# Large Scale and Multi-Structured Databases Project Report

Lorenzo Catoni    Leonardo Giovannoni    Marco Lucio Mangiacapre

## Contents

# 1 Introduction

Today is easy to browse the internet and find a lot of information about substantially anything: literature, science, engineering, entertainment and much more. There is so many information a simple search for something you believed was simple brought you much more text you could read in a year! That is amazing but it can reveal extremely frustrating if you only need for a simple specific yes/no answer and a search engine only show you hundred-pages long results. Sometimes nothing is more valuable than the help and suggestions of who already masters a topic and can give you a direct and self-contained answer. So here is our idea, why not build an easy-to-use website where users can freely ask questions about anything the like or they are interested in, a place where also newbie can freely ask questions and meta-questions – that is, ask what they need to know before search or ask for meaningful information on a specific topic – without being afraid anyone will be rude with them? And, why not, search all the knowledge collected by others?

# 2 Description and Requirements

## 2.1 Functional Requirements

- Guest user
  - Browse questions [1]
  - View question
  - Signup and login
  - Browse tags [1]
  - View statistics
- Regular user
  - Post questions, answers, and comments
  - Follow tags
  - Create tags
  - Subscribe to receive update about questions
  - Vote and comment questions
  - View own statistics
  - Logout
  - Post comment
  - Post answer
- Admin
  - Moderate (remove) questions, comments, and answers
  - Browse regular users [1]
  - Browse own banned users [1]
  - Ban regular users
  - Can force automatic database resynchronization

## 2.2 Non-Functional Requirements

- High Availability: the ability of supplying answers to users' questions is what make our application so valuable, and due to the nature of the product an old result is always better that no results.
- Eventual consistency: if a user postes a question probably it has not found any suitable answer before, so it is reasonable it is about an uncommon topic and few people know about it. Then a few minutes wait for replica synchronization is negligible respect to the time passed before someone come with the required response.
- Maintainability: the code must be well organized and conceptually different system must be strictly separated. How functionalities (e.g.: creating a new question) work must be clearly separated from how they are implemented. Introducing new features must not require refactoring the whole application.
- Usability: the user interface must be extremely simple to use

[1] visualizing their preview disposed in multiple page

### 2.3 Cap triangle

- Partition tolerance: required
- Availability: required
- Consistency: not necessary

## 3 UML Diagrams

### 3.1 Actors and Use Case Diagram

The application provides that there may be 3 types of users:

- *Abstract User*: this abstract actor has the purpose of bringing together all the functionalities that are common to the actors present below.
- **Guest User**: this actor can browse the tag section, the tags themselves, search for questions and scroll through them as well as see statistics about the interactions between questions and users and statistics on the tags. It is provided for this user the ability to register and log into the site.
- *Logged User*: This abstract actor can do everything the unregistered user can do, except signup and login, also can ask questions, give answers, write comments, create and follow tags. He has his own personal profile, where his own statistics, interactions with the site, and the questions / tags he is interested in are summarized.
- **Regular User**: this actor can do everything the logged user can do, but also be banned. The use case is the same of the abstract user, but this is conceptually different from the latter, in fact a simple extension of the application who justify this choice could be the use case of *ask for unban*: abstract user can't ask for unban because can't be banned.
- **Admin**: this actor can do everything the logged user can do, but also can ban users, for an hour[2], and can remove questions, answers and comments deemed inappropriate.

[2]this time has been chosen for semplicity, a simple extension would be give to admin the power to ban for a more arbitrary period of time

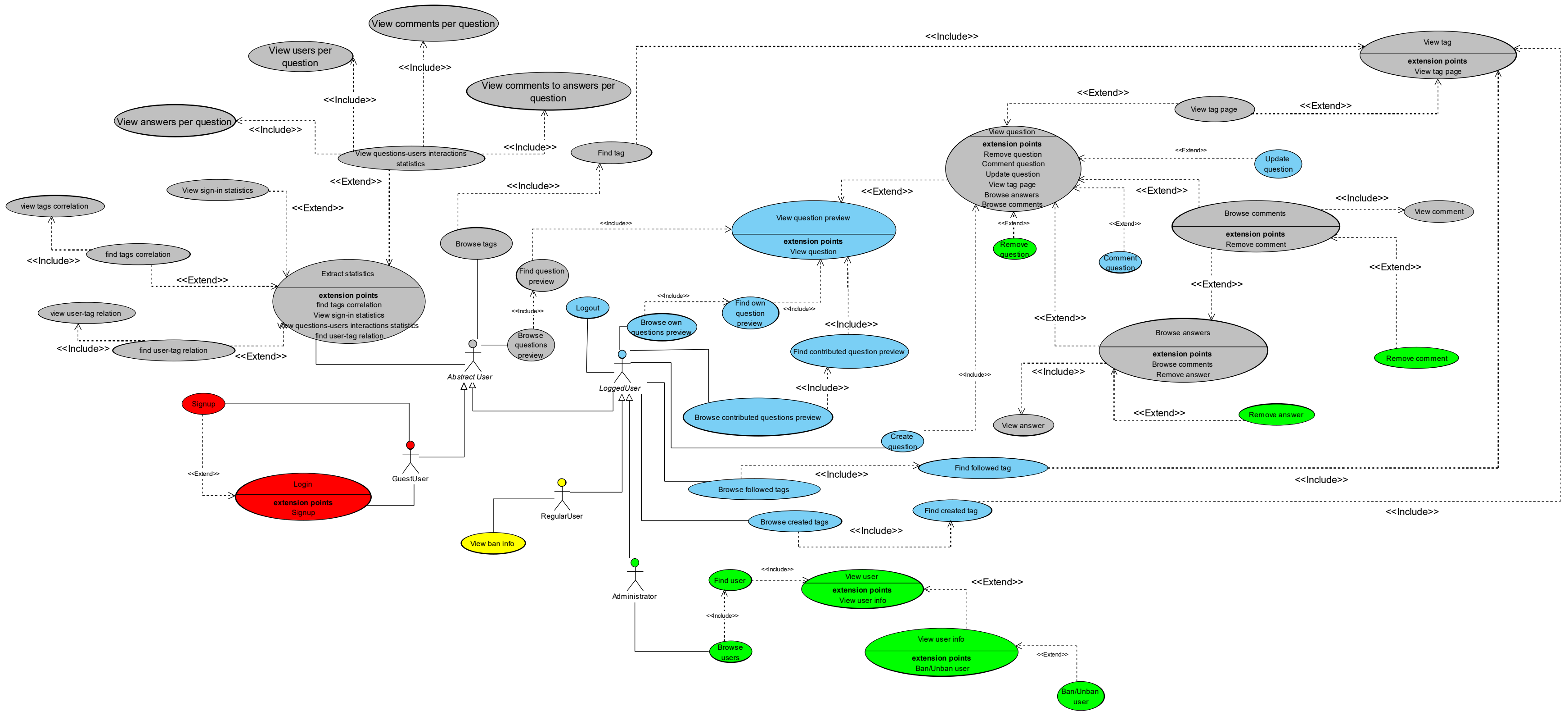


Figure 1: Use case diagram

## 3.2 Class Diagram

This is a brief description of the entities that are present in the class diagram, see Figure 2:

- **Question**: represents information about a question
- **Answer**: represents information about an answer to a question
- **Comment**: represents information about a comment
- **QuestionUpdate**: represents a user's interest in a question
- **Tag**: represents tags that summarize the topics a question is inherent to
- **LoggedUser**: abstract class representing any user registered on the site
- **Admin**: represents a site administrator
- **RegularUser**: represents a logged in user with the minimum (default for new users) number of privileges
- **Ban**: represents information related to a ban applied by an admin on a regular user

