Computer Science Project

Game Comparison Website/App

Link to our website:

http://www.doc.gold.ac.uk/usr/900/

http://www.doc.gold.ac.uk/usr/900/api

http://www.doc.gold.ac.uk/usr/900/Complaintsadmin

Contents

Introduction

The concept of this project is to create a price comparison web app designed to enhance and simplify the e-commerce experience, from the very hard-core gamers to the casuals. Price comparison sites do exactly what their name implies: compare the price of goods and services from a range of companies. This allows consumers to make sure they get a deal that's tailored to them at the best price. In 2018, it was estimated that more than 11 million people used price comparison sites and it's clear why they appeal. They have had an almost revolutionary impact, allowing people to save time and access a wider range of products.

We decided to aim for a smaller (but still quite massive) segment of that market - the video game industry. The exact same principles apply, except the comparisons will only consist of games. The web app interface on the customer side will offer fluidity and dynamic content types - features that existing sites do not have. The inspiration for this concept derives from the will to reduce the complexity and countless hours wasted searching for the cheapest games on the internet. With so many options to choose from - consumers are dealt with the difficult task of finding the best deal for them.

Features & Objectives

- ❏ All prices over time using API (Graphs, dates)
- ❏ Player base over time since release
- ❏ Live player count
- ❏ Community reviews
- ❏ Prediction of prices in the future.
- ❏ Create forums page where users can login and add hot deals on certain games for others to view *(added on after stakeholder interview)*

USP

- ❏ We only use official and legitimate retailers such as Steam, Epic Games, Ubisoft, Origin, Green Man Gaming, GOG
- ❏ We are going to have live player counts of games and show reviews from the community.
- ❏ We would show users a prediction of prices in the future by implementing machine learning that can use previous data on that game and predict future prices.

Analysis

Identification

Video games have been around since the 1970s and to this date there are now more than 2.5 billion active gamers around the world - an increase of a billion from just five years ago. This surge of consumers has seen the video games market value reach $155 billion with it also expected to grow to over $200 billion by 2023. Simultaneously, the average price of video games over time has also increased, with recently 2K games becoming the first publisher to set a $70 asking price for its franchise NBA 2K21.

However, there are many third-party retailers selling keys at cheaper prices but with so many options to choose from - consumers are dealt with the difficult task of finding the best deal for them. This is why we have decided to create a games comparison website which simply compiles all retail prices on the internet and provides the cheapest option to them.

Stakeholders

Our application will be aimed at every single person in the gaming community - we do not want anyone to be left out. The consumer is at the forefront of our minds, essentially their needs are our wants. We believe that we can significantly help an industry that has still yet to peak. Not only do we aid the consumer, we also enable them to reach retailers which means both sides benefit. For example, Shopper A will look at the main distributors such as Steam, GOG and Epic Games who are likely to sell at original prices with very few discounts; whereas, Shopper B browsing our website will see several more retailers selling their favourite games for much cheaper. Shopper B will save their money and at the same time, smaller retailers are able to generate some more revenue. In these scenarios, publishers and developers will always see the same earnings however way the consumer decides to purchase the game. This is because third party retailers choose to accept less profit per sale in order to make more of these sales - we are just helping them achieve this.

However, the deals consumers see on our website come at a cost - something we want to minimise on the users end. Some similar price comparison sites add on fees to purchases which goes against the point of hosting a service like this. This is where our sponsors and partners come in. Ads are an essential to us as we will rely on this as our only source of income therefore, we hope to earn our keep through the use of Google Adsense. However, unlike our competitors, our aim is to partner with Gener8. They are one of the world's first browser technologies to enable people to take control of their data and earn money from the ads that they see. Therefore, not only will website viewers obtain the cheapest games but also earn money at the same time. This once again emphasises our mission to assist our target audience as much as possible.

Something that may also benefit the video game culture are potential partnerships with various developers and publishers. Gaming companies are likely to be interested in projects such as these because, as stated before, they receive the same return regardless on what platform their game is sold on. Our website is a perfect place for them to advertise their product further and gain

additional revenue. These partnerships can also bring out giveaways and exclusive access to new games - a great way to improve public relations between a business and its consumer.

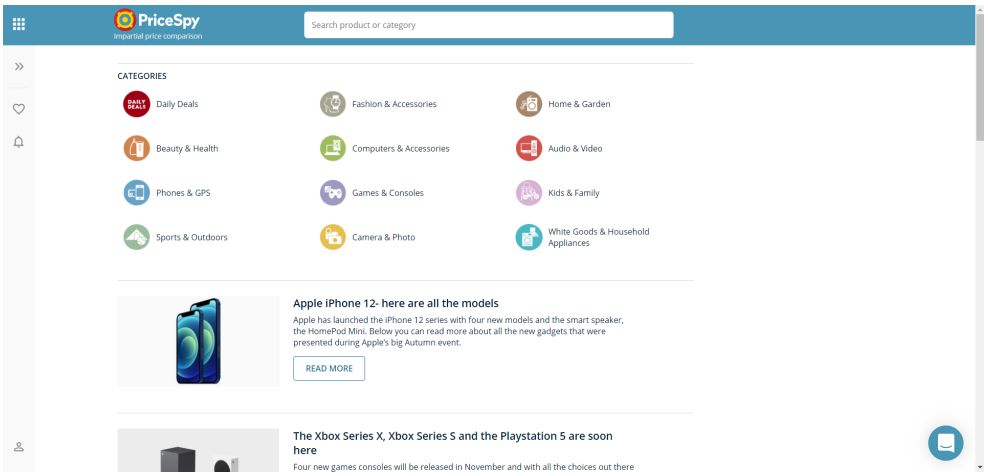
Research

Research into existing services

There are many existing problems and solutions that have similar features to the website that we are hoping to create, the basic need of all of these solutions will be similar. The problem arises when creating a working program that can fulfil all the needs of the user such as reliability and impartial behaviour to all distributors on the market and at the same time being unique.

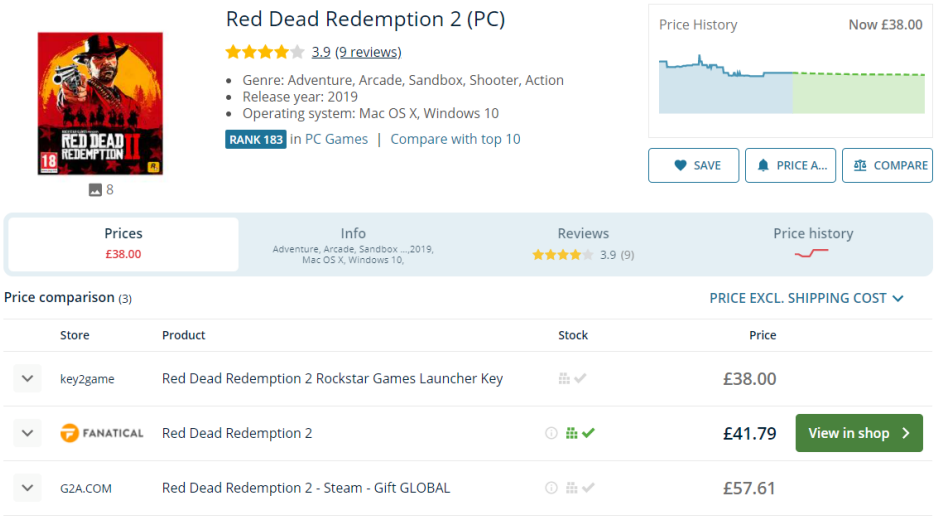
The following example is a mainstream comparison website which is used for nearly every product and service on the internet.

URL: [PriceSpy]



Overview: It is unlikely that the UI will be similar to ours but a reason why this is an extremely popular comparison tool is that the site is very simple and seems easy to use, although it does not provide all the functions that our target audience require and neither does it appeal to the whole gaming market.

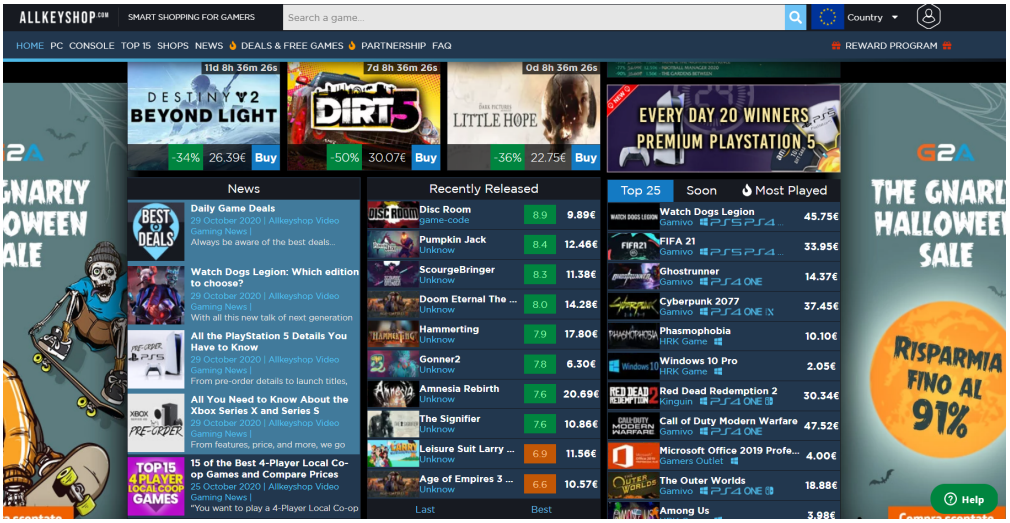
Things we may use: The main thing that we can take inspiration from is that PriceSpy includes a price history graph which is extremely beneficial as the customer can see at what period of time the game was highest/lowest and perhaps make an estimate of when it will go even lower. Additionally, PriceSpy also lists out clearly where you can buy the game from and whether it's in stock or not. This is great as the customer can clearly identify the purchase they're looking for.



Things we may not use: A problem with PriceSpy is that they also list third party websites in their comparison results. This is a serious issue as many times over the past years, G2A have been caught reselling stolen game keys - something they have neither apologised or corrected. Due to this some users have been reported saying their keys have been revoked and due to the nature of how G2A and other key sellers work, a refund is always out of the question. This is why we promise to only list verified retailers so a scenario like this never occurs.

This example is more of a localised comparison website. It looks at all games for all consoles.

URL: [ALLKEYSHOP]



Overview: With this website, consumers can straight away see what the best deals are. They also include the latest releases along with ratings for those games. These features are great for those looking to quickly skim through offers. However, the UI does seem very clunky and excessive so it may confuse people, especially first time viewers.

Things we may use: We admire the different lists that they have: Top 25, Soon and Most Played and that they also have sections for each genre and then list those top games and their prices - this is very beneficial especially for niche gamers. They also have reviews and comments from people that have played the game - word of mouth is useful in these scenarios.

Things we may not use: The UI is very unappealing especially as you scroll further down the site. Unfilled black space along with a long list of ads - these are not user friendly aspects of a website. Functionality wise, ALLKEYSHOP also has deals for Playstation and Nintendo but our focus is only on PC. This is because not only is the demographic for PCs much higher, digital console games are never sold on legitimate retailers and as said before it is not something we want to risk with our audience.

Changes to our Implementation Process

In our Technical Specification from our proposal, we stated the tools we will use to create our initial concept but did not specify how we will carry that out. Since MongoDB is the database we have chosen to store our data, we will be using MongoDB Atlas as our running server and MongoDB Compass as a way to structure our data. We will also require Node.js and Express as 'Node Package manager' contains a large library of useful modules that can be used in our project, reducing the need to build things from the ground up, while Express vastly simplifies the process of handling requests, creating routes, and rendering views in a web application. To put this together we will be using a mixture of Vim and Visual Studio Code with the aim of fully moving to VSCode further into the development process.

The reason why we aim to do this is because it is easier to edit and write code on VSCode perhaps due to our familiarity with VSCode compared to Vim. Vim requires us to know how to navigate through the terminal which requires an excess amount of commands. For example to delete a line of code you would need to press the key "d" twice and to save your work you would need to type in ":wq"; whereas on VScode you would need to highlight and delete the line of code and to save your work you would need to hold Ctrl + S. As our application becomes larger we will make the decision to fully integrate, especially as Vim makes our code unreadable. For example, when we pasted some code it would be presented all on one line and it would make it harder for us to figure out where in that lengthy line of code as to where the error is occurring from. However, by doing it in VSCode our code is more legible and we are able to use certain functions that come with the app to help us for example word wrap. Additionally, because VSCode is an IDE it can aid us in spacing and indentation along with the actual coding.

These technologies were chosen for the following reasons:

Node.js

- Node package manager contains a large library of useful frameworks that we will require to execute the project, for example Express and.
- Our application will mostly use JavaScript and html, and as Node is a JavaScript runtime, we can easily code all the features required without using any additional programmes.
- Why node.js? (*https://learn.gold.ac.uk/course/view.php?id=16001* , lecture slides)
- A common task for a web server can be to open on the server and return the content to the client. How node.js handles a file request is that the user sends the task to the computer's file system then the system is ready to handle the next request. When the file system has opened and read the file, the server returns the content to the client. Node.js eliminates waiting for the file to be opened and read and it simply continues with the next request allowing the server to handle large numbers of connections and requests due to the convenience that node.js provides.
- Node.js runs single threaded ,non-blocking asynchronous programming, which is also very memory efficient.
- Node.js can generate dynamic page content and it can create, open, read, write, delete and close files on the server (CRUD operations). Node.js can collect form data and can add,delete,modify data in the database.
- Node.js files contain tasks that will be executed on certain events. Node.js is good for input and output intensive apps which is what our website is essentially going to be doing. It would give an output to the users input data. Also node.js Is highly scalable but it is not good to use for CPU intensive apps because if it is cpu intensive node will need to make other clients wait until.
- The reason why we are using node.js instead of PHP or its other competitors is because node.js eliminates the waiting time allowing the server to handle more get and post requests which would make the server more efficient when large numbers of users try to get data from the server and post data to the server whilst PHP would wait until the server gives a response to a request from the user and when many users request for data from the server it could end up causing users to wait longer than they should for a simple request. Hence we are using Node.js as it does not need to wait for the server to give a response to a request it would just handle the next request whilst the server is getting a response for the past request.

Mongo

- As we are creating a price comparison website, we will have a lot of data to sort and collect from a number of different retailers. So, we would require a database system that is flexible and can handle the load of the data. Mongo will also allow us to shard our data, our database can be partitioned into several small databases.
- MongoDB also has a high speed query response, which is important as we have a lot of different data types within our database and may require quick access when being used by the customer. However, the downside of this is that Joins are not supported, and we will have to manually code them into our code.

- We decided to use MongoDB instead of MySQL because mongodb can work with big data whilst MySQL cannot and as our application uses large data collections we would need to work with a database that can work with large data collections and MySQL does not work very well with large cluster of data hence we decided to use MongoDB instead as it can work well with large cluster of data. MongoDB is also more scalable and provides superior performance compared to MySQL meaning that we can access large data with MongoDB compared to MySQL and as we are using large unstructured data it is more beneficial to use MongoDB rather than MySQL.

Machine Learning

- In order to extend the scope of the project, we will be implementing machine learning in order to make accurate predictions on future prices of a particular game.
- The programming language we will use in order to utilise machine learning will be python due to its huge popularity in this field, above all the other programming languages. Furthermore, an article from Stack Overflow clearly outlines why python is the best choice - *“There are 69% of machine learning engineers and Python has become the favourite choice for data analytics, machine learning, and AI – all thanks to its vast library ecosystem that let’s machine learning practitioners access, handle, transform, and process data with ease. Python wins the heart of machine learning engineers for its platform independence, less complexity, and better readability.” (Source: Stack Overflow).* We intend to make use of the different libraries python has to offer in order to achieve our goal.
- We will achieve this by importing numerous data about price history on all of the games we have in our database.
- We will build a model, which will learn from the data and accurately predict future prices
- **Linear Regression** - this is a supervised machine learning algorithm with a continuous and constant slope expected performance. Rather than attempting to classify values into groups (e.g. cat, dog), it's used to forecast values within a continuous range (e.g. sales, price).
- **Simple regression** - uses the equation of a line (y = mx + b) where m and b are the variables our algorithm will try to “learn” to produce the most accurate predictions. X represents our input data and y represents the prediction.

If we have time for our final iteration we would like to implement simple regression where we would give data on either price of the games on certain sites and calculate the price in the future by using the equation of a line and as prices of the game over time would have a pattern which can help us find if it was on sale or not and we can give a recommendation to users to when the best time is to buy that specific game. Instead we may use simple regression to calculate how many people will play the game in the future to show users whether the game has longevity and would still have a player base in the future and how fast the player base would go down this would be harder to implement as there may be external factors that may affect the games player base for example a lot of twitch streamers may decide to play the game which can cause a spike which we can not factor in.

Development and Testing

Before we actually start our development, it is important we plan ahead so we know what work we will have to complete and be made ready for review in a set period of time. To do that we have created a sprint log which includes all key features our website will require.

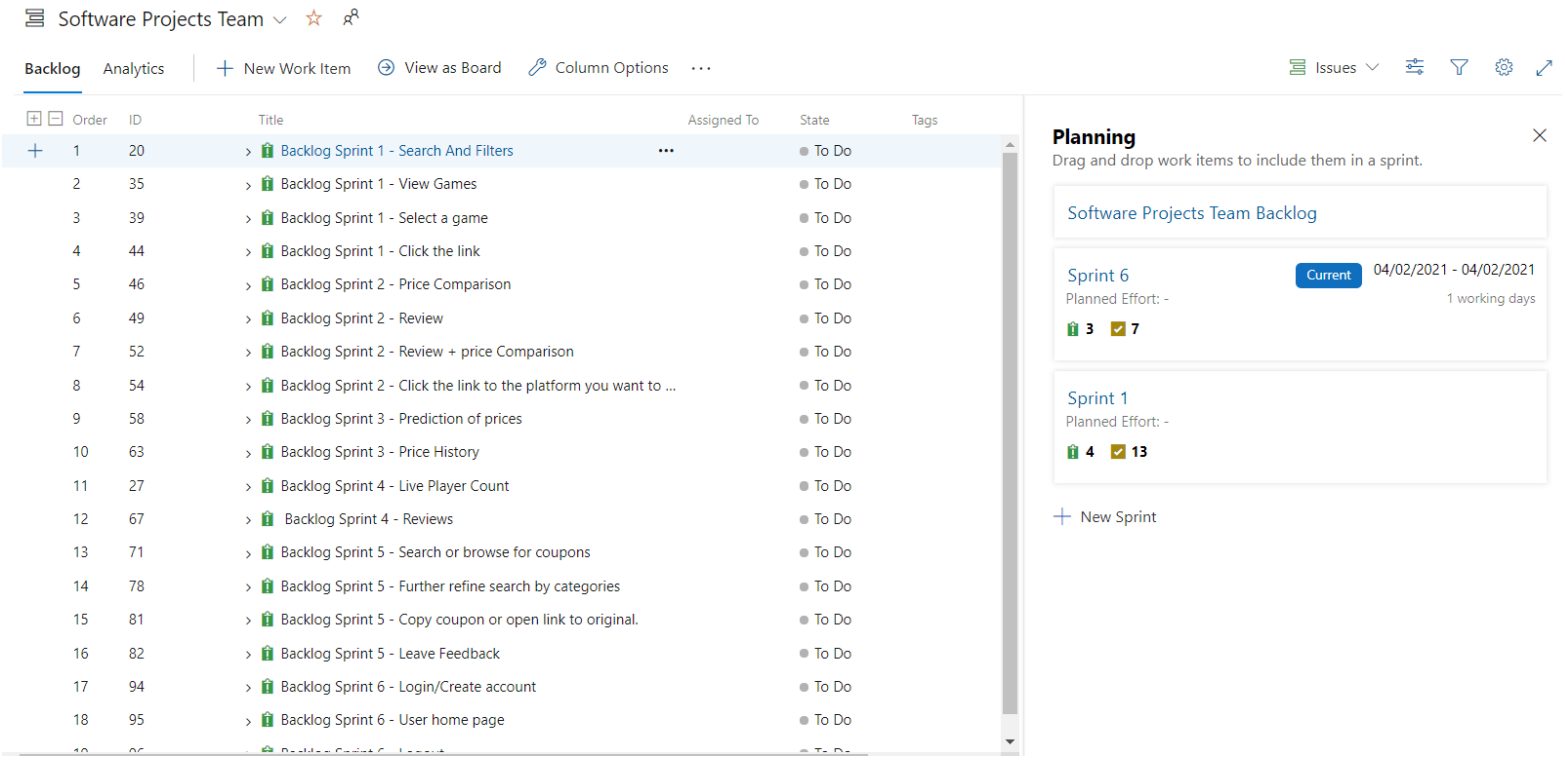


Figure 1.1: Sprint log

Sprints are a part of the agile methodology which we have talked about in our planning report. They are important because it makes the project more manageable and allows us to ship high-quality work faster and more frequently, and gives us more flexibility to adapt to change.

Iteration 1

In this iteration, we will be focussing on the following criteria:

- Setting up and creating the database
- Setting up the Express web server

Setting up the database

Firstly, we required a server for our database to run. We identified MongoDB Atlas as the best service due to its low cost and ability to provide availability with the most demanding databases.

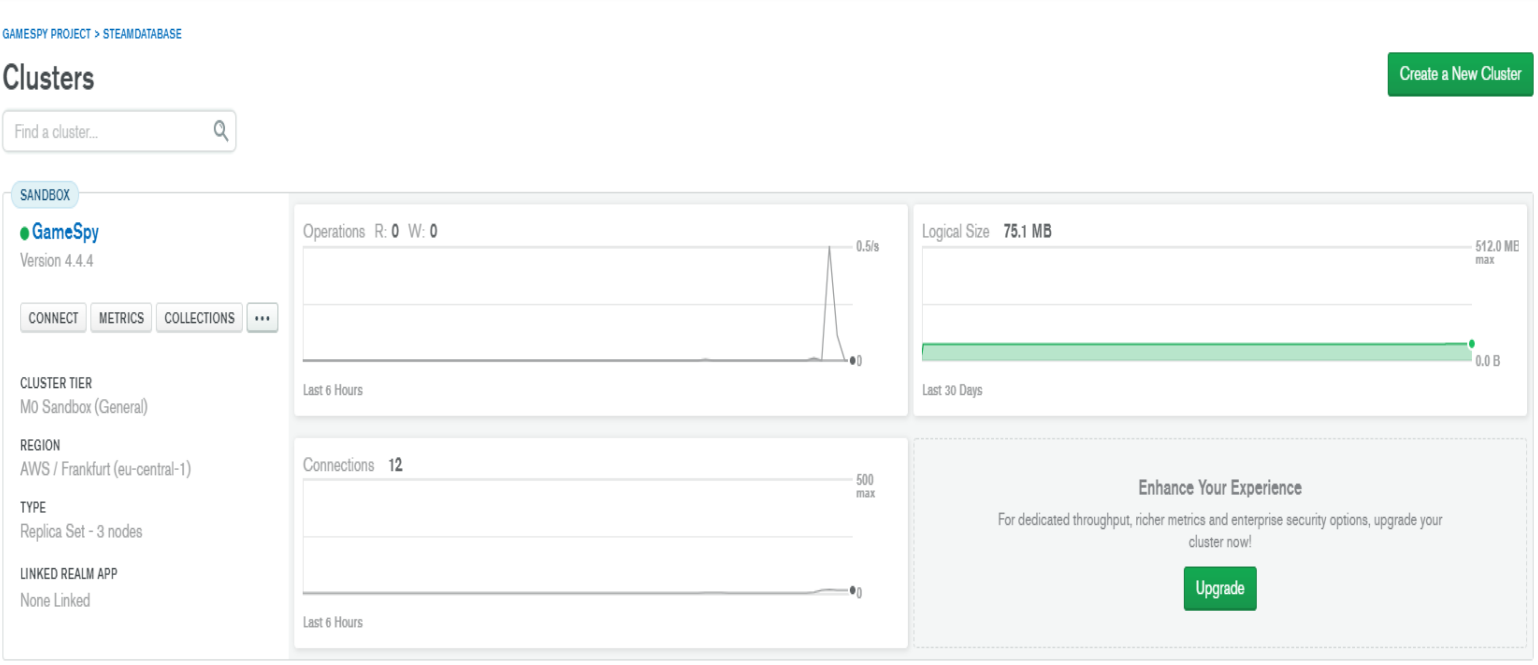


Figure 1.2: Setting up the cluster server

Once this was done, we moved on to the actual database software - MongoDB Compass. It includes a graphical view of the MongoDB schema without the need of query language and also analyses documents which allows us to be efficient and saves us time.

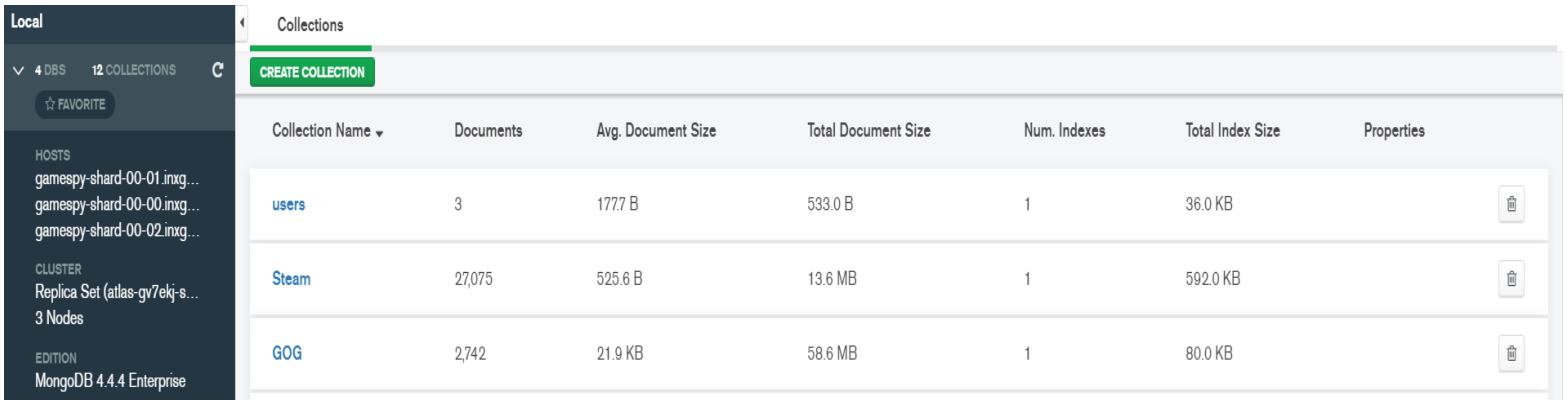
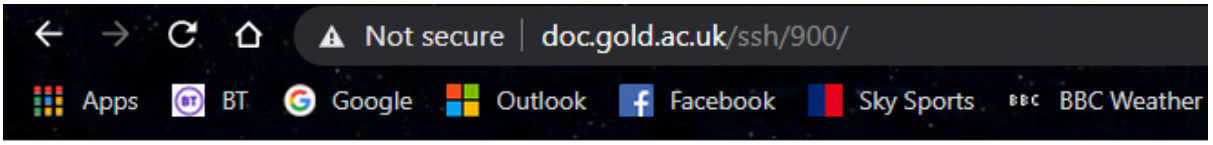


Figure 1.3: Creating the database on MongoDB

We created a database called GameSpy then inside it, we created collections which will contain data from the different gaming platforms. Currently, we gathered the data through external data science websites which have csv and json files of what we need. Although this is a prompt method of collecting data, it isn't reliable and neither is it likely to be up to date; however, this is only our first iteration and so for now we will be using these datasets as a basis to work on.

Setting up the Express web server

After connecting MongoDB Atlas and Compass, we went on to create the web server. Using the resources obtained from our university we set up our server.




```
doc900 login: sysadmin
sysadmin@doc900's password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-93-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
New release '18.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

sysadmin@doc900:~$
```

Figure 2: Our Express web server

We indicated that technologies such as Vim and Node.js were required and so installed both. So far we have made satisfactory progress, implementing our web server and database now gives us foundations to build upon. We now move our focus on actually developing web pages and then hopefully connect our web server to the database.

Iteration 2

In this iteration, we will be focussing on the following criteria:

- Creating our web pages
- Establishing a link between our server and the database

Creating our web pages

Firstly, we need to serve a web page. We created a javascript file from which we will use to run our website:

```
var express = require ('express')
var bodyParser= require ('body-parser')
const app = express()
const port = 8000
app.use(bodyParser.urlencoded({ extended: true }))

// new code added to your Express web server
require('./routes/main')(app);
app.set('views',__dirname + '/views');
app.set('view engine', 'ejs');
app.engine('html', require('ejs').renderFile);

//////////////

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

Figure 3.1: Code used to run the website

This set of code allows us to get access to the express module in node.js and get access to the body-parser to be able to hand post data requests. Because we will have several html files it is not

advised to include html inside the server code. Instead we created separate html files and we use templating engines to handle these html files. This is the purpose of the middle block of code. We created two directories, one called "views" to store html files and the second one to store routes called "routes". Views and routes are both subdirectories of the main directory called “project”.

We created an initial search html file:

```
<!doctype html>
<html>
 <head>
    <title>This is the title of the webpage!</title>
  </head>
  <h1> This is Search page </h1>
  <body>
    <p>Search for what you want.</p>
<form action="/search-result" method="GET">
 <input id="search-box" type="text" name="keyword" value="Default">
 <input type="submit" value="OK" >
 </form>
  </body>
<a href='./'>Home</a>
</html>
```

Figure 3.2: Search.html code

And in the main.js file located in routes:

```
module.exports = function(app) // this file has been exported as a function

    //get requests to the pages and res.render creates html output
     app.get('/',function(req,res){
        res.render('index.html')
     });

    app.get('/search',function(req,res){
        res.render("search.html");
     });
```

Figure 3.3: Main.js code

This is how the the website looks when run:

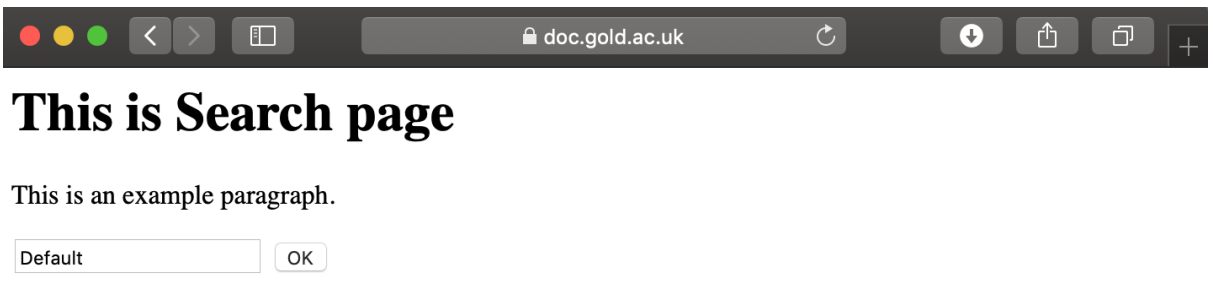


Figure 3.4: Search page

Establishing a link between our server and the database

Our search page is up and running but our website is not connected with our database hence making the page currently redundant. This is where our problems arose. Despite countless hours of research, we struggled with finding a method which would allow us to connect to our database through node.js.

For example, with the help of the guidance from the MongoDB developer page, we tried this method:

```
async function main(){
    const client = new MongoClient(uri);
    try {
        // Connect to the MongoDB cluster
        await client.connect();
        // Make the appropriate DB calls
        await  listDatabases(client);
    } catch (e) {
        console.error(e);
    } finally {
        await client.close();
    }
}
main().catch(console.error);

async function listDatabases(client){
    databasesList = await client.db().admin().listDatabases();
    console.log("Databases:");
    databasesList.databases.forEach(db => console.log(` - ${db.name}`));
};
```

Figure 4.1: Not functioning code for connecting database and website

Here, we use try...catch statement that marks the block of statements to try and specify a response should an exception be thrown. This is paired with async functions to ensure the function only commits when the conditions are met, but this way only prints out the name of our databases in the console and has no relevance with the website while countless other pieces of code only gave us errors. Ultimately, what gave us our breakthrough was Stack Overflow:

```
const mongoose = require('mongoose');
mongoose.connect("mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test");
mongoose.connection
 .once('open', () => console.log('Good to go!'))
 .on('error', (error) => {
 console.warn('Warning', error);
 });
// Express web server
```

Figure 4.2: Functioning code for establishment

We required mongoose which is an Object Data Modeling library for MongoDB and Node. It is a method which allows us to connect to MongoDB. We needed to install mongoose as a dependency first and then the connection was successful.

This method was very different to what we tried at first:

```
var MongoClient = require('mongodb').MongoClient; // retrieves mongo client
var url = "mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test"; // sets url

MongoClient.connect(url, function(err, db) { // connect to the db
  if (err) throw err;
  console.log("Database created!");
  db.close();
});


// Express web server
require('./routes/main')(app);
app.set('views',__dirname + '/views');
app.set('view engine', 'ejs');
app.engine('html', require('ejs').renderFile);
```

Figure 4.3: Code for showing our previous unworking methods

This code only retrieves the mongo client located on the express server while mongoose retrieves mongo databases outside of the server. And so with Fig 4.1, the connection url always produced unsuccessful connections. However, in the end we did get the website and the database to link and now we needed to verify if this was actually the case. So we created an ejs file which should print out all the games from the Steam collection:

```
<h3>You can see names and prices of all available games here:</h3>
<ul>
  <% availablegames.forEach(function(game){ %>
  <li><%= game.name %>, £<%= game.price %></li>
 <% }) %>
</ul>
```

Figure 4.4: Ejs file used to show the list of games

And then amended Search in main.js by creating a GET request which is used to send data and then we use res.render to output our ejs file:

```
app.get('/search-result', function (req, res) {
      var MongoClient = require('mongodb').MongoClient; //retrieve
     var url = "mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test"; // set url
     MongoClient.connect(url, function (err, client) { //connect to the db

      var db = client.db('GameSpy'); // name of db

      db.collection('Steam').find({name:{$regex:new RegExp(req.query.keyword,"i")}}).toArray((findErr, results) => { // finds name of games with specific key words

     if(findErr) throw findErr // outputs error

      else

       res.render('list.ejs', {availablebooks:results}); // otherwise produce output

         client.close(); // closes all open connections

    });

   });
});
```

Figure 4.5: Search result code in main.js

GET is used to retrieve and request data from a specified resource in a server. GET is one of the most popular HTTP request techniques. It is used to retrieve whatever information is identified by the Request-URL. We first identify where to retrieve our database to which the server should connect to and then we use db.collection to select the collection we want to use.

```
.find({name:{$regex:new RegExp(req.query.keyword,"i")}}).toArray((findErr, results) => {
```

Figure 4.6: Searching based on keyword

While this line of code searches for what the user types in and produces results for any given characters. For example, if 'c' is inputted, the result will include all games containing 'c. This is done by using new RegExp which is a Javascript constructor. And .toArray retrieves all those elements which are then output in our res.render which complies our function.

We used the same function for our list page GET request but did not use the RegExp constructor as that is not required to print all the games.

