CIS550 Project Final Report

Project Name: Gameeee!
Group Members: Yiwei Liu, Wenbo Tian, Yuhan Gu, Hanrong Zhu

I. Introduction and Project Goals

Everyone loves video games. According to MobyGames.com, more than 63,762 video games have been released since 1950 across all platforms. Among all these platforms, Steam has definitely gained the most popularity among teenagers and young adult players. While Steam itself is a great game community where people can easily look up, download and rate the games they like, there are also times when the website's bountiful resources and information can be a distraction or even an obstruction to users' purpose: maybe sometimes what they need is just a simple lookup of a game, or sharing a few words about it. In this situation, our website would be a more convenient and helpful platform for them. With our website, players can easily search for a game and get all the information they need with a simple click, read the most complete player reviews and add their own, and get the customized recommendation based on their tastes, etc. The goal of our project is to make everything simpler and easier for Steam players when they need.

II. Data Sources and Technologies

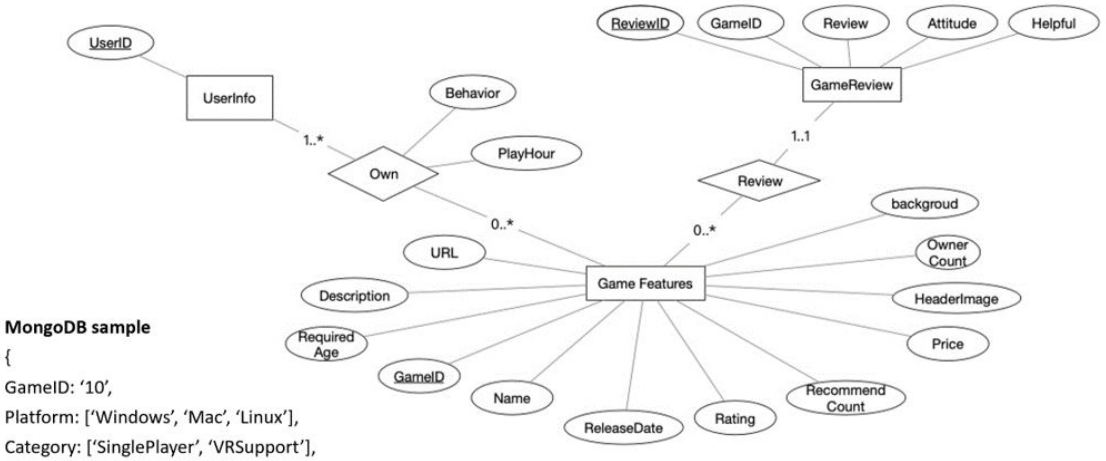
1. **Game_features.csv** (https://data.world/craigkelly/steam-game-data)
Games_features.csv contains all the basic information of games, including game id, name, description, rating, etc. Since there are so many attributes in the dataset, we first selected out the variables we needed and did some data cleaning (e.g. attribute integration; null value removal) before importing the data into the relational database.
2. **Game_features.json** (https://github.com/CraigKelly/steam-data)
This json file is similar to Game_Features.csv, but it is larger and contains some information that Game_Features doesn't have. These extra information are usually in the form of an array (e.g. Genre: adventure, romantic), which makes it difficult to integrate them into relational database. Thus we chose to build this part of data with NoSQL solution(MongoDB).
3. **Game Reviews.csv** (https://zenodo.org/record/1000885#.W7FD1ChKg2w)
This dataset contains user's reviews on games and their attitudes toward different games (positive / negative)
4. **User behavior.csv** (https://www.kaggle.com/tamber/steam-video-games)
This dataset contains user's purchase and playing information, such as playing hours.

