Large-Scale and Multi-Structured Databases

Project Presentation

MyPodcastDB

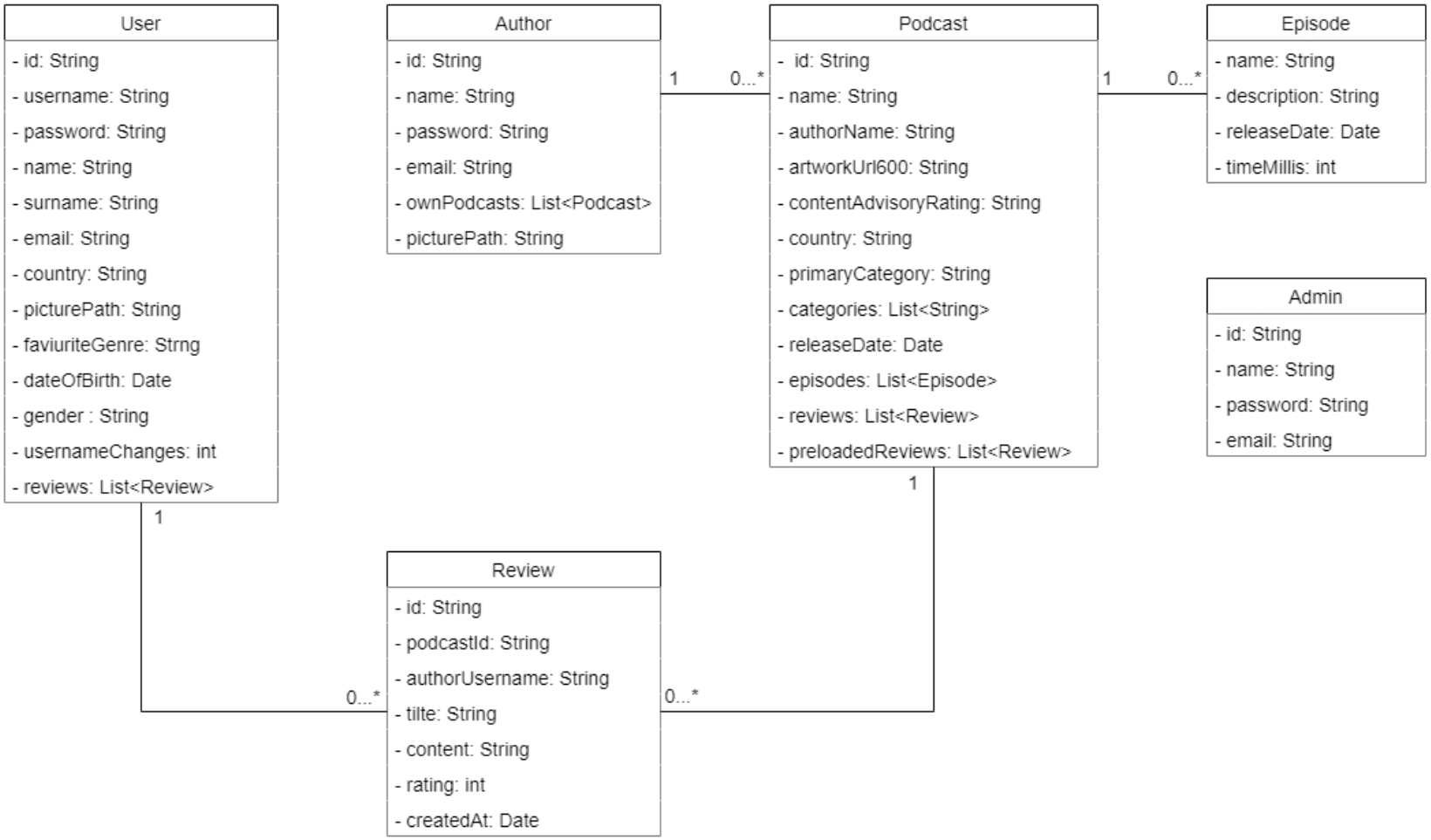
Biagio Cornacchia
Gianluca Gemini
Matteo Abaterusso

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
UNIVERSITÀ DI PISA
CROSSLAB Innovation for industry 4.0

