웹 크롤링과 인공지능을 활용한 개인화 추천 반응형 웹사이트 개발

팀　　명 : GAMFLIX
지도 교수 : 양환석 교수님
팀　　장 : 윤다예
팀　　원 : 오예지

2021. 11.
중부대학교 정보보호학과

목 차

4. 결론

5. 별첨

1. 서론

1.1 연구 배경 및 필요성

빠르게 발전하는 정보화 사회에서 데이터의 양과 종류가 늘어남에 따라 콘텐츠 이용자들은 수많은 정보를 탐색한다. 이용자들은 다양한 선택 대안들 가운데에서 최선의 결정을 내려 유용한 정보를 찾는 과정에서 많은 시간과 노력을 들이는 것에 어려움을 느끼고 있다. 정보의 과잉 환경은 결정 장애와 햄릿 증후군이라는 신조어를 만들어 낸 만큼 개인형 맞춤 서비스의 필요성과 가치를 더욱 높이고 있다. 이에 따라 여러 분야에서는 무분별하게 늘어나는 정보들로 인한 정보 과잉시대에서 자신에게 맞는 콘텐츠를 찾기 쉽도록 개인화된 맞춤형 서비스를 제공하고 있다. 따라서 이번 연구를 통해 이용자의 관심사에 맞게 맞춤형 정보를 제공한다.

NAVER	행동 데이터 기반 상품 추천 시스템인 'AiTEMS' 도입 이미지로 원하는 상품 검색할 수 있는 '쇼핑렌즈' 개발
coupang	.AI가 고객 데이터 매일 3억 건 학습하는 맞춤형 검색
kakaocommerce	카카오 AI 기술을 활용해 패션 맞춤형 서비스 제공
Market Kurly	데이터 수집 및 분석을 통해 다음날 주문량 예측
ebay	풀 필먼트 서비스인 스마일 배송 자동화에 AI 활용
ssg.com	AI 머신러닝 기술을 통해 향후 일주일간 수요 예측

[기업의 맞춤형 활용 사례]

1.2 연구 목적 및 주제선정

사이트 이용자들이 개인에게 맞는 맞춤형 정보를 받아 정보 선택과 사용에 편의를 돕는 시스템으로 어떤 정보를 원하는지 웹 크롤링을 통하여 정보를 수집하고 분석하여 개인화된 최적의 맞춤형 서비스를 제공해주는 지능형 지식 플랫폼을 개발하는 것이 목표이며, 주요수행 내용은 이용자들에게 콘텐츠 기반 추천 알고리즘, 협업 기반 추천 알고리즘, 인공지능을 사용하여 맞춤형 추천 서비스를 제공한다.

2. 관련 연구

2.1 HTML

하이퍼 텍스트 마크업 언어(Hyper Text Markup Language)인 HTML은 1980년 유럽 입자 물리 연구소의 계약자인 물리학자 팀 버너스리가 연구원들이 문서를 이용하고 공유하기 위한 체계인 인콰이어를 제안하면서 시작됐다. HTML은 제목, 단락, 목록 등과 같이 구조적인 의미뿐만 아니라 링크, 이용 등 본문 이외의 항목으로 구조적 문서를 만들 방법을 제공한다. 또 이미지와 객체를 내장하고 대화형 양식을 생성하는 데 사용된다. HTML은 웹 브라우저와 같이 HTML 처리 장치의 행동에 영향을 주는 자바스크립트와 본문 이외의 항목의 외관이나 배치를 정의하는 CSS와 같은 스크립트를 포함하거나 불러올 수 있

다.

2.2 CSS

1994년 하콤 비움 리가 처음 제안한 종속형 시트 또는 캐스케이딩 스타일 시트(Cascading Style Sheets)는 마크업 언어(ex: HTML)가 실제로 표시되는 방법을 나타내는 스타일 언어로 HTML과 XHTML에서 주로 사용되며, XML에서도 사용할 수 있다. 웹사이트에서 마크업 언어가 몸체를 담당한다면, CSS는 옷이나 액세서리와 같이 꾸미는 역할을 담당한다.

2.3 Javascript

객체 기반 스크립트 프로그래밍 언어인 자바스크립트는 웹 브라우저 내에서 주로 사용하며, 다른 응용프로그램의 내장 객체에도 접근할 수 있는 기능이 있다.

2.4 Python

1991년 프로그래머인 귀도 반 로섬이 발표한 고급 프로그래밍 언어이다. 플랫폼에서 독립적으로 사용하며, 인터프리터 방식, 객체 지향적, 동적 타이핑 대화형 언어이다.

2.5 Flask

웹 프라임워크인 플라스크는 WSGI 인터페이스 기반으로 웹 서버와 웹 응용프로그램 간의 웹 애플리케이션을 쉽게 개발할 수 있는 파이썬 모듈이다. 플라스크는 작고 확장하기 쉬운 코어를 가지고 있는데, ORM(Object Relational Manager)등의 기능이 포함되어 있지 않은 마이크로 프레임워크이다. 또 플라스크는 URL 라우팅, jinja2 같은 템플릿 엔진과 많은 기능이 있다.

2.6 콘텐츠 기반 추천 알고리즘

콘텐츠 기반 추천은 각 아이템 간 속성 정보 유사도에 기초하여 추천 대상을 선정한다. 각 콘텐츠를 설명하는 요소로 쪼갠 후 비슷한 요소를 가진 콘텐츠를 다른 이용자에게 추천한다. 예를 들어 이번 연구에서 한 콘텐츠의 요소가 3D, 유료, RPG(Role Playing Game)의 요소를 가지고 있다면, 같은 요소를 가지고 있는 콘텐츠를 다른 이용자에게 추천하는 방식이다. 콘텐츠 기반 추천 알고리즘은 콘텐츠 자체를 분석하기 때문에 이용자 행동 정보가 많이 필요한 협업 기반 추천 알고리즘의 단점을 보완하지만, 다양한 형식의 항목을 추천하기 어렵다.

