Large-Scale and Multi-Structured Databases

Project Documentation
GameCritic

Filippo Lucchesi
Ettore Ricci
Martina Speciale

University of Pisa
Department of Information Engineering

Academic Year 2023/2024

Contents

Chapter 1

Introduction

- Allow users to retrieve various informations on videogames
- Allow users to review videogames and give them a score
- Allow users to follow other users, like and comment other users' reviews
- Give users recommendationson on who to follow and what games to play
- Allow videogame company managers to retrieve analytics on the sentiment of the reviewers for a particular game
- Allor the application manager to see which users are the most active on the site, so that he can promote them as the best reviewers

Chapter 2

Dataset and Web Scraping

- **Variety** : we exploited two real different sources, having different formats
- **Velocity/Variability** :
  - Comments are frequently added/liked/responded to
  - Most active users change periodically

2.1 Raw Data

Data was retrieved from:

- MobyGames
- MetaCritic
- generated data

2.2 Scraping Algorithms

We used *Python* for scraping and data generation

2.2.1 Videogames Scraping

Data is scraped from *MobyGames* for videogames info

2.2.2 Users and Review Scraping

Some data is scraped from *Metacritic* for reviews, while other reviews are generated. Comment on reviews are generated using algorithms

2.3 Resulting Dataset

The final volume wanders around 400 MB, which are the result of:

- 70k videogames
- 250k reviews
- 625k comments

Chapter 3

Design

3.1 Actors

There are three main actors:

- Reviewer (User)
- Company Manager (Super User)
- Administrator The *Reviewer* is the end-user of the application, who is able to search for *videogames*, review them, comment on other users' reviews, etc The *Company Manager* is an entity that represents the owner

3.2 Requirements

3.2.1 Functional Requirements

Handled by MongoDB

- Average review score per game producer
- Most active Users (looking at the number of posted reviews)
- Most discussed reviews
- Most popular games in the latest week/month/year

