Information Management & Systems Engineering

Milestone 2: Migration from a relational DBMS to NoSQL

Fabian Frauenberger

11807201

a11807201@unet.univie.ac.at

Samuel Mitterrutzner

11911464

a11911464@unet.univie.ac.at

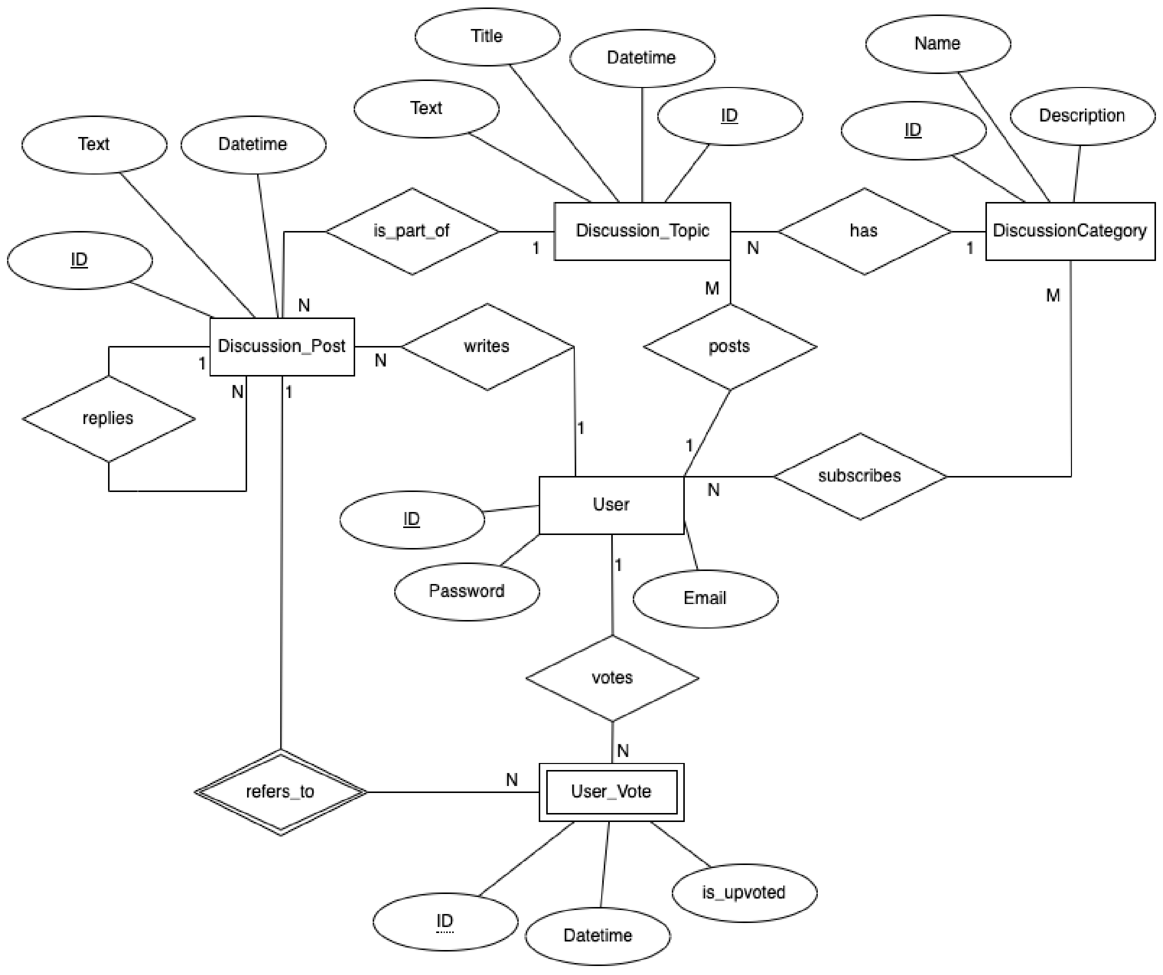
June 23rd 2022

Configuration of Infrastructure

To access the web application, go to the folder submitted and run the docker-compose up command in a CLI. This will launch all the required containers. Then open http://localhost:8080 or https://localhost in your browser. Because we used a self-signed certificate, you may get a certificate warning when using https.