2.7 협업 기반 추천 알고리즘

협업 기반 추천 알고리즘은 이용자의 패턴이나 평점을 가지고 다른 사람들의 패턴, 평점을 통해 추천하는 방법이다. 추가적인 이용자의 개인정보가 없어도 추천할 수 있는 것이 장점이며, 2006년부터 2009년까지 개최된 Netflix Prize Competition에서 우승한 알고리즘으로 널리 알려졌다. 이용자들로부터 얻은 정보에 따라 관심사들을 자동으로 예측하

게 해주는 방법으로 행동, 활동, 선호도 등에 대한 많은 정보를 수집하고 분석하여 다른 이용자와의 유사성을 기반으로 이용자가 선호할만한 정보를 예측한다. 이번 연구에서 협업 기반 추천 알고리즘을 통해 이용 가능한 항목 집합과 다른 이용자의 같은 항목 집합과의 참여 정보를 기반으로 사용자의 선호도를 학습하여 관심사를 자동으로 예측한다.

2.8 AI

Artificial Intelligence 흔히 인공지능이라고 불리는 AI는 인간의 학습능력이나 추론 능력, 지각능력을 인공적으로 구현하기 위한 컴퓨터 과학의 세부 분야로 하나의 인프라 기술이다. 지능을 가진 기능을 갖춘 컴퓨터 시스템이며, 인간의 지능을 기계 등에 인공적으로 구현한 것으로 챗봇 작성에 사용했다.

2.9 웹 크롤러

조직적이고 자동화된 방법으로 Word Wide Web을 탐색하는 컴퓨터 프로그램이다. 웹 크롤러가 하는 작업을 웹 크롤링(web crawling) 또는 스파이더링(spidering)이라고 하며, 검색 엔진과 같은 여러 사이트에서는 데이터의 최신 상태 유지를 위해 웹 크롤링을 한다. 크롤러는 링크 체크나 HTML 코드 검증과 같이 웹사이트의 자동 유지 관리 작업을 위해 사용되며, 자동 수집과 같이 웹 페이지의 특정 형태 정보를 수집하는 데에 사용한다.

2.10 Hive

하이브는 하둡 에코 시스템 중에서 데이터를 모델링하고 프로세싱하는 경우 가장 많이 사용하는 데이터 웨어하우징용 솔루션이다. RDB의 데이터베이스, 테이블과 같은 형태로 HDFS에 저장된 데이터의 구조를 정의하는 방법을 제공하며, 이 데이터를 대상으로 SQL과 유사한 HiveQL 쿼리를 이용하여 데이터를 조회하는 방법을 제공한다.

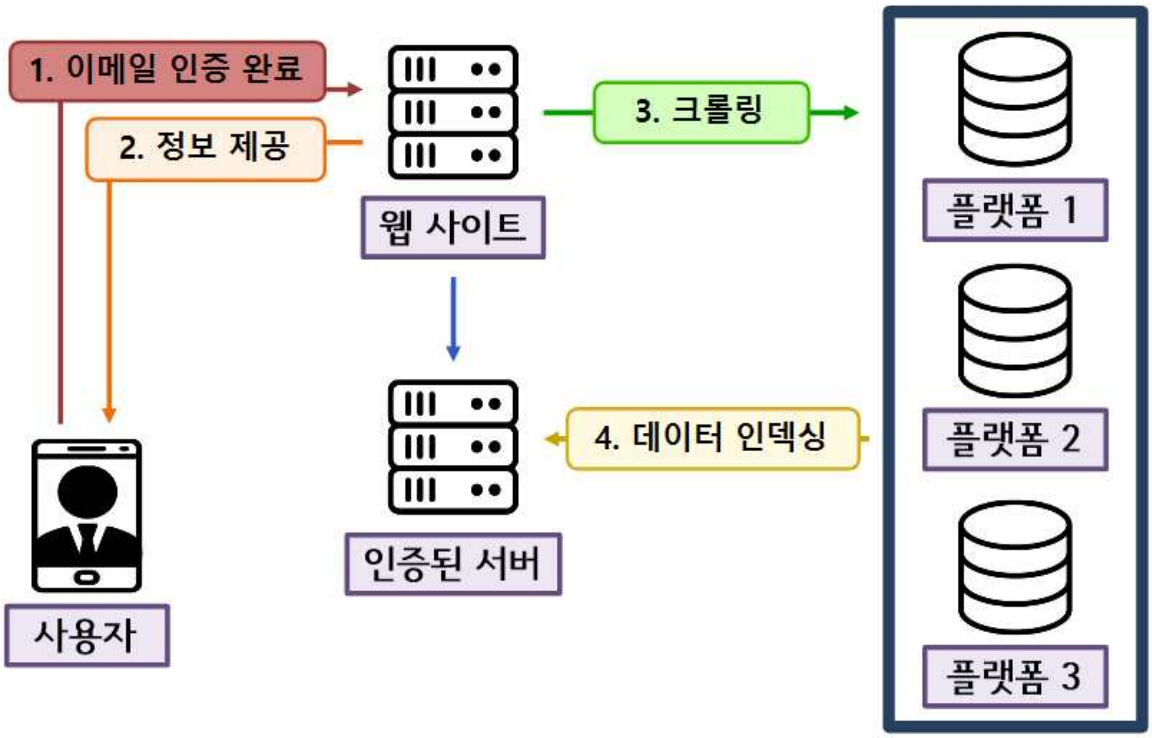
2.11 Apache Spark

Apache Spark는 메모리 내 처리를 지원하며 빅데이터를 분석하는 애플리케이션의 성능을 향상하는 오픈 소스 병렬 처리 프레임워크이다. 빅데이터 솔루션은 기존 데이터 베이스에 비해 너무 크거나 복잡한 데이터를 처리하도록 설계되었다. Spark는 메모리에서 대량의 데이터를 처리하므로 디스크 기반 대체 방법보다 훨씬 빠르다.

3. 본론

3.1 시스템 구성

이용자의 개인정보를 보호하고 안전하게 웹사이트를 사용할 수 있도록 이메일인증 시스템을 구현하였다. 이메일인증을 통해 이용자가 웹사이트에 가입하고 로그인을 한 후 웹사이트에 접근한다. 이용자가 개인화 맞춤형 추천 서비스에서 선호하는 콘텐츠를 선택하면 웹사이트에서는 각 게임 플랫폼에서 데이터를 크롤링한 후 게임의 속성 정보 태그의 유사도 또는 다른 이용자의 평점, 패턴 등을 기반으로 추천 대상을 선정하여 추천한다.



