UNIVERSITÁ DI PISA
ENGEGNERIA DELL'INFORMAZIONE

PC Performance Evaluation on games

Project Report

Bruno Augusto Casu Pereira de Sousa

Master's degree Computer Engineering – Computer Systems and Networks
883II 22/23 Large-Scale and Multi-Structured Databases

Pisa, 2023

Contents

1. Introduction **3**

2. Preliminary evaluation and datasets **4**

3. Design **6**

3.1 Actors and Requirements 7

3.2 Use Case diagram 8

3.3 Class Diagram 10

4. Data model and DBMS architecture **13**

4.1 Document DB (MongoDB) 13

4.2 Key-Value DB (Redis) 16

4.3 Replicas and Clustering 17

4.4 Proposing a Sharding scheme 21

5. Implementation and Results **22**

5.1 PC Components (Evaluation Functions) 23

5.2 Reviews and PC games requirements (Browsing Functions) 28

5.3 Statistics (Additional Functions) 34

6. Conclusions **35**

1. Introduction

The application proposed in this project is a platform for PC systems performance evaluation for games, allowing users to submit its personal computer system parts, and receive and evaluation if its system can run a specific game. In general, video games for the PC platform provide a set of components, such as the CPU and GPU models as reference for the minimum requirements to execute the game without performance problems. However, for a regular consumer, comparing PC parts without a specific metric is not an easy task, as the number of available CPUs and GPUs in the market is very large, and doing and objective evaluation if certain hardware is more powerful than other is not simple. Not only that, but the requirements vary a lot from one specific game to another, as different graphics engines and other features may influence the use of computational resources, making the comparison and evaluation of different system configurations not trivial. The objective of the service provided is then comparing benchmarks and performance test results for the components involved (User system and the game Requirements) to obtain an objective comparison between the expected performance, thus resulting in the evaluation if a users' system can run the game based on the metrics evaluated.

In the case of not meeting the requirements, the developed application will suggest some hardware upgrades for the user, based on relevant metrics, including a price range and desired performance. As the components specifications and prices are stored in the platform database system, an additional functionality will be provided for the users, which will allow them to check benchmarks and compare the PC components performance and value. Not only the PC games system requirements will be managed, but to provide a better user experience, the presented platform will also provide some reviews for the games, in case a User just want to browser and check popular games and obtain information if its system can run those games. An additional feature will also allow a User to post a simple review of one or more games during its session in the platform, complementing exiting reviews and metrics that can be seen on the available games listed. This type of system evaluation system combined with the PC games overview is, therefore, useful as often when a new PC game is released many consumers asks questions like "Can I Run It?" or "What do I need to Run It?". Finally, managers will be able to obtain statistics and additional information about which games the Users are searching, as a way to better identify trending games.

To handle the operations, a document, and a key-value database management systems (DBMS) were used, respectively, MongoDB and Redis, and the application modules were designed using Python, with the available libraries for those DBMS. With that the PC games and components information will be stored and managed in the document database (DB) and the user session and searches will be managed by the key-value DB, providing a fast and reliable platform.

2. Preliminary evaluation and datasets

The first step to develop the proposed application was to obtain the datasets that are used on the platform, as well as to analyze its main characteristics in terms of data structures and organization. The evaluation is also fundamental to assess the feasibility of the proposed system as the database will be constructed based on the items, values and other information found in the researched datasets. As the primary source for PC games information, Steam is the largest gaming platform in the world, and it gather a huge collection of available games for purchase while also providing community support for discussions and games evaluations (https://store.steampowered.com/about/). Steam then contains most of the data necessary for the evaluation proposed, as its large game inventory along with the review data are made available.

For the platform proposed, three main sections are necessary to be implemented in the database system, so that it is possible to provide the proper information for the end users:

1. The PC game minimum requirements, defined by a CPU and GPU reference, along with addition descriptions (storage size or memory requirements). For example, the game “Forza Horizon 5”, found on Steam, has a system requirement of an Intel i5-4460 as the CPU, and the NVidia GTX 970 as the GPU.
2. A list of reviews for recent PC games, which must contain a brief description of the User experience with the game, and if that user recommends the game. For example, a review for the game "Dead by Daylight" was posted on the date 2017-11-22 recommending the game. This review also contained the text: “Fun to play with friends”, as a description.
3. The description and benchmark performance of the main PC components relevant for executing a software like a video game. This includes for which platform the component is designed (Desktop, Laptop, etc.), test results showing a metric for the component performance, and the retail price. For example, the Graphics card from NVidia model GeForce GTX 1650 for the Desktop platform has a price of 209 USD with a G3Dmark result (benchmark test) of 7807 points.

For the first two items on the list, the data can be extracted from the Steam platform, as for every game listed, the website provides the system requirements and the User reviews. To obtain then a compilation of the data from Steam some datasets were found in the Kaggle platform (https://www.kaggle.com/). A number of datasets were found, containing a significant amount of information, however, the focus of this platform is not to manage and provide a complex platform for games reviews. To provide a reasonable amount of reviews a smaller and organized dataset was selected, containing over 260.000 reviews for a selection of games on Steam. In addition to this, a dataset was also found containing in a very organized way all the system requirements for the Steam games, where for every item, a CPU and GPU model were referenced, allowing the proposed evaluation platform to be developed.

Aside from the PC games general information and reviews, to obtain the information in the third item of the list, a consistent dataset must be used, and it should provide some known performance metrics for the PC parts, to compare them with the game system requirements. two additional datasets were found also using the Kaggle platform, and they contain lists for many PC parts (CPU and GPU) along with its retail price and benchmarks results. These datasets were fundamental for the project development as the proposed application must have a way to evaluate if a submitted hardware component can meet the minimum requirements to execute the game. The datasets then used for the development of the platform are listed below:

PC video games requirements
Source: *https://www.kaggle.com/datasets/baraazaid/pc-video-game-requirements*
CSV File size: 10MB
Number of documents: 80k

Steam review dataset
Source: *https://www.kaggle.com/datasets/luthfim/steam-reviews-dataset*
CSV File size: 120MB
Number of documents: 260k

CPU benchmark v4
Source: *https://www.kaggle.com/datasets/alanjo/cpu-benchmarks*
CSV File size: 300kB

GPU benchmark v7
Source: https://www.kaggle.com/datasets/alanjo/gpu-benchmarks
CSV 120 kB
CPU+GPU Number of documents: 6k

To organize these datasets in the MongoDB collections, some modifications were done adapting and removing some data fields that were unnecessary. Also, the formatting of some of the fields had to be corrected, as it was making it difficult to quickly find components based on the system requirements, as on the dataset used, the name of the component suppressed the manufacturer name, which caused some discrepancy when querying the components dataset (as in that dataset, the name of the manufacturer was kept). To correct and improve the query description, a custom identifier (UUID) was included to better identify the components included in the DB. With all the needed modifications the final collections were then imported in MongoDB, converting the CSV files in the JSON document format. Figure 1 contains a diagram illustrating the process done for the dataset management:

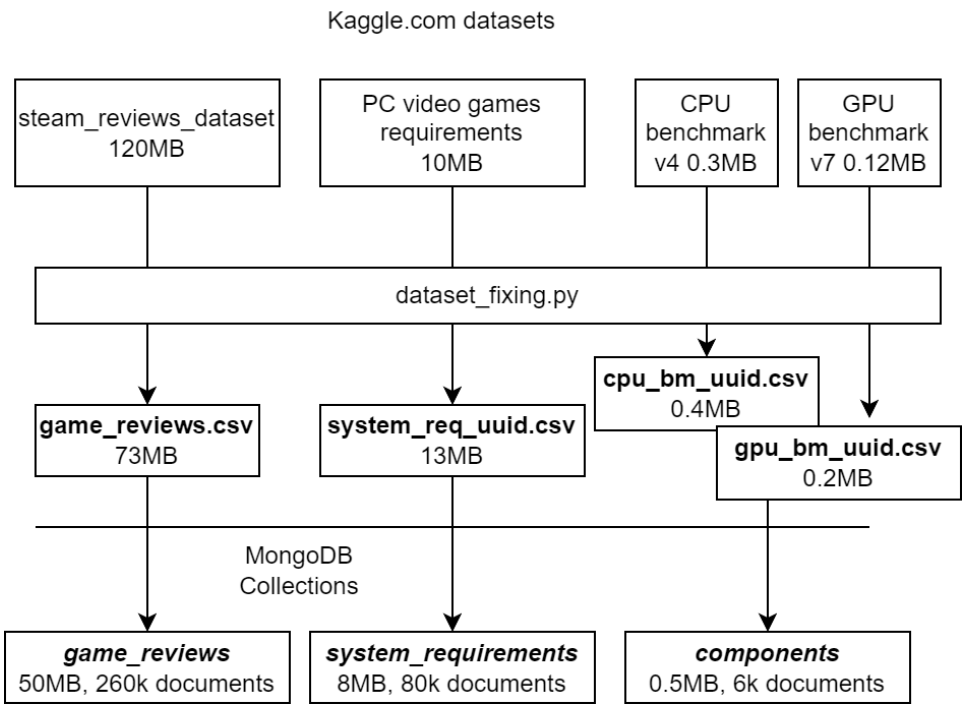


Figure 1 – Diagram of the management of datasets for the platform developed.

3. Design

Staring the design of the platform, it is fundamental to understand the functional and non-functional requirements needed to provide the service. Also, the actors that are involved in the system must also be highlighted, with its respective functions inside the platform operation. Following this analysis, the use cases will be defined using the appropriated diagram, along with the Class diagram, providing an overview of all the modules that will be implemented to handle the system.

Another relevant mention to the overall design of this platform is the evaluation and characterization of the operations and access in the DBMS, considering the CAP theorem (Consistency, Availability, Partition tolerance). In the theory, DBs are then classified depending on the predominant types of operation, as well as the performance indicators that must be implemented in the system, following the functional and non-functional requirements. For the PC games evaluation platform, most of the access in the system are Read operations, as the application works more as a consulting system, where Users go to obtain needed information about their computer systems and its performance in video games.

With that, consistency is not the key performance parameter to be enforced in the platform, as eventually, the access to the requested data will be consistent, even though at some point a User may see a deprecated version of some game or review information due to the distributed DB. For most part, updates should not be recurrent in the

collections, except for the comments and review collection, where then there is more dynamic and fast insertion of new data. It may occur that new comments were written in some portion of the database while a read request was being processed, and the replicas were not updated yet. In that case the Eventual Consistency paradigm would be used, and this issue should not detriment extensively the user experience as, eventually, the User will retrieve the most recent version of the PC game of interest with the proper reviews updated.