```
db.collection('Steam').find().toArray((findErr, results) => { // produces all available games
if (findErr) throw findErr;
else
   res.render('list.ejs', {availablegames:results});
```

Figure 4.7: List code in main.js

The output when run is:

Figure 4.8: List of games produced when run

This confirms that we have successfully connected our database to our website. This iteration was evidently the most difficult and we did not expect to face these kinds of issues early into the

project; however, we are pleased that in the end we overcame the complications and hopefully we can use this experience to assist us in any further ones.

Iteration 3

In this iteration, we will be focussing on the following criteria:

- Adding css to the website
- Adding a further database collection

Adding css to the website

As a starting point, we used our own css resources, previously used on other individual projects. However, when pasted into VSCode, all the lines of code got put into a single line as shown here:

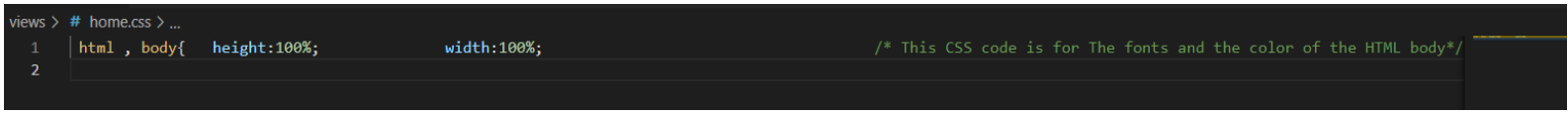
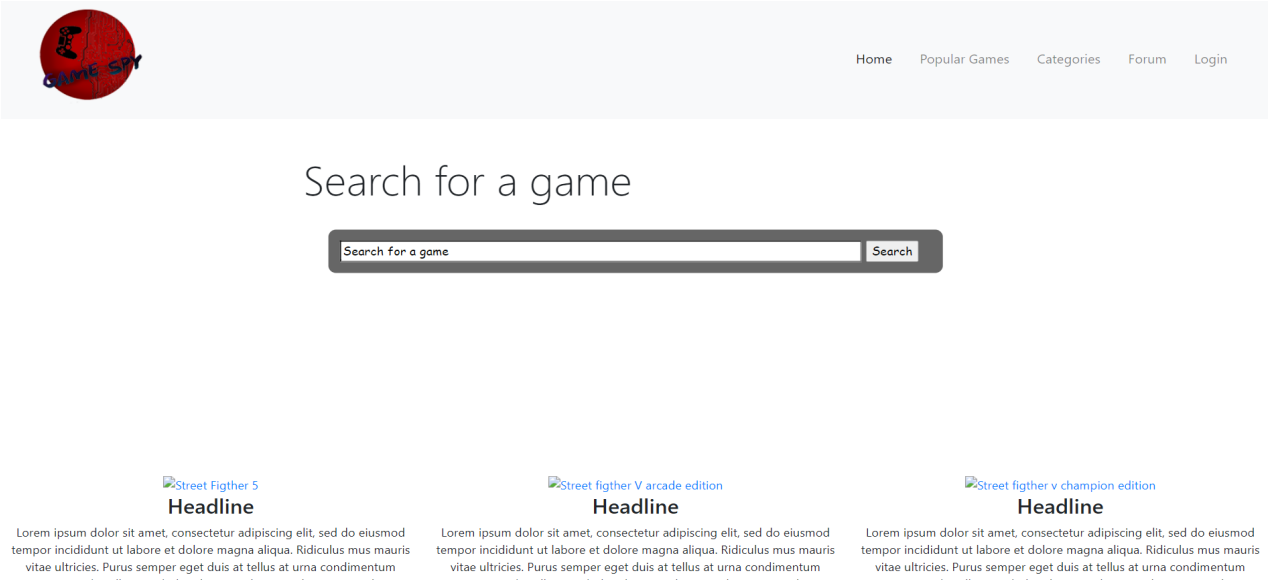


Figure 5.1: Unusable css code

Noticeably, this is something we would need to address but at this point we just wanted to see how the website looked:



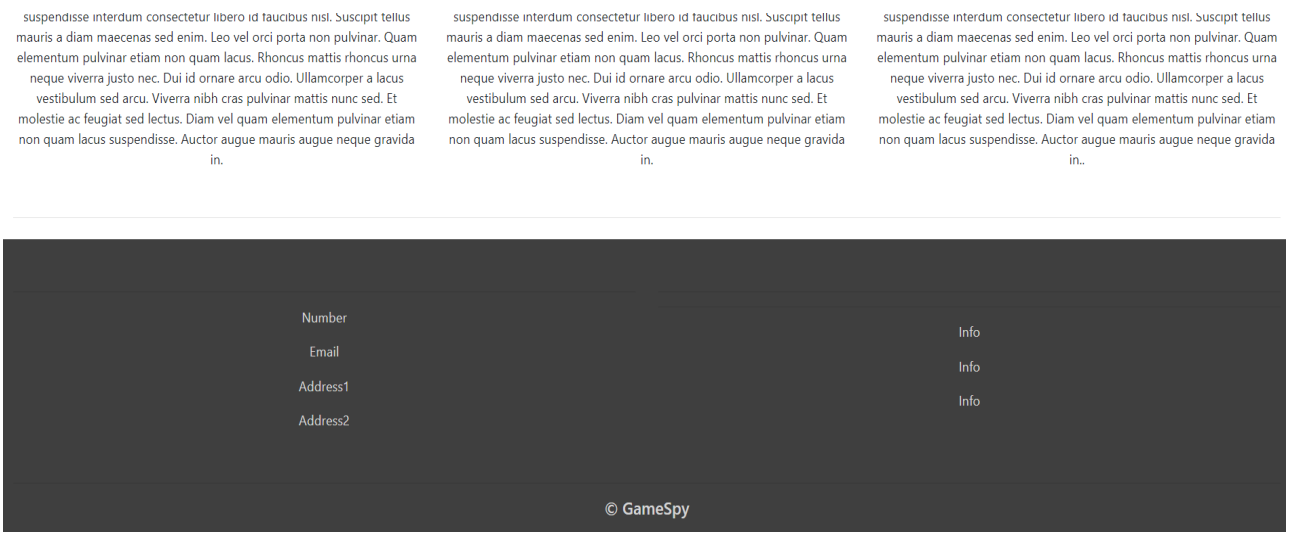


Figure 5.2: Homepage

We have the search bar being the main standout in the page since this is the most crucial so it is centered and larger than everything else. Below we will have a list of selected handpicked games with their game image and some descriptive text. Our idea is to decouple our content into large sections with an emphasis on cleanliness and legibility.

The visual layout is quite clean and simplistic but our views are not of importance, so we decided to host a short feedback session to see if this is worth working from. We gathered the contact details from those who filled out our questionnaire and presented it on a Discord call.

The main points we got were:

- The simple UI requirement seems to have already been met at this stage which is great
- The logo and font used on the search page seems to be consistent and is a major factor to making it simple
- The footer may need to be adapted much more but understand everything is at its initial stage
- The content on the search page also has to be improved further perhaps by making it smaller and more minimalistic.

We've received a great amount of feedback and can now work on this in the next iteration.

Adding a further database collection

We currently only have Steam as our only set of games available which clearly isn't enough so we searched through datasets on the internet for other readily available data. There didn't seem to be any data which included games from the big developers such as Ubisoft, Origin and Epic Games so we settled with GOG on Kaggle:

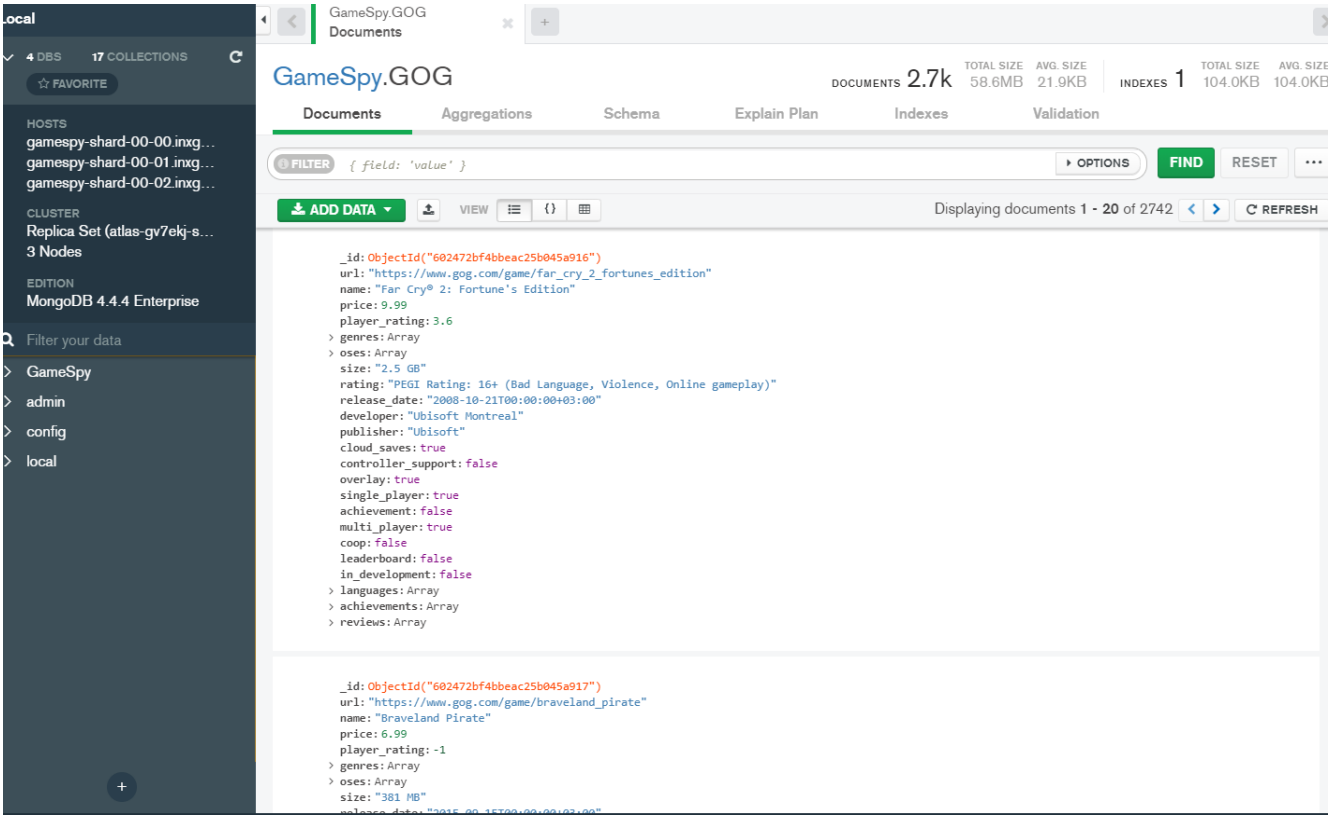


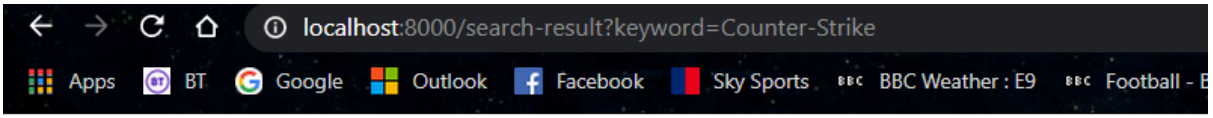
Figure 6.1: GOG collection in our database

To check if GOG games appeared in our search result we added a line of code which should hopefully do this in our /search-result in main.js:

```
db.collection('Steam').find({name:{$regex:new RegExp(req.query.keyword,"i")}}).toArray((findErr, results) => {
db.collection('GOG').find({name:{$regex:new RegExp(req.query.keyword,"i")}}).toArray((findErr, results) => {
if(findErr) throw findErr // outputs error
```

Figure 6.2: Adding a further db.collection.find in our code

We searched for a game which is only available on Steam:



This is List Page

You can see names and prices of all available games here:

Home

Figure 6.3: List page with nothing outputted

But nothing was outputted, whereas when we searched for a game which is only available on GOG that did work. So this is something we need to solve.

In this iteration, we did not make the best progress in terms of development but we did gather valuable information in terms of visualisation which is guaranteed to help us in further iterations. Our next steps is to solve our problem with the multiple collections search result and to ensure our css is readable and editable.

Iteration 4

In this iteration, we will be focussing on the following criteria:

- Fixing search results
- Adding further html pages
- Reconstructing the css file

Fixing search results

Looking at the MongoDB Developer Hub, we studied the documentation and the best way of operating the search results would be through aggregation. This is used to process the data that returns the computed results. It groups the data from multiple documents and returns into one combined result.

```
dbo.collection('Steam').aggregate([
  { $lookup:
    {
      from: 'GOG',
      localField: '_id',
      foreignField: '_id',
      as: 'name'
    }
  }
  ]).toArray(function(err, result) {
  if (err) throw err;
  //console.log(JSON.stringify(res));
  res.render('Game.ejs', {pagetitle:"Game",availablegame:result});
  db.close();
```

Figure 6.4: db.collection.aggregate to amend search results

$lookup takes the documents in the collections and uses the fields to select the data which then outputs the data using in this case ‘name’. Yet when this is run we only get the prices of the games:

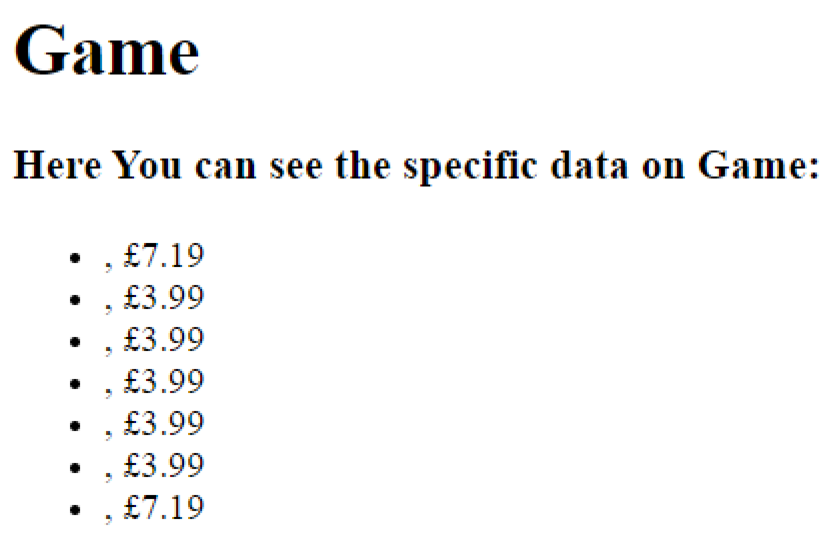


Figure 6.5: Only prices being outputted this time - different from Fig 6.3

But the prices of games that get produced are once again from a single collection which we cannot understand why as we followed the syntax line by line. A stumbling block once again.

Adding further html pages

The popular games page was the next page we developed:

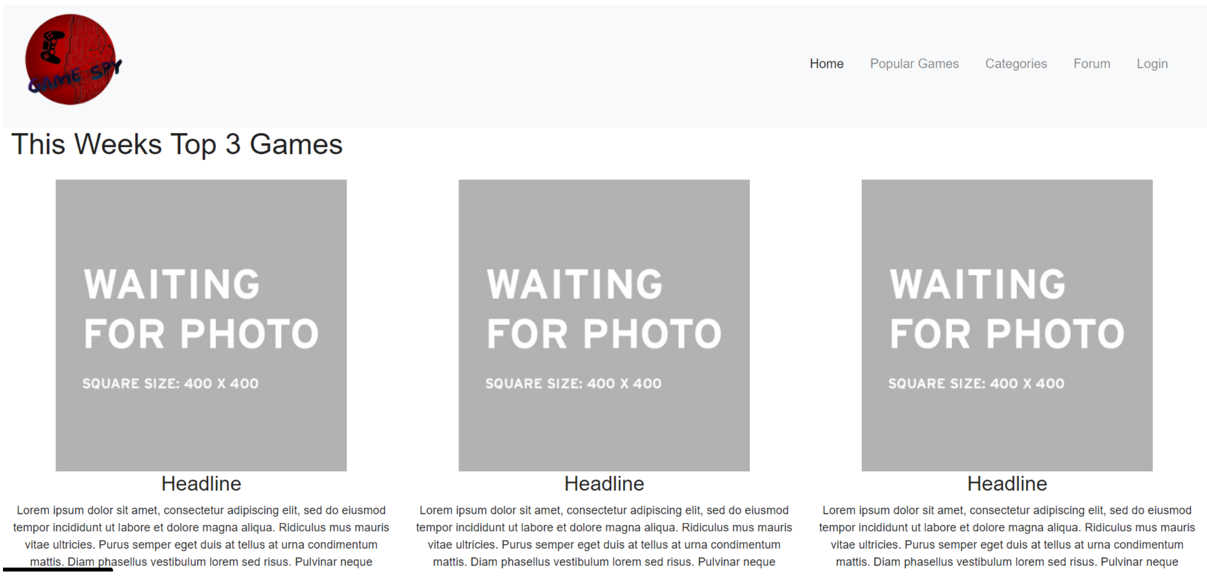


Figure 7.1: Popular game page

Using multiple div classes as placeholders so that we can easily place our content in:

```
<h1>This Months Top 3 Games</h1>

<div class="row text-center padding">
    <div class="col-s-10 col-xs-1 col-md-4">
        <a href="game1.html"><img src="https://www.mountaineers.org/images/placeholder-images/placeholder-400-x-400/image" alt="Street Figther 5" class=
        <h3>Headline</h3>
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ridiculus mus mau
    </div>
                        <!--- Link for the second game with the image -->
```

Figure 7.2: Our placeholder code in populargames.html

We would obviously need to get some user feedback so this is something we will try to do once we create further html pages.

Reconstructing the css file

This was completed.

Slight progressions so far but still a lot of work to do. We still need to figure out our search results so this is something we may try to get some guidance on if we cannot do ourselves. We will continue developing our front end so we are not completely at an impasse.

Iteration 5

In this iteration, we will be focussing on the following criteria:

- Fixing search results - Part 2
- Adding further html pages

Fixing search results - Part 2

We continued to try and fix search results but we failed to document our coding method attempts - as of yet this is something we haven't accomplished.

Adding further html pages

The login and register pages were the next to be developed:

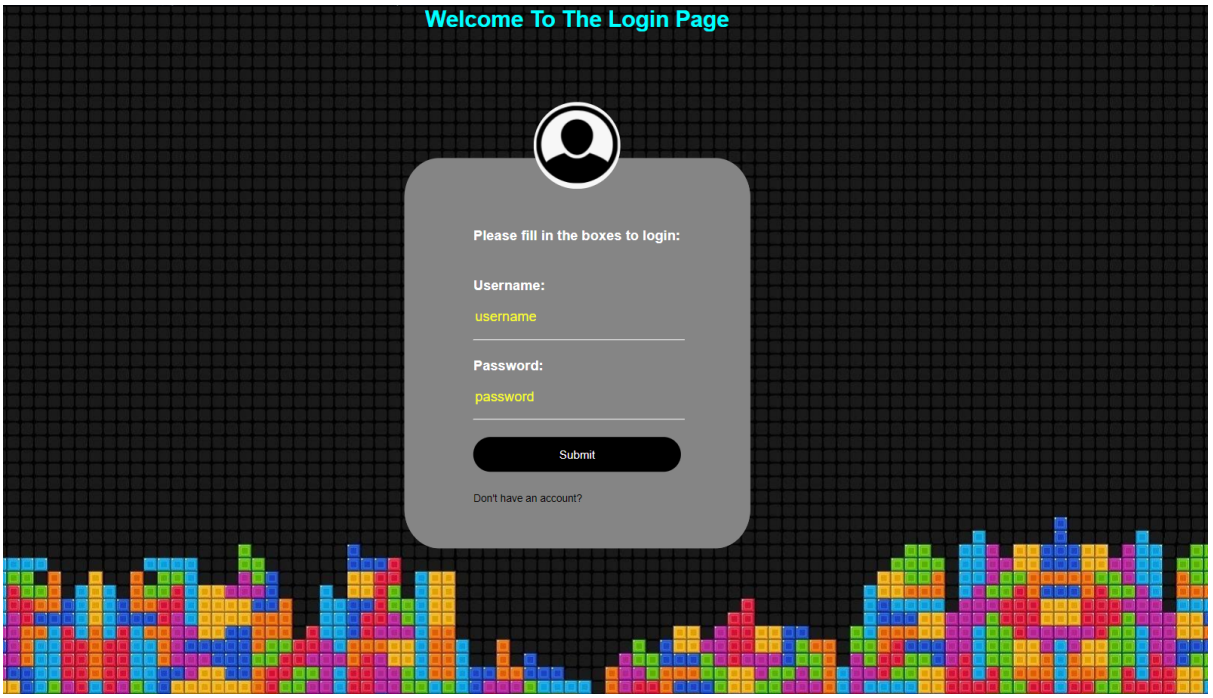


Figure 8.1: Login page

If the user does not have an account they can click the "Don't have an account?" button which leads them to:

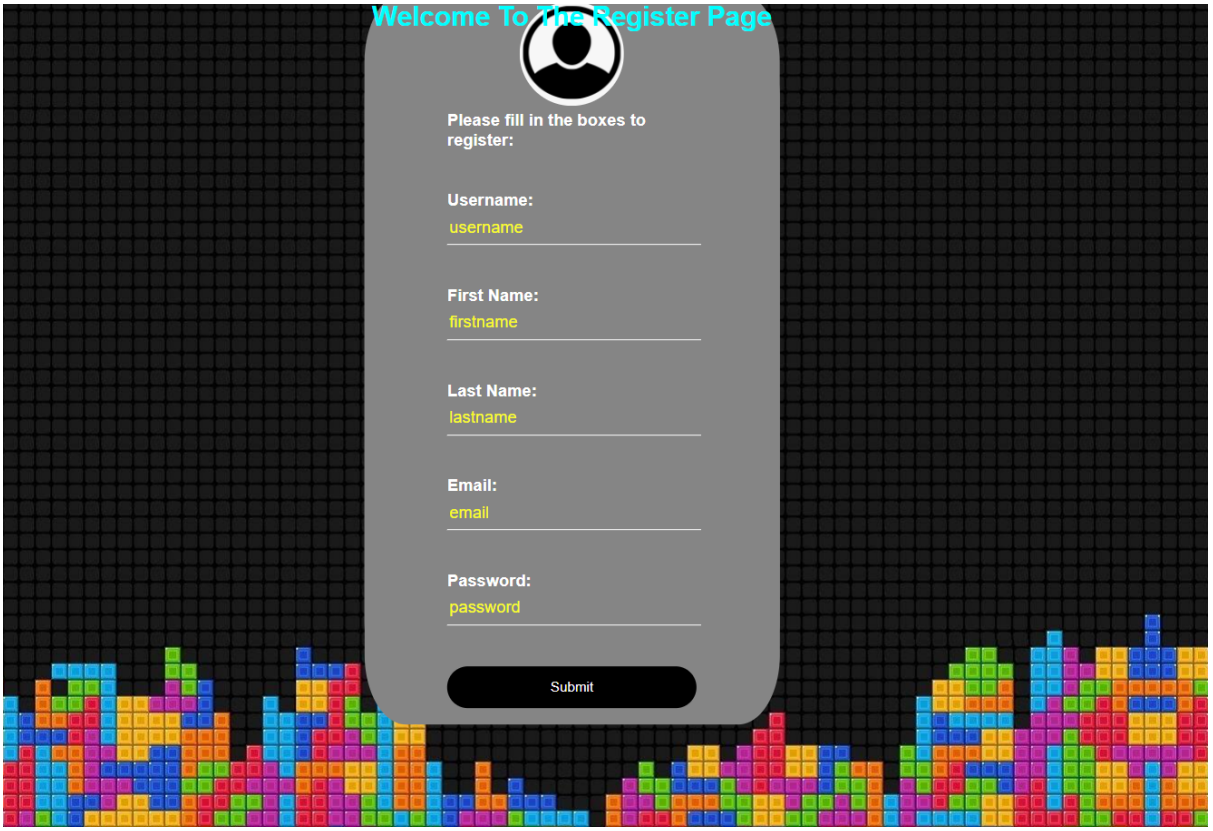


Figure 8.2: Register page

We designed our register and login pages like this to put an emphasis on retro and essentially give a feeling of nostalgia for the avid gaming fans. We believe this static design is what our users would want to see in gaming related sites.

Currently, there isn't an app POST method for when the user submits their details as we first want to ensure these pages are suitable for our users which is why we called for another feedback session.

We first presented our Popular Games page and the main points that were discussed were:

- Clean and simple UI once again - good consistency so far
- Looks quite bland - colours are scarce
- Good layout of how the games will be

We then presented our Login and Register pages and a major design flaw that was pointed out was how much of a stark contrast the pages are. Login and Register are very vibrant yet the rest are quite boring. We need to find a good balance between the two. In addition, the fonts and colours used need to be changed as our users found themselves squinting to read the text.

Once again not a lot of progress has been made in terms of back end but we're very happy with the front end especially due to how great the feedback we received was. We now want to ensure that once the user registers, their data is stored on our database so they can also login. Search results is also what we will be focusing on.

Iteration 6

In this iteration, we will be focussing on the following criteria:

- Fixing search results - Part 3
- Storing user details into the database

Fixing search results - Part 3

Once again, we continued to try and fix search results but we failed to document our coding method attempts - as of yet this is still something we haven't accomplished.

Storing user details into the database

We first created a new collection called "users" in our database in MongoDB Compass and then added a POST method route called "registered" in our main.js:

```
const bcrypt = require('bcrypt'); //hashing password
const saltRounds = 10; //random values
const plainPassword = req.body.password; //plain password set to password
```

Figure 9.1: How we store the user inputted details

"bcrypt" is a library for NodeJS and allows us to hash the passwords for the user - this is required to help us comply with privacy and safety laws. We installed that into our express server and then we used saltRounds to declare the cost factor which is how much time is needed to calculate a single BCrypt hash.

This is the method for the hashing and storing of the data:

```
bcrypt.hash(plainPassword, saltRounds, function(err, hashedPassword) { //method for all const
//Store hashed password in your database.
  MongoClient.connect(url, function(err, client) {
  if(err) throw err;
  var db = client.db ('GameSpy');
    db.collection('users').insertOne({ // collection and documents
      first: req.body.firstname,
      last: req.body.lastname,
      email: req.body.email,
      username: req.body.username,
      password: hashedPassword
    });
```

Figure 9.2: Hashing and storing of data

We store user details using the req.body object which allows us to access data in a string from the client side. This is stored using the id from our register.html.

The data once inputted will look like this in our database:

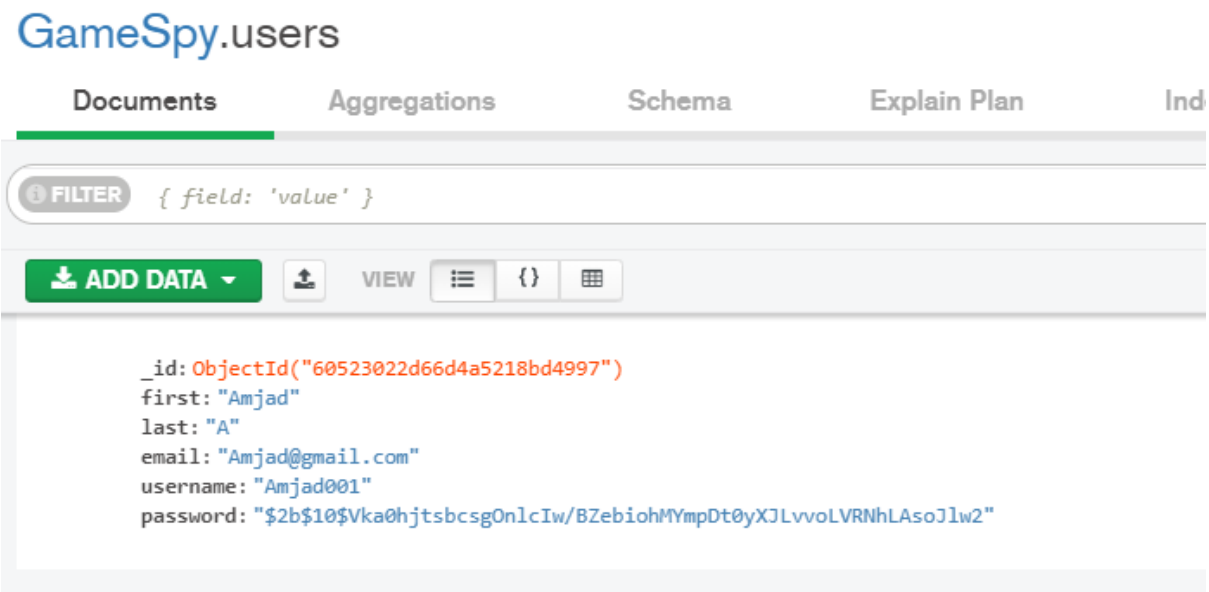


Figure 9.4: How we store the user inputted details in our database

As shown the password is hashed, which means no one can know the user password except for the user themselves. Once this is successful the user will be able to login and will be met with this test message:

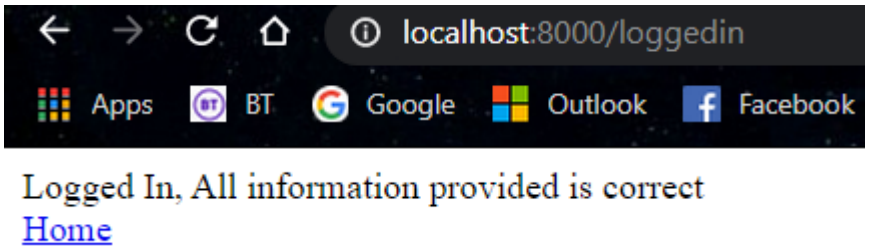


Figure 9.5: The message shown when user has successfully logged in

We have essentially finalised our form pages but still need to revamp the design of them. Still no breakthrough for the search results but we will continue testing and trying methods.

Iteration 7

In this iteration, we will be focussing on the following criteria:

- Fixing search results - Part 3
- Adding further html pages

Fixing search results - Part 3

Once again, we continued to try and fix search results but we failed to document our coding method attempts - as of yet this is still something we haven't accomplished. However, we had

thought of a plan which we would use if we could not get it to work but it is a last resort. Our method would be to create more collections in our database which will only include the names and prices of all the games and then get our GET method to only render that collection. This isn't the most efficient way of achieving this which is why we won't be implementing it unless we have to.

Adding further html pages

Our forum page was next to be developed, we used the feedback from our fidelity design which we presented to our stakeholders in our proposal:

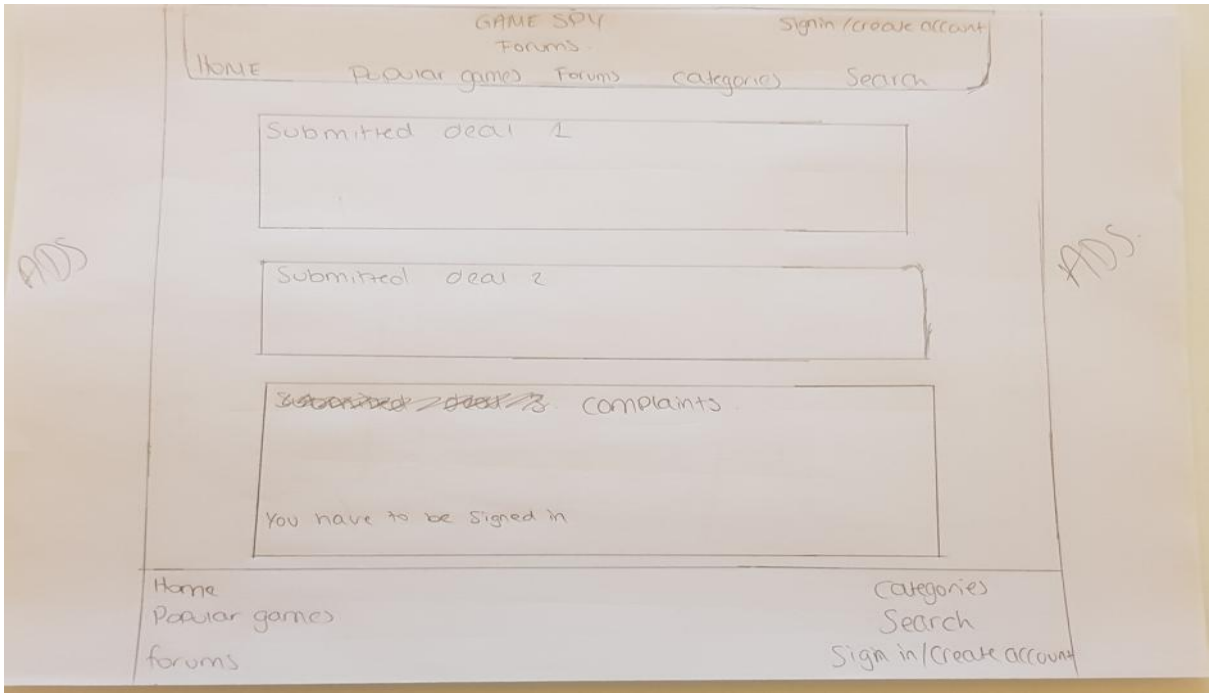
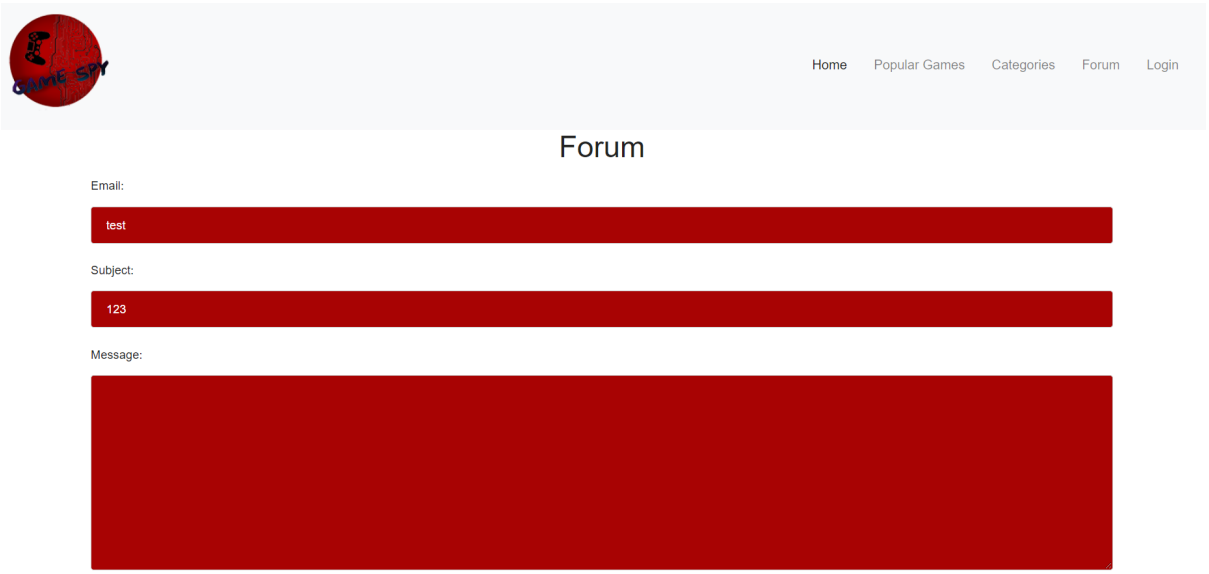


Figure 10.1: Low fidelity forum page

They said they disliked that we had deals and complaints so we improved on that and just kept the page less complicated:



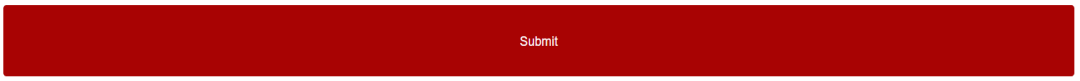


Figure 10.2: Forum page when run

The colours on the forum pages were chosen because we wanted it to link with our logo. The red background is great especially if you match it with white text as research suggests “white is light and bright and will sharpen any shade of red. It is a great base when making a bold statement, offering contrast and also some breathing room”. The message box is spacious enough for lengthy texts but can also be enlarged if required. With the submit button we have currently made it big so that it can be accessed for all.

As stated previously, all current designs are not at its final stage, we will continuously adapt all pages if and when needed based on user interactions. Therefore, this will be presented at our next user feedback session.

Iteration 8

In this iteration, we will be focussing on the following criteria:

- CSS changes

CSS changes

At this stage of the development process, we decided to make a few minor changes to the design of the website. Mostly removing redundant code and changing the colour of the forum page. We internally agreed that having dominantly red on our forum page isn't beneficial for anyone and so we changed it to grey. The total changes we made to our files can be seen here:

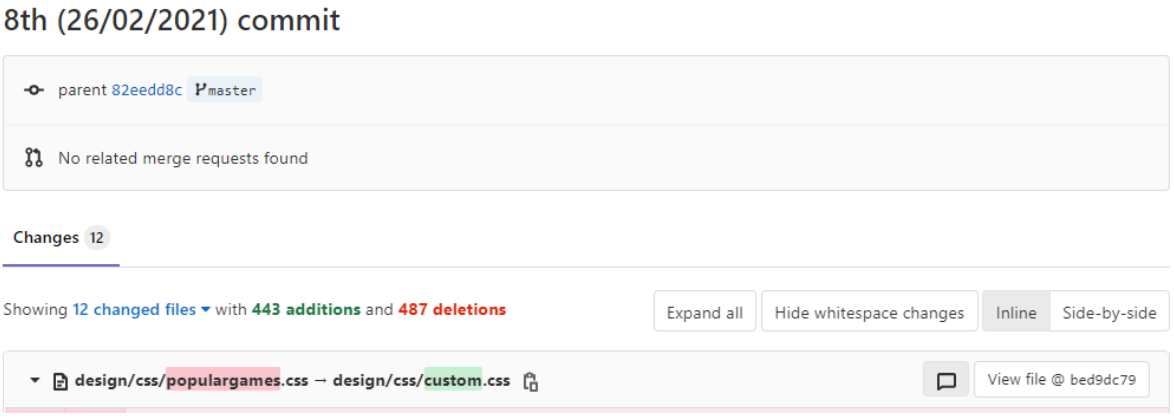


Figure 11: Commit changes on our gitlab page

Iteration 9

In this iteration, we will be focussing on the following criteria:

- Fixing search results - Part 4

Fixing search results - Part 4

After many iterations, coding attempts and multiple calls for help, we decided to resort to our ultimatum. As stated previously, the plan is to create another collection which will contain the names and prices of all the games. Usually each individual collection will have numerous field types but we only need the name and price since this is all the user will require when quickly searching for games.

To do this, we used our .csv and .json and copied the names and prices of those games into another .csv file. This was then uploaded to MongoDB Compass. At this point we only had Steam and GOG as our only two collections so this process was quite simple. We then changed the db.collection.find in main.js to the name of our new collection:

```
db.collection('ALL').find({name:{$regex:new RegExp(req.query.keyword,"i")}}).toArray((findErr, results) => {
```

Figure 12: Changing the collection name - different to Fig 6.2

And now when we test for a game only available on either Steam and GOG, the output is what we would expect.

Although we are extremely relieved that we have finally got this working, we are unsure if this will be an obstacle in the future. With this completed we can now focus towards the other areas of the website such as finding more datasets of games and adding more web pages.

Iteration 10

In this iteration, we will be focussing on the following criteria:

- Adding individual game pages
- Adding new game collections to the database

Adding individual game pages

At this stage of development, we have our list of games to view on the website. So now we need to be able to ensure that those games have separate pages for the users to view. To do this we created an ejs file. We chose ejs over html because we need to be able to create a dynamic page which ejs provides us with. Every game has different data and outputs and since we have data from our database that needs to be rendered to HTML, we can use the ejs template engine to take the data and a template and render it to the HTML instead which then gets served to the client.

We created a file called gamepage.ejs and inside we embedded player count information taken from SteamDB:

