Large-Scale and Multi-Structured Databases

BoardVerse

Martina Fabiani
Tommaso Falaschi
Emanuele Respino

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB
Innovation for industry 4.0

Application Highlights

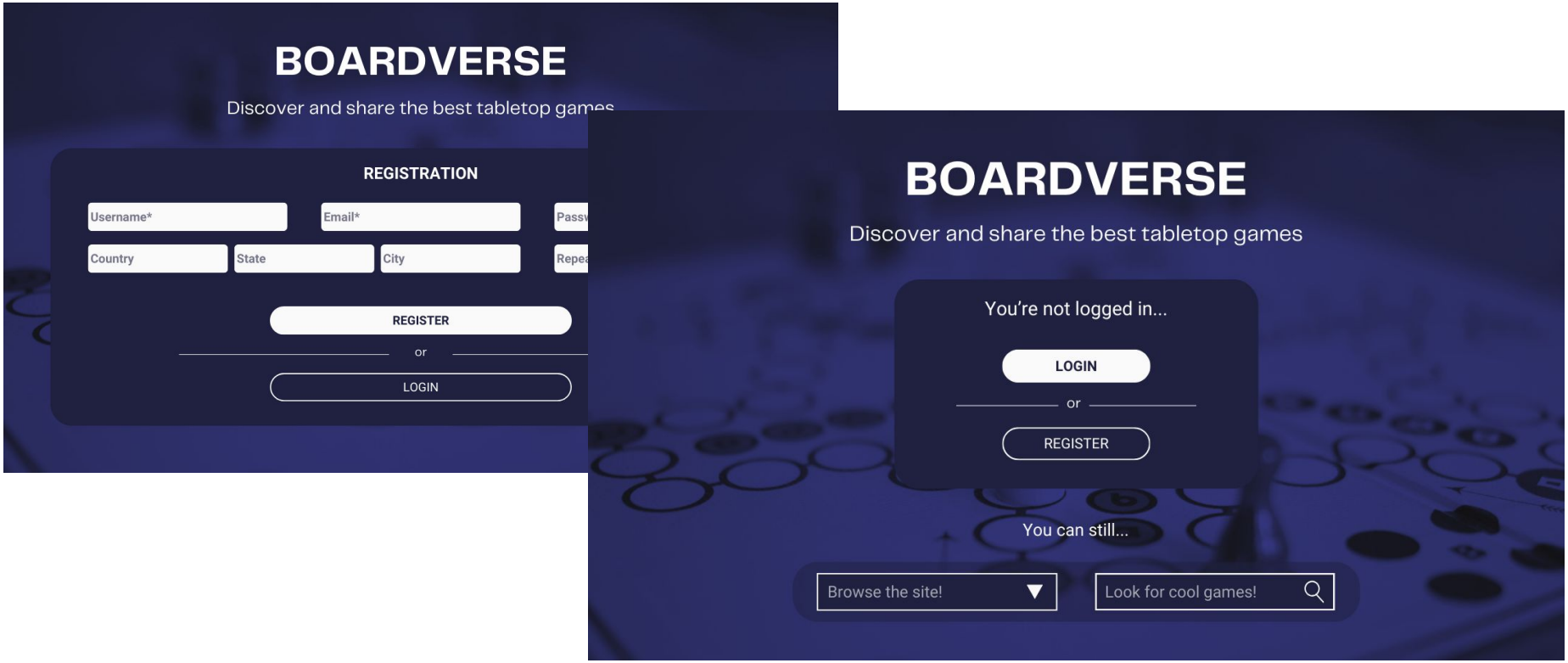
BoardVerse is a web application for board game players, providing a hub to discover games, engage with other players, share interests, and join exciting tournaments.

Main Features:

- **Game discovery:** BV provides an extensive board game catalogue with detailed descriptions, allowing users to search using various kind of filters and game rankings based on popularity users-related activity.
- **Community Interaction**: the platform gives registered users the opportunity to review games, create and participate in discussions and tournaments, giving the possibility to create a network following other users.
- **Personalized Recommendation**: exploiting the user network, the platform gives players the opportunity to discover new games, tournaments and potential friends based on similar interests or network.
- **Admin Analytics & Moderation**: administrators can manage the game catalogue, moderate user content and access insights on community trends.

Actors and main mock-ups (i)

Unregistered User

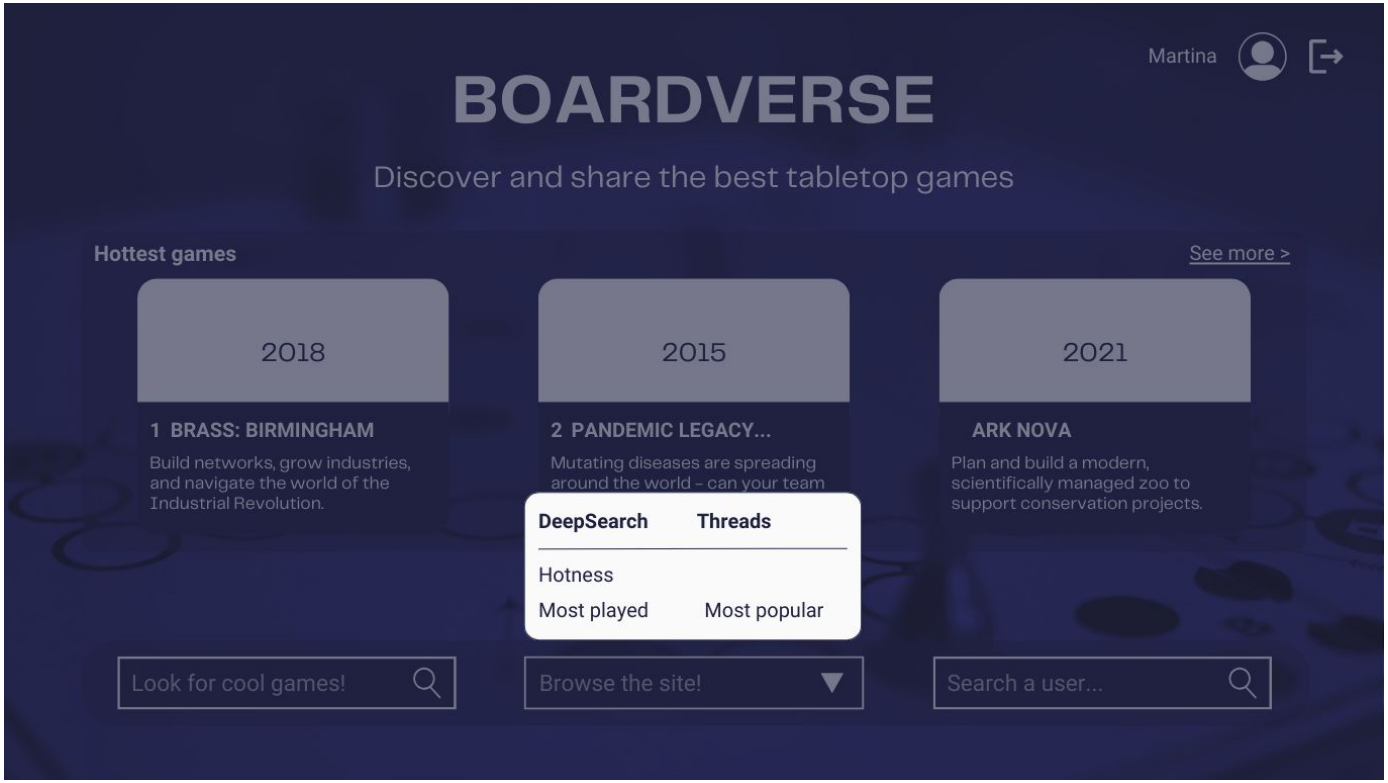


Welcome page for a non-registered user where you can search for a game by name or browse games by filtering various parameters, or register to get additional features

Actors and main mock-ups (ii)

Registered User and Administrator

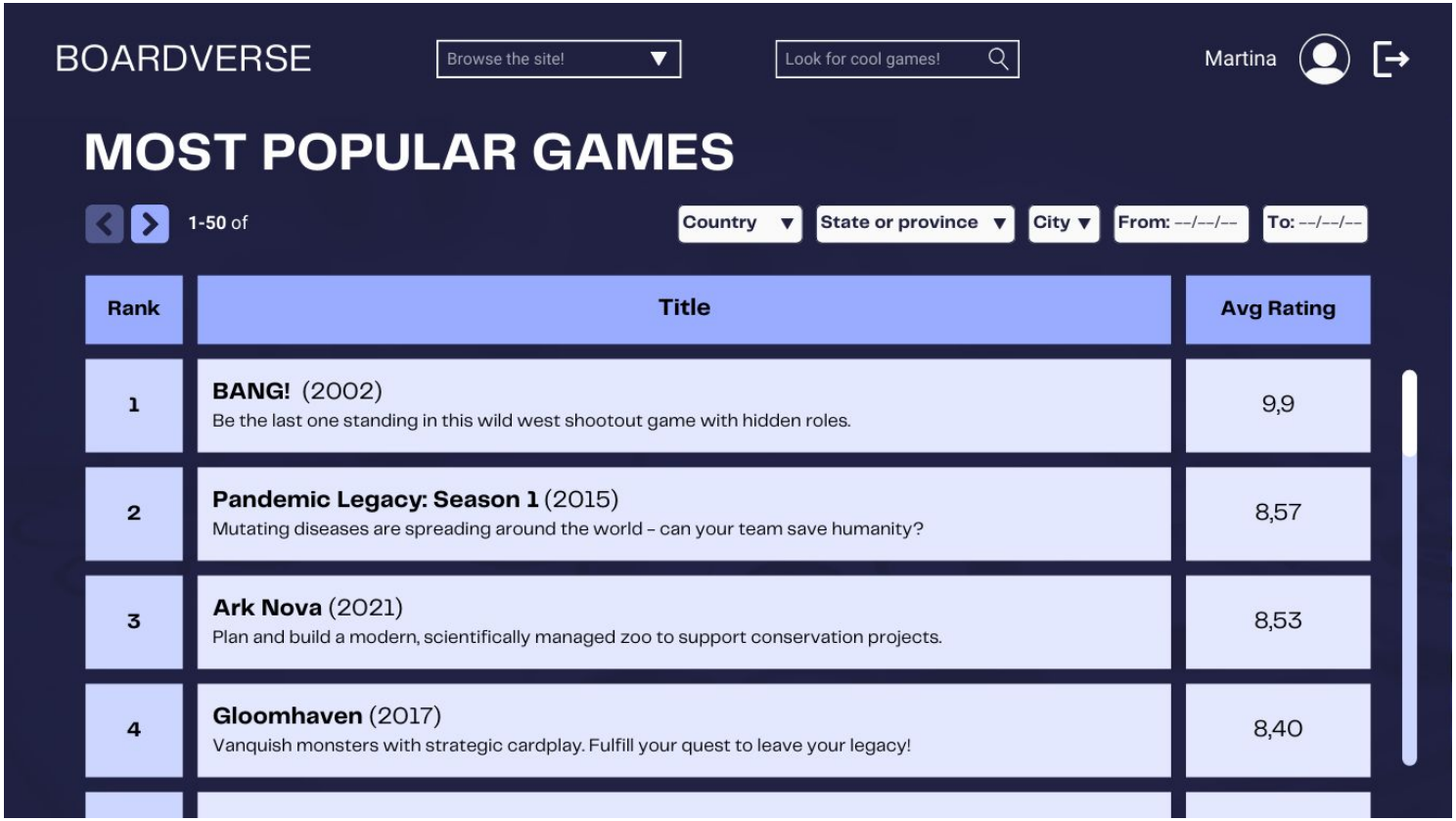
Actions that can be performed by both registered user and admin are highlighted with a *red button*, while actions that can only be done by the admin are indicated with a *green button*.



