Video Game Recommendations using Machine Learning.

Project Engineering

Year 4

Darragh Fahey

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Galway-Mayo Institute of Technology

2020/2021

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

Darragh Fahey

_____

Acknowledgements

I would like to acknowledge my parents Michael and Pauline, for their continued support. My sister Rebecca for her constant countdowns of how long I have left, and my brother Niall for making me laugh.

I would also like to thank all my friends who have helped make an isolated year much less lonely.

Table of Contents

1 Summary

For my project I wanted to create a website that would help people chose what game to buy and play, as an avid gamer myself I often find myself continually browsing the PlayStation store trying to pick a new game in between major releases.

For this project I had 9 months to design and deliver my idea. I wanted to create a recommender that could be modular, meaning that it could work with things other that games. I decided to create a Machine Learning algorithm that would base its predicted ratings off other user's game ratings. I decided to use the singular value decomposition algorithm (SVD) as it is a popular algorithm in recommendation systems. The information would be stored in a PostgreSQL relation database, which could be read from the website I would design. Each user would be able to search for a video game and give it a numerical rating from 1-10 which would then be passed into the SVD algorithm upon its next iteration. To get the initial data set I build a web scraper to get video game data from Metacritic.

For this Project I used React Native for the website User Interface, I used PostgreSQL for the database. For the Machine learning algorithm and web scraper I used python, using libraries such as scikit-surprise, beautifulsoup4 and psycopg2.

In the end I was able to get all the systems working together, but I did run into some problems along the way. My algorithm predicts ratings for games the user hasn't already rated, and this can be displayed on the website for the user, but if a new user registers, they will not see any predicted ratings, as they need to rate at least one game and wait for the algorithm to be run before the database will have updated with ratings for them.

In conclusion I believe my project has some very interesting features to it, and I believe that I succeeded in making a system that could be used for other media, an initial dataset would be all anyone needs to change it.

2

Video Game Recommendations using Machine Learning Darragh Fahey G0035047

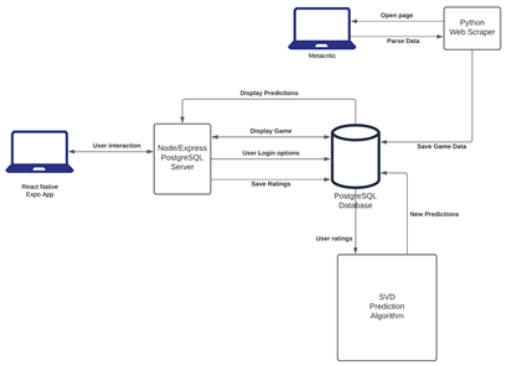
Introduction

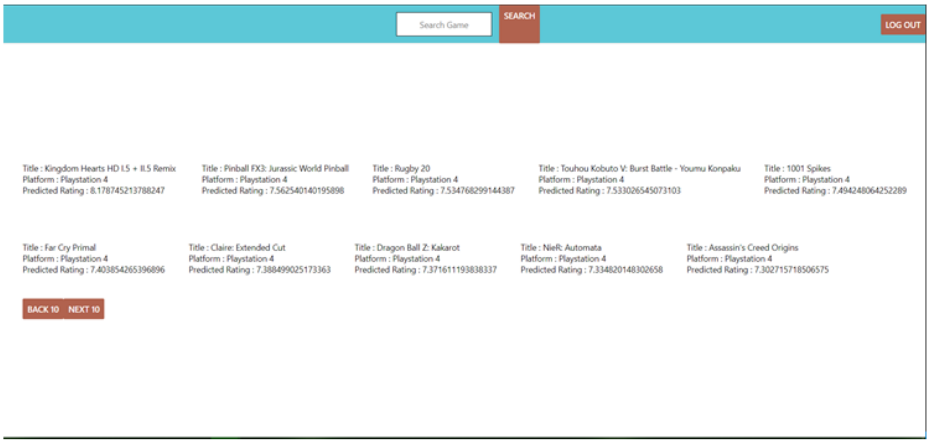
For my project I wanted to create a website that would help people chose what game to buy and play, as an avid gamer myself I often find myself continually browsing the PlayStation store trying to pick a new game in between major releases.

Technologies Used

The main technologies I used for this project were

- React Native
- Express/Node js
- PostgreSQL
- Python
- Surprise scikit
- Beautiful Soup
- Fetch API





Web scraper

I created a web scraper that opened a connection to Metacritic using the urllib python library, once I had the connection opened, I used beautiful soup to parse through the html file and get the information I was looking for.