```
<iframe src="https://steamdb.info/embed/?appid=730" height="389" style="border:0;overflow:hidden;width:100%"></iframe>
```

Figure 13.1: Embed data from SteamDB for a game

This currently gets all player related data from SteamDB and outputs on our web page for the game Counter-Strike: Global Offensive:

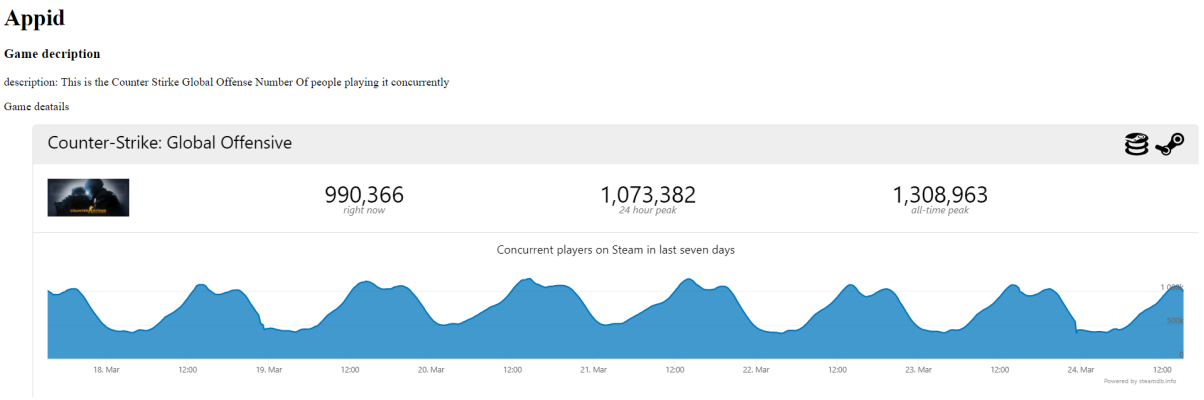


Figure 13.2: Player count of the game

This is currently third-party data but it is very likely to be reliable since it is data published by the developers of the game. One of our objectives was to include a live player count and we have currently achieved this; however, it is still only the framework and we want it to be advanced so it will be something we will carry out when possible but at this stage we will continue with this method.

Adding new game collections to the database

As stated previously in Iteration 3, there weren't many available datasets on the internet so instead we looked to create our own. We talked about web scraping in our proposal and there were many ways of going about this. One was building our own web scraping tool through Python where we could write a script that uses Python's requests library to scrape data. Or we could use software that could help us gather data in a quicker and efficient way. At this time of the development, with only a few weeks till release, we decided with the latter. This is because we feared there may not be enough time to create a tool from the ground up especially considering the many features of the website we have yet to develop.

We identified ParseHub as the best software considering it is free and easy to use. What was required of us was to direct to the website we wanted to scrape and then select the elements of the pages which then the software will automate the rest:

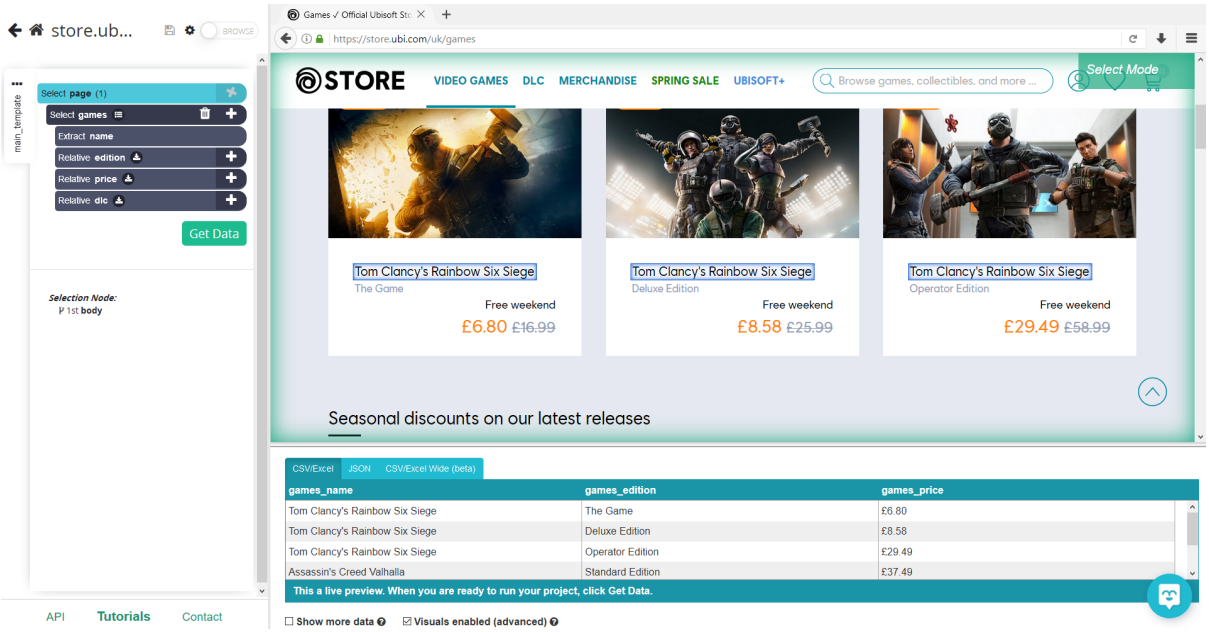


Figure 14.1 ParseHub - software for web scraping

As you can see, we set up the names, game editions, prices and whether it is a dlc or not. We select the games while scrolling through the page, then the software outputs that as either CSV or JSON. We can then easily download the file and upload it to our database:

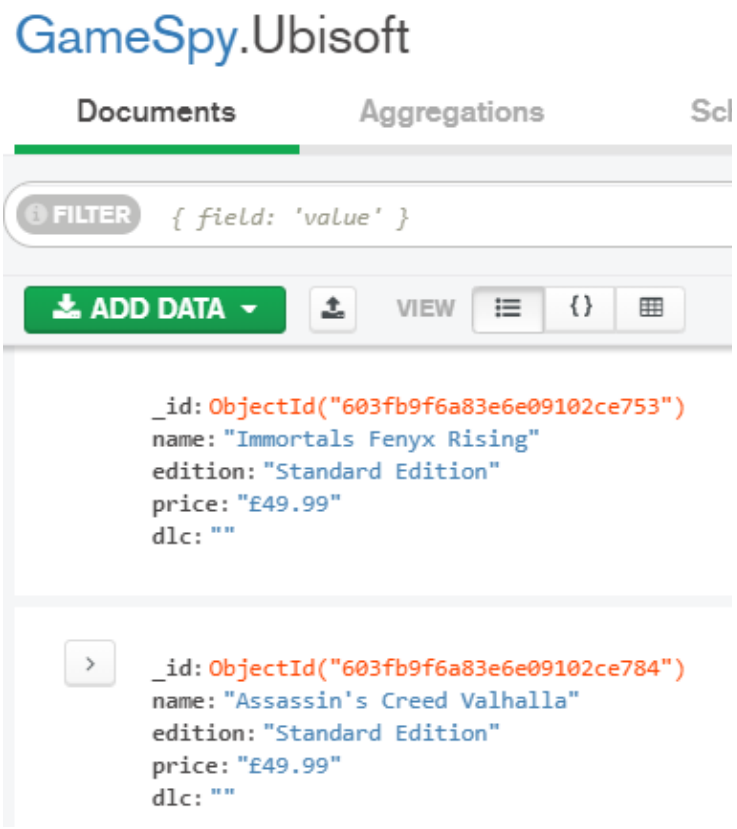


Figure 14.2: Ubisoft dataset taken from ParseHub

We continued using ParseHub for other game stores such as Epic Games and Origin. Once we gathered that we copied the names and prices of those games and put them in our ALL collection - this way they will appear when searched.

There is a much better progression at this stage of development with our backend being vastly enhanced while our front end is seemingly coming together. We now need to go further by creating more pages and improving on current ones.

Iteration 11

In this iteration, we will be focussing on the following criteria:

- Adding a complaints page
- Implementing access control & validation
- Changes to user complaints page

Adding a complaints page

At this stage of development, we made some design changes. We disliked how our forum page looked and believed that it does not serve the purpose of a forum. The page only had a form hence

it did not allow visitors to communicate with each other by posting messages. We want our forum to engage different groups of the same space together so they discuss some topic or give the opinion about something or provide useful information that can help another person or group and solve their problems or doubts. Therefore, we changed our forum page to complaints and then created a complaints page just for admins. We will go on to describe what we mean below.

Before creating the admin complaints page we decided we would use EJS again instead of HTML since we are going to output HTML with a lot of javascript involved.

```
<div class="ptable">
<table class="table">
    <tr class="col">
        <th>_Id</th>
        <th>Email</th>
        <th>Subject</th>
        <th>Message</th>
    </tr>
    <% availableuser.forEach(function(user){ %>
    <tr class="wpos">
        <td><%= user._id %></td>
        <td><%= user.Email %></td>
        <td><%= user.Subject %></td>
        <td><%= user.Message %></td>
    </tr>
```

Figure 15.1: Structuring the data taken from the complaints page

We structured the data which is submitted from the user complaints page into a table. For example, for user.Email, 'user' is the name of our collection in the database while Email is the data that is pushed from the user complaints page. All of that data then appears on the admin complaints page:

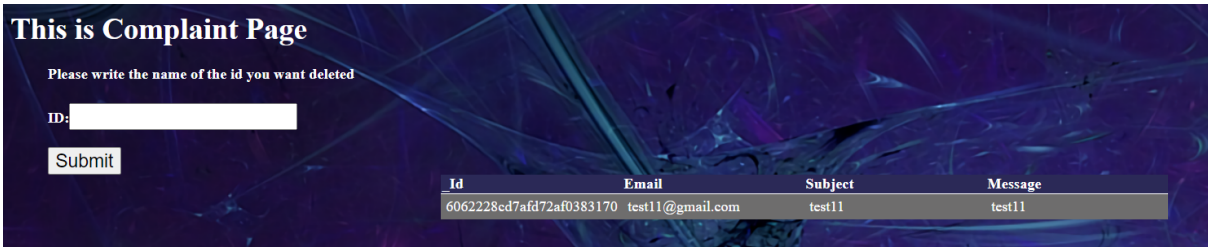


Figure 15.2: Admin Complaints page

Implementing access control & validation

To prevent unauthorised users from viewing this page, we implemented an access control which is a way of controlling what resources a user can create/read/update/delete given their role. How this will work is that users will be required to be logged in to be able to access certain pages. This will be done by using their login id which will be saved onto the server:

```
const redirectLogin = (req, res, next) => {

  if (!req.session.userId) {
    res.redirect('./login')
  }
  else {
    next();
  }
}
```

Figure 16.1: Access Control

This block of code checks whether the user is logged in and if they are not, the page will redirect them to the login page for them to access their desired page to view by adding redirectLogin to our GET method in main.js:

```
app.get('/Forum', redirectLogin, function (req, res) {
```

Figure 16.2: Redirects the user

It redirects them by using the parameter next which is a function that executes the next part of our middleware.

Additionally at this stage we are also starting to implement express validation. This is where we check to ensure that any data entered is sensible and reasonable. Where there are any form inputs is where there is a need for validation as we should consider that any inputs a user makes may be incorrect and should plan arrangements for such unexpected actions.

Firstly, we installed the validator module on to our server and then imported the module in to our index.js file:

```
var validator = require('express-validator');
```

Figure 16.3: Adding the module to index.js

And in our main.js file we then declared it:

```
const { check, validationResult } = require('express-validator');
```

Figure 16.4: Declaring the validator as a const

With the validation module now set up, we used the Express-Validator tutorial to help us determine which validators would work best. For our login page, we added this to the route:

```
app.post('/loggedin', [check('username').notEmpty(), check('password').notEmpty()], function (req, res) {
```

Figure 16.5: Validating login route

This tells the system to check whether both username and password are not empty and if it is; when the user submits, the page will just refresh and tell the user to submit their details properly.

Using validation helps us to ensure that any data input is possible and sensible.

Changes to user complaints page

Previously, as shown in Fig 10.2, we had our then “forum” page as mainly red with a white background. Taking into account when our users stated we needed to find a balance between vibrance and dullness, we decided to change that appearance:

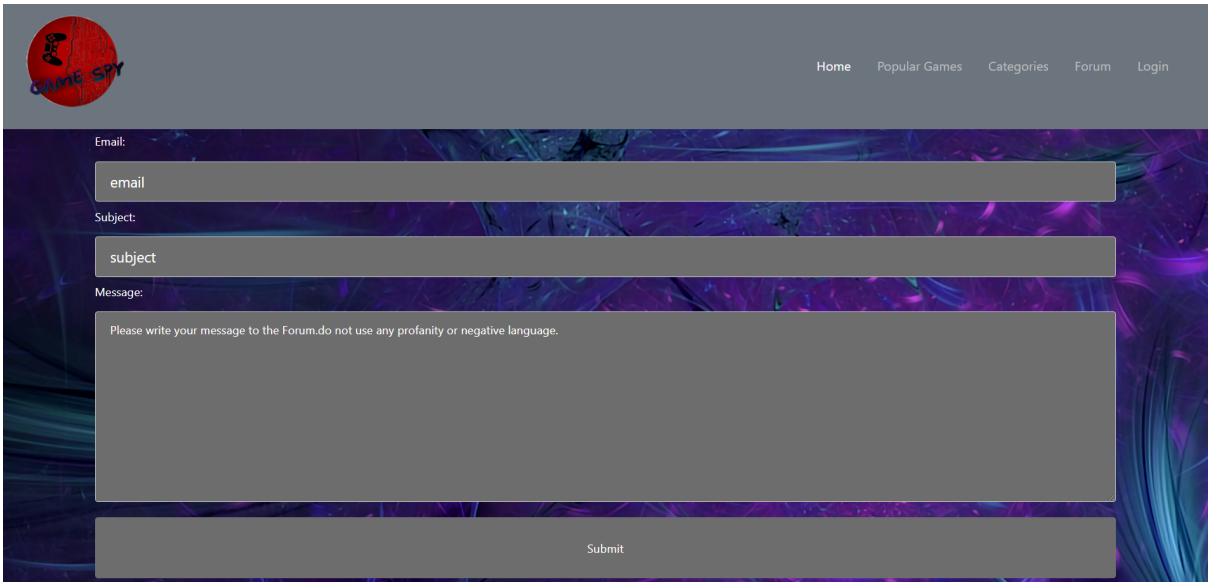


Figure 17: Complaints page

Our background has undergone a full change with us using a much more lively wallpaper but at the same time bringing back a grey navigation bar and form colours. We believe this is the balance of colours our users are looking for but will once again showcase this in our next presentation.

At this junction, we believe everything is coming together quite well. If we review our progress so far, we have achieved 3 out of 6 of our features and objectives: player base over time since release, live player count and community reviews. However, the most difficult features are still yet to be worked so we still have a lot to be done.

Iteration 12

In this iteration, we will be focussing on the following criteria:

- Redesigning current web pages
- Implementing search autocomplete
- Feedback session

Redesigning current web pages

After making the design changes to our complaints page, we went on to change all our other pages. All pages will now be consistent with each other which reduces the user issues of pages being contrastingly different to each other which gave the impression they were on completely different websites. (Iteration 5)

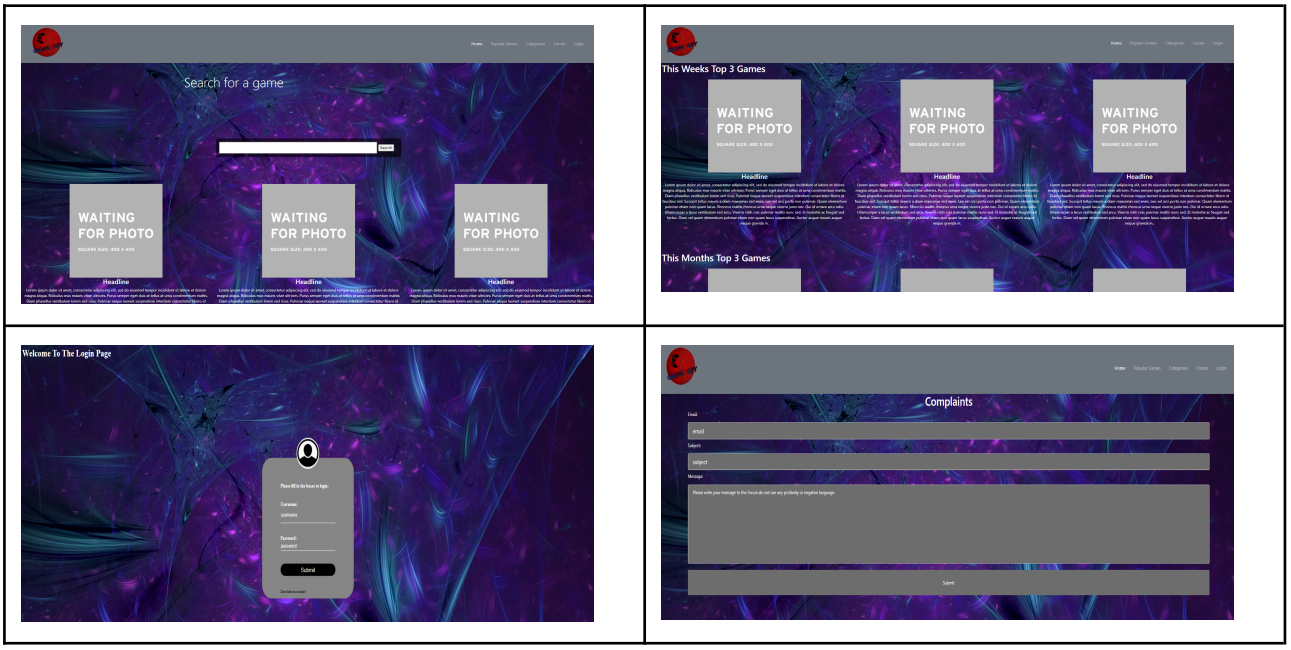


Figure 18: Table showing our pages now revamped

The content itself is the same but instead we changed all the backgrounds and use the same colours across the website. We now want to present all of this in our next feedback session.

Implementing search autocomplete

When searching for games, we wanted users to quickly complete searches that they're beginning to type. There are many ways we could have gone on to do this such as using Elasticsearch which allows us to store, search, and analyze huge volumes of data quickly and in near real-time and give back answers in milliseconds. It does this by instead of searching the text directly, it searches an index. It is basically a server that can process JSON requests and give you back JSON data. Instead we opted for jQuery Autocomplete which achieves the same goal as Elasticsearch but is easier to implement.

We used jQuery UI to help us - this provides an autocomplete widget to facilitate users by giving a list of suggestions to type in a text box. It is a control that acts a lot like a <select> dropdown, but filters the choices to present only those that match what the user is typing.

We followed the syntax and created a query which should create an autocomplete function.

We first had to set the id of our search form in search.html:

```
<input name="keyword" type="text" class="Searchbox" id="tags" placeholder="Search for a game...." value="">
```

Figure 19.1: Creating id for jQuery

And then the actual code:

```
$( "#tags" ).autocomplete({
source: gamename,
maxResults:10
});
```

Figure 19.2: Autocomplete with maximum number of results

especially since we have over 20,000 games in our database. However, this did not work - there The source here is the 'gamename' which is what is produced from our ejs file and then we set the maximum number of results as 10 to prevent the whole page being filled with predictive results were no search predictions and no errors were produced, so we studied the syntax further and realised we required a function to actually gather the names:

```
$("#tags").autocomplete({
        source: function(request, response) {
        var results = $.ui.autocomplete.filter(gamename, request.term);
 response(results.slice(0, 8));
```

Figure 19.3: Autocomplete function

We direct the function to what we want it to autocomplete using the id we set as "tags" and then we call the response function which passes the array that contains items to display and then we have our autocomplete. filter which will filter the desired elements from our games list being provided and callback the same function to extract more. results.slice(0,8) outputs only 8 searches so that screen doesn't get filled with search suggestions.

So now when it is run:

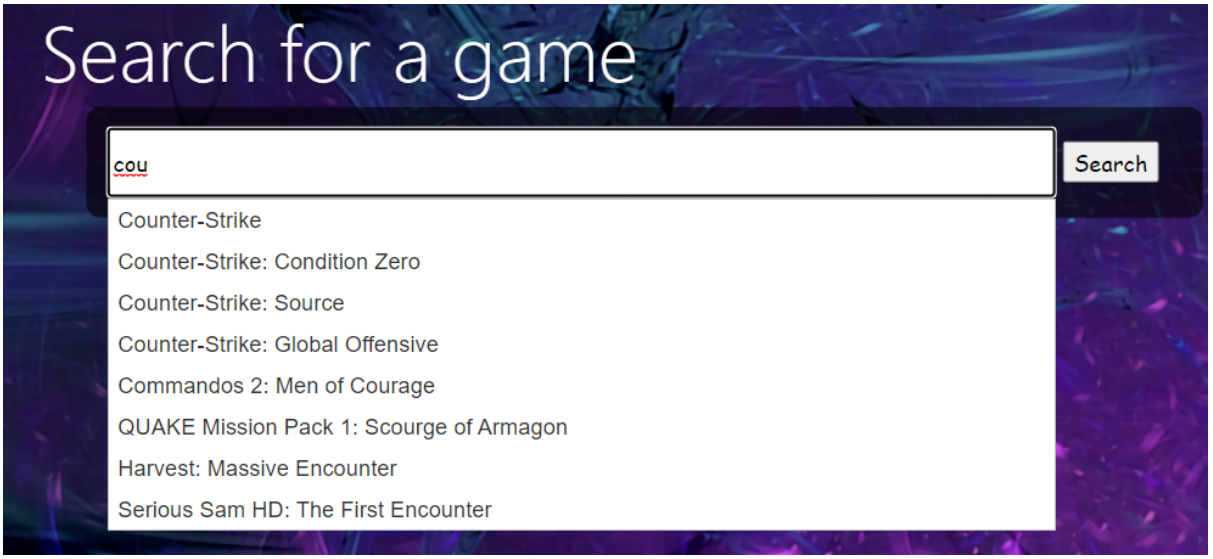


Figure 19.4: Autocomplete on the website

In the example above, you can see that typing the letters “cou” brings up predictions such as “Counter-Strike” or “Commandos 2: Men of Courage,” making it easy to finish entering their search on these topics without typing all the letters.

Autocomplete is especially useful for those using mobile devices, making it easy to complete a search on a small screen where typing can be hard. For both mobile and desktop users, it’s a huge time saver all around and according to Google it reduces typing by about 25 percent.

Additionally, we also changed how our search results now look. When the user continues their search without clicking any of the search predictions they will see the list of games and prices in a tabular format as we believe is the most simplest form of data representation:

Game Name	Game Price
Counter-Strike	7.19
Counter-Strike: Condition Zero	7.19
Counter-Strike: Source	7.19
Counter-Strike: Global Offensive	0
Counter-Strike Nexon: Zombies	0

Figure 19.5: Revamped Search results

However, the way we carried this out is causing a major problem. For the autocomplete to work, it has to read the data for what it should predict, so the way we did it was we pasted in all the names of the games from our database into our search.html file:

Figure 19.6: Names of all games in the code

This is a dilemma because in the end we are left with over 90,000 lines of code which not only makes it impossible to edit but also increases the loading time of our pages. We will need to fix this so this will be one of our focuses in the next iteration.

Feedback session

With most web page designs likely to be at the latest phase of its appearance, we thought it would be a good idea to host another feedback session especially since it has been a while since our last one. We once again mostly focussed on gathering user feedback on the visual design elements, such as fonts, colours, layout, and imagery. We presented what is shown currently in Fig. 18 and

carried out a preference test where we showed the user two design variations, our former and latest designs, and asked which they prefer and why.

	Former design (Fig 5.2, 7.1, 8.1, 8.2, 17)	Latest design (Fig. 18)
Good points	● "I like what you did with the header and nav, they're clean and to the point. So it's good you've kept that even with the latest design." ● "Layout is clean and not cluttered like many other websites you're trying to compete with."	● "Better user friendly especially with navigation bars (following the user when they scroll down or up)." ● "Better user friendly especially with navigation bars (following the user when they scroll down or up)." ● "I really liked the styling and setup of the home page and found the structure very appealing and helpful with the navigation to the top games."
Bad points	● "The remaining parts of the homepage and popular games page looks very bland and boring. Compare this to your login and register pages, it's like night and day"	● "The search bar needs to have more advanced options like searching by genre or maybe game studio." ● "Maybe think about adapting the colour of your navbar and footer a little, perhaps a bit more transparent or something"

We've received great valuable insights from our users which enables us to evaluate our product so far and now we hope to create a human-centric webiste with this. We will try to implement the user feedback while simultaneously developing the rest of our website.

Iteration 13

In this iteration, we will be focussing on the following criteria:

- Cleaning up our autocomplete code

<u>Cleaning up our autocomplete code</u>

Because we have all our 28,900 names of the games from our database embedded into the code, we were compelled to figure out another way of doing the autocomplete function. It is clear that if we continued with keeping 90,000 lines of code in our search.html file, it will reduce reusability which is a bad practice in software development. In order to reuse code, our code needs to be

high-quality and reliable which it is not currently. The best way of approaching this would be using NodeJS to read files. We decided to create a JSON file as it uses less data which increases the parsing speed. Additionally, the JSON structure is straightforward and readable all which are big pluses over file types such as XML especially since XML has no visual separation between the markup and the content, making it hard to read.

Once we setup the JSON file we also had to initialise it in our index.js file:

```
// JSON
const fs = require('fs');
const data = fs.readFileSync('list.json');
const gamename = JSON.parse(data);
```

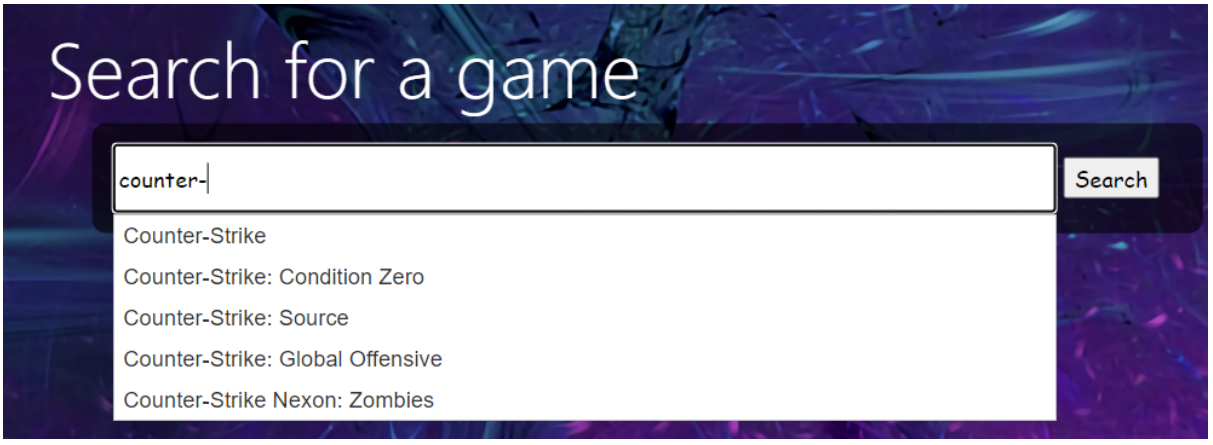
Figure 20: Initialising JSON file system

“const fs = require(‘fs’)” is a Node.js file system module which allows us to work with the file system on our local computer and is included by using the require() method. “fs.readFileSync” reads the full content of the file in memory before returning the data. And then “JSON.parse” parses the data for it to become a Javascript object.

And now when we run the server, these three lines of code do the equivalent of the 90,000 lines of code and in the end we have a much improved set of written code.

Now that our search code is all cleaned up, we can now move on to create our game pages. Our intent is once the user searches for the game they want, they can then select that game and then they will be taken to another page where they will get all the information they require before they purchase it. For example:

User searches game:



Search result brings up some games:

User clicks the game they want to view and are led to a page like this:

How we carried out populating our game page was discussed in Iteration 10. So far we only have one game page that is populated and with nearly 30,000 games in our database, it will be a huge struggle to complete everything. We want to try and implement something that will create multiple templates for us but we are unsure if the technology exists for it. Therefore, we will continue creating more game pages but at the same time research for methods of doing this automatically.

Iteration 14

In this iteration, we will be focussing on the following criteria:

- Cleaning up the entire code

Cleaning up the entire code

The last time we cleaned up leftover data structures and other unwanted materials from memory and the filesystem was in Iteration 8. We decided to do this again so that source code itself is easier to understand, maintain, and modify. These were some of the modifications we made to our code over some of our version controls:

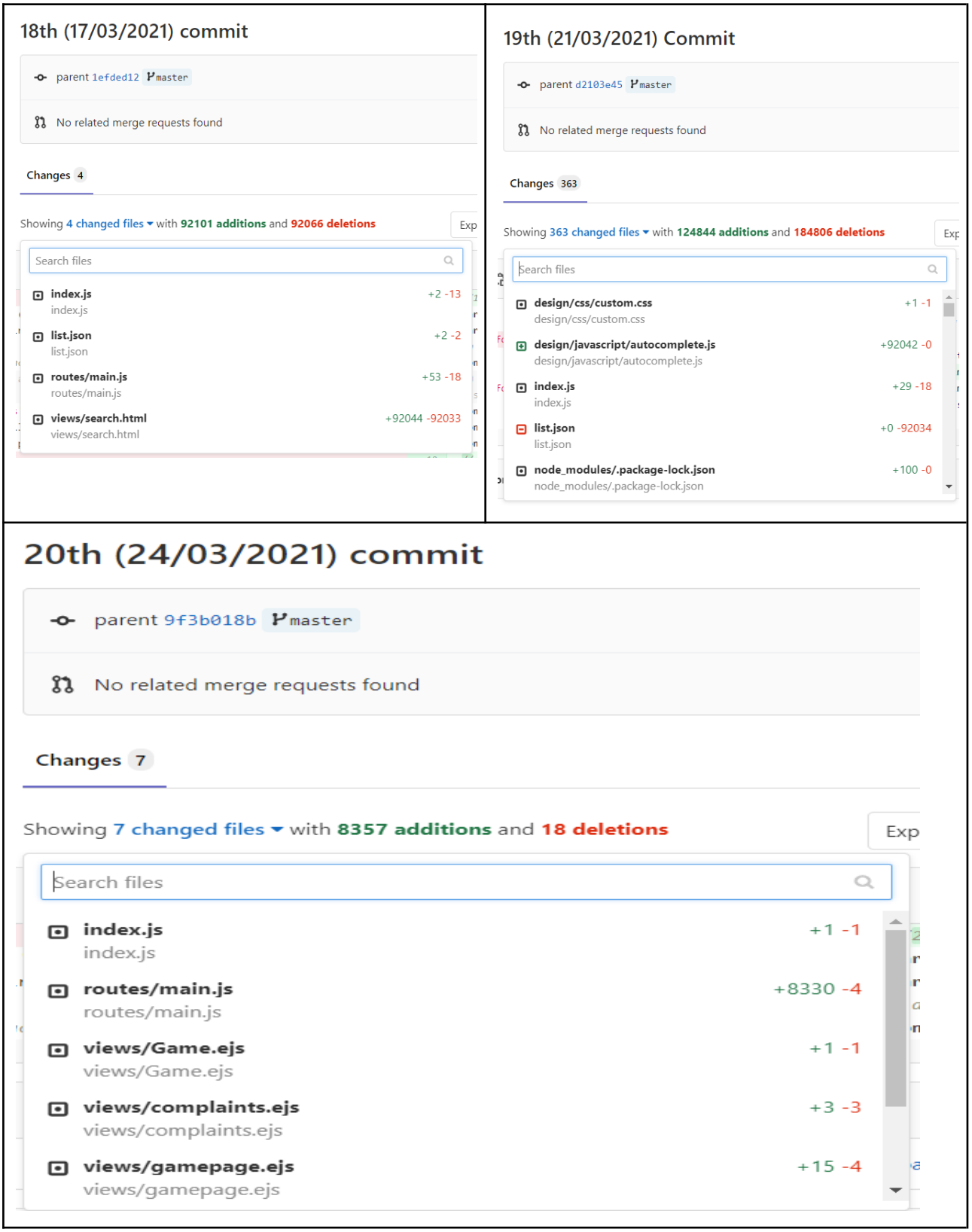


Figure 21: Commit changes - example of Version Control

The changes we made were mostly down to redundant and commented code that were not required anymore while additions were mainly based on spaces between functions so that they are more easily readable and accessible. Some additions and deletion numbers are exaggerated due to the GitLab not registering differences between commits; otherwise the changes made were quite few but nonetheless needed.

In this iteration, we will be focussing on the following criteria:

- Adding more details to games after searching

Adding more details to games after searching

As gamers ourselves, before we purchase games we always ensure that we're making the correct decision. Looking at reviews, player counts and ratings are great indicators to whether the game you are purchasing is a great or terrible buy. We want our users to also feel educated and not waste money so we are going to add these key details into the game pages.

We use our gamepage.ejs file to get and store the ratings taken from our database:

```
<% availablegame.forEach(function(game){ %>
<p><%= game.name %>, £<%= game.price %></p>
<p>Game Average Ratings: <%= game.positive_ratings %></p>
```

Figure 22.1: Storing the game details

So when we search for a game:

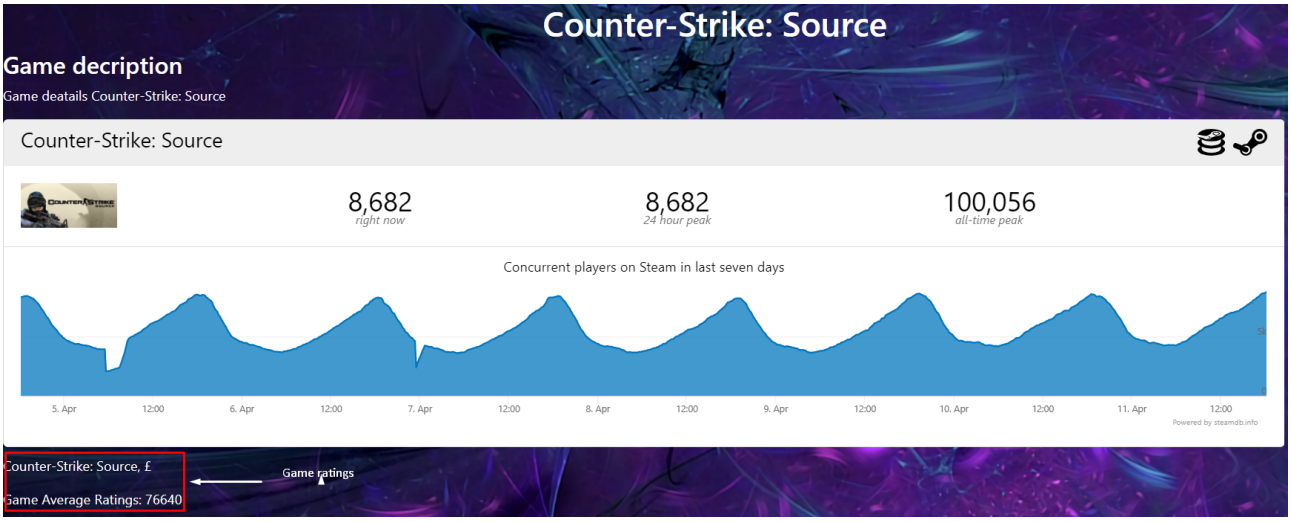


Figure 22.2: Presenting the game details

Currently, the game ratings are shown below the live player count which is not ideal but at this moment we can get the ratings presented on the page which is great.

However, if the user only searches for "Counter-Strike" - which there are several of, they will see all the Counter-Strike series:

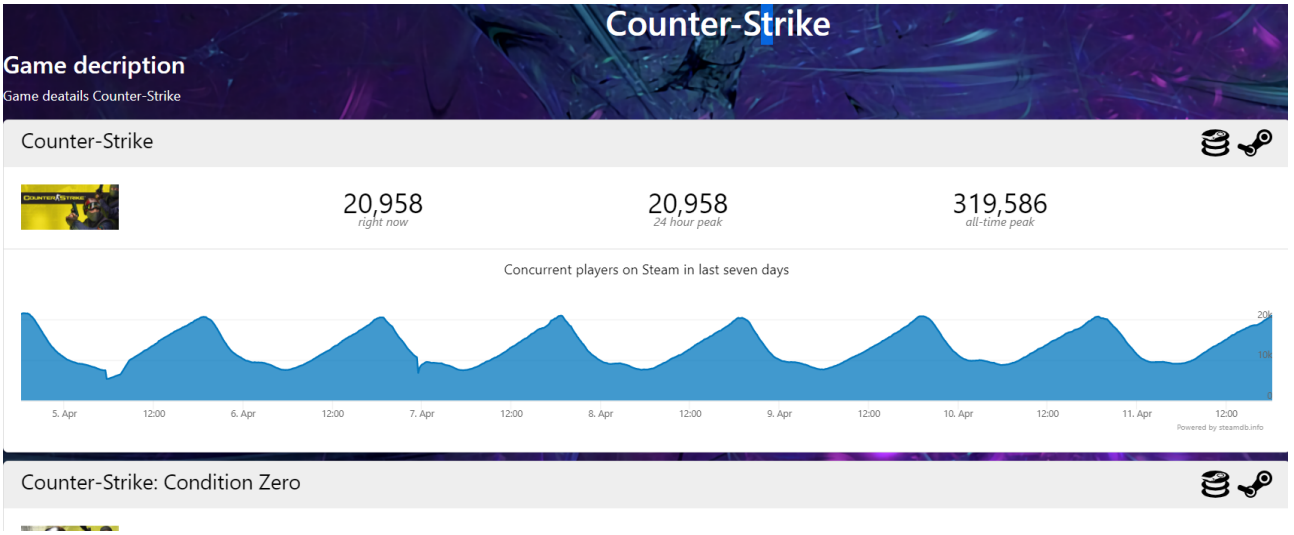


Figure 22.3: "Counter-Strike page"

But the average game ratings are still only printed out at the bottom of the page:

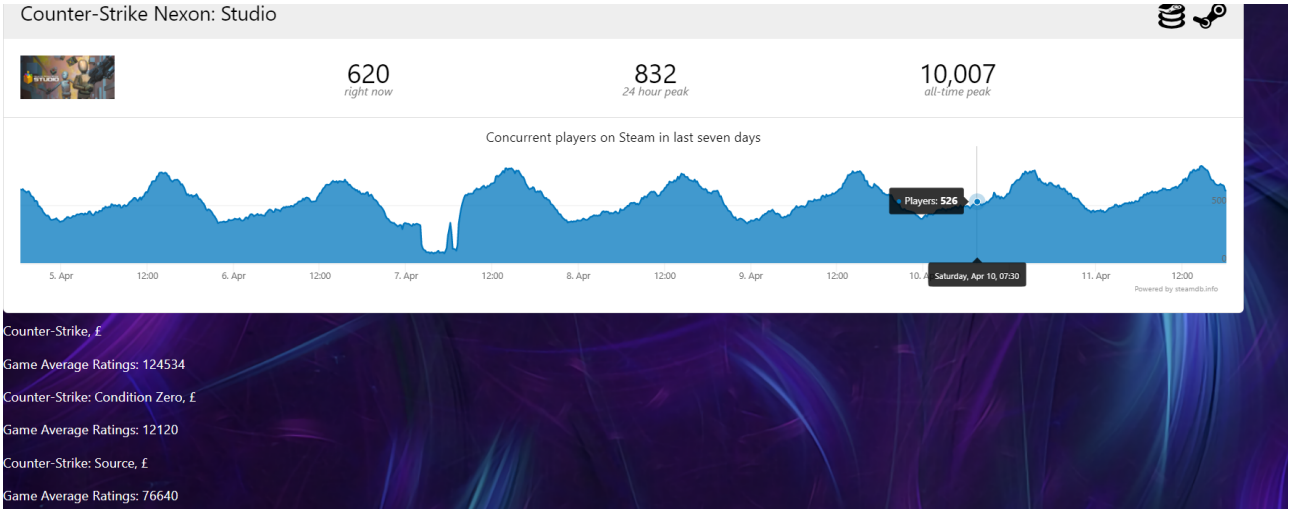


Figure 22.4: "Counter-Strike" page

Link will take you to a gif of how the page looks when scrolling:
https://gyazo.com/ffaf9d7f9d151c6433b5dca0f92add32

What we would like is for the ratings to be directly below it's subsequent game as the way this is presented is highly confusing and flawed. Meanwhile, how we got the website to produce the individual game pages was by:

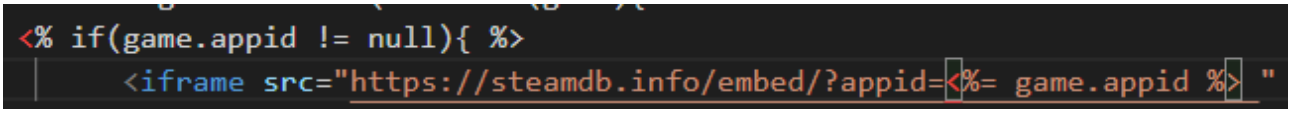


Figure 22.5: Code to check if the game exists in our database

We use the id of the game in our database to produce the games when searched, the id and the name of the are linked and if it is a match the game is presented once searched. At the moment, this only works with Steam games as their app ids are linked with what they have on SteamDB

which is why we can show the live player count. Therefore, we would need to find a similar method of doing this for our other game publishers.

Iteration 16

In this iteration, we will be focussing on the following criteria:

- Creating the forum page
- Improving individual game pages - Part 1

Improving the forum page

In Iteration 11, we made changes to our previous forum page and converted it into a complaints page due the lacklasture features it contained. But we couldn't just disregard it especially as a forum page was specially requested by our stakeholders.

We previously created our forum page as a html file, this time we used ejs which as stated before provides more functionality. We used some elements from our complaints page as a base to work on but we also added in a message box where all users can see each others messages:

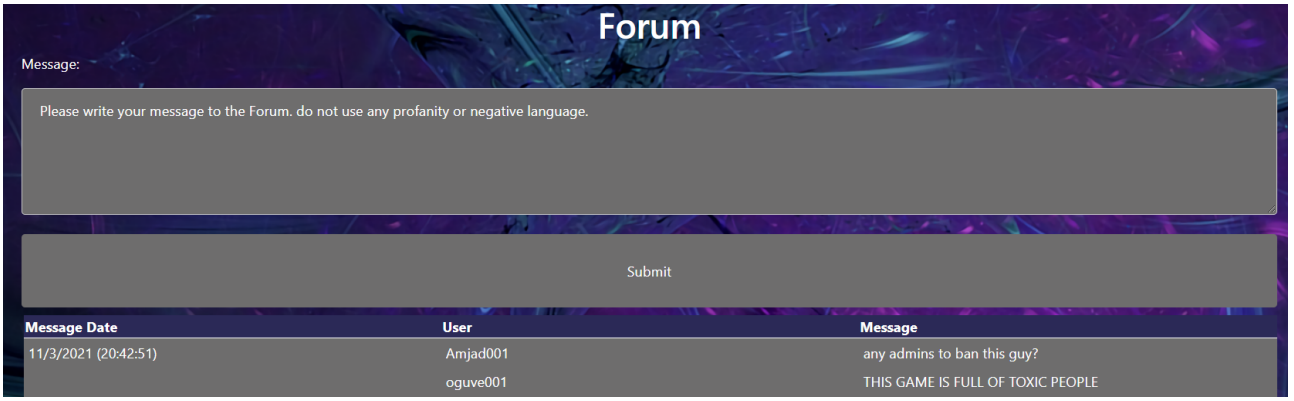


Figure 23.1: Our new forum page

The form contents are sent to our database and then the code will get the details and store it in a table format.

