Large-Scale and Multi-Structured Databases

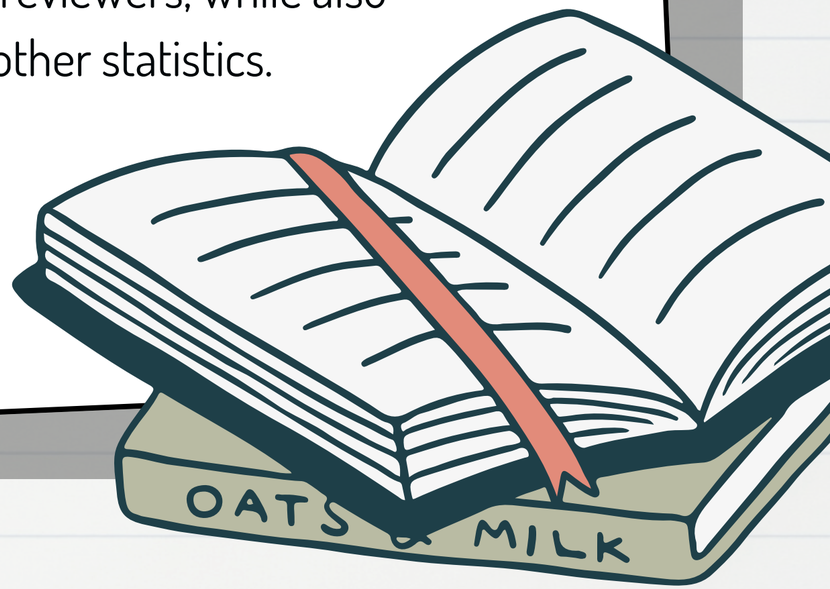
BookAdvisor

Daniele Giaquinta
Francesco Londretti
Giulio Bello

Idea

The idea of the project is to create a social application concerning books. Reviewers can leave their opinion on books, search for them with special criteria, check other reviewers' opinions, agree and disagree with them and also follow other users to check recent activity and get customized recommendations for books and users.

Authors can upload and share their works and interact the same way as reviewers, while also getting the ability to keep an eye on the appreciation of their books and other statistics.



Functional Requirements

Unsigned User

- Log in
- Sign up
- Browse books
- Search book
- Find book
  - View book info

Reviewer

- All actions of a Signed User
- View personal reviews
  - Edit personal review
  - Delete review
- In view book info:
  - Rate the book
  - Review the book
  - Browse book reviews
    - Upvote/Downvote user ratings
- View personal account
  - Update account information
  - Delete personal account
  - View personal reviews
    - Edit personal review
- Search user
- Browse users
  - Find user
    - View user profile
      - Browse user reviews
      - Follow user
- View Feed
  - Get Book Recommendations
  - Get User Recommendations

Author

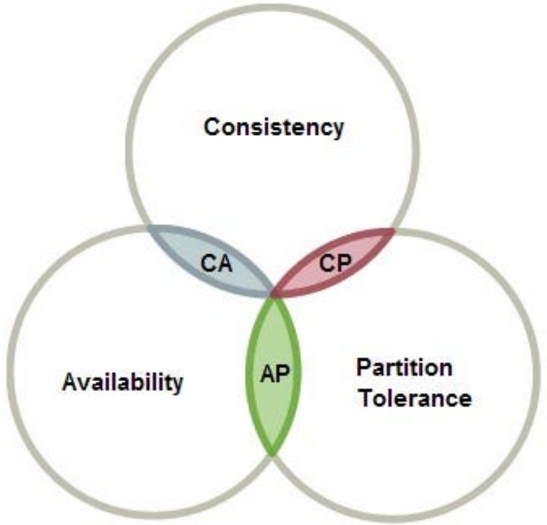
- All actions of a Reviewer
- Upload a book
- Delete a book
- View Stats

Admin

- All actions of a Reviewer
- Delete any book
- Delete any user
- Delete any review