The main page is slightly different for the registered user.
In this case, other registered users can also be searched.

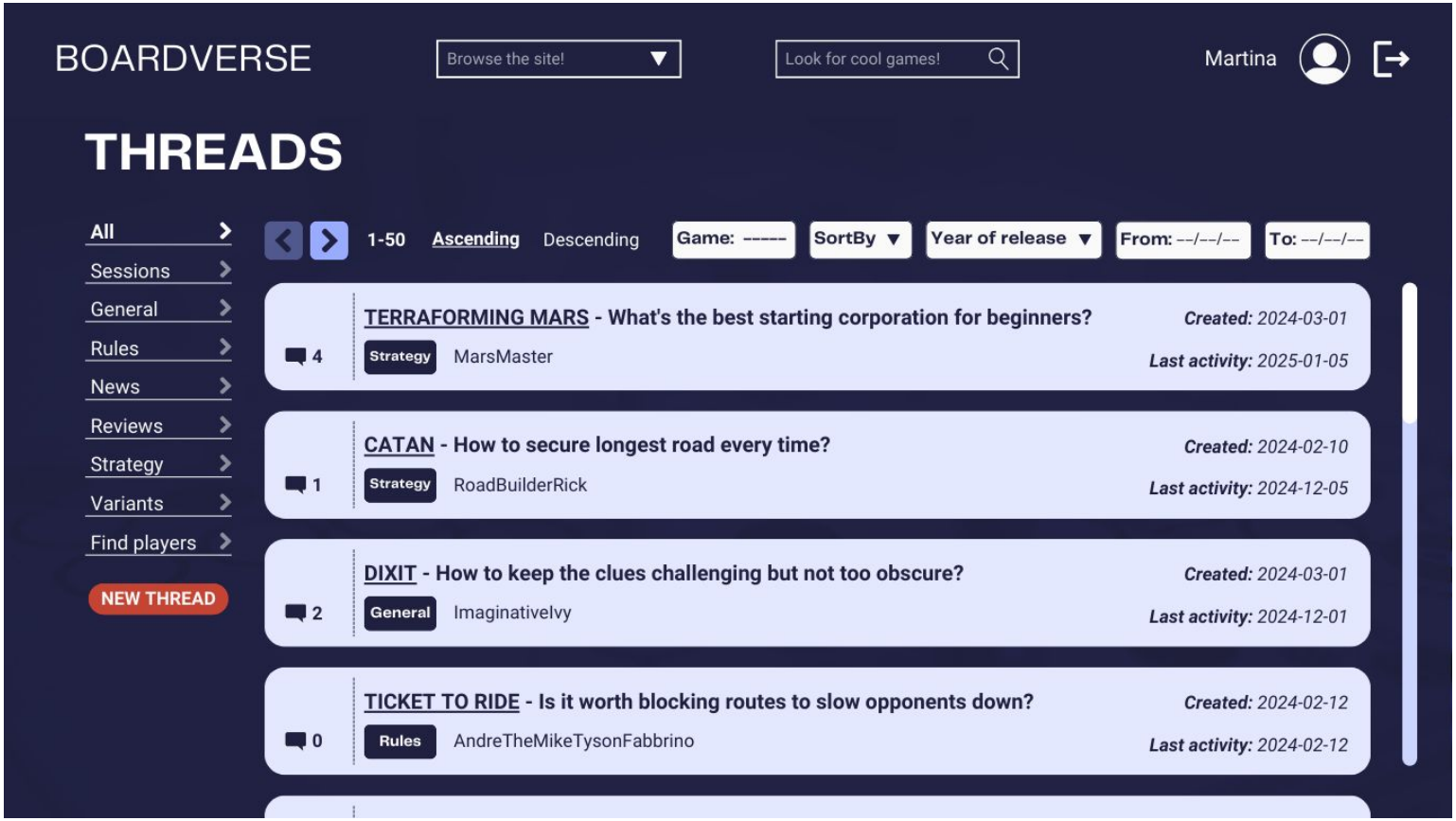
Actors and main mock-ups (ii)

Registered User and Administrator



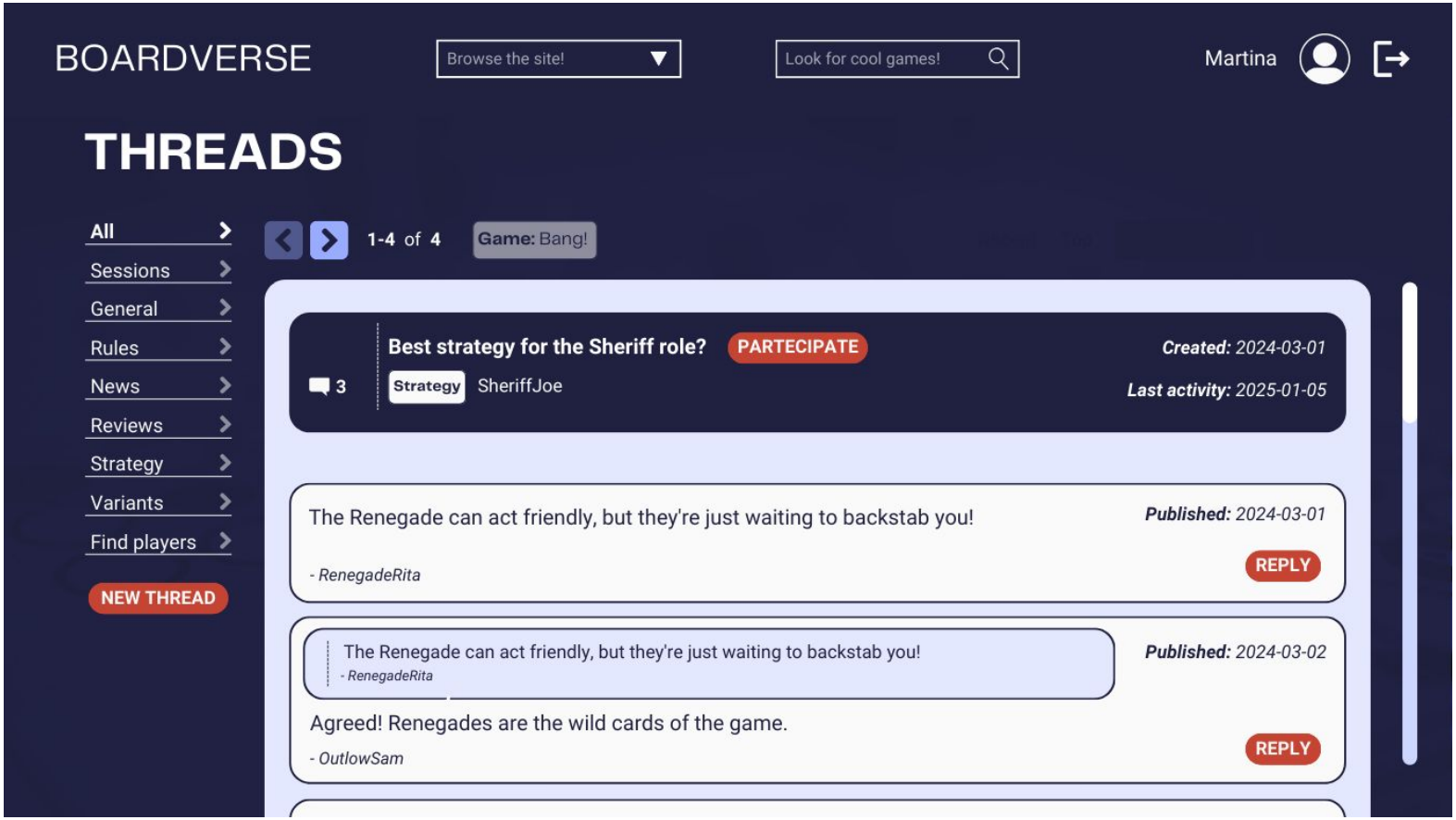
The rankings can be viewed by modifying several parameters to customize the search.