Relational Schema

After entity resolution and normalization, we built our relational schema:

- **GameFeatures** (GameID, Name, ReleaseDate, RequireAge, Rating, RecommendCount, Price, OwnerCount, URL, Description, HeaderImage, Background)
- **GameReview** (ReviewID**,** GameID, Reivew, Attitude, Helpful)
    - GameID is a foreign key of GameFeatures.GameID
- **UserInfo** (UserID**,** GameID, GameName, Behavior, PlayHour)
    - GameID is a foreign key of GameFeatures.GameID

E-R Diagram



MongoDB sample

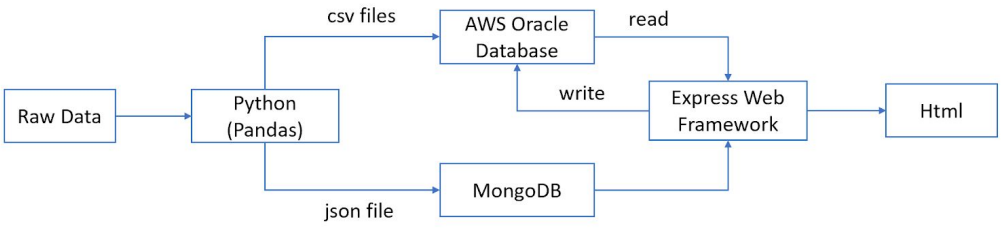
{
GameID: '10',
Platform: ['Windows', 'Mac', 'Linux'],
Category: ['SinglePlayer', 'VRSupport'],
Genres: ['Adventure', 'Action'],
Languages: ['English', 'French', 'German'],
...}

Technologies

We used pandas package in python to clean the data, and imported our csv files into AWS Oracle database and json file into our local MongoDB database. For backend, we used Express Web Framework (angular js/ node js / javascript). For frontend, we used HTML / CSS. During the project, we also used git to update our latest version.

III. Diagram & description of system architecture

A flow chart of our project design



Data Source	Data Cleaning	Data Storage	Backend	Frontend
			Read / write data	Get values passed by backend, and show them in the web pages

The above diagram is a flow chart showing the whole process of how we build our project: from processing raw data to showing the final html pages.

Data cleaning

The first thing we did is data cleaning with python. The major goal of this step is to make our dataset meet constraints in our relation schemas. We removed duplicates, null values and empty strings on keys, and built new column as primary key if there's no suitable key value of a dataset. We dropped rows in user and review dataset to meet foreign key constraints.

We also did lots of cleaning to make our datasets less messy. We checked missing values in each column and remove rows if some important attributes are missing (such as game name). We changed data type of date column from string to datetime so that they could have a uniform format. Since there's tons of columns in the original dataset, we only kept features we want and removed the rest. In the review dataset, there's about 1/3 reviews are 'Early Access Reviews' with no useful information. So we removed those reviews and significantly made the review dataset smaller.

For json data, we kept only the records that have the same game_id with the game_id in game feature dataset. The reason we did that was to make it consistent with our csv data, so that we could have less trouble when passing values between them in our backend.

Data Storage

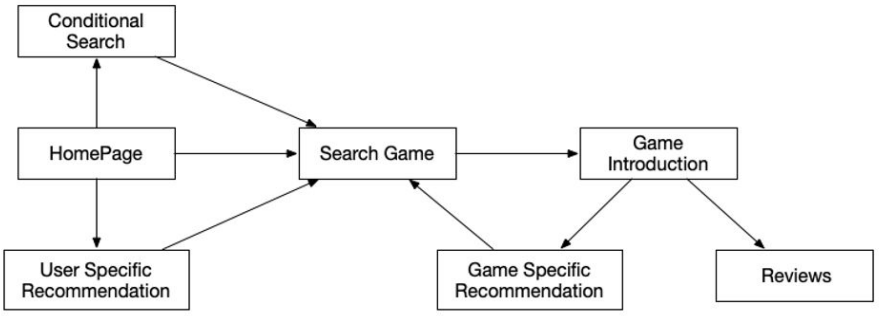
We imported our three cleaned csv files into our AWS Oracle Database, and the json file into our local MongoDB. After importation, we set key constraints in our relational database, and set the key value of review table to auto increment (make it more convenient for insertion operation in our web application).

Backend and frontend solutions

We use Express Web Framework(node.js/JavaScript) as our backend solution and HTML/CSS as our frontend solution.

At backend, we use routes and controllers in Express to handle HTTP requests. In the router, we define functions to connect to database and execute SQL/MongoDB queries regarding different HTTP requests. When receiving a HTTP request, the router forward the request to particular controllers so that we can read data from database. In the controller, we define functions to process data retrieved from database, and store them in variables so that they could be get in the frontend.

At frontend, we use HTML to present the data on the webpage, and use CSS to style our web pages. The connections between different HTML pages are shown below:



IV. Implementation of features

According to the motivation and goal of this web project, we designed two main features, which were searching and recommendation, and extended to other relevant functionalities.