React Native App

Above you can see an example of 10 predictions for a test user.
I created a react native app and used fetch API calls to get the data you see from my PostgreSQL database. When not logged in the user can see the top-rated games from Metacritic.

SVD Algorithm

The predictions were made using the singular value decomposition model, and collaborative filtering. If a user A has made similar ratings to user B, then user A will get higher ratings for items B has recommended.

3 Introduction

During my placement and in the fourms on the internet I am subscribed to that discuss programming, one subject continually gets brought up. Machine Learning. Due to its increasing usage in the workplace and its interesting nature I knew going into the year that I would want to create a project based around machine learning. I didn't know what topic to pick for a while, until one day I was looking for a new game to play. I found it difficult and thought there had to be a better way to do this. I thought about how I usually pick out games to play and the most obvious answer was recommendations.

I started to investigate recommendation systems using machine learning and found that there is already a lot of systems in place to use it the most well know of which would be for big companies like amazon recommending new products based off other users, or Netflix's tv and movie recommendations. I knew then I wanted to make one that catered to games.

I had nine months to create a dataset, train an algorithm, and create a website. I started with by doing research on all the different topics that I would need during my project. After looking at the options for recommenders I decided to go with a collaborative filtering approach, where based off the games I have already rated, I could get a predicted rating for games I had not yet rated. I would be using the singular value decomposition algorithm from the surprise scikit to make these predictions.

I decided that I would use PostgreSQL for the database systems as it was a relational database that I had used during my workplace, not to mention that it has great scalability and would be able to handle an ever-growing data set and predicted ratings.

For the frontend web development, I decided to use React Native as it is an open-source cross platform framework. It allows you to create one code base that works well on android, iOS, and web. This meant that I wouldn't have to worry about how to port it to those other devices and just work on creating one site.

Once my research was complete, I started gathering the data that I would need for the project. I found it difficult to find an already made data set for video games that had all the information I

needed. There were some but they were either very old and thus not relevant or they only had a handful of games. Because of this I decided to create a web scraper that would get all the information I would need from Metacritic. I used the beautiful soup library to make this scraper, and it allowed me to tailor the exact type of data I would get for each game.