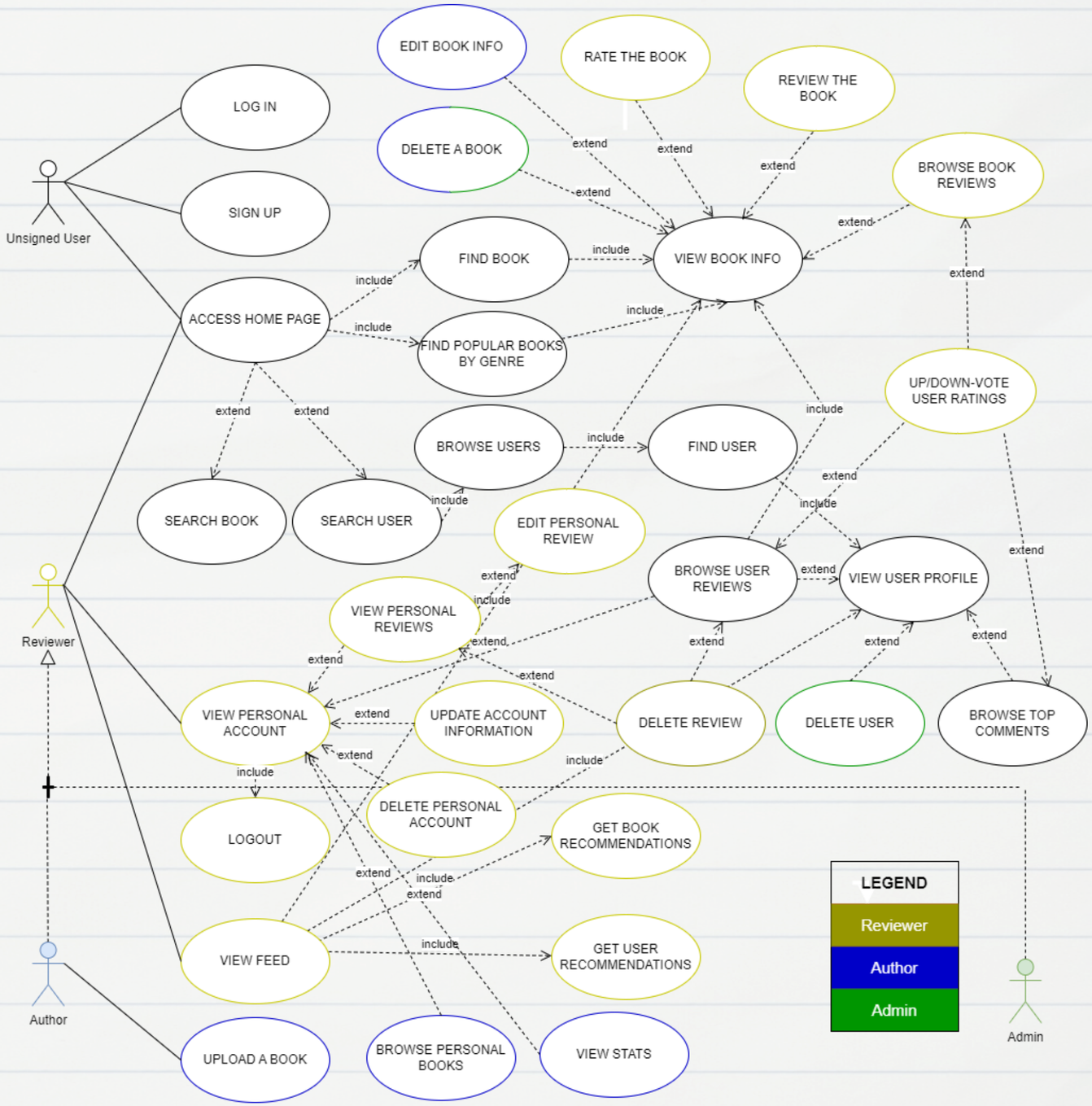
Non-Functional Requirements

- The system must be fast-responsive.
- The system must ensure high availability of content.
- The system must be fault-tolerant, so that the failure of a node does not compromise the service.
- The system must be user-friendly and intuitive to use.
- The system must be available 24/7.