Search

We designed three forms of searching that can satisfy different requirement of users. If they have specific interesting games, they can use search box in the homepage and find all the relevant results; If they just search for some interesting games without specific target, the games recommended in the homepage can give them some ideas; also, if they just skim through the website and look for some interesting games based on some conditions, the filter search page would suit them perfectly. The detailed features are introduced as following:

Basic Information Search: when the user enters the name (or keywords of the name) of a game he or she wants to know about, a form of the game's basic information will be displayed, which includes Game ID, Game Name, Game Image, Description, Required Age, Price and Rating and Top Reviews

Filter Search: the user can apply filters with respect to the game features he or she is looking for, and a list of games that satisfy the criterion will be displayed, sorted by the popularity of the games

Game Reviews Search: if the user would like to read more reviews of a game, he or she can first search for the name of the game, and from the basic information page, click on the "view more reviews" option and redirected to the review page of the game

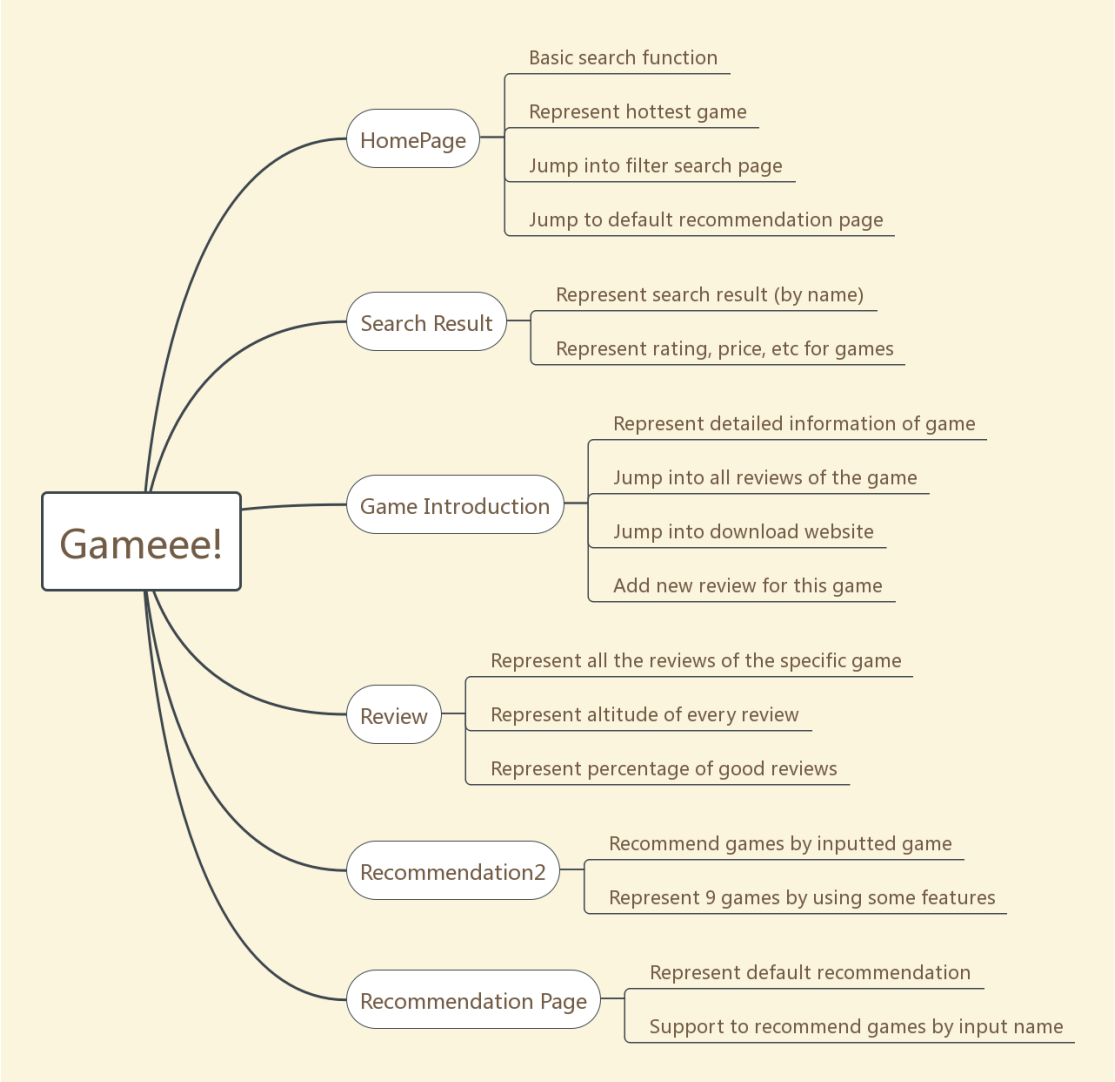
Recommendations

General Recommendation: a list of latest, hottest, best free games will be displayed. In the homepage, we retrieved data from games table from database and used different categories (hottest, latest and best free) to represent some default recommendation to users. If user is interested in a specific game, they can click on the picture and enter the game search result page to see all relevant games.

User specific recommendation: from the homepage, users can jump into recommendation page and find randomly selected games. Via clicking on their interested games, they could find another 9 recommendations that are based on the game that they clicked on. During this process, users can have much more interaction with our website and there are chances that they would find their interested games.

User Personalized Recommendation: recommend a list of games that's similar to the game the user searched. Based on the games that they are looking at on the game introduction page, we would recommend another 9 games based on our recommendation algorithms. These recommended games would share some common characteristics with specific games and are also played by those users who play the previous searched game. On the other hand, there is also a search box in the recommendation page. This design would make much more convenience for users to search for much more recommendations.

During the process of implementation, we transformed these features into function points of our webpages. The following picture will show detailed functional points of every page.



V. Performance evaluation

At the time when we finished building our website and tested it, we found that the time needed for our router to respond was quite long, especially those need to read data from review table. Thus we need to optimize our query and data set to speed up our website.