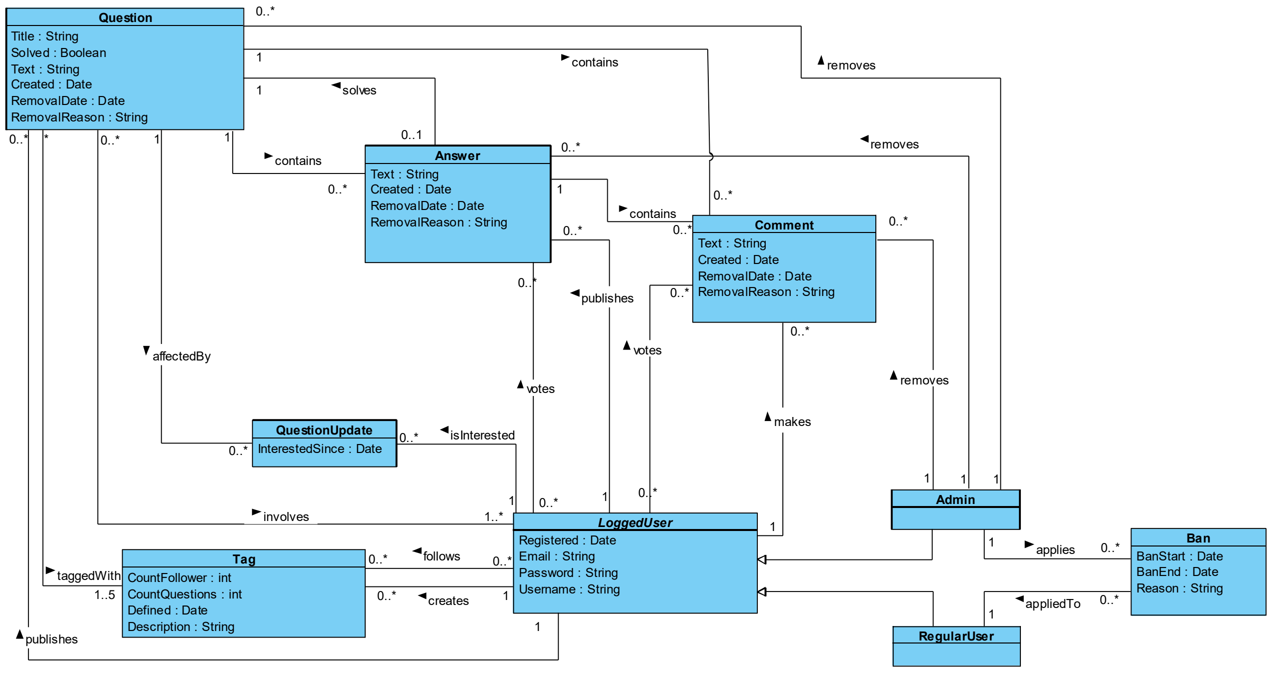


Figure 2: Class Diagram

# 4 Database

## 4.1 Dataset

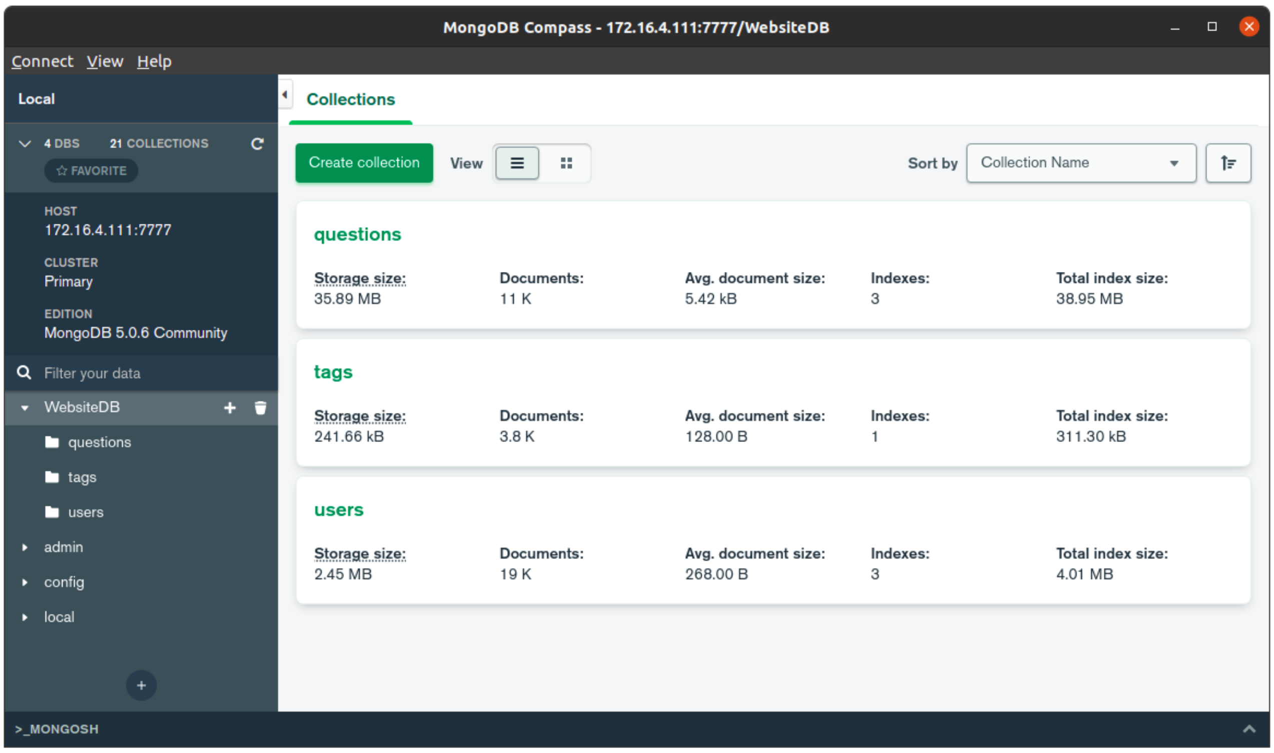
For the dataset we chose two very popular question and answer websites: stackoverflow.com and reddit.com, specifically the subreddit r/AskProgramming. In total our dataset was roughly 50MiB, see Figure 3

### 4.1.1 StackOverflow

Public download of the stackoverflow database are available publicly [1] in the format of a Microsoft SQL database, as of 2018 the database is 180GB in size, we used the version with data up to 2010, which was 10GB in size and thus much easier to handle. We did not use the whole content of the database to fill up our database, we used roughly 50MB worth of data. To copy the data from the Microsoft SQL database to the MongoDB one we wrote a python script that would retrieve data from the former, perform some data manipulation to match the format we needed, and send the data to special endpoints of the application web server that would finally insert them in the latter.

### 4.1.2 Reddit

The Reddit data is not available publicly for download, so we had to scrape the data with an automated script. Luckily Reddit is quite popular so there are plenty of libraries to interface with the Reddit api. We used the PRAW [2] library for python. In this case manipulating the data to make it fit in our MongoDB was trickier, since Reddit allows infinite nesting in the answer section, but our application only allows nesting with depth equal to two (answers and comment to answers). We decided to discard all answers with nesting greater than two, we considered answers from Reddit with nesting depth of one as answers in our application, and answers from Reddit with nesting depth of two as comment to answers in our application, it comes very naturally to parse the data in this way. We chose the subreddit r/AskProgramming because users ask questions that fit very well in our application.



HOST
172.16.4.111:7777
CLUSTER
Primary
EDITION
MongoDB 5.0.6 Community

| Collection | Storage size: | Documents: | Avg. document size: | Indexes: | Total index size: |
|---|---|---|---|---|---|
| questions | 35.89 MB | 11 K | 5.42 kB | 3 | 38.95 MB |
| tags | 241.66 kB | 3.8 K | 128.00 B | 1 | 311.30 kB |
| users | 2.45 MB | 19 K | 268.00 B | 3 | 4.01 MB |

Figure 3: Size of the collections in our mongo database

## 4.2 Mongo

MongoDB [3] is an extremely flexible document database management system that imposes no restrictions on how applications should use databases and collections so that they can be created on demand when the application first try to access them. We chose it as main database for our application to leverage on his ability to handle large data sets and support for full text searches, a fundamental feature in our project.

All the data managed by our application are stored in our mongo replicas and they are organised to limit the necessity of joins and multi-collections read operations that would be otherwise necessary with a relational database.

MongoDB has many useful characteristics but, due to the total freedom developers have on the document format, it might need to relocate documents in collections if they become bigger than their reserved space on the disk. To prevent an excessive number of relocations we chose to not end up with constantly growing documents by avoiding excessive costly redundances and to introduce a secondary database to easily store relationships between related data.

We made some hypothesis on data growth and evolution over the time, based on which we chose our solutions. Some analytics has been implemented to check the validity of these hypothesis and, in case they will reveal unexpected results in our website usage, new approaches to reorganise our system could be studied.

