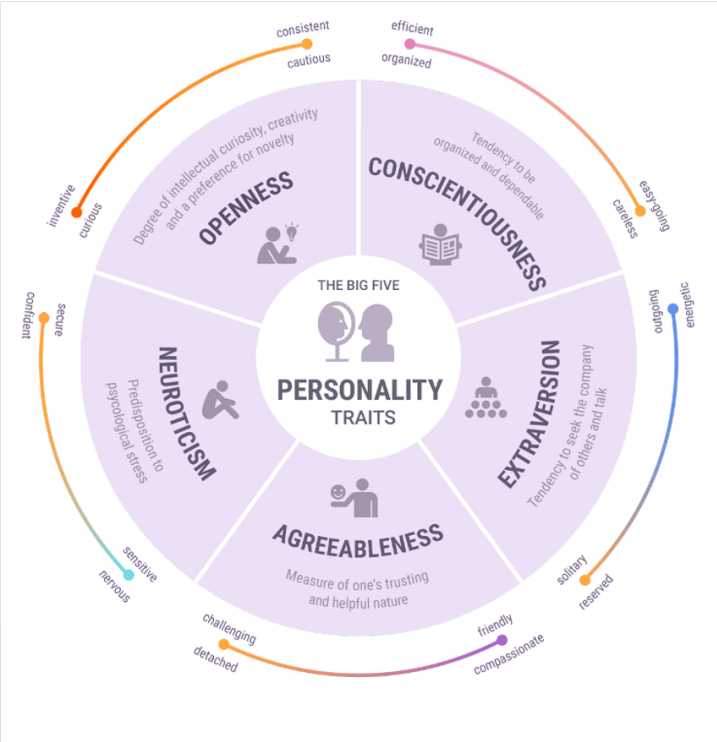
Music & Personality

Project Idea



Many contemporary personality psychologists believe that there are five basic dimensions of personality, often referred to as the "Big 5" personality traits.
The five broad personality traits described by the theory are extraversion, agreeableness, openness, conscientiousness and neuroticism.

Numerous studies have been conducted to show that individual personality can have an effect on music preference.
The relationship between musical preference and personality has remained a long-standing topic of contention for researchers.

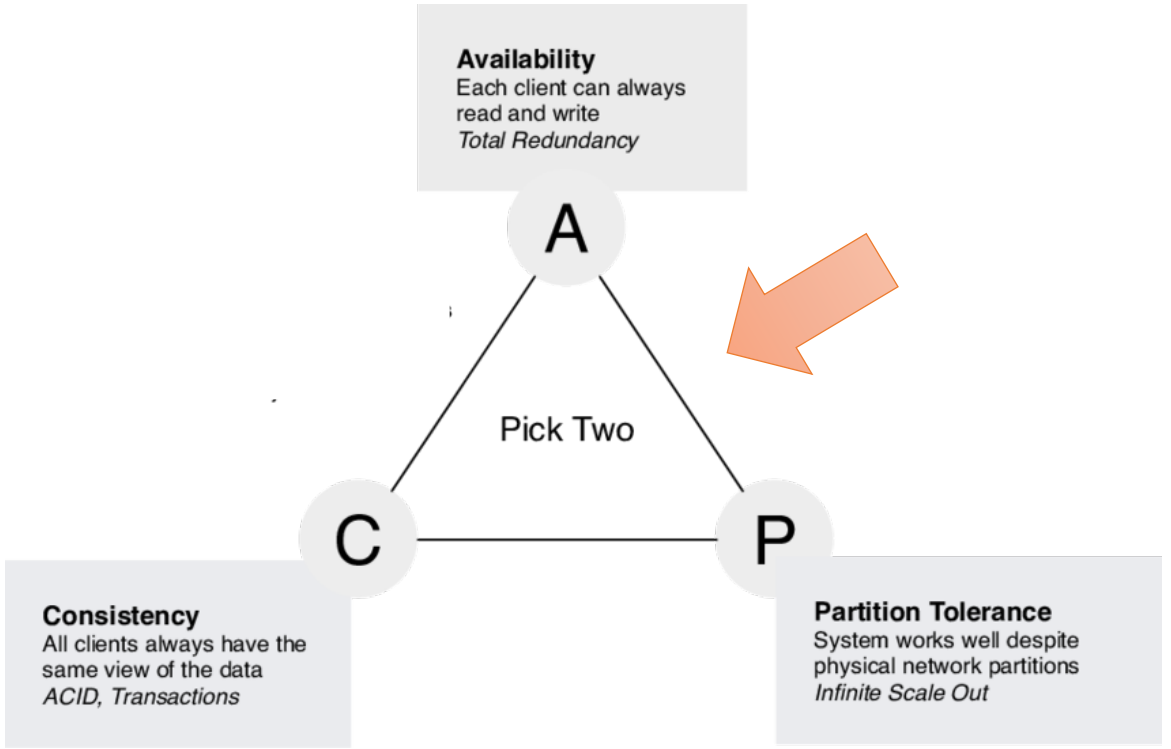
Aim of the project

- Starting from clusters of people with similar personality given by a clustering algorithm (K-Means with K = 5, our analysis can be found at https://github.com/terranovaa/PersonalityClusteringAnalysis), we created an application that allows people with similar personality to know each other and recommend songs most suited for their personality, exploiting songs preferred by similar users.

- Our intelligent recommendation system will adapt its recommendations depending if similar users seems to prefer/not prefer similar songs.
- Our application's aim is also to help the research to determine the link between these two hemispheres.

Non-Functional Requirements

- The application needs to provide **low latency**, **high availability** and **tolerance** to **single points of failures** and **network partitions**, for which we determined that the application has been designed in order to prefer the **Availability** (A) and **Partition Protection** (P) vertices of the CAP triangle.
- **Usability**. The application needs to be user friendly, providing a *GUI.*
- **Privacy**. The application shall provide the security of users' credentials.
- **Scalability**. The system shall work on a non-prefixed number of cluster nodes.
- **Portability**. The application shall be environment independent.
- **Maintainability**. The code shall be readable and easy to maintain.



CAP Thereom

We've guaranteed **availability** and **network partition protection** expoiting replicas in our cluster of Servers. Therefore, we decided to choose:

- **Write Concern Type**: W1, waiting for an acknowledgment from a single member. It allows the system to be as fast as possible during write operations.
- **Write Timeout**: 0, because at least one write operation should be performed.
- **Read Concern**: Nearest, in such a way that read operations are performed on the nearest node considering responsiveness, measured in pings.

Our choices introduce the need to implement some kind of **eventual consistency paradigm**, which we guaranteed with the use of a log file, that will be periodically checked by the administrator.

Dataset

The dataset we used for the users comes from Kaggle (https://www.kaggle.com/tunguz/big-five-personality-test) and was collected through an interactive on-line personality test.

