Large-Scale and Multi-Structured Databases

Project Design

LearnIt!

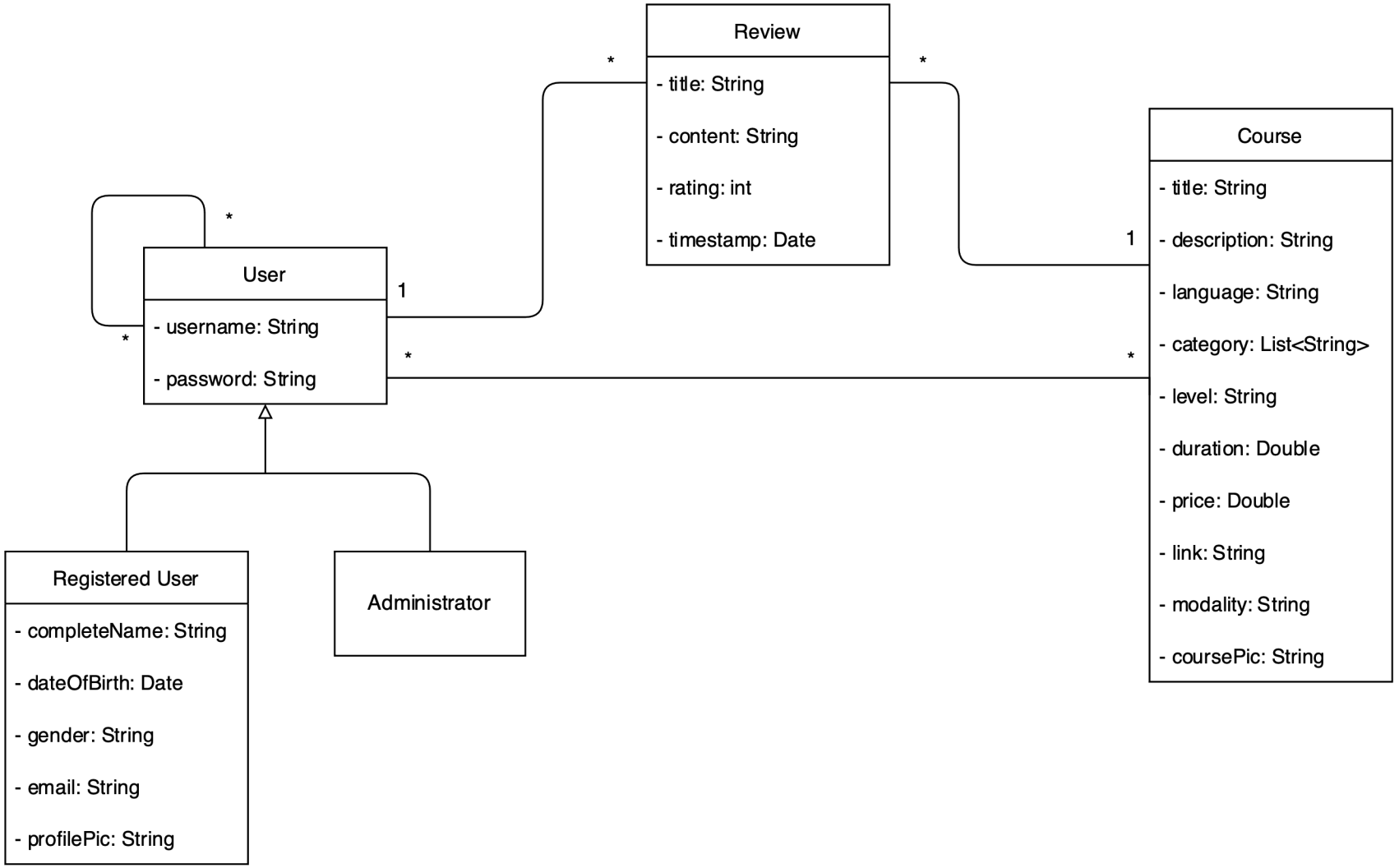
Federico Minniti
Francesca Pezzuti
Matteo Del Seppia