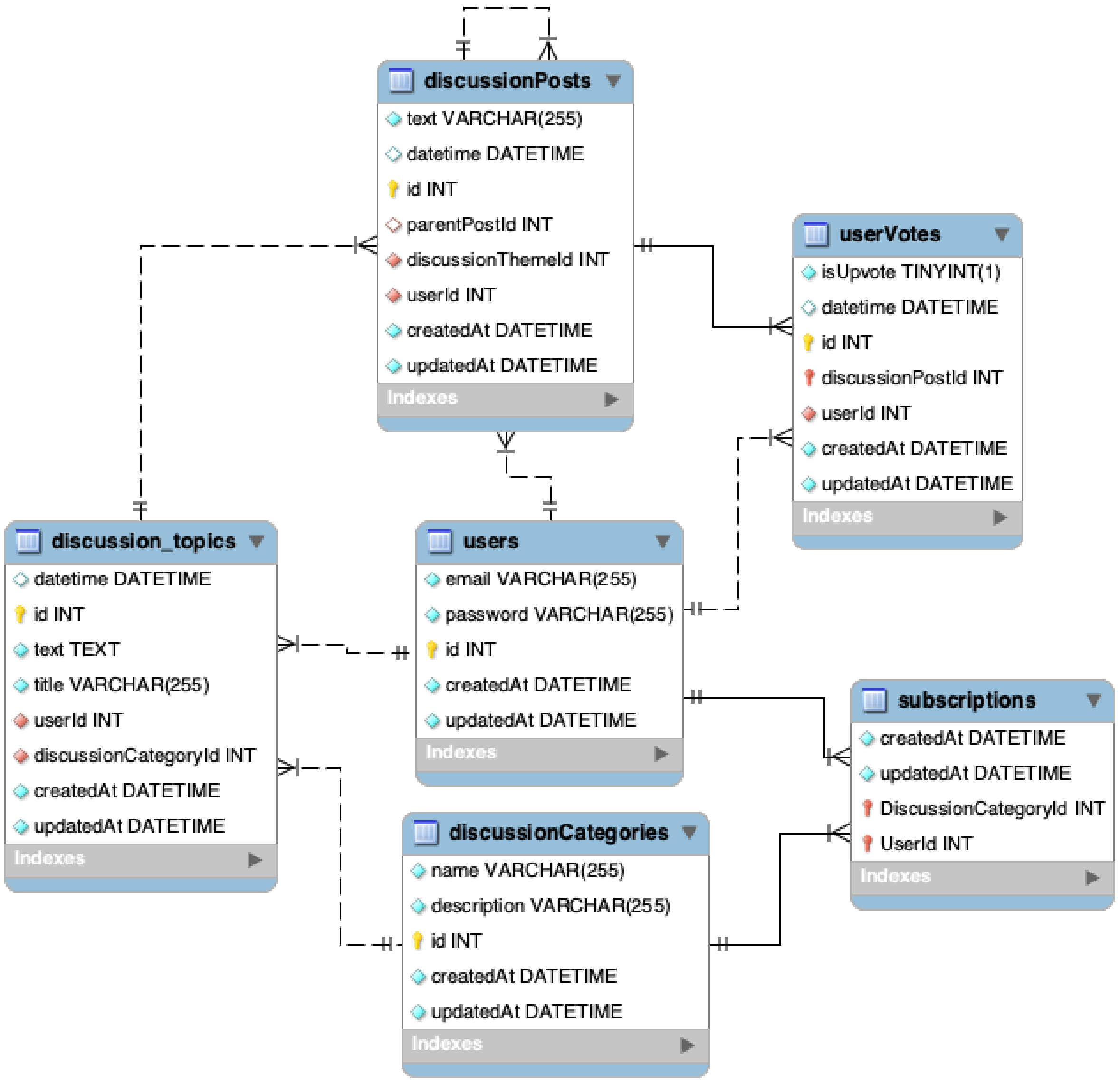
RDBMS Design

Entity Relationship Diagram



The idea of our project is to implement an online discussion platform, where users can write anonymously. In comparison to Milestone 1 the username has been removed and only the email (that is not shown publicly) is stored in the user table. Users can create new topics or write posts about topics. A Voting system for posts is another way to agree with other people's opinions.

Logical Database Design Graph



Logical Database Design

users (id,email,password)
PK: id

discussionCategories (id, name, description)
PK: name

subscriptions (*DiscussionCategoryId*, *UserId*)
PK: DiscussionCategoryId, UserId
FK: subscriptions.DiscussionCategoryId <> discussionCategories.id
subscriptions.UserId <> users.id

discussion_topics (id, title, text, datetime, *discussionCategoryId, userId*)
PK: id
FK: discussion_topics.discussionCategoryId <> discussionCategories.id
discussion_topics.userId <> users.id

discussionPosts (id, text, datetime, *discussionThemeId, userId, parentPostId*)
PK: id
FK: discussionPosts.discussionThemeId <> discussion_topics.id
discussionPosts.userId <> users.id
discussionPosts.parentPostId <> discussionPosts.id

userVotes (id, *discussionPostId*, isUpvote, datetime, *userId*)
PK: id, discussionPostId
FK: userVotes.discussionPostId <> discussionPosts.id
userVotes.userId <> users.id