In the dataset, 10 questions are available for each personality trait, with answers on a five point scale, labeled 1 = Disagree, 3 = Neutral and 5 = Agree.

The corresponding time needed to answer each question is also present in ms.

Extroversion			Neurotic			Agreeableness			Openness			Conscientiousness			Time	
10			10			10			10			10			50	
EXT1	**…**	**EXT10**	**EST1**	**…**	**EST10**	**AGR1**	**…**	**AGR10**	**OPN1**	**…**	**OPN10**	**CSN1**	**…**	**CSN10**	**EXT1_E**	**…**
2		4	5		1	3		5	4		5	1		5	3400 ms	
4		5	1		2	2		4	5		3	2		3	1900 ms	

1015341 x 100

416.27 MB
(***Volume***)

Dataset (2)

The dataset we used for the songs comes from Kaggle (https://www.kaggle.com/rodolfofigueroa/spotify-12m-songs) and contains the audio features of 1.2M+ Spotify Songs obtained with the Spotify API. Among the information for each song, we have: Name, Album, Artists, Danceability and other interesting music features.

Name	Album	Artists	Danceability	Energy	Loudness	Speechiness	…
Guerrilla Radio	The Battle Of Los Angeles	['Rage Against The Machine']	0.599	0.957000000000001	-5.763999999999999	0.188	…
…	…	…	…	…	…	…	…

1200000 x 24
345.74 MB
(***Volume***)

Variety is given by the use of two different datasets.

Heuristic for the initial interaction between the datasets

Since we've used two different datasets which do not interact with each other and since we need starting preferences to make our application work, we decided to use an heuristic to allocate the first preferences of the users, exploiting a paper studying the correlation among the big 5 personality traits and music streaming behavior of Spotify users: https://dl.acm.org/doi/pdf/10.1145/3468784.3469854.

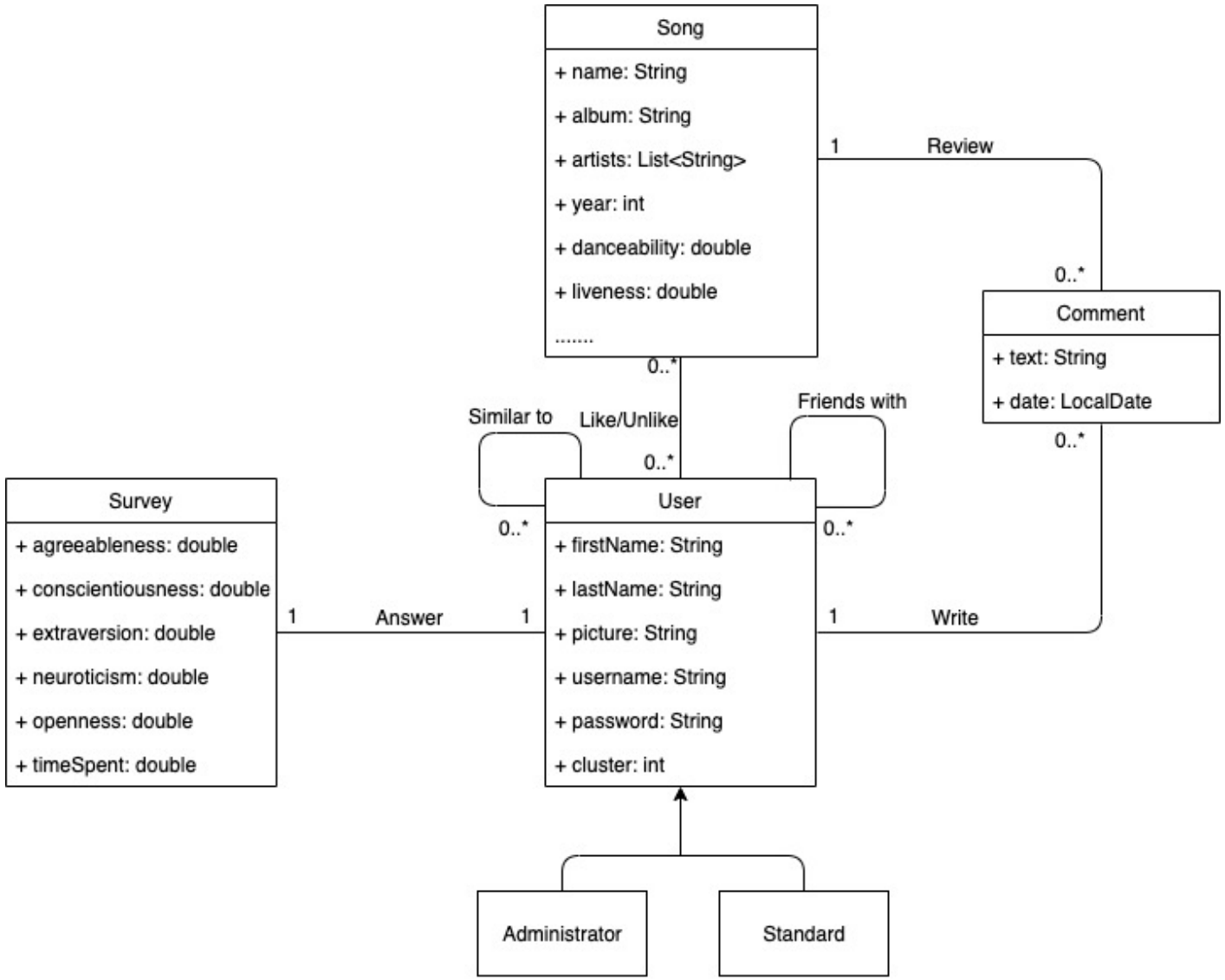
Table 2: Results of the Correlation Analysis.

	Openness to Experience	Conscientiousness	Extraversion	Agreeableness	Neuroticism
Acousticness	.076	.112	**.405***	.304	-.128
Danceability	-.056	.240	.102	-.006	-.244
Energy	-.215	-.230	**-.475****	-.287	.065
Instrumentalness	.002	-.253	.140	-.028	**.479****
Liveness	**.376***	.198	-.217	.096	.004
Speechiness	.120	.013	**-.456****	-.182	-.015
Valence	.022	-.029	-.122	.020	-.092
Tempo	-.190	-.192	-.329	**-.452****	.070

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

UML Analysis Class Diagram



Graph DB

We used Neo4j to store information about friendships and similarity among users, and preferences of users on the songs.