Analyzing the other two pillars of the CAP theorem with the platform proposed system, those performance indicators then have significant more relevancy, as low availability may in fact be detrimental to the user experience, as they will not be able to quickly get the information they were looking for. Also, partitioning the large number of reviews, in a way that data loss is minimized in case of failure is also a relevant parameter that must be present in the PC games evaluation platform. Therefore, the system proposed is more oriented to the AP side of the CAP theorem diagram.

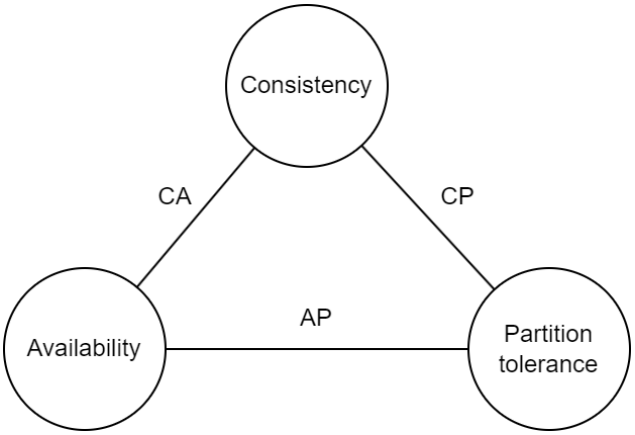


Figure 2 – CAP Theorem diagram.

3.1 Actors and Requirements

Moving to the requirement for the system, as previously presented, the overall objective of the platform is to provide information about PC games and its computational requirements, allowing users to check if its system is able to run a PC game. A fundamental consideration on this platform, is that as the service aims to provide evaluation and hardware information, meaning that there is not a significant importance in maintaining User personal information. In that way the design of the platform considers the User as an anonymous actor, and the information provided by this entity is only kept as a Session information (like tracking cookies in a browser search, for example). In that way, there is no registration in the platform, and any user can access the database resources, and submit its system for a quick evaluation. The same is true for the reviews management, as the goal of the platform is not to serve as a reference for games reviews, meaning that a simpler structure was built to integrate the reviews and games statistics on

the DBMS. Thought they are not considered as the main objective of the platform, providing additional information about the PC games and trending games can be appreciated as most of the target users for the proposed system also have this as a common interest.

Two actors are then defined in the PC games and hardware evaluation platform, **Users** and **Managers**. The functional and non-functional requirements are listed below:

Functional Requirements

Users:

1. View most reviewed PC games.
2. View best reviewed PC games.
3. View latest reviews of a selected PC game.
4. View the System Requirements for a selected PC game.
5. View a selected Component benchmarks and price.
6. View the Components with the highest value metric.
7. Submit its own System Configuration.
8. Submit a simple review of a PC game.
9. Obtain a System Evaluation: verify if its own System Configuration meets the System Requirements for a selected PC game, in terms of CPU and GPU.
10. Obtain a suggestion for a System Upgrade, based on a selected PC game.

Managers:

1. Create/update/delete a PC game information.
2. Create/update/delete a component information.
3. Delete a PC game review.
4. View statistics for Components, based on the categories.

Non-Functional Requirements

1. The platform should provide a simplified way for users to verify if it submitted system is compatible with a particular PC game.
2. The access to the database should have low latency, improving user experience when evaluating its system.
3. The platform should be reliable, avoiding data loss in the case of hardware failure.
4. The access to the reviews and the game requirements should have good availability for improved user experience.

3.2 Use Case diagram

The Use Case diagram the illustrates the overall actions performed by the actors in the system. The diagram has been divided in four sections, to better illustrate the operation in each domain, and the correlation between the actions performed.

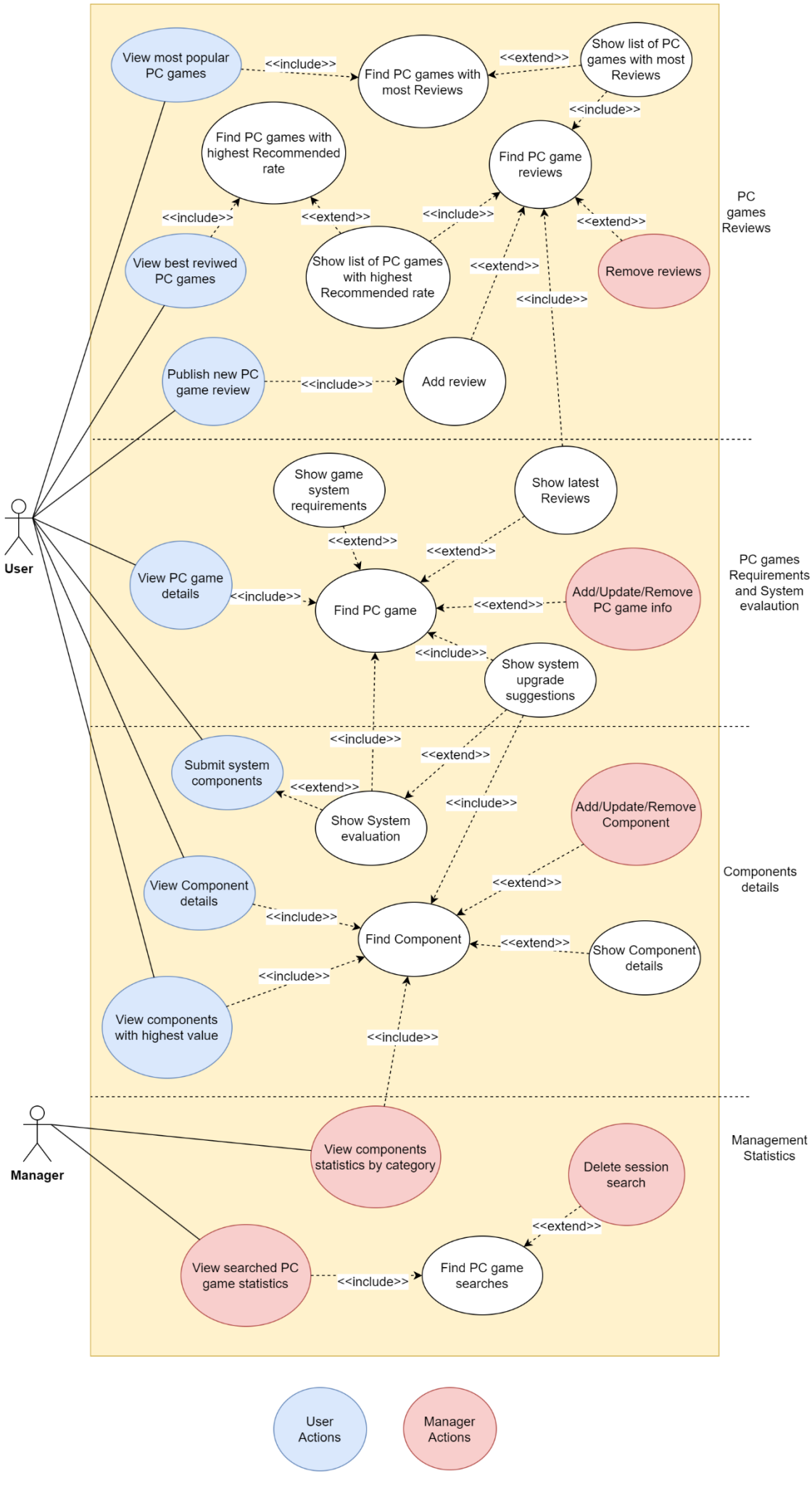

View most popular PC games
<<include>>
Find PC games with most Reviews
<<extend>>
Show list of PC games with most Reviews
<<include>>
Find PC game reviews
Find PC games with highest Recommended rate
<<include>>
<<extend>>
<<include>>
<<extend>>
View best reviwed PC games
Show list of PC games with highest Recommended rate
<<extend>>
<<include>>
Remove reviews
PC games Reviews
Publish new PC game review
<<include>>
Add review
Show game system requirements
Show latest Reviews
<<extend>>
<<extend>>
User
View PC game details
<<include>>
Find PC game
<<extend>>
Add/Update/Remove PC game info
PC games Requirements and System evalaution
<<include>>
Show system upgrade suggestions
<<include>>
<<extend>>
Submit system components
<<extend>>
Show System evaluation
<<include>>
Add/Update/Remove Component
View Component details
<<extend>>
<<include>>
Find Component
<<extend>>
Show Component details
<<include>>
View components with highest value
<<include>>
Components details
Manager
View components statistics by category
Delete session search
<<extend>>
View searched PC game statistics
<<include>>
Find PC game searches
Management Statistics
User Actions
Manager Actions

Figure 3 – Platform Use Case diagram.