Actors and main mock-ups (iii)

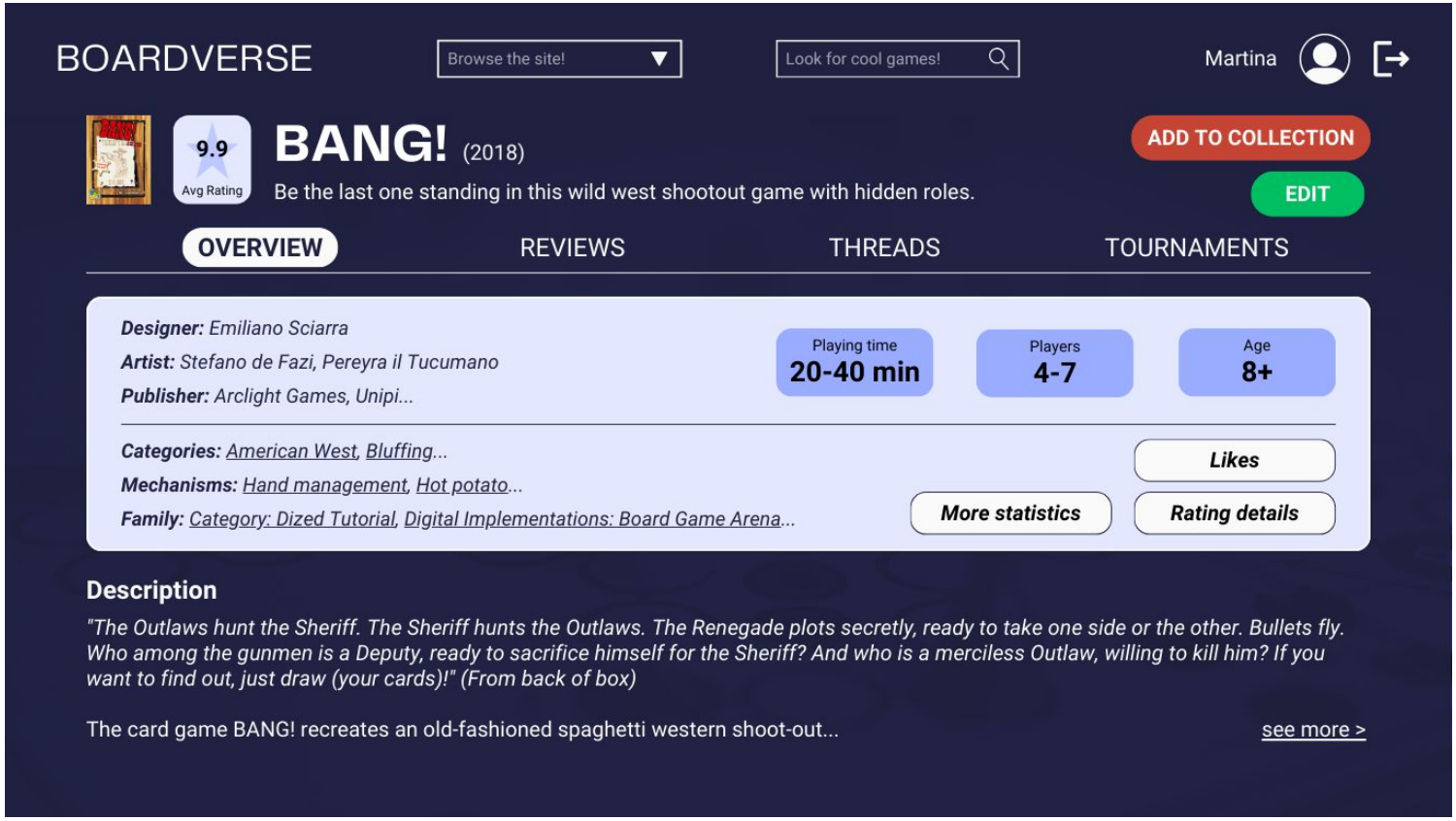


This is the *Threads page*, where you can apply filters to find and view the threads that match your preferences.

Actors and main mock-ups (iv)



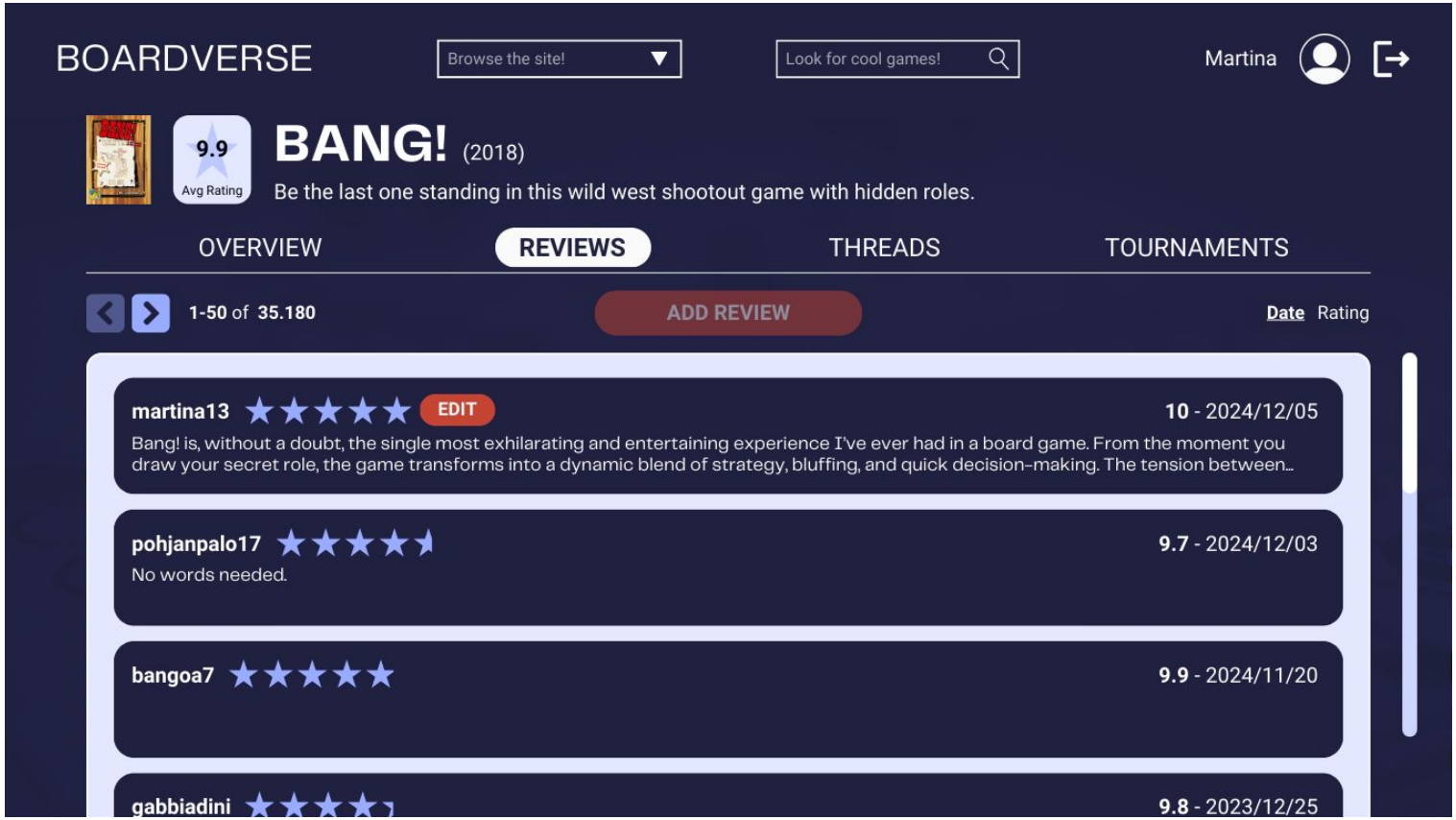
Actors and main mock-ups (v)



This is the main page of a game.

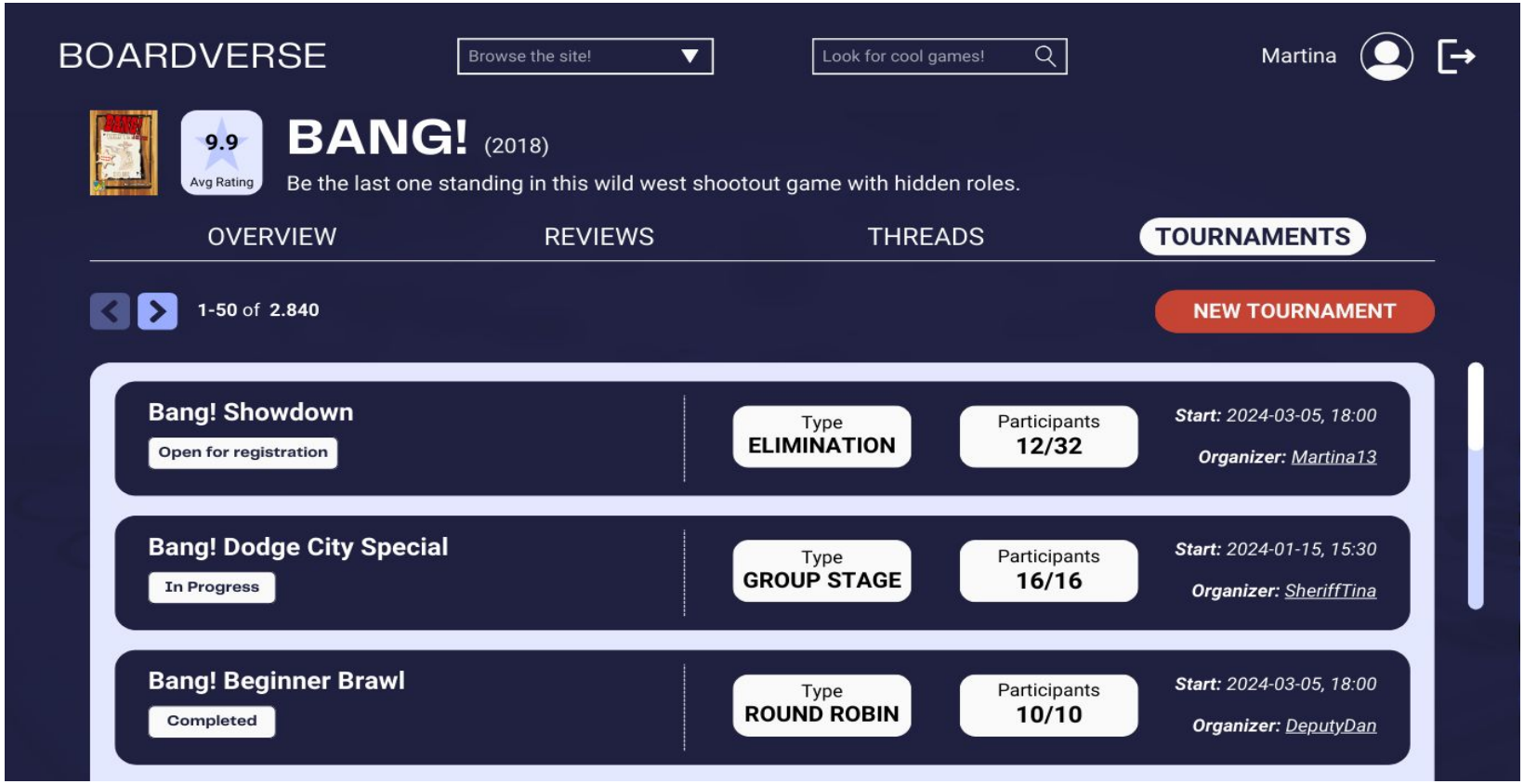
Here you can view information about the board game, having also the possibility to switch between threads - *similar to the previous page, omitted for brevity* - …