### 4.2.1 Entities

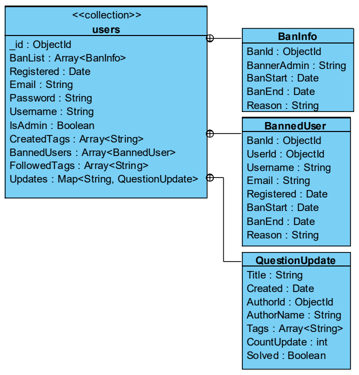


Figure 4: Users collection

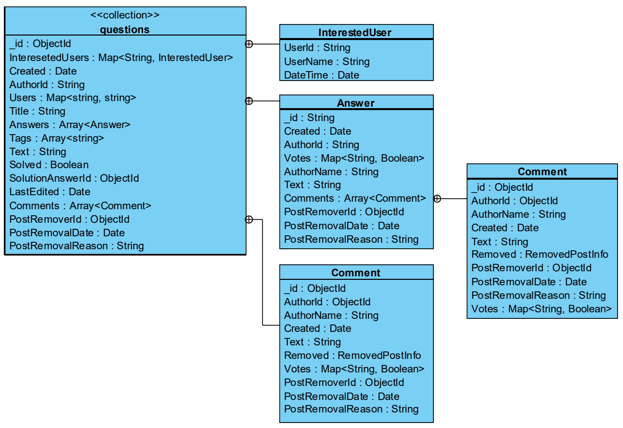


Figure 5: Questions collection

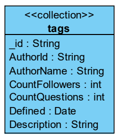


Figure 6: Tags collection

### 4.2.2 Indexes

In addition to the default indexes, the ones build on the _id property of the documents, we defined four more indexes, two on the collection "users" and two on the collection "questions".

**Indexes on "users"** We decided to force the uniqueness of the username chosen by the members of our community to avoid the unpleasant "who-is-who" problem. To obtain this we defined a unique index on the Username field of the user documents. We did this in our application code performing an action equivalent to de following one in mongosh syntax:

```
db.users.createIndex({ "Username": 1 },{ unique: true, })
```

We also needed to assert the uniqueness of the email addresses submitted in the Sign In form to prevent the same user from creating multiple accounts – and other user from shadowing accounts since only email address and password are required to log in:

```
db.users.createIndex({ "Email": 1 },{ unique: true, })
```

**Indexes on "questions"** We need to define two indexes on the "questions" collection to support and speed up the functionalities we wanted implemented. First, this website must allow users to search for previously answered questions, so we need a manner to search document based on their text content. This is an extremely complex result to be achieved in application code, fortunately we could leverage on the MongoDB support for full text searches [4]. To perform text searches special text indexes must be defined, so we defined a text index that cover on the question documents the fields "Title" and "Text". We did it in our application in a manner equivalent to the following mongosh command:

```
db.questions.createIndex( { Title: "text", Text: "text" } )
```

Moreover, we wanted users to be able to easily find the last questions using a specified tag, so we defined a compound index to get millisecond execution time with an expression equivalent to:

```
db.questions.createIndex( { Tags: 1, Created: -1 } )
```

### 4.2.3 Analytics

We defined and implemented four analytics in our mongo implementation to analyse the user interaction with our system and gain insight on their behaviour to verify the hypothesis that led to implementation decisions match the real system usage. We admit there are simple, but they are useful and grant valuable information about our system and this is the only point that matters.

**Community Growth** How does out community evolve? This is probably one of the most interesting information a website owner could ask for, both for planning future system upgrades and – why not – to proudly display it to new visitors and partners. To obtain this information we perform an aggregation on the collection "users" to count, for each day in which a new user became a member in our community, how many users joined us in that day. Having the possibility to plot diagrams, the result could be displayed in a time graph to easily recognise patterns in the community evolution. Not handling diagrams in our front-end, we display the results of this analytics in a table in the Statistics section of our website, showing for each day for which data are available the absolute and relative growth in that day.

**The query** As previously stated, the query is quite simple:

```
[{$group: {
_id: {
  $dateToString: {
   format: '%Y-%m-%d',
   date: '$Registered'
  }
},
count: {
  $sum: 1
}
}}]
```

Listing 1: Query for community growth in the mongo query language

```
// UsersService.cs
public async Task<IDictionary<string, int>> GetSignInStats()
{
  var pipeline = new BsonDocument[]
  {
    new BsonDocument("$group",
    new BsonDocument
      {
        { "_id",
    new BsonDocument("$dateToString",
    new BsonDocument
          {
            { "format", "%Y-%m-%d" },
            { "date", "$Registered" }
        }) },
      { "count",
    new BsonDocument("$sum", 1) }
      })
  };
  var res = await this.users.Aggregate<BsonDocument>(pipeline).ToListAsync();
  var ans = new Dictionary<string, int>();
  foreach (var doc in res)
  {
    ans[doc["_id"].ToString()] = doc["count"].ToInt32();
  }

  return ans;
}
```

Listing 2: Query for community growth in C#

We simply group users by registration date and then count them. Further elaborations, like the relative increment, are performed by the web server.

**Mean and standard deviation of the number of users interacting with a question** In mongo all the data related to a question are stored in the same document with multiple levels of nesting. These documents can quickly become very large – they contain all the text of questions, answers, and comments, and text could be very long, along with information about votes – forcing the DBMS to relocate them multiple times. We do not consider this an issue because this is a "questions and answers website", not a "discussion platform", we expect users to pose specific questions to obtain some insight, not to simply starts a discussion – it is not a case that we do not support multiple levels of nested comments – so we also expect the number of users interacting with a question to be generally small. The mean number of answers and comments per question are also expected to be low. Different results could suggest an unexpected use of the platform and require further investigations of the community behaviour.

We have mentioned that documents relocations in the collection "questions" might be required and are likely to occur as answers and comments are added to them, but until the number of users interacting with a question is limited, we could expect relocations of the same document to occur very few times, probably distributed across many hours or days. This is not a real time application, users ask to no one in particular and answers are likely to be waited for days before a volunteer come. Moreover, posts – be they answers or comment – are expected to be much more elaborated and reasoned than the one in a generic comment section as many social networks have, so we expect users to spend at least some minutes before posting more answers or comments. Having clarified that, cost of updates in this collection is not expected to be heavy, but this is likely to be a read heavy application, so storing all the data related to a question together and fetching them all with just one simple read could greatly improve performances, highlighting why a NoSQL DBMS is a good right choice in this context.

**The query** So, we have some hypothesis on our system usage, we just need to verify them. To do that we wrote this simple query:

```
[{$project: {
 Users: { $size: { $objectToArray: '$Users' } },
 Comments: { $size: { $ifNull: [ '$Comments', [] ] } },
 Answers: { $ifNull: [ '$Answers', [] ] }
}}, {$addFields: {
 CommentsToAnswers: {
  $sum: {
   $map: {
    input: '$Answers',
    'in': {
     $size: { $ifNull: [ '$$this.Comments', [] ] }
    }
   }
  }
 },
 Answers: { $size: '$Answers' }
}}, {$group: {
 _id: null,
 AvgUsers: { $avg: '$Users' },
 StdDevUsers: { $stdDevPop: '$Users' },
 AvgAnswers: { $avg: '$Answers' },
 StdDevAnswers: { $stdDevPop: '$Answers' },
 AvgComments: { $avg: '$Comments' },
 StdDevComments: { $stdDevPop: '$Comments' },
 AvgCommentsToAnswers: { $avg: '$CommentsToAnswers' },
 StdDevCommentsToAnswers: { $stdDevPop: '$CommentsToAnswers' }
}}]
```