[그림 1. 프로그램 제작 구상도]

3.2 프로그램 구성

3.2.1 이메일인증

256bit로 맞춰진 문장인 년, 월, 일과 인증번호, 솔트값으로 이루어진 64비트로 나눈 후에 각각 and, or, XOR, 게이트로 복잡성을 더한 로테이션 연산을 통해 DB에 인증정보를 등록한다. 그 후, 인증을 시도할 때 DB에서 해쉬정보를 얻어와 해쉬값이 같은지 인증정보를 확인한다. 이메일 코드를 전송하고, 인증정보를 확인하는 과정을 통해 이용자들이 안전하게 사이트를 이용할 수 있도록 한다.

```
for z in range(64):
    f = (a & b) | (a & c) | (a & d)
    g = (~b & c) | (~a & d)
    h = (a | ~c) ^ (b | ~d)
    i = a ^ ~b ^ c ^ ~d

    a = i
    b = h
    c = f
    d = g
    if(z % 2 == 0):
        a, b, c, d = Rotr(a, b, c, d, (64*z)%256)
    else:
        a, b, c, d = Rotl(a, b, c, d, (64*c)%256)
```

[그림 2. 암호화 과정]

```python
def GetHashInDB(email):
    sql = 'SELECT * FROM authtable WHERE email="{0}"'.format(email)
    authdb_cursor.execute(sql)
    authdb.commit()

    data = authdb_cursor.fetchone()
    #print(data)
    return data['timestamp'], data['salt'], data['hash']
```

[그림 3. 해쉬정보]

```python
def AddAuth(email, timestamp, salt, hashed):
    # insert into authtable (email, timestamp, salt, hash) values (email, timestamp, salt, hash) on duplicate key update
    # email=email, timestamp=timestamp, salt=salt, hash=hash
    #print((binascii.crc32(bytes(email.encode("utf-8")))) & 0xffffffff)
    sql = 'INSERT INTO `authtable` (emailhash, email, timestamp, salt, hash) VALUES ("{0}", "{1}", "{2}", "{3}", "{4}") \
        ON DUPLICATE KEY UPDATE timestamp="{2}",salt="{3}",hash="{4}"'.format((binascii.crc32(bytes(email.encode("utf-8")))) &
    #print(sql)

    authdb_cursor.execute(sql)
    authdb.commit()
    return
```

[그림 4. 인증정보 등록]

```python
def CheckAuth(self, auth, timestamp, salt, thash):
    target = auth + timestamp + salt
    byte_array = target.encode()
    hashed = self._hash(byte_array[0:8], byte_array[8:16], byte_array[16:24], byte_array[24:])
    if(thash == hashed):
        return 'correct'
    else:
        return 'fail'
```

[그림 5. 인증번호 확인]

```python
@app.route('/auth', methods=['POST', 'GET'])
def auth():
    if(request.method == "POST"):
        result = request.form
        #print(result)

        # Hashing
        email = result['email']
        authcode = CreateAuthcode()
        print(authcode)
        time, salt, hashed = hash.getHash(authcode)
        AddAuth(email, time, salt, hashed)

        # Email
        html = open("mailTemplate.html", encoding="UTF-8").read().replace("email_code", authcode[:3] + " " + authcode[3:])
        msg = MIMEText(html, 'html')
        msg['Subject'] = '[GAMFILX] 요청하신 이메일 인증번호입니다'
        s.sendmail("official.gamfilx@gmail.com", email, msg.as_string())

        return render_template("auth.html", email=email)
    else:
        return "잘못된 접근입니다!"
```

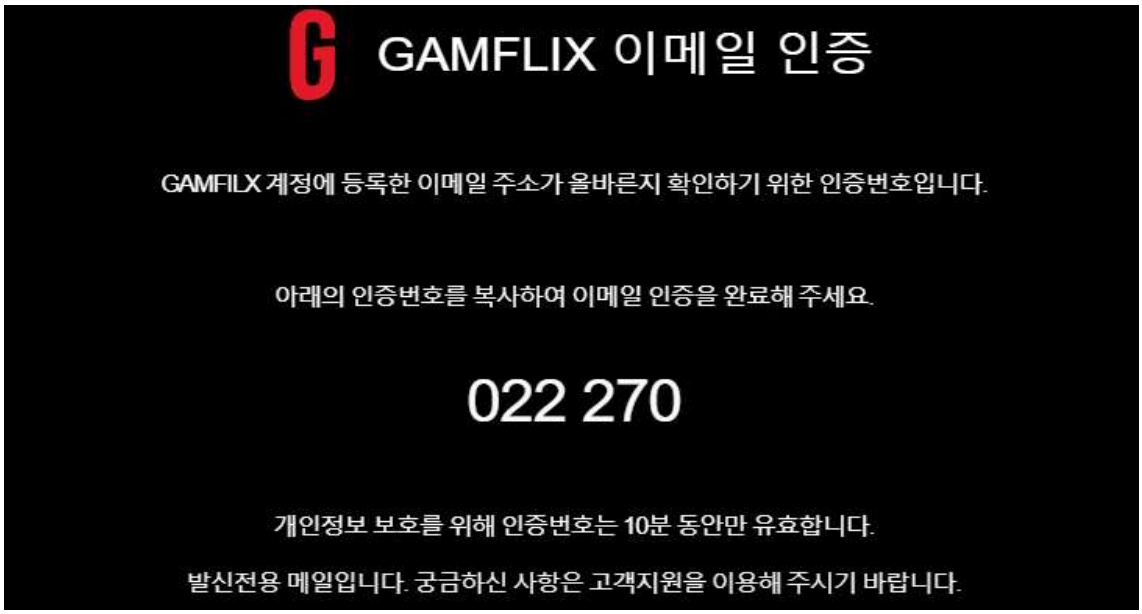
[그림 6. 이메일 코드 전송]

```python
@app.route('/authentification', methods=['POST', 'GET'])
def auththentification():
    if(request.method == "POST"):
        #print(request.form)
        email = request.form['email']
        authcode = request.form['authcode']
        authcode = str(authcode).strip()
        if(len(authcode) != 6): # Check authcode length
            return render_template("authfail.html")

        #print(email, authcode)
        gtime, gsalt, ghashed = GetHashInDB(email) # Get Data In DB
        if(hash.CheckAuth(authcode, gtime, gsalt, ghashed) == "correct"): # If Corret
            return render_template("authsucc.html")

    return render_template("authfail.html") # else All Fail
```

[그림 7. 인증번호 확인]



[그림 8. 이메일인증 메시지]