Handled by Neo4j

- New friend recommendations
- Videogame recommendations

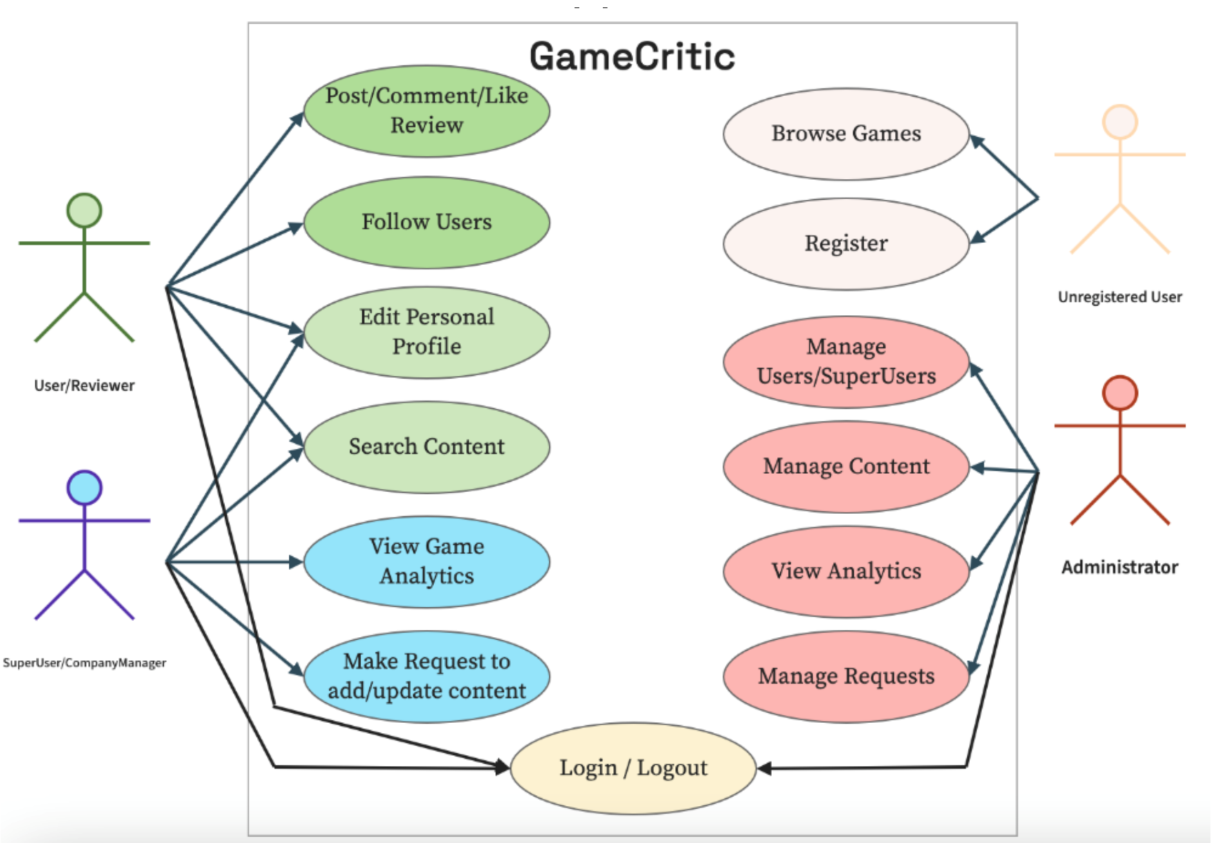

Figure 3.1: Actors and main supported functionalities

- identify the most influential users

3.2.2 Non-Functional Requirements

3.3 Use Case Diagram

3.4 UML Class Diagram

3.5 Data Models and Implementation

3.5.1 Document DB Collections

- Videogames
- Users
- Reviews
- Comments

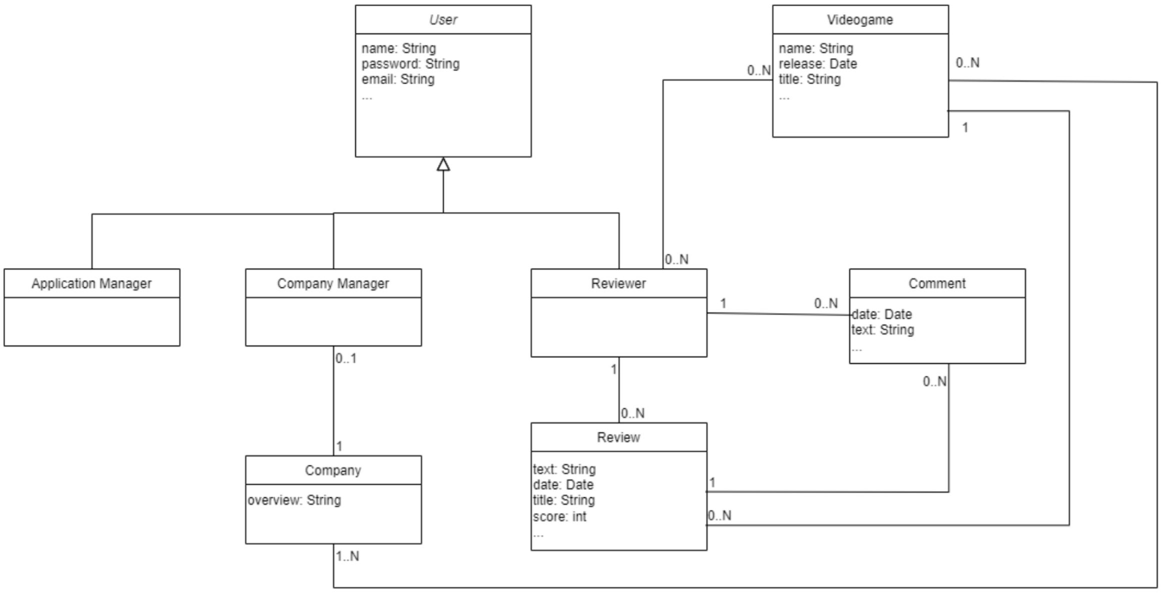

Figure 3.2: UML Class Diagram

- userImages

3.5.2 Document DB Relationships

3.5.3 Document DB Examples

```
  {
"Name": "(Almost) Total Mayhem",
"Released": "January 14th, 2011 on Xbox 360",
"Publishers": "Peanut Gallery",
"Developers": "Peanut Gallery",
"Genre": "Action",
"Perspective": "Side view",
"Gameplay": "Platform",
"Setting": "Fantasy",
"Media Type": "Download",
"Multiplayer Options": "Same/Split-Screen",
"Number of Offline Players": "1-2 Players",