4 Project Architecture

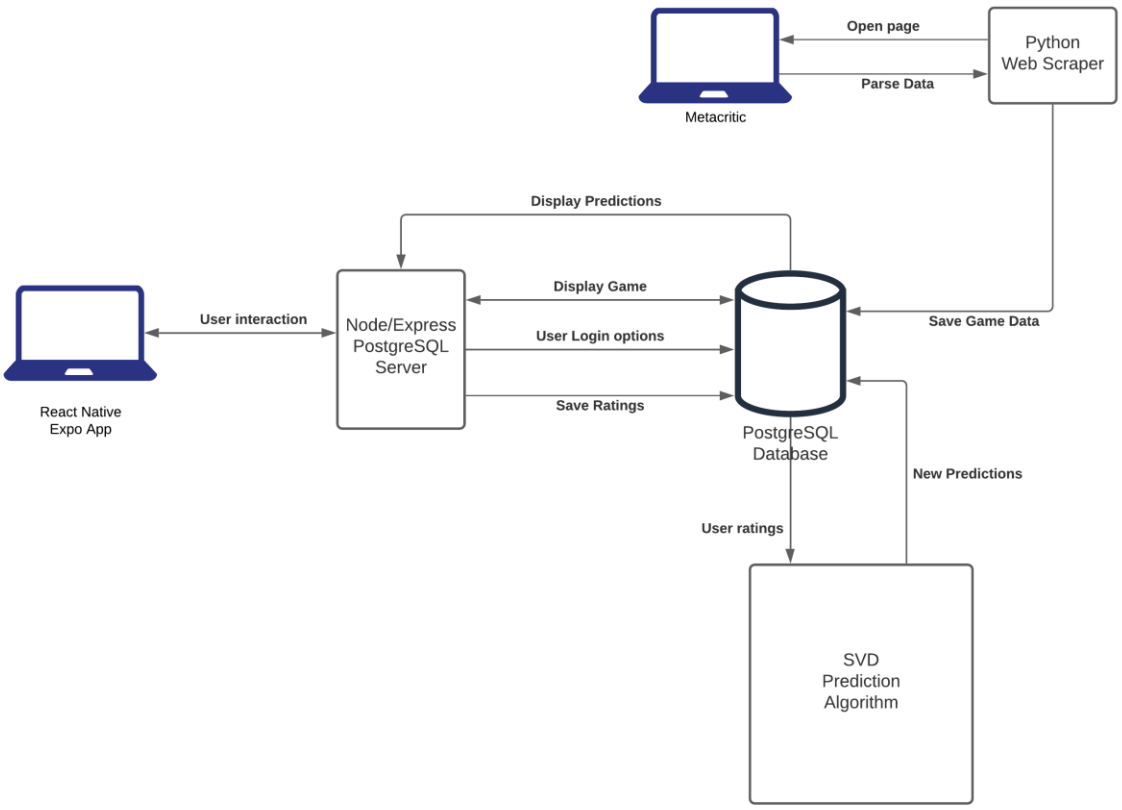


Figure 4-1 Architecture Diagram

5 Project Plan

6 Web Scraper

In this section I will be describing the method of how I built the web scraper I used to gather my data for games and the survey I created to get some user data.

6.1 Beautiful Soup

For the web scraper I decided to use python as it is easy to understand, it has a wide array of open-source libraries that give it a versatility that is hard to find in other languages. I investigated web scrapers others had built previously but I found that most were not focused on video games themselves or they didn't gather the information for all the games I would need. So, I decided to make my own scraper. The package I decided on using for the web scraper was the Beautiful Soup library [1]. It allows the code to pull data from HTML and XML files which is then easier to parse. Along with Beautiful Soup I would need URLlib package, which will handle URL requests.

6.2 Code

Once I had decided on what package to use, it was time to start writing the code for the web scraper. The scraper is 6 files that handle different aspects of the Metacritic website. There is the ConsoleNavigator.py, LetterNavigator.py, TitleNavigator.py, ScrapeGameData.py, radomSleep.py, FileHandler.py. I decided to lay them out like this in case there was a change in the Metacritic website you would know at what point in the line-up the change needs to go into effect. For example, if you there was a change in how the titles are displayed on the website, you would make the change in the title navigator file.

The ConsoleNavigator.py file just sends in which console you want to scrape and passes the base URL "https://www.metacritic.com" along with the sub-URL "/browse/games/title/ps4", in the PlayStation 4's case, to LetterNavigator.py where we can make a request to open the page.

I will explain how the request work but as they are the similar for each file I will work through the LetterNavigator.py file below. We need to import Beautiful Soup and URLlib at the top of the file. Now we will open each game ordered by title alphabetically.

```
for i in range(len(pageLetters)):
    currentLetterUrl = url + pageLetters[i] + "?view=detailed"
    req = Request(currentLetterUrl , headers={'User-Agent': 'Mozilla/5.0'})

    webpage = urlopen(req).read()
    page_soup = soup(webpage, "html.parser")
```

In a for loop in the range of the length of an array called pageLetters which is [“”, “/a”, “/b”,…,”/z”] which we add onto the current URL to open each page. We take the letter at the index of the for loop, add it and detailed view type of Metacritic tiles.

After we have the URL created we make a request using the Request function of the URLlib package [2]. We send a request along with a header which is used to spoof Metacritic into thinking that this is an actual URL request. If this is not used and we tried to open the webpage we would get a 403 forbidden error.

Once we have the request made we can try to open the page using the urlopen function which is a part of the Request function. Where we pass in the request and it is read. We then use Beautiful Soup to make the soup [3]. The soup is what we will use to parse through the data from the website. And we pass the webpage and HTML.parser as it is a website. This is similar for each page that we want to open with the web scraper the only difference being the URL that is passed into the request.

```
pageNumbers = len(page_soup.findAll("li", "page"))
print(pageLetters[i])
```

Now we have the soup made we want to find specific elements of the page, these are consistent across each letter page, but different on the title pages. On the letter page we on how many pages each letter has, each page has up to 100 games, and letters with less than 100 games they have one page. Using page_soup.findAll() we can get a number on how many pages there is [4].

```
if (pageNumbers == 0):
    currentPageUrl = currentLetterUrl

    titleNavigator(currentPageUrl, platform)
    randSleep_obj.fireRandomSleep()

else :
    for j in range(pageNumbers):
        currentPageUrl = currentLetterUrl + "&page=" + str(j)

        titleNavigator(currentPageUrl, platform)
        randSleep_obj.fireRandomSleep()
```

Once we have the number of pages we can loop through them and open the title list pages for each letter. If there are less than 100 games on the page we can just call the titleNavigator function and pass it the currentPageURL and platform. Otherwise, we must loop through the numbers and open pass the new currentPageURL which will add the index to the URL. After the page is closed we call the randSleep_obj function which will sleep for a random number of seconds. This stops Metacritic from closing the connection as the pages are opening at sudo random periods.