Listing 3: Mean and standard deviation of the number of users interacting with a question in the mongo query language

```
// QuestionsService.cs
public async Task<QuestionStats> GetQuestionStats()
{
 var pipeline = new BsonDocument[]
 {
  new BsonDocument("$project", new BsonDocument
   {
    { "Users", new BsonDocument("$size", new BsonDocument("$objectToArray", "$Users")) },
    { "Comments", new BsonDocument("$size", new BsonDocument("$ifNull", new BsonArray { "$Comments", new BsonArray() })) },
    { "Answers", new BsonDocument("$ifNull", new BsonArray { "$Answers", new BsonArray() }) }
   }),
  new BsonDocument("$addFields", new BsonDocument
   {
    { "CommentsToAnswers", new BsonDocument(
     "$sum", new BsonDocument(
      "$map", new BsonDocument {
       { "input", "$Answers" },
       { "in", new BsonDocument("$size", new BsonDocument("$ifNull", new BsonArray { "$$this.Comments", new BsonArray() })) } })) },
    { "Answers", new BsonDocument("$size", "$Answers") }
   }),
  new BsonDocument("$group", new BsonDocument
   {
    { "_id", BsonNull.Value },
    { nameof(QuestionStats.AvgUsers), new BsonDocument("$avg", "$Users") },
    { nameof(QuestionStats.StdDevUsers), new BsonDocument("$stdDevPop", "$Users") },
    { nameof(QuestionStats.AvgAnswers), new BsonDocument("$avg", "$Answers") },
    { nameof(QuestionStats.StdDevAnswers), new BsonDocument("$stdDevPop", "$Answers") },
    { nameof(QuestionStats.AvgComments), new BsonDocument("$avg", "$Comments") },
    { nameof(QuestionStats.StdDevComments), new BsonDocument("$stdDevPop", "$Comments") },
    { nameof(QuestionStats.AvgCommentsToAnswers), new BsonDocument("$avg", "$CommentsToAnswers") },
    { nameof(QuestionStats.StdDevCommentsToAnswers), new BsonDocument("$stdDevPop", "$CommentsToAnswers") }
   })
 };

 var res = await this._questions.Aggregate<BsonDocument>(pipeline).FirstOrDefaultAsync();
 return new QuestionStats
 {
  AvgUsers = res?[nameof(QuestionStats.AvgUsers)].ToDouble() ?? 0.0,
  StdDevUsers = res?[nameof(QuestionStats.StdDevUsers)].ToDouble() ?? 0.0,
  AvgAnswers = res?[nameof(QuestionStats.AvgAnswers)].ToDouble() ?? 0.0,
  StdDevAnswers = res?[nameof(QuestionStats.StdDevAnswers)].ToDouble() ?? 0.0,
  AvgComments = res?[nameof(QuestionStats.AvgComments)].ToDouble() ?? 0.0,
  StdDevComments = res?[nameof(QuestionStats.StdDevComments)].ToDouble() ?? 0.0,
  AvgCommentsToAnswers = res?[nameof(QuestionStats.AvgCommentsToAnswers)].ToDouble() ?? 0.0,
  StdDevCommentsToAnswers = res?[nameof(QuestionStats.StdDevCommentsToAnswers)].ToDouble()
      ↪ ?? 0.0,
 };
}
```

Listing 4: Mean and standard deviation of the number of users interacting with a question in the C#

It might appear rather complex, but the result is simple the set of means and standard deviations of the number of users interacting with a question, the number of answers, comments to the question, and the total number of comments to the answers. This pipeline is a quick way to note anomalies in our platform usage. Of course, if strange results were obtained it would be easy to write other histogram-based analytics but starting by filling pages with statistics is rarely the best approach to choose when a simpler one gives similar insights in a more concise format.

Let's discuss what the pipeline does. First, it scans all the documents in the collection, keeping – or adding, if missing – only fields about users, answers, and comments, then it counts the number of users interacting with a question and the number of comments to the question. In the second stage, it computes the total number of answers and comments to answers per question. In the third and last stage, the means and the standard deviations of the searched parameters are computed.

As always, the results of this analytics are showed in the section "Statistics" of the website.

**Distribution of # of followed tags and # of users following a tag** These two analytics are performed together and displayed in the same page of the Analytics section of our website, so we describe them together. Their names could appear a little confusing and, in fact, they are conceptually similar: the only difference is the point of view from which they display the same information.

One possibility offered to our users is to follow a tag, that means it will appear in their personal page so they can easily find the last questions using them. While defining the structure of our collections, we decided to store the names of followed tags in a set-like array in the documents on the collection "users". We did this to easily build a user home page by fetching quite all the required information for that in the same location. In each document in the collection "tags" is maintained a counter – non a set – of the users following it. We chose this approach because we expect users to follow a reduced number of tags, but we expect some tags to be followed by a huge group of users. Storing the set of users following a specific tag in the tag document could then led to an enormous number of relocations to be necessary – imagine thousand users start following a specified tag, that could require thousands of relocations which is generally bad.

**The queries** The purpose of these two analytics is to verify we are right in assuming these behaviours. Both produce histograms-like results, respectively from the collections "users" and "tags.

The first one presented here is the one performed on the collection "tags" count how many tags have a certain number of followers – that is, how many tags are followed by 0, 1, 2, 3, N users:

```
[{$project: {
 CountFollowers: {
  $ifNull: [
   '$CountFollowers',
   0
  ]
 }
}}, {$group: {
 _id: '$CountFollowers',
 count: {
  $count: {}
 }
}}]
```

Listing 5: Distribution of # of users following a tag in the mongo query language

```
// TagsService.cs
public async Task<UserTagStats> FollowedTagsStats()
{
 var ans = new UserTagStats
 {
  TagsPerUser = new SortedDictionary<int, int>(),
  UsersPerTag = new SortedDictionary<int, int>(),
 };
 var tagsPerUserPipeline = new BsonDocument[]
 {
  new BsonDocument("$project", new BsonDocument(
   "CountFollowers", new BsonDocument("$ifNull", new BsonArray
   {
    "$CountFollowers",
    0
   }))
  ),
  new BsonDocument("$group", new BsonDocument {
     { "_id", "$CountFollowers" },
     { "count", new BsonDocument("$count", new BsonDocument()) },
  })
 };
 var usersPerTagPipeline = new BsonDocument[]
 {
  new BsonDocument("$project",
   new BsonDocument("followedTagsCount",
    new BsonDocument("$cond", new BsonDocument {
     { "if", new BsonDocument("$isArray", new BsonArray { "$FollowedTags" }) },
     { "then", new BsonDocument("$size", "$FollowedTags") },
     { "else", 0 }
    }))),
   new BsonDocument("$group", new BsonDocument {
     { "_id", "$followedTagsCount" },
     { "count", new BsonDocument("$count", new BsonDocument()) }
    })
 };

 var tagsPerUserTask = this._tags.Aggregate<BsonDocument>(tagsPerUserPipeline).ToListAsync()
    ↪ ;
 var usersPerTagHisto = await this._users.Aggregate<BsonDocument>(usersPerTagPipeline).
    ↪ ToListAsync();
 var tagsPerUserHisto = await tagsPerUserTask;

 foreach (var doc in usersPerTagHisto)
 {
  ans.UsersPerTag[doc["_id"].ToInt32()] = doc["count"].ToInt32();
 }
 foreach (var doc in tagsPerUserHisto)
 {
  ans.TagsPerUser[doc["_id"].ToInt32()] = doc["count"].ToInt32();
 }

 return ans;
}
```

Listing 6: Distribution of # of followed tags and # of users following a tag in C#

The first stage assures each document has the field CountFollowers defined[3]. The second one does the real

[3]Tags with no followers could have this field missing.

work; it counts how many tags have the same number of followers. We expect the variance of the result to be relatively high, some tag will have many followers and others around zero. The other analytic is like this but on the collection “users”, it counts the number of users that follow the same number of tags – that is different from the number of users that follow the same tags – which give us an information on the size of the array of followed tags in their document. We are interested in this information to estimate the number of relocations user’s documents require over the time and if this value would reveal to be big – and we do not expect that, in fact we expect the mean and the variance to be little, having more than 20 followed tags seems too much to us – a redefinition of our collections could be required[4].

Here is the code:

```
[{$project: {
 followedTagsCount: {
  $cond: {
   'if': { $isArray: [ '$FollowedTags'] },
   then: { $size: '$FollowedTags' },
   'else': 0
  }
 }
}}, {$group: {
 _id: '$followedTagsCount',
 count: {
  $count: {}
 }
}}]
```

Listing 7: Distribution of # of followed per user in the mongo query language

As the previous one, the first stage asserts the user document contains the required field, the list of followed tags, and if available get its size, otherwise store 0 in a temporary field. The second one performs the count of the number of users who follow the same number of tags.

As previously said, the results are displayed in the same page under the section “Statistics” of the website and, due to the lack of support for front-end diagrams, they are displayed in two tables.