3.3 Class Diagram

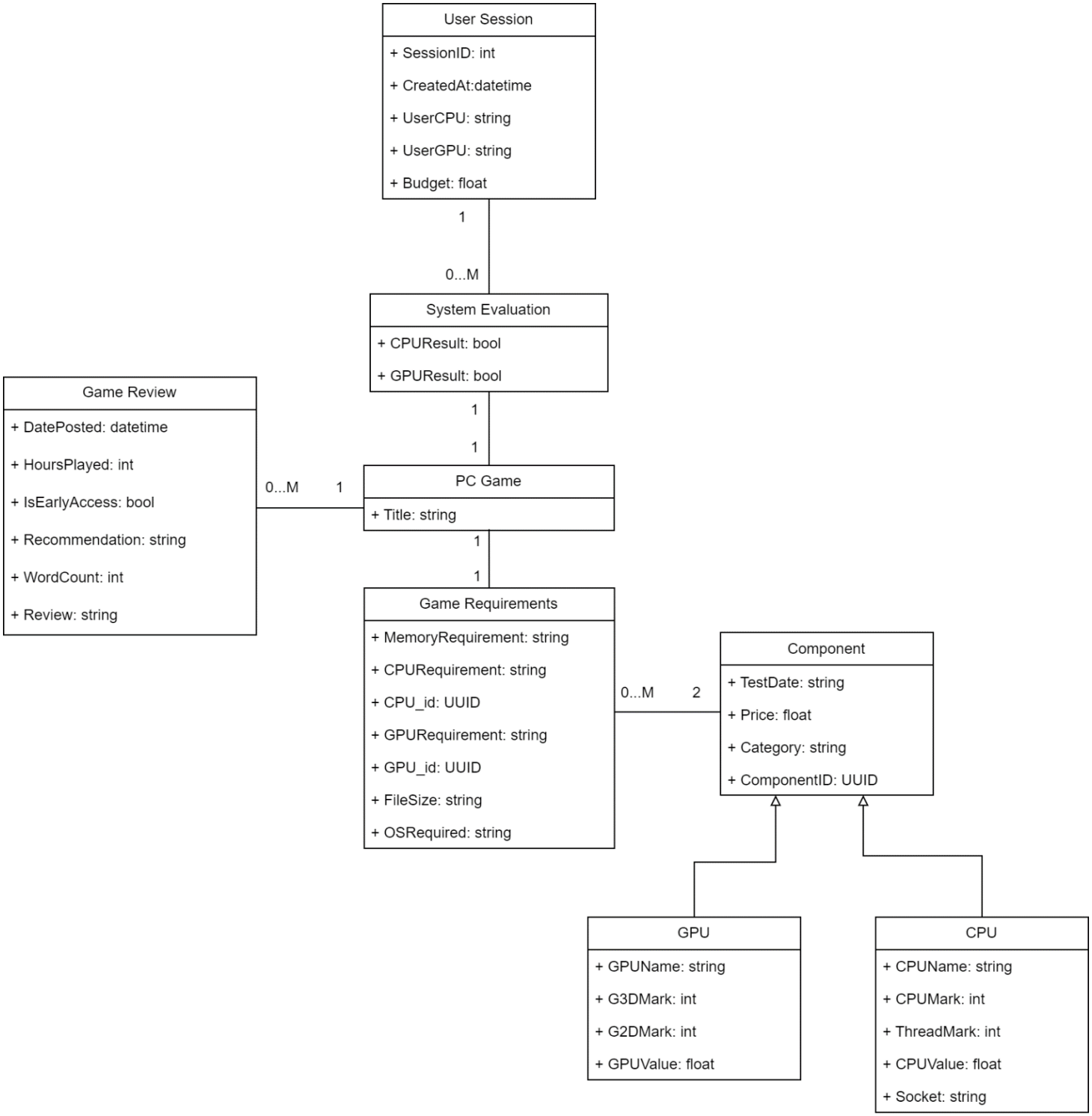


Figure 4 – Class diagram.

A brief description of the relation between entities is:

- A User (Session) have a One-to-Many relation with a system evaluation. Users obtain a system evaluation when submitting their information and searching for a PC game.
- A system evaluation has a One-to-One relation with a PC game. In this application Users then can search for multiple games in one session, thus having multiple evaluations, but each evaluation must be tied to one PC game.
- A PC game has a One-to-One relation with a Game requirement, and have a One-to-Many relation with a Game review, since one game may have multiple reviews.
- A Game requirement have a Many-to-Two relation with a Component, since for every game exactly two components (One CPU and One GPU) are present in its requirements, and those components may be included in multiple system requirements.
- A Component may be either a CPU or a GPU, having specific fields that differentiate between the two hardware.

When implementing the classes in Python, the structure of the classes was adapted to better organize the operations. A complete description of the classes and attributes created is shown:

System Requirements Class		
Attribute	**Type**	**Description**
_id	string	MongoDB id
title	string	Title of the PC game (unique)
memory	string	Memory requirement
file_size	string	Required disk space
os_required	string	Operating system required
cpu	string	CPU model required
gpu	string	GPU model required
cpu_id	string	UUID of the CPU component
gpu_id	string	UUID of the GPU component

Game Review Class		
Attribute	**Type**	**Description**
_id	string	MongoDB id
title	string	Title of the PC game (unique)
is_early_access_review	bool	Review posted in the early access period (true of false)
recommendation	string	If the user recommends the game or not
hour_played	int	Number of hours that the user played the game
word_count	int	Number of words in the review text
review	string	User review text

Component Class		
Attribute	**Type**	**Description**
_id	string	MongoDB id
component_id	string	Custom identifier for components
category	string	Component category (Desktop, Laptop, Mobile, Workstation or Server)
price	float	Retail price in USD ($)
test_date	int	Year when the benchmark test was done
CPU Class		
name	string	CPU model name
cpu_mark	int	CPUMark benchmark score
thread_mark	int	ThreadMark benchmark score
cpu_value	float	Metric that consists in dividing the CPUMark score by the price (references how much performance per dollar the component has)
socket	string	Socket name for the CPU model
cores	int	Number of CPU cores
GPU Class		
name	string	GPU model name
g3d_mark	int	G3DMark benchmark score
g2d_mark	int	G2DMark benchmark score
gpu_value	float	Metric that consists in dividing the G3DMark score by the price

User Session Class		
Attribute	**Type**	**Description**
session_id	string	MongoDB id
created_at	datetime	Title of the PC game (unique)
user_cpu	string	Submitted CPU id
user_gpu	string	Submitted GPU id
budget	float	Maximum value the user is willing to spend on a system upgrade (in USD)

Evaluation Class		
Attribute	**Type**	**Description**
session_id	string	MongoDB id
title	datetime	Title of the PC game (unique)
user_cpu_result	bool	After the comparison with the minimum requirement performance, the result can be true (the CPU can run the game) or false (the CPU can nor run the game)
user_gpu_result	bool	Same as user_cpu_result, but for the GPU model

4. Data model and DBMS architecture

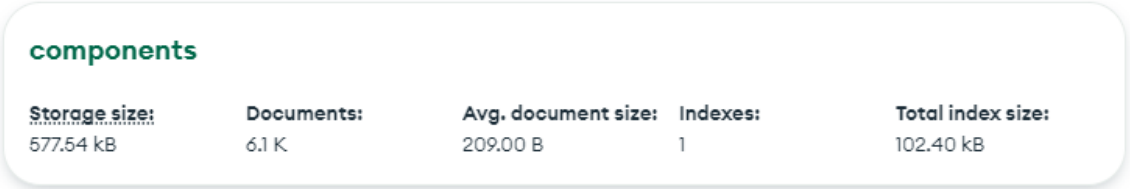
For the application proposed two types of DB systems will be used, a Document DB and a Key-Value DB. The document DB allows the information on the components, games, and reviews to be stored in a readable and consistent way, where the data structures are then stored in JSON format. The NoSQL structure that this type of DB manages data in a flexible and schema-less format, providing an intuitive and developer-friendly approach to data management. They are also optimized for high-performance read and write operations, thanks to their efficient schema design and support for distributed data storage.

Considering that the proposed platform should provide high availability, as discussed in the previous sessions mentioning the CAP theorem, a Key-Value DB was selected to handle the information of the users during a session. In this model, each piece of data (the value) is associated with a unique identifier (the key), optimizing data retrieval as values are directly accessed using their associated keys. Key-value databases are known for their simplicity and efficiency in handling basic data storage and retrieval tasks, which is the main motivation for selecting this type of data management system for the session information.

4.1 Document DB (MongoDB)

For the documents DB organization three collections were designed, based on the datasets found, they are:

1. ***components***



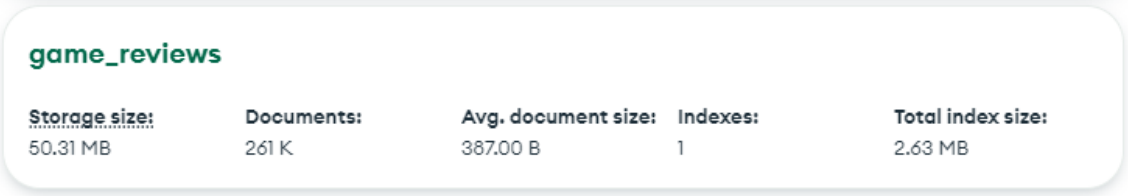
With over 6.000 components imported from the dataset, the documents for the CPUs and GPUs have the following structure, respectively:


```
{
  "_id": {
    "$oid": "64b32f37c7c0483e58ed4834"
  },
  "cpuName": "AMD EPYC 7763",
  "price": 7299.99,
  "cpuMark": 88338,
  "cpuValue": 12.1,
  "threadMark": 2635,
```

```
  "cores": 64,
  "testDate": 2021,
  "socket": "SP3",
  "category": "Server",
  "componentID": "2f35cddf-10a7-48eb-afa5-d2716d4fa113"
}

{
  "_id": {
    "$oid": "64b32f65c7c0483e58ed5745"
  },
  "gpuName": "GeForce GTX 1080 Ti",
  "G3Dmark": 18284,
  "G2Dmark": 934,
  "price": 604.15,
  "gpuValue": 30.26,
  "testDate": 2017,
  "category": "Desktop",
  "componentID": "dc07dd3b-125a-4b19-9f7c-9417329d58e8"
}
```


2. ***game_reviews***

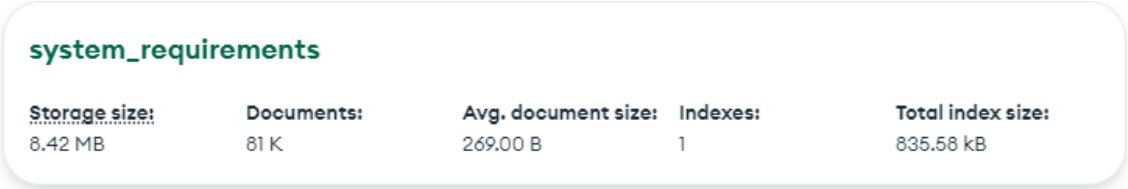


The game reviews were extracted from the Steam database, and over 260.000 reviews were included, covering over 40 different games. The general document structure for this collection is as the example:


```
{
  "_id": {
    "$oid": "64b481baa2900f566a40140a"
  },
  "date_posted": {
    "$date": "2017-09-12T00:00:00.000Z"
  },
  "hour_played": 474,
  "is_early_access_review": false,
  "recommendation": "Recommended",
  "title": "Dead by Daylight",
```

```
  "word_count": 3,
  "new_review": "fun to play  "
}
```


3. ***system_requirements***




```
{
  "_id": {
    "$oid": "64b475f4a2900f566a3ed8da"
  },
  "Memory": " 4 GB",
  "GPU": "Intel HD 4000",
  "CPU": "Intel Core2 Duo E8400  or Athlon 200GE",
  "File Size": "23 GB",
  "OS": "Windows 7 64-bit",
  "title": "Valorant",
  "CPU_id": "f56d95f9-c561-4880-b414-394354cf6332",
  "GPU_id": "c70e31fd-687c-4901-9968-3b9d3c574e9a"
}
```