```
<% availablemessage.forEach(function(message){ %>
  <tr class="wposforum">
    <td>
        <%= message.date %></a>
    </td>
    <td>
        <%= message.username %>
    </td>
    <td>
        <%= message.message %>
      </td>
  </tr>
```

Figure 23.2: Gets the date, username and message and stores in a table

Before the user can view the forums page, they must be logged in - this is part of the action controls we implemented. This way the users username will be logged in the message box so that other users can see who posted the message as well as admins to carry out moderations. Once the user sends their message they will be met with this after submitting:

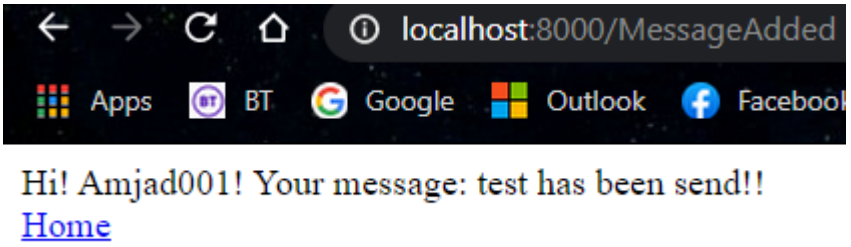


Figure 23.3: res.send from the main.js forum route

This forum page, we believe is what our users would want and it actually defines the idea of a forum page where users can communicate with each other. To ensure this is indeed the case we will call another feedback session once we have continued to develop more features and improve messages shown in Fig 23.3.

Improving individual game pages

Previously, as shown in Fig 22.4, details of a game will appear in very small text at the bottom of the page and not directly below that game. We changed that so that the visibility is more clear and that the game details now match to its game:

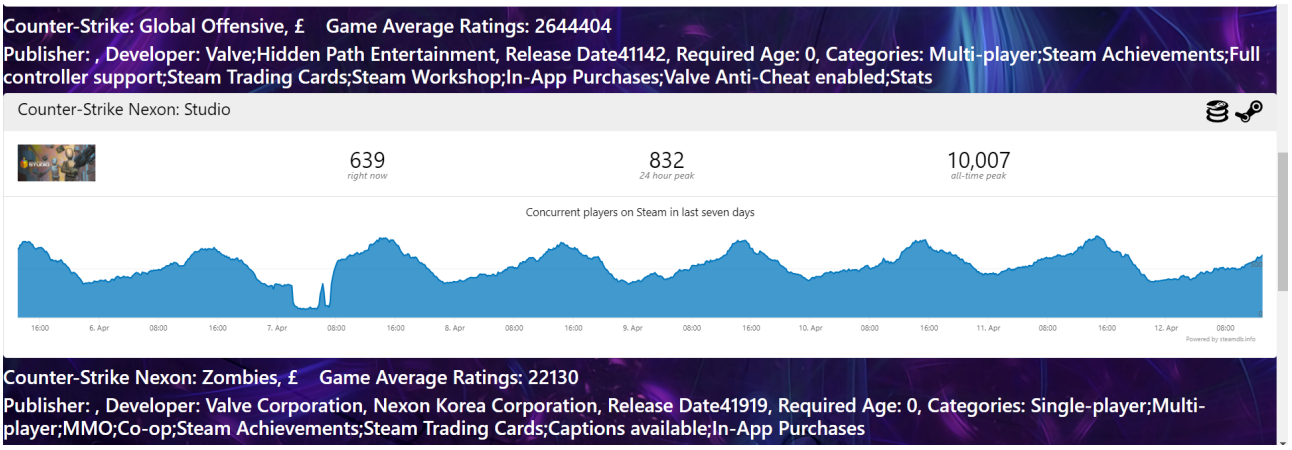


Figure 24.1: Improved game page

We've now added other key details such as publisher and developer names along with their game tags such as whether they're single/multiplayer. Although many users may already be knowledgeable with the types of games they're buying, it is still beneficial to have this especially for those uninformed. How we got the information of the games to be shown directly below the game is by changing the code which renders the variables, previously we used the <p> element so that it was represented in blocks but now we used H2 subheading tags so that it can break up content and make it scannable and easy to read.

```
<h3>
    <%= game.name
</h3>
<h3>Publisher: <%=
</h3>
```

Figure 24.2: Changed code from p tag to h3 tag

We are still unhappy with how the appearance of the game pages with the same big text and font being used making it indistinguishable and creating a bulky visual. These are some usability features that will need adapting before the next feedback session is called. Despite this, we believe that the progress is going quite well but our main feature of the website - the actual comparisons are yet to be worked so it will be something we try to press ahead with.

Iteration 17

In this iteration, we will be focussing on the following criteria:

- CSS: Presentation and design
- Improving individual game pages - Part 2
- Feedback session

CSS: Presentation and design

It is a good practice in web design to ensure there is readability. If visitors have trouble reading the content, we won't be able to achieve our marketing objectives. It is important for people to be able to read the details on our website without difficulty. We made quite a few changes and have showcased below how they were formerly and what they are now:

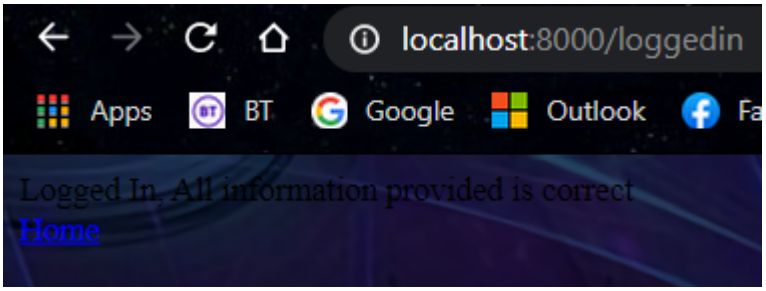
Old

New

Figure 25.1: Adding active nav item

If a user navigates to a page e.g “Popular Games”, previously the navbar would not highlight the current page they were on so we changed that so the user's active page is highlighted. We did this because we wanted to improve the findability and introduce a sort of breadcrumbs effect. Breadcrumbs enhance the way users find their way around. It reduces the number of actions a website visitor needs to take in order to get to a higher-level page and is also an effective visual aid that indicates the location of the user within our website hierarchy - which we have done here. Additionally, we have centered the heading “This Weeks Top 3 Games” so that it sets it apart from the rest of the content.

Old



New

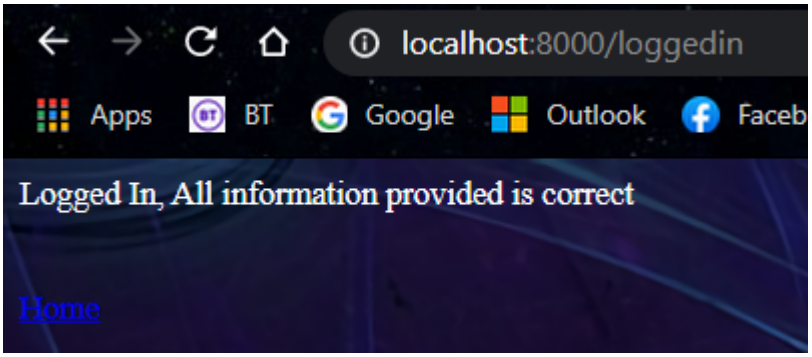
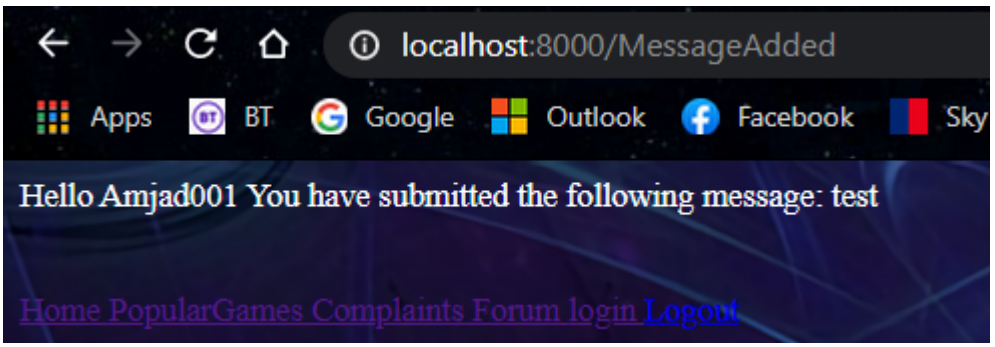


Figure 25.2: Changed font colour and increase distance between text and link

We have increased the visibility of the logged in message and increased the leading space between each text line. This basic component enhances text readability and allows users to efficiently read and take in the information in the text.

Likewise, we did the same with the forum page:

New



Improving individual game pages - Part 2

We added further details to the list of games table when a game is searched:

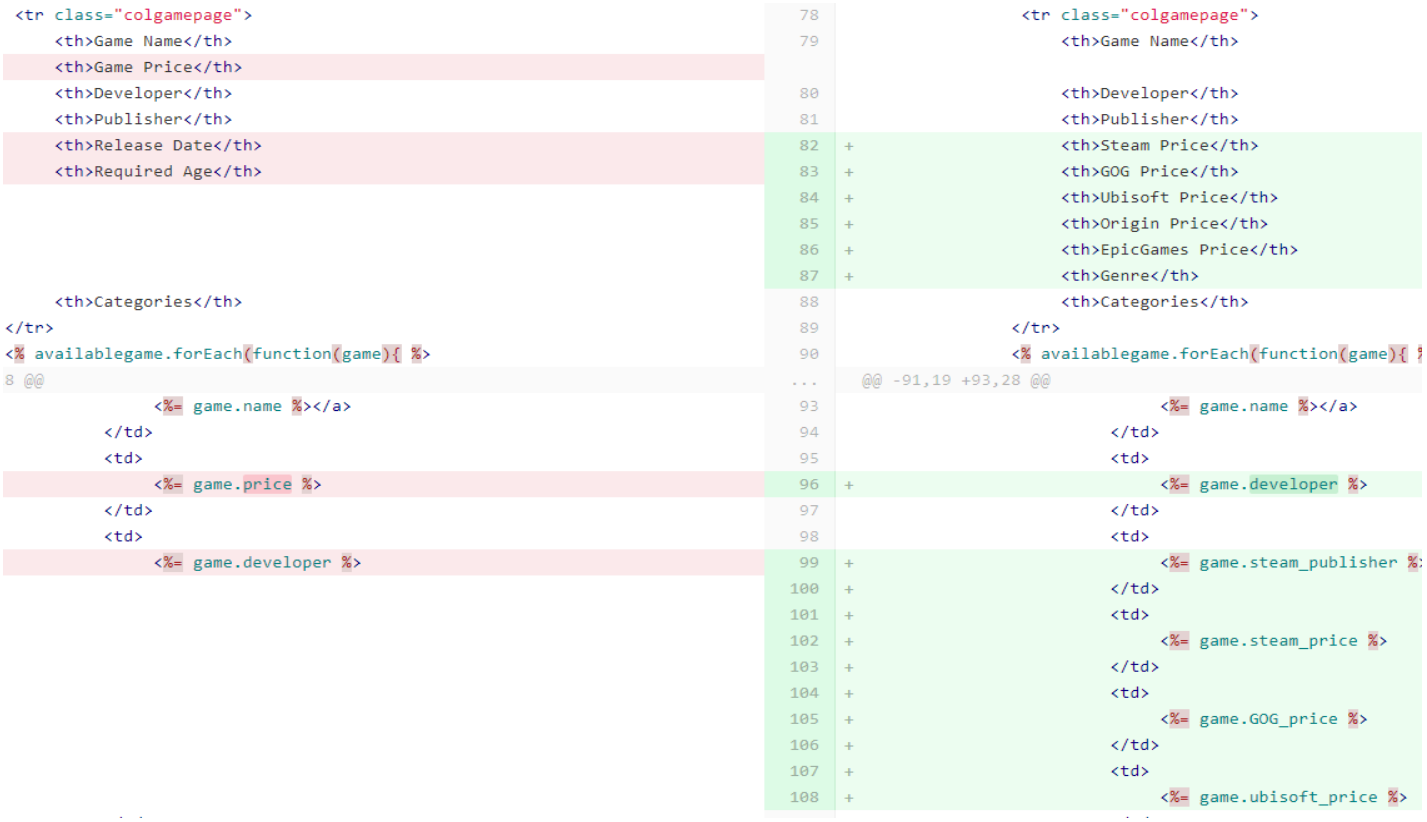


Figure 26.1: Screenshot showcasing the additions we have to this iteration

Previously, if “Far Cry” was searched all subsequent games of the Far Cry series would appear in the table:

Game Name	Game Price	Developer	Publisher	Release Date	Required Age	Categories
Far Cry 3		Ubisoft Montreal, Massive Entertainment, and Ubisoft Shanghai		41241	0	Single-player;Multi-player;Co-op;Partial Controller Support
Far Cry 3		Ubisoft Montreal, Massive Entertainment, and Ubisoft Shanghai		41241	0	Single-player;Multi-player;Co-op;Partial Controller Support

Figure 26.2: Previous iteration table

With these changes, prices from other platforms will also appear:

Game Name	Developer	Publisher	Steam Price	GOG Price	Ubisoft Price	Origin Price	EpicGames Price	Genre	Categories
Far Cry 3	Ubisoft Montreal, Massive Entertainment, and Ubisoft Shanghai	Ubisoft	16.99	null	16.99	null	£16.99	Open World;FPS;Action	Single-player;Multi-player;Co-op;Partial Controller Support
Far Cry 3	Ubisoft Montreal, Massive Entertainment, and Ubisoft Shanghai	Ubisoft	16.99	null	24.99	null	£16.99	Open World;FPS;Action	Single-player;Multi-player;Co-op;Partial Controller Support

Figure 26.3: Current iteration with prices from other platforms

This is a much improved table which includes further information to help our users make informed decisions as to whether to go ahead with their purchase. The table vaguely shows some comparison - users can see the different prices and identify which platform is currently selling the game at a cheaper price. What we could try to do to supplement this is by allowing the user to sort the prices from low to high and vice versa this way the price can easily be recognised.

Nevertheless, a big issue we have, which goes back to Iteration 9, is that if a platform does not have the game the user searches then it will output “null”. This is a problem because not only does it not look appealing but the way the table is structured does not allow for us to change it in any other way. This is because we put all the game details from all the platforms in one single collection - they were not called separately which means all the redundant information was not removed. For example, if Cyberpunk 2077 was only available on Steam and GOG, it will only have the prices for these platforms and only produce this but since we did not do it this way, the Epic Games and Ubisoft prices will produce “null”. This really is substandard and not what we want but at this point of development, with very less time remaining, it is something we will need to work with and hopefully if we have time at the end, we will try to change it.

Feedback session

We’re now at the business end of the season where changes are likely to be few and far between. This is likely to be our last feedback session before we move on to testing. We have showcased the changes we have made along with the new additions: forum, game pages.

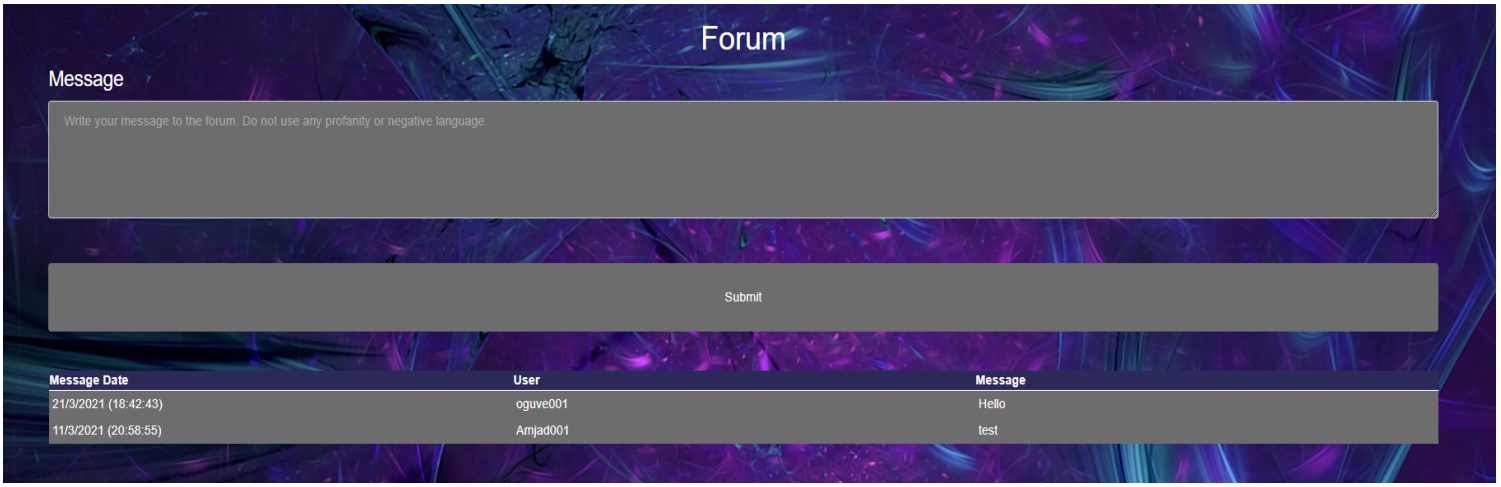


Figure 57.1: Forum page

We carried out this feedback session a little bit different. Instead of us giving our users the designs and them giving feedback, we asked them specific questions:

1. What do you like about the forum page?
2. What would you change?
3. What would you change within the styling?

The main points we got were:

- Submit button is way too big - could be easily mistaken for something else
- Looks like a table of information - doesn’t feel like it's a forum page
- White text on grey background makes it a little hard to read
- Spacing between messages in the table - very bad use of content
- Cannot reply to any messages which once again doesn’t feel like a forum page
- Cannot keep track of who is talking to who
- Format - easy to understand - you have a message, submit, then your message gets posted into the message box.

The feedback we got was quite damning but it is what we expected. We fully agree the forum page is inadequate and how we designed it with the big buttons and large space in the table would not appeal to everyone. We styled it this way to think about some of our visually impaired audience and so to ensure that both sides interests are considered we may make it so that the height of the button is the same but we would decrease the length of it and when asked if our users will like that change, one said “Yes, I think that would be the best for both parties - not too big for us and neither too small for them.” As for the actual messaging features, we will likely keep it as it is. Our users may want to be able to reply and quote messages, and although, we would love to have these features as well but at the moment this isn’t the priority; however we have promised that if we have time at the end we will absolutely try to add it in.

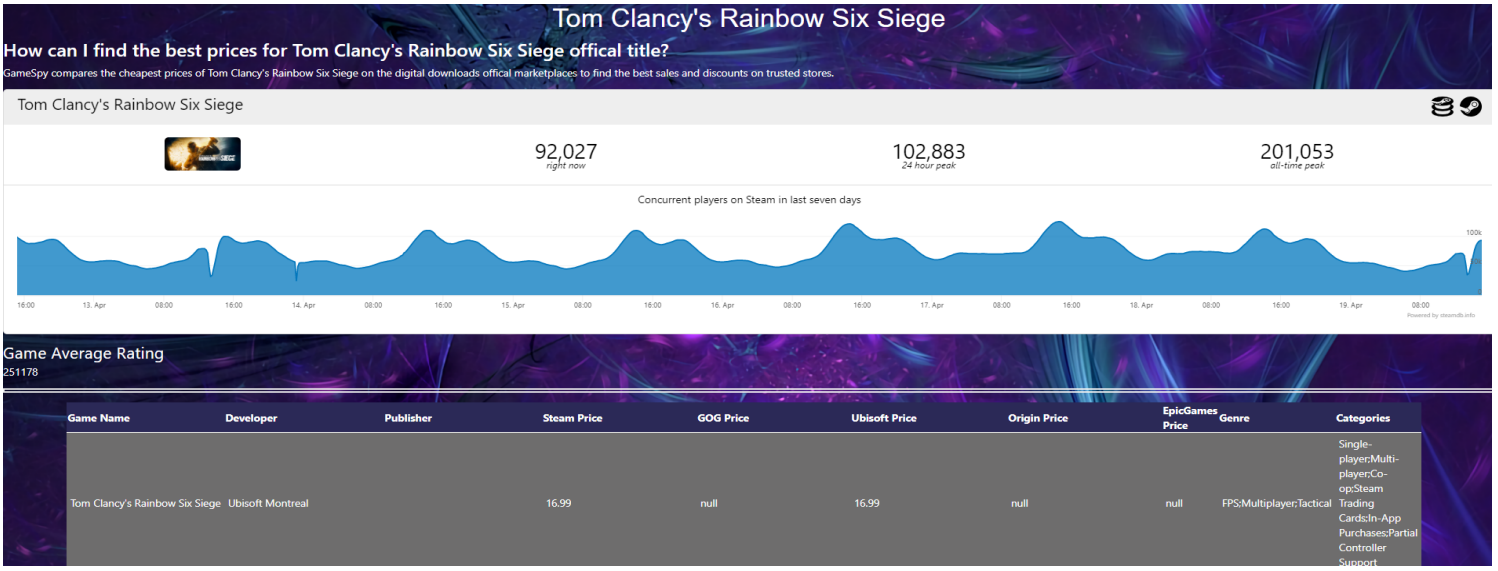


Figure 27.2: Game page

We asked the same questions for our game page and the feedback we got was:

- Good indicator of live player count
- Able to go to the steam page from the graph which is helpful
- Table could be more graphical
- Table could have more details
- Categories and genre help them get information about the game

- Good to see you have prices for different platforms so we can see what is the cheapest but there is no actual technological comparison on the website
- Game average ratings so we can tell if it is a good game or not
- Table is easy to read but too simple

The main area our users decided to focus on was our table of game details. They suggested we should have more game attributes and that we should make it more graphical such as different font colours and sizes. This is something we can do but we may struggle on the comparison element; however, our website is based on price comparisons so we will unconditionally make this our priority.

Once again we have received amazing feedback and would like to thank all our users for the time they have put in throughout the development of the project.

To-Do List at this point of Development

Throughout the iterations, we have stated all the things we still would like to do if we have the time. Here is everything we would still like to add after completing Iteration 17:

High Priority	Medium Priority	Low Priority
Live player count for other publishers	Add advanced options to Search bar e.g genre	Search Results - Database
Machine learning for price history	Adapt colour of navbar and footer	
Price comparison	Revamp forum page	
	Revamp game page table	

We sorted the features into priority lists depending on what our users would desire the most. For example, creating a price history graph which predicts the future prices would be more attractive compared to our database backend which doesn't affect them as much. While our medium priorities are mostly design and aesthetics which are important but not urgent. These are only features that we have stated we would like to add if we have the time - there are many other features of the website we still need to do before we move on to this.

Iteration 18

In this iteration, we will be focussing on the following criteria:

- Populating Popular Game page
- Improving individual game pages - Part 3

Populating Popular Game page

We have added in game images and the description of those games to our Popular Games page:

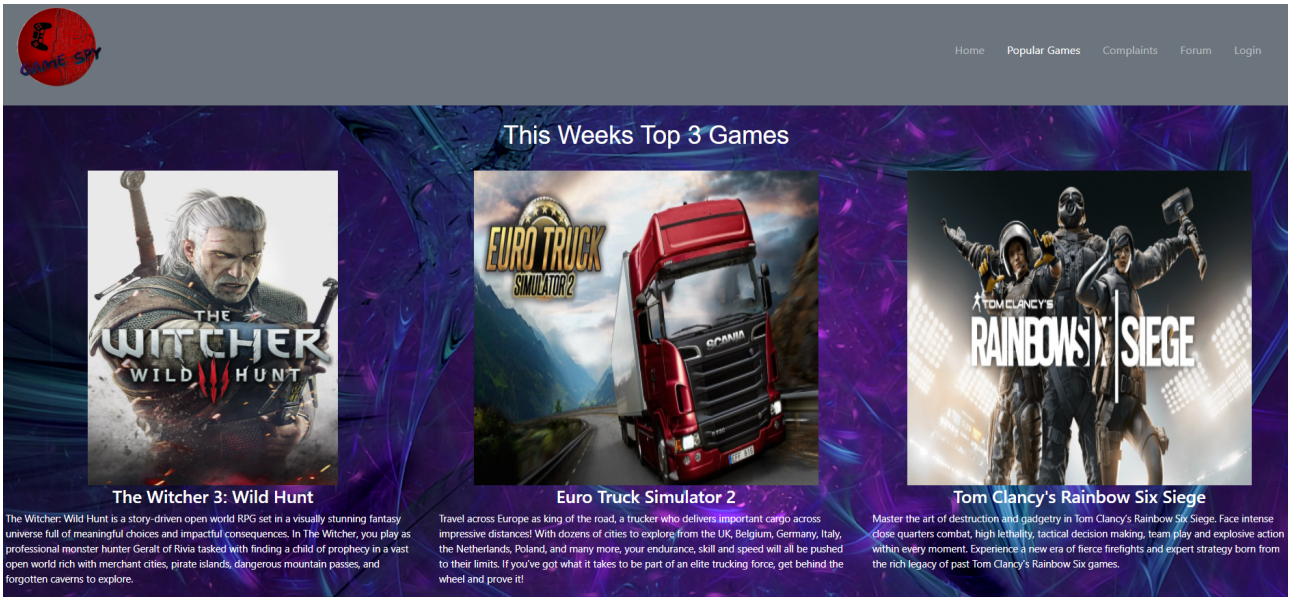


Figure 28.1: Popular Games page

How we rank the weekly top 3 games is based on Steam's Weekly Global Top Sellers for the week. The page will be updated manually every Sunday at around 17:00 BST.

Improving individual game pages - Part 3

After the most recent user feedback session, we noticed that users wanted further details of the game so this is what we did. Previously, we only had the game's average rating but this time we went forward and added both the positive and negative ratings and then we tried calculating the percentage of how many positively rated the game but had some troubles with that.

Figure 28.2: Updating game ratings

Right now it isn't clear what these numbers stand for so we need to add labels to them but the first number is the number of positive ratings, the second number is the number of negative

ratings while the last number should be a percentage of the overall ratings. How we carried that out is:

```
<span class="heading">Game Average Rating</span>
<p><%= game.positive_ratings %></p>
<p><%= game.negative_ratings %></p>
<!-- <p><%= (((game.positive_ratings / (game.positive_ratings + game.negative_ratings)) - ((

<p><%= ((game.positive_ratings) / (game.positive_ratings + game.negative_ratings)) %></p>;
```

Figure 28.3: Creating the game ratings

When we calculate the overall ratings, this should have given us a small decimal number instead of a percentage which we're not sure why. We will try to fix this for the next iteration.

And then as normal we have publisher, prices and categories of the game:

Game Name	Developer	Publisher	Steam Price	GOG Price	Ubisoft Price	Origin Price	EpicGames Price	Genre	Categories
F1 2018	Codemasters		44.99	null	null	null	null	Racing;Simulation;Sports	Single-player;Multi-player;Steam Achievements;Full controller support;Steam Workshop;Steam Cloud;Steam Leaderboards

Figure 28.4: F1 2018 game details

Iteration 19

In this iteration, we will be focussing on the following criteria:

- Populating Popular Game page - Part 2
- Amending overall rating calculator

Populating Popular Game page - Part 2

Similar to what we did for "This Weeks Top 3 Games" we also added games to "This Months Top 3 Games" and "This Years Top 3 Games":

Figure 29.1: This Months Top 3 Games

Figure 29.2: This Years Top 3 Games

Amending overall rating calculator

We're still confused by the fact that our formula gave back a very small decimal when calculating the overall ratings of the game. We created a js file called ratingCalc and used a function taken from SteamDB but that function didn't work:

```
function GetRating( positiveVotes, negativeVotes ) {
    const totalVotes = positiveVotes + negativeVotes;
    const average = positiveVotes / totalVotes;
    const score = average - ( average - 0.5 ) * Math.pow( 2, -Math.log10( totalVotes + 1 ) );
    console.log(score)
    return score * 100;
}
```

Figure 30.1: Function for getting the rating

This was because the function was not getting the values from our database. So we tried creating our own formula:

```
((positiveVotes / totalVotes) - ((positiveVotes / totalVotes) - 0.5) * Math.pow(2, -Math.log10( (positiveVotes + negativeVotes)  + 1 ))) * 100
```

Figure 30.2: Calculating the ratings

This too did not give us the answer we wanted. Here we try getting the total ratings using our gamepage.ejs file yet to no avail. However, when we tried hard coding some values in this gave us a percentage so we're not sure what the problem is.

What we tried doing instead is using Microsoft Excel. We pasted all the positive and negative ratings into the columns:

	M	N	O
genr	positive_ratings	negative_ratings	total_ratings
Actic	124534	3339	127873
Actic	3318	633	3951
Actic	3416	398	3814
Actic	1273	267	1540
Actic	5250	288	5538
Actic	2758	684	3442
Adve	588	89	677
Actic	27755	1100	28855
Actic	12120	1439	13559

Figure 30.3: Excel file

And then we created the average using the formula:

positive_ratings	negative_ratings	total_ratings	average_ratings
124534	3339	127873	=(M2/O2)*100
3318	633	3951	83.97873956
3416	398	3814	89.56476141
1273	267	1540	82.66233766
5250	288	5538	94.79956663
2758	684	3442	80.12783266
588	89	677	86.85376662
27755	1100	28855	96.18783573
12120	1439	13559	89.38712294
3822	420	4242	90.0990099
67902	2419	70321	96.56006029

Figure 30.4: Correct average rating calculated

This was then added to our database:


```
_id: ObjectId("607856f3fab6da1be442fc6f")
appid: "130"
name: "Half-Life: Blue Shift"
release_date: "37043"
english: "1"
developer: "Gearbox Software"
publisher: "Valve"
platforms: "windows;mac;linux"
required_age: "0"
categories: "Single-player"
genres: "Action"
steamspy_tags: "FPS;Action;Sci-fi"
achievements: "0"
positive_ratings: "3822"
negative_ratings: "420"
total_ratings: "4242"
average_ratings: "90.0990099"
average_playtime: "361"
median_playtime: "205"
owners: "5000000-10000000"
steam_price: "3.99"
```


Average rating calculated

Figure 30.5: Single document example from database

We then called the average ratings inside gamepage.ejs file:

```
<p><%= Math.round(game.average_ratings)%>%</p>
```

Figure 30.6: Pulling from database

We added Math.round to the rating so that the percentage gets rounded to the nearest integer and as seen in Fig 30.4, this was a necessity.

So now when this is all run:

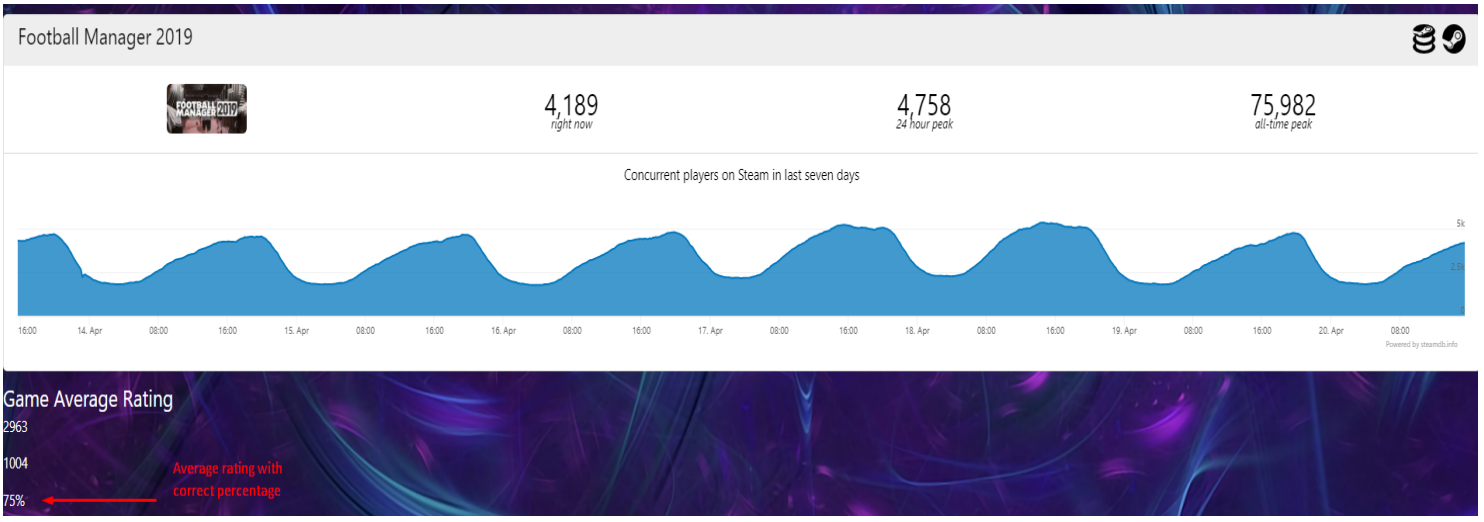


Figure 30.7: Correct rating percentage

We still haven't added the labels to the numbers but we are delighted that we were able to find a sustainable workaround when calculating the average rating. If the game ratings ever change, which they will, we can just easily update them in the datasets we have and still have functioning ratings.

Iteration 20

In this iteration, we will be focussing on the following criteria:

- Improving individual game pages - Part 4
- Starting machine learning - price history

Improving individual game pages - Part 4

Along with adding the labels:

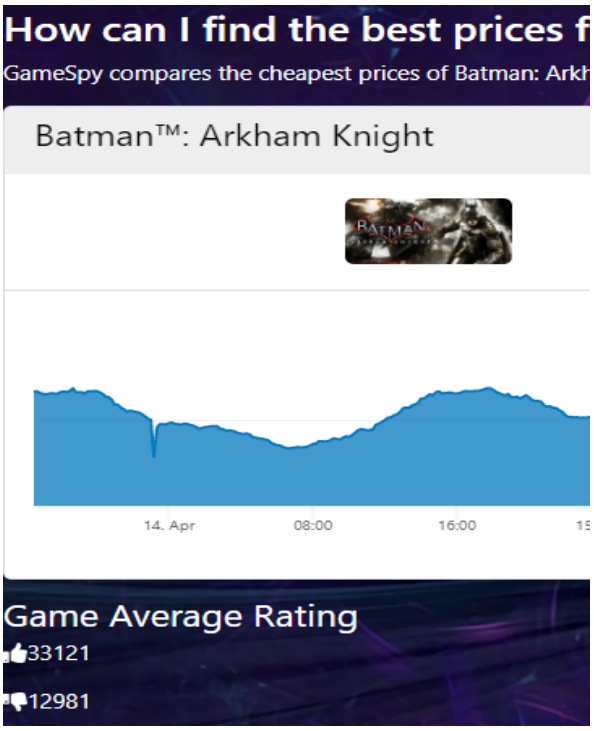


Figure 31.1: Added thumbs and down for positive and negative ratings

We also added some text to the page:

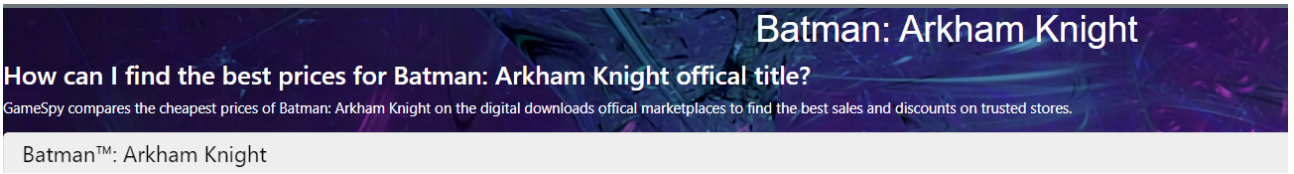


Figure 31.2: Headings and sub text

This gives the impression that they are actually looking at a price comparison website and that they will see only the best prices around on our site.

Starting machine learning - price history

We talked about machine learning in our implementation process on Page 9. The idea is to implement simple regression where we would give data on either price of the games on certain

sites and calculate the price in the future by using the equation of a line and as prices of the game over time would have a pattern which can help us find if it was on sale or not and we can give a recommendation to users to when the best time is to buy that specific game.

As we went along with the development of the project, we also took the time to research what machine learning is and how we would apply it in our project. The process was quite lengthy and difficult but we believe we now have the capability to carry this out.

After reading through several resources and even more videos, we started the task of implementing some linear regression code.

First we needed some data to work on. We will be using Tom Clancy's Rainbow Six Siege as our instance. SteamDB already had a price history graph for the game and allowed us to download the data as a CSV file:

Figure 32.1: Price history graph for Rainbow Six Siege

Our idea is to create a graph similar to this but also be able to predict prices for the future.

We programmed some code which initializes our variables and creates some arrays. First we created a javascript file called linear.js:

```
function Linear(price){
    n =price.length;
    m =price.length;
    var x = [n];
    var y =[m];
    var z = [m];
    var a = [m];
```

Figure 32.2: Setting up the linear regression

We set n and m of the price length which is all the prices from the first day of release to now as is presented in the csv file:

	A	B
1	DateTime	Final price
2	14/05/2015 17:00	49.99
3	05/11/2015 16:00	49.99
4	01/12/2015 04:00	0
5	01/12/2015 05:00	49.99
6	24/12/2015 18:00	37.49
7	28/12/2015 18:00	49.99
8	11/01/2016 16:00	39.99
9	05/02/2016 20:00	29.99

Figure 32.3: Rainbow Six dataset

The variable values are:

- x = price
- y = month
- z = square value used in least square method which is a standard approach in regression analysis to approximate solution
- a = used to multiple with month

These are all used to store the arrays of the price length. How we based the values of the variables by using this equation:

$$A = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

Figure 32.4: Regression equation

This equation uses the sum of all our variables and calculates the regression - usually the line in a graph as to whether it is positive or negative. It uses an independent variable (month) and dependent variable (price) to find the estimates using the actual data.

What it does is:

1. Multiply the sum of y (month) into sum of square of x (prices)
2. Multiply sum of x (price) and sum of y (month) then subtract from 1.
3. Subtract sum of x^2 (price^2) and sum of (x)^2 ((prices)^2) and divide from answer of 2.

How we wish to present our graph of price history is by creating a scatter graph. This is because our data will be all numeric so the graph will be able to show the relationship between the data we plot. Additionally, the dots in a scatter plot not only report the values of individual data points, but also patterns when the data are taken as a whole.