Physical Database Design

```
CREATE TABLE  users (
     email      VARCHAR(255) NOT NULL UNIQUE,
     password   VARCHAR(255) NOT NULL,
     id         INTEGER auto_increment,
     createdat DATETIME NOT NULL,
     updatedat DATETIME NOT NULL,
     PRIMARY KEY (id)
  )


CREATE TABLE  discussioncategories (
     name         VARCHAR(255) NOT NULL UNIQUE,
     description VARCHAR(255) NOT NULL,
     id           INTEGER auto_increment,
     createdat    DATETIME NOT NULL,
     updatedat    DATETIME NOT NULL,
     PRIMARY KEY (id)
  )


CREATE TABLE  subscriptions (
     createdat            DATETIME NOT NULL,
     updatedat            DATETIME NOT NULL,
     discussioncategoryid INTEGER,
     userid               INTEGER,
     PRIMARY KEY (discussioncategoryid, userid),
     FOREIGN KEY (discussioncategoryid) REFERENCES discussioncategories (id) ON
     DELETE CASCADE ON UPDATE CASCADE,
     FOREIGN KEY (userid) REFERENCES users (id) ON DELETE CASCADE ON UPDATE
     CASCADE
  )
```

```
CREATE TABLE  discussion_topics (
     datetime              DATETIME,
     id                    INTEGER auto_increment,
     text                  TEXT NOT NULL,
     title                 VARCHAR(255) NOT NULL,
     userid                INTEGER NOT NULL,
     discussioncategoryid INTEGER NOT NULL,
     createdat             DATETIME NOT NULL,
     updatedat             DATETIME NOT NULL,
     PRIMARY KEY (id),
     FOREIGN KEY (userid) REFERENCES users (id) ON DELETE CASCADE ON UPDATE
     CASCADE,
     FOREIGN KEY (discussioncategoryid) REFERENCES discussioncategories (id) ON
     DELETE CASCADE ON UPDATE CASCADE
  )

CREATE TABLE  discussionposts (
     text               VARCHAR(255) NOT NULL,
     datetime           DATETIME,
     id                 INTEGER auto_increment,
     parentpostid       INTEGER,
     discussionthemeid INTEGER NOT NULL,
     userid             INTEGER NOT NULL,
     createdat          DATETIME NOT NULL,
     updatedat          DATETIME NOT NULL,
     PRIMARY KEY (id),
     FOREIGN KEY (parentpostid) REFERENCES discussionposts (id) ON DELETE SET
     NULL ON UPDATE CASCADE,
     FOREIGN KEY (discussionthemeid) REFERENCES discussion_topics (id) ON DELETE
     no action ON UPDATE CASCADE,
     FOREIGN KEY (userid) REFERENCES users (id) ON DELETE no action ON UPDATE
     CASCADE
  )

CREATE TABLE  uservotes (
     isupvote         TINYINT(1) NOT NULL,
     datetime         DATETIME,
     id               INTEGER auto_increment,
     discussionpostid INTEGER NOT NULL,
     userid           INTEGER NOT NULL,
     createdat        DATETIME NOT NULL,
     updatedat        DATETIME NOT NULL,
     PRIMARY KEY (id, discussionpostid),
     FOREIGN KEY (discussionpostid) REFERENCES discussionposts (id) ON DELETE no
     action ON UPDATE CASCADE,
     FOREIGN KEY (userid) REFERENCES users (id) ON DELETE no action ON UPDATE
     CASCADE
  )
```

NoSQL Design

Collections

There are the following collections in our MongoDB database: discussioncategories, discussionposts, discussiontopics, and users. Compared to the MySQL database, the table subscriptions and userVotes are not represented as separate collections. subscriptions are saved in the user Collection. userVotes are embedded in the discussionposts collection.

Discussioncategories

The Categories are stored in its own Collection. A Category contains the same data as in the SQL Database. Categories are stored in its own collection because they are needed on their own frequently. Every time a new topic is created all categories are queried. If they were embedded in the discussion topics collection all topics (we assume that there will be much more topics than categories) needed to be queried to find all categories. Our second reason is to make it easier to update or add categories.