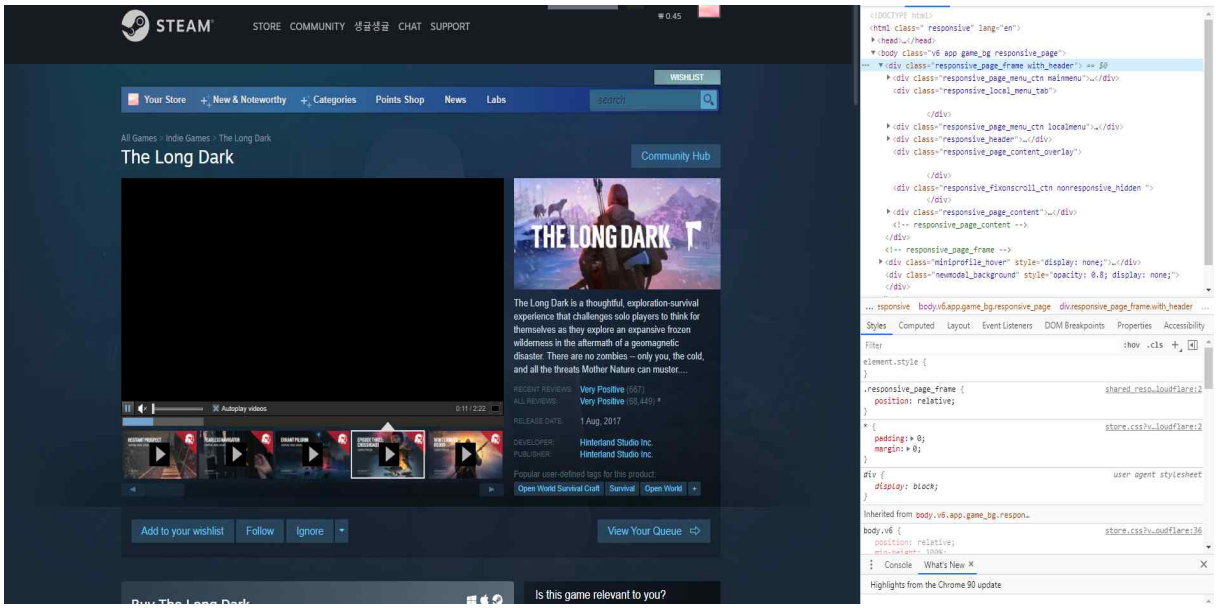
A	B	C	D			
64bit	64bit	64bit	64bit			
F	(A XOR B) + (A XOR C) + (A XOR D)					
G	(!B AND C) AND (!A AND D)			Rotate Right/Left 32bit And Repeat 64		
H	(A XOR !C) OR (B XOR !D)					
I	A XOR !B XOR C XOR !D					
64bit	64bit	64bit	64bit			
I	H	F	G			

[그림 9. 이메일인증 방법 모식화]

3.2.2 웹 크롤링

3.2.2.1 콘텐츠 기반 추천

게임 플랫폼에서 게임의 제목, 가격, 정보, 속성 등을 가져온 후 게임의 그래픽이나, 주제를 나타내는 태그를 가져와 CSV 파일로 저장하여 추천한다.



[그림 10. 가격, 정보, 속성 등 태그 탐색]

	A	B
74	https://store.steampowered.com/app/601150/Devil_May_Cry_5/	Steam
75	https://store.steampowered.com/app/594650/Hunt_Showdown/	Steam
76	https://store.steampowered.com/app/221640/Super_Hexagon/	Steam
77	https://store.steampowered.com/app/238320/Outlast/	Steam
78	https://store.steampowered.com/app/414700/Outlast_2/	Steam
79	https://store.steampowered.com/app/999220/Amnesia_Rebirth/	Steam
80	https://store.steampowered.com/app/57300/Amnesia_The_Dark_Descent/	Steam
81	https://store.steampowered.com/app/239200/Amnesia_A_Machine_for_Pigs/	Steam
82	https://store.steampowered.com/app/1259790/Puyo_Puyo_Tetris_2/	Steam
83	https://store.steampowered.com/app/546050/Puyo_PuyoTetris/	Steam
84	https://store.steampowered.com/app/286160/Tabletop_Simulator/	Steam
85	https://store.steampowered.com/app/646570/Slay_the_Spire/	Steam
86	https://store.steampowered.com/app/1222680/Need_for_Speed_Heat/	Steam
87	https://store.steampowered.com/app/620/Portal_2/	Steam
88	https://store.steampowered.com/app/400/Portal/	Steam
89	https://store.steampowered.com/app/70/HalfLife/	Steam
90	https://store.steampowered.com/app/220/HalfLife_2/	Steam
91	https://store.steampowered.com/app/546560/HalfLife_Alyx/	Steam
92	https://store.steampowered.com/app/380/HalfLife_2_Episode_One/	Steam
93	https://store.steampowered.com/app/420/HalfLife_2_Episode_Two/?snr=1_7_15__13	Steam
94	https://store.steampowered.com/app/50/HalfLife_Opposing_Force/?snr=1_7_15__13	Steam
95	https://store.steampowered.com/app/221910/The_Stanley_Parable/?snr=1_7_15__13	Steam
96	https://store.steampowered.com/app/500/Left_4_Dead/	Steam
97	https://store.steampowered.com/app/550/Left_4_Dead_2/	Steam
98	https://store.steampowered.com/app/644930/They_Are_Billions/	Steam
99	https://store.steampowered.com/app/231430/Company_of_Heroes_2/	Steam
100	https://store.steampowered.com/app/228200/Company_of_Heroes/	Steam
101	https://store.steampowered.com/app/389730/TEKKEN_7/	Steam
102	https://store.steampowered.com/app/477160/Human_Fall_Flat/	Steam
103	https://store.steampowered.com/app/1097150/Fall_Guys_Ultimate_Knockout/	Steam
104	https://store.steampowered.com/app/1313860/EA_SPORTS_FIFA_21/	Steam
105	https://store.steampowered.com/app/381210/Dead_by_Daylight/	Steam
106	https://store.steampowered.com/app/391540/Undertale/	Steam
107	https://store.steampowered.com/app/206440/To_the_Moon/	Steam
108	https://store.steampowered.com/app/250900/The_Binding_of_Isaac_Rebirth/	Steam
109	https://store.steampowered.com/app/239030/Papers_Please/	Steam
110	https://store.steampowered.com/app/620980/Beat_Saber/	Steam
111	https://store.steampowered.com/app/438100/VRChat/	Steam
112		
113		