What we did first was to remove the unnecessary join clauses in our query. When finding reviews of a specific game, at first we chose to join the game table and the review table because we took game name to be the search key, but that was slow and redundant. We used game id instead, which is already in our review table, and found a great improvement of our performance.

Meanwhile, we found that lots of our queries need to read tuples based on game id or game name, or find games whose rating are within a specific range, so we added indexes on GameFeatures .rating, GameFeatures .name, GameReview .game_id, UserInfo.game_name and UserInfo.game_id, all of which show up in our 'where' and 'group by' clauses. After that we saw a speed up from 50% to 90%, evaluated by the time needed for our router to response after receiving a request.

The table below shows the time needed before and after adding the indexes, using ‘Dota 2’, an extremely popular game, to test. For most of the games, time needed should be less than one second.

	Search games	Read 5 reviews	Read all reviews	Recommend games
Before adding indexes	300-500 ms	7000 ms	11000 ms	8000 ms
After adding indexes	170 ms	500 ms	5000 ms	1000-2000 ms

VI. Technical challenges

During the process of implementing our web project and optimization, we encountered several technical challenges. And after searching for relevant materials and learning from tutorials, we finally solved them and harvested a lot from this process.

Pass values

The first big challenge we encountered was how to pass value between different controllers. Since we have finished the implementation of functional points of single page and need to receive and send value between different pages. We tried several methods. The most common method was to use broadcast and listener to finish passing values. However, this method was much more suitable for hierarchy controllers, which could send message from ‘parent’ controller to ‘child’ controller; another common method is to use global variable $rootscope

to pass values between different controllers. This method would be very convenient but could be much easier to cause conflicts during updating the values of $rootscope.

At last, we found two method to pass values and made it. Firstly, we use id to uniquely identify a tag in html and store its value using HTML local storage. When we need to use it again in other controllers, we call it back and get its value. Secondly, in most cases, the value that we pass between controllers is really short and all of integers. So we make use of URL to pass value, and parse the variables (for example, the id of games) in another controller. These two methods were both used in our project and when we needed to pass id of games, we would prefer the latter method to make a lower space complexity.

Recommendation algorithms

The second challenge we solved for our project was how to use information stored in database to finish recommendation. It is extremely hard to find relation and similarity between different games only use the game information, so at first how to recommend proper games based on one given game to our user was a big problem for us.