Category names should be unique. To ensure this constraint we use an unique index on the name field of a category. As we are sorting by the category name in one of our reports this index will increase the performance of this operation.

```
{

    _id: ObjectId('62b2c6f1abc30fb4b02db9f1'),

    description: 'Local News description',

    name: 'Local News',

}
```

Users

The Users Collection contains all data, just like the table in the SQL database. Every subscribed category from the User is saved in an array. We decided to save the subscriptions directly in the User collection to prevent infinitely growing arrays. The array can reach a maximum size of the number of categories. If we would have stored it directly in the categories then the array size would be bound by the number of users that would be much greater than the number of categories. The subscribed categories are not directly saved in the document, only their id is saved.

User login data should be mapped to only one user. Therefore an unique index is used to ensure that each user has its unique email address. This index will be useful to speed up the login process as not all documents need to be queried.

```
{
    _id: ObjectId('62b2c6f1abc30fb4b02db9e7'),
    email: 'Levi_Gerhold@yahoo.com',
    password: 'RtrKFGapBGzI7dN',
    subscriptions: [
        ObjectId('62b2c6f1abc30fb4b02db9f1'),
        ObjectId('62b2c6f1abc30fb4b02db9f3'),
        ObjectId('62b2c6f1abc30fb4b02db9f5'),
        ObjectId('62b2c6f1abc30fb4b02db9f8')
    ],
}
```

Discussiontopics

The Discussiontopics Collection contains all of the SQL database's data. The number of posts is also saved in the topic. Every time a post is added this field will be increased by one for the corresponding topic. We decided against storing references to posts because we anticipate that there will be a large number of posts per topic, which will lead to an infinite growth of the array and, as a result, we may be dangerously close to exceeding the size limit of a document. The count of posts is important for displaying it on the starting page and our first report.

We used several indexes in the Discussiontopics Collection. First we use a decreasing index for the creation time of the article (datetime). This index will be important for our first report to efficiently find all topics that haven been written in after a specified time. An index on the postsCount attribute will help sorting the result. An index on the userId is needed to find all topics written by a user, without needing to check all documents in the collection. This will be important in our second report.

```
{
    _id: ObjectId('62b2c6f1abc30fb4b02dba20'),
    datetime: ISODate('2021-12-23T07:45:59.000Z'),
    discussionCategory: ObjectId('62b2c6f1abc30fb4b02db9fb'),
    text: 'Ad quos sed nemo.\nDucimus odit fugiat.\nProvident quae ut omnis quod.',
    title: 'topic 24',
    userId: '62b2c6f1abc30fb4b02db9e7',
    postsCount: 6,
}
```

Discussionposts

All data from the discussionPosts table in the SQL database is also included in the Discussionposts Collection. In addition, the discussionThemeId is used to store a reference to the topic. As a result, each post can be assigned to a specific topic. To keep the postsCount in the topic up to date, adding a post for the related topic will increase the postsCount. The userVotes are attached as an array of subdocuments. These contain the same fields as the corresponding table in the SQL database. UserVotes are directly integrated because they are always used in conjunction with the post and thus a separate collection would make little sense in our opinion. Naturally, there may be issues with the maximum document size, but because userVotes don’t store lots of information, we don't see this as a major issue.