Application Highlights

- **MyPodcastDB** is a **social network** that allows to recommend and review **podcasts**.
- The **user** can like a podcast, follow an author and follow other users. Based on these activities the user is shown suggested podcasts and authors
- The **user** can review a podcast through a vote and a comment
- Each **user** can add a podcast he wants to listen in the future in his personal watchlist
- An **author** can create, update and delete own podcasts, and follow other authors
- **Admins** can manage users, authors, podcasts and reviews. They also have access to the usage analytics of the application

UML Diagram


User
- id: String
- username: String
- password: String
- name: String
- surname: String
- email: String
- country: String
- picturePath: String
- faviuriteGenre: Strng
- dateOfBirth: Date
- gender : String
- usernameChanges: int
- reviews: List<Review>
Author
- id: String
- name: String
- password: String
- email: String
- ownPodcasts: List<Podcast>
- picturePath: String
1
0...*
Podcast
- id: String
- name: String
- authorName: String
- artworkUrl600: String
- contentAdvisoryRating: String
- country: String
- primaryCategory: String
- categories: List<String>
- releaseDate: Date
- episodes: List<Episode>
- reviews: List<Review>
- preloadedReviews: List<Review>
1
0...*
Episode
- name: String
- description: String
- releaseDate: Date
- timeMillis: int
Admin
- id: String
- name: String
- password: String
- email: String
1
0...*
Review
- id: String
- podcastId: String
- authorUsername: String
- tilte: String
- content: String
- rating: int
- createdAt: Date
0...*

Dataset Description

Source:

- https://www.kaggle.com/thoughtvector/podcastreviews (reviews in SQLite)
- https://itunes.apple.com (info podcasts in JSON)
- https://randomuser.me (users in JSON)

Description: *Dataset contains real podcasts information and reviews. All the users are randomly generated.*

Volume:

- *Users (116 MB)*
- *Authors (20.6 MB)*
- *Podcasts (414 MB)*
- *Reviews (168 MB)*

Variety: Three different sources are used to build the dataset.

Non-Functional Requirements

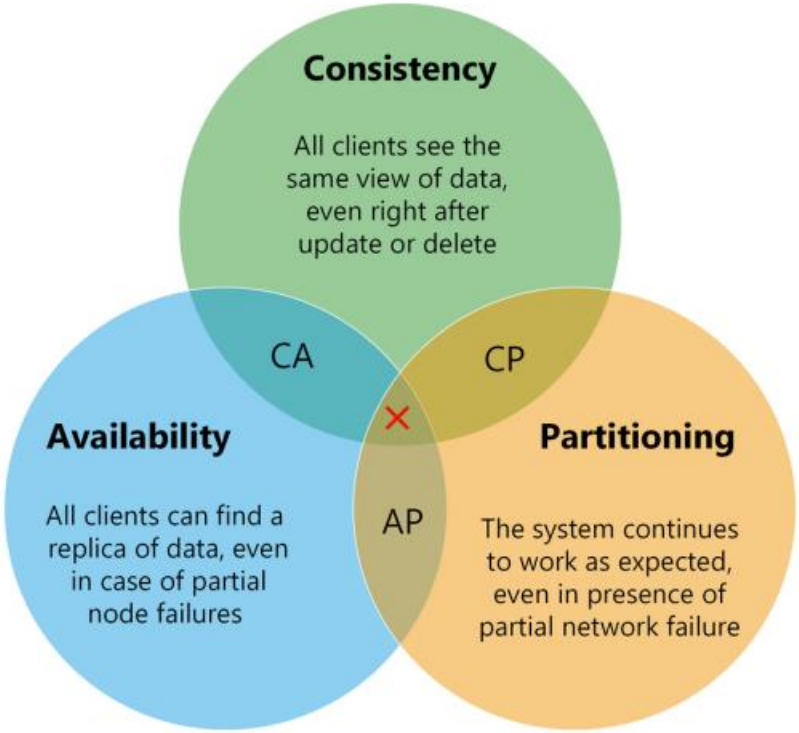
Product Requirements:

- **Usability**, the application must be simple and user friendly
- **High availability**, the displayed data might not be always up to date
- **Low latency** in accessing the database to have a responsive application
- **Tolerance** to the loss of data, avoiding a single point of failure

Organizational Requirement:

- When a user deletes his account, his **reviews** must be maintained

Handling CAP theorem issue



In order to satisfy the non-functional requirements, it is reasonable to sacrifice **consistency** in favor of **high-availability** and **partition tolerance**. Thus, an **AP solution** is used.