## 4.3 Neo4j

Neo4j [5] is a graph database management system, an extremely interesting technology that allows extremely fast queries on graph-like data. We chose it to maintain a subset, without replicating the posts content to limit waste of space, of the data stored in our mongo replicas to permit fast relation-based searches. Data are organised in nodes and relationships just as in many graph systems and properties (values associated to objects) can be created on demand. We did not plan to store massive objects in this database but many small entities, so we do not expect many relocations or shrinks of stored items to be required and, consequently of our choice of not storing voluminous texts here, we do not expect the limitations of the community version (e. g.: no replication support) to be a problem for us.

### 4.3.1 Entities

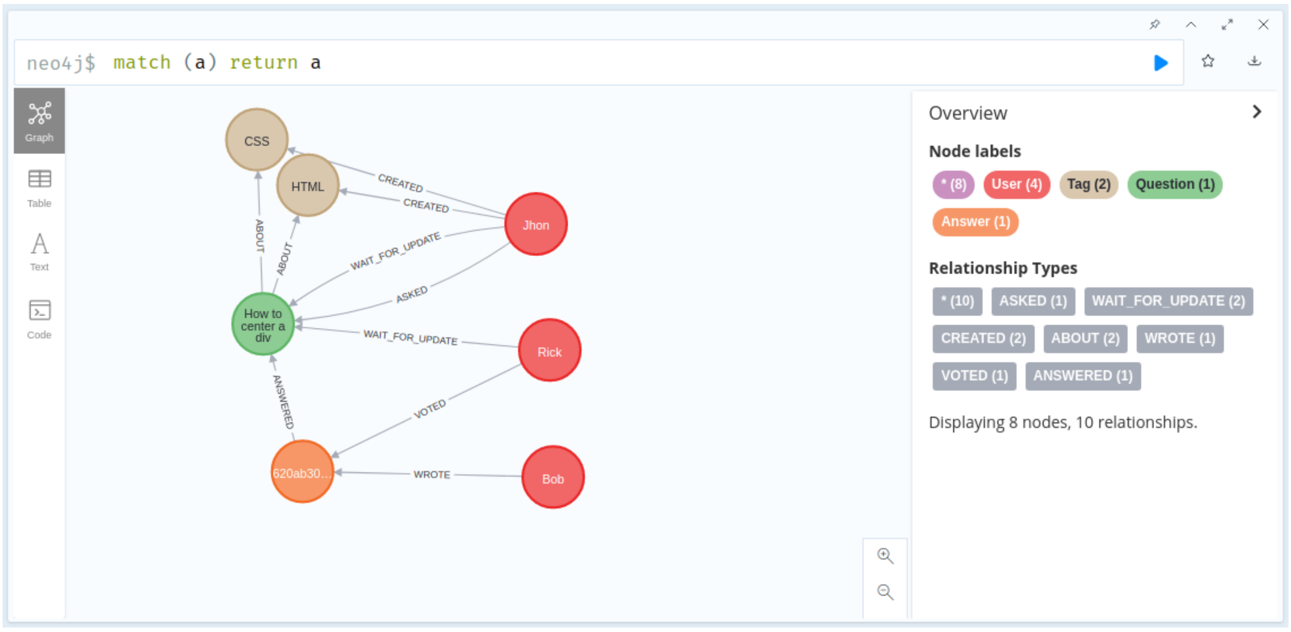
Entities represented in the graph are just the same as in mongo, except they maintain only a subset of the original attributes. Nevertheless, they hold always enough information to find the corresponding entity and the missing values in the other db.

The complete list of entities we used with neo4j is the following:

The complete list of entities we used with neo4j is the following:

- Question: `{Id: string, Title: string, Created: DateTime, Solved: bool}`
- Answer: `{Id: string}`
- Comment: `{Id: string}`
- User: `{Id: string, Username: string}`
- Tag: `{Id: string}`

[4]For example, we could decide to pre-allocate some space for the tags.



```
neo4j$ match (a) return a
```

Displaying 8 nodes, 10 relationships.

Figure 7: Neo4j minimal database example

It should be clear that they correspond to the same elements in the mongo database, but they keep only a reduced set of the original properties. Other information is stored as relationships:

```
(:User)-[:ASKED]->(:Question)
(:Question)->[:ABOUT]->(:Tag)
(:User)-[:CREATED]->(:Tag)
(:User)-[:FOLLOWS]->(:Tag)
(:User)-[:WAIT_FOR_UPDATE]->(:Question)
(:Answer)-[:ANSWERED]->(:Question)
(:User)-[:WROTE]->(:Answer)
(:Comment)-[:REFERS_TO]->(:Question)
(:Comment)-[:REFERS_TO]->(:Answer)
(:User)-[:COMMENTED]->(:Comment)
(:User)-[VOTED {Useful: bool}]->(:Answer)
(:User)-[VOTED {Useful: bool}]->(:Comment)
(:User)-[:HID]->(:Question)
(:User)-[:HID]->(:Answer)
(:User)-[:HID]->(:Comment)
```

Relationship names have been chosen to be self-explanatory so here we limit to add some notes:

`[:WAIT_FOR_UPDATE]` means a user has subscribed a question to be notified for updates when they come.

`[:ANSWERED]` means the question refers to a specific question, it does not imply it is the solution.

`[:COMMENTED]` should be intended as "wrote".

`[:HID]` is the past participle of "hidden", it is used to remember the administrator who obscured a certain post.

### 4.3.2 Indexes

We defined just one unique index for each type of entity we used in Neo4j on the Id field, that is just a string representing the ObjectId of the same object stored in the MongoDB database.

The used queries are all in the form: `CREATE CONSTRAINT IF NOT EXISTS FOR (x:Type)REQUIRE x.Id IS UNIQUE` where Type is replaced by each of the entity names (labels) listed in the previous paragraph.

These indexes have been defined because all the queries we executed on the Neo4j DB have their starting node uniquely identified by its id.

### 4.3.3 Analytics

We implemented three complex operations on our Neo4j database.

**Count votes received by a user on his answers and comments** Our website allows users to vote comments and answers, so it is interesting to know for a user how many positive and negative votes he received. For that, we implemented the following Cypher query.

```
MATCH (u:User {Id: $userId})
WITH u
OPTIONAL MATCH (u)-[:WROTE]->(:Answer)<-[v:VOTED]-(:User)
WITH u, count(CASE v.Useful WHEN true THEN 1 ELSE null END) as aLike,
count(CASE v.Useful WHEN false THEN 1 ELSE null END) as aDislike
OPTIONAL MATCH (u)-[:COMMENTED]->(:Comment)<-[v:VOTED]-(:User)
WITH u, aLike, aDislike,
count(CASE v.Useful WHEN true THEN 1 ELSE null END) as cLike,
count(CASE v.Useful WHEN false THEN 1 ELSE null END) as cDislike
RETURN aLike AS ALike, aDislike AS ADislike, cLike AS CLike, cDislike AS CDislike
```

Listing 8: Count votes received by a user on his answers and comments query in cypher language

- Graph-centric query: starting from a User node identified by a given Id find all the Answer or Comment nodes written by him such that another user8 voted them and count separately likes and dislikes considering if they refers to answers or comments.
- Domain-specific query: count all the votes received by a user on his answers and comments

```
public async Task<UserVoteStats> CountVotesReceivedByUser(string userId)
{
 var query = this.graphClient
  .Cypher
  .Match($"(u:{nameof(User)} {{Id: ${nameof(userId)}}})")
  .With("u")
  .OptionalMatch($"(u)-[:WROTE]->(:{nameof(Answer)})<-[v:VOTED]-(:{nameof(User)})")
  .With("u, count(CASE v.Useful WHEN true THEN 1 ELSE null END) as aLike, count(CASE v.
     ↪ Useful WHEN false THEN 1 ELSE null END) as aDislike")
  .OptionalMatch($"(u)-[:COMMENTED]->(:{nameof(Comment)})<-[v:VOTED]-(:{nameof(User)})")
  .With("u, aLike, aDislike, count(CASE v.Useful WHEN true THEN 1 ELSE null END) as cLike,
     ↪ count(CASE v.Useful WHEN false THEN 1 ELSE null END) as cDislike")
  .WithParam(nameof(userId), userId)
  .Return((aLike, aDislike, cLike, cDislike) => new UserVoteStats
  {
   ALike = aLike.As<int>(),
   ADislike = aDislike.As<int>(),
   CLike = cLike.As<int>(),
   CDislike = cDislike.As<int>(),
  });
 try
 {
  var ans = await query.ResultsAsync;
  return ans.FirstOrDefault() ?? new UserVoteStats();
 }
 catch (Exception)
 {
  if (!this.ignoreConnectionFault)
   throw;
  return new UserVoteStats();
 }
}
```