The titleNavigator.py file opens pages and parses through them in a similar way just looking for different elements in the soup, which in turn opens a new page for each game using the ScrapeGameData.py file. I used the FileHandler.py file to save the data to a .csv file as I hadn't set up the PostgreSQL table at this point of the development process.

```
def fileWriterMetacriticGameDataCSV(title, platform, alsoOn, released, developer,
                                    publisher, genre, metascore, userScore, ageRating):

    fileName = platform + "Data.csv"
    fieldNames = ['Title', 'Platform', 'Also On', 'Released', 'Developer',
                  'Publisher', 'Genre(s)', 'Metascore', 'User Score', 'Age Rating']
    with open(fileName, 'a', newline = '') as file:

        writer = csv.DictWriter(file, fieldnames=fieldNames)
        writer.writerow({'Title': title, 'Platform': platform, 'Also On': alsoOn, 'Released': released,
                         'Developer': developer,'Publisher': publisher, 'Genre(s)': genre,
                         'Metascore': metascore, 'User Score': userScore, 'Age Rating':ageRating})
```

I passed in the info I had scraped from the webpage and using python's csv library [5] I was able to create a new file if it doesn't already exist and on a new row for each game, save the data .

The randomSleep.py file was a file I found while looking at other open-source web scrapers before building mine [6]. I thought it was an interesting idea and took the file to implement

mine. It randomly sleeps for a number between a range you set in an array, when a counter hits a predefined threshold. The counter is then reset, and it sleeps for a random time.

6.3 Survey

You can't scrape the web for user data, and I wanted to use some real people when creating the initial ratings as I felt this would give me better ratings that if I was to solely create them from scratch. So, I created a survey and distributed it to my friends on social media. I also shared it on reddit and was invited to join a survey exchange Facebook group. I was able to gather a total of 145 responses because of this. I didn't think this would be enough data but instead of hoping more people would answer my survey I randomly generated ratings and profiles for people.

I found out a lot of information during the survey, but the most important along with the ratings were the amount of people who play which console. The most used console was PC followed by PlayStation 4. Because the web scraper takes quite a long time to run, I decided to focus on just PlayStation 4 games as the PC has so many that it would take the web scraper too long to run for this project.

7 Postgres

When choosing what type of database, I would use in my project I had to decide if I was going to use Relation or Non-Relational databases [7]. I decided quickly upon my research to go for a relational database as they use a structured query language, and they scale well. Data being stored in different tables meant it would be easy to look for specific data when I needed it and would be able to get that information using the primary and foreign keys of the tables.

I wanted to use PostgreSQL as I was familiar with it having used it during my placement. But it also is a great database because it is community driven, has high security and has great scalability. [8]

7.1 Creating the Database

I used HeidiSQL as my administration tool as we had been using it for another module. Using this program, I was able to set up my PostgreSQL (TCP/IP) database and create the tables that I would need. I created a table for user_game_recomendations, user_game_predictions, playstation_4_data, and user_login_details. The first two tables are where the users' recommendations and predicted scores will be kept. All the tables will be used in the backend server to connect and get the information needed to be displayed or saved by the website.