```
x1,y1,x2,y2 =0;
slope,constant = 0
```

Figure 32.5: Creating slope and sum variables

This code here gets the mean sum of the variables from Fig 32.2 while we also initialise the slope constant.

We used the equation to create our for loops so that the program keeps repeating the instruction to get our regression line and A from the equation.

```
for(i=0;i<n;i++)
{
    x1 =x[i]+x1;
    y1= y[i]+y1;
}
x1 = x1/n;
   y1= y1/n;
   for(j=0;j< n;j++)
   {
       x[j] = x[j]- x1;
       y[j] = y[j] -y1;
   }
```

Figure 32.6: 1st part of getting equation A

In the first for loop, x1 and y1 are used to store the mean of prices and days. The loop adds the values, finds the mean of those values and then gets the sum of values stored in the price. This is then divided over the length of the database.

The second for loop supplies the price and then minuses it from the mean price.

```
for(k=0;k< n;k++)
{
   z[k]= x[k]*x[k];
      y[j] = y[j] *y[j];
}
for(l=0;l< n;l++)
{
    a[l]= x[l]*y[l];
}
for(t=0;t< n;t++)
{
    x2 = a[t]+x2;
    y2 = z[t] +y2;
}
```

Figure 32.7: 2nd part of getting equation A

The third for loop is used to find the square of every price and then store that in variable z and then the fourth for loop stores the multiple of x and y into variable a. x2 and y2 are the sum of array a and z which is carried out in the final for loop.

Our thought process through our implementation is we decided that the day the game was released we marked that as day 1, then next month as 2 as so on. This is because it would be harder to present it as a date as it would be difficult to convert it into an array. It is much easier to represent it as a small number rather than a date we converted it into days. If a game came out on the 21st of April then on the 21st of May it would be classed as 31 days.

At the end of the for loops we then tried calculating the properties of the scatter graph:

```
slope = x2/y2;
constant = y1/(slope * x1);
```

Figure 32.8: Properties of scatter graph

We find slope by dividing x2 and y2 which as said before x2 is the top half of the equation in Fig 32.4 while y2 is the bottom half. Finding the constant is the equation of y = mx + c where m is the slope and c is the constant.

How we developed the code was by using a mixture of video resources which are cited in our bibliography.

Right now we cannot check whether the code we have done is working or not. To do that we need to create a function which would draw the scatter graph for us using our for loops. This is what we will continue working on in our next iteration.

Iteration 21

In this iteration, we will be focussing on the following criteria:

- Cleaning the code - Part 2

Cleaning the code - Part 2

With the stakeholder deadline closing in on us, we took the time to remove some leftover lines of code from previous iterations and added a few design pieces to the website as well:

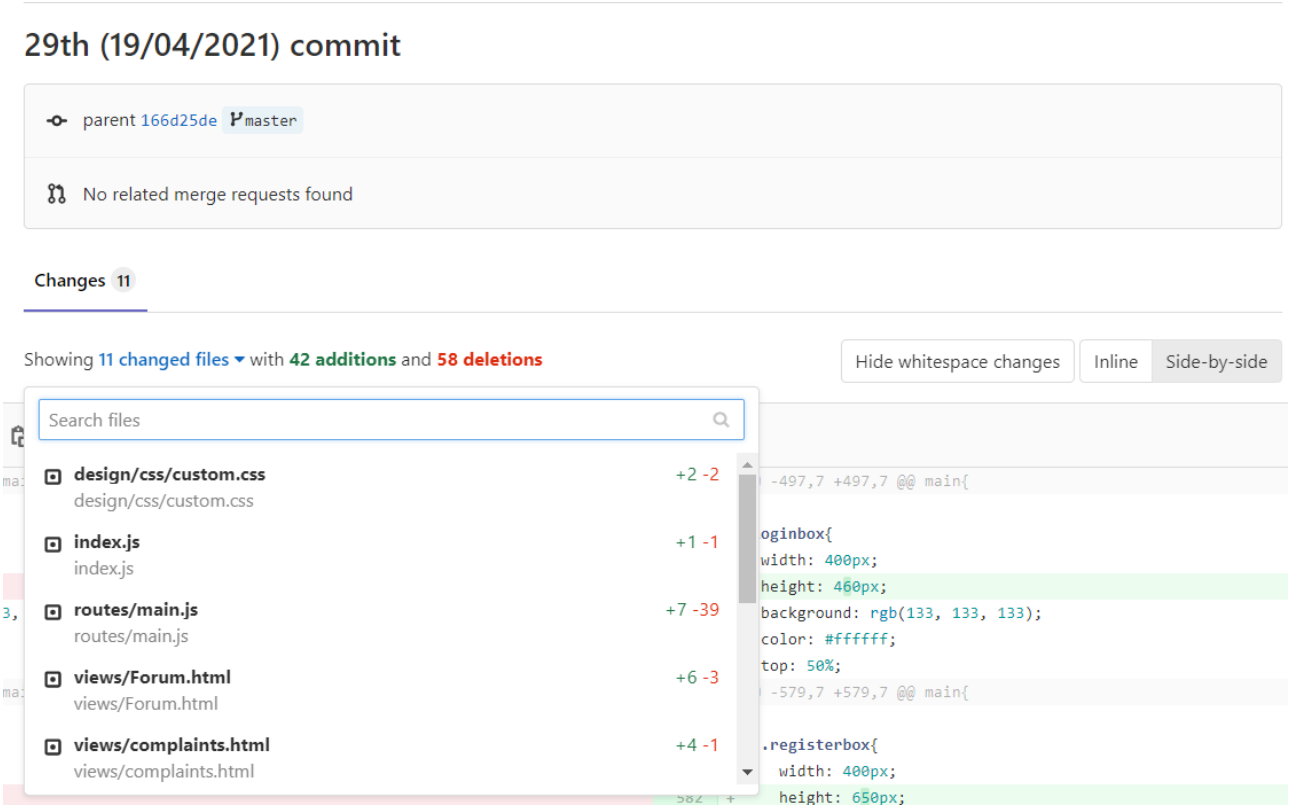


Figure 33.1: Showcasing the changes we made in GitLab

We added a Logout option to our navigation bar:

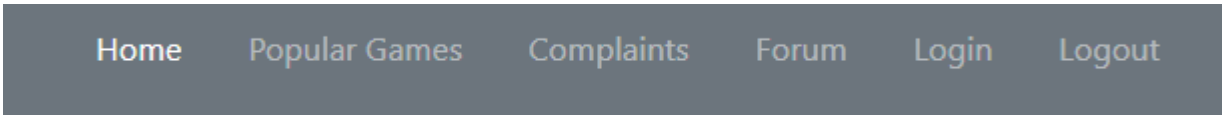


Figure 33.2: Logout in navbar

A problem we have with this is that the Logout option will always appear even if the user is logged. This is an extremely bad practice for web design so we would need to ensure it only appears if a user is logged in.

Additionally, we made some style changes to our Popular Games page. Previously, some of the game images were quite different in size and so now we changed it so that they're all the same size.

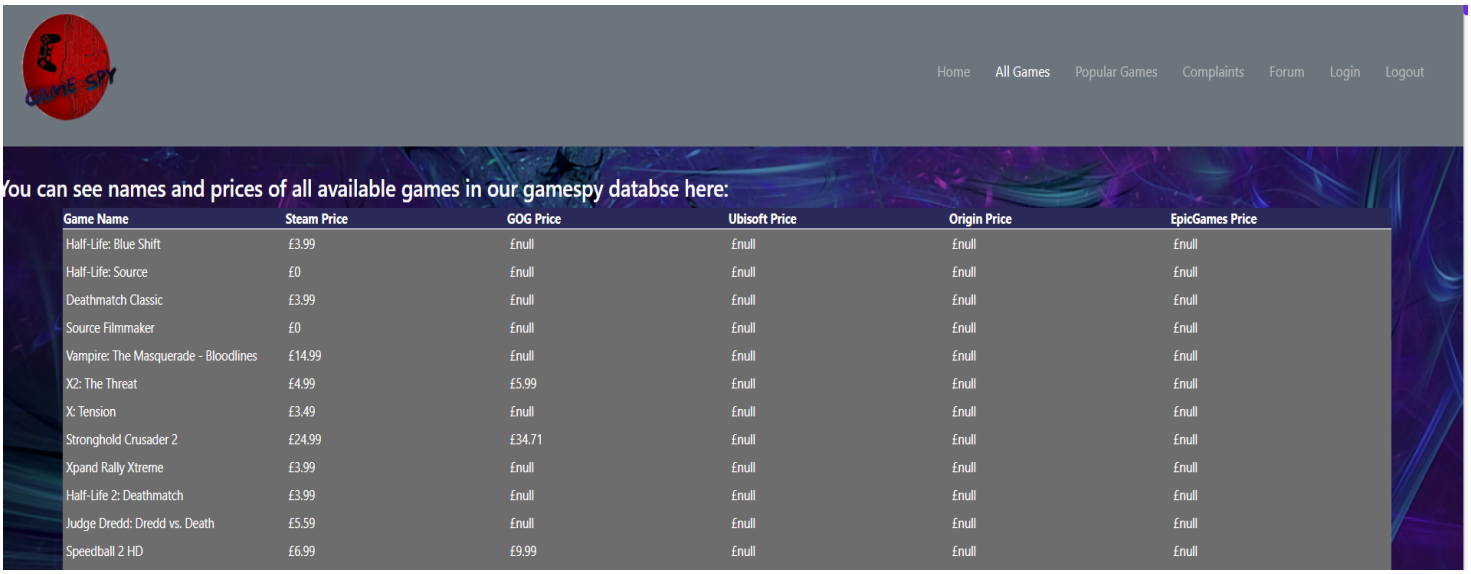
Iteration 22

In this iteration, we will be focussing on the following criteria:

- Adding All Games page
- Continuing machine learning
- Minor design changes

Adding All Games page

At the beginning of development, in Iteration 2, we created a list page to check if we had established a link between the database and the webserver. The page listed the names and prices of all games from our database and as we went along through the iterations we removed it. With the All Games page we are inheriting the structure of our individual game pages and presenting all those games like that rather than just as a list:



Game Name	Steam Price	GOG Price	Ubisoft Price	Origin Price	EpicGames Price
Half-Life: Blue Shift	£3.99	£null	£null	£null	£null
Half-Life: Source	£0	£null	£null	£null	£null
Deathmatch Classic	£3.99	£null	£null	£null	£null
Source Filmmaker	£0	£null	£null	£null	£null
Vampire: The Masquerade - Bloodlines	£14.99	£null	£null	£null	£null
X2: The Threat	£4.99	£5.99	£null	£null	£null
X: Tension	£3.49	£null	£null	£null	£null
Stronghold Crusader 2	£24.99	£34.71	£null	£null	£null
Xpand Rally Xtreme	£3.99	£null	£null	£null	£null
Half-Life 2: Deathmatch	£3.99	£null	£null	£null	£null
Judge Dredd: Dredd vs. Death	£5.59	£null	£null	£null	£null
Speedball 2 HD	£6.99	£9.99	£null	£null	£null

Figure 34: All Games page

The reason why we created a page which presents all our games is for the users who want to quickly view the games we have on the site and then they can filter it by using Ctrl+F or Cmd+F and search for it that way. Currently, users can only view the list of games and so cannot click to take them to the individual game. A drawback of having this page right now is that the page loading time is very lengthy which is due to the fact the server is calling all games from our large database. Not ideal but we will keep that in for now and see if our users find any use cases for this page

Continuing machine learning

To try and see if anything is outputted with the code so far we coded a function which should draw our scatter graph:

```
function graph(b,c){

    var svg = d3.select("body")
        .append("svg")
        .attr("width", 250)
        .attr("height", 250)

        svg.selectAll("circle")
        .b(b).enter()
        .c(c).enter()
        .append("circle")
        .attr("cx", function(d) {return d[0]})
        .attr("cy", function(d) {return d[1]})
        .attr("r", 4)

    }
```

Figure 35.1: Draw scatter graph code

We adapted the code from this webpage. First we created new variables b and c which is the price and date and are passed as parameters into our graph function. d3 is a JavaScript library for producing data visualizations in web browsers and .append and .attr are functions of that library which are used to create the size of the graph. selectAll("circle") draws our scatter points and is how they will be presented. We set the circle radius as 4 then draw our x and y axis with function d being our parameters for .attr.

However, what we realised after completing this is that we don't know how to access all the prices from the database using our linear.js function. Without this we cannot do any debugging. The data we have has to be presented as an array so that it can be passed into function. Essentially, we're unsure if it is possible to pass the prices and dates individually and then store it which is what we do with our gamepage.ejs. But the entire concept is different with machine learning. If this did perform how we wanted it to, what the users would have seen on the game pages is a graph like this:

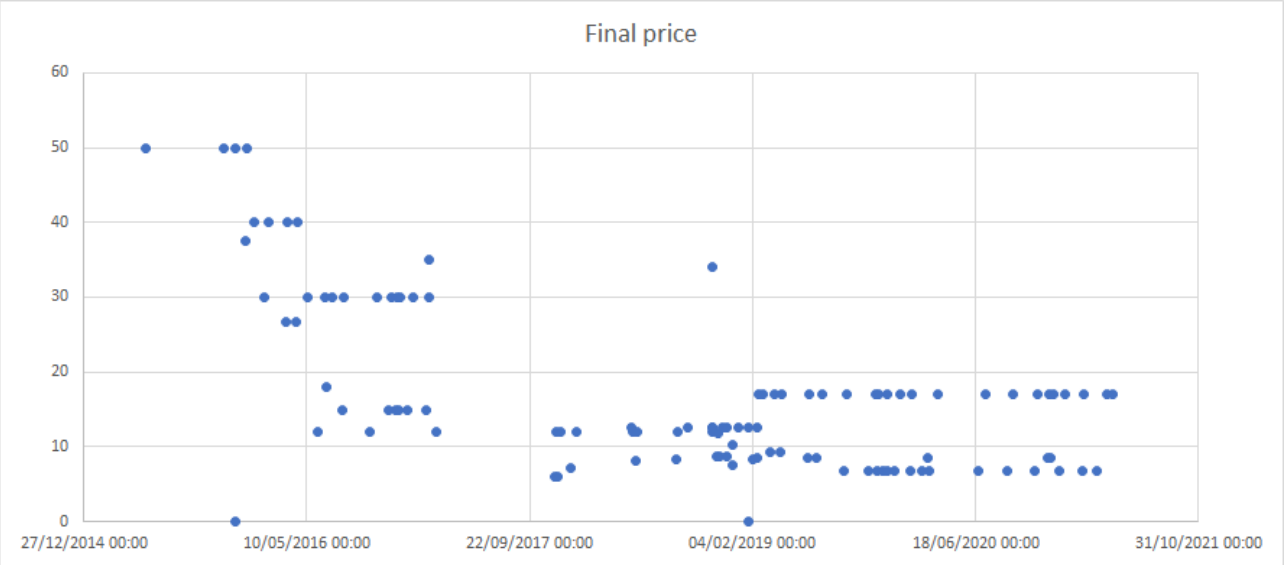


Figure 36.2: Price graph created in Excel

And then what our code should have done is create a regression line which would go through the plots and draw what the prices would be like in the future. Unfortunately, we have not achieved that yet but we will continue working on it as it is something we have put as one of our objectives.

Minor design changes

- We populated our Home page similar to what we did for Popular Games page
- Added home button to Login form:

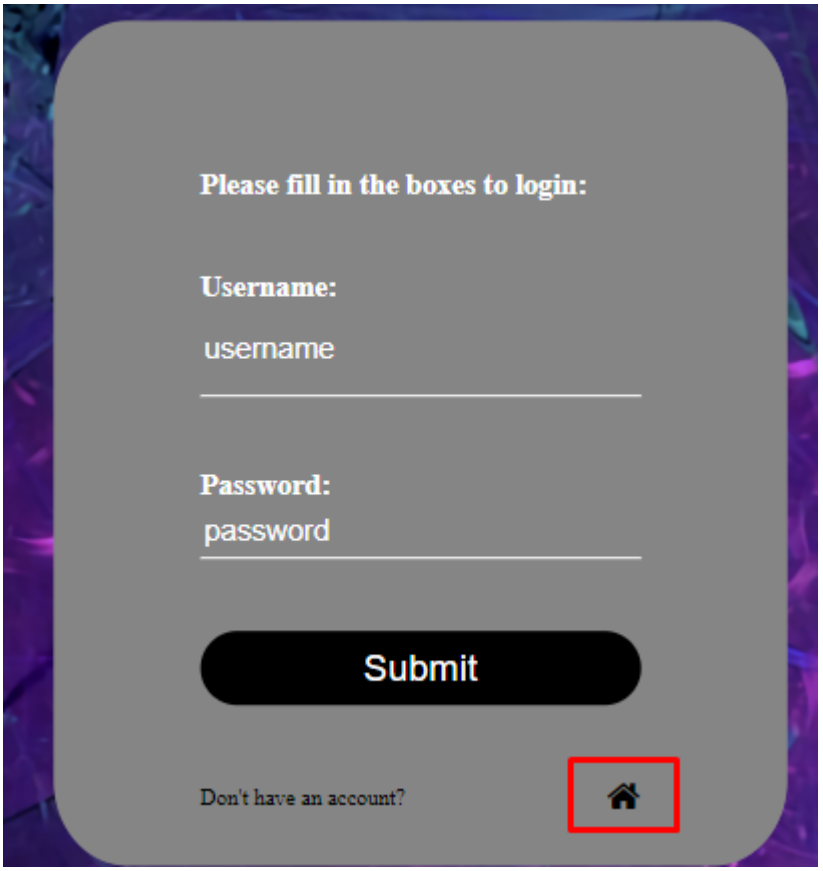


Figure 37.1: Login form home button

- Added styling to scrollbar Gif: https://gyazo.com/75f38df001247ee672a9a497f58878ec

This is the final iteration before we start doing our in house and user testing. Any other changes we make will be only done during and after testing is completed and will be produced as one final iteration.

Testing

In this major phase of developing our software. We will be using test plans and carry out different types of testing. We will be basing it on test cases which is an individual set of variables or conditions that is used to see if features of the software work as expected. In the test plan, various test cases may be planned and then tested. We will be designing test cases that will check if the software will fail and some that will check if it succeeds.

Also considering the current climate we are in, we will also make use of remote usability testing which allows us to conduct user research with participants in their natural environment by employing screen-sharing software or online remote usability.

Additionally, we have already been keeping detailed records of all the tests that have been carried out as well as any changes that have been made after the tests. GitLab is the version control system we have used to track changes in the development of our application. If errors are found in the software later on, the tracking that GitLab provides enables us to go back and see what changes were made.

Unit Testing

The first set of testing we did was Unit Testing. This was where we checked all our functions and subroutines individually and independently. As each new part of the system was completed, it is tested to make sure that it works properly - otherwise other parts of the system that will rely on it could fail. If the new part of the system works, it can be used by other parts of the system.

```
1  module.exports = function (app) // this file has been exported as a function
2  {
3    //to validate user inputs
4    const { check, validationResult } = require('express-validator');
5    //redirect to login pages if the user have not logged in
6    const redirectLogin = (req, res, next) => {
7
8      if (!req.session.userId) {
9        res.redirect('./login')
10     }
11     else {
12       next();
13     }
14   }
15
```

Figure 38: Redirect login code to check if user has not logged in

This function redirects the user if they do not have an active session on the website. This is a form of access control whereby we are not allowing general users to access parts of the website which

are locked behind a login page. When tested with a newly registered account this function allows the users to log into the page with the redirectLogin feature.

```
//////////////////////---------------------------- REGISTER PAGE ----------------------------------\\\\\\\\\\\\\\\\\\\\\\

app.get('/register', function (req, res) {
  res.render("register.html");
})
//conditions that should be met when registering
app.post('/registered', [check('username').notEmpty(), check('firstname').notEmpty(), check('lastname').notEmpty(), check('email').isEmail(), check('password').isLength({ min: 8 })], function (req, res) {
  var MongoClient = require('mongodb').MongoClient;
  const url = "mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test";

  MongoClient.connect(url, function (err, client) {
    if (err) throw err;
    var db = client.db('GameSpy'); //access to the mybookshopdb database
    const bcrypt = require('bcrypt'); //access to bcrypt functions
    const saltRounds = 10;  //declare the length of the saltRounds it is going to use
    const plainPassword = req.sanitize(req.body.password); //saves inputted password as plainPassword
    const errors = validationResult(req);
    //if nothing is written then hte user is redirected to the register page
    if (!errors.isEmpty()) {
      res.redirect('./register');
    }
    else {
      bcrypt.hash(plainPassword, saltRounds, function (err, hashedPassword) { //hashes plainPassword using saltRounds and calls it hashedPassword
        //sanitizing and registering the user to the database
        db.collection('users').insertOne({
          firstname: req.sanitize(req.body.firstname),
          lastname: req.sanitize(req.body.lastname),
          email: req.sanitize(req.body.email),
          username: req.sanitize(req.body.username),
          password: hashedPassword
        });
        res.send('<link rel="stylesheet" type="text/css" href="css/custom.css">' + '<p style= "color:#FFFFFF;">' + 'You are now registered, Your user name is: '
        + req.body.username + ' your password is: ' + req.body.password + ' and your hashed password is: ' + hashedPassword + '</p>'
        + '<br />' + '<a href=' + './' + '>Home </a>' + '<a href=' + './PopularGames' + '> PopularGames </a> ' + '<a href=' + './Complaints' + '> Complaints</a>'
        + '<a href=' + './Forum' + '> Forum </a>' + '<a href=' + './login' + '> login </a>' + '<a href=' + './logout' + '> Logout</a>');
        client.close();
      })
    }
  });
});
```

Figure 39: Register code

The register page allows any user to register into our website. Only registered users can access many of the functions that are provided by the website. Details such as username,email,name are required to be filled out by users that wish to register to the website. We have also set certain conditions to all the forms that need to be filled in such as, the password should have a minimum length of 8, none of the form columns are empty etc. Once a user is registered, their details are stored in our database and a message shows on the webpage telling the user that their registration was successful.The testing was done successfully, the users details were registered into our database as well as, they were able to login using their username and password.

```
159   //////////////////////---------------------------- COMPLAINTS PAGE ----------------------------------\\\\\\\\\\\\\\\\\\\\\\
160   //Users can raise their concerns in the complaint page. (provided to only logged in users)
161   app.get('/Complaints', redirectLogin, function (req, res) {
162     res.render('complaints.html')
163   });
164   //checks if everything is filled
165   app.post('/Complained', [check('email').isEmail(), check('message').notEmpty(), check('subject').notEmpty()], function (req, res) {
166     // saving data in database
167     var MongoClient = require('mongodb').MongoClient;
168     const url = "mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test";
169     //Store hashed password in your database.
170     MongoClient.connect(url, function (err, client) {
171       if (err) throw err;
172       var db = client.db('GameSpy');
173
174       const errors = validationResult(req);
175       //if there is any errors then the user is redirected to the complaints page
176       if (!errors.isEmpty()) {
177         res.redirect('./Forum');
178       }
179       //if everything is filled then the complaint is stored into the database called Complaints
180       else {
181         db.collection('Complaints').insertOne({ // collection and documents
182           Email: req.body.email,
183           Subject: req.body.subject,
184           Message: req.body.message,
185         });
186         res.send('<link rel="stylesheet" type="text/css" href="css/custom.css">' + '<p style= "color:#FFFFFF;">'
187         + ("Your Email is: " + req.body.email + "The subject of the issue is: " + req.body.subject + "The message you are sending is: " + req.body.message) + '</p>'
188         + '<br />' + '<a href=' + './' + '>Home </a>' + '<a href=' + './PopularGames' + '> PopularGames </a> ' + '<a href=' + './Complaints' + '> Complaints</a>'
189         + '<a href=' + './Forum' + '> Forum </a>' + '<a href=' + './login' + '> login </a>' + '<a href=' + './logout' + '> Logout</a>');  //sends response to user
190         client.close();
191       }
192     })
193   })
194
```

Figure 40.1: Complaints code

Complaints Page works if the user is logged in then the user would be able to get access to the webpage for complaints (Fig 40.2) and the user can only submit a complaints if they have filled in all the form boxes on the webpage if any form boxes are empty then an error would be shown to the user telling them to fill in the form box (Fig 40.3) . Once the user fills in all the boxes and submits the complaint they would be redirected to a page which confirms the complaint was submitted.(Fig 40.4)

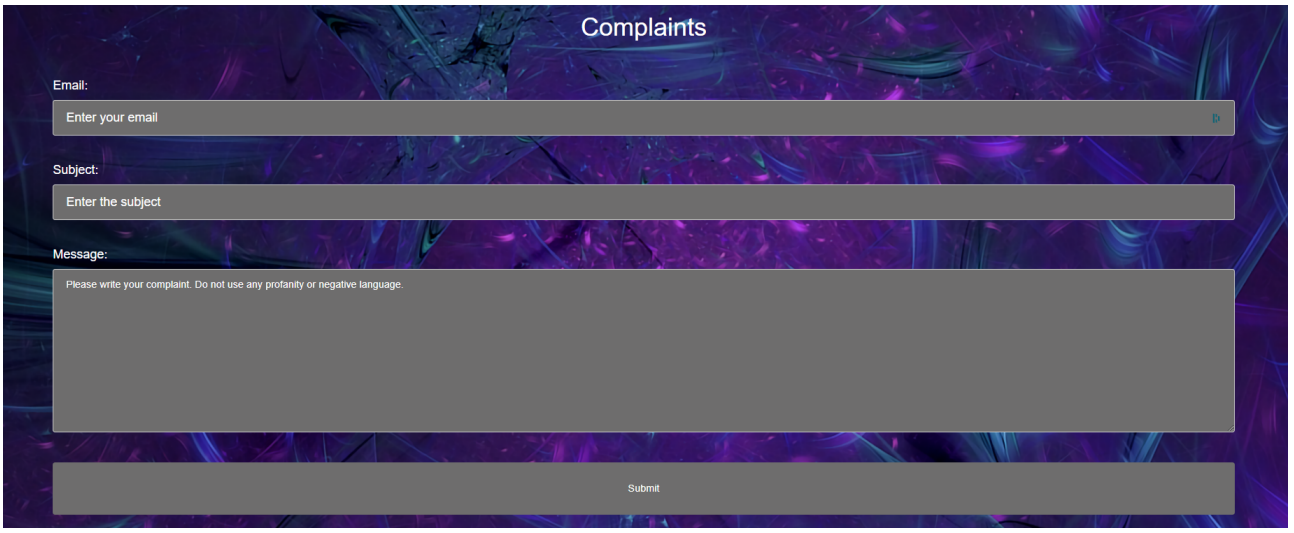


Fig 40.2: Complaints page when run

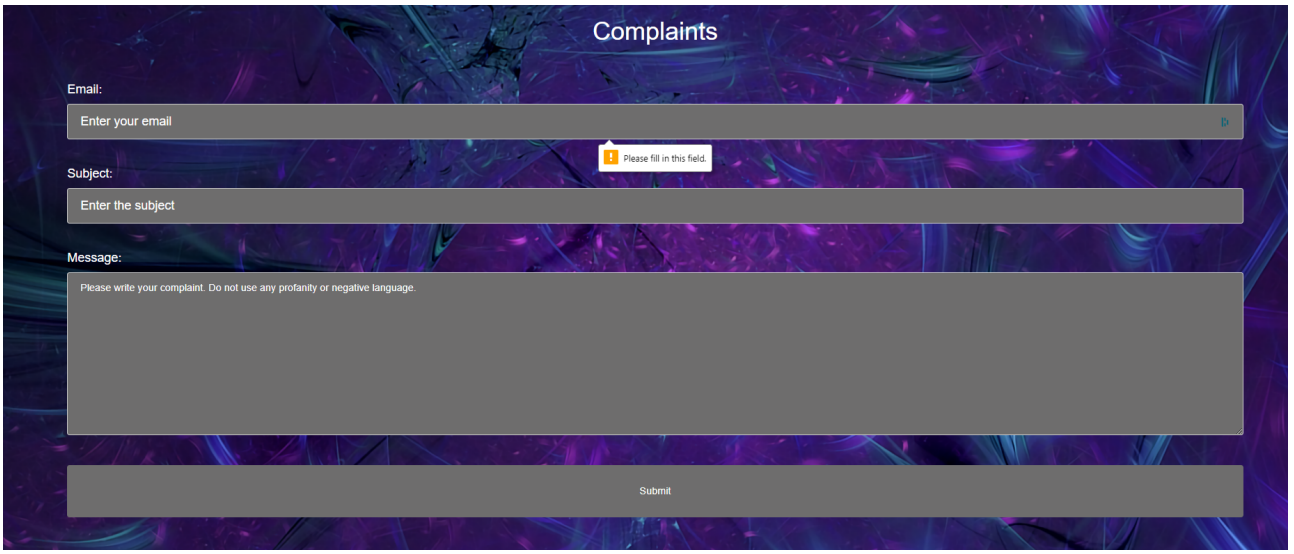


Fig 40.3: Complaints page: telling user to fill in email form

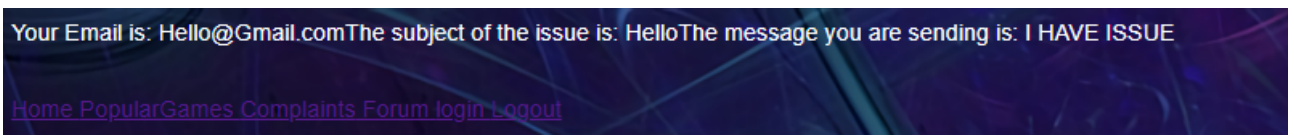


Fig 40.4: Complaints page when from is bumitted

```
//////////////////////------------------------------ COMPLAINTS PAGE FOR ADMIN ----------------------------------\\\\\\\\\\\\\\\\\\\\\\
//only accessable by admins
app.get('/complaintsadmin', redirectLogin, function (req, res) {
  var MongoClient = require('mongodb').MongoClient;
  const url = "mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test";
  MongoClient.connect(url, function (err, client) {
    if (err) throw err;
    var db = client.db('GameSpy');
    db.collection('Complaints').find().toArray((findErr, results) => { // produces all available books
      if (findErr) throw findErr;
      else
        //web page with all active complaints
        res.render('complaintsadmin.ejs', { availablegame: results });
      client.close();
    });
  });
})

app.post('/deleted', [check('email').isEmail()], function (req, res) {
  //saving data in database
  var MongoClient = require('mongodb').MongoClient;
  var url = "mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test";

  MongoClient.connect(url, function (err, client) {
    if (err) throw err;
    var db = client.db('GameSpy');  //access to the database
    //gets the email address of the user
    var word1 = req.body.email;
    console.log(word1);
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      res.redirect('./complaints');
    }
    else {
      //if the user email is found, then the complaint is delete
      db.collection('Complaints').findOne({ Email: word1 }, function (err, result) { //find if the username entered is in the database if so it would get all the information for that user
        if (err) throw err;
        if (result != null) { //if result is not equal to null that means the username exists in the database so it would execute the code within the if statement

          db.collection('Complaints').deleteOne({ Email: word1 }, function (err, results) { //delete the user that has the username entered in the users collection by comparing username entered w
            res.send('<link rel="stylesheet" type="text/css" href="css/custom.css">' + '<p style= "color:#FFFFFF;">'
            + ('Deleting User ' + word1 + ' from the database') + '</p>' + '<br />' + '<a href=' + './' + '>Home</a>')
          })
        }
        else { //if username is not found in the database that means results is equal to null then tell the user that the username entered does not exist in the database
          res.send('<link rel="stylesheet" type="text/css" href="css/custom.css">' + '<p style = "color:#FFFFFF;">' + ("User with the username " + word1 + " does not exist in the database")
          + '</p>' + '<br />' + '<a href=' + './' + '>Home </a>' + '<a href=' + './PopularGames' + '> PopularGames </a> ' + '<a href=' + './Complaints' + '> Complaints</a>'
          + '<a href=' + './Forum' + '> Forum </a>' + '<a href=' + './login' + '> login </a>' + '<a href=' + './logout' + '> Logout</a>')
        }
      })
    }
  })
});
```

Complaints Admin Page only works if you are logged in it works for any logged in user which we want to change in the future to only admins. It allows you to see all complaints submitted with the user's email and the complaint with its subject (Fig 40.5) and the admin can use that information to reply back to the user via email. Once the admin is done the admin can delete the complaints by putting the users email into the form box and submitting it which would delete the first occurrence of that email in the collection and remove the complaint from the database (Fig 40.6). If the form box is left empty then an error would be shown to the admin when the admin is trying to press the submit button saying that the field cannot be empty (Fig 40.7) . Also if the email is incorrect then the admin would be redirected showing that the email does not exist in the database (Fig 40.8).

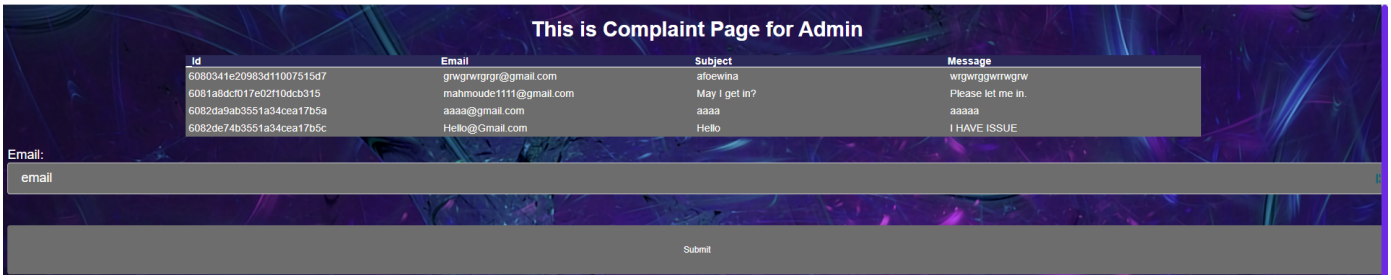
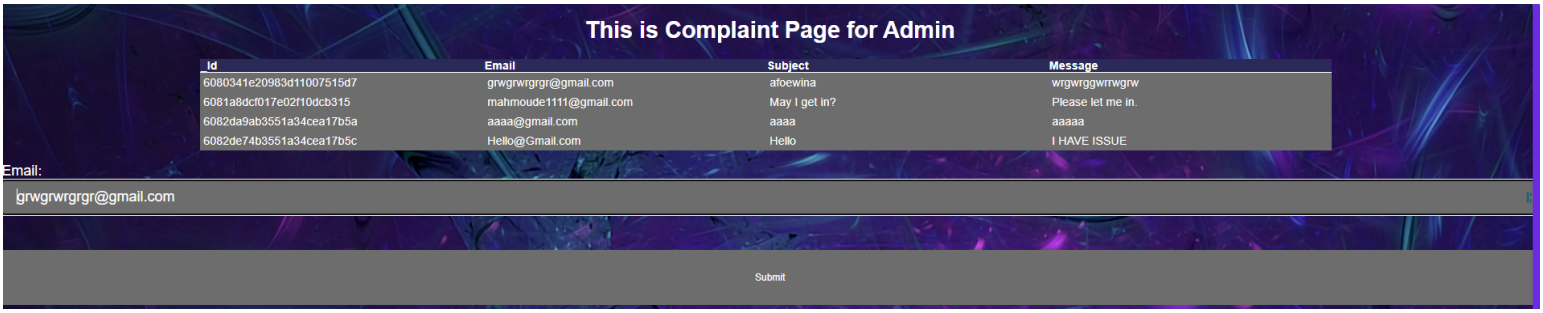


Fig 40.5: Complaints page for admin access



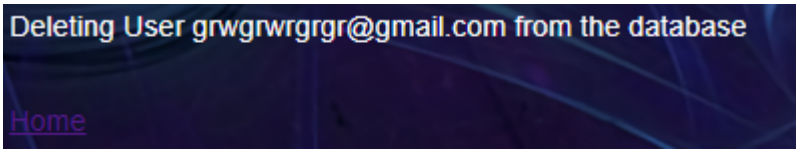


Fig 40.6: Delete complaints page

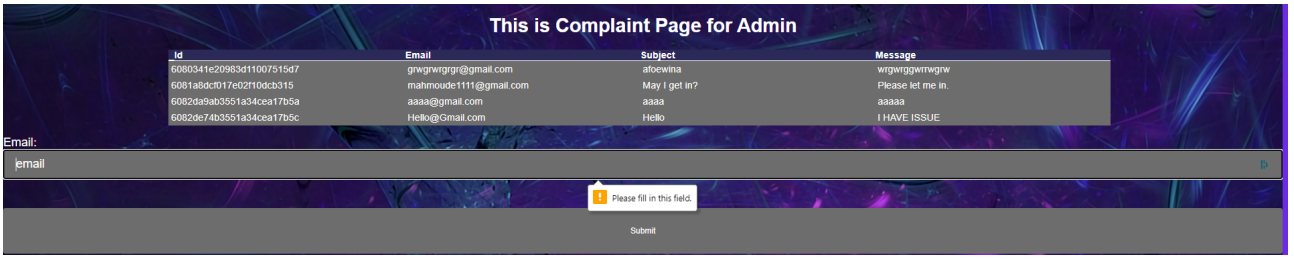


Fig 40.7: Field cannot be empty for userID

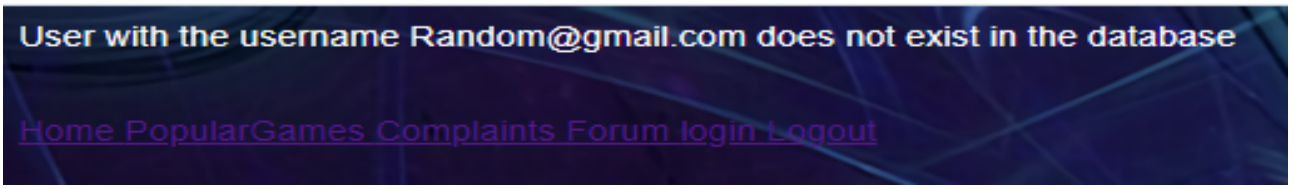


Fig 40.8: If email is incorrect then this is outputted