Actors and main mock-ups (vi)



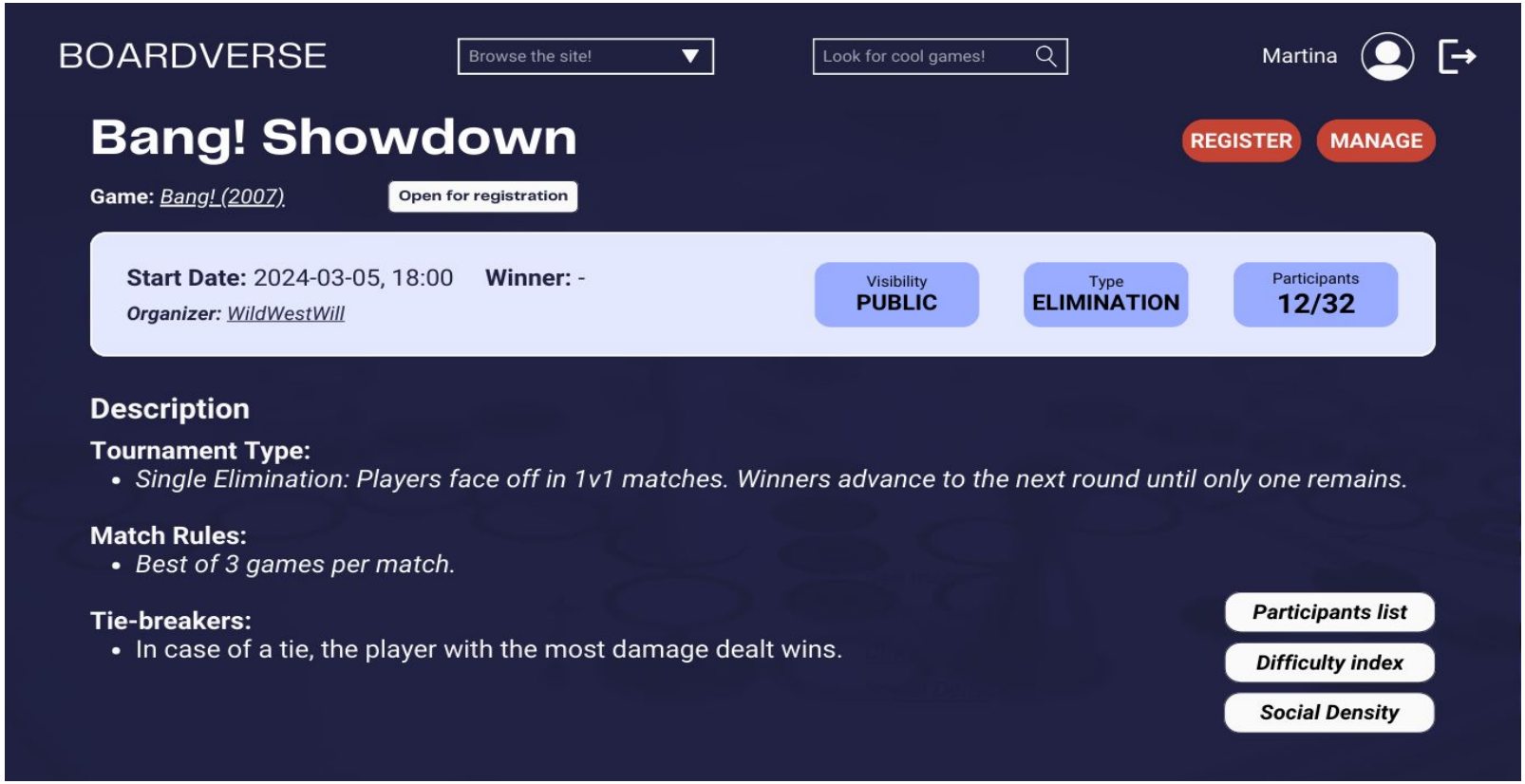
…reviews…

Actors and main mock-ups (vii)



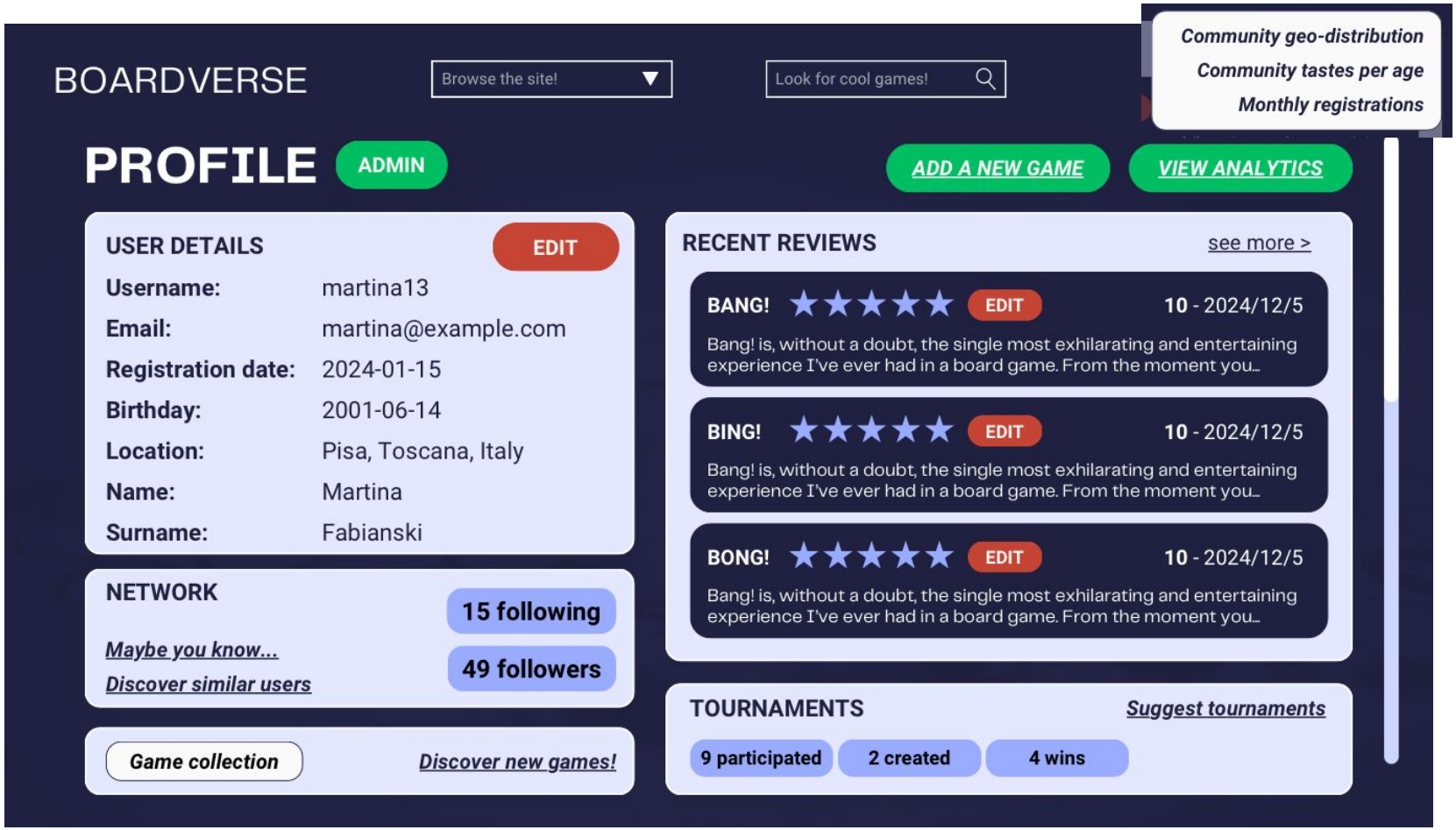
…and *tournaments*, which are displayed through a simple preview.

Actors and main mock-ups (vii)



Clicking on the preview, we can have access to the tournament page with additional informations and the option to join when possible.

Actors and main mock-ups (viii)



On the *main profile* of a registered user its details can be found, including personal information, most recent reviews, its game collection, and information about its network and the tournaments it's involved in.

Dataset Description

Sources: Two platform, **BoardGameGeek (BGG)**, which offers an XML API (*API2*) to retrieve data, and **BoardGameArena (BGA)** that instead needed web scraping techniques, as no dedicated APIs are available, neither official and unofficial.