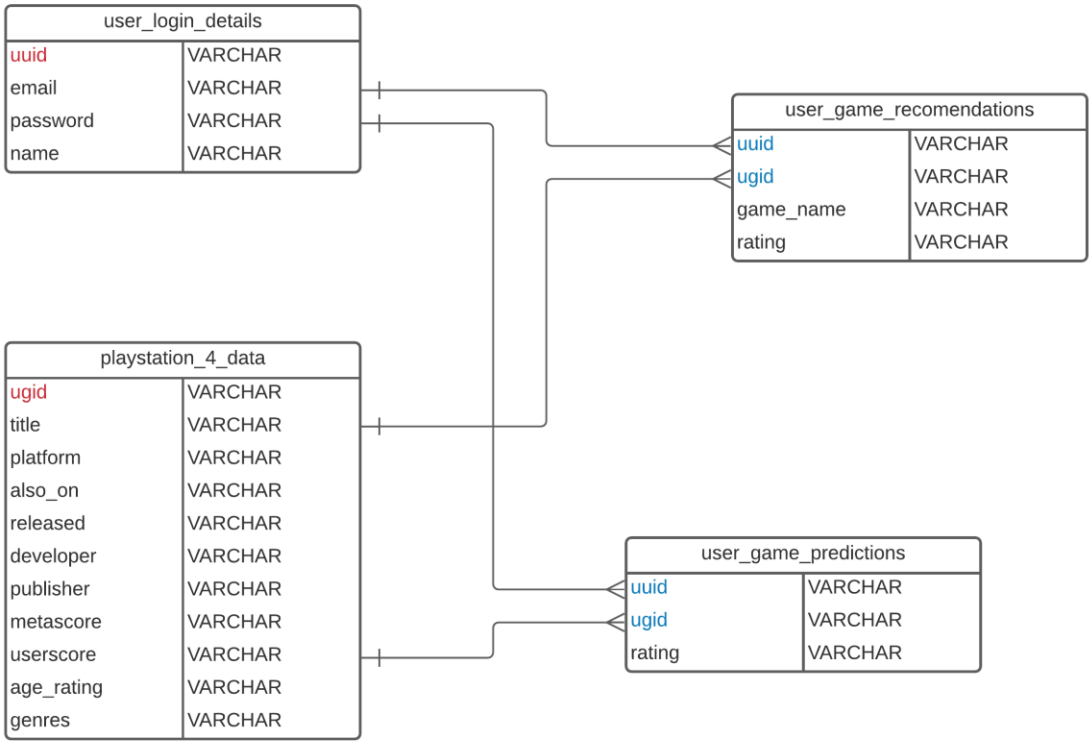
I needed unique identifiers for the games and users, so I gave them the variable names unique user id (uuid) and unique game id (ugid). As the data was originally in .csv files and I was viewing them in excel I was able to find a formula online which creates a unique identifier [9].

" =LOWER (CONCATENATE (DEC2HEX (RANDBETWEEN (0,POWER(16,8)),8), "-" , DEC2HEX (RANDBETWEEN (0, POWER(16,4)),4) ,"-", "4", DEC2HEX (RANDBETWEEN (0,POWER(16,3)),3), "-", DEC2HEX (RANDBETWEEN (8,11)), DEC2HEX (RANDBETWEEN (0,POWER(16,3)),3), "-", DEC2HEX (RANDBETWEEN (0,POWER(16,8)),8), DEC2HEX (RANDBETWEEN (0,POWER(16,4)),4)))"

This formula creates a unique id like '997bca61-47e9-4c0b-8f59-fd0694ad50b4'. The formula is a bit hard to read but it creates a string that is all lower case with 5 segments concatenated

together. Each segment is random number between 0 and another number gotten by either 16 or 8 to the power of another number, then converted from decimal to hexadecimal. Using this I was able to create unique identifiers now, but also be able to create them in the future.

7.2 Relational Diagrams



8 React Native

The front-end website was designed using React native and Expo. Using these two frameworks allowed me to create a website that would work on the web, android, and iOS without having to develop separate codebases for them. To navigate from screen to screen I used stack navigation, and I combined this with fetch API calls to get the data from the back-end server.

I also needed users to be able to log in and out so they could get their predicted ratings. I did this using AuthContext.

8.1 Stack Navigation

Stack navigation is used to provide a way for the app to transition between screens [10]. It also allows for familiar transitions on android and iOS . I had two navigators in mind when I was designing the website, what a user would see why they are not logged in, and how that would be different from how someone who was logged into the site would see it. As such I created two navigation containers wrapped in an if else check based on the userToken from the AuthContext.

```
if (loginState.isLoading) {···
}
return (
  <AuthContext.Provider value= {authContext}>
    { loginState.userToken == null ? (
      <NavigationContainer>
        <Stack.Navigator ···
      </NavigationContainer>

      ) :

      <NavigationContainer>
        <Stack.Navigator >···
      </NavigationContainer>
    }

  </AuthContext.Provider>
);
;
```

If the userToken is null, then it will show the user the NoLogin Screens and if it is anything but null, which in this case would be the uuid then it will show them the Login screens. Which you can see below.

```
  <NavigationContainer>
    <Stack.Navigator >
      <Stack.Screen
        name="Home"
        component={HomePageLogin}
        options={{
          headerTintColor: 'white',
          headerStyle: { backgroundColor: '#5CC8D7FF' },
        }}
      />
      <Stack.Screen
        name="searchPage"
        component={SearchPageLogin}
        options={{
          headerTintColor: 'white',
          headerStyle: { backgroundColor: '#5CC8D7FF' },
        }}
      />
    </Stack.Navigator>
  </NavigationContainer>
}
```

You can see that in the navigator I have two screens the HomePageLogin, which displays the predicted ratings for the user, whereas the NoLogin page shows the games with the highest metascore. You can also see the SeachPageLogin which lets the user rate the game where the NoLogin Page does not.