```
//////////////////////------------------------------ LOGIN PAGE ----------------------------------\\\\\\\\\\\\\\\\\\\\\\\\

app.get('/login', function (req, res) {
  res.render('login.html')
});

// Login page - checks if all the conditions are met
app.post('/loggedin', [check('username').notEmpty(), check('password').notEmpty()], function (req, res) {
  var MongoClient = require('mongodb').MongoClient;
  const url = "mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test";

  MongoClient.connect(url, function (err, client) {
    if (err) throw err;
    var db = client.db('GameSpy');
    const bcrypt = require('bcrypt');
    const saltRounds = 10;
    const plainPassword = req.body.password;
    var word1 = req.body.username;

    const errors = validationResult(req);
    //checks if the fields are left emoty
    if (!errors.isEmpty()) {
      res.redirect('./login');
    }
    else {
      //checks if the user name is in our system
      db.collection('users').findOne({ username: word1 }, function (err, results) {
        if (err) throw err;
        if (results != null) {
          //compares the password given and the stored password
          bcrypt.compare(plainPassword, results.password, function (err, result) {
            if (err) throw err;
            if (result == true) {
              req.session.userId = req.sanitize(req.body.username);
              //if the provided username and password are correct the user session starts
              res.send('<link rel="stylesheet" type="text/css" href="css/custom.css">' + '<p style= "color:#FFFFFF;">' + ('Logged In, All information provided is correct') + '</p>'
              + '<br />' + '<a href=' + './' + '>Home </a>' + '<a href=' + './PopularGames' + '> PopularGames </a> '
              + '<a href=' + './Complaints' + '> Complaints</a>' + '<a href=' + './Forum' + '> Forum </a>' + '<a href=' + './login' + '> login </a>' + '<a href=' + './logout' + '> Logout</a>')
            }
            else {
              //if username is correct and the password wrong, then the following line prints out
              res.send('<link rel="stylesheet" type="text/css" href="css/custom.css">' + '<p style= "color:#FFFFFF;">' + ('Invalid Password Entered') + '</p>' + '<br />'
              + '<a href=' + './' + '>Home </a>' + '<a href=' + './PopularGames' + '> PopularGames </a> ' + '<a href=' + './Complaints' + '> Complaints</a>' + '<a href=' + './Forum' + '> Forum </a>'
               + '<a href=' + './login' + '> login </a>' + '<a href=' + './logout' + '> Logout</a>')
            }
          })
        }
        else {
          //if provided user name is not in the database then the following line prints out
          res.send('<link rel="stylesheet" type="text/css" href="css/custom.css">' + '<p style= "color:#FFFFFF;">' + ('Invalid Username Entered') + '</p>'
          + '<br />' + '<a href=' + './' + '>Home </a>' + '<a href=' + './PopularGames' + '> PopularGames </a> ' + '<a href=' + './Complaints' + '> Complaints</a>'
          + '<a href=' + './Forum' + '> Forum </a>' + '<a href=' + './login' + '> login </a>' + '<a href=' + './logout' + '> Logout</a>')
        }
      })
    }
  })
});
```

Fig 41.1: Login code

Login page allows the user to login with an already registered username and password saved to the mongodb database. The login page requires the two fields to be filled in if not the validator will refresh the page. This code prints 3 res.sends depending on the situations if username is wrong, password is wrong or all correct logged in. Once logged in the user can access all the restricted pages such as complaints and the forum page.

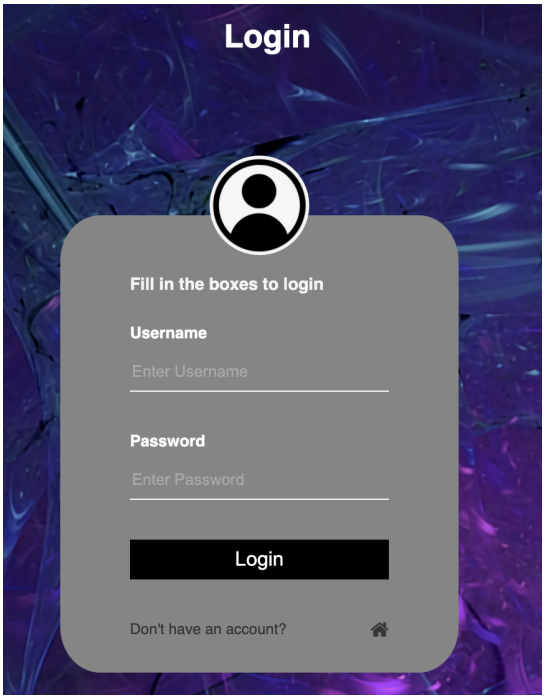
```
//////////////////////------------------------------ LOGOUT ----------------------------------\\\\\\\\\\\\\\\\\\\\\\\\

p.get('/logout', redirectLogin, (req, res) => {
req.session.destroy(err => {
  if (err) {
    return res.redirect('./')
  }
  res.send('<link rel="stylesheet" type="text/css" href="css/custom.css">' + '<p style= "color:#FFFFFF;">' + ('you are now logged out.') + '</p>'
  + '<br />' + '<a href=' + './' + '>Home </a>' + '<a href=' + './PopularGames' + '> PopularGames </a> ' + '<a href=' + './Complaints' + '> Complaints</a>'
  + '<a href=' + './Forum' + '> Forum </a>' + '<a href=' + './login' + '> login </a>' + '<a href=' + './logout' + '> Logout</a>');
})

```

Fig 42.2: Logout code

Logout page is linked to the login page. Without the login code it will just redirect to the login html page whenever clicked. The logout page when working is supposed to end the session and redirect to a res.send letting the user know the task is complete.

```
//////////////////////------------------------------ FORUM PAGE ----------------------------------\\\\\\\\\\\\\\\\\\\\\\

app.get('/forum', redirectLogin, function (req, res) {
  var MongoClient = require('mongodb').MongoClient;
  const url = "mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test";
  MongoClient.connect(url, function (err, client) {
    if (err) throw err;
    var db = client.db('GameSpy');
    db.collection('Messages').find().sort({ TimeDate: -1 }).toArray((findErr, results) => {
      if (findErr) throw findErr;
      else
        //prints the messages into a webpage
        res.render('forum.ejs', { availablemessage: results });
      client.close();
    });
  });
})

app.post("/MessageAdded", (req, res) => {
  var MongoClient = require('mongodb').MongoClient;
  var url = "mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test";
  var date = new Date();

  MongoClient.connect(url, (err, client) => {
    if (err) throw err;
    var db = client.db('GameSpy');
    //enters the message and all the datas into the database Messages
    db.collection("Messages").insertOne({
      username: req.session.userId,
      message: req.body.message,
      TimeDate: date.getDate() + '/' + date.getMonth() + '/' + date.getFullYear() + ' (' + date.getHours() + ':' + date.getMinutes() + ':' + date.getSeconds() + ') '
    }, function (err, result) {
      if (err) throw err;
      res.send('<link rel="stylesheet" type="text/css" href="css/custom.css">' + '<p style= "color:#FFFFFF;">'
      + ("Hello " + req.session.userId + " You have submitted the following message: " + req.body.message + '</p>')
      + '<br />' + '<a href=' + './' + '>Home </a>' + '<a href=' + './PopularGames' + '> PopularGames </a> '
      + '<a href=' + './Complaints' + '> Complaints</a>' + '<a href=' + './Forum' + '> Forum </a>' + '<a href=' + './login' + '> login </a>' + '<a href=' + './logout' + '> Logout</a>')

    });
    client.close();
  });

})
```

Fig 43.1: Forum code



When the forum page is clicked on, it redirects the user to the login page. This is due to the ‘redirectLogin’ in the parameter of our app GET post, which can be used for access controls.

Fig 43.2: Login to access forum

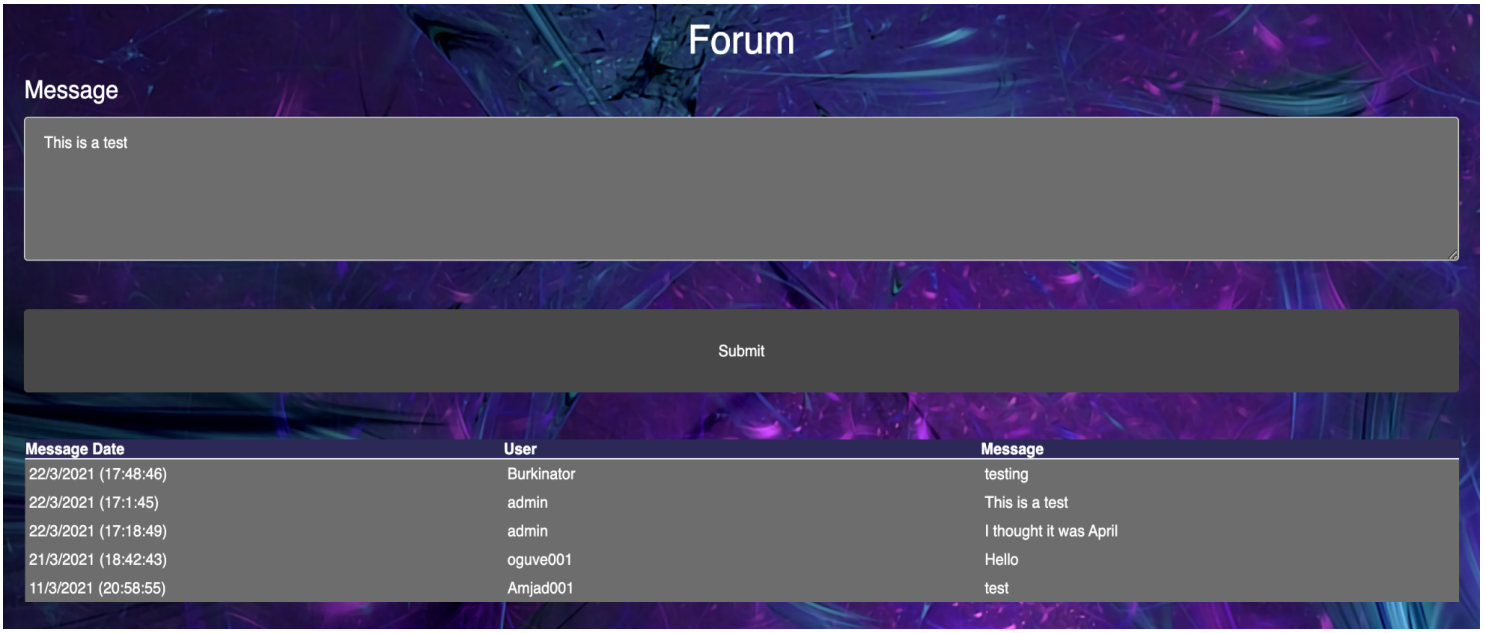


Fig 43.3: Forum page

When the user has successfully logged in and now enters the forum page, the following webpage would be presented. A message box for the user to enter their message or questions and a huge submit button below it. Underneath it is the forum table, where all the previously added messages appear

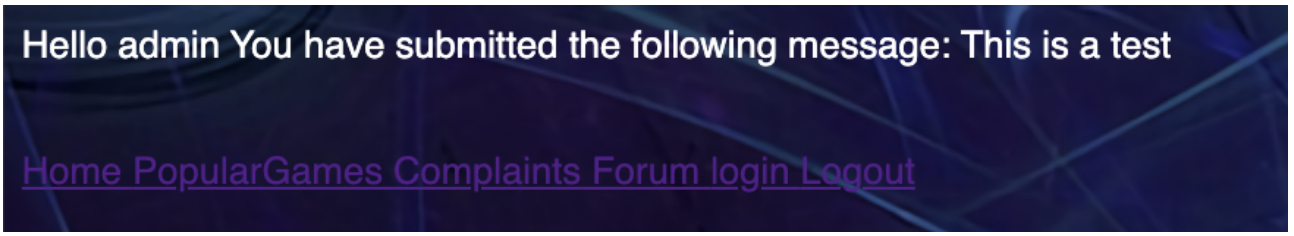


Fig 43.4: Successful forum post

Once a message is submitted, it is saved to the Database into the collection 'Messages'. This collection has the following fields; message, users and TimeDate. The TimeDate is a JavaScript Date function, which gets the exact current Date and Time. A simple message will be shown to the user to notify them that their message has been successfully submitted and saved to the database

Fig 43.5: Post showing on forum page

After the message has been shown, once the user enters the forum page, they can see their message appended to the forum box below. There is one slight issue with the TimeDate as it shows the previous month rather than the current month. This is because the date.getMonth() is indexed at 0 and in order to get the current month, we must add 1.

```
252  //////////////////////------------------------------ API PAGE ------------------------------------\\\\\\\\\\\\\\\\\\\\\\\\
253
254  app.get('/api', function (req, res) {
255    var MongoClient = require('mongodb').MongoClient;
256    var url = "mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test";
257    MongoClient.connect(url, function (err, client) {
258      if (err) throw err
259      var db = client.db('GameSpy');
260      db.collection('ALL').find().toArray((findErr, results) => {
261        if (findErr) throw findErr;
262        else
263          //shows the games as a json file
264          res.json(results);
265      });
266    });
267  });
268
```

Fig 44.1: API code

[{"_id":"607856f3fab6da1be442fc66","appid":"10","name":"Counter-Strike","release_date":"36831","english":"1","developer":"Valve","publisher":"Valve","platforms":"windows;mac;linux","required_age":"0","categories":"Multi-player;Online Multi-Player;Local Multi-Player;Valve Anti-Cheat enabled","genres":"Action","steamspy_tags":"Action;FPS;Multiplayer","achievements":"0","positive_ratings":"124534","negative_ratings":"3339","total_ratings":"127873","average_ratings":"97.38881547","average_playtime":"17612","median_playtime":"317","owners":"10000000-20000000","steam_price":"7.19","ubisoft_edition":"null","ubisoft_price":"null","epicgames_url":"null","epicgames_price":"null","origin_price":"null","origin_edition":"null","GOG_price":"null"},
{"_id":"607856f3fab6da1be442fc67","appid":"20","name":"Team Fortress Classic","release_date":"36251","english":"1","developer":"Valve","publisher":"Valve","platforms":"windows;mac;linux","required_age":"0","categories":"Multi-player;Online Multi-Player;Local Multi-Player;Valve Anti-Cheat enabled","genres":"Action","steamspy_tags":"Action;FPS;Multiplayer","achievements":"0","positive_ratings":"3318","negative_ratings":"633","total_ratings":"3951","average_ratings":"83.97873956","average_playtime":"277","median_playtime":"62","owners":"5000000-10000000","steam_price":"3.99","ubisoft_edition":"null","ubisoft_price":"null","epicgames_url":"null","epicgames_price":"null","origin_price":"null","origin_edition":"null","GOG_price":"null"},

Fig 44.1: Return as JSON format

The API page collects all the data from our collection 'ALL' and displays results in a JSON format. There is no 'redirectLogin' in the parameter of our app.get therefore anyone has access to this page.

```
16  //////////////////////------------------------------ SEARCH PAGE ------------------------------------\\\\\\\\\\\\\\\\\\\\\\\\
17  //home Page
18  app.get('/', function (req, res) {
19
20    res.render("home.html");
21  });
22  // search result for games
23  app.get('/search-result', function (req, res) {
24    //calls the database
25    var MongoClient = require('mongodb').MongoClient; //retrieve
26    var url = "mongodb+srv://GameSpy:gamespy123@gamespy.inxg2.mongodb.net/test"; // set url
27    MongoClient.connect(url, function (err, client) { //connect to the db
28      if (err) throw err;
29      var db = client.db('GameSpy'); // name of db
30      db.collection('ALL').find({ name: { $regex: new RegExp(req.query.keyword, "i") } }).toArray((findErr, results) => { // finds name of books with specific key words
31        if (findErr) throw findErr // outputs error
32        else {
33          res.render('gamepage.ejs', { gametitle: req.query.keyword, availablegame: results }); // otherwise produce output
34          client.close(); // closes all open connections
35        }
36      });
37    });
38  });
39
```

Fig 45.1: Search Result code

We checked to see if we're able to connect to our database using the retrieve and set functions and then call the db using db.collection. If this was the case then we can search using RegExp and then render our ejs files.

```
//////////////////////------------------------------

app.get('/popular games', function (req, res) {
  res.render("popular games.html");
});
```

Fig 46: Popular games code

The above code takes the user to the popular games pages. Here the user can see all current popular games. We have tested the page and it works accordingly. The images also redirect the user to the game page and its contents.

Action to test	Completed Y/N	Evidence
Functioning navigation bar links	**Y**	https://gyazo.com/e56b82109ca51cc8fb63911b7ee6b493
Web forms: testing if the forms are passing the right information and not letting users type invalid characters within the input boxes	Y	If you enter any characters that are not allowed in the specific fields for example in the register page then the webapp will refresh the pages and ask the user to try again. As an example if you try to enter text in the email field when it is not an email, registering will fail and it will take you back to the register page so you can try again.
Compatibility: ask the users to test the website with as many devices as they can to see if our website is responsive.	Y	Our website was designed for computers in mind because we thought that most people would shop for their PC games using a PC. However, when we simulated a mobile device we found that most of our website is responsive apart from the images which scaled in weird ways.
Images : do all the images render? Are there any issues with certain image references?	Y	Images all render properly however when zoomed in or out they do not scale relative to the screen or device size. This is something we will need to look at. (1)
Fonts	Y	Went through all the pages and made sure the fonts are consistently the same. All text

		is also aligned in the same way.
HTML markup validation service	Y	Uploaded all of our pages to the W3C markup validation service and tested it against the standards which should be followed when using HTML. (2). These revealed some issues which we Fixed in the next iteration (3).

(1) Page perspective

Zoomed in.

Normal perspective.

(2) html validation

Search.html

Nu Html Checker

This tool is an ongoing experiment in better HTML checking, and its behavior remains subject to change

Showing results for search.html

Checker Input

Show source outline image report Options...

Check by file upload Choose file No file chosen

Uploaded files with .xhtml or .xht extensions are parsed using the XML parser.

Check

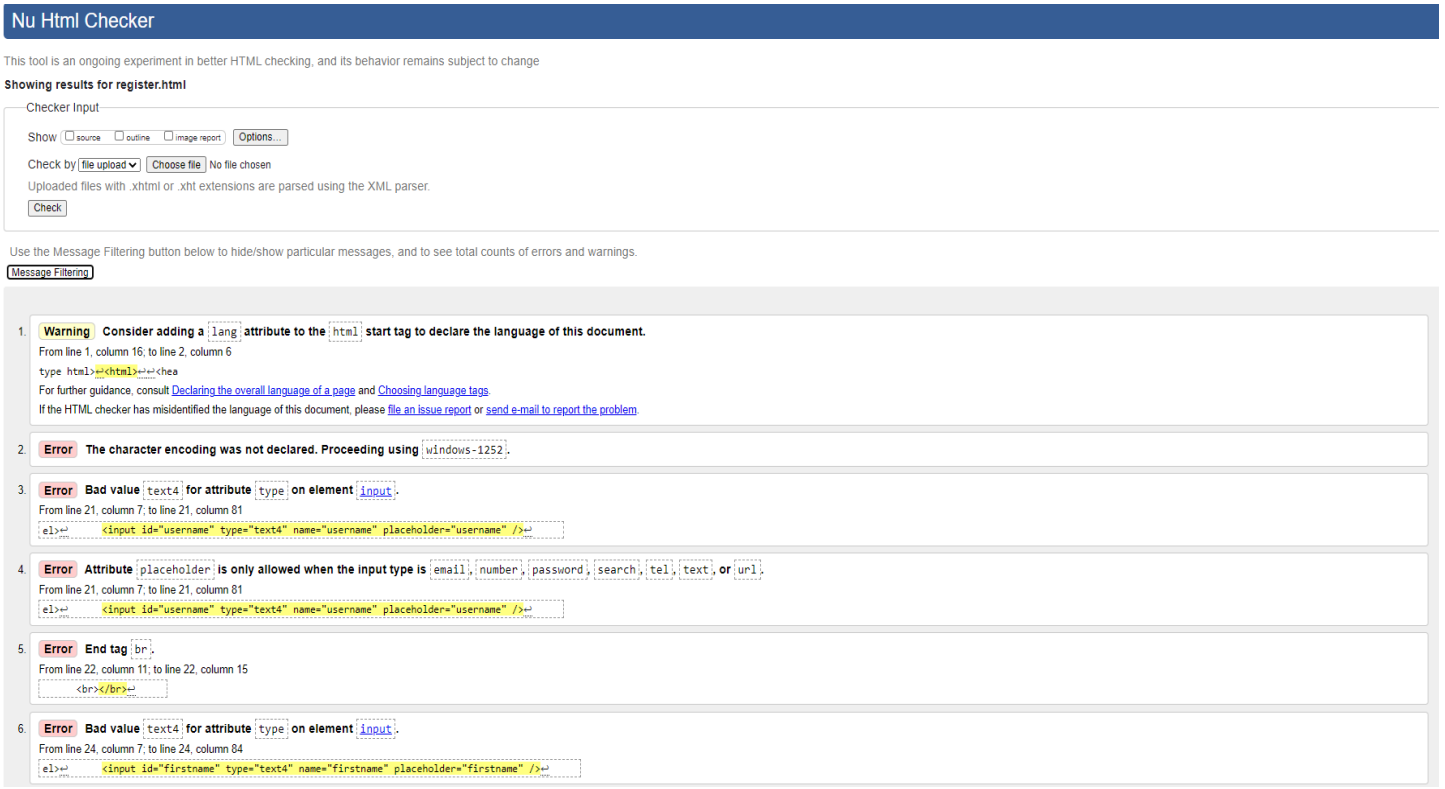
Use the Message Filtering button below to hide/show particular messages, and to see total counts of errors and warnings.

Message Filtering

1. Error A `charset` attribute on a `meta` element found after the first 1024 bytes.
   At line 25, column 69
   `charset="UTF-8" />↵</head>↵↵<`
2. Error Attribute `charset` not allowed on element `meta` at this point.
   From line 25, column 3; to line 25, column 72
   `ipt>↵↵↵↵ <meta http-equiv="Content-Type" content="text/html" charset="UTF-8" />↵</hea`
   Attributes for element `meta`:
   Global attributes
   `name` — Metadata name
   `http-equiv` — Pragma directive
   `content` — Value of the element
   `charset` — Character encoding declaration
3. Error Bad value `text/html` for attribute `content` on element `meta`: The legacy encoding declaration did not start with `text/html;`.
   From line 25, column 3; to line 25, column 72
   `ipt>↵↵↵↵ <meta http-equiv="Content-Type" content="text/html" charset="UTF-8" />↵</hea`

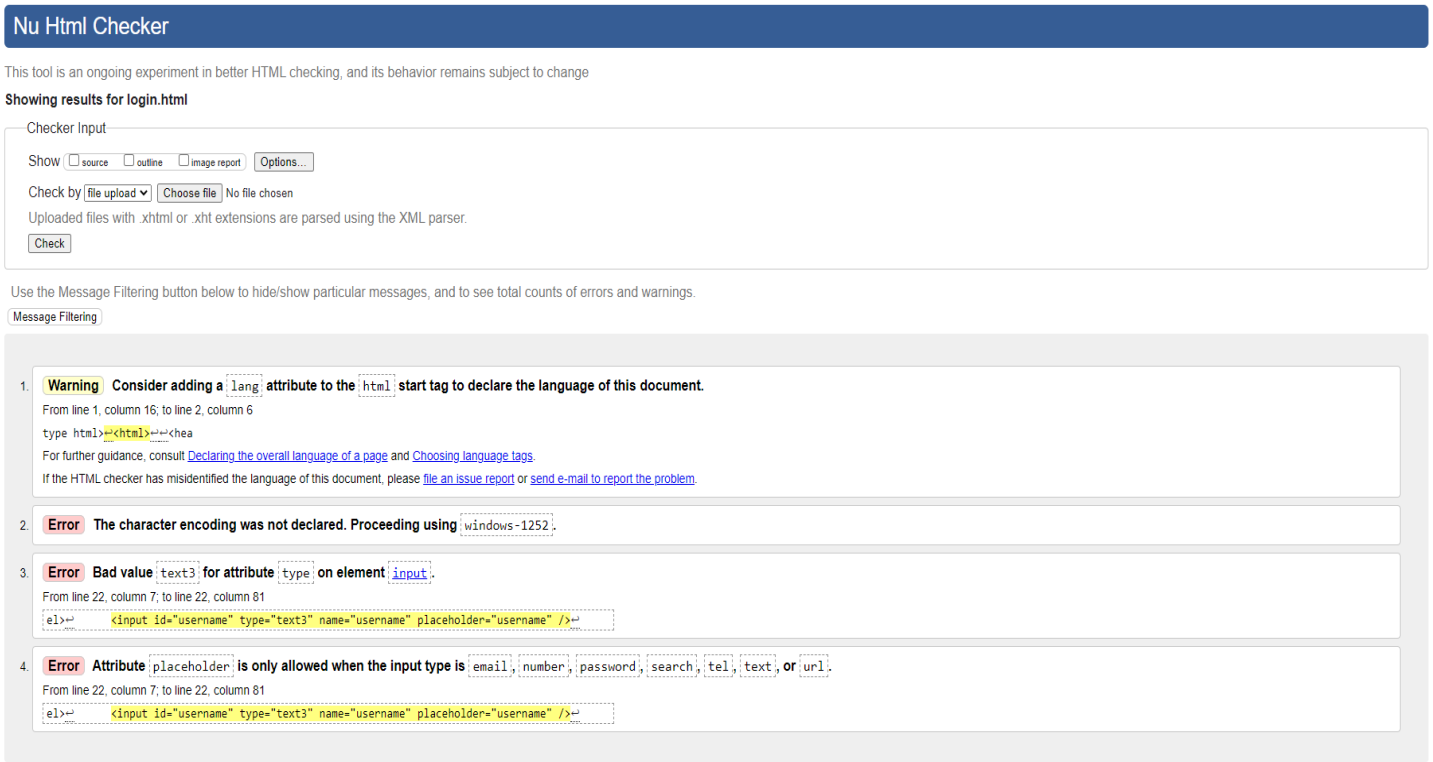
This is the search.html going through html validator

Register.html



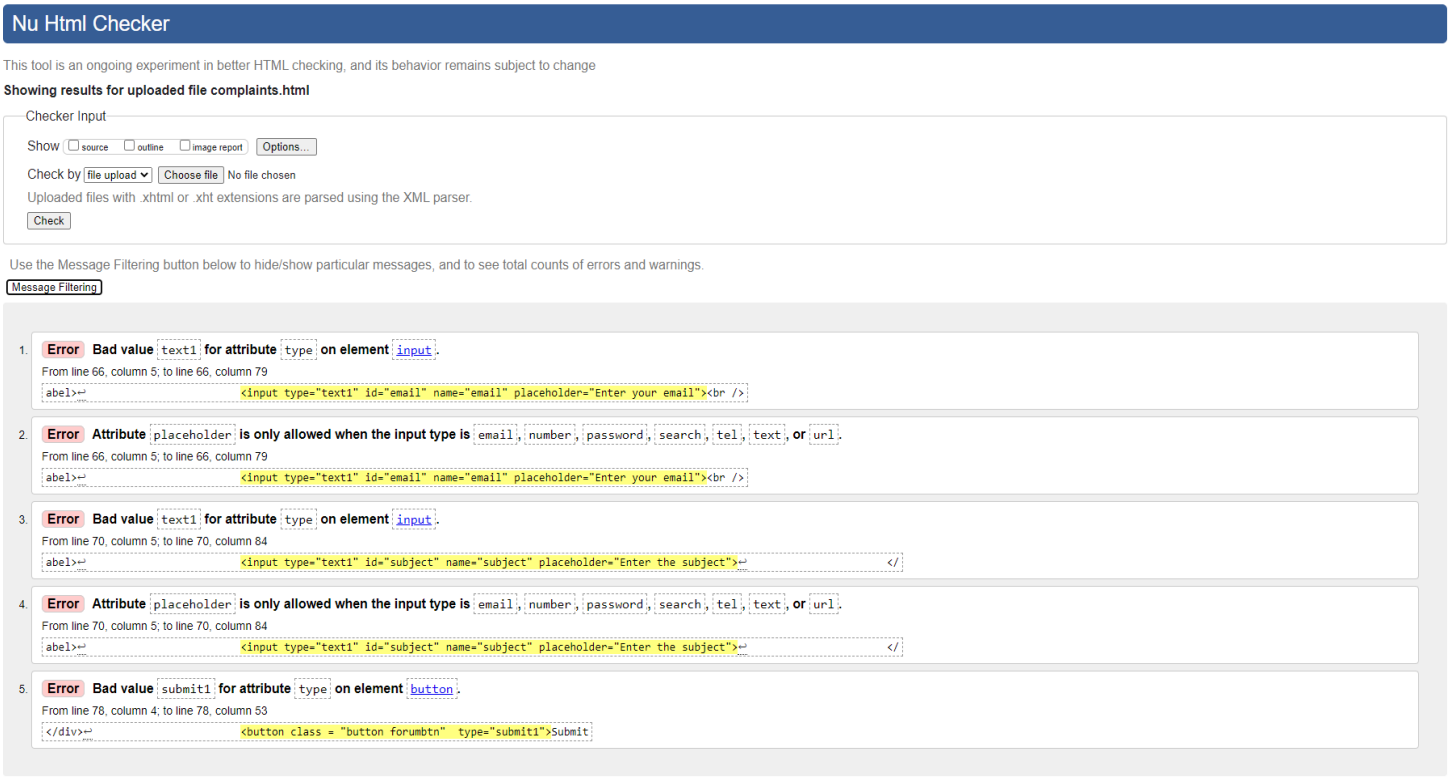
This is the register.html going through html validator

Login.html



This is the login.html going through html validator

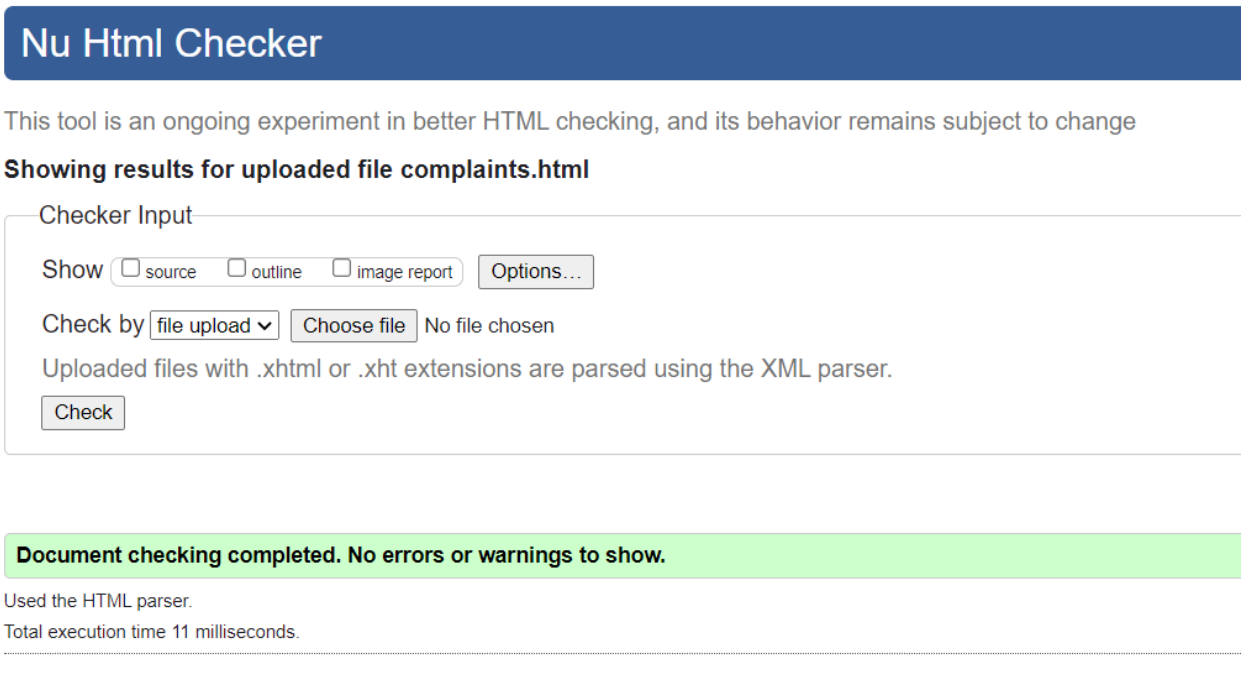
Complaints.html



This is the complaints.html going through html validator

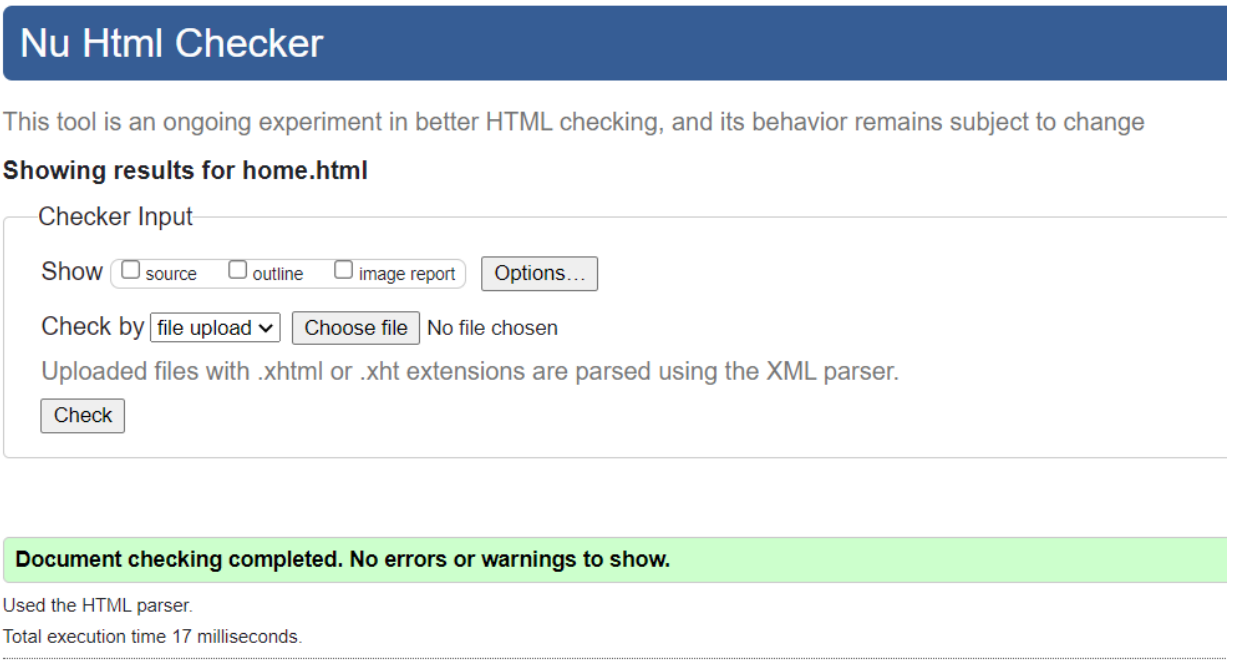
(3)

Complaints.html



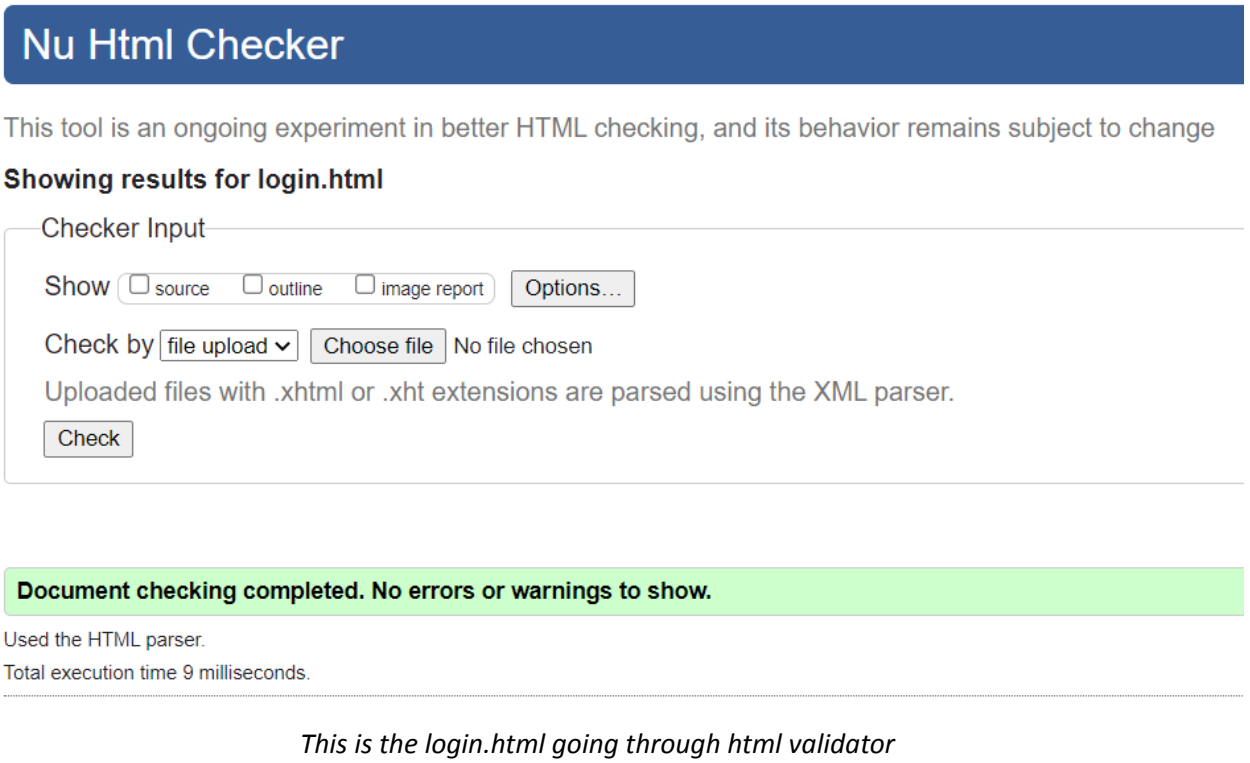
This is the complaints.html going through html validator

Home.html



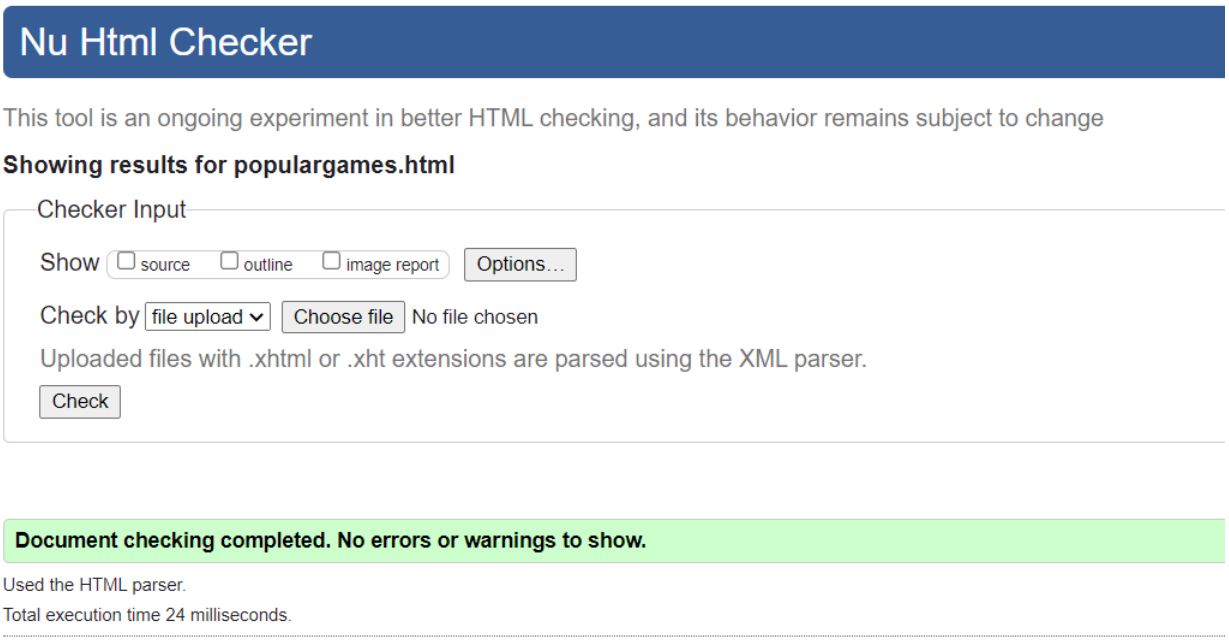
This is the home.html going through html validator

Login.html



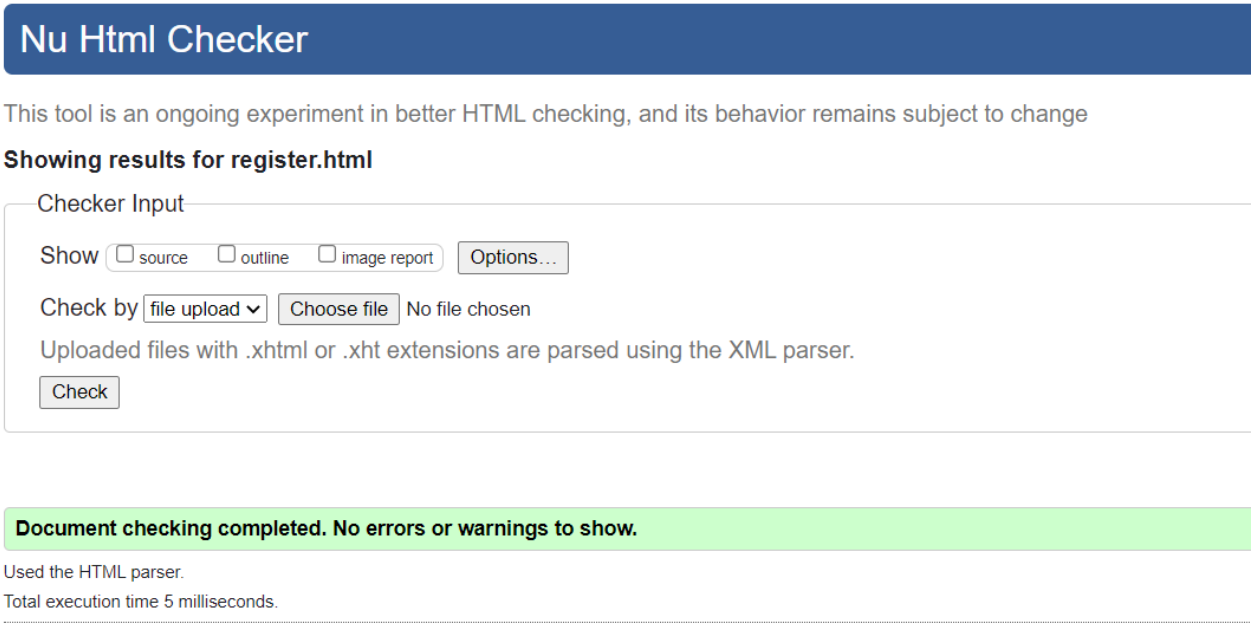
This is the login.html going through html validator

populargames.html



This is the populargames.html going through html validator

register.html



This is the register.html going through html validator

Test Table

Purpose of test	Test Data	Expected Result	Achieved Y/N
Does the website load?	Nodejs Page does not load User can see errors on the page	Home page should open with all text and images loaded. All pages load as intended.	Y
Complaints and forum page can not be accessed unless logged in	Nodejs Page does not load User can see errors on the page	When selecting complaints or forums from navbar get prompted to login page. Can not be accessed if logged in.	Y
Search for a game	Nodejs Page does not load User can see errors on the page	Typing in search bar populates the all games containing the input in a dropdown list format. And when clicked are taken to the game page for that particular game.	Y
Register an account	Nodejs Page does not load User can see errors on the page	Users can register for an account. The input requires the validations to be met like actual email address, no empty boxes and min 8 character password.	Y
Login to account	Nodejs Page does not load User can see errors on the page	Users can login to account. The input requires the validations to be met with no empty boxes.	Y
Log out of account	Nodejs Page does not load	User logs out and is unable to access user only pages.	Y

	User can see errors on the page		
Popular games page, games linked to thier correct page	Nodejs Page does not load User can see errors on the page	Page loads as intended, dynamic so changes to screen resolution. Linked to game pages of popular games.	Y
Complaints page	Nodejs Page does not load User can see errors on the page	Users can access the complaints page and submit a complaint.	Y
Forum page	Nodejs Page does not load User can see errors on the page	Users can access the complaints page and submit a forum addition.	Y

UI evaluation

The user interface for our website is mainly built on Bootstrap. Bootstrap makes creating the UI seamless and easy to integrate into our website. For example, our navbar was made using bootstrap and it provides an easy way for the users to navigate the website and sift through the pages in order to get to where they need to.

When the user searches for a game they are presented with a graph which shows the live player data and the methodology behind this is that it will help the user make a decision on whether or not they would like to buy the game based on the player count. This is especially important for games which are mainly multiplayer. We tried experimenting with changing the colours of the graph however, since we are pulling it in an <iframe> this proved difficult to do. What we found we could do however, was we could change the size and stack the graphs on top of each other and this makes it easier when the users are searching for games which are similar in name.

In the popular games page we have the top popular games listed so if any person is finding it hard to decide on a game then they can refer to this and it will help them decide based on what people are playing. The user is presented with a grid of images and descriptions underneath them. The

pictures help users visualise what the game may be like such as if it's a shooter or a driving simulator.

User Testing

User testing was done with a handful of people that were within our target audience range. We got them to use the site and asked them for their views on three major points, accessibility, aesthetic and comparison of prices from different retailers.

Most users had no accessibility issues while using the website, but one user had minor issues when accessing this website from firefox the submit box will come as a submit query and the search bar will have a condensed shape.

In the aesthetic sense of the website the users found the grey navbar and footer too dull for games comparison websites making it less appealing. Only use of white text throughout the site makes the website seem bland and also the use of default text.

The comparison aspect of the price clearly indicates the current prices of the games on the main retailers and also provides the right game related data like genre and developer. The player count was also found very helpful as it let the user know if the game still had a high player base, this was really liked by the users. The setup for the data on the game page was quite simple and users suggested a more appealing approach.

After all the feedback was collected we had a good indication of what was working and what we needed to improve upon. We worked on the accessibility issue first and fixed it by bug testing our css and html code. We kept the navbar as it is as it had a nice contrast between the page background colour. The feedback gave us an overview on how the final build of the website is the good and the bad, which can be very helpful in our future projects as this allowed us to understand the user mindset. However, something that we can say about our “poor” UI is that in this scenario it may appeal more if it is unattractive. If you go on a slick site like Apple’s, you think it is “expensive”. Whereas, if you go on eBay you think “cheap” - which is what they are. Sites like ours may be deliberately kept looking outdated because we implant ideas to find a bargain here.

Formative Evaluation

Initial Approach To Stakeholders

During the early planning stages of our web app, we approached our main stakeholder, Founder and CEO of Green Man Gaming: Paul Sulyok, to gather his opinion on what he thought of our initial development idea.
We carried out an interview and here is the transcript from it:

1. **First of all, what do you think of the concept of this idea?**

“Price comparison websites in general are amazing especially with how much e-commerce has boomed over the past decade. And when you target a market as big as the video games market, you’re pretty much hitting the nail on the head in terms of demand. Specifically, looking at your points of differences and product brief, if you’re able to implement them as well as fulfilling the basics of a price comparison website then I think you shall do very well. I’m especially looking

forward to the way you show live player counts since the only way to view this kind of information is through external sites."

2. **What features would you like the website to have?**

"Adding on to your list of features already, I think it would be great if there was a way to increase communication between yourselves and the consumers. Perhaps implementing a forum of sorts to enable interaction, this would also allow consumers to talk to each other with yourselves also chiming in now and again. They would be able to post their own deals they've found and just have general conversations about games."

3. **What is your definition of success for this project?**

"The way I measure project success is by the outcome of the project that is acceptable to the end user, customer and the stakeholder. You want to try and keep to your original timeline this way your project management will be successful. Quality is also important - ensure that both the UI and UX are at high standards. Stakeholder satisfaction is difficult to measure but I would advise you to keep in contact with us so that we can engage and support you if required."

4. **Are there any concerns you have?**

"I think this kind of idea is quite ambitious, especially with the deadline that you have set yourself. There are alot of things you have to carry out and I wish you luck. Other than that, not necessarily; but you do state that the only income you would receive are through ads, which is strange. Some price comparison websites receive pay for the traffic they send retailers and in exchange they appear on their features list. Additionally, you could use a cost per click model and a cost per acquisition model so that your business is more profitable. It is amazing that you have a customer orientated approach to your market but it is also beneficial for you to think about yourselves too. Other than that I wish you good luck! "

Showing Our Final Product

Taking into account all these points during the whole development process, we presented our final product - or what we thought would be our final product. Paul Sulyok was encouraged with how much progress we had made in very little time. He said that we created all the essential ingredients needed to create an amazing price comparison website but that we are still far from there yet. Although, he pointed out that there is no actual technical comparison being carried out - none being done from the website itself. However, he was mostly disappointed with the fact we had not made the website as fluid and dynamic as promised as this was one of our USPs. He suggested the importance of a responsive web design should not be ignored and that we should work on this before releasing it into the market.

Summary of Our Interaction

As the days progressed, we went on to conclude the development of our product keeping in mind some of the improvements that had been suggested to us. Unfortunately, we couldn't implement the comparisons due to a lack of proficiency within the team but we did ensure the website is as responsive as possible and have shown testing for this above. As a result, we've extensively reviewed the product for bugs and glitches, and we're happy to report that nearly all of the features we originally wanted to add all function well and do exactly what they're supposed to. Ultimately, we ended up with a completely functional product at the end of the 10-week development process, which we are content with and are confident that our partners will be as well.

Formative Evaluation

Unit testing was carried out to thoroughly check the features implemented, as depicted above. The majority of features result in anticipated behaviour. Any glitches and unusual behaviours that were discovered during the testing phases have yet to be resolved and are mentioned below, along with a brief explanation of their causes and possible solutions:

- Player count graphs not showing up for certain games
    - Caused by those games not being part of the SteamDB website so we cannot pull any information as there isn't any
        - Possible fixes are by creating our own graph but this requires data that other game publishers are not making public

- Game Average Rating showing null for certain games
    - Similar to the cause above - not part of Steam and also not available elsewhere
        - Any updated datasets in the future

- Logout button constantly shows even if user is not logged in
    - We don’t have any conditions to determine if a user is logged in so that logout only shows when they are logged in
        - Possible fix is store the user login as an ID then use that in an IF condition so that this isn’t the case

- Horizontal scrollbar appears on login and register pages
    - Likely due to bad CSS
        - Could be amended by reducing the form sizes but not sure as to the root of the issue

- When submitting a forum message, the date shown is incorrect - month is wrong and is shown as a month before e.g April (4) becomes March (3)
    - getDate function could be wrong or user’s date system could be faulty
        - Amending the function is a possible fix

The features we've introduced have very few unexpected behaviours. We referred to suggestions made in user consultations and concept presentations regularly during production, attempting to adapt our design as closely as possible to the specifications they outlined.

We're confident that what we've designed includes both the essential functionality of any website and many of the features requested. Whatever input we get, we are confident that what we've built will serve as a stable base for future growth, even if requirements change.

Final Iteration

After some extensive and robust testing from our side and while taking in all the feedback we received from our users and stakeholders, we created our now most complete application. Our audience really wanted an aesthetically pleasing website that is responsive. Although we could not improve on the UI due to the little time remaining we had, we did ensure every page was responsive. A responsive design makes our site mobile-friendly, improves the way it looks on devices with both large and small screens, and increases the amount of time that visitors spend on our site according to DigitalNext.

This is how our website now appears when made smaller:
https://gyazo.com/3708de7645866aa84cacae34b6685b9c

How we did it was using a mixture of inline and using our .css file. For example, for our images:

```
.sf1, .sf2, .sf3{
  /* SF logo images if any change it needed */
  width: 100%;
  max-width: 550px;
  height: auto;

  /*Scale down will take the necessary specified space that is 400px x 200px without stretching the image*/
  object-fit:scale-down;
}
```

Figure 47.1: Adding responsiveness to images in css file

“Object-fit:scale-down” decreases the image to the smallest version possible without affecting the visual quality.

And inline responsiveness example is:

```
<div class="ptableforum" style="overflow-x:auto;">
```

Figure 47.2: Adding responsiveness to images in css file

“Overflow-x:auto” specifies whether to display overflow content of this forum table in this case when it overflows at the left and right edges.

We’ve also improved the design of our search bar:

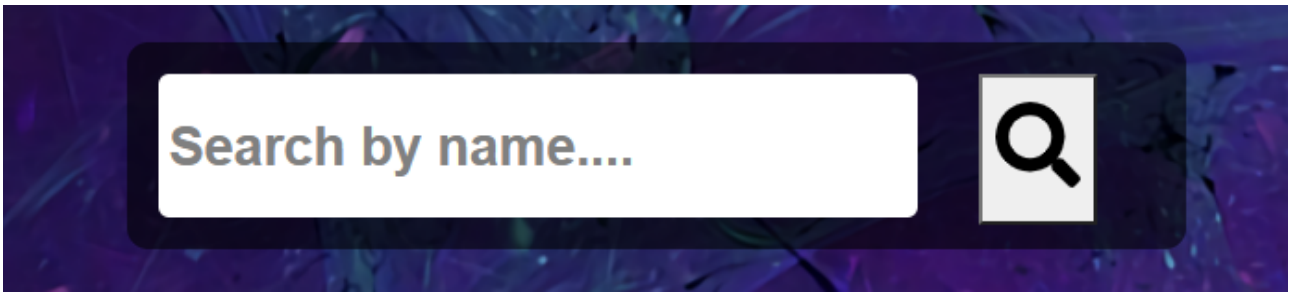


Figure 47.3: Improved search bar

If you compare it to Fig 19.4 we have made it stand out more by making the search box bigger and the icon too. This shows we’ve improved the accessibility, something that we lacked as pointed out in our user testing.

We've tried to match our design as closely to the stakeholder requirements they outlined as much as possible. We can not know for certain until we re-validate those designs so far we are getting closer to what they want.

Conclusion & Evaluation

Final Product

The following video will show a demonstration of the program working:

https://www.youtube.com/watch?v=-GJbqwDfXG4

Overall, we are disappointed with the end product and do not believe it has been quite as successful as we hoped. Despite us having explored in depth the viability of the project, both under technical and commercial aspects and evaluating and testing the basic functionality of the application & reviewing the coding used to construct it - we have not arrived at a long and stressful journey that we are proud of.

There are many things we could have done better. Starting off from our planning which from looking at it now was quite substandard. We should have delved into the technology required to complete features like price comparison further such as researching what framework would work best and how we would incorporate that to our front and backend. This essentially starts off a chain reaction where we are left scrambling of where to start and how to start it.

At this point we are already behind before any development has begun. However, once we actually commenced, the progress we were making was quite reasonable. This was mainly due to the purpose of working iteratively as it allowed for more flexibility for changes. Usually in software development, the focus would be on things like requirements, design and testing sequentially. Whereas, the iterative model effectively slims down each stage by allowing us to work on each stage simultaneously. This can be seen throughout our development process where we would pursue the coding and user requirements concurrently - this operation is then repeated several times through development. Nevertheless, it wasn't all smooth sailing. There were a few occasions where progress was halted due to us having to endure some difficulties. For example, between Iterations 4 and 10 where we could not find an efficient solution to our search results and instead had to settle with a quick fix which in the end caused us further problems as we continued the development. Looking back, instead of wasting valuable time, what we should have done is make use of forums such as Stack Overflow and then be patient with that and see if we would come across a more logical solution. But we did not do this and rather were too stubborn and thought we could do it ourselves.

Fundamentally, we were too ambitious. We made a lot of promises, set up many objectives and created an end goal to have the most fluid and dynamic website in the market only to not achieve most of them. Some were partly due to features not being readily available such as games datasets from other retailers apart from the main ones. But our issues came down mostly due to a lack of proficiency. We really did struggle with the most technical and complex areas of the application. Our predictive price history using machine learning was never going to work, price comparison was nonfunctional and design was inadequate from ours and our users perspective.

Although, we do want this to suggest we were imcompetent. We fully believe that if we had more time, we would be able to create a price comparison website which would compete with the likes

of Isthereanydeal and Allkeyshop. This may seem like a far fetched pledge but we feel we underestimated the difficulty of the work we needed to accomplish in the little time frame we had. This is because we could have spent more longer on complicated features rather than the need to rush through it.

As for testing, despite the process being lengthy and vigorous, it was quite enjoyable paying excruciating attention to the details and reporting bugs from some of the far corners of our website. The same thing could not have been said about our user testing. The lack of face-to-face with stakeholders and users caused difficulty which made it harder to organise and get the answers we required. But we did take advantage of remote softwares which did help us in observing what the users were doing and still get feedback in real time.

Despite this, the COVID-19 outbreak did not hurt us too much. As a group, collaboration was never impacted. The use of Discord and Whatsapp helped us stay on top of activities. All in all, there is no doubt room for improvement is vastly required but the tasks we accomplished during the project did help us gain a good understanding of the steps taken to create a successful project and the work needed to put in, in a professional and competitive environment. This will definitely assist us in future projects - for both education and work capacities.

Maintenance and Development

Perhaps a big benefit of this program is the length of the code. It is quite short and concise compared to many modern-day programs hence the execution times would be less lengthy. This is because as JavaScript is an interpreted language there would not be many lines of code that the interpreter would have to go through in order to execute the program. Additionally, our backend is strong enough to store all the relevant data and requirements and that we have tried to minimise redundant data as much as possible so that there is no wasted space which increases the likelihood of errors and inconsistencies. Because MongoDB is highly scalable and reliable it isnt likely any problems will arise in the future.

If the stakeholders would like to see some new features or add changes to the program, this can be easily done. Many of the lines have been commented so that it is easy to reference the function they want to improve on. Each function could be developed, tested and maintained very easily. This is made possible due to the programming being structured in a clear and simple way.

In addition, because web app and comparison methods have always stayed the same, there is unlikely anything required to be developed as the core functions of the program will never change. The only developments that would be needed are aesthetically. For example, both UI’s are not visually pleasing so my stakeholders may wish this to be changed while also making the program much more user friendly. Maintaining the program shouldn’t be an issue and it should become easier to continue further development in the future due to the rapid advancement of technology.

Future iterations could incorporate features such as the ability to actually predict future game prices. In addition, the website UI can be vastly improved visually as well as making it dynamic and animated. More features could be added such as being able to filter through the games and then being able to compare prices side by side to each other.

Currently, we do not believe this application is ready to be of any use in any public capacity at this time. What we've created is a good start towards demonstrating the feasibility of a price comparison platform like this. The product is not shippable in its current state, despite having a strong demonstration and basis for future growth. Before an initial alpha or beta release to customers, some steps will need to be taken and certain functionality introduced.

Bibliography

DATABASE comparison:
https://www.guru99.com/mongodb-vs-mysql.html#:~:text=MongoDB%20uses%20JavaScript%20as%20query,need%20a%20traditional%20relational%20database.
https://www.mongodb.com/compare/mongodb-mysql
https://medium.com/@rsk.saikrishna/when-to-use-mongodb-rather-than-mysql-d03ceff2e922
We used different websites to compare between different databases.

Databases used and conversion:
https://steamdb.info/
https://www.kaggle.com/beastovest/gog-games-with-reviews
https://steamdb.info/blog/steamdb-rating/
https://json-csv.com/
http://convertcsv.com/json-to-csv.htm

Web pages:
https://learn.gold.ac.uk/mod/page/view.php?id=861882
https://learn.gold.ac.uk/mod/page/view.php?id=861885
Used to create different web pages of our websites

Links between database and server:
https://www.mongodb.com/blog/post/quick-start-nodejs-mongodb--how-to-get-connected-to-your-database
https://stackoverflow.com/questions/41318354/mongodb-failed-to-connect-to-server-on-first-connect
https://www.w3schools.com/tags/ref_httpmethods.asp
Used to establish a relation between our server and database

HTML and Styling:
https://www.w3schools.com/css/
https://www.w3.org/TR/css-backgrounds-3/
https://getbootstrap.com/
https://www.w3schools.com/howto/howto_css_fullscreen_search.asp
https://www.w3schools.com/howto/howto_css_custom_scrollbar.asp
https://stackoverflow.com/questions/1733006/how-to-set-an-images-width-and-height-without-stretching-it
https://stackoverflow.com/questions/12070631/how-to-use-json-file-in-html-code
https://stackoverflow.com/questions/11471633/syntax-for-linking-documents-in-mongodb
https://stackoverflow.com/questions/13515141/html-javascript-how-to-access-json-data-loaded-in-a-script-tag-with-src-set
https://www.w3schools.com/css/css3_gradients.asp
https://getbootstrap.com/docs/4.0/components/navbar/
https://www.w3schools.com/html/html_id.asp
https://getbootstrap.com/docs/4.0/utilities/colors/

https://www.compose.com/articles/nodejs-mongodb-elasticsearch-and-postgresql-the-compose-grand-tour/
https://www.youtube.com/watch?v=HQRU8NZZNCQ
Simple references used for coding and styling

FIX search:
https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/
https://docs.mongodb.com/datalake/reference/pipeline/lookup-stage/
https://studio3t.com/knowledge-base/articles/mongodb-lookup-aggregation-stage/
https://jqueryui.com/autocomplete/
Used to fix search result page

Storing data:
https://learn.gold.ac.uk/mod/page/view.php?id=928912
https://solidgeargroup.com/en/hashing-passwords-nodejs-mongodb-bcrypt/
Used to store data including hashed passwords. We also implemented saltRounds and bcrypt function

Machine Learning:
https://www.youtube.com/watch?v=zPG4NjIkCjc
https://www.youtube.com/watch?v=JvS2triCgOY
https://www.youtube.com/watch?v=ZkjP5RJLQF4
https://www.youtube.com/watch?v=iAgYLRy7e20