[그림 11. 주소를 통한 태그 탐색]

.

```
if(line[1] == "Steam"):
    game_title = soup.find('div', {'class': 'apphub_AppName'})
    total.append(game_title.get_text())

    print("Done.\t\t", end='')
    print(game_title.get_text() + "(Steam)")

    try:
        game_price = soup.find('div', {'class' : 'game_purchase_price price'})
        total.append(game_price.get_text().strip())
    except:
        try:
            game_price = soup.find('div', {'class' : 'discount_original_price'})
            total.append(game_price.get_text().strip())
        except:
            game_price = "UNKNOWN"
            total.append(game_price)
```

[그림 12. 가격, 정보 탐색]

```
game_sub = soup.find('div', {'class' : 'glance_ctn_responsive_right'})
game_sub = game_sub.get_text().replace("+", "").replace("\t", " ").split("\n")
for i in range(0, len(game_sub)):
    game_sub[i] = game_sub[i].strip()
total = total + game_sub[6:-3]
```

[그림 13. 연관 태그 탐색]

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Among Us	₩ 18,480	Multiplayer	Online Co-(	Space	Social Dedu	Survival	2D	Funny	Co-op	Psychologic	Local Multip	Cartoony	Action	Casual	Party Game	Sci-fi	Colorful	PvP	Minigames	Top-Down	Aliens	
2																							
3	A Dance of	₩ 6,500	Rhythm	Indie	Music	Difficult	Great Soun	Singleplayer	2D	Action	Level Editor												
4																							
5	DJMAX RES	₩ 49,800	Casual	Rhythm	Action	Sports	Music	Great Soun	Anime	Multiplayer	Early Acces	Competitive	Singleplayer	Difficult	Arcade	Nudity							
6																							
7	ARK: Survi	₩ 51,000	Open World	Survival	Open World	Multiplayer	Dinosaurs	Crafting	Building	Adventure	Base Buildir	Co-op	Action	First-Persor	Sandbox	Massively M	Singleplayer	Early Acces	RPG	Dragons	Sci-fi	Indie	
8																							
9	ASTRONEE	₩ 31,000	Open World	Open World	Multiplayer	Space	Survival	Exploration	Base Buildir	Sandbox	Co-op	Adventure	Building	Crafting	Sci-fi	Simulation	Third Persor	Colorful	Singleplayer	Automation	Atmospheric	Indie	
10																							
11	The Forest	₩ 20,500	Open World	Survival	Open World	Horror	Crafting	Adventure	Building	Survival Ho	First-Persor	Action	Exploration	Sandbox	Atmospheric	Simulation	Singleplayer	Indie	Realistic	Gore	Early Acces	Zombies	
12																							
13	Raft	₩ 21,000	Survival	Open World	Multiplayer	Co-op	Crafting	Open World	Building	Base Buildir	Adventure	Online Co-(	Sandbox	Early Acces	First-Persor	Singleplayer	Underwater	Simulation	Action	Indie	Strategy	Massively Multiplayer	
14																							
15	Rust	₩ 41,000	Survival	Crafting	Multiplayer	Open World	Open World	Building	Sandbox	PvP	Adventure	First-Persor	Action	FPS	Nudity	Co-op	Shooter	Online Co-(	Indie	Early Acces	Post-apoca	Simulation	
16																							
17	Valheim	₩ 20,500	Open World	Online Co-(	Survival	Open World	Crafting	Building	Exploration	Multiplayer	Co-op	Base Buildir	Sandbox	Mythology	Adventure	Sailing	Early Acces	RPG	Action	Indie	PvP	Fishing	
18																							
19	Don't Starv	₩ 10,500	Survival	Open World	Crafting	Adventure	Indie	Sandbox	Singleplayer	Perma Deat	Open World	Exploration	Multiplayer	Roguelike	Replay Valu	Moddable	Difficult	Simulation	2D	Top-Down	Survival Ho	Horror	
20																							
21	Subnautica	₩ 31,000	Open World	Survival	Open World	Exploration	Underwater	Crafting	Base Buildir	Horror	Singleplayer	Adventure	First-Persor	Sci-fi	Sandbox	Aliens	Atmospheric	Survival Ho	Action	Early Acces	Multiplayer	Indie	
22																							
23	The Long D	₩ 17,000	Open World	Survival	Open World	Exploration	Atmospheric	First-Persor	Singleplayer	Adventure	Crafting	Sandbox	Simulation	Post-apoca	Indie	Walking Sim	Survival Ho	Strategy	Action	Difficult	Early Acces	Horror	
24																							
25	No Man's S	₩ 60,000	Open World	Open World	Space	Exploration	Sci-fi	Survival	Procedural	Adventure	First-Persor	Singleplayer	Sandbox	Atmospheric	Crafting	Multiplayer	Space Sim	Indie	Action	Simulation	FPS	Great Soundtrack	
26																							
27	Garry's Mo	₩ 10,500	Sandbox	Multiplayer	Funny	Moddable	Building	Comedy	Co-op	Mod	First-Persor	Physics	Simulation	Online Co-(	FPS	Singleplayer	Action	Shooter	Animation &	Indie	Massively M	Adventure	
28																							
29	Microsoft Fl	₩ 59,900	Simulation	Flight	Open World	Multiplayer	Realistic	Singleplayer	Atmospheric	Physics	Real-Time	Adventure	Colorful	VR	Family Frien	TrackIR	Logic	Life Sim	Surreal	Beautiful	Short	Auto Battler	
30																							
31	Sid Meier's	₩ 65,000	Strategy	Turn-Based	Historical	Multiplayer	Singleplayer	Turn-Based	Grand Strat	4X	War	Simulation	Tactical	City Builder	Great Soun	Moddable	Online Co-(	Building	Managemer	Co-op	Hex Grid	Atmospheric	
32																							