With the navigators created, I created a folder called pages where I would store all the separate JavaScript files for each new page.

8.2 Fetch API

To get the information from the back-end server I used Fetch requests. I used fetch due to its power and flexibility. [11] Using fetch you pass in the path of the resource you want to get, the fetch response returns a promise, which can be resolved in the body of the response.

```
function getApiData(input) {
  fetch("http://127.0.0.1:5000/" + input )
    .then(function (response) {
      if (response.status !== 200) {
        console.log('Looks like there was a problem. Status Code: ' +
          response.status);
        return;
      }
      response.json().then(function (data) {
        console.log(data.length);
        if(data.length !== 0){
          setName(data['0']['title']);
          setPlatform(data['0']['platform']);
          setAlsoOn(data['0']['also_on'])
          setReleaseDate(data['0']['released']);
          setDeveloper(data['0']['developer']);
          setPublisher(data['0']['publisher']);
          setGenre(data['0']['genres']);
          setRating(data['0']['metascore']);
          setAgeRating(data['0']['age_rating']);
        }
        else {
          alert('No game');
        }
      });
  }
    )
    .catch(function (err) {
      console.log('Fetch Error :-S', err);
    });
}
```

Here with this fetch, you can see its going to the backend server port 5000 with an input at the end. The input in this case is the name of the game being looked for. If the response from the fetch is good then it will receive a json object back which is used to set the information for the

game. If the json object is empty then an alert box will appear showing that no game with that name can be found.

8.3 AuthContext

For the user to be able to login I created a global variable that the React app can use to know if it should be showing the private Login pages or the public NoLogin pages as discussed in the stack navigation earlier. [12]

First I created a context.js file which I used to create the context which I named AuthContext. Then I imported the AuthContext into the App.js file.

```
const initialLoginState = {
  isLoading: true,
  userName: null,
  userToken: null,
};

const loginReducer = (prevState, action) => {
  switch (action.type) {
    case 'RETRIEVE_TOKEN' :···
    case 'LOGIN' :···
    case 'LOGOUT' :···
    case 'REGISTER' :
    return{
      ...prevState,
      userName: action.id,
      userToken: action.token,
      isLoading: false,
    };
  }
}

const [loginState, dispatch] = React.useReducer(loginReducer, initialLoginState)

const authContext = React.useMemo(() => ({
  signIn: async(userName, password) => {···
  },
  signOut: async() => {···
  },
  signUp: async(name, email, password) => {···
  },
}))
```

I set an initialLoginState data set where isLoading is set to true, the userToken and username are both set to false. If this is thee first time someone views the website then they will not be able to view the private pages.

Next I created a loginReducer which will remember if a user was logged in or out when they last viewed the site. It also handles the new action and updates the username, userToken and isLoading variable from the loginState if a user logs in, out or registers.

```
<Text
  style={styles.loginBtn}
  onPress={() => signIn(email, password)}
>LOGIN</Text>
```

When the login button is pressed it calls the signIn function and passes the email and username the user entered to the AuthContext.

```
if(userName == recievedEmail && password == recievedPassword)
{
  try {
    userToken = (data[0]['uuid']);
    AsyncStorage.setItem('userToken', userToken)
    global.userToken = userToken;
  } catch(e) {
    console.log(e);
  }
  dispatch({type: 'LOGIN', id: userName, token: userToken});
}
```

Then using the signIn function it gets the users login details from the server and compares them. If they match the user is logged in and the userToken is set in the asyncStorage. Then you dispatch the correct type and the username and userToken.

When The user is logged out we remove the userToken from asyncStorage and dispatch with just the LOGOUT type.

9 Backend Server

For the backend server I created a node.js project using Express a , CORS and Knex. Using these combined I was able to send GET and POST request to my localhost connected to my PostgreSQL database, allowing me to look for and display the predicted ratings for users.

Knex is a is a SQL query builder for Postgres, MSSQL, MySQL, MariaDB, SQLite3, Oracle, and Amazon Redshift designed to be flexible, portable, and fun to use. [13] I had to create a .env file in my backend server project structure that would contain my Database information, such as the username, host, and password. With this added I was able to open a connection using Knex.

```
const db = knex({
    client: 'pg',
    connection: {
        host: process.env.DATABASE_HOST,
        user: process.env.DATABASE_USERNAME,
        password: process.env.DATABASE_PASSWORD,
        database: process.env.DATABASE,
    },
});
```

This opens a connection to the database if it is available and the information in the .env file is correct.