Nodes:

Node Properties

User

<id>	11
country	France
firstName	Martin
lastName	Muñoz
mongoId	61e91a0713c1e6256f5bf3b4
picture	https://randomuser.me/api/portraits/men/1.jpg

User Node Properties Example

Node Properties

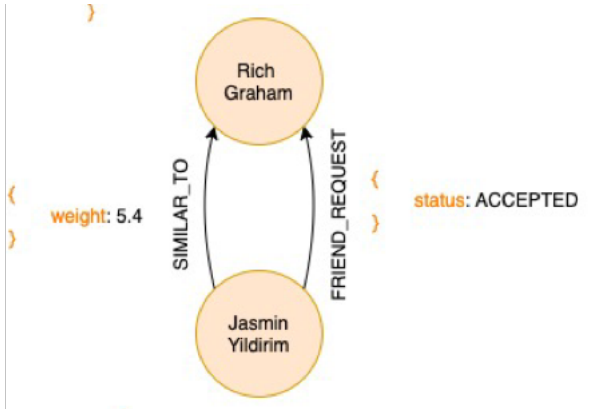
Song

<id>	842536
album	Rage Against The Machine
artists	Rage Against The Machine
mongoId	61e87a6513c1e6256f49947f
name	Fistful of Steel

Song Node Properties Example

Relations:

- SIMILAR_TO (User -> User)
  - weight ∈ [0,1] $w_i = \frac{1}{dist^2(x,y)}$ $dist(x,y) = \sqrt{\sum_{i=1}^{6}(personality_x - personality_y)^2}$
- FRIEND_REQUEST (User -> User)
  - status ∈ {ACCEPTED, REFUSED, UNKNOWN}



- LIKES (User -> Song)
  - value ∈ $\{-1, +1\}$

Recommended Songs

$$Strength_{Userx \rightarrow Song} = \sum_{paths:Userx \rightarrow Usery \rightarrow Song} (f(\alpha) * Weight + \beta * Coherence) * PreferenceValuey$$

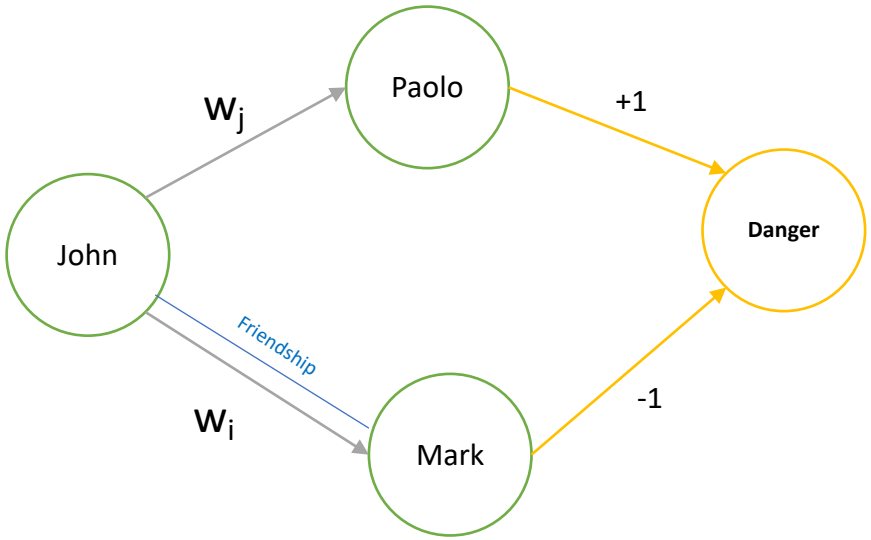
$\alpha \in [0,1]$ $\beta \in [0,1]$

Clustering Algorithm & Survey

1) $Weight = \frac{1}{dist^2(x,y)}$

$$dist(x,y) = \sqrt{\sum_{i=1}^{6} (personality_x - personalityy)^2}$$

$$f(\alpha) = \begin{cases} 2 & if\ x\ and\ y\ are\ friends \\ 1 & otherwise \end{cases}$$



John->Danger = w_i*(+1) + 2*w_j*(-1)

Recommended Songs

$$Strength_{Userx \rightarrow Song} = \sum_{paths:Userx \rightarrow Usery \rightarrow Song} (f(\alpha) * Weight + \beta * Coherence) * PreferenceValuey$$

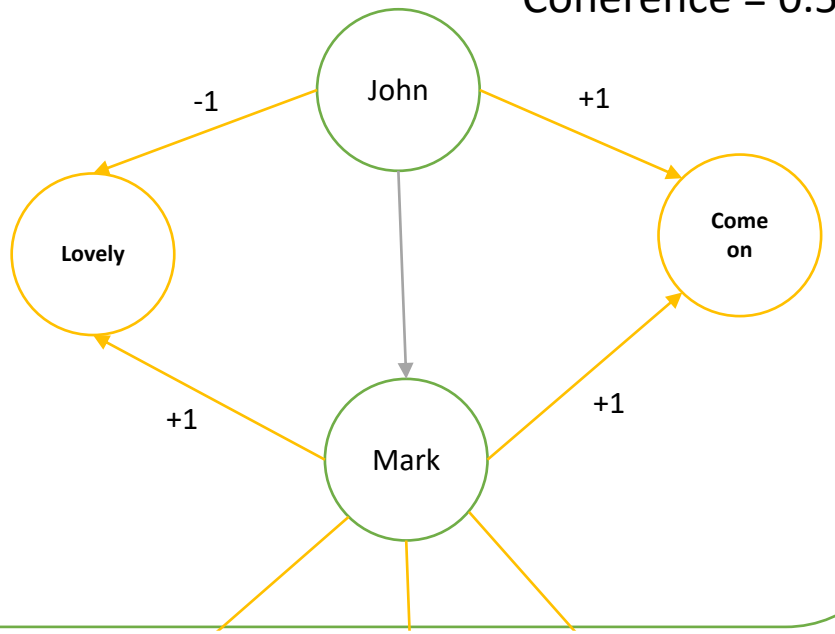
$\alpha \in [0,1]$ $\beta \in [0,1]$

Use of the application

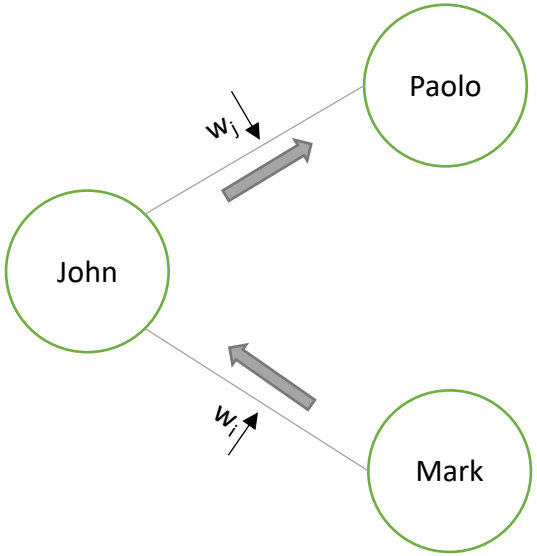
2) $Coherence = \frac{\#CoherentPreferences}{\min(\#PreferencesX,\ \#PreferencesY)}$