Description: The dataset includes real-world data information about games, reviews, forums, users **(BGG)** and tournaments **(BGA)**, with synthetic enrichment.

Volume: starting from an overall amount of 950MB of scraped data, the final dataset volume achieved is approximately **1.2GB**.

Variety: Achieved gathering on two different sites (***BGG, BGA***), with different game catalogue and resource-identification systems.

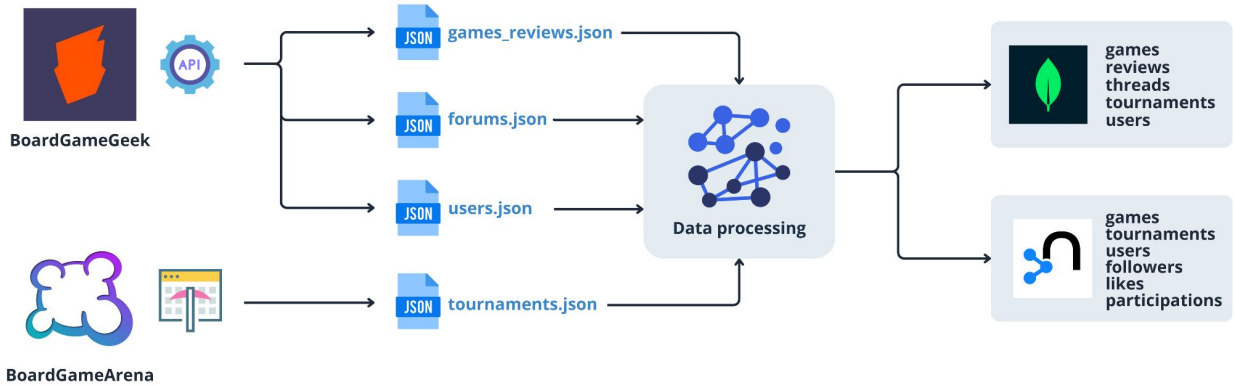
Velocity/Variability: Data remains mostly static, though certain elements have a shifting relevance over time. Tournament-related data initially can hold higher importance but gradually loses user interest. In contrast, discussions retain long-term value, as users may revisit past topics, engage in older threads, or reference previous insights.

Dataset Description

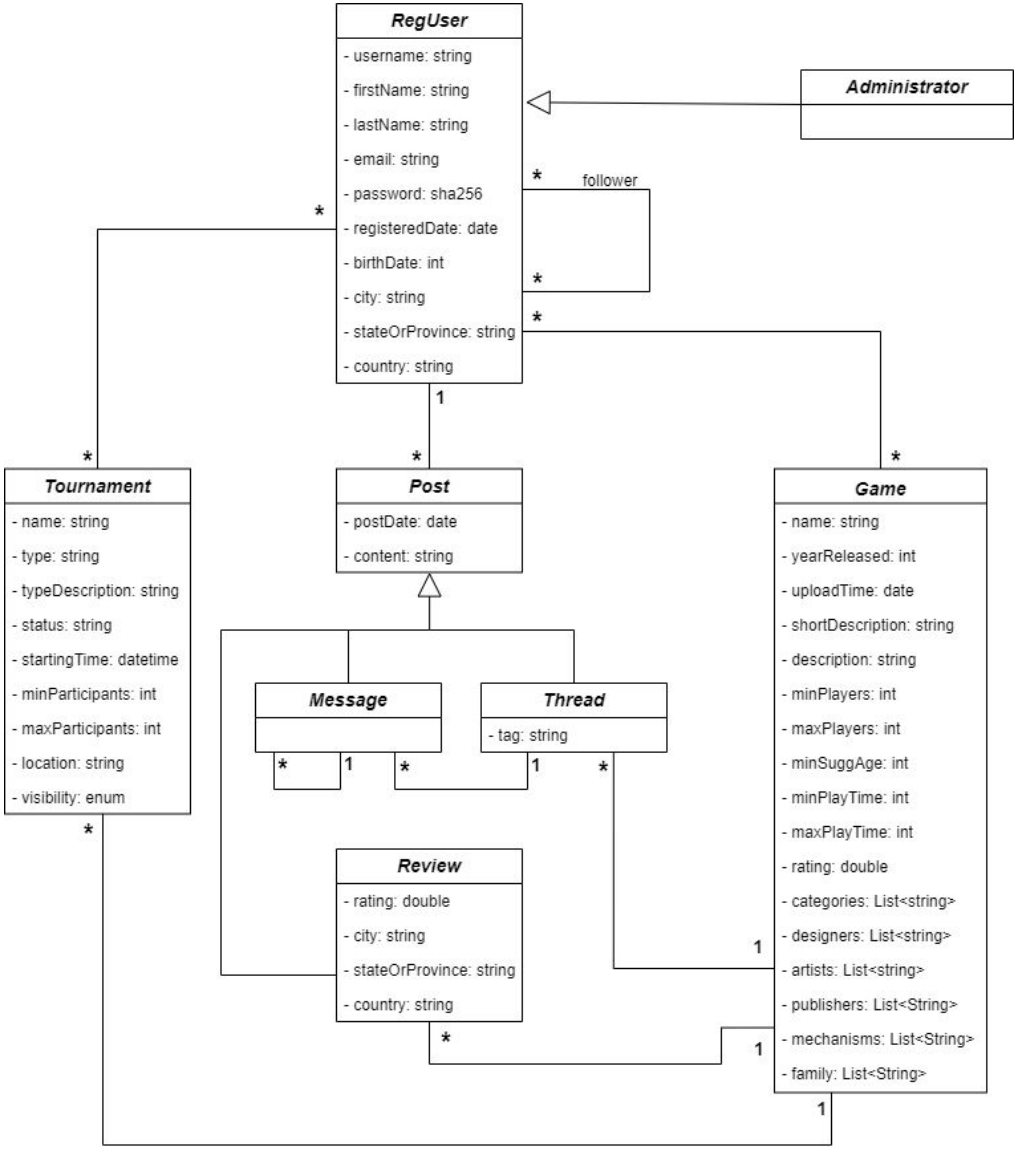
Data Retrieval & Processing

Data was collected initially from **BGG**, retrieving metadata about the best 2.000 ranked board games, including at most 500 review each. Then, 70 discussion threads per game (*avg.*) were retrieved from the forum section, including 40 messages per thread (*avg.*). From **BGA** instead, we extracted information about approximately 2.000 tournaments from 200 games, accordingly to platform policies.

Collected data underwent cleaning and normalization to ensure consistency, including standardizing formats and correcting location inconsistencies. Missing attributes were enriched, while user network was simulated based on silimal behaviours.



UML Design Class Diagram



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Application non-functional requirements

- The system must be expose its resources accordingly to OpenAPI specifications using RESTful APIs.
- The system must be developed in an OOP language.
- The system must be capable of returning to the user quick responses.
- The system must ensure high availability.
- The system should be capable of handling growth in user traffic by scaling effectively.
- The system must be fault-tolerant and capable of maintaining essential functionality in the event of partial system failures.
- The system must use a basic authentication protocol to secure access.
- The system must encrypt users' passwords.

Document DB Design

Users collection


```
_id: "40acdbcf-be6d-4e9c-a6bc-a23e3515d3fd"
username : "respino"
firstName : "Emanuele"
lastName : "Respino"
email : "emanuele.respino.4517@hushmail.com"
password : "87b555c0457fb41ebe56d8777613763151044416 93aece27297664aa02541aeb"
birthDate : 1956-05-01T00:00:00.000+00:00
location : Object
  city : null
  stateOrProvince : "Toscana"
  country : "Italy"
followers : 9
following : 40
tournaments : Object
  partecipated : 20
  won : 0
  created : 4
mostRecentReviews : Array (3)
  0: Object
    postDate : 2022-11-25T00:00:00.000+00:00
    rating : 7
    content : "Play with a friend who owns a copy. Brilliant game, terrific story and…"
    _id : "bed14528-79a1-44f1-950c-637b1950cbb0"
    game : Object
  1: Object
  2: Object
role : "ROLE_ADMIN"
registeredDate : 2024-06-22T00:00:00.000+00:00
```


Games collection


```
_id: "5f5bd4bb-db8b-4003-820c-61d6eb9cbe2e"
name : "Alma Mater"
yearReleased : 2020
uploadTime : 2020-12-10T00:00:00.000+00:00
description : "Alma Mater has players serving as a headmaster of one of the independe…"
shortDescription : "Players serve as headmasters of universities seeking to establish them…"
averageRating : 6.988235620689656
ratingVoters : 290
minPlayers : 2
maxPlayers : 4
minSuggAge : 12
minPlaytime : 90
maxPlaytime : 150
designers : Array (5)
artists : Array (1)
publishers : Array (5)
categories : Array (1)
mechanics : Array (5)
family : Array (1)
```


Reviews collection


```
_id: "63a3a9ed-d61c-4e86-a5c2-6111d405b9cf"
postDate : 2022-09-14T00:00:00.000+00:00
location : Object
  city : null
  stateOrProvince : "Toscana"
  country : "Italy"
game : Object
  name : "Frosthaven"
  yearReleased : 2022
  shortDescription : "Adventure in the frozen north and build up your outpost throughout an …"
  _id : "5f0df14a-8760-47b8-8130-8da8591af0c9"
rating : 10
content : "Lots of game play packed into this box."
authorUsername : "respino"
authorBirthDate : 1956-05-01T00:00:00.000+00:00
```

Document DB Design

Tournaments collection


```
_id: "3a294b18-2477-4110-b28b-42f531363976"
name : "The Round • Stephenson's Rocket month"
▾ game : Object
    name : "Ark Nova"
    yearReleased : 2021
    _id : "b8e4e97d-e348-4ee4-a6d9-81590a4cbca2"
type : "Groups Stage"
typeDescription : "Players are first divided in groups for a first stage, winners advance…"
startingTime : 2025-01-26T16:00:00.000+00:00
▾ location : Object
    city : null
    stateOrProvince : "California"
    country : "United States"
numParticipants : 225
minParticipants : 45
maxParticipants : 225
administrator : "davido"
winner : null
visibility : "PUBLIC"
▾ options : Array (18)
  ▾ 0: Object
      optionName : "Number of players in a match"
      optionValue : "2"
  ▾ 1: Object
      optionName : "Number of players in a match (minimum)"
      optionValue : "2"
```