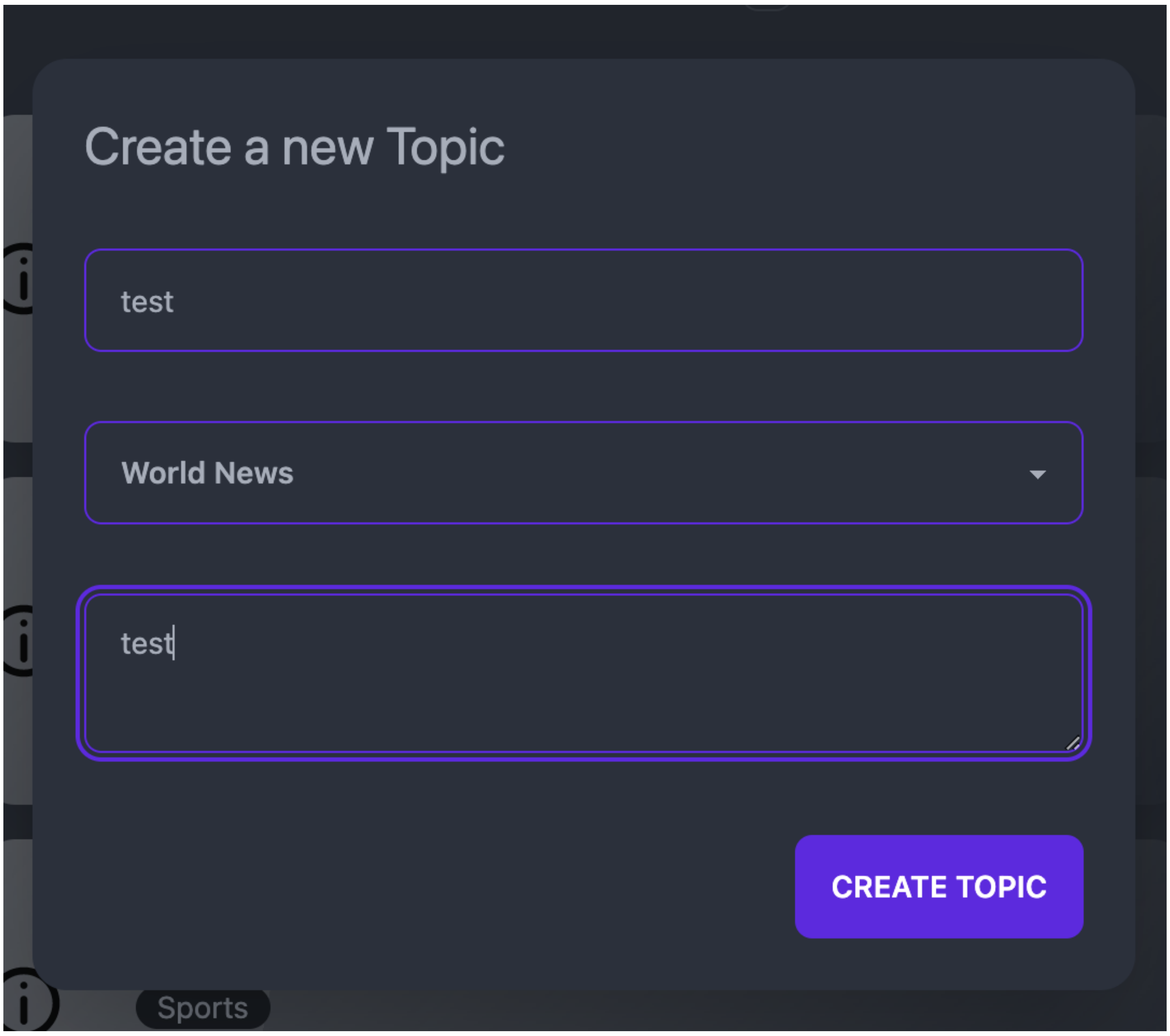
Because posts are frequently searched for by their discussionThemeId, we use an index to improve query performance. As we query for replies of comments separately the parentPostId also uses an index to improve the query performance for finding all replies of a post.

```
{

    _id: ObjectId('62b2c6f2abc30fb4b02dba48'),

    text: 'Nostrum facere ipsa molestiae molestiae eius veniam quasi.',

    datetime: ISODate('2022-05-11T11:57:21.000Z'),

    discussionThemeId: '62b2c6f1abc30fb4b02dba20',

    parentPostId: null,

    userId: '62b2c6f1abc30fb4b02db9e7',

    userVotes: [

        {

            datetime: ISODate('2022-05-20T00:27:21.000Z'),

            isUpvote: true,

            userId: '62b2c6f1abc30fb4b02db9eb',

            _id: ObjectId('62b2c6f2abc30fb4b02dba49')

        },

        {

            datetime: ISODate('2022-05-12T10:45:40.000Z'),

            isUpvote: true,

            userId: '62b2c6f1abc30fb4b02db9e7',
          _id: ObjectId('62b2c6f2abc30fb4b02dba4a')
        }
     ],
  }
```

Use-cases and Reports

Use Case: Creating Discussion Topic

The first main use case is creating topics. This can be done via the topic panel by clicking on “Create topic”. The topic will then be added to the database and displayed on the topic page.



Create a new Topic

test

World News

test

SQL Query:

```
INSERT INTO discussion_topics
            (datetime,
             id,
             text,
             title,
             userid,
             discussioncategoryid,
             createdat,
             updatedat)
VALUES         ('2021-06-22 20:25:00',
             DEFAULT,
             'Test text',
             'test',
             2,
             1,
             '2021-06-22 20:25:00',
              '2021-06-22 20:25:00')
```

