Large-Scale and Multi-Structured Databases

BeerHub

Marco Lari, Luca Minuti, Tommaso Pellegrini

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB
Innovation for industry 4.0

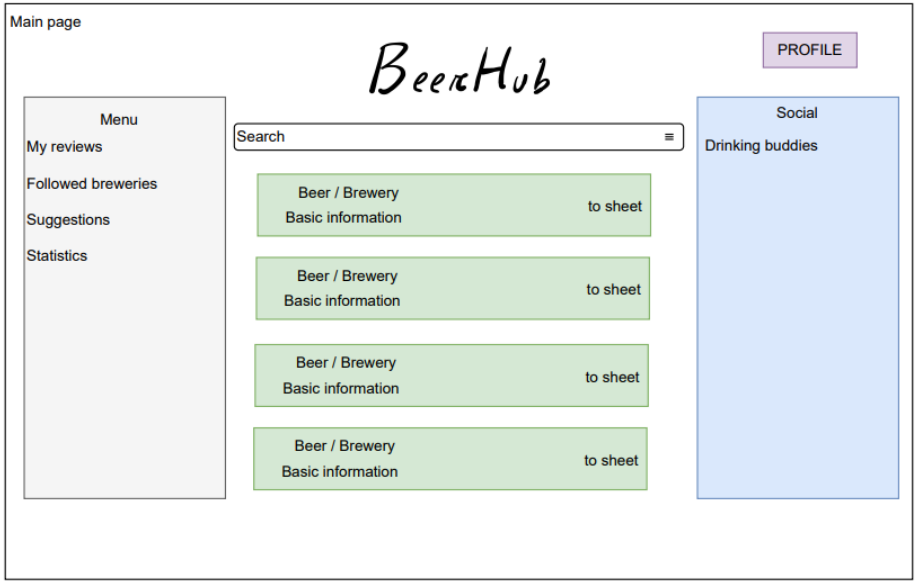
Application Highlights

BeerHub is a data-driven application designed for beer lovers to explore, compare and share experiences about thousands of beers from around the world.

- Browse and filter beers by style, country, brewery and ABV
- View detailed beer profiles enriched with user reviews
- Write and manage personal reviews
- Perform analytics and aggregations on beer, user and review data
- Follow breweries to stay updated on their beers
- Discover new beers and breweries through graph-based recommendations
- Find "drinking buddies" with shared tastes and location

Actors (i)

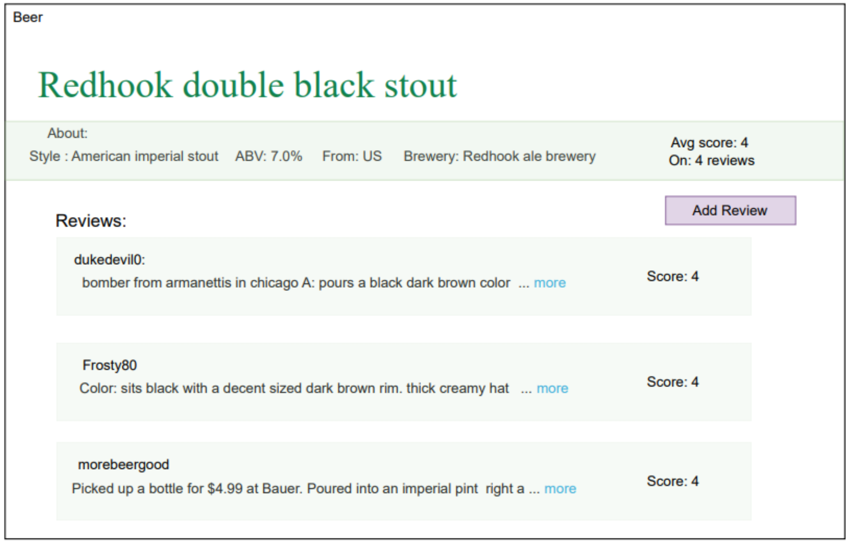
The application has three main actors: Administrator, Registered user and unregistered user.



On the main page any user can access the main functionalities of the application, namely the browsing tab for both beer and breweries as well as a profile tab, where unregistered user can register to become normal users, and users can manage personal information.
A registered user can also access more detailed functions like following breweries, suggestions and statistics.

Actors (ii)

On a beer tab any user can read information and reviews, but only registered user and administrators can write new ones. Administrator also has the possibility to remove reviews if they consider to do so.



Beer

Redhook double black stout

About:
Style : American imperial stout   ABV: 7.0%   From: US   Brewery: Redhook ale brewery

Avg score: 4
On: 4 reviews

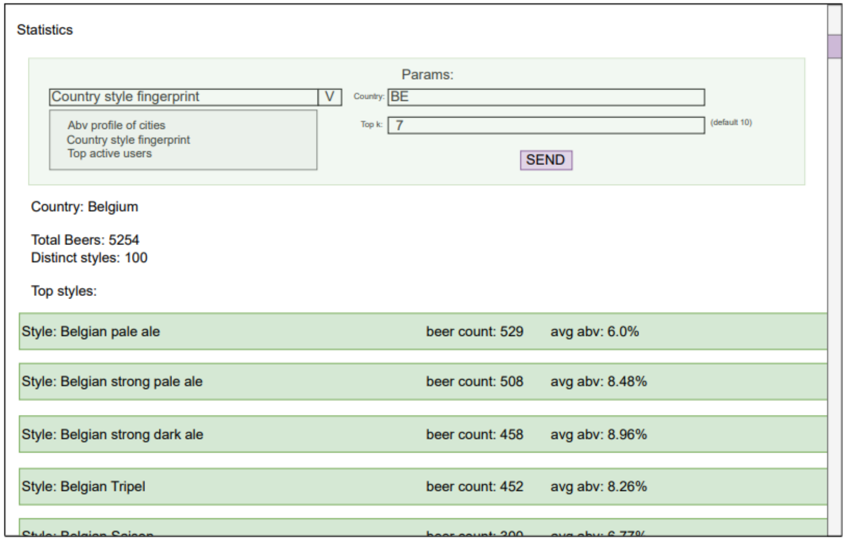
Reviews:

dukedevil0:
bomber from armanettis in chicago A: pours a black dark brown color ... more
Score: 4

Frosty80
Color: sits black with a decent sized dark brown rim. thick creamy hat ... more
Score: 4

morebeergood
Picked up a bottle for $4.99 at Bauer. Poured into an imperial pint right a ... more
Score: 4

Registered user can access all the functionalities of the app, like the statistic functions that show interesting data based on database records and user input.



Statistics

Params:

Country style fingerprint

Country: BE

Top k: 7 (default 10)

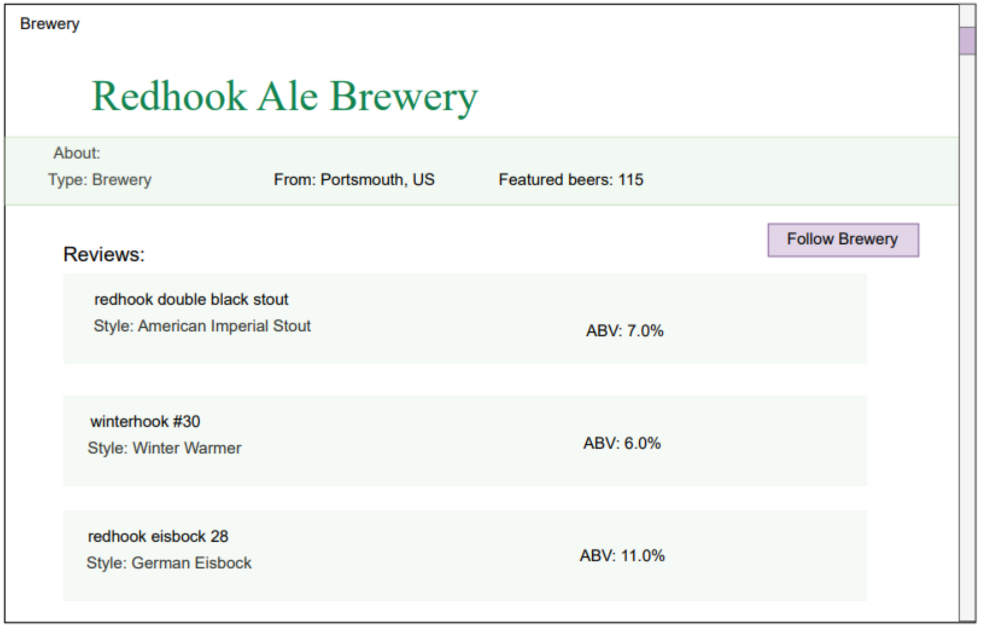
Country: Belgium

Total Beers: 5254
Distinct styles: 100

Top styles:

Style: Belgian pale ale	beer count: 529	avg abv: 6.0%
Style: Belgian strong pale ale	beer count: 508	avg abv: 8.48%
Style: Belgian strong dark ale	beer count: 458	avg abv: 8.96%
Style: Belgian Tripel	beer count: 452	avg abv: 8.26%
Style: Belgian Saison	beer count: 300	avg abv: 6.77%

Actors (iii)



Brewery

Redhook Ale Brewery

About:

Type: Brewery   From: Portsmouth, US   Featured beers: 115

Reviews:

redhook double black stout
Style: American Imperial Stout   ABV: 7.0%

winterhook #30
Style: Winter Warmer   ABV: 6.0%

redhook eisbock 28
Style: German Eisbock   ABV: 11.0%

On the brewery tab everyone can see the specific data; a registered user can decide to follow the brewery to save it in his personal list. An administrator can access all the crud functionalities of the application, which are reserved to him; besides of the register (insert user) and write reviews (insert review).