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

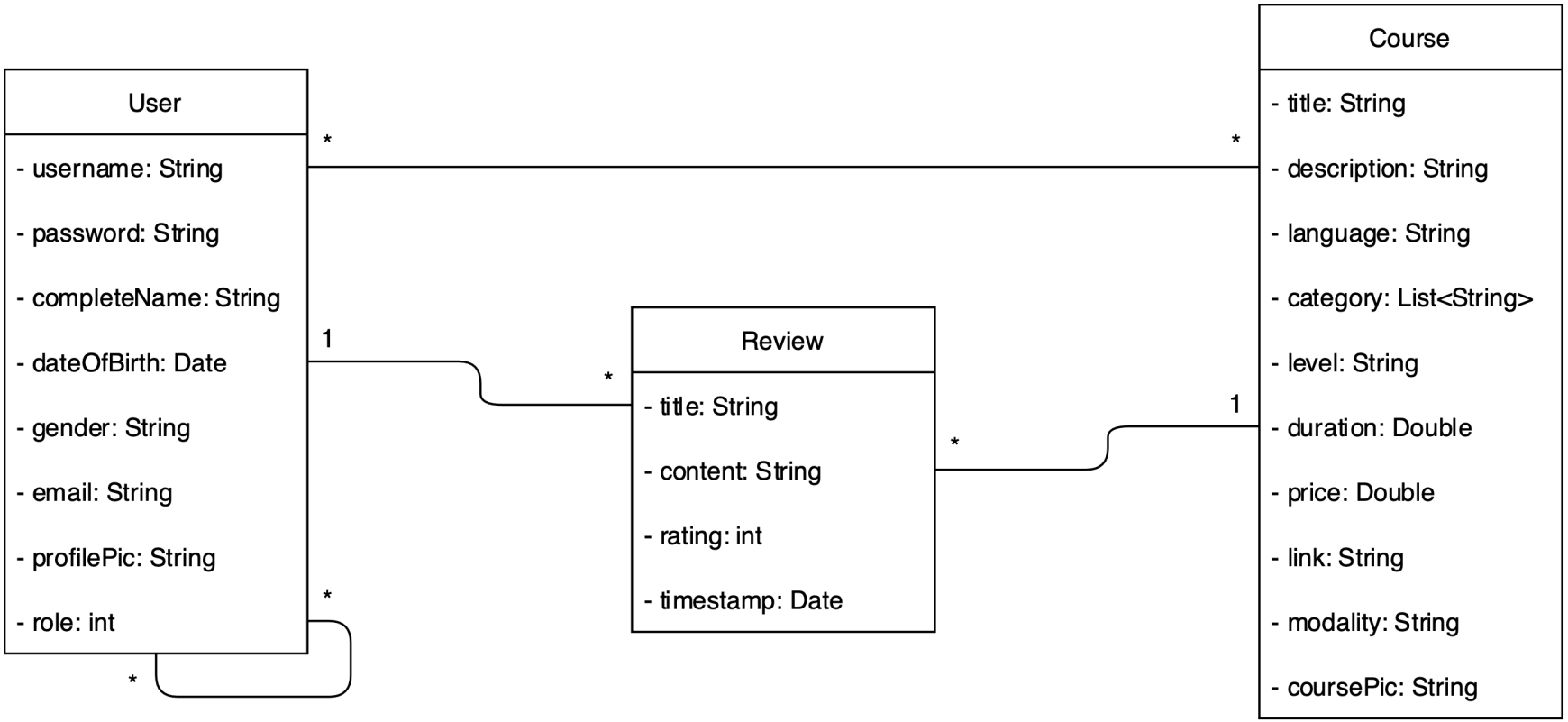
UNIVERSITÀ DI PISA

CROSSLAB
Innovation for industry 4.0

UML analysis class diagram



UML analysis class diagram



The generalization was solved by inserting the role attribute indicating whether the user is an administrator (A) or a registered user (RU). An RU can like 0 or N courses, a course can like 0 or N RU. An RU can follow and be followed by 0 or N users. An A can delete 0 or N courses/users/reviews, a course can be deleted from one and only one A.