Threads collection


```
_id: "5198e8bd-6a5f-4bb1-9af7-352f780b3c69"
content : "Mini-review after playing tons of Lancashire ("In Your Face" -element …"
postDate : 2024-09-05T19:27:02.000+00:00
lastPostDate : 2025-01-10T13:57:07.000+00:00
tag : "Reviews"
▸ game : Object
▾ messages : Array (11)
  ▾ 0: Object
      postDate : 2024-09-05T19:27:02.000+00:00
      content : "Both are great games. As most people know, 80% of the rules are the sa…"
      authorUsername : "seepieceeggshell"
      _id : "74384a9e-74f6-456d-8bac-b542b8f591b3"
  ▾ 1: Object
      postDate : 2024-09-06T00:04:38.000+00:00
      content : "I wouldn't say one game is "better" than the other because they are bo…"
      authorUsername : "keso55"
      _id : "a2043ad5-97ee-45a5-ac5e-ec222801afad"
  ▸ 2: Object
  ▸ 3: Object
  ▸ 4: Object
  ▸ 5: Object
  ▸ 6: Object
  ▸ 7: Object
  ▸ 8: Object
  ▸ 9: Object
  ▸ 10: Object
authorUsername : "seepieceeggshell"
```

Relevant MongoDB queries

Hottest games based on threads activity

```
@Aggregation(pipeline = {  1 usage  EmanueleRsp
        "{ $addFields: { messages: { $filter: { input: '$messages', as: 'message', cond: { $and: [ { $gte: ['$$message.postDate', ?0 ] }, { $lte: ['$$message.postDate', ?1] } ] } } } } }",

        "{ $addFields: { messages: { $map: { input: '$messages', as: 'message', in: { $let: { vars: { maxDate: ?1, minDate: ?0, dateDif: { $subtract: [?1, ?0] } },"  +
                " in: { $mergeObjects: ['$$message', { weight: { $add: [0.1, { $divide: [ { $multiply: [1, { $subtract: ['$$message.postDate', ?0] } ] }, { $subtract: [?1, ?0] } ] } ] } } ] } } } } } } }",

        "{ $project: { game: 1, totalWeight: { $sum: '$messages.weight' } } }",
        "{ $group: { _id: '$game._id', game: { $first: '$game' }, importanceIndex: { $sum: '$totalWeight' } } }",
        "{ $project: { _id: '$_id', name: '$game.name', yearReleased: '$game.yearReleased', importanceIndex: 1 } }",
        "{ $sort: { importanceIndex: -1 } }"
})
Slice<BestGameThreadDTO> getNormalizedGameRankingsWithDetails(Date startDate, Date endDate, Pageable pageable);
```

Ranking of games based on reviews filtered by postDate and user Location

```
@Aggregation(pipeline = {  1 usage  new *
        "{ '$match': { 'postDate': { '$gte': ?0, '$lte': ?1 } } }",
        "{ '$match': { '$and': [ "
                + "  { '$or': [ { '$expr': { '$eq': [ :#{#country}, null ] } }, { 'location.country': :#{#country} } ] }, "
                + "  { '$or': [ { '$expr': { '$eq': [ :#{#state}, null ] } }, { 'location.stateOrProvince': :#{#state} } ] }, "
                + "  { '$or': [ { '$expr': { '$eq': [ :#{#city}, null ] } }, { 'location.city': :#{#city} } ] } "
                + "] } }",
        "{ '$group': { "
                + "  '_id': '$game._id', 'name': { '$first': '$game.name' }, 'yearReleased': { '$first': '$game.yearReleased' }, "
                + "  'shortDescription': { '$first': '$game.shortDescription' }, 'averageRating': { '$avg': '$rating' }, "
                + "  'reviewCount': { '$sum': 1 } "
                + "} }",
        "{ '$match': { 'reviewCount': { '$gte': 30 } } }",
        "{ '$sort': { 'averageRating': -1 } }",
        "{ '$project': { "
                + "  '_id': 1, 'name': 1, 'shortDescription': 1, 'yearReleased': 1, "
                + "  'averageRating': { '$round': [ '$averageRating', 2 ] } "
                + "} }"
})
Slice<GameRankPreviewDTO> findAverageRatingByPostDateLocation(Date startDate, Date endDate, String country, String state, String city, Pageable pageable);
```

The weight of each message is computed using the formula:

0.1 + (1 * ((postDate - startDate) / (endDate - startDate)))

which normalizes the weight based on the postDate relative to the reference range.

The weight increases **linearly** as the message date gets closer to endDate

This step ensures **more recent messages contribute more** to the ranking.

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Relevant MongoDB queries

Top 10 most played games

```
@Aggregation(pipeline = {  1 usage   Martina
        "{ $match: { 'startingTime': { $gte: ?0, $lt: ?1 }, 'numParticipants': { $gt: 10 } } }",
        "{ $addFields: { 'weight': { '$switch': { 'branches': [ " +
                "{ 'case': { '$eq': ['$visibility', 'PUBLIC'] }, 'then': 2 }, " +
                "{ 'case': { '$eq': ['$visibility', 'INVITE'] }, 'then': 1.5 }, " +
                "{ 'case': { '$eq': ['$visibility', 'PRIVATE'] }, 'then': 1 } ], " +
                "'default': 1 } }, " +
                "'weightedParticipants': { '$multiply': ['$numParticipants', " +
                "{ '$switch': { 'branches': [ " +
                "{ 'case': { '$eq': ['$visibility', 'PUBLIC'] }, 'then': 2 }, " +
                "{ 'case': { '$eq': ['$visibility', 'INVITE'] }, 'then': 1.5 }, " +
                "{ 'case': { '$eq': ['$visibility', 'PRIVATE'] }, 'then': 1 } ], " +
                "'default': 1 } }] } } }",
        "{ $group: { '_id': '$game._id', 'name': { '$first': '$game.name' }, 'yearReleased': { '$first': '$game.yearReleased' }, " +
                "'totalWeightedParticipants': { '$sum': '$weightedParticipants' }, 'totalWeight': { '$sum': '$weight' } } }",
        "{ $addFields: { 'averageWeightedParticipants': { '$divide': ['$totalWeightedParticipants', '$totalWeight'] } } }",
        "{ $sort: { 'averageWeightedParticipants': -1 } }",
        "{ $limit: 10 }",
        "{ $project: { '_id': 0, 'gameID': '$_id', 'name': 1, 'yearReleased': 1, 'averageWeightedParticipants': 1 } }"
})
List<MostPlayedGameDTO> findTop10GamesWithHighestAverageParticipation(Date startDate, Date endDate);
```

Community tastes per age bucket (admin)

```
@Aggregation(pipeline = {  1 usage   new *
        "{ $match: { rating: { $ne: null } } }",
        "{ $addFields: { age: { $dateDiff: { startDate: \"$authorBirthDate\", endDate: \"$$NOW\", unit: \"year\" } } } }",
        "{ $match: { age: { $gte: 10, $lte: 99 } } }",
        "{ $addFields: { ageBracket: { $concat: [ " +
                "{ $toString: { $multiply: [10, { $floor: { $divide: [\"$age\", 10] } }] } }, \"-\", " +
                "{ $toString: { $add: [ { $multiply: [10, { $floor: { $divide: [\"$age\", 10] } }] }, 9 ] } } ] } } }",
        "{ $group: { _id: { ageBracket: \"$ageBracket\", game: \"$game._id\" }, " +
                "name: { $first: \"$game.name\" }, yearReleased: { $first: \"$game.yearReleased\" }, " +
                "averageRating: { $avg: \"$rating\" }, totalReviews: { $sum: 1 } } }",
        "{ $match: { totalReviews: { $gte: 30 } } }",
        "{ $sort: { \"_id.ageBracket\": 1, \"averageRating\": -1 } }",
        "{ $group: { _id: \"$_id.ageBracket\", gameID: { $first: \"$_id.game\" }, " +
                "name: { $first: \"$name\" }, yearReleased: { $first: \"$yearReleased\" }, " +
                "bestAvgRating: { $first: \"$averageRating\" }, totalReviews: { $first: \"$totalReviews\" } } }",
        "{ $sort: { \"_id\": 1 } }"
})
List<BestGameAgeDTO> findBestGameByAgeBrackets();
```

Other relevant queries:

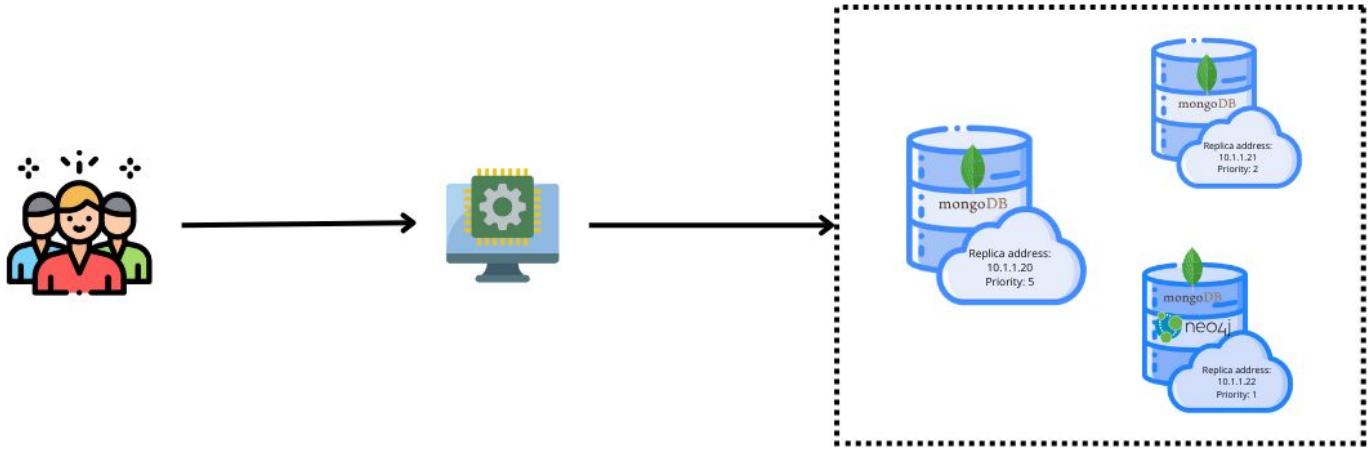
- Game rating statistics
- Community geographical distribution (admin)
- Monthly user registrations by Year (admin)

MongoDB Replica Set Configuration

Goal: Ensure uninterrupted data access by implementing redundancy and failover mechanisms.