Requirements and Entities handled by Document DB

Entities:

- User
- Author
- Podcast
- Review
- Admin
- Query

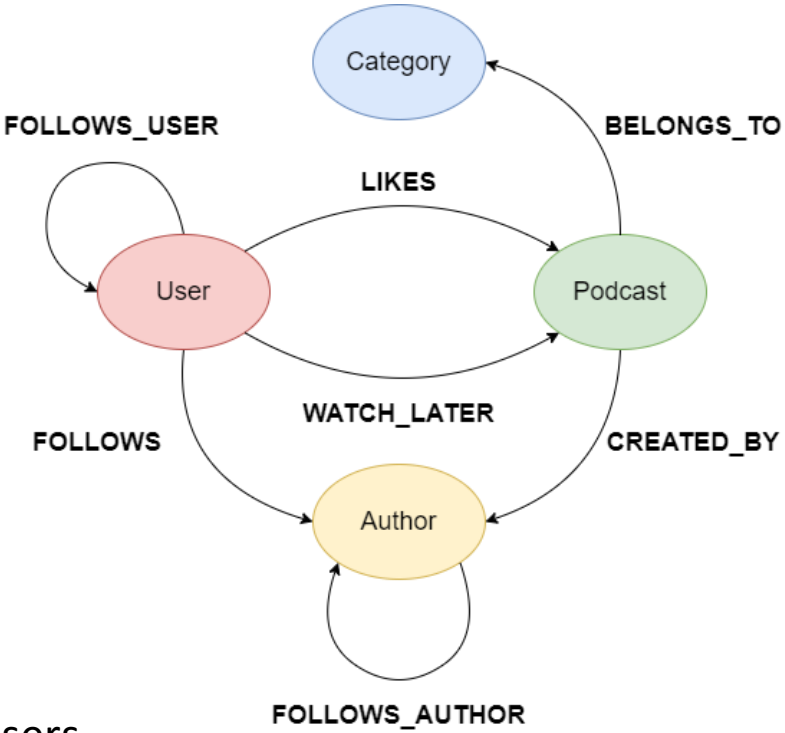
Queries:

- Show podcasts with highest avarage rating
- Show podcasts with highest average rating in a country
- Show average age of users per favourite category
- Show average age of users per country
- Show number of users per country
- Show podcasts with highest number of reviews
- Show countries with highest number of podcasts
- Top favourite categories for male, female and other

Requirements and Entities handled by Graph DB

Entities:

- User
- Author
- Category
- Podcast



Queries:

- Show the most followed author
- Show the most liked podcast
- Show the most numerous category
- Show the most appreciated category
- Show suggested users by liked podcasts
- Show suggested users by followed authors
- Show suggested podcasts liked by followed users
- Show suggested authors followed by followed user
- Show suggested podcasts based on the category of podcasts user liked
- Show suggested podcasts based on the authors of the podcasts in the watchlist

Database Consistency Management

An example of consistency management can be seen in the user update. The workflow is the following:

1. Update **user** document on Mongo
2. Update **user** node on Neo4J, if needed
3. Update **review** document on Mongo, if needed
4. Update **preloaded reviews** in **podcast**'s document, if needed

It is necessary to ensure the consistency between the **username** and **picture path** fields either for Mongo and Neo4J. Moreover, is necessary to update all the related embedded documents.

Possible data sharding

To guarantee **availability** and **fast responses**, the sharding proposal for the application uses the following fields as **sharding keys**:

- The **podcast id** for the podcast collection
- The **review id** for the review collection
- The **username** for the user collection
- The **author name** for the author collection
- The **admin name** for the admin collection
- The **query name** for the query collection

These fields have been chosen because they are the most used ones in the **CRUD operations**. Regarding the partition method, it has been decided to adopt a **hashing strategy** in order to distribute in a homogeneous way the documents.

Software and Hardware Architecture

Programming language:
- Java

Frameworks:
- Maven
- JavaFX

DBMS:
- MongoDB
- Neo4J