Dataset Description

Source:
https://www.kaggle.com/trajput508/coursera-data
https://www.kaggle.com/rexxxxxxx/udemy-courses-datasets?select=courses_info_photoNvideo.csv
https://www.kaggle.com/koutetsu/udemy-courses-and-reviews?select=courses.yaml
Generated users and empty text reviews

Volume:
107.1 K different courses (original size 61.25 MB)
Variety:
We are using 2 datasets of courses taken from Udemy and Coursera. We have manipulated the 2 datasets in order to uniform data and apply attribute reduction when necessary
Velocity:
Variability and Velocity are ensured by adding courses and reviews

Dataset Description

Courses:

Courses and reviews are taken from the first and the third dataset.
Similar attributes have been standardized and put in the same format.
When irrelevant or website-specific attributes have been removed.

Reviews:

Complete revies (the ones that contains all the information: title, description, rating and date) have been taken and also standardized.
We also create with a Python script new ”white reviews” (reviews without description) to balance the number of reviews with the number of courses and to make the dataset more realistic. Total reviews at the end: around 541k.

User:

We have generated with a Python script 6.3K users, all with different complete names and usernames.
In order to cover all the dataset we assigned randomly 17 offered courses for each user. We gave to each courses 3-15 reviews wrote by random users.
Finally we have randomly created follow and like relationships.

Non-Functional requirements

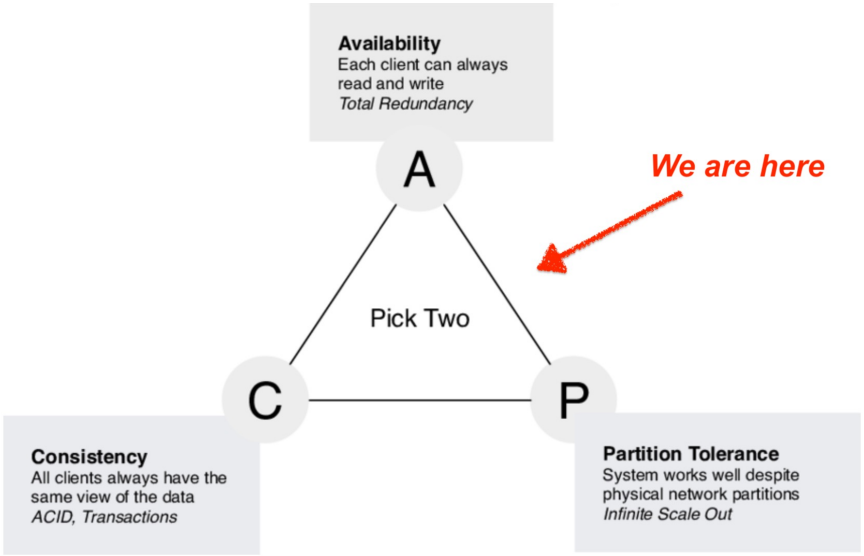
- **Usability**: the application must be user-friendly (with a simple GUI) and must ensure a low response-time.
- **Low latency**: the response to the requests should be fast.
- **Reliability**: the system must be stable, must return reproducible results and handle exceptions if needed.
- **Flexibility**: the user that makes a course available may choose only the fields they find appropriate to describe their course. The data should be handled in a flexible way.
- **Portability**: the system can be ported to different operating systems with no visible difference in its behavior.
- **Privacy**: user credentials must be handled securely
- **Maintainability**: the code should be maintainable and readable

Non-Functional requirements and **CAP** theorem

The application must provide ***availability*** because we want the service and contents to be always available for users. We also ensure ***partition protection*** (exploiting replicas in our cluster of servers) because the service must be tolerant to partitioning caused by failures. For that reason we adopt only eventual consistency on our dataset.

Therefore for MongoDB we opted for:

- Write concern = W1. It increases the speed of write operation because only waits for acknowledgement from a single member node.
- Read concern = NEAREST. Read operations are performed on the nearest node (the one who responds fastest in terms of ping).