Listing 9: Count votes received by a user on his answers and comments query in C#

**Tag co-usages** Questions can be marked with tags so it could be interesting find if some relationships among them exists and how often two tags are used together. To get this information we defined the following query that finds all the tags used with a given one and how many times each both are used together.

```
MATCH (t1:Tag {Id: $tag})--(q:Question)--(t2:Tag)
WHERE t1 <> t2
RETURN t2.Id AS Name, count(q) AS Count
```

Listing 10: Tag co-usages query in cypher language

```
public async Task<IDictionary<string, long>> GetTagCousages(string tag)
{
 var ans = new Dictionary<string, long>();
 var query = this.graphClient
  .Cypher
  .Match($"(t1:{nameof(Tag)} {{Id: $tag}})--(q:{nameof(Question)})--(t2:{nameof(Models.
      ↪ Discussions.Tag)})")
  .Where("t1 <> t2")
  .WithParam(nameof(tag), tag)
  .Return((t2, q) => new
  {
   Name = t2.As<Tag>().Id,
   Count = q.Count(),
  });
 try
 {
  var res = await query.ResultsAsync;
  foreach (var item in res)
  {
   ans.Add(item.Name, item.Count);
  }
  return ans;
 }
 catch (Exception)
 {
  if (!this.ignoreConnectionFault)
   throw;
  return ans;
 }
}
```

Listing 11: Tag co-usages query in C#

- Graph-centric query: given a Tag node, for each other Tag node for which a Question node connected to both exists count how many questions connecting both exist
- Domain-specific query: which tags are used with the given one and how many times?

This query will be executed for all the tags listed in the corresponding form in the section "Statistics" of the website.

**Find all questions a user contributed to without being the asker** Sometimes a user might want to find a question he contributed to, by writing answers or comments. To find them all we implemented the following Cypher query:

```
MATCH (u:User {Id: $userId})
MATCH (q:Question)<-[:ASKED]-(author:User)
WHERE
```

```
  NOT (u)-[:ASKED]->(q)
  AND
  (
    (u)-[:WROTE]->(:Answer)-[:ANSWERED]->(q)
  OR
    (u)-[:COMMENTED]->(:Comment)-[:REFERS_TO]->(q)
  OR
    (u)-[:COMMENTED]->(:Comment)-[:REFERS_TO]->(:Answer)-[:ANSWERED]->(q)
  )
MATCH (q)-[:ABOUT]->(t:Tag)
WITH *, t.Id as tag
RETURN distinct q.Id AS Id, q.Title AS Title, q.Created AS Created,
collect(distinct tag) AS Tags, author.Id AS AuthorId,
author.Username AS AuthorName
ORDER BY q.Created DESC
```

Listing 12: Query to find all questions a user contributed to without being the asker in cypher language

```
public async Task<IEnumerable<Question>> FindDiscussionsUserContributedTo(string userId)
{
 var query = this.graphClient
  .Cypher
  .Match($"(u:{nameof(User)} {{Id: ${nameof(userId)}}})")
  .Match($"(q:{nameof(Question)})<-[:ASKED]-(author:{nameof(User)})")
  .Where("NOT (u)-[:ASKED]->(q) AND ("
  + $" (u)-[:WROTE]->(:{nameof(Answer)})-[:ANSWERED]->(q)"
  + $" OR (u)-[:COMMENTED]->(:{nameof(Comment)})-[:REFERS_TO]->(q)"
  + $" OR (u)-[:COMMENTED]->(:{nameof(Comment)})-[:REFERS_TO]->(:{nameof(Answer)})-[:
     ↪ ANSWERED]->(q)"
  + ")"
  )
  .Match($"(q)-[:ABOUT]->(t:{nameof(Tag)})")
  .With("*, t.Id as tag")
  .WithParam(nameof(userId), userId)
  .ReturnDistinct((q, author, tag) => new Question
  {
   Id = q.As<Question>().Id,
   Title = q.As<Question>().Title,
   Created = q.As<Question>().Created,
   Tags = tag.CollectAsDistinct<string>(),
   AuthorId = author.As<User>().Id,
   AuthorName = author.As<User>().Username,
  })
  .OrderByDescending($"q.{nameof(Question.Created)}");
 try
 {
  return await query.ResultsAsync;
 }
 catch (Exception)
 {
  if (!this.ignoreConnectionFault)
   throw;
  return Enumerable.Empty<Question>();
 }
}
```

Listing 13: Query to find all questions a user contributed to without being the asker in C#

### 4.4 Consistency between databases

Since we had no way to tie transactions executed in mongo with transactions executed in neo4j we add some application code to optionally force automatic database re-synchronization on admin request.

### 4.5 Sharding and Replicas

Mongo is deployed as a replica set of three nodes with write concern equal to two and read preference set to nearest.

Sharding on Mongodb is not convenient due to the reduced set size (GBs, not TBs) and the central importance of full text searches on the only voluminous collection, that would span on all shards.

Replicas and sharding is not available in our version of Neo4j.

## 5 Software Architecture

**.NET** .NET (pronounced dotnet) is an open-source development platform8 developed by Microsoft for building many types of applications, spacing from class command line applications to modern mobile and web apps, including real-time and machine learning projects. It is important to note that it does not force the developers to use a specific language, but they are free to use any among C#, F#, VB.NET, C++, PowerShell scripts and mixing libraries written in any of them in the same project.

We chose dotnet and C# to leverage on an extremely advanced reflection system that allow developers to write common language expressions –checked by the compiler, then avoiding the risk of dummy runtime errors – that library function can parse – that is, the code itself inspects and analyses code expression – and translate into protocol specific formats. We could spend tens of pages trying to describe it in detail but an example values more than thousand words, so let see an example based on MongoDB C# driver extracted by our code:

```csharp
// Code extracted from QuestionsService.cs:
// method private field
private readonly IMongoCollection<Question> _questions;
// in public async Task<Question> Get(string id)
var q = await this._questions.Find(q => q.Id == id).FirstOrDefaultAsync();
```

Listing 14: C# reflection example

The first line of code should be auto explicative, we just defined an instance private field that is initialised in the class constructor as readonly [6] requires; the field is a reference to an object implementing the `IMongoCollection<T>` interface10, so it is used to handle a MongoDB collection whose documents are mapped to objects of type T. The second line is much more interesting, the presence of the async/await keywords imply the statement is executed asynchronously – that is, a sort of intra application scheduling is used to avoid blocking a thread waiting for the database call termination – but in this explaination it can be considered as executing sequentially11. The point on which we want to concentrate is the Find12 method that is invoked with a lambda function as parameter. An unexperienced user would think that it causes the application to download all the documents in the collection from the DBMS and filter them only in application causing an enormous amount of data to be transferred and discarded. That is where dotnet magic comes in help! The lambda function is not executed at all, it is examinated and translated into mongo query language! If we carefully look at the driver documentation, we can see the filter parameter has type `System.Linq.Expressions.Expression<Func<TDocument, Boolean>>`, that means it is not a lambda but an object describing the expression used in the given lambda, so the consumer can detect the developer required an equality match on the _id13 field of the documents of the collection and make the DBMS to handle it.

As last note on dotnet traits we mention, C# has a syntax like the Java one but a more complex type-system and a full support for "template like" generics that keeps type information at runtime, then without the limitations caused by the erasure mechanism.

### 5.1 Our Application

Our application is an ASP.NET Core [7] Model-View-Controller (MVC) web application. We chose this framework to leverage on his advanced features to focus on the functionalities we required and not on how they are provided. As example, the framework provides an extremely useful dependency injection system and integrated support for handling authentications and authorisation policies, moreover it defines many constructs to easily build views[5] mixing C# code and HTML in a safe manner. Finally, the framework can automatically serialize and deserialize API [8] argument leaving the developers free from writing a lot of boiled plate code.