"Description": "description",
"user_review": 6.0,
"reviews": [
{
  "score": 8,
  "quote": "stunning graphhcs!",
```

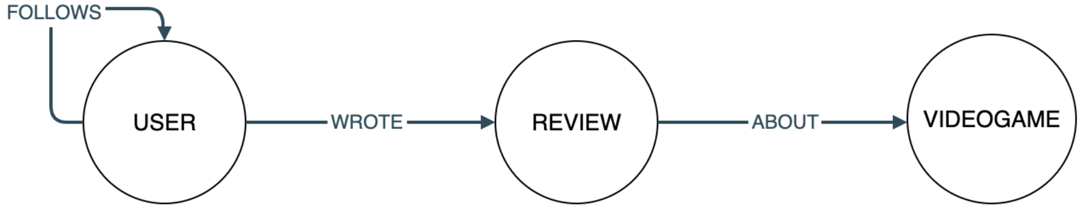

Figure 3.3: Entities handled by *GraphDB* and their relationships

```
    "author": "Yusoreqa",
    "date": "2023-10-30",
    "source": "random",
    "comments": [
    {
      "author": "Hojosu",
      "quote": "I see that Yusoreqa agrees with me.",
      "date": "2023-12-06"
    }
    ]
  }
  ]
}
```

3.5.4 Graph DB - Neo4j

We handled the following entities via GraphDB (figure **??**) :

- Users
- Videogames
- Reviews

3.6 Distributed Database Design

Eventual consistency is sufficient, enforcing strong consistency would be too costly.

3.6.1 Replicas

We were given the chance to exploit a cluster of three nodes for the project. We deployed one *MongoDB replica* for each node (replicas were not implemented for *Neo4j* since to do