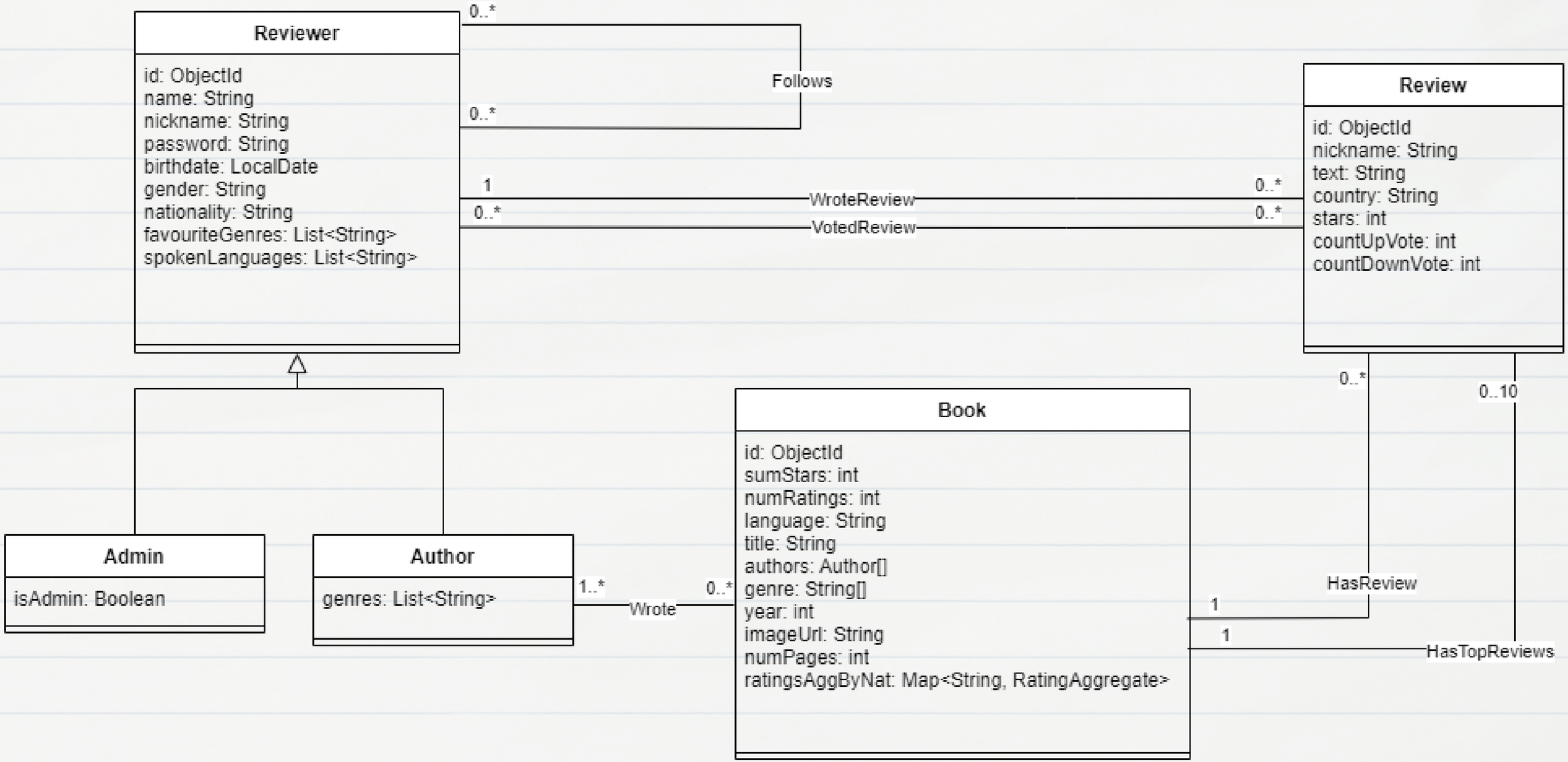


- Our application is positioned in AP
- Only guarantee Eventual Consistency

Use-Case Diagram



UML Class Diagram



User Sample


```
{
  "_id": {
    "$oid": "60de188000000000000001510"
  },
  "name": "Louis Luna",
  "nickname": "kevin15",
  "birth": "1953-12-29",
  "nationality": "Puerto Rico",
  "spoken_languages": [
    "en-US",
    "en-GB",
    "eng",
    "spa",
    "en-CA"
  ],
  "gender": "M",
  "password": "974db8d0ec12c332b39a3c563edf06b3caa6f9b62ac3f29ae439420727bcec0d",
  "up_voted_reviews": [],
  "down_voted_reviews": [],
  "favourite_genres": [
    "comics",
    "graphic",
    "history",
    "historical fiction",
    "biography",
    "mystery",
    "thriller",
    "crime"
  ],
  "genres": [
    "non-fiction",
    "fiction"
  ]
}
```