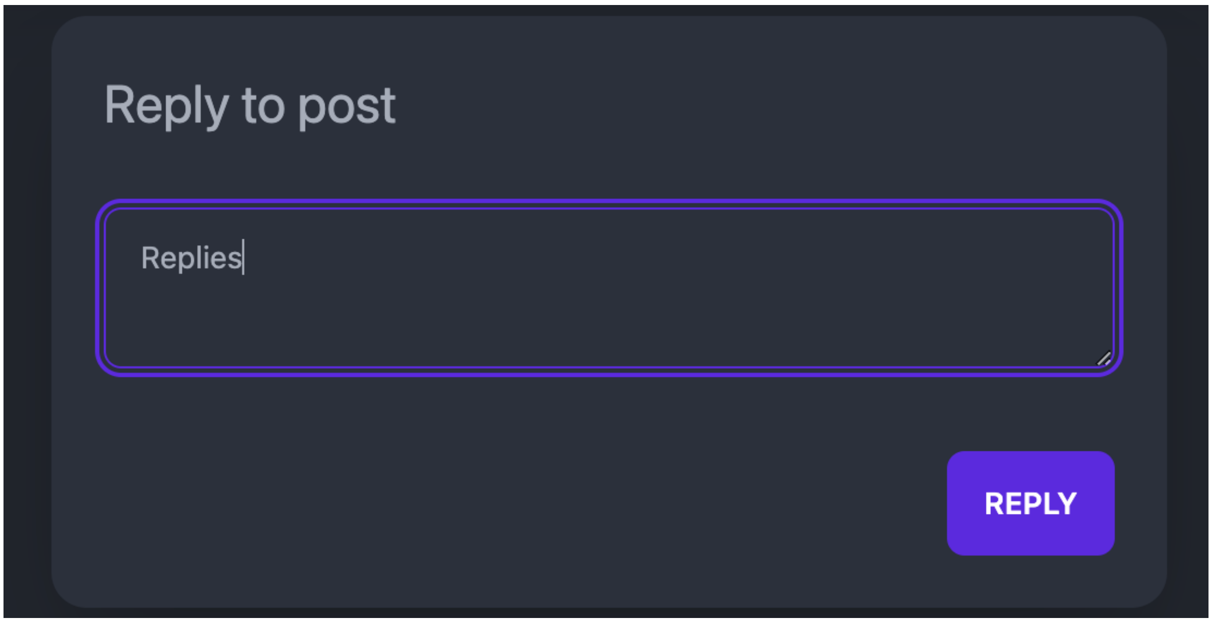
MongoDB Query:

```
db.discussiontopics.insertOne({
    text: 'some text',
    title: 'title',
    datetime: '2021-06-22 20:25:00',
    userId: '62b2c6f1abc30fb4b02db9e7',
    discussionCategory: '62b2c6f1abc30fb4b02db9fb',
    postsCount: 0
})
```

Both Queries insert the data with a single statement. The same operation is done in the SQL and MongoDB statement. Therefore there will be no performance improvements

Use Case: Writing a Post

The second main use case is writing a post. This can be done via the post panel by clicking on “Create post” or “reply”. The post will then be added to the database and displayed on the post page.



SQL Query:

```
INSERT INTO discussionposts
            (text,
             datetime,
             id,
             discussionthemeid,
             userid,
             createdat,
             updatedat)
VALUES      ('testing',
             '2021-06-22 20:25:00',
             DEFAULT,
             6,
             3,
             '2021-06-22 20:25:00',
             '2021-06-22 20:25:00')
```

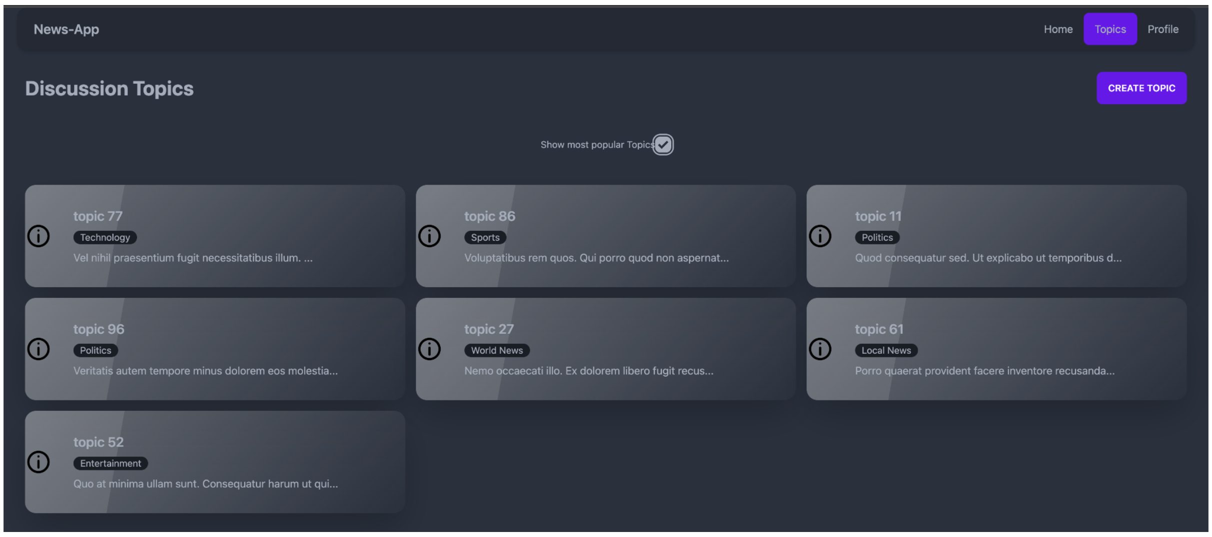
MongoDB Query:

```
db.discussionPosts.insertOne({
    text: 'Nostrum facere ipsa molestiae molestiae eius veniam quasi.',
    datetime: ISODate('2022-05-11T11:57:21.000Z'),
    discussionThemeId: '62b2c6f1abc30fb4b02dba20',
    parentPostId: null,
    userId: '62b2c6f1abc30fb4b02db9e7',
    userVotes: []
})

db.discussiontopics.updateOne({
    _id: '62b2c6f1abc30fb4b02dba20'
}, {
    $inc: {
        postsCount: 1
    }
})
```