We solved this problem from the insight that people playing the same game should have similar interest, so we could use other players' information to infer the interest of our user. Given a game, we find players who played this game and the time they spent on other games, use this as our first component of recommendation algorithm.

Meanwhile, we need to give a bonus to the games with a high rating and high proportion of positive review. We need both of them because sometimes there is a disagreement between the game media (rating) and the players (positive reviews). So we joined our intermediate result with the 'GameFeatures' table and 'GameReview' table to find the mentioned two components. Weighted summing the three components gives the final recommendation index.

What is a little tricky here is that some unpopular games have no game media ratings on them and no users in our data set played them, which will turn out to have two zero components in our algorithm. We believe these games should be somewhat worse than average but not as bad as rated zero. So we assume their rating as a standard deviation less than average and 50% of their reviews being positive(which is worse than average). As for the games with no rating in metacritic (a world-wide game rating website), it should commonly be worse than average but better than a zero-point game, so we use the average of games minus 5 as its rating. Then we weighted sum the two components as our final recommendation index.

The complex query that realize our recommendation algorithm is shown below:

```
WITH average AS (SELECT Avg(rating) FROM games WHERE rating <>0),
raw2 AS
(SELECT * FROM (SELECT * FROM (
            SELECT  games.NAME,
            Cast(CASE WHEN games.rating>0 THEN G1.avg_hour/2+games.rating
               ELSE G1.avg_hour/2+Cast((SELECT  * FROM average) AS INT)-11 END AS INT)AS
            R_index ,games.gameid,games.headerimage,games.rating
            FROM games JOIN
            (SELECT  U.gameid AS games,Avg(U.play_hours) AS avg_hour FROM users U WHERE U.user_id IN
            (SELECT  * FROM (SELECT DISTINCT user_id FROM users WHERE game_name= req.params.GameName))  GROUP BY U.gameid) G1
            ON games.gameid=G1.games
            WHERE games.NAME <> req.params.GameName  ORDER BY r_index DESC) WHERE rownum<41) Raw_rec LEFT OUTER JOIN
            reviews_whole ON reviews_whole.game_id = Raw_rec.gameid )

SELECT * FROM
(SELECT DISTINCT raw2.NAME, Cast(r_index*0.7+(CASE WHEN c1/c2 IS NULL THEN 0.5
          WHEN c1/c2<0.5 THEN 0.5 ELSE c1/c2 END)*30 AS INT)  AS REC_INDEX,raw2.gameid,raw2.headerimage
FROM raw2 LEFT OUTER JOIN (SELECT raw2.gameid, Count(*) AS C2 FROM raw2 GROUP BY gameid) SUM_COUNT ON
          raw2.gameid=SUM_COUNT.gameid
LEFT OUTER JOIN (SELECT raw2.gameid, Count(*) AS C1 FROM raw2 WHERE sentiment=1 GROUP BY gameid)
          POSITIVE_COUNT ON raw2.gameid=POSITIVE_COUNT.gameid ORDER BY rec_index DESC)
WHERE rownum<10
```

Better UI design

The third main challenge we encountered was how to style our pages to make our website professional and user-friendly. Thus we spent a lot of time designing our interfaces after finishing the basic implementation of the project,.

Firstly, we make the style and colors consistent between pages and create navigation bars on each page. Secondly, we use some common method to modify pages, for example, we use hover effect to represent recommended games in homepage. Thirdly, we skimmed through quantities of game websites to learn other websites' styles and make the background much more consistent to styles of game websites.

Finally, we also use version control tool to manage our project. Since this was a collaborative work and every time we would make a lot of modifications of it, GitHub provided us a strong tool to guarantee that we can maintain the latest version of project, which improved our efficiency in a large degree. Also, during the process of implementation, we found a big bug in our latest version, and via using version control tool, we rollbacked and recover it in a few minutes.

VII. Extra Credits

We used oracle database on AWS, which makes things much easier to manage the database. We also used sophisticated web-database technology such as MEAN stack. Most of our backend works are built on Express web framework.

The use of NoSQL and MongoDB is an important part in our project. Since there's many attributes that could be expressed as multiple value arrays (e.g. genres: ['Action', 'Adventure']), NoSQL with embedded document structure is a much better choice than relation schemas. In relational schema, we may need to make a bunch of binary variables to represent genres, or to make a new table to store game genres. But with NoSQL, no further work needs to be done.