Review Sample


```
{
    "_id": {
        "$oid": "666c83a8ffaccedfd61229fb"
    },
    "user_id": {
        "$oid": "60de188000000000000a6da"
    },
    "book_id": {
        "$oid": "60de188000000000009f621"
    },
    "rating": 4,
    "date_added": "2013-12-27 12:20:41",
    "review_text": "A compelling drama that explores the depths of the human psyche.",
    "count_up_votes": 912,
    "count_down_votes": 807,
    "nickname": "joanna56",
    "country": "Norfolk Island"
}
```

Book Sample


```
{
    "_id": {
        "$oid": "60de188000000000000053c"
    },
    "language": "eng",
    "title": "Until I Find You",
    "image_url": "https://s.gr-assets.com/assets/nophoto/book/111x148-bcc042a9c91a29c1d680899eff700a03.png",
    "num_pages": 928,
    "year": 2006,
    "genre": [
        "fiction"
    ],
    "ratings_agg_by_nat": {
        "Austria": {
            "cardinality": 1,
            "sum_stars": 4
        }
    },
    "review_ids": [
        {
            "$oid": "666c8513ffaccedfd624582c"
        }
    ],
    "numRatings": 1,
    "sumStars": 4,
    "most_10_useful_reviews": [
        {
            "_id": {
                "$oid": "666c8513ffaccedfd624582c"
            },
            "user_id": {
                "$oid": "60de188000000000000bc499"
            },
            "book_id": {
                "$oid": "60de188000000000000053c"
            },
            "rating": 4,
            "date_added": "2008-04-10 19:18:42",
            "review_text": "What can I say? I just love the way he writes.",
            "count_up_votes": 594,
            "count_down_votes": 268,
            "nickname": "elizabeth90",
            "country": "Austria"
        }
    ],
    "authors": [
        {
            "id": {
                "$oid": "60de188000000000000c03"
            },
            "name": "Carol Smith"
        }
    ]
}
```

MongoDB Aggregation #1: Author Stats

```
db.books.aggregate([
    // Filter books by the specified author's ObjectId
    {
        $match: {
            author: authorId
        }
    },
    // Add the field averageRating in the output as division of the two
    {
        $addFields: {
            averageRating: { $divide: ["$sumStars", "$numRatings"] }
        }
    },
    // Transform the ratings aggregation by nation into a more usable array format
    {
        $addFields: {
            countryRatings: {
                // Put inside the field countryRatings the name of the country, mean score and the number of reviews
                $map: {
                    input: { $objectToArray: "$ratingsAggByNat" },
                    as: "rating",
                    // These 3 are the fields that compose countryRatings
                    in: {
                        country: "$$rating.k",
                        data: {
                            averageRating: { $divide: ["$$rating.v.sumRating", "$$rating.v.cardinality"] },
                            numRatings: "$$rating.v.cardinality"
                        }
                    }
                }
            }
        }
    },
    // Use $reduce to reformat country ratings into a more detailed array
    {
        $addFields: {
            detailedCountryRatings: {
                $reduce: {
                    input: "$countryRatings",
                    initialValue: [],
                    in: {
                        $concatArrays: ["$$value", [
                            {
                                country: "$$this.country",
                                averageRating: "$$this.data.averageRating",
                                numRatings: "$$this.data.numRatings"
                            }
                        ]]
                    }
                }
            }
        }
    },
    // Project final document structure without additionalInfo
    {
        $project: {
            bookTitle: "$title",
            bookRating: "$averageRating",
            bookTotalRatings: "$numRatings",
            bookCountryDetails: {
                $map: {
                    input: "$detailedCountryRatings",
                    as: "countryDetail",
                    in: {
                        country: "$$countryDetail.country",
                        averageRating: {
                            $round: ["$$countryDetail.averageRating", 2]
                        },
                        numRatings: "$$countryDetail.numRatings"
                    }
                }
            }
        }
    },
    // Sorting by title
    {
        $sort: { "bookTitle": 1 }
    }
]);
```