Adding Posts in the SQL Database only needs to insert the post in the discussionPosts table, while the mongodb query needs to update the postsCount of the topic and add the post to the collection. Therefore the mongodb insert is more expensive than the SQL insert

Report: Which are interesting topics of discussions?

This report shows interesting topics that have been written in the last year. Topics are considered as interesting if they get many posts. To show the most interesting first, they are ordered by their number of posts. This report can be called by ticking the checkbox on the topics page.



Discussion Topics

topic 77
Technology
Vel nihil praesentium fugit necessitatibus illum. ...

topic 86
Sports
Voluptatibus rem quos. Qui porro quod non aspernat...

topic 11
Politics
Quod consequatur sed. Ut explicabo ut temporibus d...

topic 96
Politics
Veritatis autem tempore minus dolorem eos molestia...

topic 27
World News
Nemo occaecati illo. Ex dolorem libero fugit recus...

topic 61
Local News
Porro quaerat provident facere inventore recusanda...

topic 52
Entertainment
Quo at minima ullam sunt. Consequatur harum ut qui...

SQL Query:

```
SELECT DiscussionTopic.datetime,
       DiscussionTopic.id,
       DiscussionTopic.text,
       DiscussionTopic.title,
       DiscussionTopic.userid,
       DiscussionTopic.discussioncategoryid,
       DiscussionTopic.createdat,
       DiscussionTopic.updatedat,
       Count(discussionposts.id)       AS PostCount,
       DiscussionCategory.NAME         AS discussioncategory.NAME,
       DiscussionCategory.description  AS discussioncategory.description,
       DiscussionCategory.id           AS discussioncategory.id,
       DiscussionCategory.createdat    AS discussioncategory.createdAt,
       DiscussionCategory.updatedat    AS discussioncategory.updatedAt
FROM   discussion_topics AS DiscussionTopic
       LEFT OUTER JOIN discussioncategories AS DiscussionCategory
                    ON DiscussionTopic.discussioncategoryid =
                       DiscussionCategory.id
       LEFT OUTER JOIN discussionposts AS DiscussionPosts
                    ON DiscussionTopic.id = discussionposts.discussionthemeid
WHERE  DiscussionTopic.datetime >= '2021-06-22 20:25:00'
GROUP  BY DiscussionTopic.id
ORDER  BY postcount DESC
```