Dataset Description

Sources: Kaggle-Brewery Dataset, OpenDataBay-Beer Profile, RandomUser API

Description: Dataset integrating information about beers, breweries, reviews and synthetic users. It includes beer attributes such as name, style, ABV, brewery and country; brewery data including identifiers, names and locations; review data with score, text and date; and user profiles with basic demographic information.

Pre-processing: During preprocessing, the two beer datasets were integrated into a single dataset by aligning shared attributes.
Synthetic users were generated and consistently associated with reviews to support user-centric features.

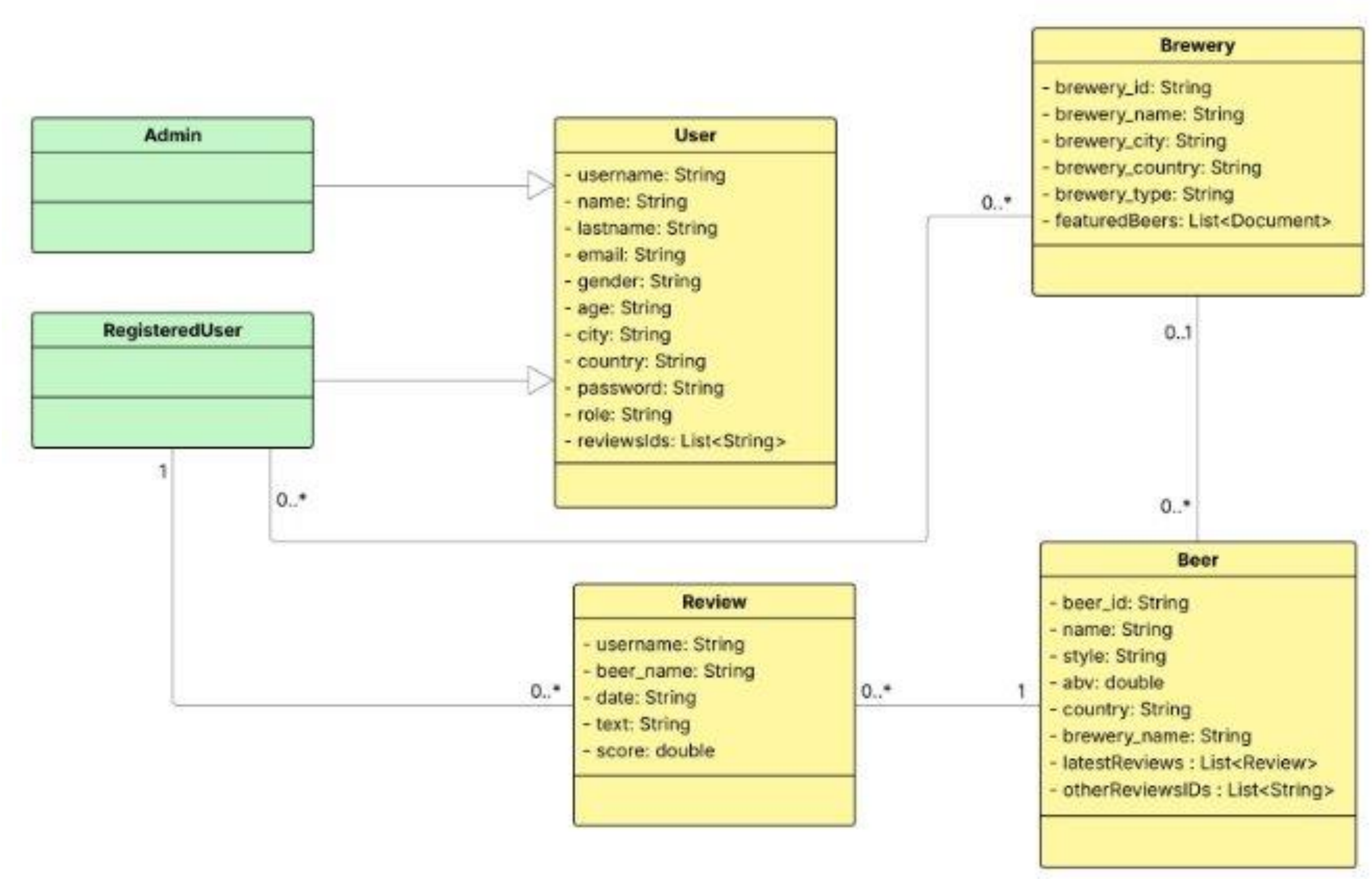
Volume: 28MB (beers) + 3MB (breweries) + 24MB (reviews) + 1MB (users) = 56MB

Variety:

Dataset Description

Variety: Ensured by combining multiple independent sources (Kaggle, OpenDataBay, RandomUser) and by integrating two beer datasets, which required aligning attributes and combining heterogeneous information such as beer details, brewery data, reviews and user demographics.

UML Design Class Diagram



Admin

RegisteredUser

User

- username: String
- name: String
- lastname: String
- email: String
- gender: String
- age: String
- city: String
- country: String
- password: String
- role: String
- reviewsIds: List<String>

Brewery

- brewery_id: String
- brewery_name: String
- brewery_city: String
- brewery_country: String
- brewery_type: String
- featuredBeers: List<Document>

Review

- username: String
- beer_name: String
- date: String
- text: String
- score: double

Beer

- beer_id: String
- name: String
- style: String
- abv: double
- country: String
- brewery_name: String
- latestReviews : List<Review>
- otherReviewsIDs : List<String>

Application non-functional requirements

- The system shall adopt a **multi-database architecture**, combining a document database and a graph database.
- The system must ensure **high availability** and **tolerance to single points of failure**.
- The system shall guarantee **secure handling of user credentials**, using strong password hashing techniques.
- The system shall provide **low latency** for common operations and support a **responsive user experience**.
- The document-oriented data store shall be implemented using **MongoDB**.
- The graph-based data store shall be implemented using **Neo4j**.

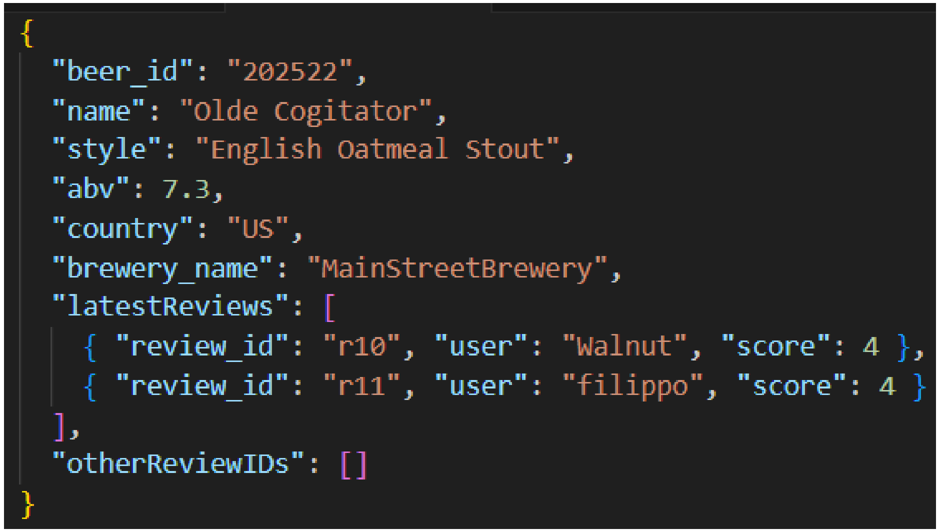
CAP Theorem

BeerHub operates in a distributed environment where network partitions must be assumed.

To satisfy non-functional requirements such as high availability and low latency, the system adopts an availability-oriented (AP) design.