One consideration on the system requirement's structure is that, by observing the read patterns to obtain the PC game information, it was noticed that embedding the latest reviews on the document for the requirements would greatly improve the speed of which the data is retrieve and displayed for the user, as accessing the reviews collection were consuming some time. In this way, the latest 20 reviews were embedded in the system_requirement documents, allowing for a much faster data acquisition when a user searches for a PC game and would like to check the latest reviews. Though this process does speed the data display (making higher availability for the user), since the database will be distributed, it might occur that at some point the reviews shown to a user are not consistent, as the replicas are still being updated. However, this should not disrupt the user experience, as the priority of this application is to provide good availability, relying on eventual consistency. The document structure for the reviewed games should be then:


```
{
  "_id": {
    "$oid": "64b475f4a2900f566a3ed8e7"
  },
  "Memory": " 8 GB",
```

```
  "GPU": "GeForce GTX 460",
  "CPU": "Intel Core i3-4170",
  "File Size": "25 GB",
  "OS": " 64-bit Operating Systems (Windows 7,Windows 8.1)",
  "title": "Dead by Daylight",
  "CPU_id": "2deb053d-3aa7-4254-baf4-8e0bdb4365f2",
  "GPU_id": "3ef6ac00-98f1-48b5-b82b-a9cace4c60dc"
  "reviews":
        {
            "_id": {
                "$oid": "64b481baa2900f566a401408"
            },
            "date_posted": {
                "$date": "2017-11-22T00:00:00.000Z"
            },
            "hour_played": 654,
            "is_early_access_review": false,
            "recommendation": "Recommended",
            "title": "Dead by Daylight",
            "word_count": 5,
            "new_review": "Fun  to play with friends"
        },
        {
            "_id": {
                "$oid": "64b481baa2900f566a40140d"
            },
            "date_posted": {
                "$date": "2018-09-29T00:00:00.000Z"
            },
            "hour_played": 12,
            "is_early_access_review": false,
            "recommendation": "Recommended",
            "title": "Dead by Daylight",
            "word_count": 1,
            "new_review": "good"
        }, ...
}
```


4.2 Key-Value DB (Redis)

For the Keys used in this DB, the fields from the user session and evaluation classes must be handled. In order to do so, the key space designed for this DBMS uses key names as the following:

```
<session:123456:created_at>  {datetime}
<session:123456:user_cpu>  {string}
<session:123456:user_gpu>  {string}
<session:123456:user_budget>  {float}
```

Where the numbers following ‘session’ (123456) reference the Session ID provided to the User. The ‘created_at’ string key then holds the time when the session was created, the ‘user_cpu’ and ‘user_gpu’ keys holds the model name for each component submitted by the user and the ‘user_budget’ key holds the maximum value the user is willing to spend on a new component (used in the suggestion search).

```
<session:123456:evaluation:1:title>  {string}
<session:123456:evaluation:1:cpu_result>  {bool}
<session:123456:evaluation:1:gpu_result>  {bool}

<session:123456:evaluation:2:title>  {string}
<session:123456:evaluation:2:cpu_result>  {bool}
<session:123456:evaluation:2:gpu_result>  {bool}
```

For the evaluation, the number following ‘evaluation’ string (1, 2) reference the number of the searches performed, counting for every game that the user has searched for an evaluation (within the same session a user may perform multiple evaluation for different games.

By using this structure, whenever a user starts a session, it may choose to submit its system configuration, by writing the model of its CPU and GPU, as well as its overall budget for a possible upgrade. In the case of not adding such information, the keys referencing these parameters would not be used, saving a large amount of storage space, as additional sessions data is not included when a user just want to search PC games specifications.

In the case of a user deciding to submit its own hardware, to obtain an evaluation, a new set of data structures will be created to store which game the evaluation should be done, and the results based on the comparison of performance of both components.

4.3 Replicas and Clustering

The proposed system will then utilize the clustering capabilities available by MongoDB, as a way to distribute the DB, providing increased reliability. The distribution of the nodes in MongoDB will use three replicas for the data storage configured in a MongoDB Replica Set. For the demonstration of the platform, the replicas will be

deployed in the local machine (one machine) by configuring multiple local servers using different connection ports. A table of the replicas used in the cluster is provided:

Replica ID	Localhost PORT	Priority Config.
2	27019	5
1	27018	3
0	27017	1

To create such distribution of nodes in the local environment, a command was executed in the primary replica (hosted in PORT 27019), enabling the cluster creation with the identification "`rs0`". The command followed the structure:

```
rsconf = {
     _id: "rs0", members: [
          {_id: 0, host: "localhost:27017", priority:1},
          {_id: 1, host: "localhost:27018", priority:3},
          {_id: 2, host: "localhost:27019", priority:5}]
};
```

Once the local server is deployed, by connecting to MongoDB Compass in the primary node, it is possible to verify the configuration developed:

A more detailed description of the Cluster developed can be obtained by running the rs.status() command in the mongo shell:

```
members: [
  {
    _id: 0,
    name: 'localhost:27017',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 1734,
    optime: { ts: Timestamp({ t: 1693344504, i: 1 }), t: Long("2") },
    optimeDate: ISODate("2023-08-29T21:28:24.000Z"),
    lastAppliedWallTime: ISODate("2023-08-29T21:28:24.625Z"),
    lastDurableWallTime: ISODate("2023-08-29T21:28:24.625Z"),
    syncSourceHost: 'localhost:27019',
    syncSourceId: 2,
    infoMessage: '',
    configVersion: 4,
    configTerm: 2,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: 'localhost:27018',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 174,
    optime: { ts: Timestamp({ t: 1693344504, i: 1 }), t: Long("2") },
    optimeDurable: { ts: Timestamp({ t: 1693344504, i: 1 }), t: Long("2") },
    optimeDate: ISODate("2023-08-29T21:28:24.000Z"),
    optimeDurableDate: ISODate("2023-08-29T21:28:24.000Z"),
    lastAppliedWallTime: ISODate("2023-08-29T21:28:24.625Z"),
    lastDurableWallTime: ISODate("2023-08-29T21:28:24.625Z"),
    lastHeartbeat: ISODate("2023-08-29T21:28:27.675Z"),
    lastHeartbeatRecv: ISODate("2023-08-29T21:28:26.664Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: 'localhost:27017',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 4,
    configTerm: 2
  },
  {
    _id: 2,
    name: 'localhost:27019',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 174,
    optime: { ts: Timestamp({ t: 1693344504, i: 1 }), t: Long("2") },
    optimeDurable: { ts: Timestamp({ t: 1693344504, i: 1 }), t: Long("2") },
    optimeDate: ISODate("2023-08-29T21:28:24.000Z"),
    optimeDurableDate: ISODate("2023-08-29T21:28:24.000Z"),
    lastAppliedWallTime: ISODate("2023-08-29T21:28:24.625Z"),
    lastDurableWallTime: ISODate("2023-08-29T21:28:24.625Z"),
    lastHeartbeat: ISODate("2023-08-29T21:28:27.675Z"),
    lastHeartbeatRecv: ISODate("2023-08-29T21:28:26.664Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1693344354, i: 1 }),
    electionDate: ISODate("2023-08-29T21:25:54.000Z"),
    configVersion: 4,
    configTerm: 2
```

The overall architecture of the database system developed for the application follows the structure illustrated in Figure 5. Also, by exploiting the MongoDB

capabilities, the system can handle faults in the nodes, as the election process selects and establishes a new Primary replica, as illustrated in Figure 6.

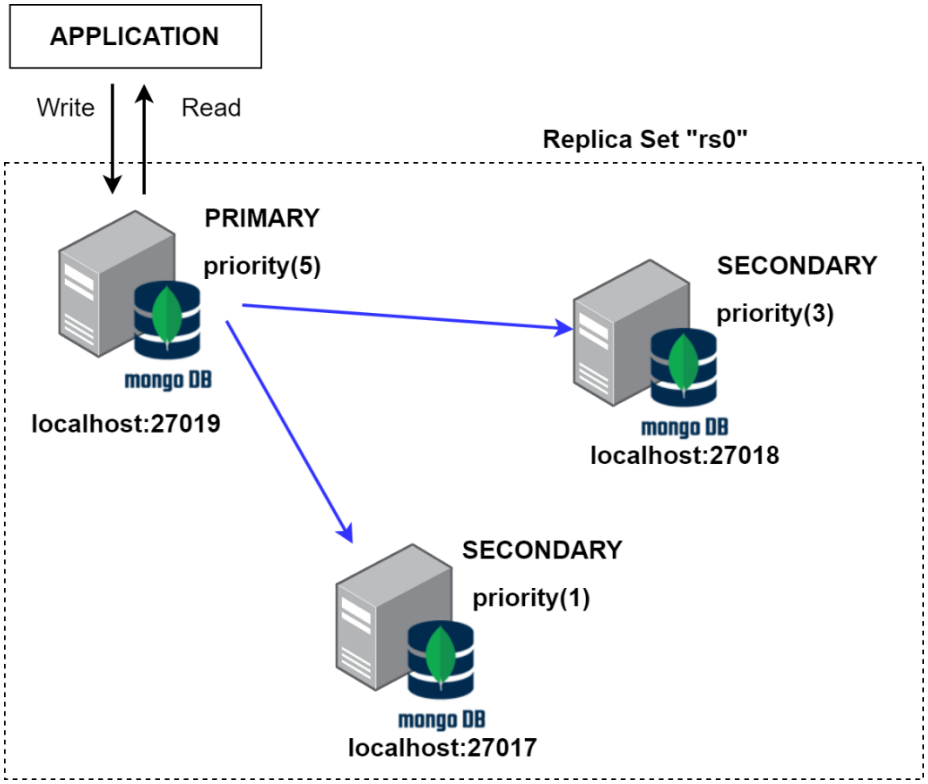


Figure 5 – Distributed DBMS architecture.

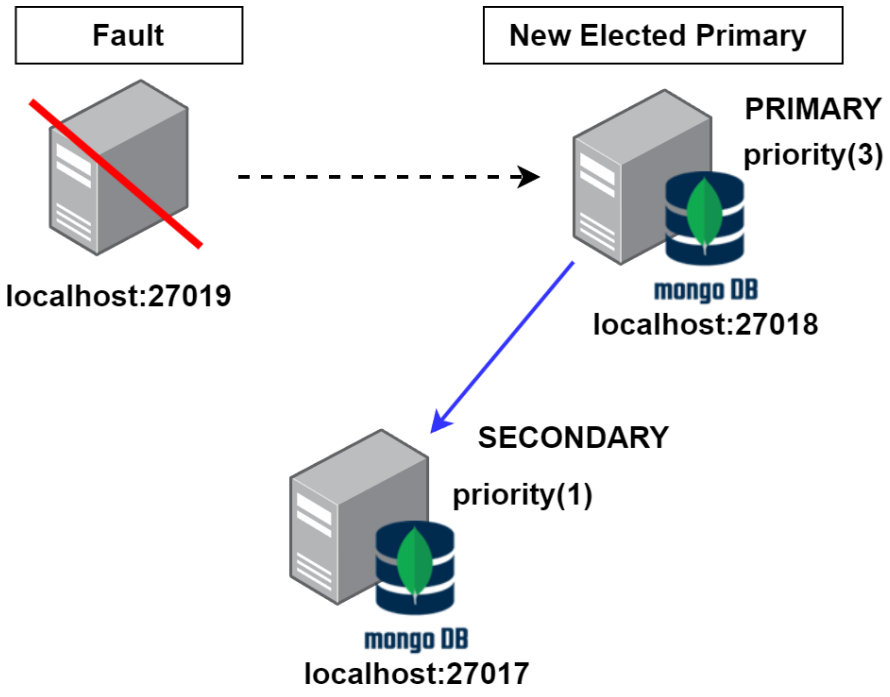


Figure 6 – Fault handling in the Distributed DBMS architecture.