$\#CoherentPreferences =$
$\#edges\ towards\ the\ same\ song\ with\ same\ preference$

Coherence = 0.5



Recommended Songs




```
MATCH (u:User)-[r:SIMILAR_TO]->(su:User)-[p:LIKES]->(s:Song)
WHERE NOT (u)-[:LIKES]->(s) AND u.mongoId = '$mongoId'
OPTIONAL MATCH (u)-[f:FRIEND_REQUEST {status: "accepted"}]->(su)
WITH s AS Song, u AS User, su as SimilarUser, p as Preference,
    CASE when count(f)>0 then r.weight * 2 else r.weight end AS Weight
OPTIONAL MATCH (u:User)-[p:LIKES]->(s:Song)<-[p2:LIKES]-(su:User)
WHERE su.mongoId <> u.mongoId AND p.value = p2.value AND u = User AND su = SimilarUser
WITH User, Song, SimilarUser, count(p) AS Coherence, Weight, Preference
OPTIONAL MATCH (u:User)-[pr:LIKES]->(s:Song)
WHERE u = User
WITH count(pr) AS numLikes1, Song, SimilarUser, User, Coherence, Weight, Preference
OPTIONAL MATCH (u:User)-[pr:LIKES]->(s:Song)
WHERE u = SimilarUser
WITH count(pr) AS numLikes2, numLikes1, Song, SimilarUser, User, Coherence, Weight, Preference
UNWIND [numLikes1,numLikes2] AS numLikes
WITH User, SimilarUser, Song, Preference, CASE when min(numLikes) <> 0 then Coherence/(toFloat(min(numLikes))) else 0 end AS BetaStrength, Weight
WITH Song, sum(0.8 * Weight * Preference.value + 0.2 * BetaStrength * Preference.value) AS Strength
RETURN Song
ORDER BY Strength DESC
LIMIT 50
```

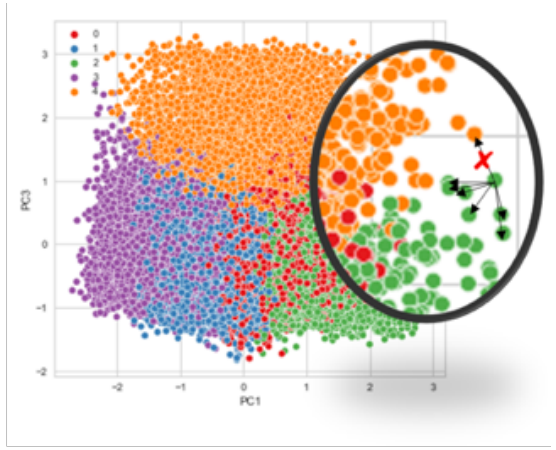
Cypher Query

application.properties

```
recommended.songs.alpha = 0.8
recommended.songs.beta = 0.2
```

Application Properties File

Recommended Users



$$w_i = \frac{1}{dist^2(x,y)}$$

```
MATCH (a:User)-[f:FRIEND_REQUEST]-(b:User)
WHERE a.mongoId = '$mongo_id'
AND f.status <> 'UNKNOWN'
WITH collect(b) AS excluded
MATCH (a:User {mongoId:$mongo_id })-[r:SIMILAR_TO]-(b:User)
OPTIONAL MATCH (a)-[outgoing:FRIEND_REQUEST]->(b)
OPTIONAL MATCH (a)<-[incoming:FRIEND_REQUEST]-(b)
WITH excluded, r, outgoing, incoming, collect(b) AS users
WHERE NONE(b IN users WHERE b IN excluded)
RETURN users, outgoing, incoming
ORDER BY r.weight DESC
```

Cypher Query

Top Cluster with the highest number of Coherent similar users

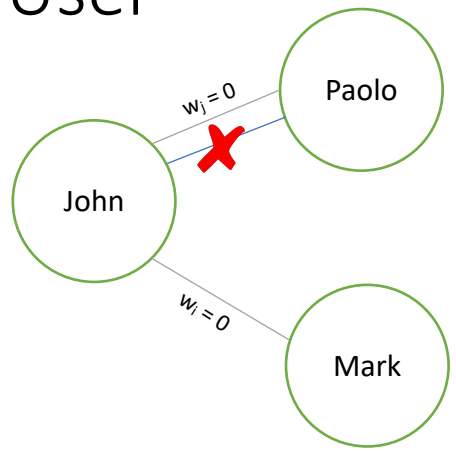
```
MATCH (u:User)-[p:LIKES]->(s:Song)<-[p2:LIKES]-(su:User)
WHERE su.mongoId <> u.mongoId AND p.value = p2.value AND EXISTS ((u)-[:SIMILAR_TO]-(su))
WITH u AS user, su AS similarUser, count(*) AS coherence
MATCH (u:User)-[pr:LIKES]->(s:Song)
WHERE u = user
WITH count(*) AS numLikes1, similarUser, user, coherence
MATCH (u:User)-[pr:LIKES]->(s:Song)
WHERE u = similarUser
WITH count(*) AS numLikes2, numLikes1, similarUser, user, coherence
UNWIND [numLikes1,numLikes2] AS numLikes
WITH  user, similarUser, coherence/(toFloat(min(numLikes))) AS Strength
WHERE Strength > 0.75
RETURN user.cluster, count(*) AS NumUsers
ORDER BY NumUsers DESC
LIMIT 1
```