The cluster is designed with **three replicas for the MongoDB document database** to provide high availability and fault tolerance, and a **single replica for Neo4j**, with scalability in mind for future enhancements.

- **readPreference=nearest**: Optimizes read performance by routing requests to the nearest node, minimizing latency and evenly distributing the workload across the cluster.
- **w=majority**: Guarantees data durability and fault tolerance by acknowledging write operations only after they are committed to the majority of nodes, ensuring data consistency even during failover.



MongoDB Indexes

Users collection:

- *username:* Adding an index on the username field is highly efficient because optimizes the performance of operations like login, signup, and browsing users, which are among the most frequent actions performed on the database.

	Without index	With index
Returned	1	1
Execution Time (ms)	36	1
Keys examined	0	1
Documents examined	5000	1

Performance of the getUserByUsername function

Games collection:

- *name:* An index has been added to the name field in the Games collection to significantly enhance the performance of the browse game operation, which is the primary activity of the application. The performance tests are similar to the ones on users name.

MongoDB Indexes

Reviews collection:

- game.id: Given the high frequency of queries that load reviews for a specific game or perform searches using game.id, we decided to add an index on the game.id field in the Reviews collection. This index significantly enhances query performance by enabling faster and more efficient data retrieval.

Threads collection

- game.id

Tournaments collection

- game.id

As with the Reviews collection, we have also decided to add an index on game.id in the Threads and Tournaments collections for the same reasons mentioned earlier.

Performance of the getGameReviews function

	Without index	With index
Returned	361	361
Execution Time (ms)	3321	6
Keys examined	0	361
Documents examined	674945	361

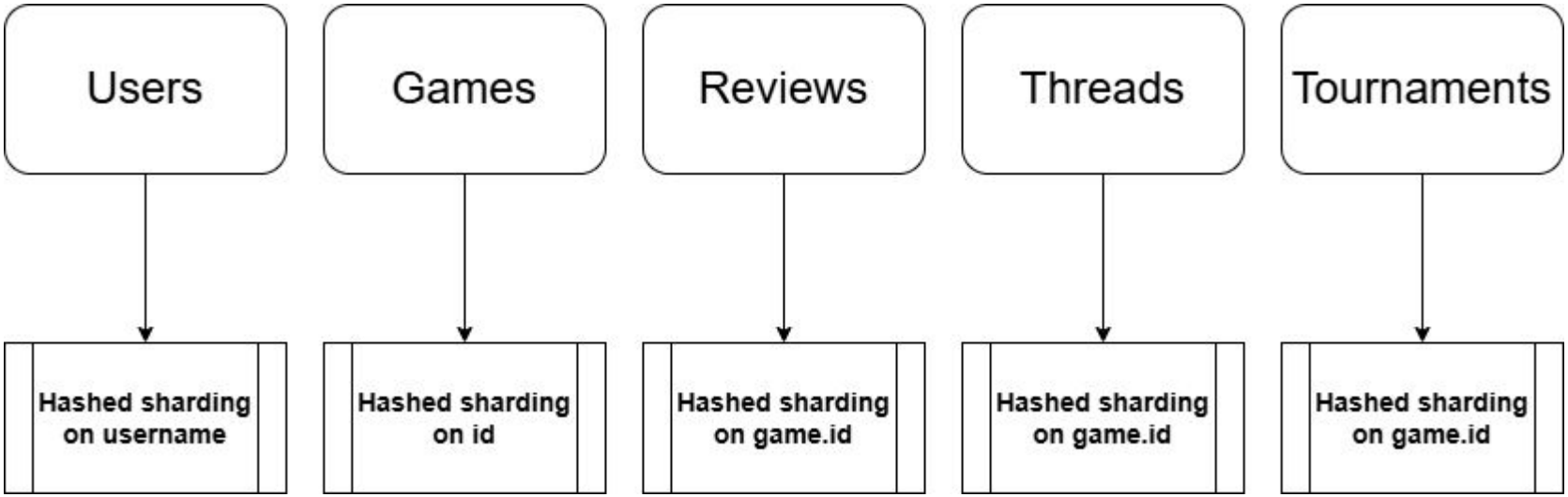
Performance of the getGameThreads function

	Without index	With index
Returned	50	50
Execution Time (ms)	11	0
Keys examined	0	50
Documents examined	9666	50

Performance of the getGameTournaments function

	Without index	With index
Returned	20	20
Execution Time (ms)	449	2
Keys examined	0	20
Documents examined	20006	20

Discussion on MongoDB Data Sharding



Graph DB Design

Nodes

User
username

Game
id, name, yearReleased, shortDescription, categories

Tournament
id, name, visibility, maxParticipants, startingTime

Relationships

User – :FOLLOWS {*Timestamp*} → User

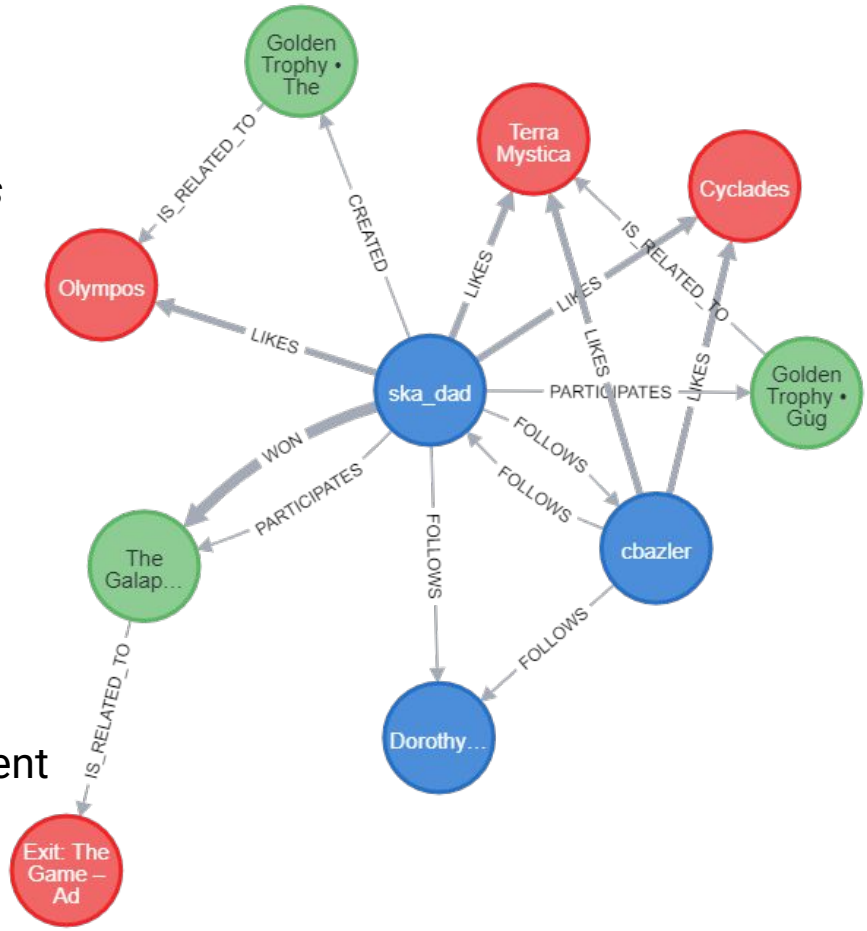
User – :LIKES {*Timestamp*} → Game

User – :CREATED {*Timestamp*} →Tournament

User – :PARTICIPATES {*Timestamp*} → Tournament

User – :WON {*Timestamp*} → Tournament

Tournament – :IS_RELATED_TO → Game



Graph DB Design

Friends suggestion

Domain Specific: Suggest new users to follow based on common follows.

Graph-centric:

1. Starting from *User A* node,
2. Identify second-degree relationships by following the *FOLLOWS* edges from A's direct connections.
3. Count mutual friends and exclude already followed users and A itself
4. Rank and return users by shared connections in descending order as friend suggestions.

```
@Query("""
    MATCH (currentUser:User {username: $username})-[:FOLLOWS]->(followedUser:User)-[:FOLLOWS]->(suggestedUser:User)
    WHERE NOT (currentUser)-[:FOLLOWS]->(suggestedUser) AND suggestedUser.username <> $username
    RETURN suggestedUser.username AS username, COUNT(suggestedUser) AS commonFollowers
    ORDER BY commonFollowers DESC
    SKIP $pageSize * ($pageNumber - 1)
    LIMIT $pageSize
""")
List<UserFollowRecommendationDTO> getUsersRecommendationBySimilarNetwork(String username, int pageSize, int pageNumber);
```

Graph DB Design

Similar user recommendation

Domain Specific: Suggest new users to follow based on similar tastes, based on liked games.