4.4 Proposing a Sharding scheme

Sharding in MongoDB is a technique used to distribute data across multiple servers or nodes to improve performance and scalability. By using this method, it is possible to handle massive amounts of data and high workloads by horizontally partitioning data across multiple machines (shards). For the collections and data structures used in the application, Sharding would be particular useful for the reviews dataset, as most of the volume in the DB is made of these types of documents. The data in the reviews collection would be divided into chunks based on a shard key, created by a Hash function (this ensures a uniform and balanced distribution). The shard key is a field or set of fields in the documents, and it determines which shard the data will be stored on. For this reason, the _id field generated by MongoDB for the reviews documents would then be the selected parameter to be used by the hash algorithm for the shard key generation.

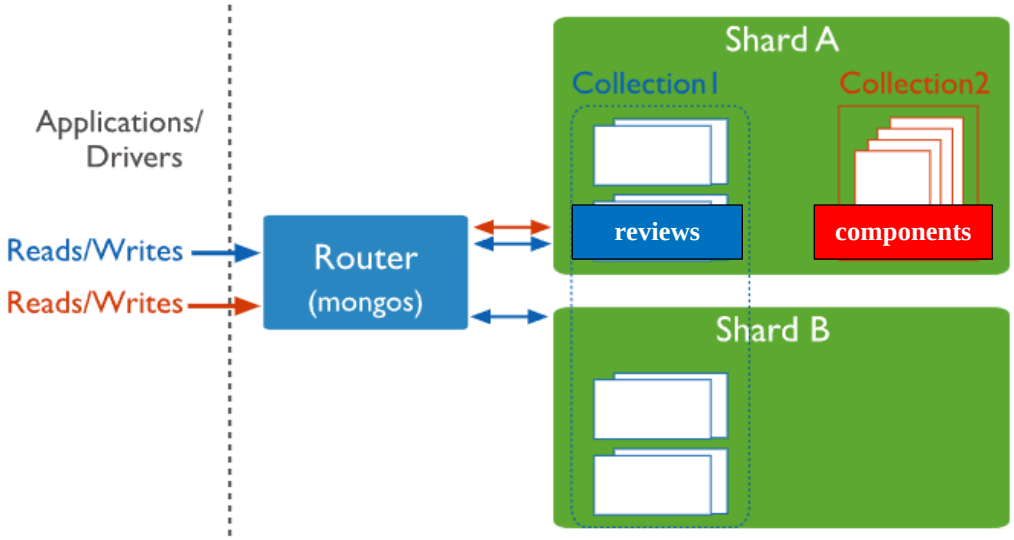


Figure 7 – Diagram of the sharding scheme used in MongoDB cluster (obtained from https://www.mongodb.com/docs/manual/sharding/)

By distributing data and queries across multiple servers, sharding can improve read and write performance for the reviews management. However, it may introduce more complexity in the queries and in the overall architecture of the system. Therefore, the alternative to better handle the volume of reviews will be to embed some of the latest reviews in the system requirements documents.

5. Implementation and Results

The implementation of the proposed platform provides several functions to handle the database access and provide the application functionalities, based on the Functional requirements listed previously. However, no graphical interface was designed, meaning that the project presented provides only the basic structure for the hardware and PC games evaluation. The section will then focus on presenting the main functions implemented as well as the pipelines developed for the access in the MongoDB collections. Also, the methods used for obtaining the components suggestion will be highlighted, as it is one of the main functionalities of the designed system.

For the overall software design some python files were created and divided following the structure illustrated:

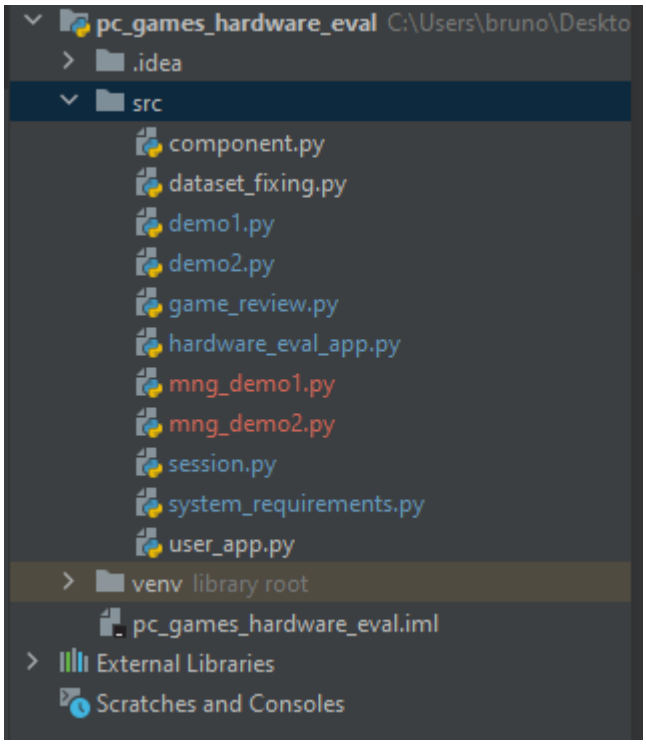


Figure 7 – IntellijIDEA file organization

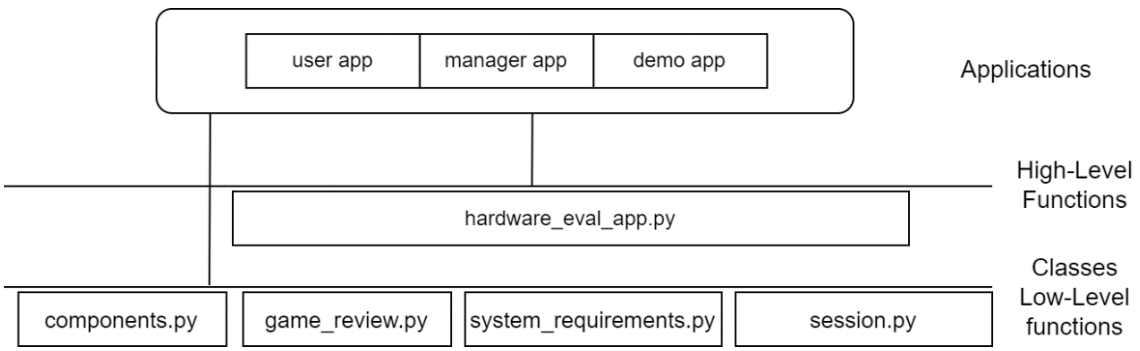


Figure 8 – Diagram of the software architecture

For a better comprehensive demonstration of the platform capabilities, the main functions were divided into three categories, which will be detailed in the next sections. For the CRUD operations, the classes and low-level function included methods for adding, updating, and deleting components and system requirements. Also, for the reviews, the creation of a new document was allowed for the user application, and the proper delete function was designed for the managers (updating a review was not considered, as it is not a usual function).

5.1 PC Components (Evaluation Functions)

The first section will provide the PC components and hardware evaluation functions, allowing the User to obtain relevant information about CPUs and GPUs, as well as to compare its system with the minimum requirements for a selected game.

The statistics function on the listed components performance were then used to show the User PC components that have superior value, when evaluating price and performance. The following captures shows examples of the retuned values when searching for components information:

```
->SHOW Best CPUs (Value Metric), category: Laptop
Intel Core i5-12500H - Value: 75.66 CPU Mark: 23530 Price: 311 USD
Intel Core i5-9400F @ 2.90GHz - Value: 73.42 CPU Mark: 9544 Price: 129.99 USD
Intel Core i5-11500H @ 2.90GHz - Value: 65.83 CPU Mark: 16458 Price: 250 USD
Intel Core i5-11400H @ 2.70GHz - Value: 64.29 CPU Mark: 16073 Price: 250 USD
Intel Core i5-12450H - Value: 63.73 CPU Mark: 19820 Price: 311 USD
```

Figure 9 – Results for best CPUs listed for the Laptop category.

As the general information for components is displayed and made available for the User, this actor may also initiate an Evaluation process. This was implemented by allowing the User to submit two components, a CPU model and a GPU models, along with how much are they willing to spend on an upgrade. This information is then stored in the Key-Value database, as a session data. The Session then contains a timestamp indication its creation time, and the User information, and if the User has requested and evaluation for a PC game, the title of the game and the result for each component are store in the DB. The following captures the illustrate the key structures and values stored in Redis.

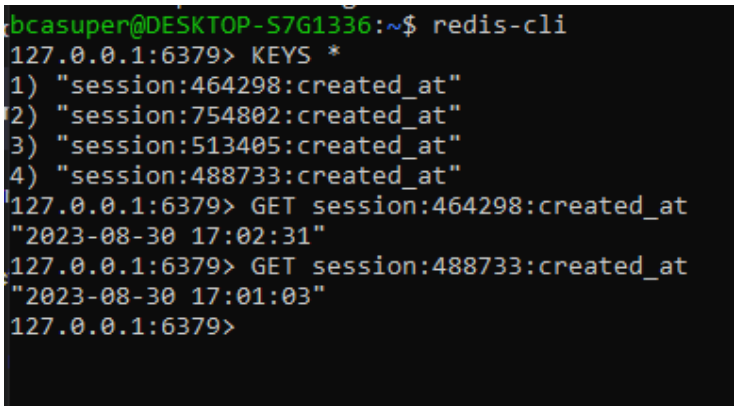


Figure 10 – Capture from the Redis terminal, with the returned values for two sessions.