It is a Model View Controller [9], that means there is:

- A set of Controllers handling requests routing and Inversion of Control1 [10] pattern that delegates requests to user defined code, defined in the Controllers folder.
- A set of injectable Services provided by the Dependency Injection (DI) system to the controllers, handling the interactions with the databases representing the Model, defined in the Services folder.
- A set of Views used to present results to users in a beautiful and clean manner, under the Views folder.

### 5.1.1 The structure of the project

ASP.NET Core MVC projects generally match the following folder organisation, that usually is in a one-to-one mapping with the namespaces[6] hierarchy.

Starting from the base folder, this is the core of our project structure:

- appsetting.json: maintains settings to configure the web server at the start-up.
- Program.cs: contains the entry point of the system, i.e., the Main function.
- Startup.cs: contains code for parsing appsetting.json, registering services, handling authentication/authorisation and mapping controllers and views.
- Models/: it is only a place where conventionally define all the classes holding data in the application. There is no specific reason to use this name, it just a convention.
- Services/: here are usually defined services, that are simply normal classes provided to the DI system. There is no specific reason to use this name, it just a convention.
- wwwroot/: this is a special folder used to serve static files. It is usually used to delivery static CSS and JS library files.
- Views/: here are defined the views[11] used by the MVC pattern. This folder contains many subfolders that match the name of the controllers and sole special files beginning with an underscore, mainly under the Shared/ folder. When a controller returning a page execute and successfully return the system look for the corresponding view in this folder and will use it to build the returned page. Files beginning with underscore are partial view21, that means they can be used in other views to render a limited part of the content, for example a form. The special _Layout[7] file contains the common structure to all the website pages – that is, the navbar, the footer and so on – that is automatically merged with the result of view processing.

We have used the convention of naming namespaces as the folder they are defined in.

### 5.1.2 Our services

In our project we defined seven services, four to be used directly by controllers and three to be used internally to hide implementation details.

**Implementation detail services** These services

- Neo4jService: handles all the operations involving the Neo4j database.
- MongoService: connects to the MongoDB replica set and provides objects to interact with collections to other services.
- MongoSessionBase: supply a protected method to ¡Collection¿Service to get an object to handle transactions.

[5]We opted for server side rendering just to speed up development, but it is easy to integrate it with client-side functionalities and dotnet also supports C# to WebAssembly compilation for building Single Page Application with client-side rendering.

[6]Namespace rules in C# are mostly the same than in C++, here they match folder organisation only for readability.

[7]Starting partial view names with underscore is just a convention and it is not strictly required.

**To-be-used-by-controllers services** These services offer a high-level view over the operations the application executes on the data, hiding all the details about the synchronisation of the used databases.

- UsersService
- TagsService
- QuestionsService
- SynchronizerService

Their names have been chosen to be self-explanatory, they handle operations related to the subject of their names. However, when some operation refers to more than one object – maybe because it involves users and tags – we located it in the service that whose name was considered the main subject of the operation.

**The flow of execution** The application behaves as follow:

1. The execution starts Main function in Program.cs that simply starts the webserver initialisation with using command line arguments and specify the Startup class as the one that will handle the remaining configurations.
2. All the methods in the Startup function in Startup.cs are invoked by the server builder, in order to parse appsetting.json, handle authentication/authorisation, register services and mapping controllers and views.
3. The application framework starts an infinite loop waiting for incoming requests:
   (a) As an HTTP request arrives it is parsed by the framework and the path is used for the next step.
   (b) If the path identifies a file in the wwwroot/ folder such file is sent back and the process restart from the beginning. Otherwise the path structure is expected to be in the following form `[controller name, default=Home]/[action name, default=Index][arguments]` and the process goes on.
   (c) The web server looks at the Controller folder[8] for a controller matching the request path. If nothing is found error 404 is returned.
   (d) In the found controller it looks for a method matching the specified action and that could accept the specified HTTP method. If nothing is found error 404 is returned.
   (e) An instance of the controller is constructed by the system to handle the request – controllers are transient, and a new instance is used for each incoming request – and required arguments are supplied by DI system.
   (f) The request arguments are deserialized and compared to the ones required by the found method, if possible, they are casted to the required types, and the method is invoked.
   (g) The method executes and returns an object representing the answer. Here the behaviour changes depending on the nature of the result.
      i. If the result is a "normal" C# object it is serialized, by default as JSON, and sent back in the HTTP response,
      ii. If the result is an object representing a view – usually obtained by invoking the View() method in the controller method – the webserver will search the Views folder[9] for a view matching the method name – if nothing else is specified – under a folder called as the controller. The execution flow is then transferred to the object representing the view.

## 5.2 Deployment

### 5.2.1 Docker

Docker [12] is a set of platforms as a service products that use OS-level virtualization to deliver software in packages called containers. It is used to combine a software application code and its dependencies as a single package, that can run independently in any computer environment, it does all that while being very lightweight and performant. One of the reasons why we chose to use docker is because it made working on the project seamless even if we had different operating systems as our primary workstations (Windows, MacOS and Linux) and made the process of deploying in the cloud quick and painless. Docker also allows to manage multiple containers and handle communication between them by emulating a layer two network between the containers. In our project we had one container for each application's services:

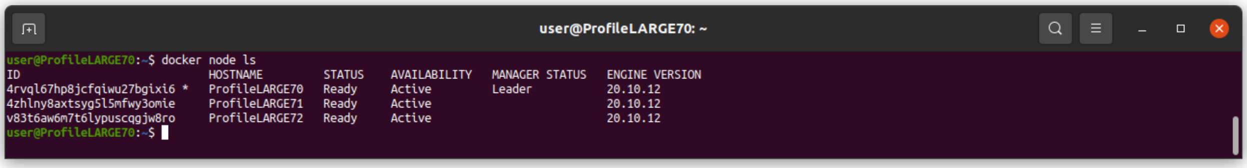
[8]Under the hood this is done by the reflection system inspecting the executable, not inspecting folders but the concept remains.
[9]As for controllers, this is not a real file search.

- Website: the main web server, it handles the application logic as well as the connection with the databases
- Mongodb{1-2-3}: the three replicas of the mongodb database
- Node4j: the node4j database
- Nginx: reverse proxy in front of the application, mainly for future ideas

Another reason why we chose Docker is because it natively can manage a cluster of Docker Engines with Docker Swarm with very little to none modification of our development setup.

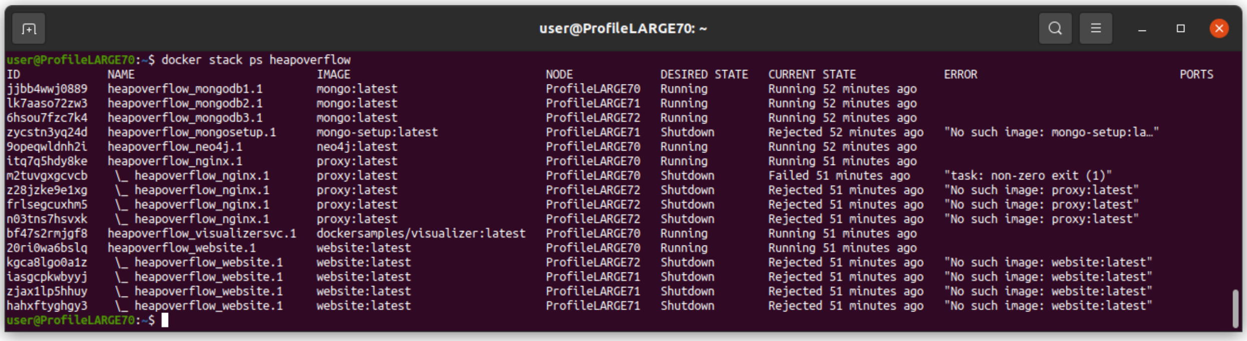
### 5.2.2 Cluster management

Docker swarm is a container orchestration tool, meaning that it allows us to manage multiple containers deployed across multiple host machines as if they where running on a single host. Docker stack allows to manage the deployment and maintenance of containers on a swarm, it can be viewed as docker-compose for a swarm. We first created a swarm containing the three servers and then we deployed the application on top of them, docker stack will automatically balance the containers evenly in the multiple servers.