Cypher Query

Quarantine User

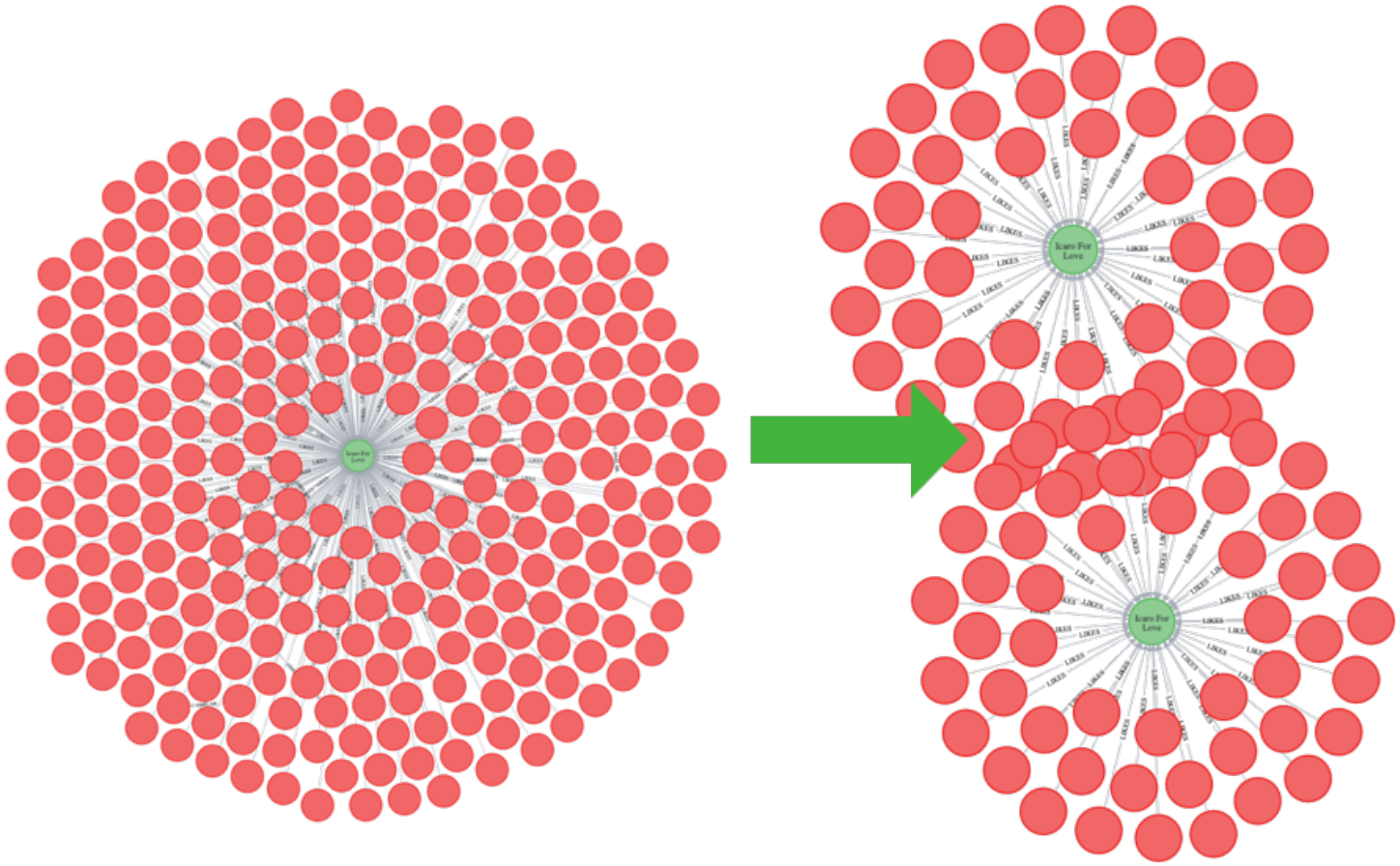
```
MATCH (u:User)-[f:FRIEND_REQUEST]-(:User)
WHERE u.mongoId = $mongo_id
DELETE f

MATCH (u:User)-[r:SIMILAR_TO]-(:User)
WHERE u.mongoId = $mongo_id
SET r.weight = 0
```

Cypher Query



Supernodes

Super-nodes may considerably slow down graph traversal and slow down our read and write operations.
Let’s think for example at songs which are liked by everyone, so with a huge amount of relations.



Example of splitting

```
/* Determine Supernodes */
MATCH (s:Song)<-[l:LIKES]-(u:User)
WITH s AS Song, count(*) AS NumLikes
WHERE NumLikes > 'Threshold'
RETURN Song

/* Determine Top 2 Clusters with the highest number of relationships */
MATCH (s:Song)<-[r:LIKES]-(u:User)
WHERE s.mongoId = $mongoId
RETURN u.cluster AS Cluster, count(*) AS Count
ORDER BY count DESC

/* Add cluster field to the supernode */
MATCH (s:Song{mongoId : $mongoId})
UPDATE SET s.cluster = $firstId

/* Create duplicate */
MATCH (s:Song{mongoId : $mongoId})
CREATE (s2:Song{mongoId : $mongoId, songName: $songName, artists: $artists, album: $album, cluster: $secondId})

/* Move relationships regarding the second highest cluster */
MATCH (s:Song{mongoId : $mongoId, cluster: $firstId})<-[r:LIKES]-(u:User {cluster: $secondId})
MATCH (s2:Song{mongoId: $mongoId, cluster: $secondId})
CREATE (u)-[rNew :LIKES]->(s2)
SET rNew = r
DELETE r

/* Distribute the remaining relationships regarding the other clusters */
/* Example 5 clusters, the 2nd and the 4th clusters goes to the duplicated Supernode */
MATCH (s:Song{mongoId : $mongoId, cluster: $firstId})<-[r:LIKES]-(u:User {cluster: $fourthId})
MATCH (s2:Song{mongoId: $mongoId, cluster: $fourthId})
CREATE (u)-[rNew :LIKES]->(s2)
SET rNew = r
DELETE r
```

Splitting Supernode Cypher Statements

Document DB

We decided to use a Document Database, with MongoDB as DBMS,

to store all main information about the entities of our application.

Population of the database

EXT1	……	OPN10_E
2	…..	6448
4	…..	2331
….	….	2564
5	…..	4623

For each instance a user was generated from https://randomuser.me


```
_id: ObjectId("61e91bc613c1e6256f65bf79")
first_name: "Blake"
last_name: "Sirko"
date_of_birth: 1977-06-18T22:00:00.000+00:00
gender: "male"
country: "Albania"
username: "blackbutterfly1630"
phone: "480-613-8037"
email: "blake.sirko2@example.com"
password: "$2a$05$pFKKJhMFsumUlQOIaPQydumjMHaOAC3i/cUTg6seGmIHgKQquHFua"
registration_date: 2004-01-17T23:00:00.000+00:00
picture: "https://randomuser.me/api/portraits/men/91.jpg"
extraversion: 2.4
agreeableness: 2.1
conscientiousness: 3.3
neuroticism: 3.4
openness: 3.1
time_spent: 3.86
cluster: 1
```