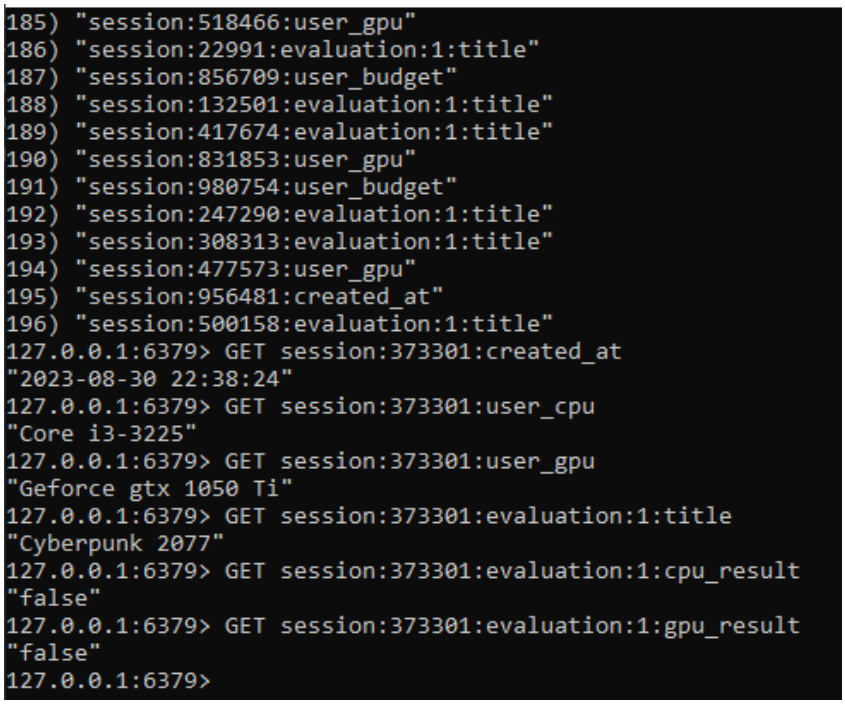


Figure 11 – Capture from the Redis terminal, with the returned values for one session containing the User submitted system components, budget, and the result of the evaluation for the game "Cyberpunk 2077".

Once the User information is available in the KV database, the data can be easily accessed for the Evaluation process. The algorithm for this process is then illustrated in Figure 12. With that, the application will use the User system information to find the components and compare them with the selected game System Requirements. To do this comparison the Benchmarks metrics for each component will be used as a base value, and if the value for the User component is lower than the required to run the game, an additional process will be initiated.

This secondary procedure is done to provide a suggestion for the User allowing them to see components that can run the game (have more computational power than the game system requirement). Figure 13 illustrates the overall selection process for a suggestion. Noticeably, the application will sort the component that meets the

requirements and will provide as the best suggestion components with the highest value. Additionally, a high-performance suggestion was included in the process, which will show the User a component that is not restricted by the budget defined, displaying the best performing PC component listed in the DB.

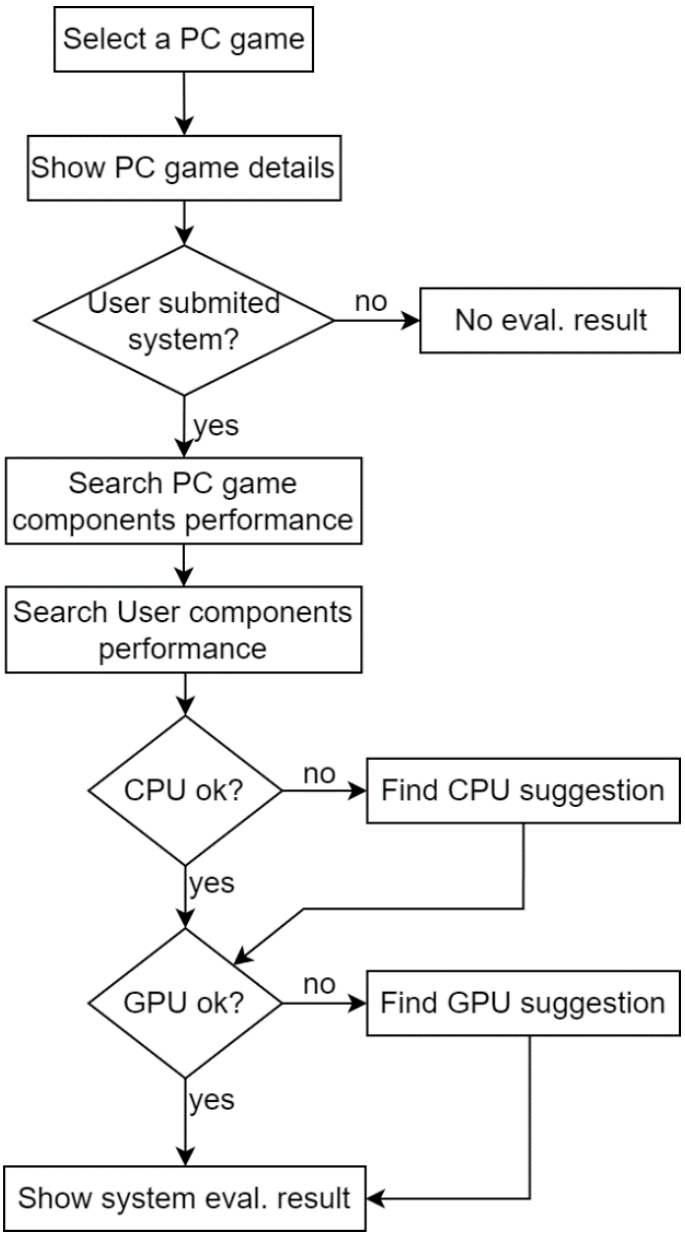


Figure 12 – Algorithm used for the Evaluation process.

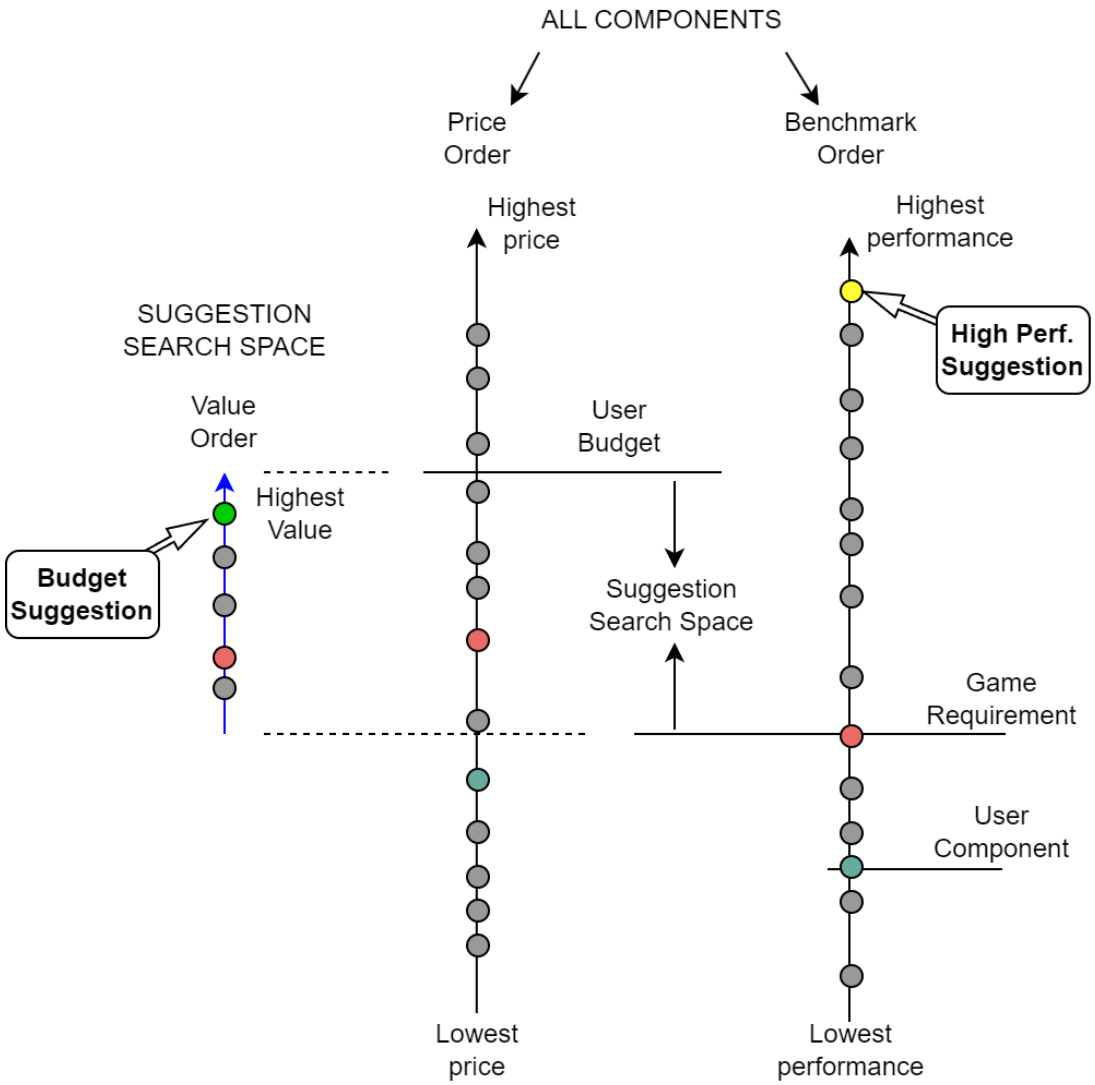


Figure 13 – Diagram of the suggestion component selection.

With all the Evaluation procedures completed the application shows the User the results, if any of the submitted components failed the comparison with the selected game requirements. An example of a submission where the GPU does not meet the requirements is shown in Figure 14, along with the components suggested for the upgrade:

```
->EVALUATION: User CPU is OK for the Game Requirements!

->EVALUATION: User GPU does NOT meet the Game Requirements!

->GPU Suggestion Found!
$$ GeForce RTX 3060
     -Performance increment: 168.6 %
     -G3Dmark: 16958
     -Price: 329 USD
     -Category: Desktop
$$ Radeon RX 6600 XT
     -Performance increment: 151.1 %
     -G3Dmark: 15853
     -Price: 399.99 USD
     -Category: Desktop

->High Performance GPU Suggestion Found!!
$$$$ GeForce RTX 3080 Ti
     -Performance increment: 325.8 %
     -G3Dmark: 26887
     -Price: 1199.99 USD
     -Category: Desktop
```

Figure 14 – Capture of the Evaluation results provided by the platform.

The code structure for the process then uses a pipeline to filter and sort the component based on the benchmark's metrics, price, and value. The designed pipeline for MongoDB is illustrated in Figure 15.

```
budget_pipeline = [
    {
        "$match": {
            "category": {'$regex': user_component_category, '$options': 'i'},
            "cpuValue": {"$exists": "true"},
            "price": {"$lte": user_budget}, "cpuMark": {"$gt": self.cpu.cpu_mark}
        }
    },
    {
        "$sort": {
            "cpuValue": -1
        }
    },
    {
        "$limit": 2
    },
]
```

Figure 15 – Pipeline used for the component suggestion (limited by the user budget).