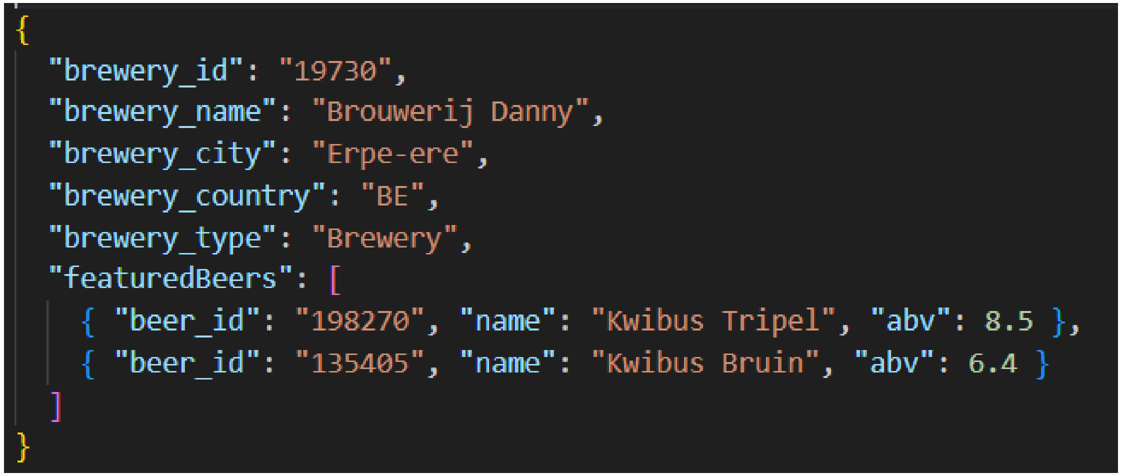
The application prioritizes remaining responsive and accessible under partial failures, accepting eventual consistency for derived and non-critical data such as aggregated statistics and rankings.

Document DB Design



```
{
  "beer_id": "202522",
  "name": "Olde Cogitator",
  "style": "English Oatmeal Stout",
  "abv": 7.3,
  "country": "US",
  "brewery_name": "MainStreetBrewery",
  "latestReviews": [
    { "review_id": "r10", "user": "Walnut", "score": 4 },
    { "review_id": "r11", "user": "filippo", "score": 4 }
  ],
  "otherReviewIDs": []
}
```

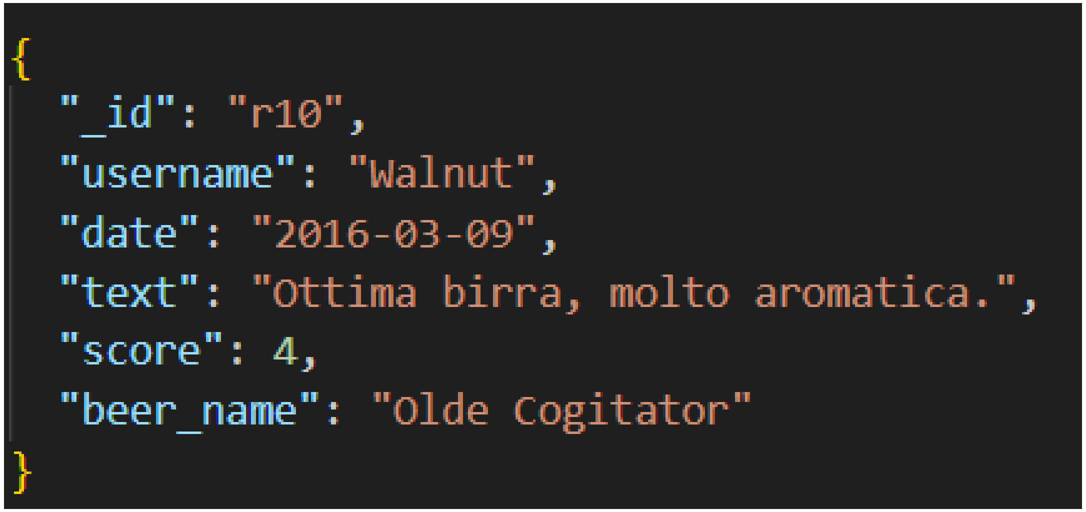
Beer collection



```
{
  "brewery_id": "19730",
  "brewery_name": "Brouwerij Danny",
  "brewery_city": "Erpe-ere",
  "brewery_country": "BE",
  "brewery_type": "Brewery",
  "featuredBeers": [
    { "beer_id": "198270", "name": "Kwibus Tripel", "abv": 8.5 },
    { "beer_id": "135405", "name": "Kwibus Bruin", "abv": 6.4 }
  ]
}
```

Brewery collection

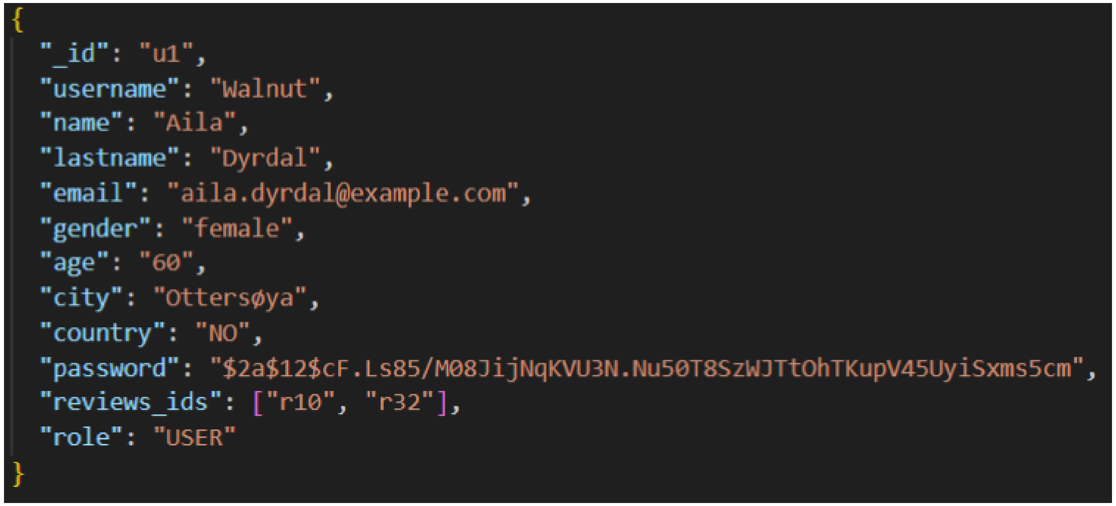
Document DB Design



```
{
  "_id": "r10",
  "username": "Walnut",
  "date": "2016-03-09",
  "text": "Ottima birra, molto aromatica.",
  "score": 4,
  "beer_name": "Olde Cogitator"
}
```