Database design MongoDB

These documents structures allow us to embed objects in other objects that are frequently used together: reviews most of the times are showed with all the information of courses, and snapshots of courses reviewed by users are showed every times we open a user profile with all the user information.


```
{
  "_id": {
    "$oid": "61bdf95cb37a46697e47da1b"
  },
  "category": [
    "Teaching & Academics",
    "Language Learning",
    "Spanish Language"
  ],
  "description": "The complete, non-stop SPEAKING Spanish course:
      Mastery of the basics for BEGINNERS - in a matter of hours, not years.",
  "instructor": "peter1",
  "language": "English",
  "duration": 4,
  "level": "Beginner",
  "link": "https://www.udemy.com/course/el-metodo-spanish-1/",
  "title": "Spanish for Beginners. The Complete Method. Level 1",
  "price": 14.99,
  "course_pic": "https://i.imgur.com/R7KOk45.png"
  "num_reviews" : 2,
  "sum_reviews" : 9,
  "reviews":·[
     {
       "content": "",
       "edited_timestamp": "2021-07-17T16:06:39Z",
       "rating": 4,
       "title": ""
       "username": "federic107"
     },
     {
        "content": "I like that is very well structured for 1 year of practice.
            Besides,\nit's easy to learn and to apply the knowledge.",
        "edited_timestamp": "2021-01-10T15:07:37Z",
        "rating": 5,
        "title": "",
        "username": "john9"
     }
  ]
}
```

```
[{
  "_id": {
    "$oid": "61f2bfc0d16c2b599a15a000"
  },
  "username": "adamanderson",
  "complete_name": "Adam Anderson",
  "password": "0f424c6e6d0a096b897a26b05423ed1b09c0c1ca6b1bc4623567f864f30bc01c",
  "birthdate": {
    "$date": "2000-10-27T00:00:00Z"
  },
  "gender": "man",
  "pic": "https://tinyurl.com/yckufye8",
  "email": "adam.anderson@email.com",
  "role": 0,
  "reviewed": [
    {
      "title": "Java Data Structures and Algorithms Masterclass",
      "duration": 45.5,
      "course_pic": "https://i.imgur.com/R7KOk45.png"
    }
  ]
}]
```

Most relevant queries on MongoDB

- View annual avg rating

```
db.courses.aggregate([
 {$match: {_id: ObjectId('<id>')}},
 {$unwind: {path: '$reviews'}},
 {$group:
       {_id: {year:
              {$year: '$reviews.edited_timestamp'}
            },
        sum_ratings: {$sum: '$reviews.rating'},
        num_reviews: {$sum: 1}
       }
 },
 {$set: {year: '$_id.year'}},
 {$sort: {year: -1}}
])
```

- Most active users

```
db.users.aggregate([
 {$match: {reviewed: {$exists: true}}},
 {$unwind: {path: '$reviewed'}},
 {$match: {'reviewed.review_timestamp': {$gt: <date>}}
    },
 {$group: {_id: {username: '$username'},
           count: {$sum: 1}
          }
 },
 {$sort: {count: -1}},
 {$project: {username: '$_id.username', _id: 0}},
{$limit: <limit>}
])
```

- Trending courses

```
db.courses.aggregate([
 {$match: {'reviews.edited_timestamp': {$gt: <date>}}}
    ,
 {$unwind: {path: '$reviews'}},
 {$group: {_id: {_id: '$_id'},
           size: {$sum: 1}
          }
 },
 {$sort: {size: -1}},
 {$limit: <limit>}
 ])
```

- Best users

```
db.users.aggregate([
 {$unwind: {path: '$reviewed'}},
 {$match: {'reviewed.duration': {$gt: 0},
           'reviewed.price': {$gt: 0}
          }
 },
 {$addFields:
            {quality_ratio: {$divide:
                                ['$reviewed.duration',
                                 '$reviewed.price'
                                ]
                            }
            }
 },
 {$group: {_id: {username: '$username'},
           value: {$avg: 'quality_ratio'}
          }
 },
 {$project: {_id: 0,
             username: '$_id.username',
             value: '$value'
            }
},
{$sort: {value: -1}},
{$skip: <skip>},
{$limit: <limit>}
])
```