5.2 Reviews and PC games requirements (Browsing Functions)

The second first section will provide the main functions for the PC games information, including the requirements and reviews. This section, then implement the functions for proving the analytics and best games listing, as it uses the review databases to obtain information on the listed games and detect which are the most relevant. This is then used in the platform as a complement to provide better user experience when browsing PC games specifications.

```
->SHOW Top 3 Most Reviewed Games
PLAYERUNKNOWN'S BATTLEGROUNDS - Number of Reviews: 87374
Grand Theft Auto V - Number of Reviews: 60120
Rust - Number of Reviews: 42736

->SHOW Top 3 Best Rated Games
Game: RESIDENT EVIL 2 / BIOHAZARD RE:2
Total Reviews: 862
Recommended Reviews: 858
Percentage of Recommended Reviews: 99.54%

Game: ASTRONEER
Total Reviews: 1612
Recommended Reviews: 1560
Percentage of Recommended Reviews: 96.77%

Game: Rocket League
Total Reviews: 40443
Recommended Reviews: 36718
Percentage of Recommended Reviews: 90.79%
```

Figure 16 – Results for the reviews analytics functions.

One of the main analytics provided is the games with the most number of Reviews in the platform. This metric is then used to show user possible trending games that have being released. Also, a second analytic listing for the games is based on games with the highest percentage of recommendations. In the documents stored in the game_review collections, there are one specific filed where the user's device if they want to recommend the game or not. By using the pipeline structure provided by MongoDB, it is possible to count the number of reviews for all the games listed, as well as to design a query that will aggregate reviews for every game listed and will calculate what is the percentage of those reviews that are recommending the game. In Figure 16, the results when executing both analytics functions is shown and in Figure 17 the pipeline design for the highest recommended games is shown.

```
pipeline = [
    {
        "$group": {
            "_id": "$title",
            "total_reviews": {"$sum": 1},
            "recommended_reviews": {
                "$sum": {"$cond": [{"$eq": ["$recommendation", "Recommended"]}, 1, 0]}
            }
        }
    },
    {
        "$match": {
            "total_reviews": {'$gte': MIN_REVIEWS},
        }
    },
    {
        "$addFields": {
            "recommended_percentage": {"$multiply": [{"$divide": ["$recommended_reviews", "$total_reviews"]}, 100]}
        }
    },
    {
        "$sort": {"recommended_percentage": -1}
    },
    {
        "$limit": limit
    },
]
```

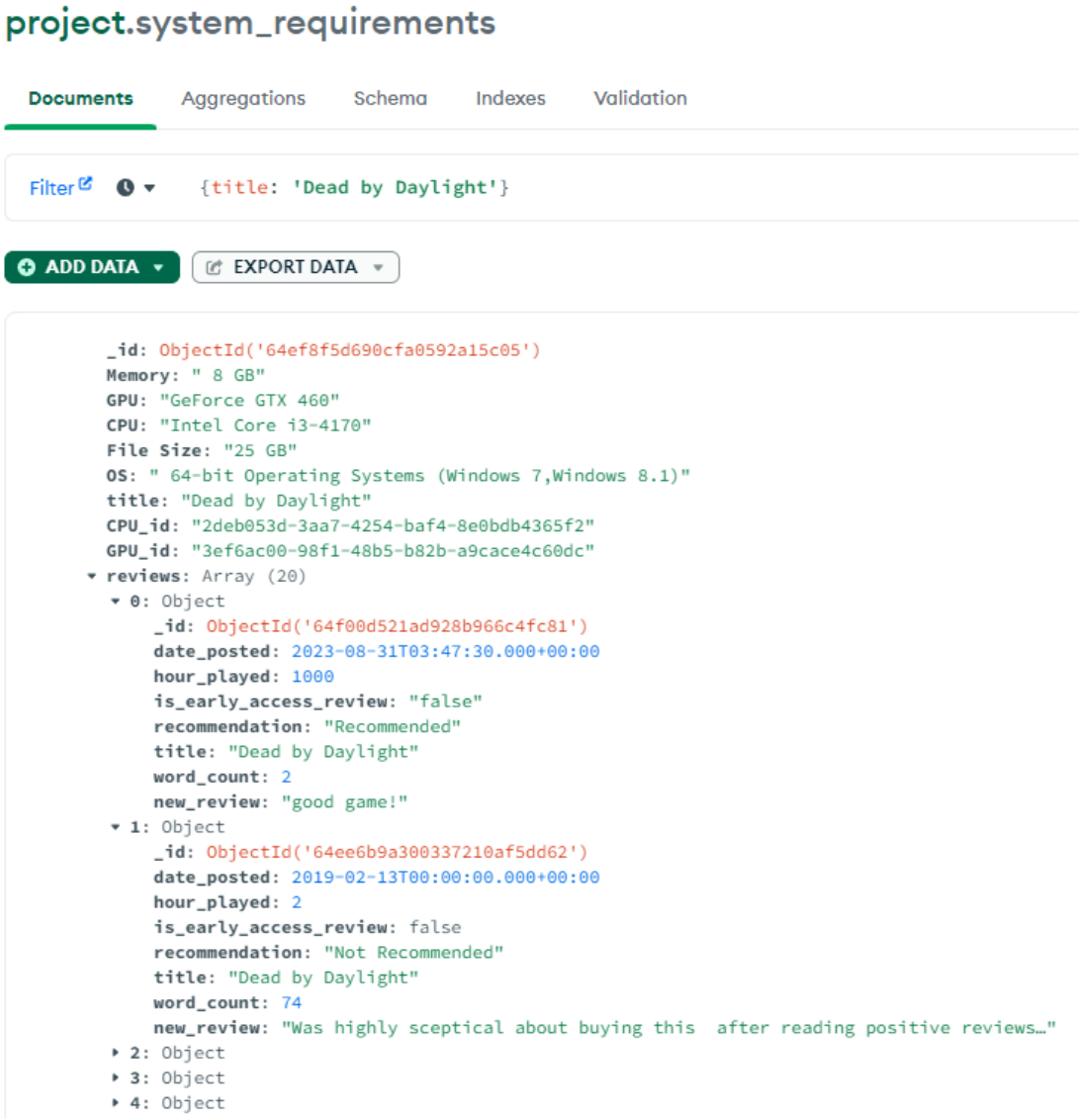
Figure 17 – Pipeline used for the component suggestion (limited by the user budget).

Complementing the analytics for the PC games reviews, when a User is browsing the trending games, it is desirable to check the actual reviews provided by other Users. In that way, a function that shows the User the latest reviews posted was designed. However during implementation, it was noticed that by embedding some of the reviews on the actual system requirements document would save time and computational resources, as when browsing for a game requirements the User would also get the latest review. The following images are then captures from the Mongo Compass software, which shows the results for the queries in the collections used.

First, by using the game_reviews collection, a query was designed to retrieve and sort the reviews for one game. The query structure and the results are shown:

```
{
      {title: "Dead by Daylight"},
      {$sort: {"date_posted": -1}},
      {$limit: 20}
}
```

For this specific search for the latest reviews, the system took 151 ms for the execution time, as many documents had to be examined. As previously mentioned, the alternative for this execution is to then embed the latest reviews in the system_requiremtns documents, exploiting the MongoDB capabilities. The following capture then shows the result of embedding some of the reviews on the game requirements.



In order to obtain now the list of latest reviews (up to a maximum of 20), the application only need to query the game requirements documents. The structure and results for this query:

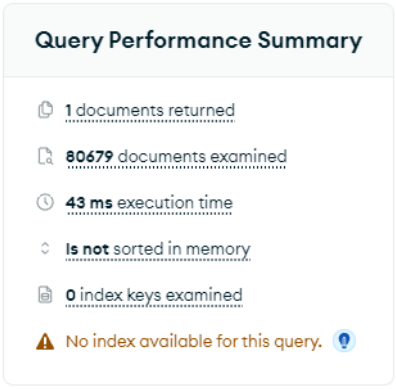
Using Mongo shell:

```
db.system_requirements.find({title: "Dead by Daylight"}).explain("executionStats")
```

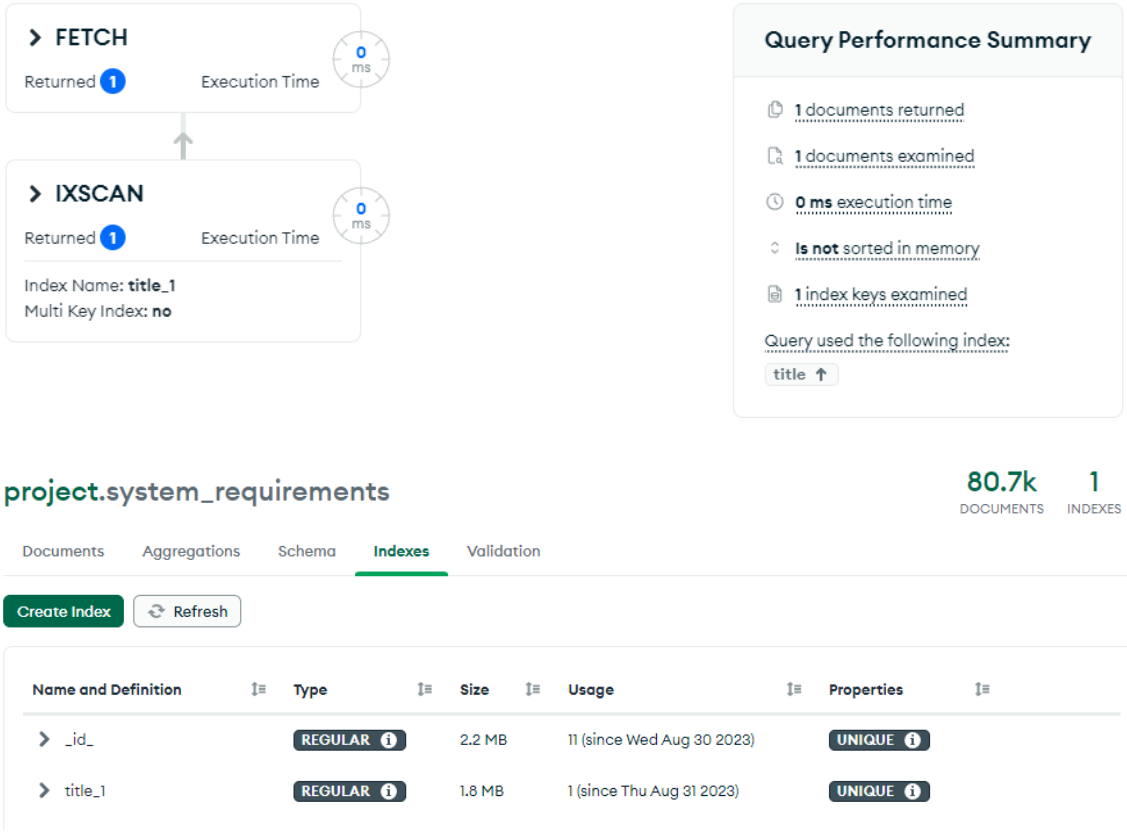
Using Mongo compass:

```
find({title: "Dead by Daylight"})
```

```
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 41,
    totalKeysExamined: 0,
    totalDocsExamined: 80679,
    executionStages: {
      stage: 'COLLSCAN',
      filter: { title: { '$eq': 'Dead by Daylight' } },
      nReturned: 1,
      executionTimeMillisEstimate: 0,
      works: 80681,
      advanced: 1,
      needTime: 80679,
      needYield: 0,
      saveState: 80,
      restoreState: 80,
      isEOF: 1,
      direction: 'forward',
      docsExamined: 80679
rs0 [direct: primary] project>
```