Review collection

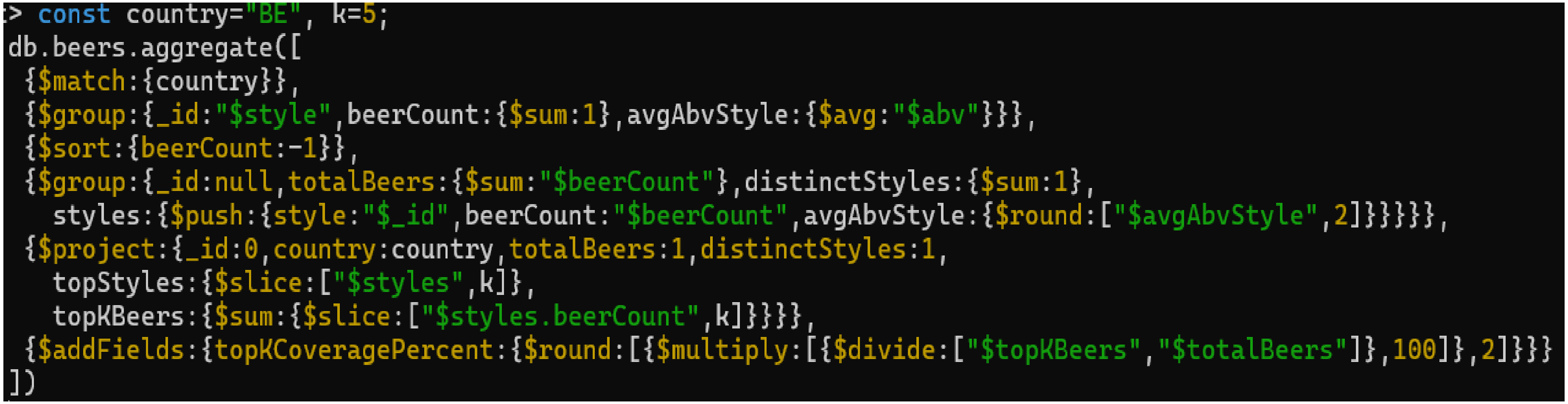
User collection



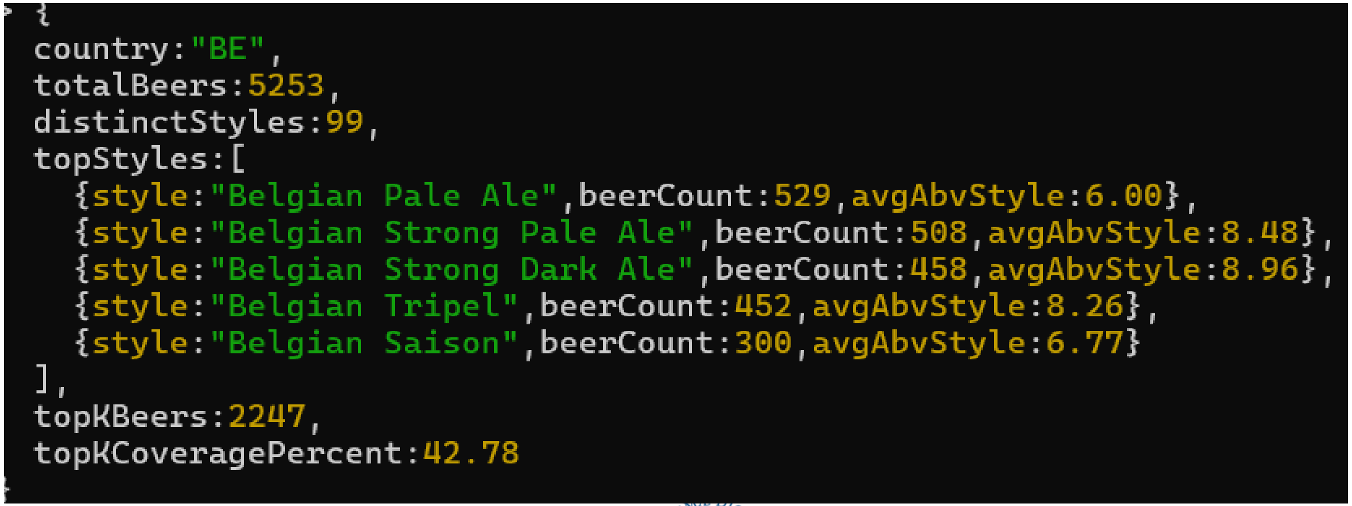
```
{
  "_id": "u1",
  "username": "Walnut",
  "name": "Aila",
  "lastname": "Dyrdal",
  "email": "aila.dyrdal@example.com",
  "gender": "female",
  "age": "60",
  "city": "Otters\u00f8ya",
  "country": "NO",
  "password": "$2a$12$cF.Ls85/M08JijNqKVU3N.Nu50T8SzWJTtOhTKupV45UyiSxms5cm",
  "reviews_ids": ["r10", "r32"],
  "role": "USER"
}
```

Document DB Design

Country Beer Styles Fingerprint (analytic)



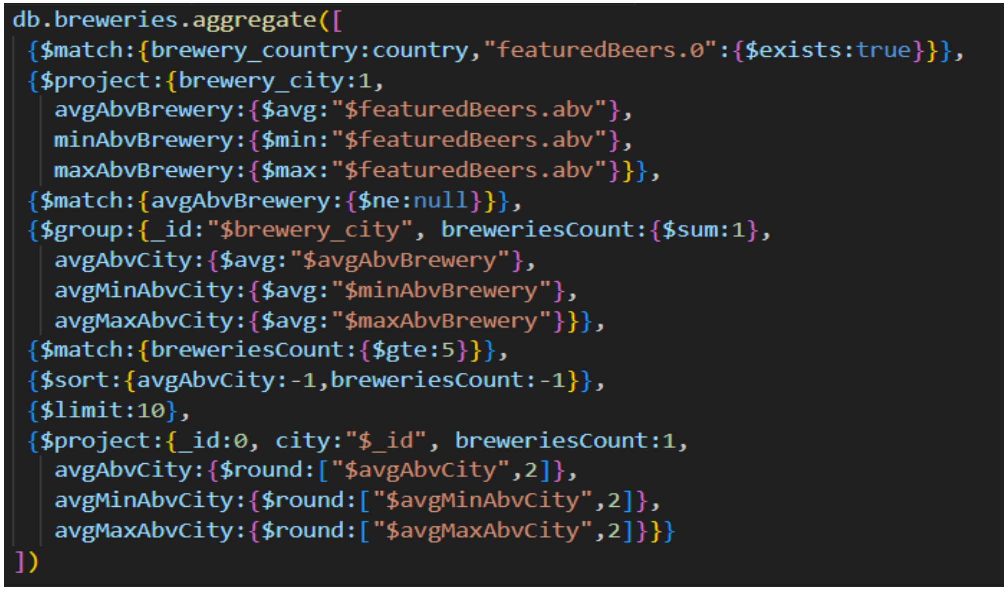
```
> const country="BE", k=5;
db.beers.aggregate([
 {$match:{country}},
 {$group:{_id:"$style",beerCount:{$sum:1},avgAbvStyle:{$avg:"$abv"}}},
 {$sort:{beerCount:-1}},
 {$group:{_id:null,totalBeers:{$sum:"$beerCount"},distinctStyles:{$sum:1},
   styles:{$push:{style:"$_id",beerCount:"$beerCount",avgAbvStyle:{$round:["$avgAbvStyle",2]}}}}},
 {$project:{_id:0,country:country,totalBeers:1,distinctStyles:1,
   topStyles:{$slice:["$styles",k]},
   topKBeers:{$sum:{$slice:["$styles.beerCount",k]}}}},
 {$addFields:{topKCoveragePercent:{$round:[{$multiply:[{$divide:["$topKBeers","$totalBeers"]},100]},2]}}}
])
```



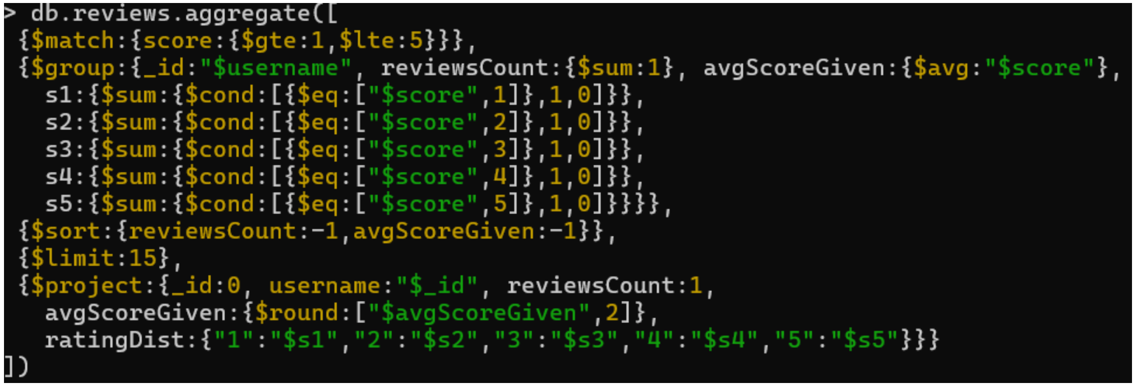
```
{
 country:"BE",
 totalBeers:5253,
 distinctStyles:99,
 topStyles:[
   {style:"Belgian Pale Ale",beerCount:529,avgAbvStyle:6.00},
   {style:"Belgian Strong Pale Ale",beerCount:508,avgAbvStyle:8.48},
   {style:"Belgian Strong Dark Ale",beerCount:458,avgAbvStyle:8.96},
   {style:"Belgian Tripel",beerCount:452,avgAbvStyle:8.26},
   {style:"Belgian Saison",beerCount:300,avgAbvStyle:6.77}
 ],
 topKBeers:2247,
 topKCoveragePercent:42.78
}
```

Document DB Design

Top Cities by Brewery Alcoholic Strenght Profile (analytic)



```
db.breweries.aggregate([
 {$match:{brewery_country:country,"featuredBeers.0":{$exists:true}}},
 {$project:{brewery_city:1,
   avgAbvBrewery:{$avg:"$featuredBeers.abv"},
   minAbvBrewery:{$min:"$featuredBeers.abv"},
   maxAbvBrewery:{$max:"$featuredBeers.abv"}}},
 {$match:{avgAbvBrewery:{$ne:null}}},
 {$group:{_id:"$brewery_city", breweriesCount:{$sum:1},
   avgAbvCity:{$avg:"$avgAbvBrewery"},
   avgMinAbvCity:{$avg:"$minAbvBrewery"},
   avgMaxAbvCity:{$avg:"$maxAbvBrewery"}}},
 {$match:{breweriesCount:{$gte:5}}},
 {$sort:{avgAbvCity:-1,breweriesCount:-1}},
 {$limit:10},
 {$project:{_id:0, city:"$_id", breweriesCount:1,
   avgAbvCity:{$round:["$avgAbvCity",2]},
   avgMinAbvCity:{$round:["$avgMinAbvCity",2]},
   avgMaxAbvCity:{$round:["$avgMaxAbvCity",2]}}}
])
```