MongoDB Aggregation #2: Find most famous book given a genre for each country

```
db.books.aggregate([
  // Match the books with the given genre
  {
    $match: {
      genre: "Adventure" // Replace "Adventure" with the genre input
    }
  },
  // Project necessary fields and initialize aggregation structures
  {
    $project: {
      title: 1,
      ratingsAggByNat: 1,
      countryReviews: {
        $map: {
          input: { $objectToArray: "$ratingsAggByNat" },
          as: "country",
          in: {
            country: "$$country.k",
            numReviews: "$$country.v.cardinality",
            bookTitle: "$title",
            bookId: "$_id"
          }
        }
      }
    }
  },
  // Deconstruct countryReviews array into individual documents
  {
    $unwind: "$countryReviews"
  },
  // Group by country and find the book with the most reviews for each countr
  {
    $group: {
      _id: "$countryReviews.country",
      mostFamousBook: {
        $first: {
          title: "$countryReviews.bookTitle",
          bookId: "$countryReviews.bookId",
          numReviews: "$countryReviews.numReviews"
        }
      },
      maxNumReviews: { $max: "$countryReviews.numReviews" }
    }
  },
  // Project the final result
  {
    $project: {
      _id: 0,
      country: "$_id",
      title: "$mostFamousBook.title",
      bookId: "$mostFamousBook.bookId",
      numReviews: "$mostFamousBook.numReviews"
    }
  }
]);
```

MongoDB Aggregation #3: Find the top useful reviews of a user in order of usefulness

```
db.books.aggregate([
  // Match books that contain reviews by the given username
  {
    $match: {
      "most10UsefulReviews.userName": "AliceJones89"
    }
  },

  // Add a new field that calculates the usefulness of each review
  {
    $addFields: {
      "most10UsefulReviews": {
        $map: {
          input: "$most10UsefulReviews",
          as: "review",
          in: {
            $mergeObjects: [
              "$$review",
              {
                usefulness: {
                  $subtract: ["$$review.countUpVote", "$$review.countDownVote"]
                }
              }
            ]
          }
        }
      }
    }
  },

  // Filter reviews to only include those by the given username
  {
    $addFields: {
      "most10UsefulReviews": {
        $filter: {
          input: "$most10UsefulReviews",
          as: "review",
          cond: { $eq: ["$$review.userName", "AliceJones89"] }
        }
      }
    }
  },

  // Sort the reviews by usefulness in descending order
  {
    $addFields: {
      "most10UsefulReviews": {
        $slice: [
          {
            $sortArray: {
              input: "$most10UsefulReviews",
              sortBy: { usefulness: -1 }
            }
          },
          3
        ]
      }
    }
  },
```

MongoDB Indexes

- Index in the books collection on the author id to speed up author stats.
- Tested on the author with the most books

Scenario	Stats
No index	executionTimeMillis: 24, totalKeysExamined: 0, totalDocsExamined: 8680
`authors.id:1`	executionTimeMillis: 6, totalKeysExamined: 6, totalDocsExamined: 6

- Index in the books collection on the genre field to speed up second aggregation.
- Tested on “fiction”.

Scenario	Stats
No index	executionTimeMillis: 104, totalKeysExamined: 0, totalDocsExamined: 8680
`genre:1`	executionTimeMillis: 85 totalKeysExamined: 6995, totalDocsExamined: 6995

- Index in the reviews collection on user_id.
- Tested on user “edward39”.

Scenario	Stats
No index	executionTimeMillis: 165, totalKeysExamined: 0, totalDocsExamined: 99026
`user_id:1`	executionTimeMillis: 2, totalKeysExamined: 6, totalDocsExamined: 6

- Index in the books collection to the title field; it proved useless.
- Tested searching “a”.

Sharding

Users

- Sharding the users collection based on hashing is generally beneficial, with minimal slowdown in user searches, which are infrequent and usually target specific usernames, thus accessing only one shard.
- Alternatively, list-based sharding on nationality could be effective based on geographic user activity.