Once it has an open connection then you can make GET and POST requests using the appropriate query builder functions. [14]

```
app.get('/predictions/:uuid', (req, res) => {
    const uuid = req.params.uuid;
    db.select('*')
        .from('user_game_predictions')
        .where('uuid', '=', uuid)
        .orderBy('rating', 'DESC')
        .then((data) => {
            console.log(data);
            res.json(data);
        })
        .catch((err) => {
            console.log(err);
        });
});
```

Here is a get request where I am getting the current predictions for the user on the website. I pass in the uuid and it selects all the rows from the prediction table where the uuid is the one we sent in the GET request, ordering them in descending order.

```
app.post('/addRating', (req, res) => {
    const { uuid, ugid, gameName, rating} = req.body;
    db('user_game_recomendations')
        .insert({
            uuid: uuid,
            ugid: ugid,
            game_name: gameName,
            rating: rating,
        })
        .then(() => {
            console.log('New rating added');
            return res.json({ msg: 'New rating added' });
        })
        .catch((err) => {
            console.log(err);
        });
});
```

Here I am using a POST request to add a new rating that has been added by the user from the website. We create a request body in the front end and send it to the server then insert the information into the database.

10 Machine Learning

When looking at machine learning algorithms, I looked at other recommenders to see what they were using. There were a couple of phrases that continually came up, content-based filtering and collaborative filtering. [15] Content-based filtering looks at what the elements of the product that the user might like and predict based on that, whereas collaborative filtering looks at how other users rated the item when considering what the current users prediction. I decided to go with a collaborative approach for my project.

I also had to decide what type of algorithm to use, did I want to use supervised or unsupervised learning? Then once I had picked that what specific algorithm would I use, K-nearnest neighbour or dimensionality reduction? I decided to use the singular value decomposition algorithm, as it is used a lot in collaborative filtering and recommenders in general.

10.1 Collaborative filtering

In collaborative filtering you are looking at how to predict a rating for a user either based off user-user filtering or item-item filtering. With user-user filtering you look at a user A who has similar characteristics to user B, in this case then A will be recommended items that B liked.

In item-item filtering if item X is rated and then item Y has similar properties it will recommended to the user. I decided to go with a user-user filtering approach. Both ways have the same mathematical equation.

$$R_{xu} = \left(\sum_{i=o} n\, R_i \right) / n$$

Where Rxu is the rating item x by user u and i=0 to n are the users who have given similar ratings to user u.

Users show different behaviours while rating, some will rate high while others may rate lower than or average. Because of this we will need to average all the ratings the user has provided and subtract that from Ri to normalise the ratings. So, our equation now looks more like this,

$$R_{xu} = \left(\sum_{i=o} n\, R_i Wi\right) / \sum_{i=o} n\, Wi$$

10.2 SVD

Once I knew that I wanted to take a collaborative filtering approach I had to decide on what algorithm to user. I investigated a few different types of algorithms like K-nearest neighbour, dimensionality reduction and classification. I decided to use Singular value decomposition (SVD) for my prediction model.

SVD is a method of decomposing a matrix into three other Matrixes

$$A = USV^T$$

Where A is an m*n matric, U is an m*n orthogonal matrix, S is an n*m diagonal matrix and V is an n*m orthogonal matrix transposed. [16] The diagonal values of the diagonal matrix are known as the singular values of the original matrix A. The columns of the U matrix are the left-singular vectors of A and the columns of V are the right-singular vectors of A.

Using the Surprise scikit you can use SVD to predict ratings using the formula [17]

$$\widehat{r_{ui}} = \mu + b_u + b_i + q_i^T (p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j$$

Where $\widehat{r_{ui}}$ is the prediction, the y_j terms are a new set of item factors that capture implicit ratings. b_u is the bias, p_u are the factors. The same applies for the item i with b_i

```
reader = Reader(line_format='user item rating', sep=',', rating_scale=(1, 10))
data = Dataset.load_from_file(file_path, reader=reader)

algo = SVD()
trainset = data.build_full_trainset()
print('Training Model')
algo.fit(trainset)
print('Done')
```