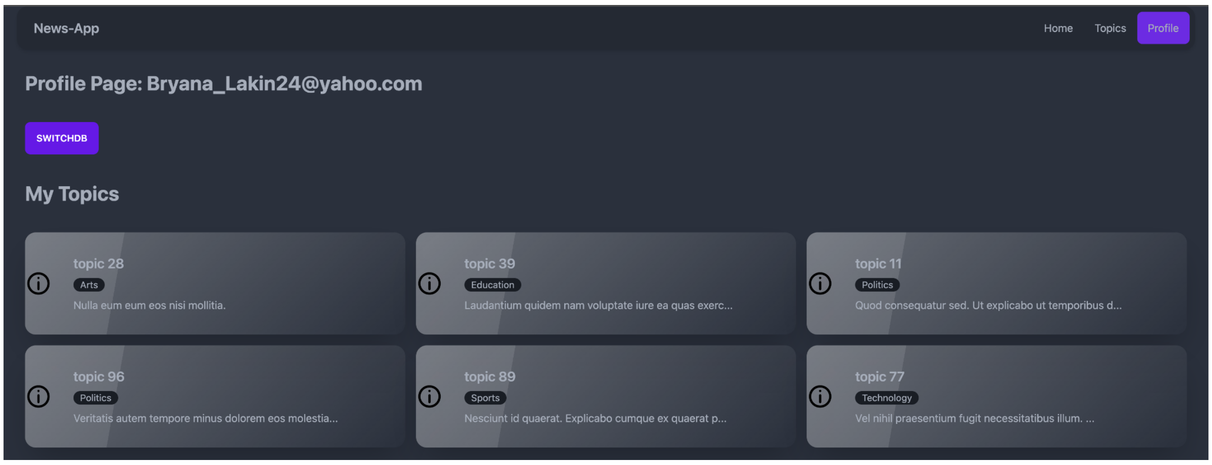
MongoDB Query:

```
db.discussiontopics.aggregate([{
    $match: {
        datetime: {
            $gte: ISODate('2021-12-23T07:45:59.000Z')
        }
    }
}, {
    $sort: {
        postsCount: -1
    }
}, {
    $lookup: {
        from: 'discussioncategories',
        localField: 'discussionCategory',
        foreignField: '_id',
        as: 'discussionCategory'
    }
}, {
    $unwind: {
        path: '$discussionCategory'
    }
}])
```

The SQL Query needs 2 joins as all data is in different tables. The MongoDB query manages this report with only one lookup operation.The aggregation does not need to lookup (and unwind) the discussionPosts as the number of Posts is stored directly in the topics. Because of our chosen indexes (datetime, postscount) not all documents need to be checked this will lead to a reduction of the query time. The MongoDB query is more performant than the sql query.

Report: What are all the topics a user has created sorted by category

Our second report shows topics that have been written by a user and sorts them by the name of the category. This report can be called by accessing the profile page.



Profile Page: Bryana_Lakin24@yahoo.com

My Topics

topic 28
Arts
Nulla eum eum eos nisi mollitia.

topic 39
Education
Laudantium quidem nam voluptate iure ea quas exerc...

topic 11
Politics
Quod consequatur sed. Ut explicabo ut temporibus d...

topic 96
Politics
Veritatis autem tempore minus dolorem eos molestia...

topic 89
Sports
Nesciunt id quaerat. Explicabo cumque ex quaerat p...

topic 77
Technology
Vel nihil praesentium fugit necessitatibus illum. ...

SQL Query:

```
SELECT DiscussionTopic.datetime,
       DiscussionTopic.id,
       DiscussionTopic.text,
       DiscussionTopic.title,
       USER.id,
       DiscussionTopic.discussioncategoryid,
       DiscussionTopic.createdat,
       DiscussionTopic.updatedat,
       DiscussionCategory.NAME        AS discussioncategory.NAME,
       DiscussionCategory.description AS discussioncategory.description,
       DiscussionCategory.id          AS discussioncategory.id,
       DiscussionCategory.createdat   AS discussioncategory.createdAt,
       DiscussionCategory.updatedat   AS discussioncategory.updatedAt
FROM   discussion_topics AS DiscussionTopic
       LEFT OUTER JOIN discussioncategories AS DiscussionCategory
                    ON DiscussionTopic.discussioncategoryid =
                       DiscussionCategory.id
       LEFT OUTER JOIN users AS USER
                    ON DiscussionTopic.userid = USER.id
WHERE  DiscussionTopic.userid = '1'
ORDER  BY DiscussionCategory.NAME ASC
```

MongoDB Query:

```
db.discussiontopics.aggregate([{
    $match: {
        userId: '62b38509d6ff45edc0a3614d'
    }
}, {
    $lookup: {
        from: 'discussioncategories',
        localField: 'discussionCategory',
        foreignField: '_id',
        as: 'discussionCategory'
    }
}, {
    $unwind: {
        path: '$discussionCategory'
    }
}, {
    $sort: {
        'discussionCategory.name': 1
    }
}])
```

For this report the needed joins could not be reduced, but the MongoDB query should be faster by utilizing our indices. As mentioned in the first report by using the index on the userId not all topics need to be checked. This will improve the query performance if many topics are stored in the collection.

Distribution of work

Fabian Frauenberger:

- Main use case: Writing a Post
- Report: Which are interesting topics of discussions?
- Data Import
- Data Migration
- SSL Certificates
- Docker

Samuel Mitterrutzner:

- Main use case: Creating Discussion Topic
- Report: What are all the topics a user has created sorted by category
- Frontend
- Docker
- Overall setup (Nx, Next.js and Express)