Graph-centric:

1. Starting from *User A* node,
2. Identify second-degree relationships by following the *FOLLOWS* edges from A's direct connections.
3. Find all game nodes these users *LIKE* and compare them with game nodes liked by *User A*.
4. Return the users who share at least one liked game with User A, sorted by the number of shared liked games in descending order.

```
@Query("""
        MATCH (user:User {username: $username})-[:FOLLOWS]->(:User)-[:FOLLOWS]->(suggestedUser:User)
        WHERE NOT (user)-[:FOLLOWS]->(suggestedUser) AND suggestedUser <> user
        MATCH (user)-[:LIKES]->(commonGame:Game)<-[:LIKES]-(suggestedUser)
        WITH suggestedUser, COUNT(commonGame) AS sharedGames
        WHERE sharedGames > 0
        RETURN
          suggestedUser.username AS username,
          sharedGames
        ORDER BY sharedGames DESC
        SKIP $pageSize * ($pageNumber - 1)
        LIMIT $pageSize
""")
List<UserTastesSuggestionDTO> getUsersRecommendationBySimilarTastes(@Param("username") String username, @Param("pageSize") int pageSize, @Param("pageNumber") int pageNumber);
```

Graph DB Design

Board games recommendation

Domain Specific: Board games recommendation based on the user's favorite board games and categories

Graph-centric:

1. Starting from *User A* node,
2. Identify all followed users via the *FOLLOWS* relationship.
3. Retrieve the game nodes liked by these users that User A has not liked.
4. Then, gather the categories of both the suggested games and the games liked by User A.
5. Count the matching categories and return games with at least one shared category, sorted by the number of common categories in descending order.

```
@Query("""
    MATCH (u:User {username: $username})-[:FOLLOWS]->(follower:User)-[:LIKES]->(g:Game)
        WHERE NOT (u)-[:LIKES]->(g)
        WITH DISTINCT g, u
        // Compute common categories
        MATCH (u)-[:LIKES]->(likedGame:Game)
        UNWIND likedGame.categories AS userCategory
        UNWIND g.categories AS gameCategory
        WITH g, userCategory, gameCategory
        WHERE userCategory = gameCategory
        WITH g, COUNT(userCategory) AS commonCategories
        // Order and return results
        WHERE commonCategories > 0
        RETURN
          g.name AS name,
          g._id AS id,
          g.yearReleased AS yearReleased,
          g.shortDescription AS shortDescription,
          commonCategories
        ORDER BY commonCategories DESC
    SKIP $pageSize * ($pageNumber - 1)
    LIMIT $pageSize
""")
List<GameSuggestionDTO> getGamesRecommendation(String username, int pageSize, int pageNumber);
```

Graph DB Design

Tournaments recommendation

Domain Specific: Tournament recommendation based on the favorite games of the user and tournaments that the user's friends are participating in.

Graph-centric:

1. Starting from *User A* node,
2. Identify Game nodes related to tournaments *User A* has played.
3. Identify tournaments related to those Games, filtering for public, upcoming, and non-full ones where A is not already a participant.
5. Rank them by the number of A's followers participating, prioritizing earlier start times in case of a tie.

```
@Query("""
    MATCH (user:User {username: $username})-[:PARTICIPATES]->(userTournament:Tournament)-[:IS_RELATED_TO]->(game:Game)
    MATCH (game)<-[:IS_RELATED_TO]-(suggestedTournament:Tournament)
    WHERE NOT (user)-[:PARTICIPATES]->(suggestedTournament)
      AND suggestedTournament.startingTime > datetime()
      AND suggestedTournament.visibility = 'PUBLIC'
      AND NOT EXISTS {
        MATCH (suggestedTournament)<-[:PARTICIPATES]-(:User)
        WITH COUNT(*) AS participantCount
        WHERE participantCount >= suggestedTournament.maxParticipants
      }
    MATCH (user)-[:FOLLOWS]->(follower:User)-[:PARTICIPATES]->(suggestedTournament)
    WITH suggestedTournament, COUNT(DISTINCT follower) AS numParticipantFollowers
    MATCH (suggestedTournament)<-[:PARTICIPATES]-(participant:User)
    WITH suggestedTournament, numParticipantFollowers, COUNT(DISTINCT participant) AS numParticipants
    MATCH (suggestedTournament)-[:IS_RELATED_TO]->(relatedGame:Game)
    WITH suggestedTournament, numParticipantFollowers, numParticipants,
         relatedGame._id AS gameId,
         relatedGame.name AS gameName
    RETURN DISTINCT
      suggestedTournament._id AS id,
      suggestedTournament.name AS name,
      suggestedTournament.startingTime AS startingTime,
      suggestedTournament.maxParticipants AS maxParticipants,
      numParticipants,
      numParticipantFollowers,
      { id: gameId, name: gameName } AS game
    ORDER BY numParticipantFollowers DESC, suggestedTournament.startingTime ASC
    SKIP $pageSize * ($pageNumber - 1)
    LIMIT $pageSize
""")
List<TournamentSuggestionDTO> getTournamentsRecommendation(String username, int pageSize, int pageNumber);
```

Neo4j Indexes

Indexes choice

All Neo4j queries starts from a certain node (*User*, *Game*, *Tournament*) looking for it using the specific identifier which is a property of the node. In particular, we have:

- User: *username* property
- Game, Tournament: *_id* property

As a result, we decided to set indexes on those nodes properties.

Performance Tests using Postman

- Execution of most relevant queries (*suggestions*).
- Tests repeated 5 times clearing query cache between each request.

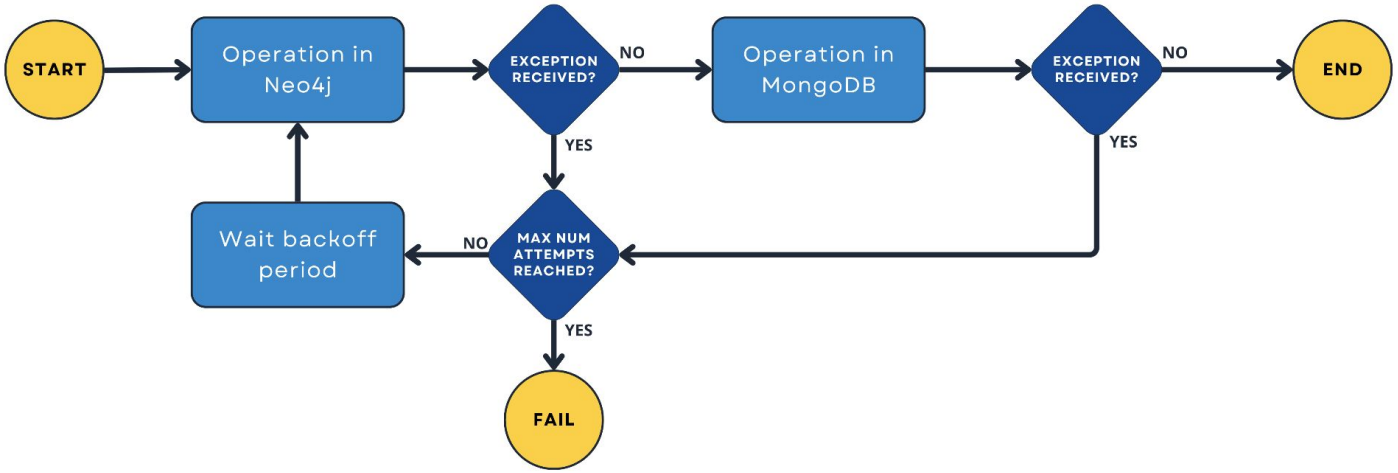
Before indexing	2376ms	8210ms	2900ms	1506ms
	-32.2% ↓	**-53.3%** ↓	-10.7% ↓	-25.4% ↓
After indexing	1611ms	3830ms	2590ms	1123ms
Query	**User suggestion based on follows**	**User suggestion based on tastes**	**Board games Recommendation**	**Tournament Recommendation**

Handling Intra-DB Consistency

Execution flow

Accordingly to non-functional requirements, we must ensure high availability and eventual consistency, ensuring both databases reach a consistent state at a certain time. The system implements the following mechanism:

1. Operation is firstly executed in **Neo4j**, which has a single replica node.
2. If successful, the operation is propagated to **MongoDB**, which has three replica minimizing the risk of failure.
3. If either step fails, a **retry mechanism** is triggered up to three times before returning an error.



Handling Intra-DB Consistency

Work Hypothesis and Drawbacks

The system relies on the hypothesis that MongoDB cannot fail on connection, having 3 replicas. Neo4j instead, is better prone to fail, having only one replica node.

The ***@Retryable*** mechanism allow to easily manage temporary connection issues.

A rollback mechanism, instead, is not implemented: a temporary MongoDB fail would not lead to problems, because Neo4j operations are idempotent.

However, if MongoDB fails, this would lead to a DB inconsistency, but this risk is really low for the previously observed reasons.

```
@Retryable(
        retryFor = {DataAccessException.class, TransactionSystemException.class},
        maxAttempts = 3,
        backoff = @Backoff(delay = 2000)
)
public String registerUser(UserRegDTO signUpRequest) {
```

Swagger UI REST APIs documentation

http://localhost:8080/swagger-ui/index.html

Admin Admin operations

POST	/api/admin/games	Add a new game
DELETE	/api/admin/games/{id}	Delete a game
PATCH	/api/admin/games/{id}	Update a game
GET	/api/admin/analytics/usersByLocation	Get number of users by country
GET	/api/admin/analytics/monthlyRegistrations	Get monthly registrations by year
GET	/api/admin/analytics/bestGamesByAge	Get best games by age
DELETE	/api/admin/users/{username}	Delete a user

User Operations related to user management

POST	/api/users/{username}/follow	Start following a user
DELETE	/api/users/{username}/follow	Stop following a user
GET	/api/users/myProfile	Get user profile
DELETE	/api/users/myProfile	Delete user profile
PATCH	/api/users/myProfile	Update user profile
GET	/api/users/{username}	Get user info
GET	/api/users/{username}/wonTournaments	Get list of won tournaments by a user
GET	/api/users/{username}/participatedTournaments	Get list of participated tournaments by a user
GET	/api/users/{username}/organizedTournaments	Get list of organized tournaments by a user
GET	/api/users/{username}/likedGames	Get games liked by a user
GET	/api/users/{username}/following	Get users followed by a user
GET	/api/users/{username}/followers	Get followers of a user
GET	/api/users/myProfile/reviews	Get user reviews
GET	/api/users/browse	Browse users

Live Demo with Postman