heeft met behulp van een update. Ook wordt het like attribuut van de desbetreffende reactie verhoogd met één. Deze methode kan ook gebruikt worden voor dislikes. Dit kan aangegeven worden door middel van een boolean.

**AddRecipe**: Deze methode voegt een recept toe aan de recept collectie.

**AddComment**: Voegt een reactie toe aan een recept. Niet aan een user. Als er aan deze methode ook een comment_id wordt toegevoegd, wordt er eerst gezocht naar het path van de desbetreffende reactie. Dan wordt ook dit path plus het meegegeven comment_id toegevoegd aan de nieuwe reactie.

### 4.2.2 Zoek query's

**FindTopNRecepten**: Deze methode geeft de beste recepten weer. Deze query kijkt naar de gemiddeld van alle beoordelingen. We hebben er voor gekozen om niet alle velden van een recept terug te geven. Omdat als we dit op een website zouden implementeren we alleen een lijst zouden laten zien met de recept namen en daarbij de gemiddelde beoordeling. We sorteren allereerst op het gemiddelde. Mocht hier een gelijke stand in voor komen hebben we er voor gekozen om degene met de meeste beoordelingen voor te laten gaan. We sturen het aantal beoordeling ook mee richting de cliënt. Deze cliënt zou dan de relevantie kunnen bepalen. Als een recept maar 1 review heeft hoeft dit niet een goed beeld te geven van het recept.

**FindRecpiesByIngredients**: Deze methode zoekt alle recepten waar op zijn minst 1 ingrediënt in voor komt. Deze methode zou bijvoorbeeld gebruikt kunnen worden wanneer een persoon een gerecht met biefstuk of spareribs zoekt.

**FindRecipesByName**: Deze methode voert een query uit waarbij de meegegeven naam in het recept moet zitten. We hebben er voor gekozen om wanneer een gebruiker zoekt op ui kool alle recepten terug te geven met daarin het woord kool. Dit leek ons de meest logische optie. Een gebruiker hoeft dan ook niet de gehele naam van het gerecht in te vullen dat is dan een klein bijkomend voordeel

**FindMinDiffucluty / FindMaxDiffucluty**: Met deze methodes kan een gebruiker recepten zoeken die qua moeilijkheidsgraad bij hun passen. De Max difficulty geeft alle recepten terug met een moeilijkheidsgraad die lager is dan de meegegeven waarde. Deze zal gebruikt worden door mensen met 2 rechter handen in de keuken. Voor de keuken prins(ses)en hebben we ook een methode gemaakt die recepten vanaf een bepaalde moeilijkheidsgraad.

**FindPreparationytime**: Deze is vergelijkbaar met de max en min van de moeilijkheidsgraad. We hebben hiervoor gekozen om zo op basis van de beschikbare tijd een goed recept te kunnen vinden

**FindRecipeByUser**: Deze methode haalt alle recepten op van een user. In een user-object zit namelijk een lijst met recept ids die door de user zijn toegevoegd aan de database. Alle data van een recept op _id na wordt weergegeven.

**FindUserPreferences**: Deze methode pakt de likes uit een user object en zoekt welke recepten daarbij hoorden. Dan worden de favoriete ingrediënten en de favoriete keukens uit die recepten gehaald. Aan de hand van die favorieten wordt het meest geschikte recept tevoorschijn gehaald. Uiteraard is het een leuke uitbreiding om meerdere factoren toe te voegen en meerdere recepten weer te geven.

## 4.3 Indexen

In de tabel hieronder staan alle indexen die gebruikt worden om het zoeken sneller te maken. Ook de verantwoording wordt gegeven.

| | |
|---|---|
| recipe._id | Een recept wordt het vaakst aan de hand van _id opgehaald. Ook staat er automatisch een index op _id, dus een verantwoording is hier eigenlijk niet nodig. |
| recipe.ingredient.name | Omdat het de bedoeling is dat het zoeken op ingrediënten veel gebruikt moet worden, staat er een index op de naam van het ingrediënt. De index staat in alfabetische volgorde, zodat deze gemakkelijk gesorteerd opgehaald kunnen worden. |
| recipe.name | Een recept zal ook via de naam worden opgehaald. Denk bijvoorbeeld aan user die wil zoeken naar een schotel. Een schotel is over het algemeen geen ingrediënt. Daarom wordt er ook een index gezet op de naam van het recept. Ook deze staat in alfabetische volgorde. |
| user._id | Hier staat automatisch een index op. Wij hebben er echter voor gekozen om hierin de username op te slaan in plaats van een Object id. Als dit wel het geval was, zouden de users opgehaald worden aan de hand van _id en de username. Dit kan net zo goed samengevoegd worden aangezien het id maar één waarde toevoegt aan de user. En dat is, dat een Object id een tijdstip bevat. Wij hebben hier niet voor gekozen, omdat dit ook in een apart veld kan als er behoefte aan is. |