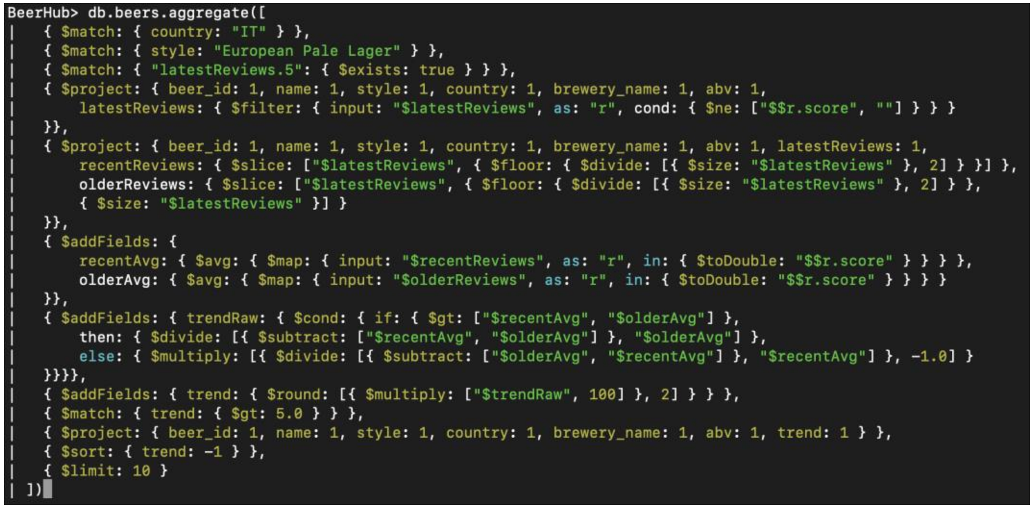


```
> db.reviews.aggregate([
 {$match:{score:{$gte:1,$lte:5}}},
 {$group:{_id:"$username", reviewsCount:{$sum:1}, avgScoreGiven:{$avg:"$score"},
   s1:{$sum:{$cond:[{$eq:["$score",1]},1,0]}},
   s2:{$sum:{$cond:[{$eq:["$score",2]},1,0]}},
   s3:{$sum:{$cond:[{$eq:["$score",3]},1,0]}},
   s4:{$sum:{$cond:[{$eq:["$score",4]},1,0]}},
   s5:{$sum:{$cond:[{$eq:["$score",5]},1,0]}}}},
 {$sort:{reviewsCount:-1,avgScoreGiven:-1}},
 {$limit:15},
 {$project:{_id:0, username:"$_id", reviewsCount:1,
   avgScoreGiven:{$round:["$avgScoreGiven",2]},
   ratingDist:{"1":"$s1","2":"$s2","3":"$s3","4":"$s4","5":"$s5"}}}
])
```

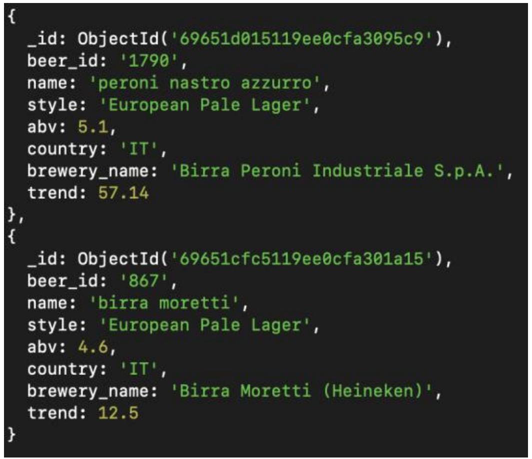
Top 15 Community Contributors & Rating Profiles (analytic)

Document DB Design

Beer Popularity Trend (analytic)



```
BeerHub> db.beers.aggregate([
|   { $match: { country: "IT" } },
|   { $match: { style: "European Pale Lager" } },
|   { $match: { "latestReviews.5": { $exists: true } } },
|   { $project: { beer_id: 1, name: 1, style: 1, country: 1, brewery_name: 1, abv: 1,
|       latestReviews: { $filter: { input: "$latestReviews", as: "r", cond: { $ne: ["$$r.score", ""] } } }
|   }},
|   { $project: { beer_id: 1, name: 1, style: 1, country: 1, brewery_name: 1, abv: 1, latestReviews: 1,
|       recentReviews: { $slice: ["$latestReviews", { $floor: { $divide: [{ $size: "$latestReviews" }, 2] } }] },
|       olderReviews: { $slice: ["$latestReviews", { $floor: { $divide: [{ $size: "$latestReviews" }, 2] } },
|       { $size: "$latestReviews" }] }
|   }},
|   { $addFields: {
|       recentAvg: { $avg: { $map: { input: "$recentReviews", as: "r", in: { $toDouble: "$$r.score" } } } },
|       olderAvg: { $avg: { $map: { input: "$olderReviews", as: "r", in: { $toDouble: "$$r.score" } } } }
|   }},
|   { $addFields: { trendRaw: { $cond: { if: { $gt: ["$recentAvg", "$olderAvg"] },
|       then: { $divide: [{ $subtract: ["$recentAvg", "$olderAvg"] }, "$olderAvg"] },
|       else: { $multiply: [{ $divide: [{ $subtract: ["$olderAvg", "$recentAvg"] }, "$recentAvg"] }, -1.0] }
|   }}}},
|   { $addFields: { trend: { $round: [{ $multiply: ["$trendRaw", 100] }, 2] } } },
|   { $match: { trend: { $gt: 5.0 } } },
|   { $project: { beer_id: 1, name: 1, style: 1, country: 1, brewery_name: 1, abv: 1, trend: 1 } },
|   { $sort: { trend: -1 } },
|   { $limit: 10 }
| ])
```



```
{
  _id: ObjectId('69651d015119ee0cfa3095c9'),
  beer_id: '1790',
  name: 'peroni nastro azzurro',
  style: 'European Pale Lager',
  abv: 5.1,
  country: 'IT',
  brewery_name: 'Birra Peroni Industriale S.p.A.',
  trend: 57.14
},
{
  _id: ObjectId('69651cfc5119ee0cfa301a15'),
  beer_id: '867',
  name: 'birra moretti',
  style: 'European Pale Lager',
  abv: 4.6,
  country: 'IT',
  brewery_name: 'Birra Moretti (Heineken)',
  trend: 12.5
}
```

MongoDB Replica Set Configuration

The BeerHub infrastructure is distributed on three VMs, on which are deployed three replicas for the Mongo DB, one for each machine.

With the replication we aim to support the AP approach chosen for the system, ensuring even more responsiveness at any time, at the cost of consistency, which is ensured but only eventually. All this is in line with what was specified in the non-functional requirements presented:

The system must ensure high availability, and tolerance to single point of failure.

The parameters we have chosen remain in line with the approach of total availability

MongoDB Replica parameters

- **Write concern (w = 1), journal = true:** Acknowledgement is required only from the Primary node. This minimizes write latency and ensures high application availability by decoupling the client response from the replication process to secondary nodes.

- **Read preference (Nearest):** Queries are routed to the node with the lowest network latency. Since BeerHub is a read-heavy application, this optimizes response times and distributes the load across the entire cluster.

- **Read Concern (Local):** Used in conjunction with *Nearest* to further reduce overhead. We prioritize system responsiveness and low latency over strict data consistency, which is acceptable for the application's browsing and analytical workloads.

MongoDB Indexes

Users

- **username_1**: to speed up user authentication or profile retrieval. Having a unique index on username ensures no duplication of usernames within the collection.
- **country_1**: frequently used for queries of searching users by their country.

Beers

- **beer_id_1**: it ensures fast retrieval of documents related to a specific beer, particularly useful when querying individual beers.

- **name_text**: text index on the name field, it improves search operations that involve beer names queries, such as searching for beers by name, making queries on name faster and more efficient.

- **country_1_style_1** (compound index): it optimizes queries that filter by country and style field. Designed to improve search operations by country and aggregations (s.a. Beer Popoularity Trend)

- **style_1**: useful for queries filtering by beer style. Designed to improve search operations and “Beer Popularity Trend” aggregate.

country_1style_1	Without index	With Index
nReturned	2	2
executionTimeMillis	293	2
totalKeysExamined	0	143
totalDocsExamined	359231	143

style_1	Without index	With Index
nReturned	13	13
executionTimeMillis	203	12
totalKeysExamined	0	3715
totalDocsExamined	359231	3715

MongoDB Indexes

Breweries

- **brewery_id_1**: It ensures that the brewery identifier is unique and useful for fast queries that retrieve information based on the brewery ID.

- **brewery_name_text** : It allows faster brewery names search

- **brewery_country_1**: Fundamental for queries that filter or sort breweries by country, like "Top cities by brewery alcoholic strength", ensuring quick access to brewery data based on geographic location.

Reviews

- **score_1_username_1** (compound index): It improves performance of "Top active users" aggregation query.