The result for this implementation shows a significant reduction in the query time, as the requirements collection is smaller in comparison with the reviews. Also, to improve even more the read operation, an Index was used on the system_requirements collection. The index used was the "title" field (name of the PC game), which is the most used parameter for searching. By including this index, the results for the same query were then:



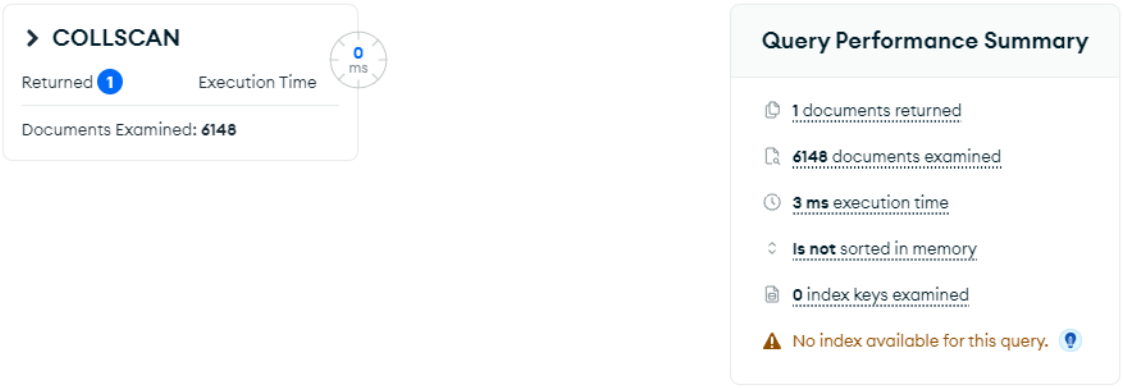
The following Table organizes the results obtained from the embedding solution and the use of the Index in the Execution time for this particular query for the latest reviews.

Search Method	Execution Time (ms)
Find and Sort “reviews” from the game_reviews collection	151
Find “game_requirement” from the system_requirements collection (NO index) – embedded reviews	43
Find “game_requirement” from the system_requirements collection (“title” used as index) – embedded reviews	0

As the Indexing method improved the performance for the review’s searches, an index was also implemented in the components searches, also resulting in an increase in performance as the execution time was significantly lower. As most of the read operation uses the components assigned ID value, an Index was created for this field, and the results for the queries are shown:

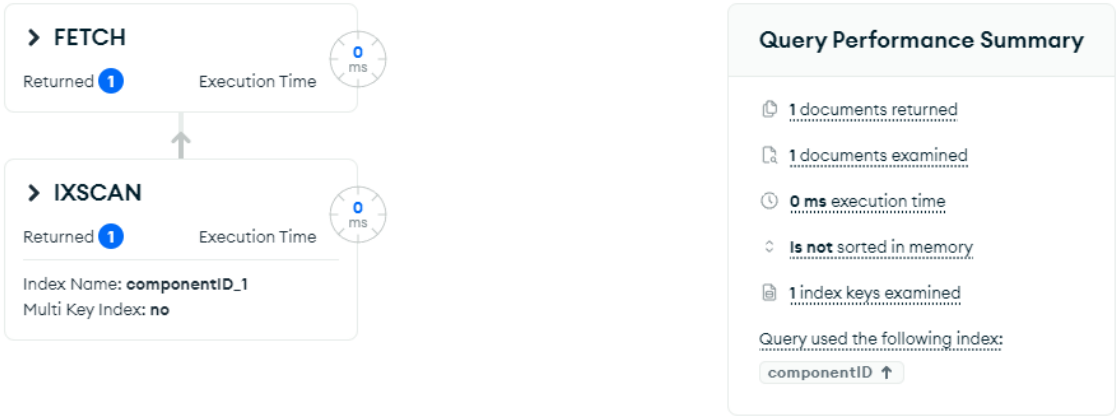
No index:

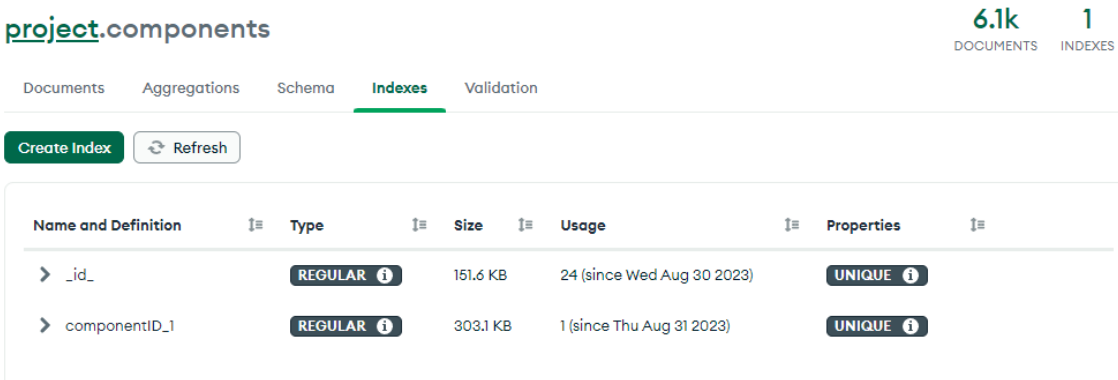
```
find {componentID: "7c1ee222-7b64-4b5e-b338-c4fd325e8449"}
```



Index for “componentID”:

```
find {componentID: "7c1ee222-7b64-4b5e-b338-c4fd325e8449"}
```





5.3 Statistics (Additional Functions)

As a final mention, a statistics function for the listed PC components was included for the manager, to obtain more information on how the distribution of components for each category is, as well as average pricing and benchmark scores. The results shown for the execution of this function for CPUs and for GPUs is shown:

```
->STATISTICS - GPUs statistics per [CATEGORY] from: 2015

[ Desktop ] Avg. G3D Mark: 11810.03 ; Avg. Price: 603.1 USD ;  Number of registered components:  76
[ Desktop ] Best Performance Component: GeForce RTX 3080 Ti (G3D Mark: 26887 ; Price: 1199.99 USD)

[ Workstation ] Avg. G3D Mark: 10787.65 ; Avg. Price: 1731.86 USD ;  Number of registered components:  26
[ Workstation ] Best Performance Component: RTX A5000 (G3D Mark: 22867 ; Price: 2631.2 USD)

[ Mobile ] Avg. G3D Mark: 5742.73 ; Avg. Price: 676.69 USD ;  Number of registered components:  11
[ Mobile ] Best Performance Component: GeForce RTX 2080 with Max-Q Design (G3D Mark: 13681 ; Price: 2099.99 USD)
```

Figure 18 – GPUs statistics for three categories.

```
->STATISTICS - CPUs statistics per [CATEGORY] from: 2020

[ Server ] Avg. CPU Mark: 30729.5 ; Avg. Price: 1954.14 USD ;  Number of registered components:  105
[ Server ] Best Performance Component: AMD EPYC 7763 (CPU Mark: 88338 ; Price: 7299.99 USD)

[ Desktop, Server ] Avg. CPU Mark: 25261.83 ; Avg. Price: 1577.61 USD ;  Number of registered components:  6
[ Desktop, Server ] Best Performance Component: Intel Xeon W-3265M @ 2.70GHz (CPU Mark: 39620 ; Price: 6353 USD)

[ Desktop ] Avg. CPU Mark: 19259.21 ; Avg. Price: 455.54 USD ;  Number of registered components:  118
[ Desktop ] Best Performance Component: AMD Ryzen Threadripper PRO 3995WX (CPU Mark: 83971 ; Price: 6807.98 USD)

[ Laptop, Server ] Avg. CPU Mark: 17921.0 ; Avg. Price: 536.5 USD ;  Number of registered components:  4
[ Laptop, Server ] Best Performance Component: Intel Xeon W-11955M @ 2.60GHz (CPU Mark: 23859 ; Price: 623 USD)
```

Figure 19 – CPUs statistics for four categories.

A pipeline structure was also used to obtain the results shown, allowing a fast access to the DB system with an efficient query:

```
pipeline = [
    {
        "$match": {
            "testDate": {"$gte": start_year}, "G3Dmark": {"$exists": "true"}, "price": {"$exists": "true"}
        }
    },
    {
        "$sort": {
            "G3Dmark": -1
        }
    },
    {  # group devices by category
        "$group": {
            "_id": "$category",
            "AvgBenchmark": {"$avg": "$G3Dmark"},
            "BestName": {"$first": "$gpuName"},
            "BestPerf": {"$first": "$G3Dmark"},
            "BestPrice": {"$first": "$price"},
            "AvgPrice": {"$avg": "$price"},
            "Registered": {"$count": {}}
        }
    },
    {
        "$sort": {
            "AvgBenchmark": -1
        }
    },
```

6. Conclusions

The results obtained from the testing of the developed function shows that the platform worked as expected, and all the services proposed were implemented, allowing a User to obtain valuable information about PC components, games and to evaluate is their system is compatible with a selected PC game. The reviews functions and analytics provided for the listed games in the platform complements well the system proposed and improves user experience when browsing for trending PC games.

In general, the access to the databases were fast and reliable, as not many entries were inserted in the DBMS and the volume of operations were also no significant. For a possible extension, more reviews could be added to the system, making the replica set implemented more relevant. In conclusion the presented functions provide a base for the development of a more complex and intuitive hardware evaluation platform, as graphical interfaces could be used to improve the user experience.