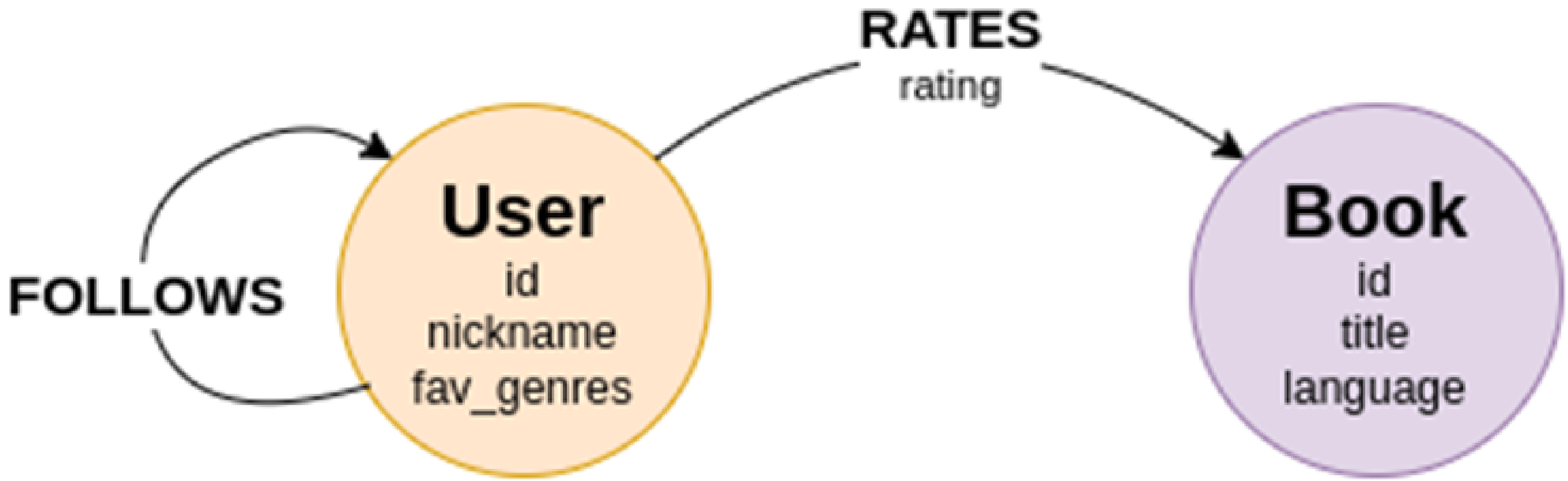
Reviews

- Sharding the reviews collection is only advisable if reviews are sharded according to the users.

Books

- We concluded that sharding the books would not be beneficial.

Neo4J Design



Neo4J Queries

- Find recommended books based on the books read and reviewed by the users we are following.

```
MATCH (user:User {id: '60de18800000000000002392b'})-[:FOLLOWS]→(other:User)-[r:RATES]→(book:Book)
WHERE book.language IN [ 'en-US', 'en-GB', 'cat', 'eng', 'ita', 'spa' ]
RETURN other, book, r.rating AS rating
ORDER BY rating DESC
LIMIT 10
```

- Find recommended users to follow based on who read and reviewed the same books as us.

```
MATCH (u1:User {id: '60de18800000000000002392b'})-[r1:RATES]→(book:Book)←[r2:RATES]-(u2:User)
WHERE id(u1) <> id(u2)
AND abs(r1.rating - r2.rating) < 2
WITH u1, u2, COLLECT(DISTINCT book.title) AS commonBooks, COUNT(book) AS commonBookCount
WHERE SIZE([genre IN u1.favouriteGenres WHERE genre IN u2.favouriteGenres]) > 0
RETURN u2.id AS user2, u2.nickname AS Nickname, commonBooks, commonBookCount
ORDER BY commonBookCount DESC
LIMIT 10
```

Neo4J Queries

- Feed: find who has been recently followed by users we are following.

```
MATCH (user:User {id: '60de18800000000000002392b'})-[:FOLLOWS]→(followed:User)-[f:FOLLOWS]→(other:User)
RETURN followed.id AS followedUserId, followed.nickname AS followedUserNickname,
other.id AS followedUserFollowId, other.nickname AS followedUserFollowNickname, id(f) AS followId
ORDER BY id(f) DESC
LIMIT 6
```