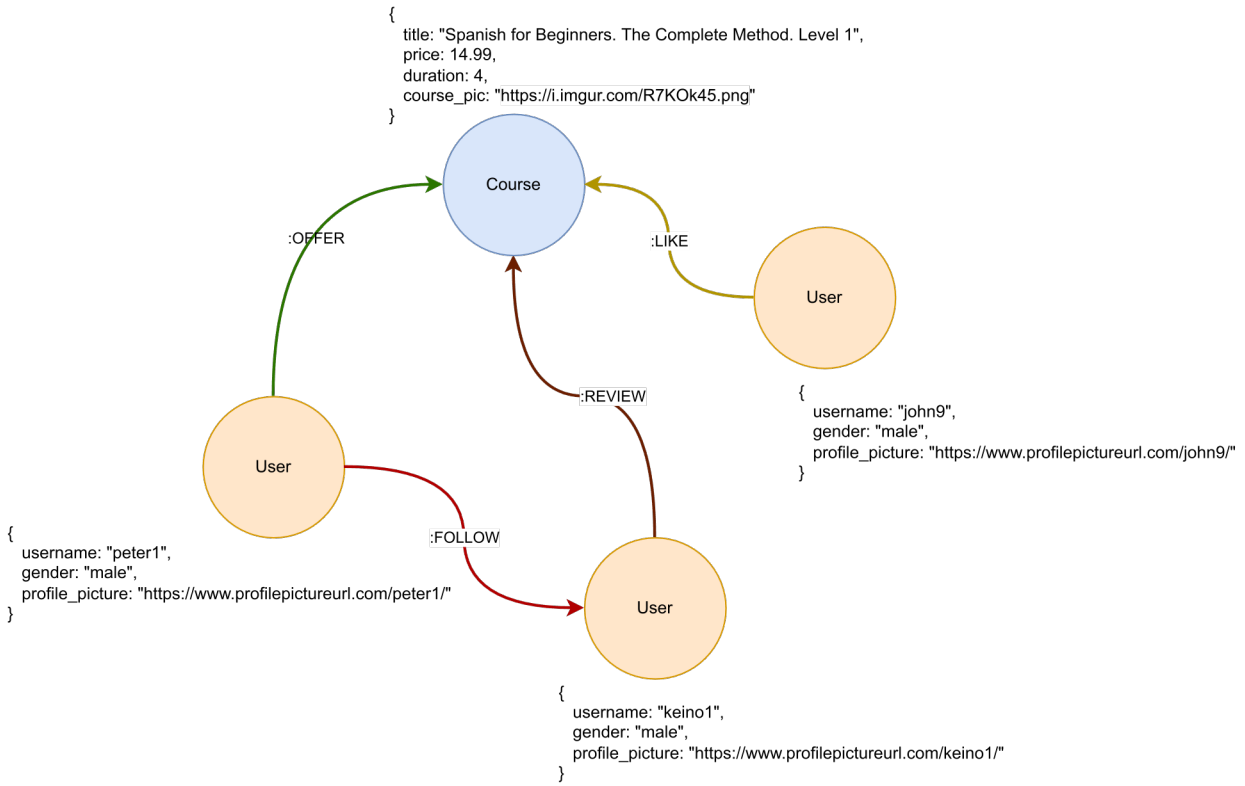
- Compute avg statistic of a user

```
db.users.aggregation([
 {$match: {username: '<username>',
           reviewed: {$exists: true}
          }
 },
 {$unwind: {path: '$reviewed'}},
 {$group: {_id: {username: '$username'},
           count: {$sum: 1},
           avgprice: {$avg: '$reviewed.price'},
           avgduration: {$avg: '$reviewed.duration'}
          }
 },
 {$project: {_id: 0,
             username: '$_id.username',
             count: 1,
             avgprice: 1,
             avgduration: 1
            }
 }
])
```

UNIVERSITÀ DI PISA

Database design Neo4j

Neo4j handle the “social” part of our application because suggest allows to make suggestions based on network relationships between nodes of the network.



Entities:

- User
- Course

Relations:

- User – Course: OFFER, REVIEW, LIKE
- User – User: FOLLOW

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Most relevant queries on Neo4j

- Most followed users

```
MATCH (u:User)<-[f:FOLLOW]-(u2:User)
OPTIONAL MATCH (u)-[l:LIKE]->(:Course)
WHERE u.username <> u2.username
WITH DISTINCT (u), COUNT(DISTINCT(f)) AS followers,
   COUNT(DISTINCT(l)) AS likes
RETURN u
ORDER BY followers DESC, likes DESC
LIMIT <limit>
```