Survey

Clustering Algorithm

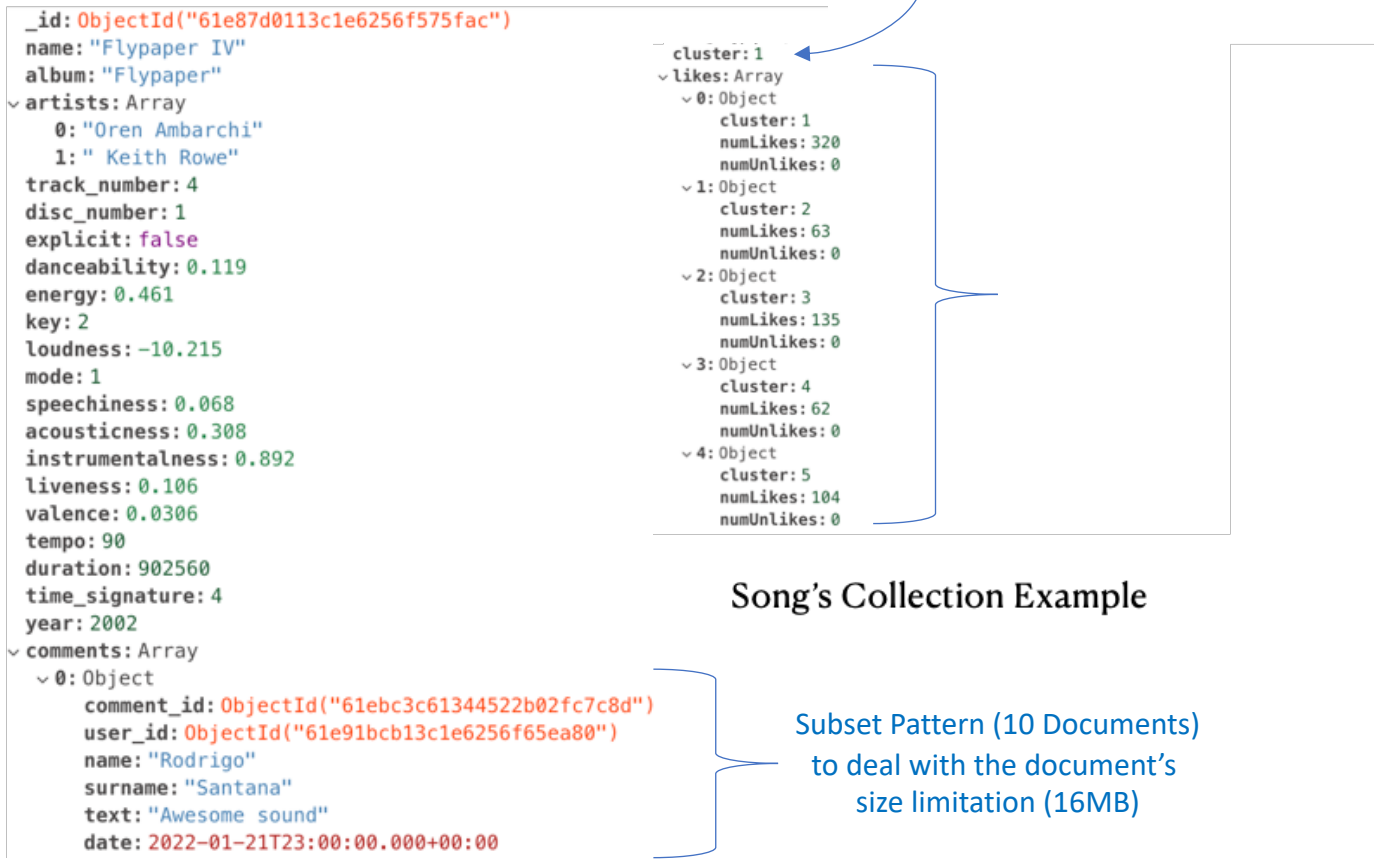
User's Collection Example

Document DB

Song's Collection:

Predominant Cluster

$$max \sum (likes - unlikes)$$



Song's Collection Example

Subset Pattern (10 Documents) to deal with the document's size limitation (16MB)

Comment's Collection:

Comment's Collection Example

Admin Analytics

Cluster with the highest variation

```
{$group: {
    _id: '$cluster',
    AGRMax: { $max: '$agreeableness' },
    AGRMin: { $min: '$agreeableness' },
   OPNMax: { $max: '$openness' },
    OPNMin: { $min: '$openness' },
    CSNMax: { $max: '$conscientiousness' },
    CSNMin: { $min: '$conscientiousness' },
    EXTMax: { $max: '$extraversion' },
    EXTMin: { $min: '$extraversion' },
    ESTMax: { $max: '$neuroticism' },
    ESTMin: { $min: '$neuroticism' },
    TimeSpentMax: { $max: '$time_spent'},
    TimeSpentMin: { $min: '$time_spent'}}},
  {$project: {
    differenceAGR: { $subtract: ['$AGRMax','$AGRMin']},
    differenceOPN: { $subtract: ['$OPNMax','$OPNMin']},
    differenceCSN: { $subtract: ['$CSNMax','$CSNMin']},
    differenceEXT: { $subtract: ['$EXTMax','$EXTMin']},
    differenceEST: { $subtract: ['$ESTMax','$ESTMin']},
    differenceTS: { $subtract: ['$TimeSpentMax','$TimeSpentMin']}
  }},
  {$project: {
    difference: {
     $sum: [
      '$differenceAGR',
      '$differenceCSN',
      '$differenceEXT',
      '$differenceEST',
      '$differenceOPN',
      '$differenceTS'
     ]
    }
  }},
  {$sort: {difference: -1}},
  {$limit: 1}]
```

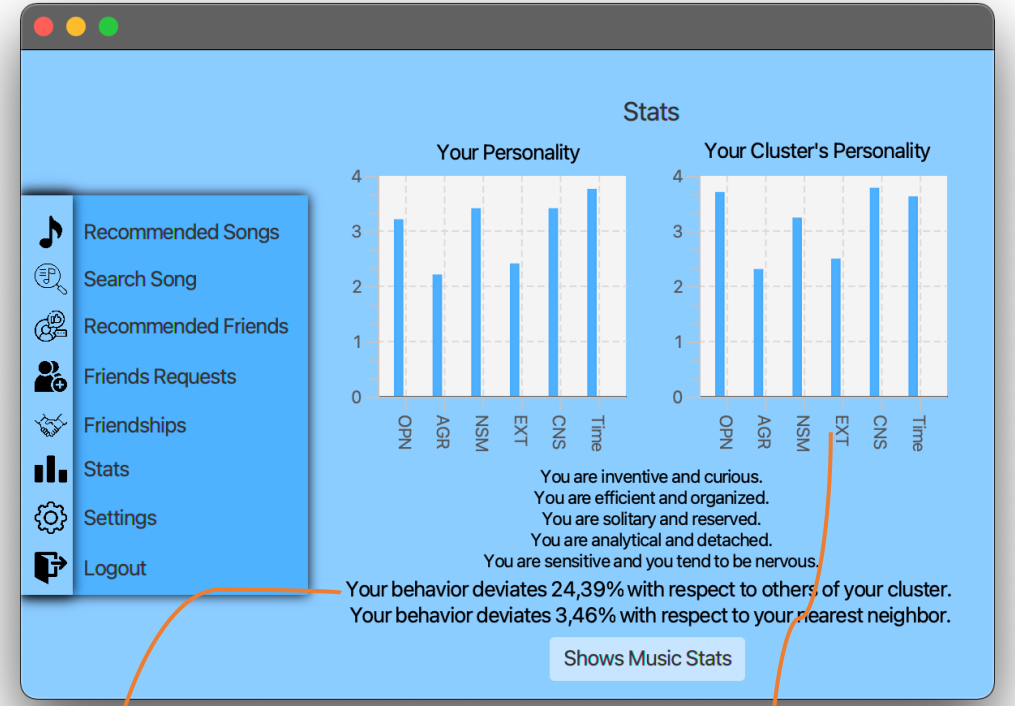
Top K Countries with the highest average of personality traits

```
[{$group: { _id: '$country',
 AGR: { $avg: '$agreeableness' },
 OPN: { $avg: '$openness'},
 CSN: { $avg: '$conscientiousness' },
 EXT: { $avg: '$extraversion' },
 EST: { $avg: '$neuroticism' }
}},
{$project: { cluster: 1, avg: { $avg: [ '$AGR', '$EST', '$EST', '$CSN', '$OPN' ]}}},
{$sort: {avg: -1}},
{$limit: K},
{$project: {cluster: 1}}]
```