- Feed: find books who have been recently reviewed by users we are following.

```
MATCH (user:User {id: '60de18800000000000002392b'})-[:FOLLOWS]→(followed:User)-[r:RATES]→(book:Book)
RETURN followed.id AS followedUserId, followed.nickname AS followedUserNickname,
book.id AS bookId, book.title AS bookTitle, r.rating AS rating, id(r) AS ratingId
ORDER BY id(r) DESC
LIMIT 6
```

Neo4J Indexes

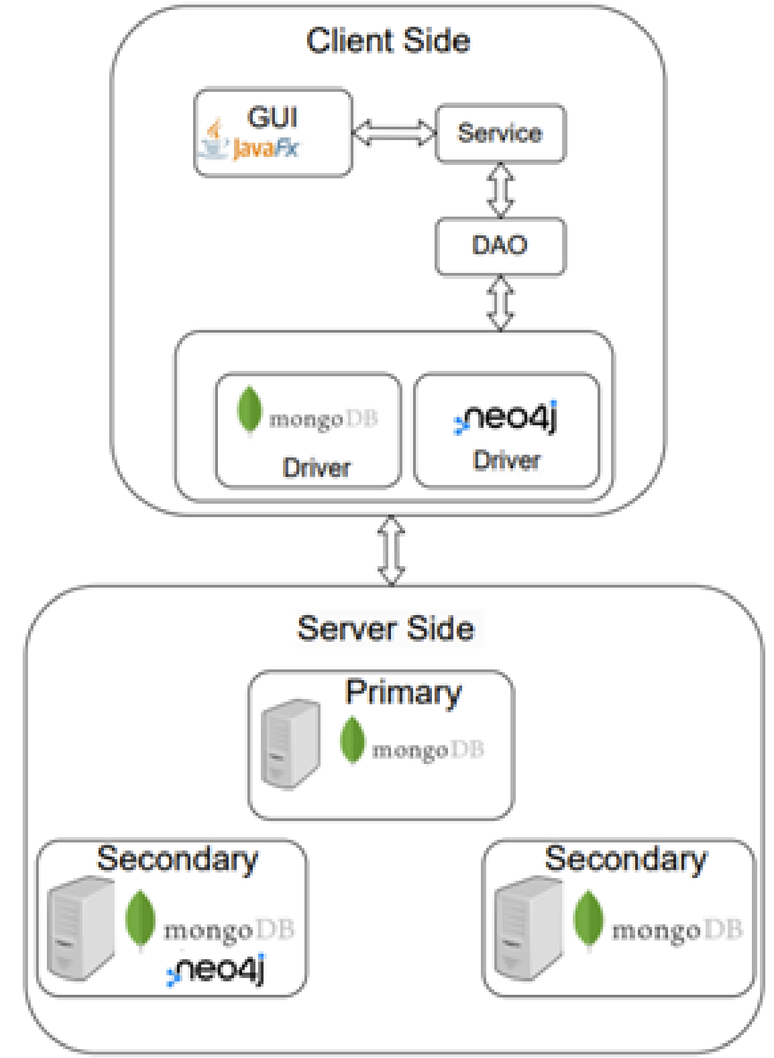
- Index on User id

```
neo4j$ CREATE INDEX FOR (n:User) ON (n.id)
```

- Index on Book id
  - We also tried to add this index since all the queries were using the Book id. Unfortunately, it didn't have the effect we hoped

	Time without index	Time with index
Book Recommendations	167ms	11ms
Users with Similar Tastes	24ms	3ms
Recent Follows by Followed Users	305ms	4ms
Ratings of Followed Users	32'491ms	321ms

Architecture



Replicas

- We set MongoDB with 3 replicas.

Write Concern

- In order to guarantee a low-latency response and high availability, we chosed that any write operation may be considered completed when one write is complete. Our global write concern is set as W1.

Read Concern

- We decided to set the read preference on nearest replica, because we wanted to guarantee low-latency, as we considered Neo4Food more oriented on reads than on writes.

Thank's For Watching