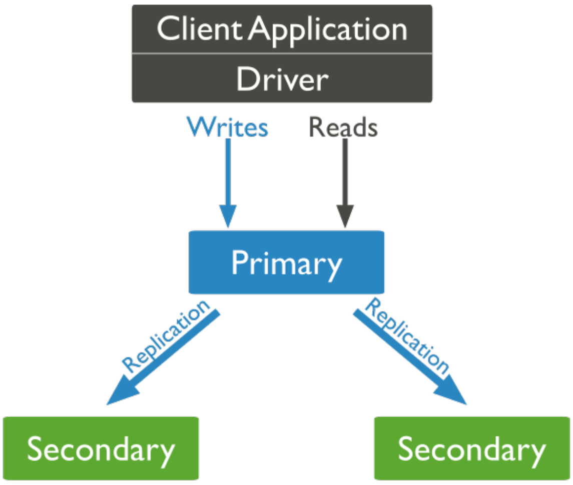

Figure 3.4: Replica Set

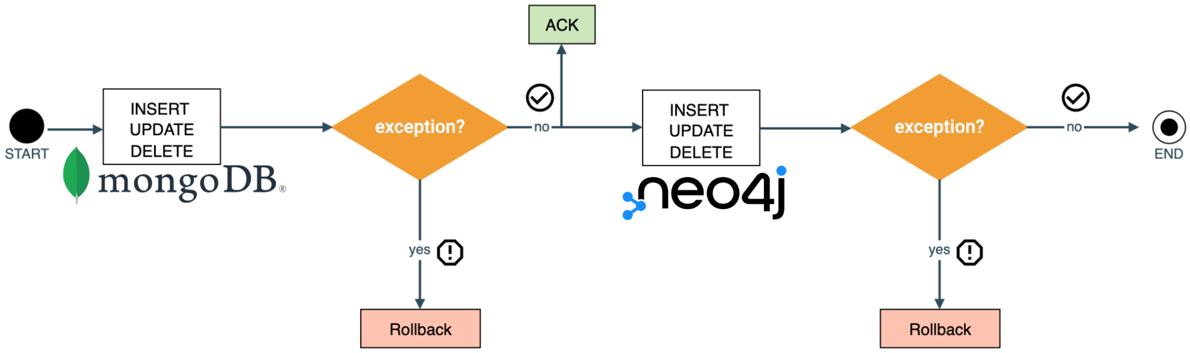

Figure 3.5: Handling consistency between *MongoDB* and *Neo4j*

so we would have needed the *Enterprise edition*). The *primary* is the only member in the replica set that receives write operations (figure **??**). MongoDB applies write operations on the primary and then records the operations on the primary's *oplog*. *Secondary* members replicate this log and apply the operations to their data sets.

Write Concern

3.6.2 Sharding

mongos instances will pass the *write concern* on to the shards

3.6.3 Handling inter-databases consistency

Having to deal with two different architectures (*DocumentDB* and *GraphDB*), we need to face the problem of redundancy and handle the consistency of data that are stored within both databases. Any rollback triggered by a failure of an insert, an update or a delete operation, can lead to inconsistencies. To address this problem, whenever a write operation is performed successfully on *MongoDB*, an ACK (*acknowledgement*) is sent to the user. Data on *GraphDB* will be eventually consistent: if any exception occurs when the updates over data on *Neo4j* are attempted, than a rollback operation will be executed in order to bring back both the databases in a consistent state (just as shown in figure **??**).

Eventual Consistency

Eventual consistency is dealt with by exploiting the *asynchronous execution support* in *Spring* and the *@Async* annotation. The caller will not have to wait for the complete execution of the called method: annotating a method of a bean with *@Async* will make it execute in a separate thread.

Chapter 4

Implementation

Application:

- *Java* with *Maven*
- *SpringBoot*
- *Javascript, HTML, CSS*

4.1 Spring Boot

4.2 Java Entities

4.2.1 Users

Simple User - Reviewer

Company Manager

Admin

4.2.2 Videogames

4.2.3 Reviews and Comments

Chapter 5

Queries

5.1 Mongo DB

5.2 Neo4j

5.2.1 Get Followers

5.2.2 Get Following

5.2.3 Friends' suggestions

From users' relationships

From common interests

5.3 Indexes

5.3.1 MongoDB

MongoDB					
Query	**Collection**	**Index Keys**	**Keys examined**	**Documents examined**	**ms**
Query1	videogames	- Name	numKeys numKeys	numDoc numDoc	*ms* *ms*
Query2	reviews	- author	numKeys numKeys	numDoc numDoc	*ms* *ms*
Query3	reviews	- {author, game}	numKeys numKeys	numDoc numDoc	*ms* *ms*
Query4	reviews	- game	numKeys numKeys	numDoc numDoc	*ms* *ms*
Query5	comments	- reviewId	numKeys numKeys	numDoc numDoc	*ms* *ms*
Query6	companies	- Name	numKeys numKeys	numDoc numDoc	*ms* *ms*
Query7	users	- username	numKeys numKeys	numDoc numDoc	*ms* *ms*
Query8	userImages	- username	numKeys numKeys	numDoc numDoc	*ms* *ms*

5.3.2 Neo4j

Neo4j					
Query	**Index Name**	**Nodes' Label**	**Property**	**Hits**	**(ms)**
Query1	- *username_index*	- User	- username	numHits numHits	*ms* *ms*
Query2	- *game_index*	- Game	- name	numHits numHits	*ms* *ms*
Query3	- *review_index*	- Review	- reviewId	numHits numHits	*ms* *ms*

Chapter 6

Chapter