- **beer_name_1**: it enhances performance for searching beer reviews by their name queries, such as the calculation of the average score of a beer.

brewery_country_1	Without index	With Index
nReturned	376	376
executionTimeMillis	66	10
totalKeysExamined	0	581
totalDocsExamined	50349	581

score_1_username_1	Without index	With Index
nReturned	10916	10916
executionTimeMillis	175	140
totalKeysExamined	0	36553
totalDocsExamined	39890	0

Discussion on MongoDB Data Sharding

Sharded collections

Reviews Collection

- Shard Key: _id (hashed)
- Reason: Uniform load distribution, not query optimization, since it is the fastest-growing collection in the system
- Benefit: Evenly distributed inserts, avoiding hotspots

Beers Collection

- Shard Key: _id (hashed)
- Reason: Large collection requiring even data distribution
- Benefit: Balanced workload across cluster

Non Sharded collections

Breweries and Users collection are not sharded beacause of

- Small size and limited growth rate
- Sharding would add unnecessary overhead
- Single shard sufficient for performance
- Can be sharded later if needed

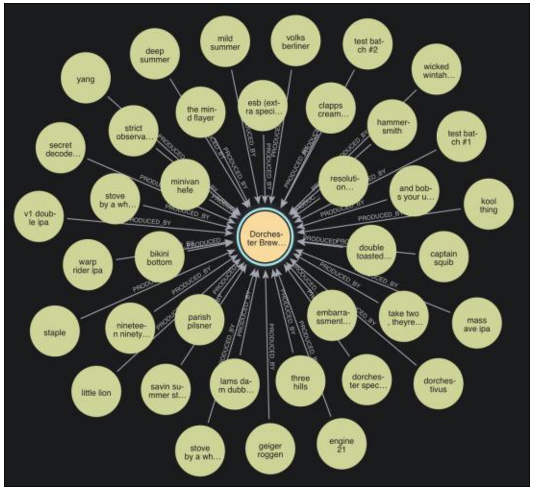
Graph DB Design

Nodes:

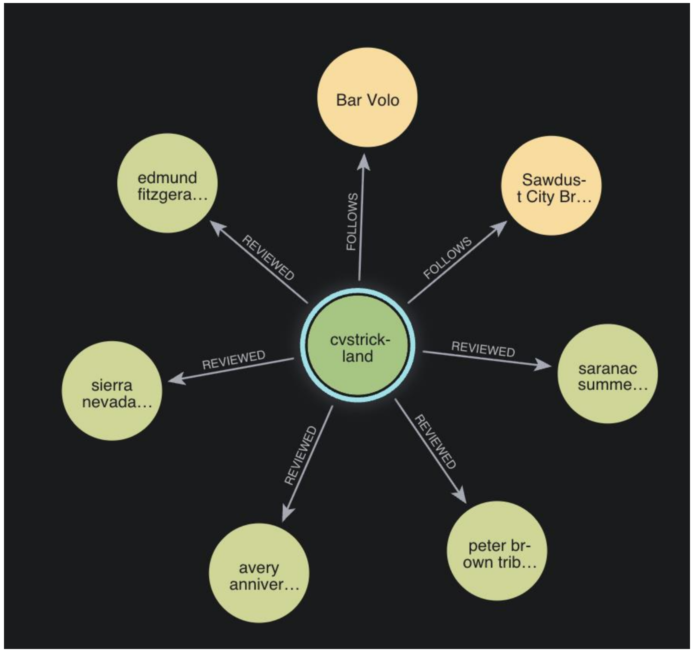
- User {username, city, country, email}
- Beer {beer_id, name, style}
- Brewery{brewery_id, name, city, country}

Relationships:

- User – FOLLOWS → Brewery
- User – REVIEWED → Beer {score}
- Beer – PRODUCED_BY → Brewery



Sample of beers produced by a brewery



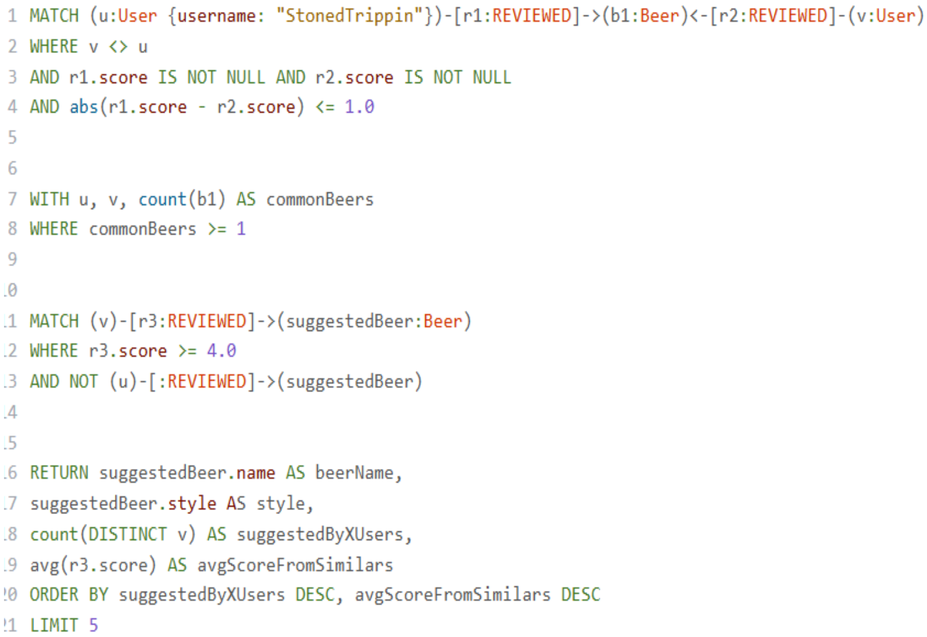
Sample of user following breweries and reviewed beers

Graph DB Design

Personalized Beer Recommendations

Algorithm Logic:

- Identify Peers: Find users who reviewed the same beers with similar scores (difference ≤1)
- Filter Candidates: Extract highly-rated beers (score ≥4) not yet reviewed by the user
- Rank & Score: Weight by cluster popularity and perceived quality



```
MATCH (u:User {username: "StonedTrippin"})-[r1:REVIEWED]->(b1:Beer)<-[r2:REVIEWED]-(v:User)
WHERE v <> u
AND r1.score IS NOT NULL AND r2.score IS NOT NULL
AND abs(r1.score - r2.score) <= 1.0


WITH u, v, count(b1) AS commonBeers
WHERE commonBeers >= 1


MATCH (v)-[r3:REVIEWED]->(suggestedBeer:Beer)
WHERE r3.score >= 4.0
AND NOT (u)-[:REVIEWED]->(suggestedBeer)


RETURN suggestedBeer.name AS beerName,
suggestedBeer.style AS style,
count(DISTINCT v) AS suggestedByXUsers,
avg(r3.score) AS avgScoreFromSimilars
ORDER BY suggestedByXUsers DESC, avgScoreFromSimilars DESC
LIMIT 5
```

Input: username (unique user identifier)

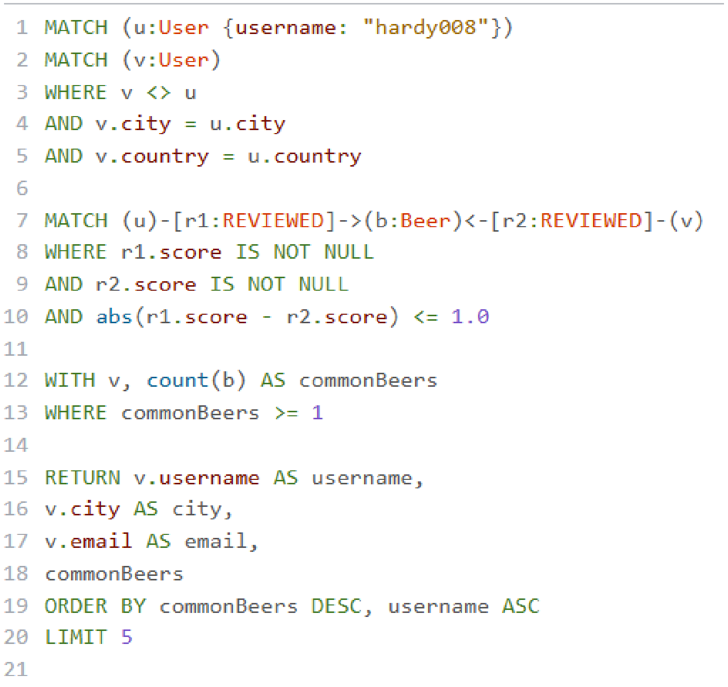
Output: Top 5 recommended beers with name, style, number of similar users who recommended it and the average rating from the peer group

Graph DB Design

Drinking Buddies

Algorithm Logic:

- Geographic Filtering: constraint on city AND country for precise location matching
- Taste Affinity: identifies local users with similar ratings (score difference ≤1)
- Connection Ranking: prioritizes users with the highest count of overlapping beer interests



```
MATCH (u:User {username: "hardy008"})
MATCH (v:User)
WHERE v <> u
AND v.city = u.city
AND v.country = u.country

MATCH (u)-[r1:REVIEWED]->(b:Beer)<-[r2:REVIEWED]-(v)
WHERE r1.score IS NOT NULL
AND r2.score IS NOT NULL
AND abs(r1.score - r2.score) <= 1.0

WITH v, count(b) AS commonBeers
WHERE commonBeers >= 1

RETURN v.username AS username,
v.city AS city,
v.email AS email,
commonBeers
ORDER BY commonBeers DESC, username ASC
LIMIT 5

```

Input: username (logged-in user)