33	Sid Meier's	₩ 32,000	Turn-Based	Strategy	Turn-Based	Multiplayer	Historical	4X	Addictive	Singleplayer	Hex Grid	Grand Strat	Replay Valu	Co-op	Tactical	Economy	Diplomacy	Moddable	Simulation	Classic	Great Soun	Touch-Friendly	
34																							
35	Euro Truck	₩ 23,500	Simulation	Driving	Automobile	Open World	Realistic	Relaxing	Singleplayer	Moddable	Exploration	First-Persor	Economy	Atmospheric	Adventure	Indie	Managemer	Casual	TrackIR	Sandbox	Racing	RPG	
36																							
37	Farming Sim	₩ 19,800	Simulation	Farming Sim	Multiplayer	Agriculture	Realistic	Driving	Open World	Automobile	Singleplayer	Managemer	Moddable	Co-op	Resource M	Relaxing	First-Persor	Online Co-(	Economy	Family Frien	Atmospheric	Strategy	
38																							
39	Hand Simul	₩ 2,200	Simulation	Funny	Multiplayer	Difficult	VR	Memes	First-Persor	Comedy	Psychologic	Intentionally	Action	FPS	Horror	Singleplayer	Sexual Con	Nudity	Atmospheric	World War I	Strategy	Open World	
40																							
195	They Are Bil	₩ 31,000	Base Buildir	Colony Sim	Strategy	Survival	Zombies	RTS	Tower Defer	Steampunk	City Builder	Building	Post-apoca	Singleplayer	Resource M	Real-Time v	Difficult	Tactical	Managemer	Early Acces	Indie	Multiplayer	
196																							
197	Company o	₩ 14,000	Strategy	World War I	RTS	Multiplayer	Tactical	Historical	War	Military	Singleplayer	Co-op	Online Co-(	Tanks	Action	Real-Time	Base Buildir	Real-Time v	Atmospheric	Simulation	Great Soun	Adventure	
198																							
199	Company o	₩ 21,000	Strategy	World War I	RTS	War	Multiplayer	Action	Singleplayer	Tactical	Violent	Military	Historical	Base Buildir	Co-op	Classic	Mod	Moddable	Real Time T	Great Soun	Atmospheric		
200																							
201	TEKKEN 7	₩ 115,270	Fighting	Action	Multiplayer	Competitive	Arcade	Local Multi	Controller	Sports	Great Soun	Singleplayer	eSports	Character C	Difficult	3D Fighter	Anime	Story Rich	Beat 'em up	Third Person	Nudity	Sexual Content	
202																							
203	Human: Fall	₩ 20,500	Funny	Multiplayer	Co-op	Puzzle	Physics	Adventure	Indie	Local Co-C	Comedy	Parkour	Local Multi	Puzzle Platf	Sandbox	Open World	Casual	Singleplayer	3D Platform	Split Screen	Simulation	Survival	
204																							
205	Fall Guys: U	₩ 20,500	Multiplayer	Funny	Battle Roya	Online Co-(	Family Frien	Party Game	Comedy	Cute	Casual	Physics	Massively M	Colorful	3D Platform	Indie	Co-op	Action	Controller	Great Soun	Difficult	Local Co-Op	
206																							
207	EA SPORTS	₩ 88,000	Sports	Soccer	Football	PvP	Competitive	eSports	Multiplayer	Simulation	Controller	Local Multi	Realistic	Online Co-(	Singleplayer	Local Co-C	Modern	Co-op	Casual	Runner	Great Soun	Souls-like	
208																							
209	Dead by Da	₩ 21,000	Horror	Survival Ho	Multiplayer	Online Co-(	Survival	Stealth	Gore	Co-op	Blood	Team-Base	Third Persor	Violent	First-Persor	Action	Atmospheric	Psychologic	Strategy	Comedy	Mature	Difficult	
210																							
211	Undertale	₩ 10,500	Great Soun	Story Rich	Choices Ma	Multiple End	Funny	Pixel Graph	RPG	Singleplayer	Indie	Comedy	2D	Replay Valu	Cute	Bullet Hell	Memes	Retro	Dark	Psychologic	Dating Sim	Horror	
212																							
213	To the Moo	₩ 10,500	Story Rich	Great Soun	Indie	Emotional	RPGMaker	Adventure	Pixel Graph	Singleplayer	Interactive F	RPG	Romance	Short	2D	Funny	Psychologic	Retro	Linear	Cute	Casual	Female Protagonist	
214																							
215	The Binding	₩ 15,500	Action Rogu	Roguelike	Indie	Replay Valu	Difficult	Pixel Graph	Dungeon Cr	Great Soun	Singleplayer	Dark	Remake	2D	Local Co-C	Gore	Procedural	Roguelite	Atmospheric	Action	Horror	Co-op	
216																							
217	Papers, Ple	₩ 10,500	Indie	Political	Simulation	Singleplayer	Point & Clic	Dystopian	Pixel Graph	Puzzle	Multiple End	Retro	Story Rich	Atmospheric	2D	Great Soun	Addictive	Strategy	Casual	Adventure	Replay Valu	Difficult	
218																							
219	Beat Saber	₩ 31,000	VR	Rhythm	Music	Great Soun	Indie	Fast-Paced	Singleplayer	Moddable	First-Persor	Difficult	Swordplay	Multiplayer	Action	Sports	Memes	Futuristic	Early Acces	Music-Base	Casual	Anime	
220																							
221	VRChat	Free To Pla	VR	Free to Play	Memes	Anime	Funny	Multiplayer	Massively M	Early Acces	First-Persor	Open World	Simulation	Psychologic	Casual	Adventure	Horror	Sports	MMORPG	FPS	Comedy	Action	
222																							
223																							

[그림 15. CSV 파일]

3.2.2.2 협업 기반 추천

엔진을 구현하기 위해 공식 사이트와 웹 API에서 이용자의 정보를 수집하는 멀티스레딩 웹 크롤러를 개발하고, 현재까지 이용자들의 동의하에 약 200여 명의 정보를 수집한다. 그 후 ID를 사용하여 API에 액세스하여 소유한 게임, 최근 플레이한 게임 및 게임 실행 시간 등의 콘텐츠를 수집하여 JSON 포맷으로 하이브에 저장한다.

패키지는 PySpark를 사용하며, 권장 엔진의 주요 업무는 Spark SQL과 Hive SQL을 통해 데이터를 처리하고 Spark의 ALS API를 통해 협업 기반 추천 알고리즘을 개발한다.

게임 API를 통해 이용자의 상세 데이터에 접근하기 위해 동의하에 이용자 ID를 수집하고, BeautifulSoup과 Regular Expression을 이용하여 이용자의 프로필 페이지를 탐색한다.