You create a reader which reads the rows of the file, where the line_format is the titles of file in a comma delimited file, and the scale is from 1-10 as that is the range of our ratings. You load in the data using Dataset module from surprise. [18] We then set the algorithm to SVD and build a full trainset , and feed that into the algorithm.

```
for user_index in range(trainset.n_users):
    # All items this user HAS rated
    user_items = set([j for (j, _) in trainset.ur[user_index]])

    # All items this user has NOT rated
    items_user_has_not_rated = [trainset.to_raw_iid(
        i) for i in trainset.all_items() if i not in user_items]

    # Get the actual user_id from index
    user_id = trainset.to_raw_uid(user_index)

    # Make predictions for all items this user has NOT rated
    predictions_for_items_not_rated_by_user = []
    for iid in items_user_has_not_rated:
        pred = algo.predict(user_id, iid)

        predictions_for_items_not_rated_by_user.append([pred.iid, pred.est])
        fileWriterPredicitonsCSV(user_id, iid, pred.est)
```

We want to create predictions for items that the user has not yet predicted. For each user in the trainset, we get the items they have already rated so they won't predict a new rating. Then we right them into another file for predictions and save that to the database.

11 Ethics

Because my project deals with the acquisition and manipulation if user's data there should be a lot of focus on how to handle the sensitive data and making sure that it is secure. I used

PostgreSQL for the database system because it has good security and I tried to make sure that when gathering information during the survey process that it was all anonymous .

I didn't use anyone's real email addressed during the development, only using fake test emails to create dummy accounts. This was to ensure that if there was a data breach then no one would be harmed during it.

12 Conclusion

After connecting all the various systems and technologies I developed I was able to create a website and server that displays the predicted ratings that I got using ,my collaborative filtering approach. It is a project that I believe is very modular as long as you can get the right datasets for users and items. I was happy with how this project turned out.

I learned a lot during the development cycle, from frontend web development to the intricacies of how recommenders work. I now feel like I have a much better grasp of how to better handle user's data in a secure manner. Overall this was a fun project to do and I feel like it could be one that I continue to develop in my personal time to make it the best it can be.

13 References

[1] Beautiful Soup Documentation [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ Accessed: Feb 04, 2021.

[2] URLlib.request [Online]. Available: https://docs.python.org/3/library/urllib.request.html#module-urllib.request Accessed: Feb 04, 2021.

[3] BS4 making the Soup [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/#making-the-soup Accessed: Feb 04, 2021.

[4] BS4 findAll [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/#calling-a-tag-is-like-calling-find-all Access: Feb 04, 2021.

[5] Python CSV library [Online]. Available: https://docs.python.org/3/library/csv.html Accessed: Feb 05, 2021.

[6] Random Sleep function [Online]. Available: https://github.com/mostafatouny/data-scraper/blob/master/other/randomSleep.py Accessed: Feb 05, 2021.

[7] SQL vs NoSQL [Online]. Available: https://towardsdatascience.com/databases-101-sql-vs-nosql-which-fits-your-data-better-45e744981351#:~:text=Simple%2C%20SQL%2C%20and%20NoSQL%20interact,interact%20with%20non%2Drational%20databases. Accessed: Sep 21, 2020.

[8] Postgres Advantages [Online] Available: https://www.cybertec-postgresql.com/en/postgresql-overview/advantages-of-postgresql/ Accessed: Sep 22, 2020.

[9] Unique identifier formula [Online] Available: https://gist.github.com/msubel/9c70951a20efe5c72195 Accessed: Feb 14, 2021.

[10] createStackNavigation [Online] Available: https://reactnavigation.org/docs/hello-react-navigation/ Accessed April 05, 2021.

[11] Fetch API [Online] Available: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API Accessed: May 02, 2021.

[12] AuthContext [Online] Available: https://www.youtube.com/watch?v=gvF6sFIPfsQ Accessed: May 10, 2021.

[13] Knex [Online] Available: http://knexjs.org/ Accessed: May 01, 2021.

[14] Knex Query Builder. [Online]. Available: http://knexjs.org/#Builder Accessed: May 01, 2021

[15] Content vs Collaborative. [Online] Available: https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421#:~:text=Content%2Dbased%20filtering%20does%20not,and%20items%20on%20its%20own. Accessed: Sep 21, 2020

[16] SVD machine learning mastery [Online] Available: https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/ Accessed: Sep 23, 2020

[17] Surprise SVD [Online] Available: https://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.matrix_factorization.SVDpp Accessed: Mar 12, 2021

[18] Surprise Dataset [Online] Available: https://surprise.readthedocs.io/en/stable/dataset.html Accessed: Mar 24, 2021