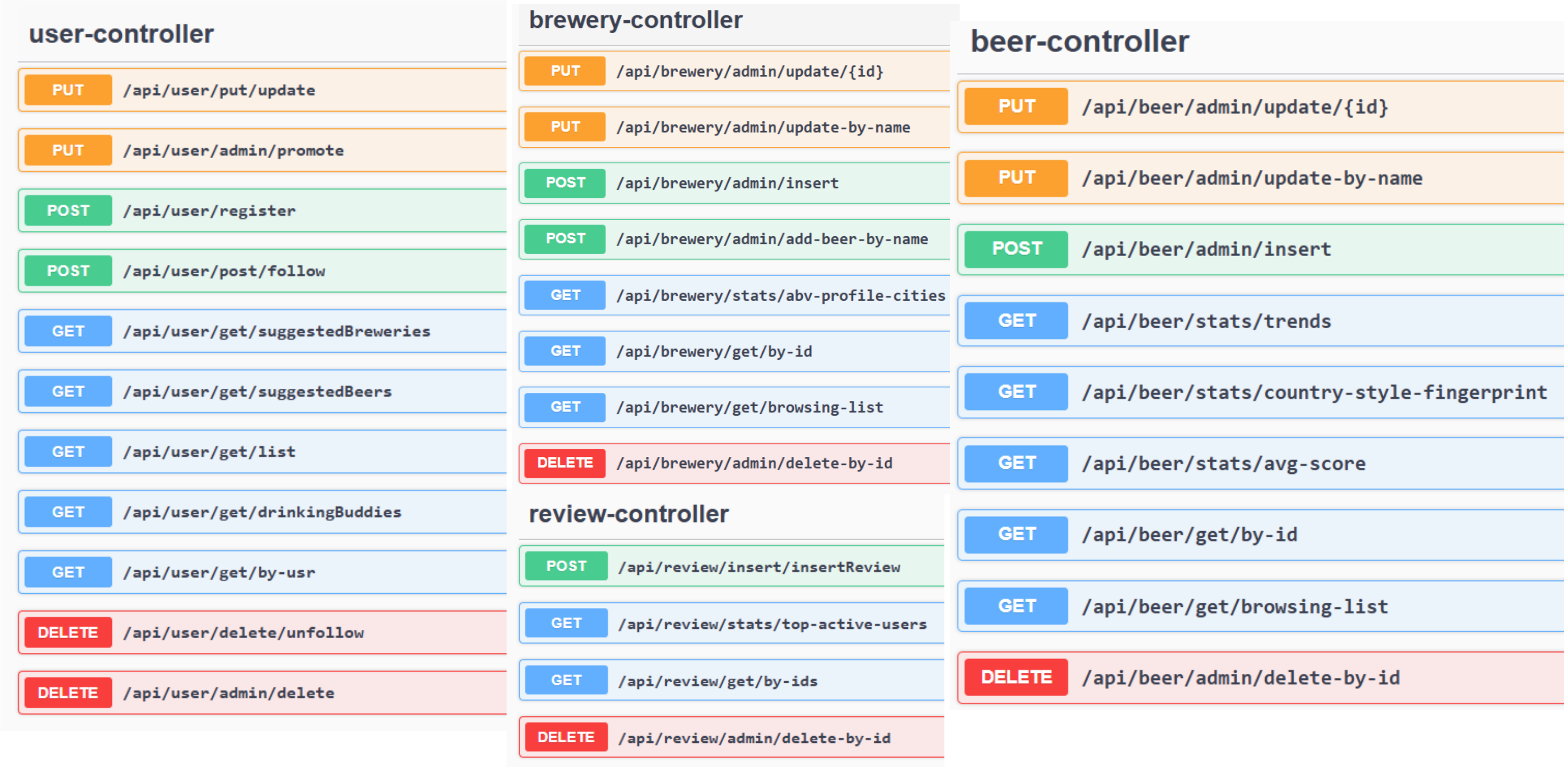
Output: contact information (username and email), city and number of commonBeers

Handling Intra-DB Consistency

- **Intra-database consistency (MongoDB)**
- Managed at **application level**
- Each operation explicitly updates all related documents and redundancies
- Ensures alignment among related collections
- **Cross-database consistency (MongoDB & Neo4j)**
- Also managed by the **application**
- Corresponding updates are executed on both databases within the same service-level operation
- **No distributed transactions**
- MongoDB and Neo4j are not synchronized via global transactions
- Temporary inconsistencies may occur in case of partial failures
- **Consistency model and rationale**
- Such inconsistencies are **tolerated and handled at application level**
- MongoDB retains the **complete and authoritative data representation**
- Neo4j maintains a **relationship-oriented view** for traversal and recommendation queries

Swagger UI REST APIs documentation

On swagger we have the endpoints relative to the main entities, CRUDs and all the functionalities proposed by the application



user-controller

PUT /api/user/put/update

PUT /api/user/admin/promote

POST /api/user/register

POST /api/user/post/follow

GET /api/user/get/suggestedBreweries

GET /api/user/get/suggestedBeers

GET /api/user/get/list

GET /api/user/get/drinkingBuddies

GET /api/user/get/by-usr

DELETE /api/user/delete/unfollow

DELETE /api/user/admin/delete

brewery-controller

PUT /api/brewery/admin/update/{id}

PUT /api/brewery/admin/update-by-name

POST /api/brewery/admin/insert

POST /api/brewery/admin/add-beer-by-name

GET /api/brewery/stats/abv-profile-cities

GET /api/brewery/get/by-id

GET /api/brewery/get/browsing-list

DELETE /api/brewery/admin/delete-by-id

review-controller

POST /api/review/insert/insertReview

GET /api/review/stats/top-active-users

GET /api/review/get/by-ids

DELETE /api/review/admin/delete-by-id

beer-controller

PUT /api/beer/admin/update/{id}

PUT /api/beer/admin/update-by-name

POST /api/beer/admin/insert

GET /api/beer/stats/trends

GET /api/beer/stats/country-style-fingerprint

GET /api/beer/stats/avg-score

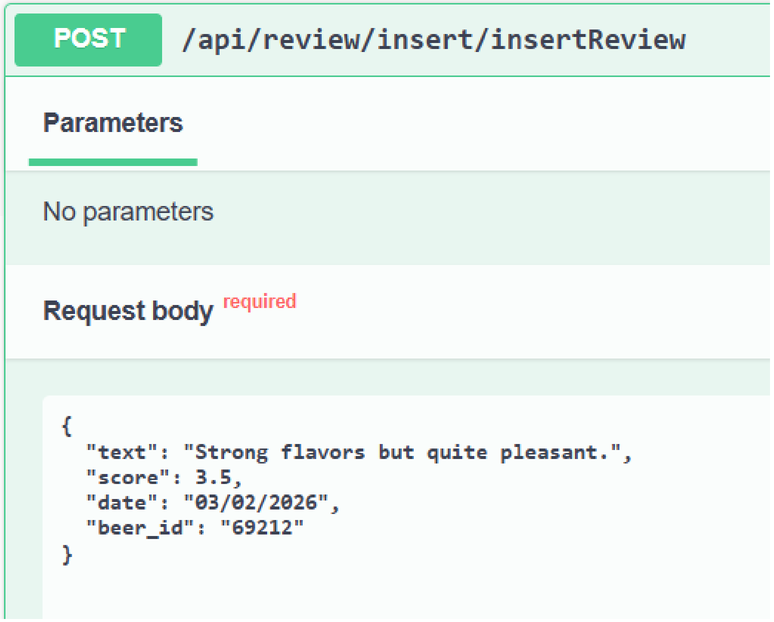
GET /api/beer/get/by-id

GET /api/beer/get/browsing-list

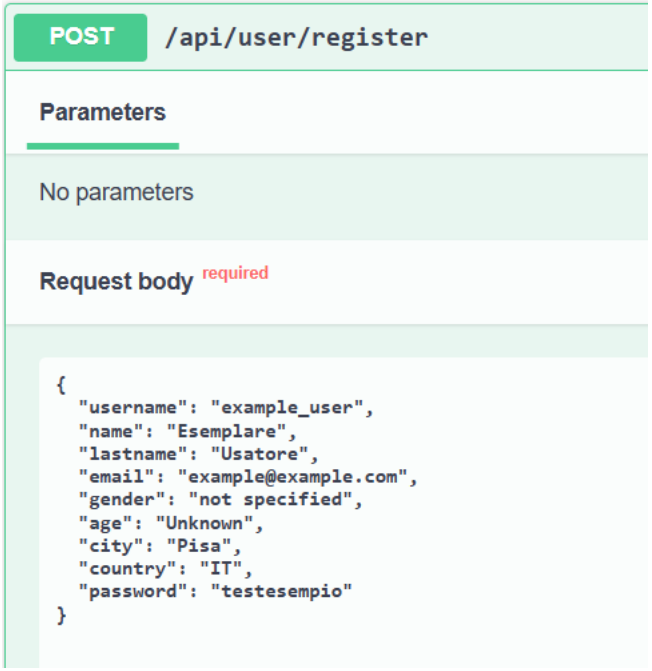
DELETE /api/beer/admin/delete-by-id

Swagger UI REST APIs documentation

Most endpoints require parameters, that can be in various form; for insert or update operation is mostly in document form:



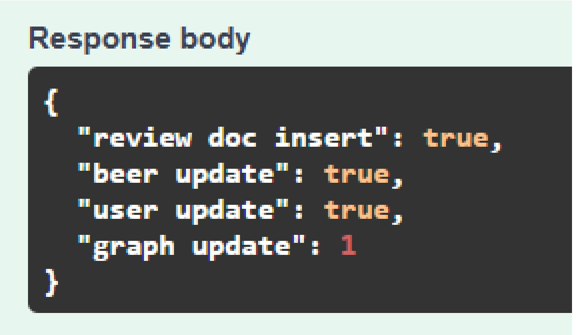
```
{
  "text": "Strong flavors but quite pleasant.",
  "score": 3.5,
  "date": "03/02/2026",
  "beer_id": "69212"
}
```



```
{
  "username": "example_user",
  "name": "Esemplare",
  "lastname": "Usatore",
  "email": "example@example.com",
  "gender": "not specified",
  "age": "Unknown",
  "city": "Pisa",
  "country": "IT",
  "password": "testesempio"
}
```

Swagger UI REST APIs documentation

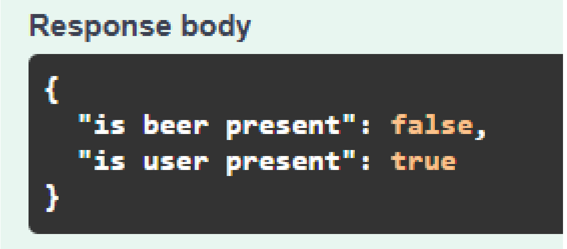
Response are either documents with operation informations or string responses



Response body

```
{
  "review doc insert": true,
  "beer update": true,
  "user update": true,
  "graph update": 1
}
```

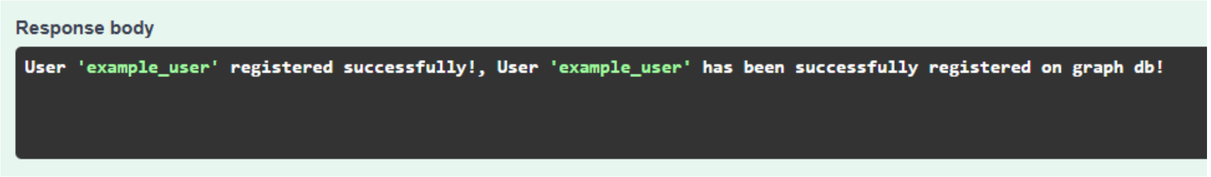
Correct review insert



Response body

```
{
  "is beer present": false,
  "is user present": true
}
```

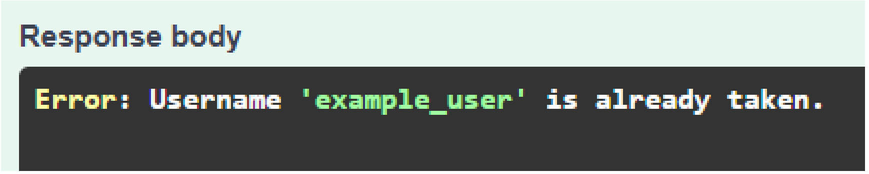
Review insert on a unexisting beer



Response body

```
User 'example_user' registered successfully!, User 'example_user' has been successfully registered on graph db!
```

Succesful register

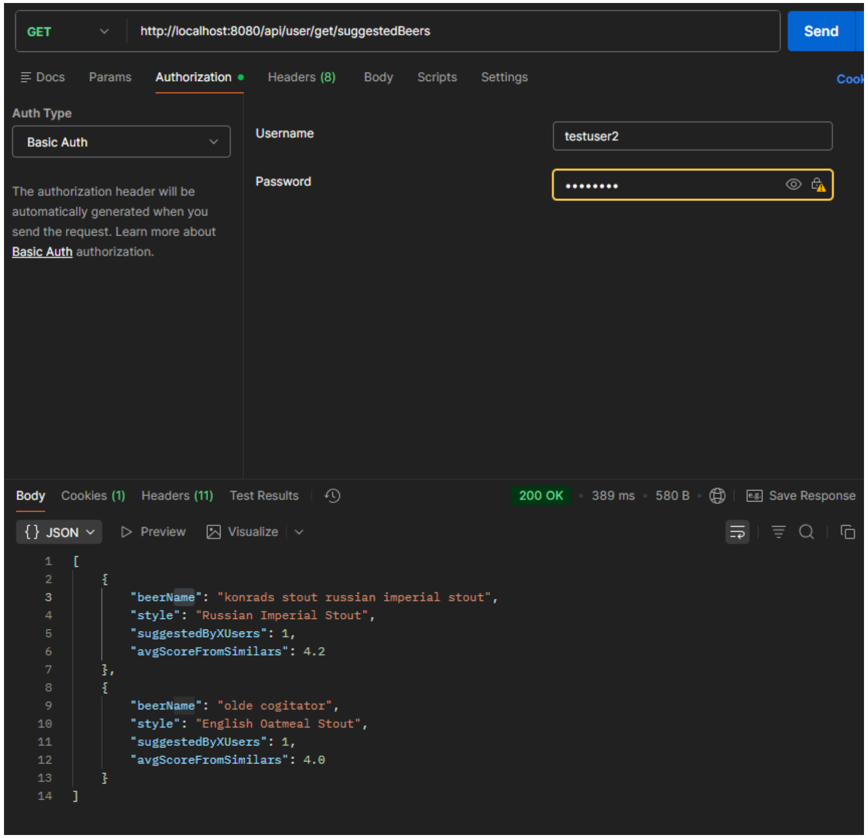


Response body

```
Error: Username 'example_user' is already taken.
```

Unsuccesfull

Live Demo with Postman



GET http://localhost:8080/api/user/get/suggestedBeers

Auth Type: Basic Auth

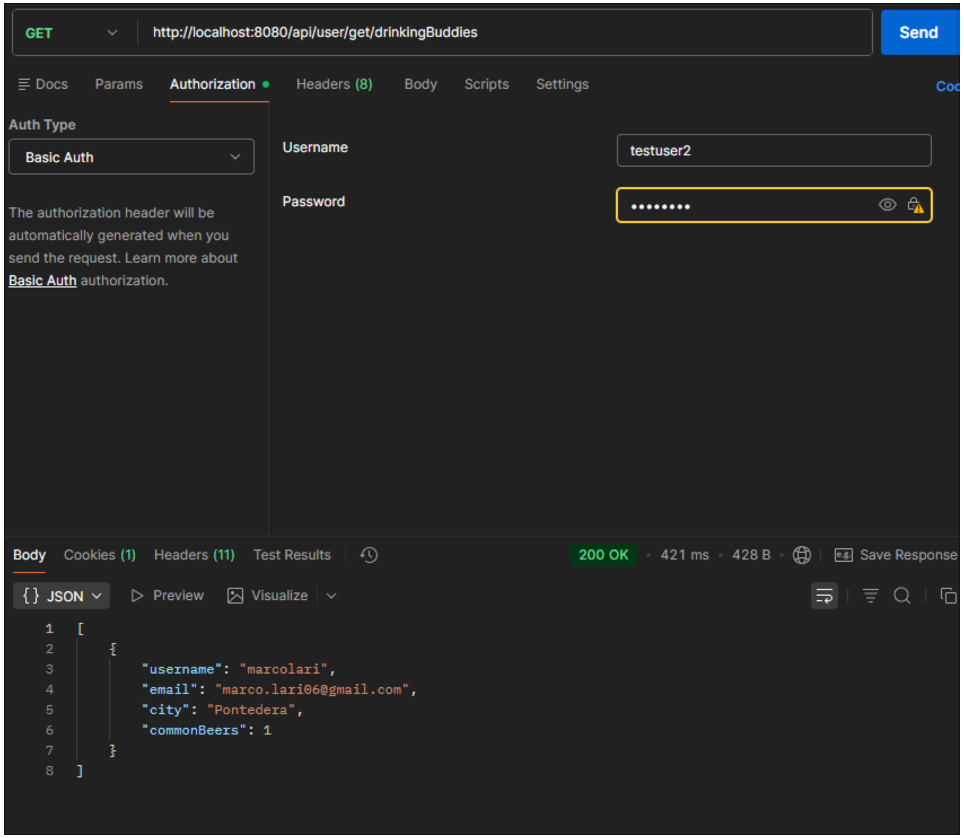
Username: testuser2

Password: ••••••••

The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.

200 OK · 389 ms · 580 B

```
[
    {
        "beerName": "konrads stout russian imperial stout",
        "style": "Russian Imperial Stout",
        "suggestedByXUsers": 1,
        "avgScoreFromSimilars": 4.2
    },
    {
        "beerName": "olde cogitator",
        "style": "English Oatmeal Stout",
        "suggestedByXUsers": 1,
        "avgScoreFromSimilars": 4.0
    }
]
```



GET http://localhost:8080/api/user/get/drinkingBuddies

Auth Type: Basic Auth

Username: testuser2

Password: ••••••••

The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.

200 OK · 421 ms · 428 B

```
[
    {
        "username": "marcolari",
        "email": "marco.lari06@gmail.com",
        "city": "Pontedera",
        "commonBeers": 1
    }
]
```

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

IN SUPREMÆ DIGNITATIS 1343

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0