[데이터 수집]

```
def get_online_users(member_list_no, user_ids):
    url = 'https://steamcommunity.com/games/steam/members?p=' + str(member_list_no)
    resp = requests.get(url)

    soup = bs(resp.text, 'html.parser')
    # print(soup.prettify())

    # search profile of users who are online/in-game
    all_users = soup.find_all("div", ₩
                onclick                                                                    =
re.compile("top₩.location₩.href='https:₩/₩/steamcommunity₩.com₩/id₩/(₩w+)'"), ₩
                class_ = re.compile("online|in-game"))

    # get user names
    for user in all_users:
        user_profile = user.div.div.div.a['href'].encode("ascii")
        # print user_profile
        get_user_id(user_profile, user_ids)
```

이용자의 게임 목록을 API를 통해 탐색한다.

[상세 데이터 수집]

```
def get_game_detail(app_id_list, num, game_detail_out_file):
    url = 'https://store.steampowered.com/api/appdetails?appids='
        'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
        'Accept-Encoding':'gzip, deflate, sdch',
        'Accept-Language':'en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4,zh-TW;q=0.2'}
    with open(game_detail_out_file, 'w') as f:
        for idx in xrange(num):
            url_temp = url + str(app_id_list[idx])
            time.sleep(.100)
```

```
            resp = requests.get(url_temp, header)

            obj = resp.json()
            for key in obj:
                if obj[key]["success"] is True :
                    json.dump(obj[key]["data"], f)
                    f.write('₩n')
```

알고리즘의 오버플로우를 방지하기 위해 ID를 인덱스로 변환한다. 이용자 ID 테이블과 game_detail 테이블에 가입하여 최종 결과가 작성된다.

[협업 필터링 시스템 엔진]

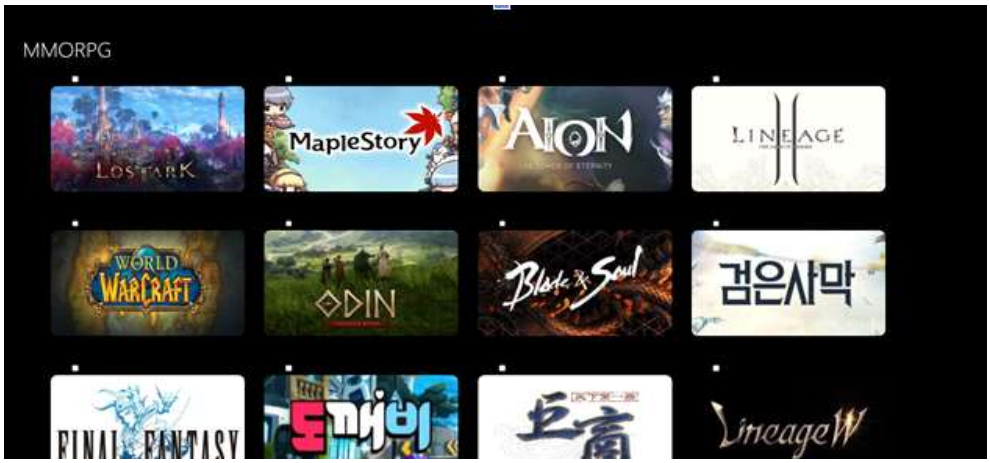
```
df_user_recent_games = hiveCtx.read.json(sample_user_recent_games)

# df_user_recent_games.printSchema()
df_user_recent_games.registerTempTable("user_recent_games")
df_valid_user_recent_games = hiveCtx.sql("SELECT * FROM user_recent_games where total_count != 0")

df_recommend_result.registerTempTable('recommend_result')
df_final_recommend_result = hiveCtx.sql("SELECT DISTINCT b.user_id, a.rank, c.name, c.header_image, c.steam_appid ₩
                                        FROM recommend_result a, user_idx b, game_detail c ₩
                                        WHERE a.user_idx = b.user_idx AND a.game_id = c.steam_appid ₩
                                        ORDER BY b.user_id, a.rank")
df_final_recommend_result.show(20)
# +-----------------+----+--------------------+--------------------+-----------+
# |          user_id|rank|                name|        header_image|steam_appid|
# +-----------------+----+--------------------+--------------------+-----------+
# |76561197960292467|   1|        Team Fortress 2|http://cdn.akamai...|        440|
# |76561197960292467|   2|PLAYERUNKNOWN'S B...|http://cdn.akamai...|     578080|
# |76561197960292467|   3|Sid Meier's Civil...|http://cdn.akamai...|       8930|
# |76561197960292467|   4|           Warframe|http://cdn.akamai...|     230410|
# |76561197960292467|   5|  Dead by Daylight|http://cdn.akamai...|     381210|
# |76561197960292467|   6|          Stellaris|http://cdn.akamai...|     281990|
# |76561197960292467|   7|          Fallout 4|http://cdn.akamai...|     377160|
# |76561197960292467|   8|     Assetto Corsa|http://cdn.akamai...|     244210|
# |76561197960292467|   9|The Elder Scrolls...|http://cdn.akamai...|     306130|
# |76561197960292467|  10|          XCOM® 2|http://cdn.akamai...|     268500|
# |76561197960315617|   1|Grand Theft Auto V|http://cdn.akamai...|     271590|
# |76561197960315617|   2|Starpoint Gemini 2|http://cdn.akamai...|     236150|
```

```
# |76561197960315617|     3|           Awesomenauts|http://cdn.akamai...|        204300|
# |76561197960315617|     4|Don't Starve Toge...|http://cdn.akamai...|        322330|
# |76561197960315617|     6|         Rocket League®|http://cdn.akamai...|        252950|
# |76561197960315617|     9|       Team Fortress 2|http://cdn.akamai...|           440|
# |76561197960315617|    10|            Fallout 4|http://cdn.akamai...|        377160|
# |76561197960384723|     1|              Dota 2|http://cdn.akamai...|           570|
# |76561197960384723|     2|          Garry's Mod|http://cdn.akamai...|          4000|
# |76561197960384723|     3|       Team Fortress 2|http://cdn.akamai...|           440|
# +-----------------+----+--------------------+--------------------+-----------+
```

마지막으로 플라스크 프레임워크 기반의 간단한 웹 UI를 개발하여 시각적 추천 결과를 표시한다. 결과 페이지에는 총 재생 시간에 기초하여 10개의 게임이 추천된다. URL에서 특정 이용자에 대한 ID를 라우팅 매개 변수로 입력하면 알고리즘에 기반을 두어 권장 결과가 표시된다.