- Find suggested courses

```
MATCH (c:Course)<-[l:LIKE]-(u:User)<-[f:FOLLOW]-(me:
   User{username:<username>})
WHERE NOT EXISTS((me)-[:LIKE]->(c))
WITH DISTINCT(c) as c, COUNT(*) as numUser
RETURN c
ORDER BY numUser DESC
SKIP <skip1> LIMIT <limit1>

UNION

MATCH (me:User{username:<username>})-[l:LIKE]->(c:
   Course)<-[l1:LIKE]-(u:User)<-[:FOLLOW*2..2]-(me)
WHERE NOT EXISTS((me)-[:FOLLOW]->(u)) AND me.username
   <> u.username
WITH DISTINCT(u) as user, COUNT(DISTINCT(c)) as
   numRelationships
WHERE numRelationships > <relationships>
MATCH (user)-[:LIKE|:REVIEW]->(c:Course)
WHERE NOT EXISTS((:User{username:<username>})-[:LIKE|:
   REVIEW]->(c))
RETURN DISTINCT c
ORDER BY CASE c.duration WHEN null THEN 0 ELSE c.
   duration END DESC,
         CASE c.price WHEN null THEN 0 ELSE c.price
            END ASC
SKIP <skip2> LIMIT <limit2>
```

- Most liked courses

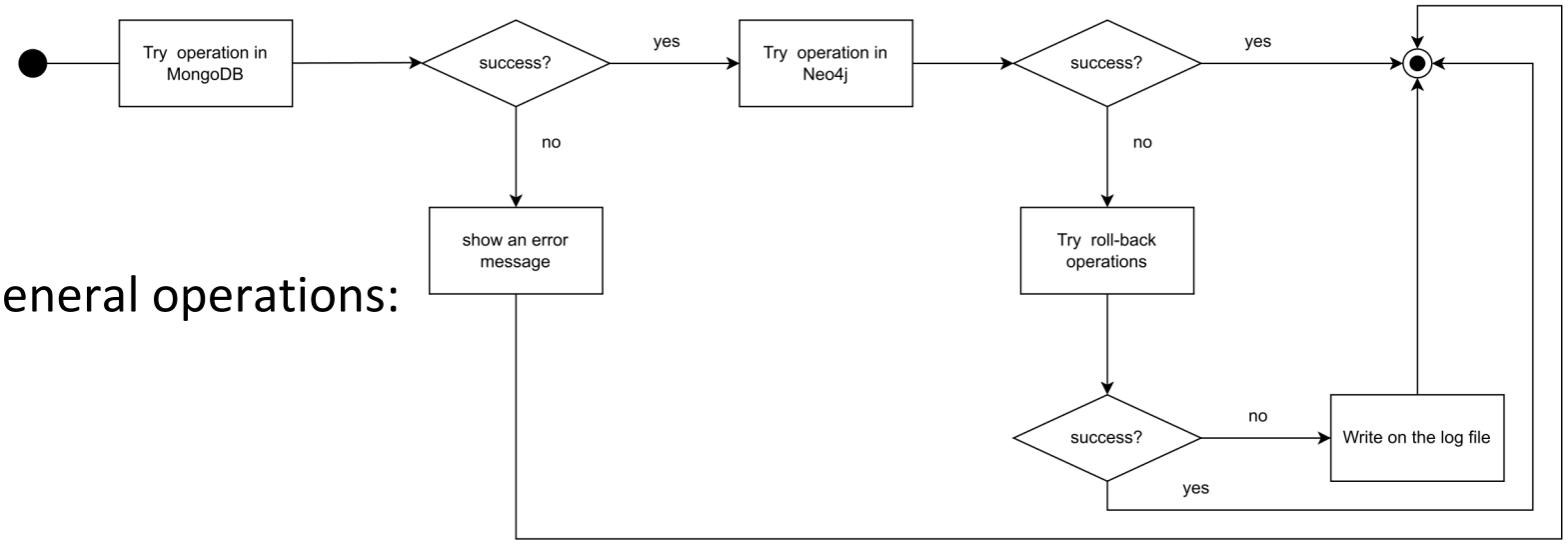
```
MATCH (:User)-[l:LIKE]->(c:Course)
RETURN c,
       COUNT (l) AS like_count,
       CASE c.duration
            WHEN null THEN 0
                ELSE CASE c.price
                         WHEN null THEN 1
                         ELSE c.duration/c.price
                     END
       END AS quality_ratio
ORDER BY like_count DESC, quality_ratio DESC
LIMIT <limit>
```