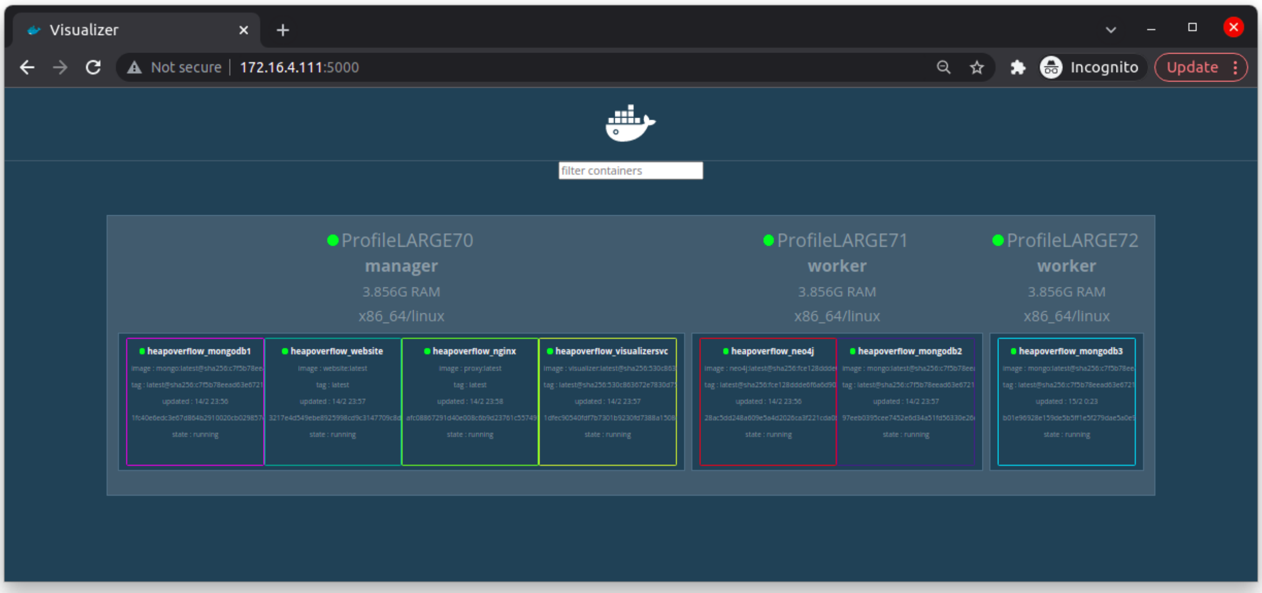
```
user@ProfileLARGE70:~$ docker node ls
ID                            HOSTNAME        STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
4rvql67hp8jcfqiwu27bgixi6 *   ProfileLARGE70  Ready     Active         Leader           20.10.12
4zhlny8axtsyg5l5mfwy3omie     ProfileLARGE71  Ready     Active                          20.10.12
v83t6aw6m7t6lypuscqgjw8ro     ProfileLARGE72  Ready     Active                          20.10.12
user@ProfileLARGE70:~$
```

Figure 8: Docker swarm with three nodes



```
user@ProfileLARGE70:~$ docker stack ps heapoverflow
ID             NAME                          IMAGE                                NODE             DESIRED STATE   CURRENT STATE            ERROR                                PORTS
jjbb4wwj0889   heapoverflow_mongodb1.1       mongo:latest                         ProfileLARGE70   Running         Running 52 minutes ago
lk7aaso72zw3   heapoverflow_mongodb2.1       mongo:latest                         ProfileLARGE71   Running         Running 52 minutes ago
6hsou7fzc7k4   heapoverflow_mongodb3.1       mongo:latest                         ProfileLARGE72   Running         Running 52 minutes ago
zycstn3yq24d   heapoverflow_mongosetup.1     mongo-setup:latest                   ProfileLARGE71   Shutdown        Rejected 52 minutes ago  "No such image: mongo-setup:la…"
9opeqwldnh2i   heapoverflow_neo4j.1          neo4j:latest                         ProfileLARGE70   Running         Running 52 minutes ago
itq7q5hdy8ke   heapoverflow_nginx.1          proxy:latest                         ProfileLARGE70   Running         Running 51 minutes ago
m2tuvgxgcvcb    \_ heapoverflow_nginx.1      proxy:latest                         ProfileLARGE70   Shutdown        Failed 51 minutes ago    "task: non-zero exit (1)"
z28jzke9e1xg    \_ heapoverflow_nginx.1      proxy:latest                         ProfileLARGE72   Shutdown        Rejected 51 minutes ago  "No such image: proxy:latest"
frlsegcuxhm5    \_ heapoverflow_nginx.1      proxy:latest                         ProfileLARGE72   Shutdown        Rejected 51 minutes ago  "No such image: proxy:latest"
n03tns7hsvxk    \_ heapoverflow_nginx.1      proxy:latest                         ProfileLARGE72   Shutdown        Rejected 51 minutes ago  "No such image: proxy:latest"
bf47s2rmjgf8   heapoverflow_visualizersvc.1  dockersamples/visualizer:latest      ProfileLARGE70   Running         Running 51 minutes ago
20ri0wa6bslq   heapoverflow_website.1        website:latest                       ProfileLARGE70   Running         Running 51 minutes ago
kgca8lgo0a1z    \_ heapoverflow_website.1    website:latest                       ProfileLARGE72   Shutdown        Rejected 51 minutes ago  "No such image: website:latest"
iasgcpkwbyyj    \_ heapoverflow_website.1    website:latest                       ProfileLARGE71   Shutdown        Rejected 51 minutes ago  "No such image: website:latest"
zjax1lp5hhuy    \_ heapoverflow_website.1    website:latest                       ProfileLARGE71   Shutdown        Rejected 51 minutes ago  "No such image: website:latest"
hahxftyghgy3    \_ heapoverflow_website.1    website:latest                       ProfileLARGE71   Shutdown        Rejected 51 minutes ago  "No such image: website:latest"
user@ProfileLARGE70:~$
```

Figure 9: Docker stack deployment on multiple nodes



● ProfileLARGE70
manager
3.856G RAM
x86_64/linux

● heapoverflow_mongodb1
image : mongo:latest@sha256:c7f5b78ee
tag : latest@sha256:c7f5b78eead63e6721
updated : 14/2 23:56
1fc40e6edc3e67d864b2910020cb029857
state : running

● heapoverflow_website
image : website:latest
tag : latest
updated : 14/2 23:57
3217e4d549ebe8925998cd9c3147709c8d
state : running

● heapoverflow_nginx
image : proxy:latest
tag : latest
updated : 14/2 23:58
afc08867291d40e008c6b9d23761c55749
state : running

● heapoverflow_visualizersvc
image : visualizer:latest@sha256:530c863
tag : latest@sha256:530c863672e7830d7
updated : 14/2 23:57
1dfec90540fdf7b7301b9230fd7388a1508
state : running

● ProfileLARGE71
worker
3.856G RAM
x86_64/linux

● heapoverflow_neo4j
image : neo4j:latest@sha256:fce128ddde
tag : latest@sha256:fce128ddde6f6a6d90
updated : 14/2 23:56
28ac5dd248a609e5a4d2026ca3f221cda0
state : running

● heapoverflow_mongodb2
image : mongo:latest@sha256:c7f5b78ee
tag : latest@sha256:c7f5b78eead63e6721
updated : 14/2 23:57
97eeb0395cee7452e6d34a51fd56330e26
state : running

● ProfileLARGE72
worker
3.856G RAM
x86_64/linux

● heapoverflow_mongodb3
image : mongo:latest@sha256:c7f5b78ee
tag : latest@sha256:c7f5b78eead63e6721
updated : 15/2 0:23
b01e96928e159de5b5ff1e5f279dae5a0e5
state : running

Figure 10: Overview of containers and hosts

## References

[1] https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent/.

[2] https://praw.readthedocs.io/en/stable/.

[3] https://www.mongodb.com/what-is-mongodb/.

[4] https://docs.mongodb.com/manual/core/link-text-indexes/.

[5] https://neo4j.com/.

[6] https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/readonly/.

[7] https://dotnet.microsoft.com/en-us/apps/aspnet/.

[8] https://dotnet.microsoft.com/en-us/apps/aspnet/apis.

[9] https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-6.0#:~:text=ASP.NET%20Core%20MVC%20The%20ASP.NET%20Core%20MVC%20framework,websites%20that%20enables%20a%20clean%20separation%20of%20concerns/.

[10] https://en.wikipedia.org/wiki/Inversion_of_control.

[11] https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor.

[12] https://www.docker.com/.