https://github.com/amoldalwai/Price_Comparison_Website
Scatter plot using d3 library
https://gramener.github.io/d3js-playbook/scatter.html
Used to understand linear regression

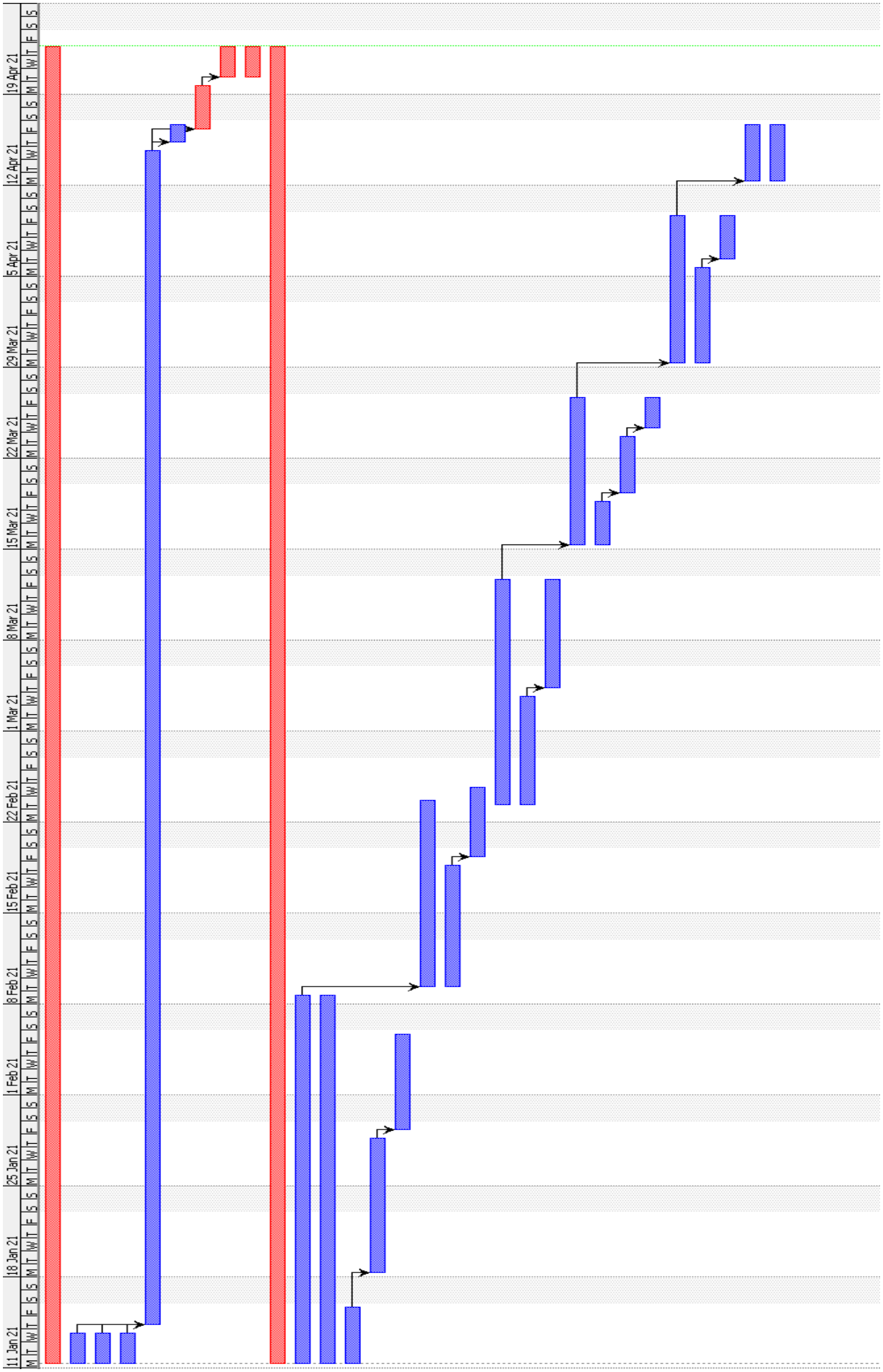
Appendix

Appendix 1: Gantt Chart (Project Libre)

Image 1

	Name	Duration	Start	Finish
1	SPRINT #7 (Report)	74 days?	11/01/21 08:00	22/04/21 17:00
2	Introduction	3 days?	11/01/21 08:00	13/01/21 17:00
3	Changes to implementation process	3 days?	11/01/21 08:00	13/01/21 17:00
4	Analysis	3 days?	11/01/21 08:00	13/01/21 17:00
5	Iteration/ development	65 days?	14/01/21 08:00	14/04/21 17:00
6	Inhouse testing	2 days?	15/04/21 08:00	16/04/21 17:00
7	User Testing	2 days?	16/04/21 08:00	19/04/21 17:00
8	Evaluation	3 days?	20/04/21 08:00	22/04/21 17:00
9	Apendix	3 days?	20/04/21 08:00	22/04/21 17:00
10	Bibliography	74 days?	11/01/21 08:00	22/04/21 17:00
11	SPRINT #1	21 days?	11/01/21 08:00	08/02/21 17:00
12	Search and filters	21 days?	11/01/21 08:00	08/02/21 17:00
13	View games	5 days?	11/01/21 08:00	15/01/21 17:00
14	Select a game	9 days?	18/01/21 08:00	28/01/21 17:00
15	Click the link	6 days?	29/01/21 08:00	05/02/21 17:00
16	SPRINT #2	11 days?	09/02/21 08:00	23/02/21 17:00
17	Price comparison	8 days?	09/02/21 08:00	18/02/21 17:00
18	Rating	4 days?	19/02/21 08:00	24/02/21 17:00
19	SPRINT #3	14 days?	23/02/21 08:00	12/03/21 17:00
20	Prediction of prices	7 days?	23/02/21 08:00	03/03/21 17:00
21	Price history	7 days?	04/03/21 08:00	12/03/21 17:00
22	SPRINT #4	10 days?	15/03/21 08:00	26/03/21 17:00
23	Live player count	4 days?	15/03/21 08:00	18/03/21 17:00
24	Complaints	3 days?	19/03/21 08:00	23/03/21 17:00
25	Complaints Admin Page	3 days?	24/03/21 08:00	26/03/21 17:00
26	SPRINT #5	10 days?	29/03/21 08:00	09/04/21 17:00
27	Login/ create account	6 days?	29/03/21 08:00	05/04/21 17:00
28	Home page	4 days?	06/04/21 08:00	09/04/21 17:00
29	SPRINT #6	5 days?	12/04/21 08:00	16/04/21 17:00
30	System Maintenance	5 days?	12/04/21 08:00	16/04/21 17:00

Image 2


11 Jan 21
18 Jan 21
25 Jan 21
1 Feb 21
8 Feb 21
15 Feb 21
22 Feb 21
1 Mar 21
8 Mar 21
15 Mar 21
22 Mar 21
29 Mar 21
5 Apr 21
12 Apr 21
19 Apr 21

Appendix 2 : Project Management (Azure DevOps)

Todo Board

Image 1

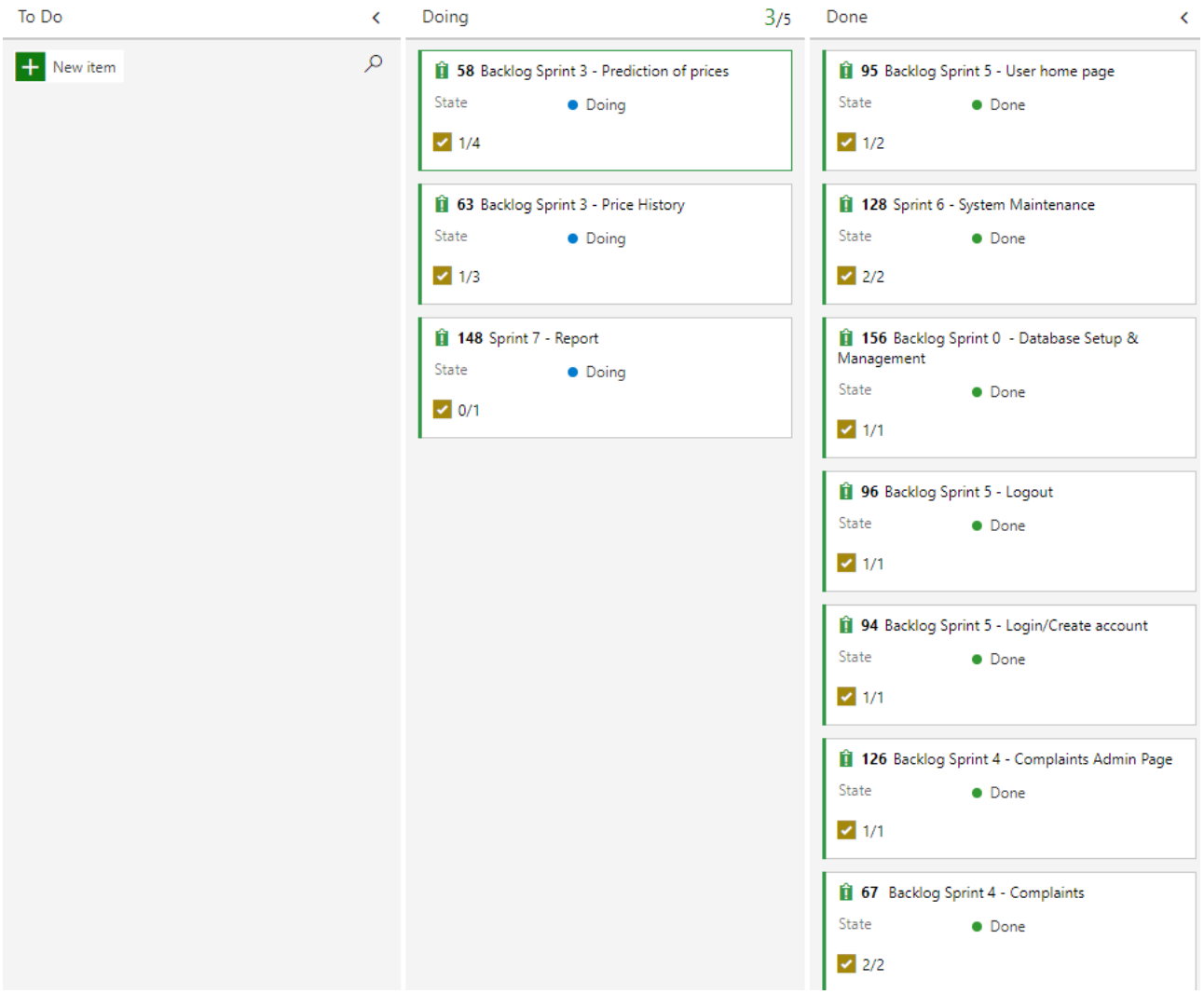


Image 2

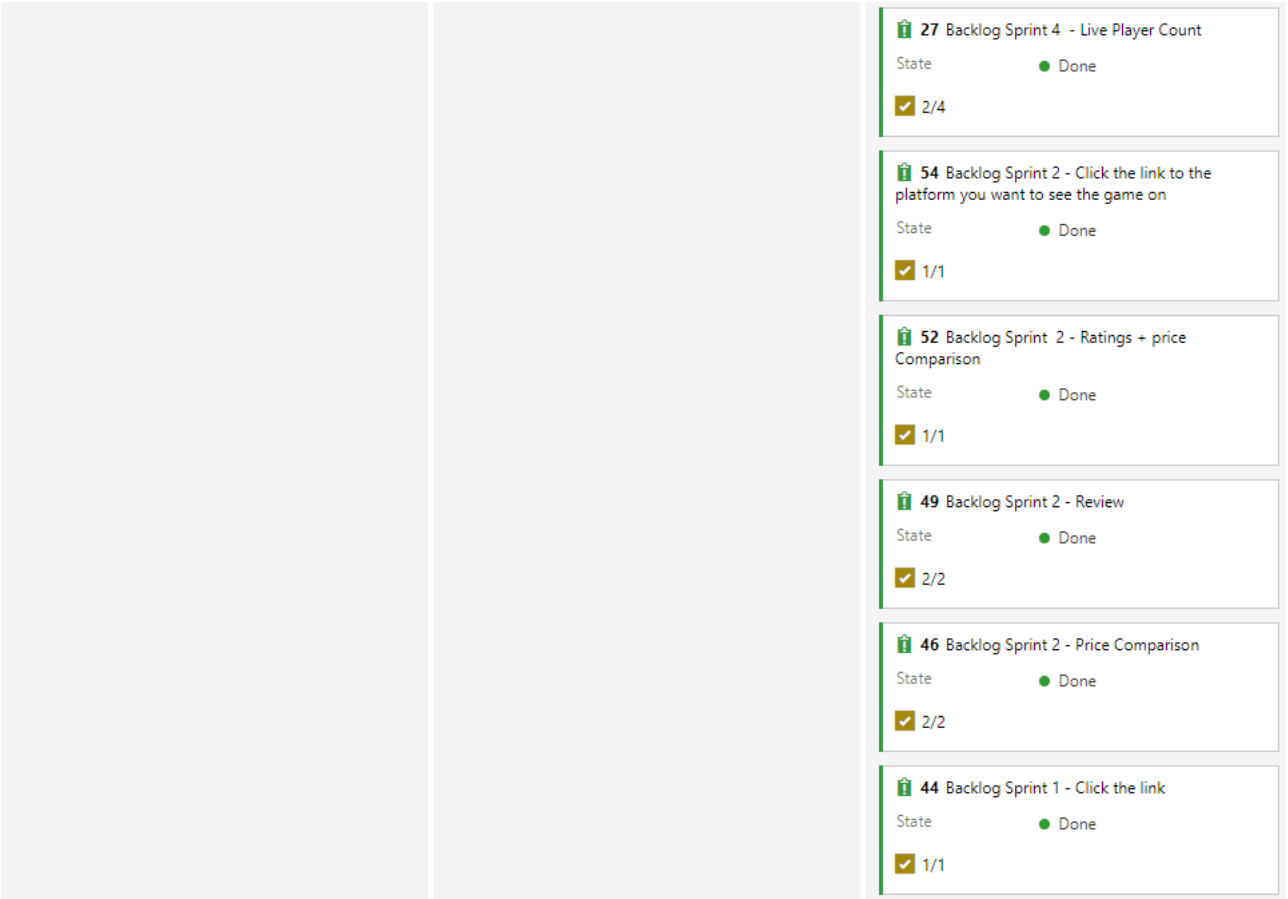
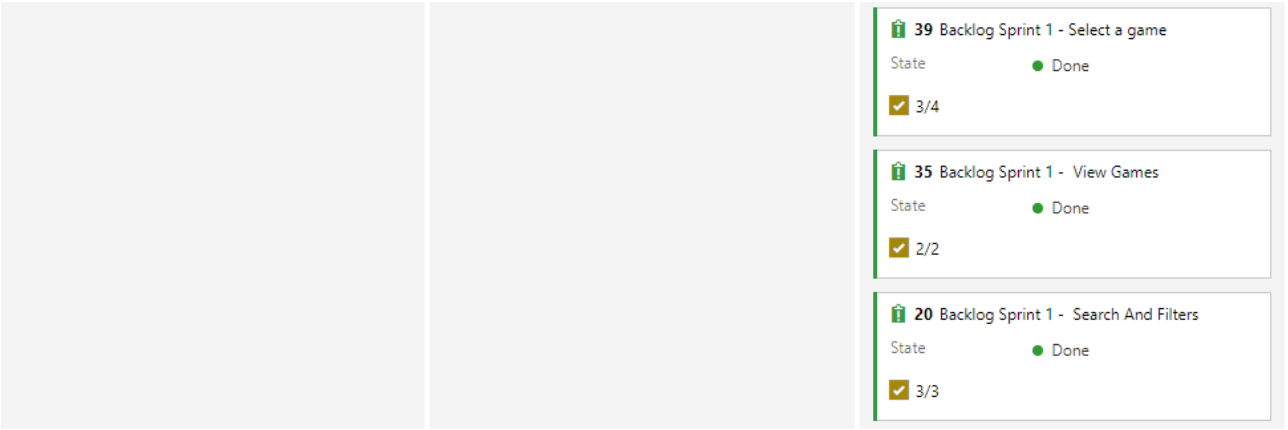
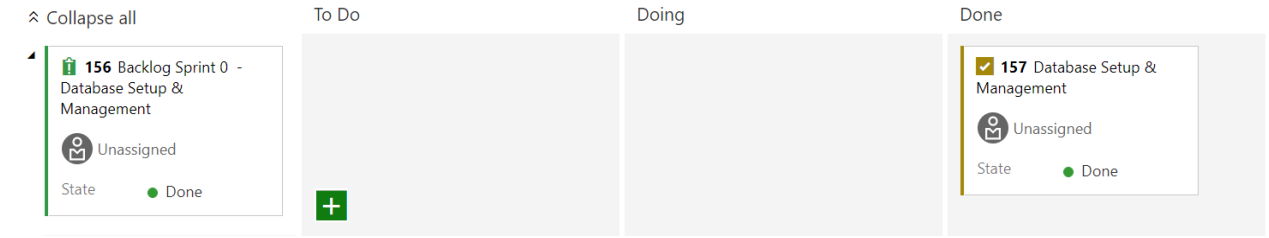


Image 3

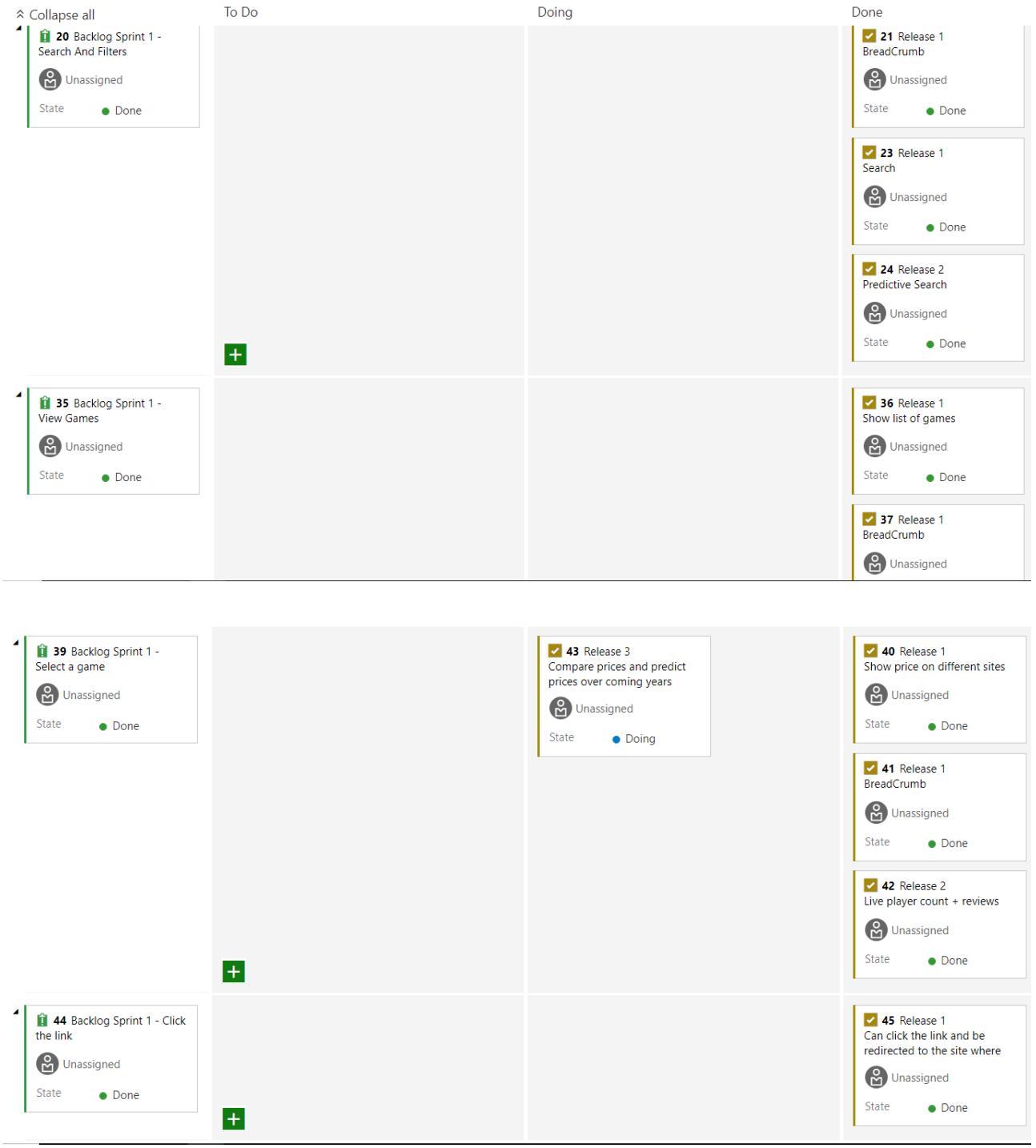


These are images of the sprints from Azure DevOps:

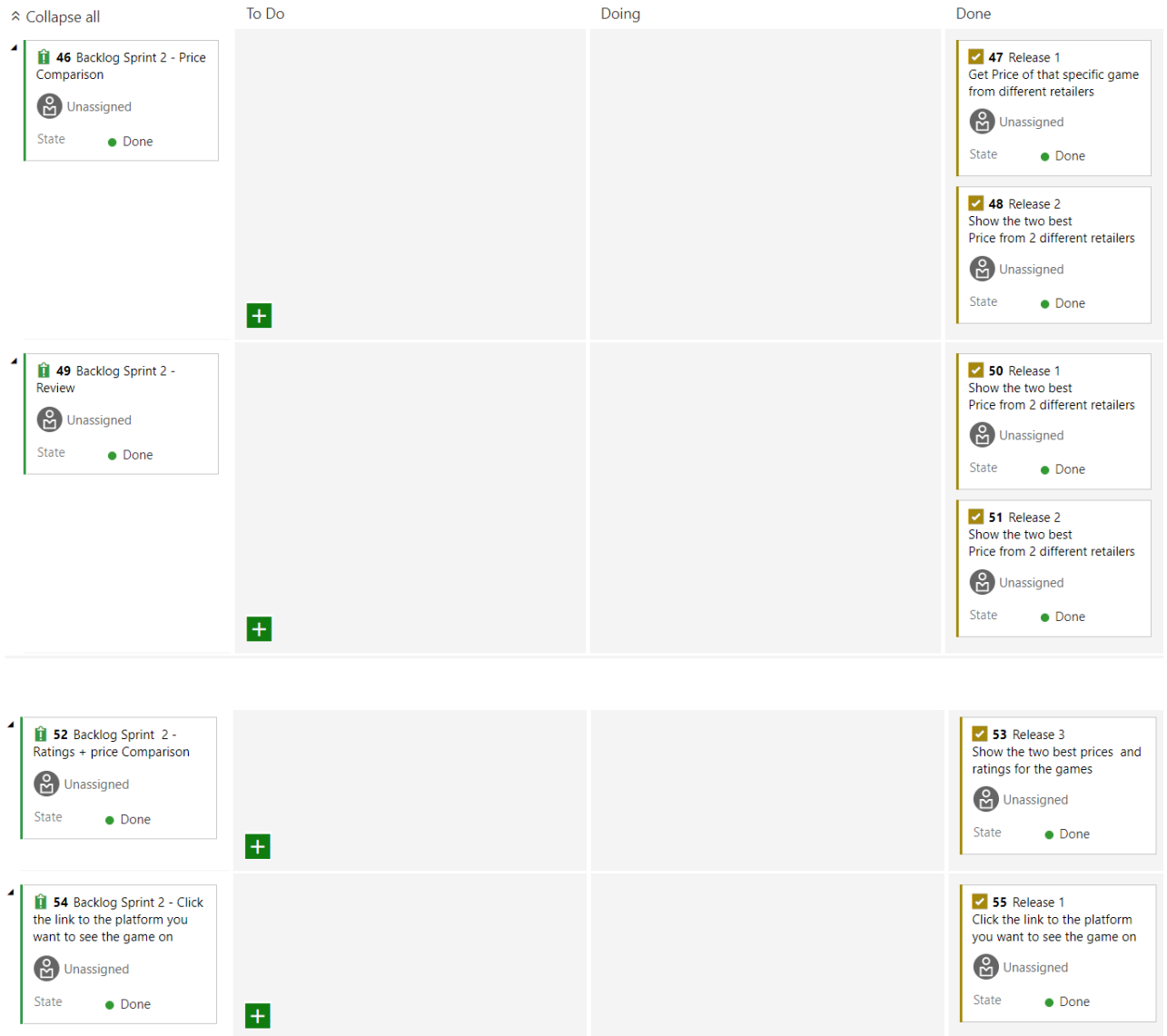
Sprint 0



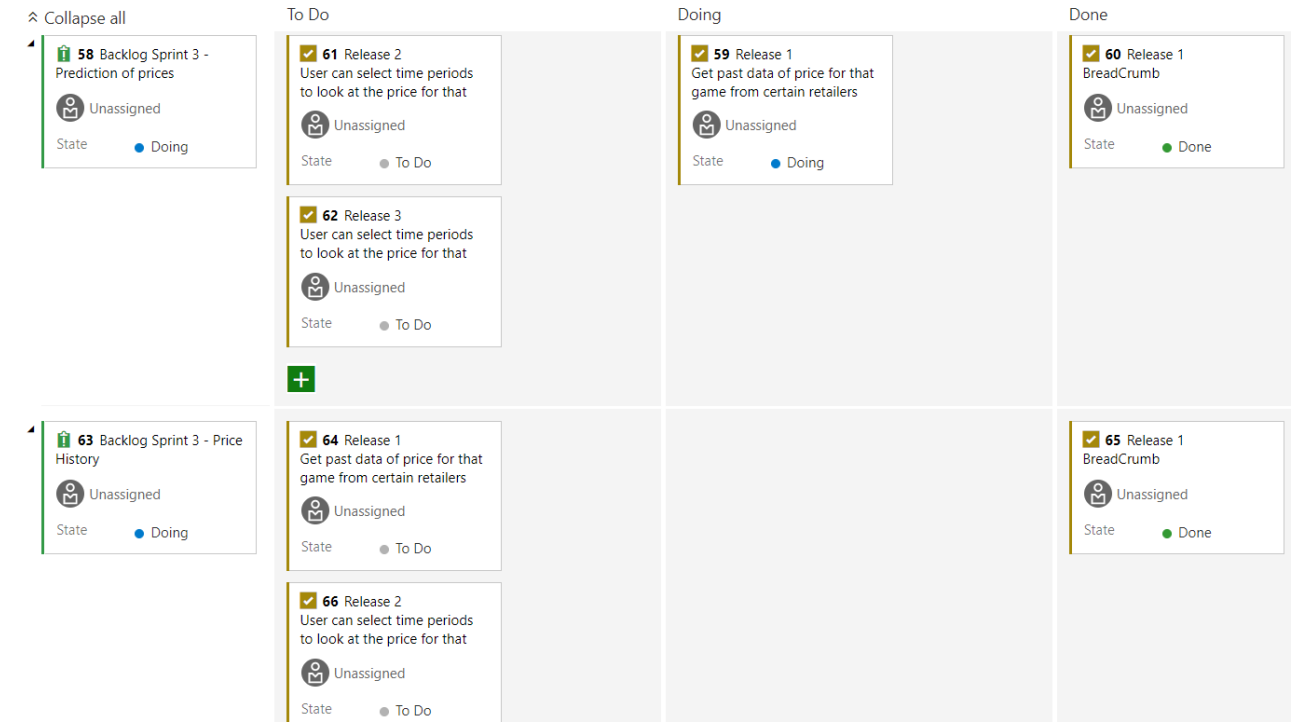
Sprint 1



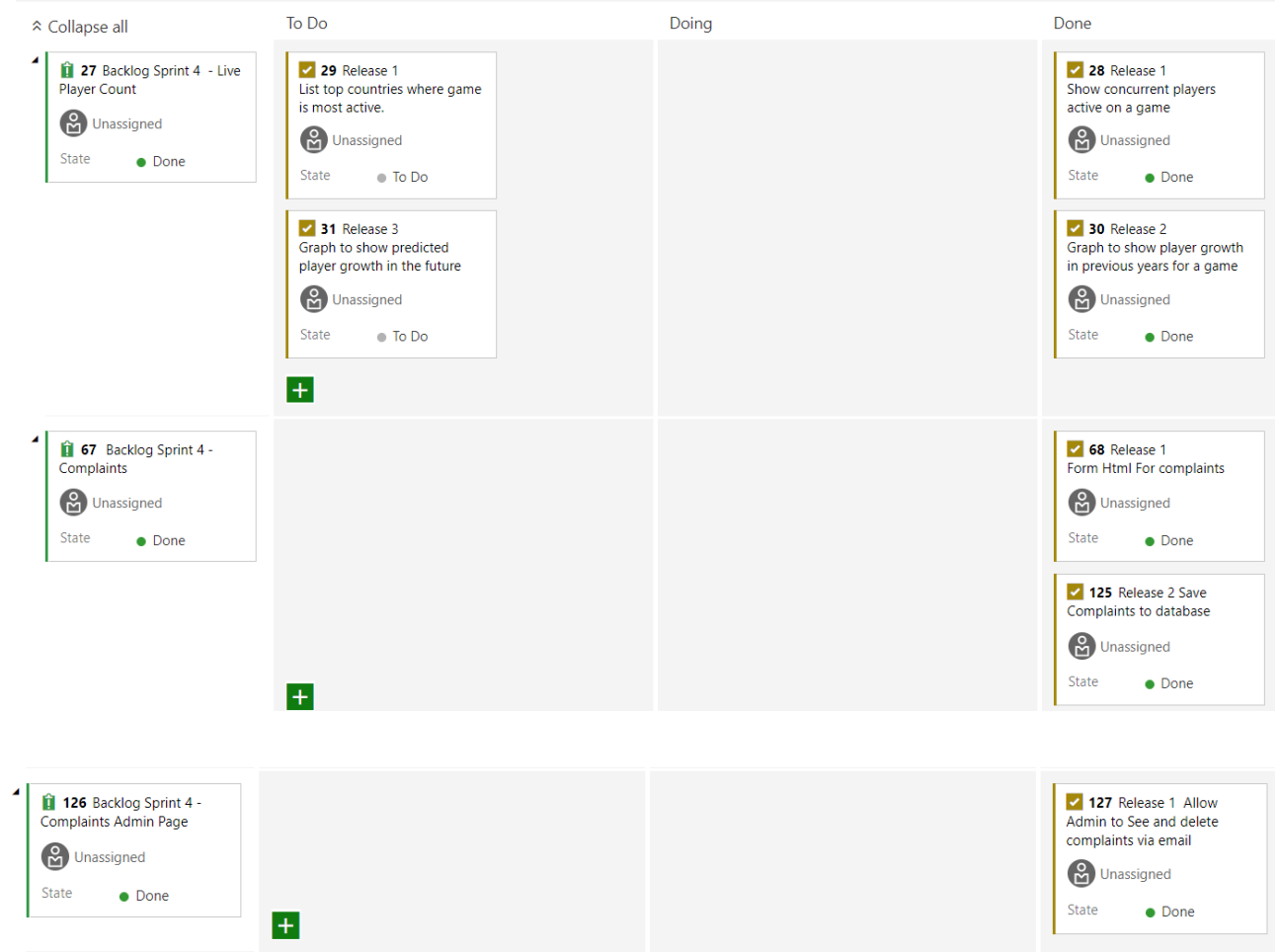
Sprint 2


Collapse all
To Do
Doing
Done
46 Backlog Sprint 2 - Price Comparison
Unassigned
State
Done
47 Release 1
Get Price of that specific game from different retailers
Unassigned
State
Done
48 Release 2
Show the two best Price from 2 different retailers
Unassigned
State
Done
49 Backlog Sprint 2 - Review
Unassigned
State
Done
50 Release 1
Show the two best Price from 2 different retailers
Unassigned
State
Done
51 Release 2
Show the two best Price from 2 different retailers
Unassigned
State
Done
52 Backlog Sprint 2 - Ratings + price Comparison
Unassigned
State
Done
53 Release 3
Show the two best prices and ratings for the games
Unassigned
State
Done
54 Backlog Sprint 2 - Click the link to the platform you want to see the game on
Unassigned
State
Done
55 Release 1
Click the link to the platform you want to see the game on
Unassigned
State
Done

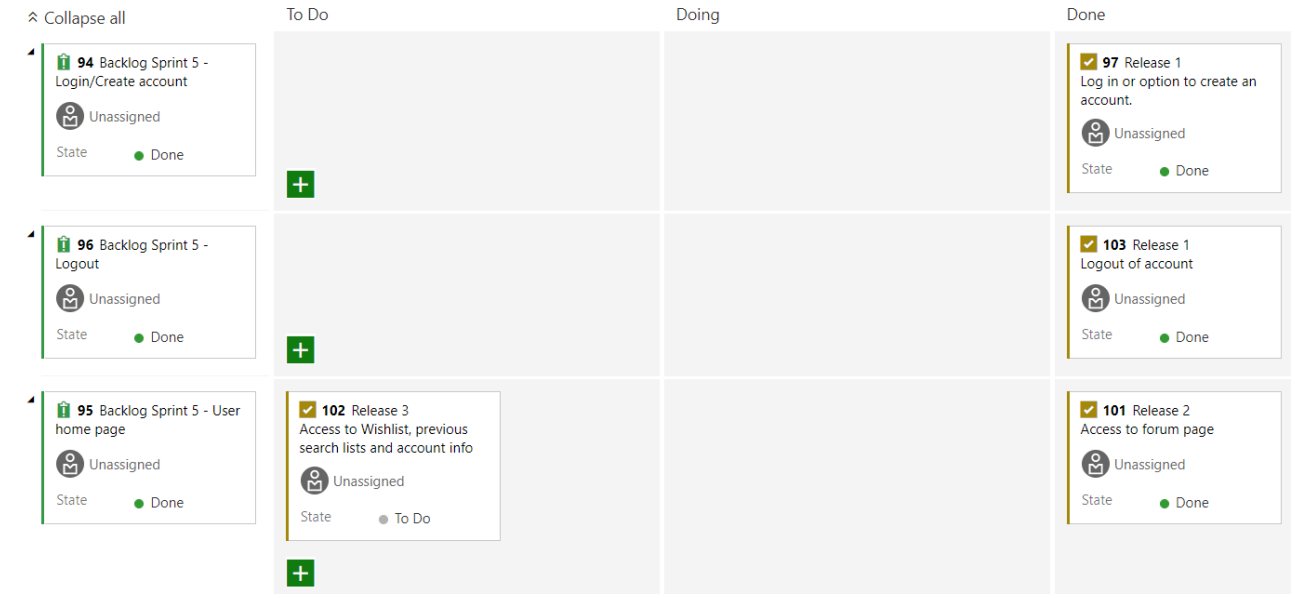
Sprint 3



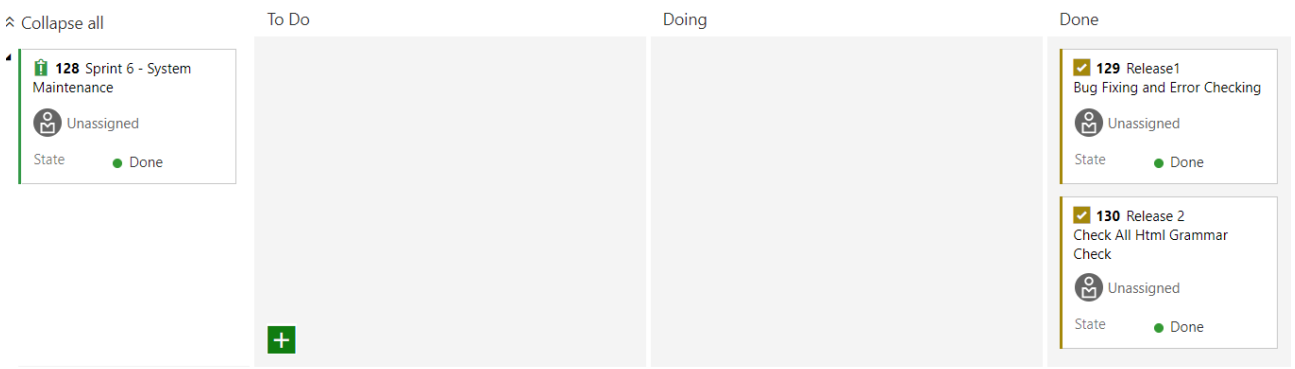
Sprint 4



Sprint 5

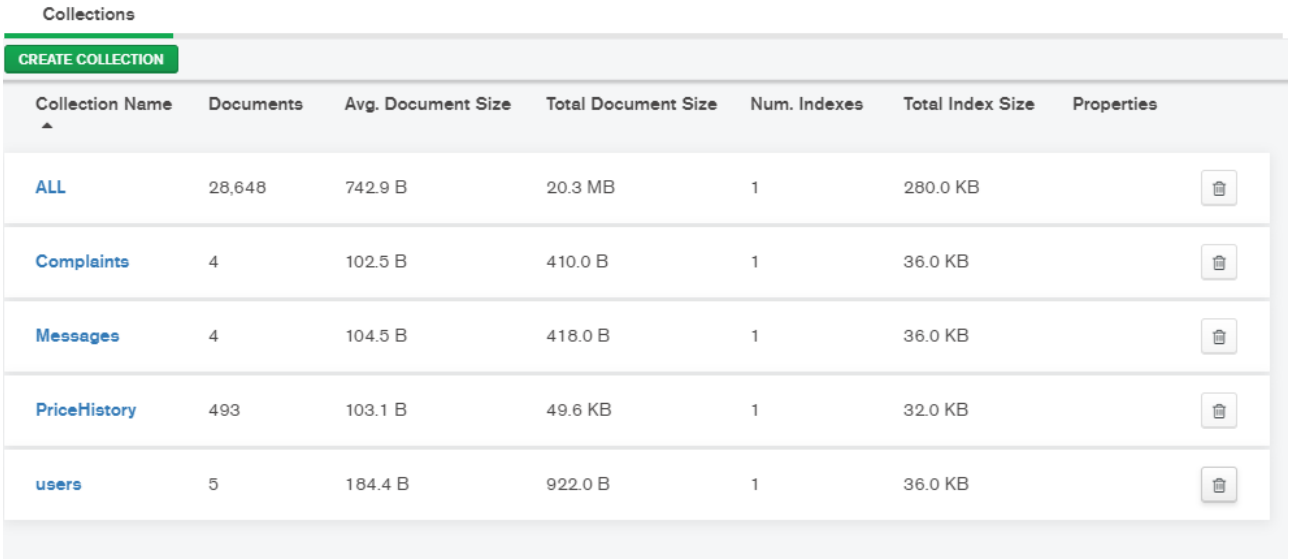


Sprint 6



Appendix 3: DataBase Schema (MongoDB)

This is where we can manage all of our collections in the database and get access to them and perform CRUD functions on the database.



(Collection 1) - ALL

(Collection 2) - Complaints


```
{
    "_id": {
        "$oid": "6080340b20983d11007515d4"
    },
    "Email": "AHHH@gmail.com",
    "Subject": "afoewina",
    "Message": "oewfinoenweonweo"
}
```


(Collection 3) - Messages


```
{
    "_id": {
        "$oid": "60819de9ebb78e2d9e3294e8"
    },
    "username": "admin",
    "message": "This is a test",
    "TimeDate": "22/3/2021 (17:1:45) "
}
```

(Collection 4) - Price history


```
{
    "_id": {
        "$oid": "6076f6393db747313cf19be3"
    },
    "Date": "2019-06-13",
    "Initialprice": "9.99",
    "Finalprice": "9.99",
    "Discount": "0"
}
```


(Collection 5) - Users


```
{
    "_id": {
        "$oid": "60819da1ebb78e2d9e3294e7"
    },
    "firstname": "Admin",
    "lastname": "Admin",
    "email": "admin@admin.co.uk",
    "username": "admin",
    "password": "$2b$10$sY47elbYOauiQoWP8f2VxejhHQqGx.FHIGuLjSizTU.EPgCfUIk9m"
}
```

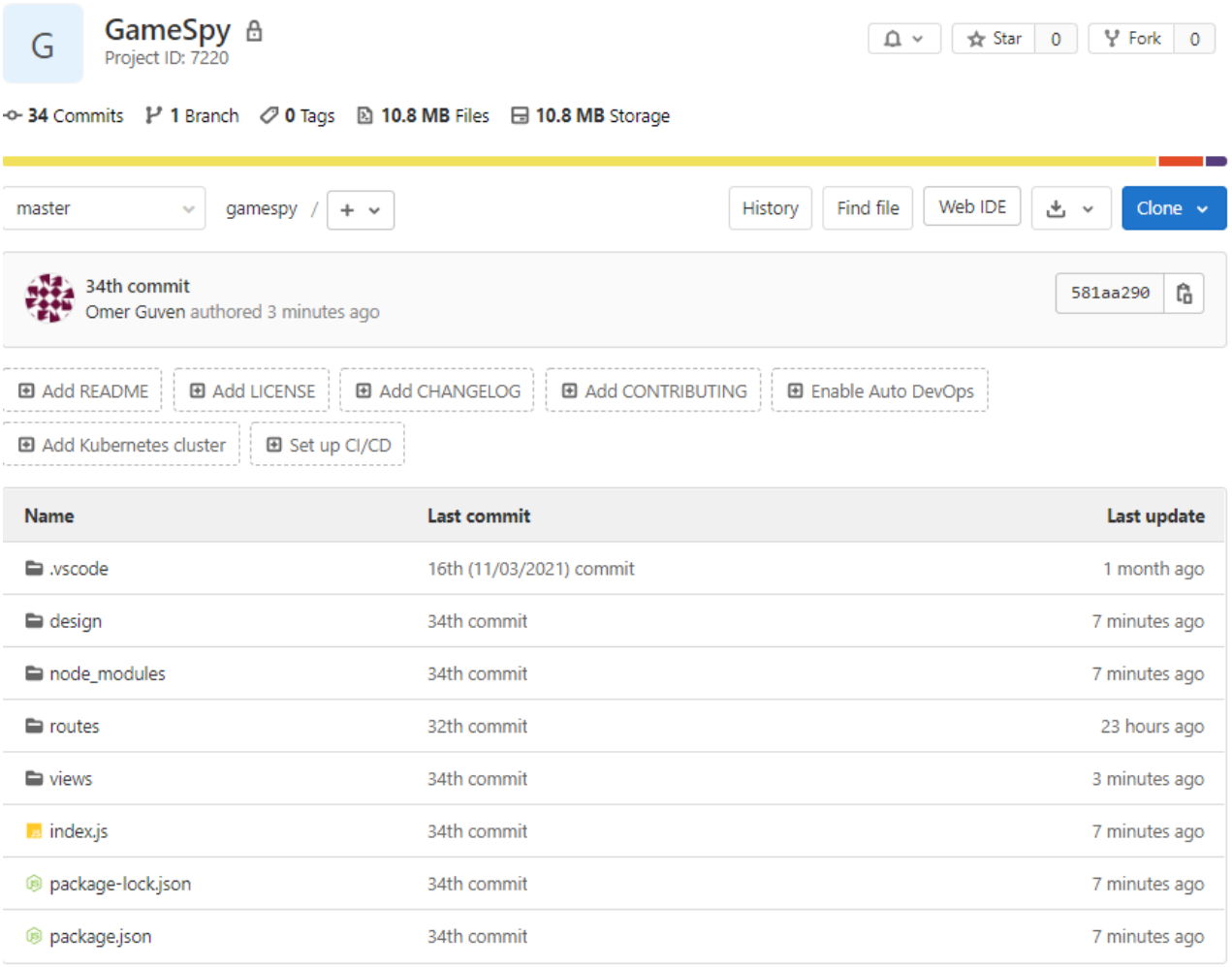
Appendix 4: Agile Sprint (Done via Excel)

This is our agile sprint and how we kept track of the work we were doing and needed to do.

TASK ID	TASK NAME	TASK OWNER	AMOUNT OF WORK IN HOURS		STORY POINTS	START DATE	DUE DATE	STATUS	PCT OF TASK COMPLETE
			ESTIMATE	COMPLETED					
	SPRINT #0		**40**	**55**	**10**	11/01/2021	16/04/2021	Completed	100%
1	Database Setup & Management	Amjad, Omer, Hashim, Haider	40	55		11/01/2021	16/04/2021	Completed	100%
	SPRINT #1		**40**	**50**	**8**	11/01/2021	08/02/2021	Completed	100%
1	Search and filters	Amjad, Omer, Hashim	10	25		11/01/2021	08/02/2021	Completed	90%
2	View games	Haider, Omer, Mahmoud	10	10		11/01/2021	15/01/2021	Completed	100%
3	Select a game	Omer, Hashim, Mahmoud	10	10		18/01/2021	28/01/2021	Completed	100%
4	Click the link	Omer, Hashim, Jensine	10	5		29/01/2021	05/02/2021	Completed	100%
	SPRINT #2		**20**	**14**	**9**	09/02/2021	23/02/2021	Completed	100%
5	Price comparison	Haider, Jensien	10	8		09/02/2021	18/02/2021	Completed	100%
6	Rating	Hashim, Omer, Jensine	10	6		19/02/2021	24/02/2021	Completed	100%
	SPRINT #3		**12**	**25**	**6**	23/02/2021	12/03/2021	In Progress	50%
7	Prediction of prices	Jensine	10	17		23/02/2021	03/03/2021	In Progress	50%
8	Price history	Amjad, Jensine	2	8		04/03/2021	12/03/2021	In Progress	50%
	SPRINT #4		**8**	**12**	**6**	15/03/2021	26/03/2021	Completed	100%
9	Live player count	Haider, Mahmoud	4	2		15/03/2021	18/03/2021	Completed	100%
10	Complaints	Omer	2	4		19/03/2021	23/03/2021	Completed	100%
11	Complaints Admin Page	Jensine	2	5		24/03/2021	26/03/2021	Completed	100%
	SPRINT #5		**12**	**10**	**8**	29/03/2021	09/04/2021	Completed	100%
12	Login/ create account	Omer, Hashim	5	4		29/03/2021	05/04/2021	Completed	100%
13	Home page	Omer, Hashim, Mahmoud	7	6		06/04/2021	09/04/2021	Completed	100%
	SPRINT #6		**20**	**15**	**8**	12/04/2021	16/04/2021	Completed	100%
14	System Maintenance	Omer, Hashim, Mahmoud, Haider, Jensine	20	15		12/04/2021	16/04/2021	Completed	100%
	SPRINT #7 (Report)		**96**	**120**	**9**	11/01/2021	22/04/2021	Completed	100%
15	Introduction	Haider	1	2		11/01/2021	13/01/2021	Completed	100%
16	Changes to implementation process	Omer, Amjad	3	3		11/01/2021	13/01/2021	Completed	100%
17	Analysis	Omer, Hashim, Mahmoud, Haider, Jensine	3	4		11/01/2021	13/01/2021	Completed	100%
18	Iteration/ development	Amjad	55	70		14/01/2021	14/04/2021	Completed	100%
19	Inhouse testing	Omer, Hashim, Mahmoud, Haider, Jensine, Amjad	7	9		15/01/2021	16/04/2021	Completed	100%
20	User Testing	Amjad, Mahmoud, Haider	7	10		16/04/2021	19/04/2021	Completed	100%
21	Evaluation	Amjad, Jensine	15	18		20/04/2021	22/04/2021	Completed	100%
22	Apendix	Amjad	3	3		20/04/2021	22/04/2021	Completed	100%
23	Bibliography	Amjad, Omer, Hashim, Mahmoud, Haider, Jensine	2	1		11/01/2021	22/04/2021	Completed	100%

Appendix 5: Gitlab With All Commits

This is evidence of the work we have committed weekly to gitlab



22 Apr, 2021 3 commits

34th commit
Omer Guven authored 3 minutes ago — 581aa290

34th commit
Omer Guven authored 7 minutes ago — 5e631918

33rd commit
Omer Guven authored 1 hour ago — 4bde50b0

21 Apr, 2021 3 commits

32th commit
Omer Guven authored 23 hours ago — e853c895

32th commit
Omer Guven authored 23 hours ago — d0cead7b

31th commit
Omer Guven authored 23 hours ago — 28f360d3

20 Apr, 2021 1 commit

30th commit
Omer Guven authored 1 day ago — 994da207

19 Apr, 2021 1 commit

29th (19/04/2021) commit
Omer Guven authored 2 days ago — f75fbeb7

15 Apr, 2021 1 commit

28th commit
Omer Guven authored 1 week ago — 166d25de

12 Apr, 2021 1 commit

27th (12/04/2021) commit
Omer Guven authored 1 week ago — 4a4bf2cb

11 Apr, 2021 1 commit

26th (11/04/2021) commit
Omer Guven authored 1 week ago — f1975f37

25th (07/04/2021) commit
Omer Guven authored 2 weeks ago — 8904dd1d

01 Apr, 2021 1 commit

24th (01/04/2021) commit
Omer Guven authored 2 weeks ago — 052a8273

31 Mar, 2021 1 commit

23 (31/03/2021) commit
Omer Guven authored 3 weeks ago — be022d59

29 Mar, 2021 1 commit

22th (29/03/2021) commit
Omer Guven authored 3 weeks ago — 4f3051eb

25 Mar, 2021 1 commit

21 (25/03/2021) commit
Omer Guven authored 4 weeks ago — aea61588

24 Mar, 2021 1 commit

20th (24/03/2021) commit
Omer Guven authored 4 weeks ago — d0e27991

21 Mar, 2021 1 commit

19th (21/03/2021) Commit
Omer Guven authored 1 month ago — 9f3b018b

17 Mar, 2021 1 commit

18th (17/03/2021) commit
Omer Guven authored 1 month ago — d2103e45

14 Mar, 2021 1 commit

17th (14/03/2021) commit
Omer Guven authored 1 month ago — 1efded12

11 Mar, 2021 1 commit

16th (11/03/2021) commit
Omer Guven authored 1 month ago — 14966ff9

07 Mar, 2021 2 commits

15th (07/03/2021) commit
Omer Guven authored 1 month ago
21983966

14th (07/03/2021) commit
Omer Guven authored 1 month ago
fc67eae7

04 Mar, 2021 1 commit

13th (04/03/2021) commit
Omer Guven authored 1 month ago
005435ae

01 Mar, 2021 1 commit

12th commit
Omer Guven authored 1 month ago
02b8c3cc

28 Feb, 2021 1 commit

11th 28/02/2021 commit
Omer Guven authored 1 month ago
8fc9b0a1

26 Feb, 2021 4 commits

10th (26/02/2021) commit
Omer Guven authored 1 month ago
686b4e3a

9th (26/02/2021) commit
Omer Guven authored 1 month ago
8d217968

9th (26/02/2021) commit
Omer Guven authored 1 month ago
abab2bd6

8th (26/02/2021) commit
Omer Guven authored 1 month ago
bed9dc79

24 Feb, 2021 1 commit

7th commit (24/02/2021)
Omer Guven authored 1 month ago
82eedd8c

21 Feb, 2021 1 commit

sixth commit
Omer Guven authored 1 month ago
f9a4a99a

18 Feb, 2021 1 commit

fifth (18/02/2021) commit
Omer Guven authored 2 months ago
84039162

17 Feb, 2021 2 commits

fourth commit
Omer Guven authored 2 months ago
f5d2d93e

second commit
Omer Guven authored 2 months ago
6306d205

15 Feb, 2021 1 commit

third commit
Omer Guven authored 2 months ago
06309901

24 Feb, 2021 1 commit

Updated 24/02/2021 commit
Omer Guven authored 1 month ago
552feec0

15 Feb, 2021 1 commit

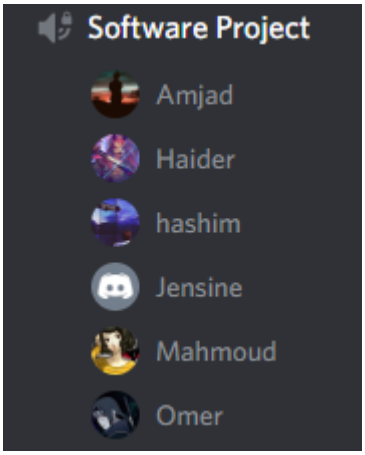
second commit
Omer Guven authored 2 months ago
2895cbcb

11 Feb, 2021 1 commit

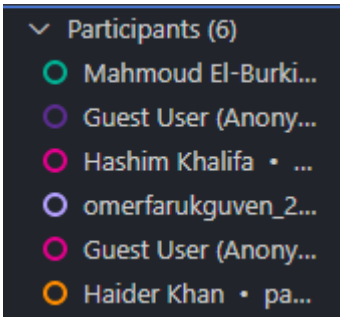
Initial commit
Omer Guven authored 2 months ago
0e5ff5ad

Appendix 6: Evidence Of Group Work Taking Place

Our main way of working was discord and using VSCode Live Share



https://gyazo.com/20c9d644679b2af7e7cd9279adf79192



Links to our website:

http://www.doc.gold.ac.uk/usr/900/

http://www.doc.gold.ac.uk/usr/900/populargames

http://www.doc.gold.ac.uk/usr/900/login

Other pages redirect to login page