- Find suggested users

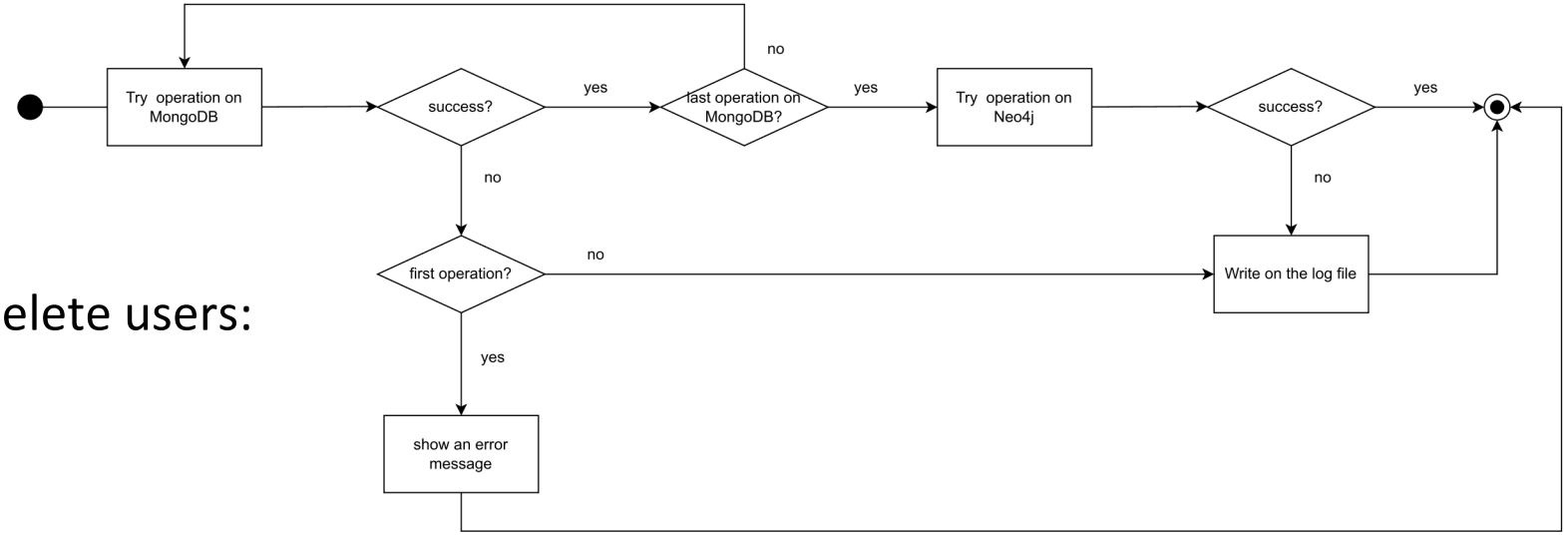
```
MATCH (me:User {username: <username>})-[:FOLLOW*2..2]
   ->(u:User)
WHERE NOT EXISTS((me)-[:FOLLOW]->(u)) AND u.username
   <> me.username
WITH DISTINCT(u) as user, COUNT(u) as followed
WHERE followed > <followed threshold>
RETURN user
ORDER BY followed DESC
SKIP <skip1> LIMIT <limit1>

UNION

MATCH (me:User {username: <username>})-[:LIKE]->(
   commonCourse:Course)<-[:LIKE]-(u:User)
WHERE NOT EXISTS((me)-[:FOLLOW]->(u))
WITH DISTINCT(u) as user, COUNT(commonCourse) as
   numCommonCourses
WHERE numCommonCourses > <common courses threshold>
RETURN  user
ORDER BY numCommonCourses DESC
SKIP <skip2> LIMIT <limit2>
```

Consistency

General operations:



Delete users:



Sharding

For both the collections we select a **shard key**, one for the course collection and the other for the user collection.
As partitioning method we want to use a **consistent hashing** algorithm. It allow us to change nodes in the cluster and remap only keys mapped to the neighbors of the new/removed node.
Sharding keys:

- User collection: “username” that is unique for each user in the collection

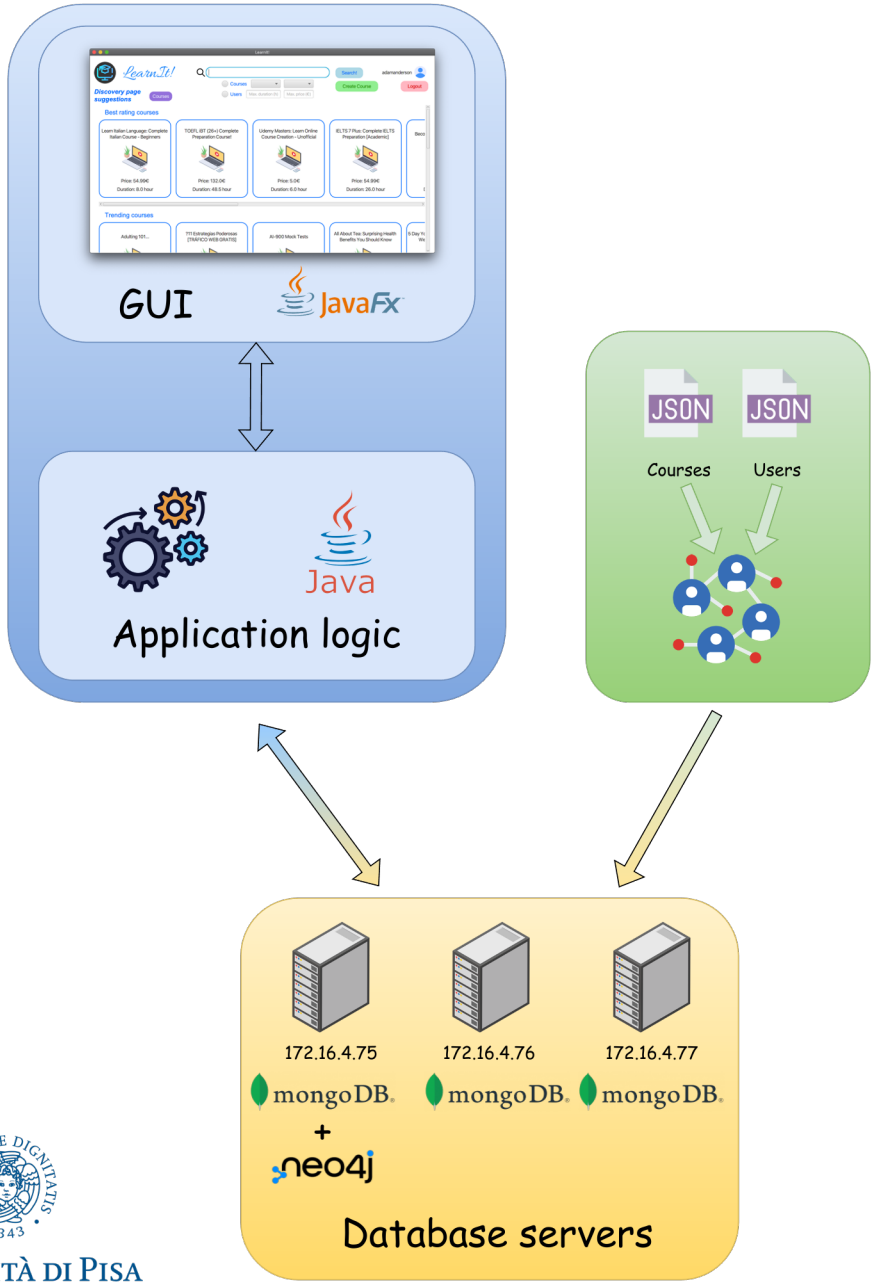
- Course collection: “_id” that is generated by MongoDB and also this one is unique in the collection

Software and hardware architecture

Programming language:
- Java
- JavaFX

DBMSs:
- MongoDB
- Neo4j

Framework:
- Maven



Final considerations

Our application perform mainly read operations (read heavy), so we decided to introduce indexes to speed up frequently queries.
We introduce index regarding user and courses fields that aren't editable after the first time. Thanks to that, there won't be an overhead on updates queries.

LearnIt! application is available on GitHub
https://github.com/federicominniti/LearnIt