Most Danceable Cluster

```
[{$sort: {danceability: -1}},
 {$limit: 500},
 {$group: {
 _id: '$cluster',
 strength: {$sum: 1}
}},
{$sort: {strength: -1}},
 {$limit: 1},
 {$project: {
 _id: 1
}}]
```

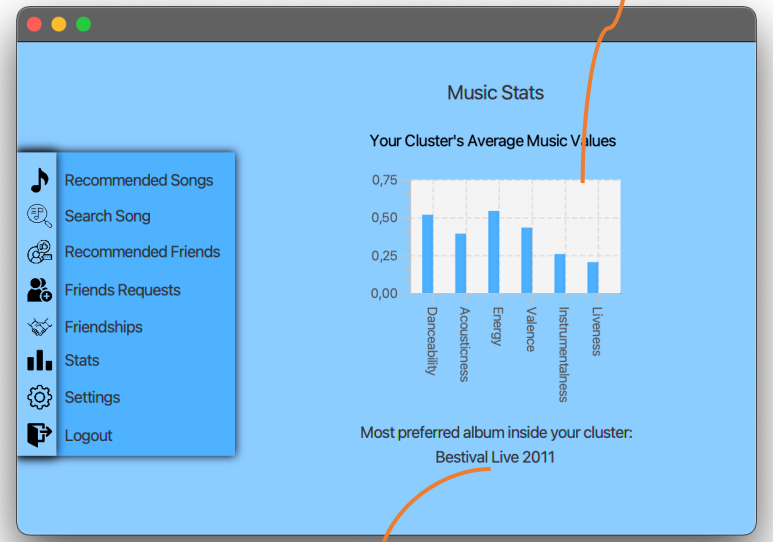
Analytics



$$dist(x,y) = \sqrt{\sum_{i=1}^{6}(personality_i - AvgClusterPersonalityi)^2}$$

```
[{
$group: { _id: "$cluster",
  AGR: { "$avg": "$agreeableness" },
  OPN: { "$avg": "$openness"},
  CSN: { "$avg": "$conscientiousness"},
  EXT: { "$avg": "$extraversion"},
  EST: { "$avg": "$neuroticism"},
  TimeSpent: { "$avg": "$time_spent"}
}]
```

```
[{
  $group: { _id: '$cluster',
      danceability: { $avg: '$danceability'},
      acousticness: { $avg: '$acousticness' },
      energy: { $avg: '$energy' },
      instrumentalness: { $avg: '$instrumentalness' },
      liveness: { $avg: '$liveness' },
      valence: { $avg: '$valence' }
  }
}]
```




```
[{$match: { cluster: $id}},
{$unwind: { path: '$likes'}},
{$project: {
 album: 1,
 likes: 1,
 result: {
  $eq: ['$cluster','$likes.cluster']
 }
}},
{$match: {result: true}},
{$project: {
 album: 1,
 algebricSum: {
  $subtract: ['$likes.numLikes', '$likes.numUnlikes']}
}},
{$group: {
 _id: '$album',
 strength: {$sum: '$algebricSum'}
}},
{$sort: {strength: -1}},
{$limit: K}]
```

Sharding

We've chosen the most appropriate shard key considering:
- The distribution of reads and writes.
- The size of the chunks.
- The number of chunks each query uses.

Users:

1. {_id} ->
   - always be adding documents to the last replica set (high write-lock-% ending in slow write operations)
   - users who subscribed recently to the application are typically the ones who uses it the most, we would always involve the last replica set for reading operations.
   - analytics would involve mutiple servers
2. {cluster} ->
   - personality clusters are quite balanced in terms of number of users
   - analytics of users involves only their cluster, embracing data locality
   - if we have too many documents with the same shard key we may end up with jumbo chunks.
3. {cluster, _id} ->
   - the addition of the _id field will break large clusters into multiple chunks and solve the problem of Jumbo chunks. Users of the same cluster will still be adjacent in index, and so most likely to end up on the same shard.

Sharding

Songs:

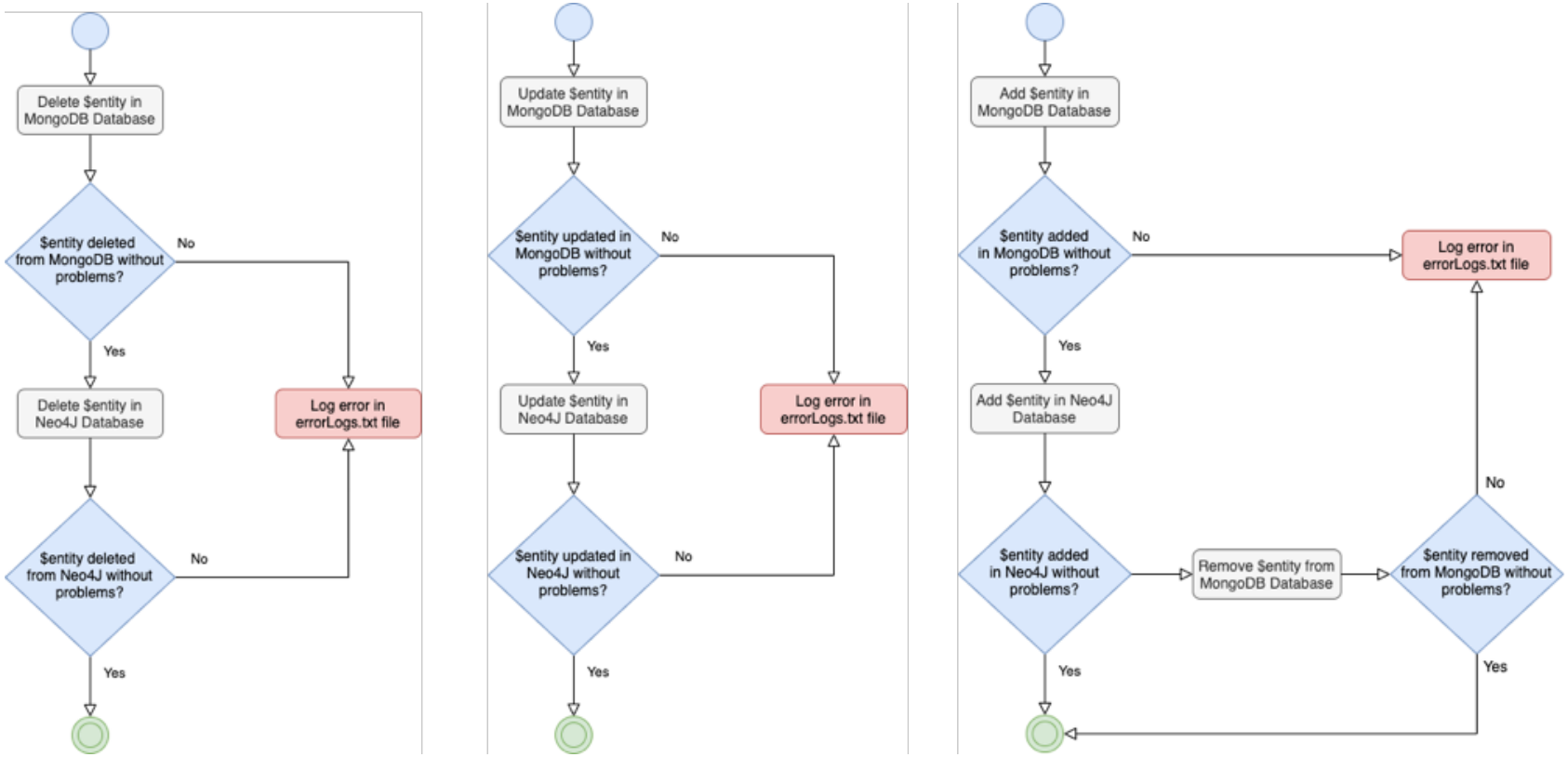
Exploiting the cluster redundant field, that represents the predominant cluster in a song, we can think about having clusters of songs too, where a song will be associated to the cluster from which it has the highest sum of likes – unlikes.
We decided, though, to use the same combination of fields {cluster, _id} for the songs as well.

{cluster, _id} ->

- the addition of the _id field will break large clusters into multiple chunks, which will still be adjacent in index, and so most likely to end up on the same shard.
- in the available analytics, a user can only ask for statistics involving songs in which its cluster is predominant, involving almost always documents present in just one shard.

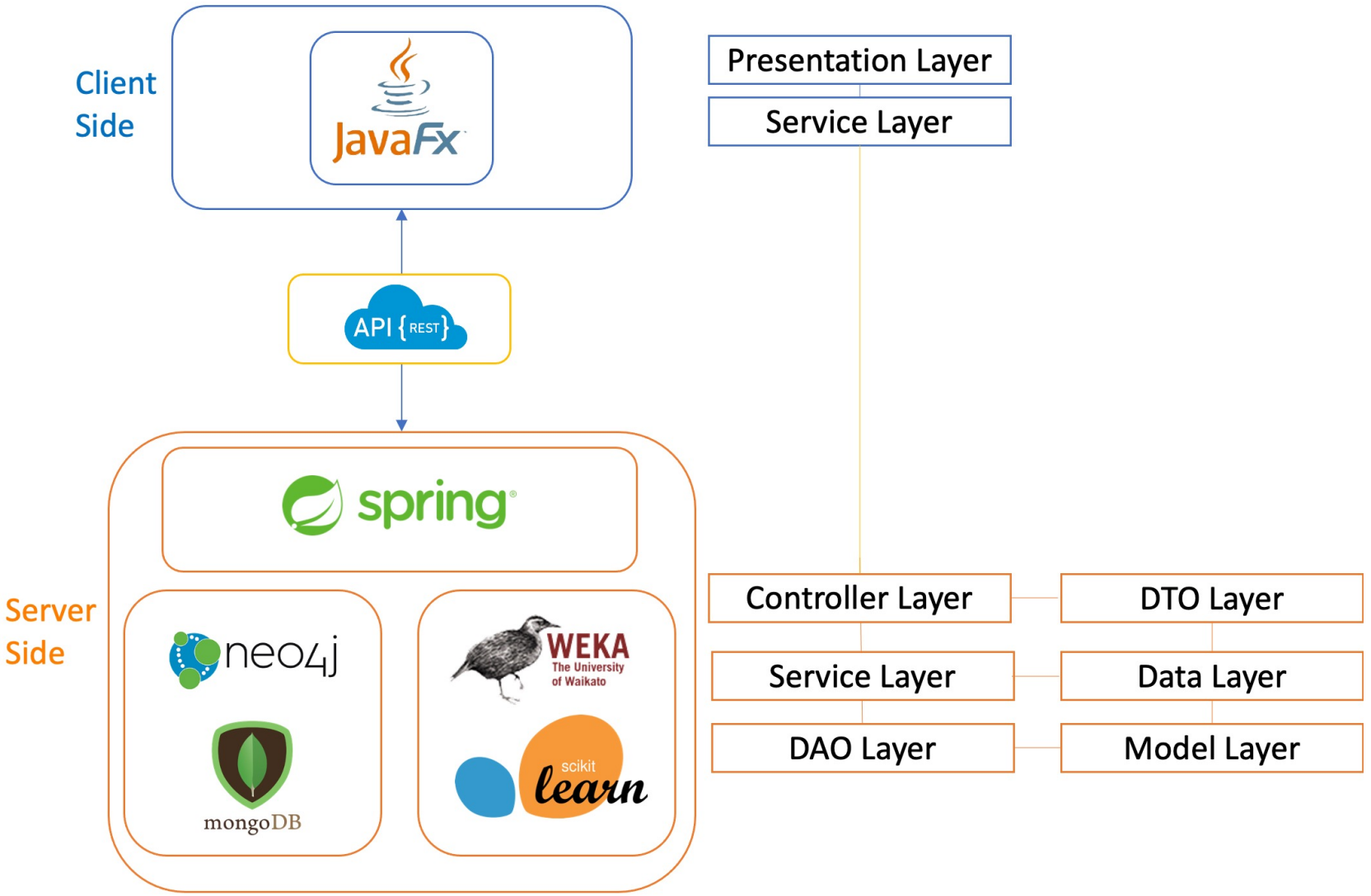
Cross-Database Consistency Management

The operations that require a cross-database consistency management are the insertion, removal and updating of a song, a user or a preference.



Administrators will manually check the errorLogs.txt file and update/add/remove entities or increment/decrement counters to restore the consistency.

Software and Hardware Architecture



Final Considerations

Our recommendation system adapts to the user's tastes, not only considering the results of the survey and the clustering algorithm, but also the preferences given during the use of the application.
Using this adapting behavior, our application will also be able to help the research to determine the link between these two hemispheres.