[그림 16. 게임 선택 화면]

Profile

Recommendation

Game recommendation based on your profile:

Show 10 entries Search:

Game	Genre	Year Of Release	Critic Score	Score of Recommendation
TOM CLANCY'S SPLINTER CELL: BLACKLIST	ACTION	2013	84	0.6807530187260948
FAR CRY 4	SHOOTER	2014	85	0.5919638687123943
XENOBLADE CHRONICLES X	ROLE-PLAYING	2015	84	0.5805810320596199
STREET FIGHTER X TEKKEN	FIGHTING	2012	84	0.5546478848775244
GUITAR HERO LIVE	MISC	2015	84	0.5511831437062652
SOUTH PARK: THE STICK OF TRUTH	ROLE-PLAYING	2014	85	0.5462968441602531
DEUS EX: MANKIND DIVIDED	ROLE-PLAYING	2016	84	0.5333683417612376
RARE REPLAY	MISC	2015	84	0.5279096823538844

[그림 17. 게임 추천 목록]

4. 결론

4.1 결론

맞춤형 서비스는 다양한 분야에서 정보 과잉시대에 효과적으로 대응할 수 있는 전략으로 많은 관심과 발전을 이루고 있다. 수많은 정보와 콘텐츠로 인해 정보 수집, 선택, 의사결정 과정에서 어려움을 겪고 있는 이용자에게 정보를 추천해주는 맞춤형 서비스는 개인의 관심이나 취향을 분석하여 자신에게 어울리는 맞춤형 정보를 제공한다.

콘텐츠 기반 추천 알고리즘과 협업 기반 추천 알고리즘을 통하여 각 사이트에서 크롤링한 정보와 인공지능을 활용한 맞춤형 웹사이트를 제작하고 사이트가 가지고 있는 자료를 인증 및 보안을 통해 이용자들이 사이트를 더 편리하고 안전하게 웹사이트를 이용할 수 있도록 도와준다. 또 개인 맞춤형 사이트로써 이용자들의 정보를 안전하게 수집하고 분석하여 원하는 콘텐츠를 선별하고 추천하여 개인에게 맞는 최적의 맞춤형 서비스를 제공한다.

4.2 기대효과

이번 연구를 통해 웹크롤링과 인공지능을 활용한 맞춤형 웹사이트 제작이라는 목표를 달성하기 위해 협력하는 과정에서 팀워크의 중요성을 깨닫고, 개인 기술 개발 역량을 발전시킬 수 있는 계기가 되었으며, 웹사이트의 데이터 인증 및 보안을 통해 이용자들이 안전하게 사용할 수 있는 사이트를 제작하여 정보보호의 중요성을 인지한다.

5. 별첨

5.1 소스코드

package

```
Flask==1.1.2
Flask-Login==0.5.0
Flask-SQLAlchemy==2.5.1
gunicorn==20.1.0
Jinja2==2.11.2
joblib==1.0.1
lightgbm==3.2.0
numpy==1.20.2
pandas==1.2.3
psycopg2==2.8.6
pylint-flask==0.6
requests==2.25.1
scikit-learn==0.24.1
scikit-optimize==0.8.1
scipy==1.6.2
```

```
SQLAlchemy==1.4.7
Werkzeug==1.0.1
```

main.py

```
from flask import Blueprint, render_template, request, flash, send_file, send_from_directory
from flask_login import login_required, current_user
from models import V_GAMES, USERGAMESPLAYED, qtd_rows, User, gamesunplayed, gamesplayed, conn, check_gamesplayed
from __init__ import db
from ml_utils import predict_api
from games import save_ml_models
import datetime
import pandas as pd
import os

main = Blueprint('main', __name__)

def get_recommendation():
    df_gamesunplayed = gamesunplayed()
    df_gamesunplayed.columns = map(lambda x: str(x).upper(), df_gamesunplayed.columns)
    p = predict_api()
    output = {"ID_USER": df_gamesunplayed['ID_USER'], "ID_GAME": df_gamesunplayed['ID_GAME'], "NM_GAME": df_gamesunplayed['NM_GAME'], "NM_PUBLISHER": df_gamesunplayed['NM_PUBLISHER'], "NM_GENRE": df_gamesunplayed['NM_GENRE'], "QT_GAMES": df_gamesunplayed['QT_GAMES'], "NR_CRITICSCORE": df_gamesunplayed['NR_CRITICSCORE'], "DT_YEAROFRELEASE": df_gamesunplayed['DT_YEAROFRELEASE'], "IC_PLAYED": df_gamesunplayed['IC_PLAYED'], "Score": p}
    recommendations_df = pd.DataFrame(output)
    recommendations_df = recommendations_df.sort_values('Score',ascending=False)
    recommendations_df = recommendations_df.reset_index(drop=True)
    return recommendations_df

@main.route('/')
def index():
    if current_user.is_authenticated:
        if " " not in current_user.name:
            first_name = current_user.name.rsplit(' ', 1)[0]
            last_name = " "
        else:
            first_name = current_user.name.rsplit(' ', 1)[0]
            last_name = current_user.name.rsplit(' ', 1)[1]
```

```
            return render_template('index.html', first_name=first_name, last_name=last_name)
        else:
            return render_template('index.html')

@main.route('/donwloadrecommendation', methods=('GET', 'POST'))
def donwloadrecommendation():
    if request.method =='GET':
        df_profile = db.session.query(USERGAMESPLAYED.ID_USER,
USERGAMESPLAYED.NM_GAME, V_GAMES.DT_YEAROFRELEASE, V_GAMES.NM_GENRE,
V_GAMES.NR_CRITICSCORE).select_from(USERGAMESPLAYED).join(V_GAMES,
V_GAMES.ID_GAME ==
USERGAMESPLAYED.ID_GAME).filter(USERGAMESPLAYED.ID_USER==current_user.id).count()
        if df_checkgamesplayed is None or df_profile == 0:
            return None
        else:
            recommendations_df = get_recommendation()
            recommendations_df.to_csv(r'recommendations.csv', index=False)
            recommendations_file = (r'recommendations.csv')
            return send_file(recommendations_file, as_attachment=True)

@main.route('/donwloadprofile', methods=('GET', 'POST'))
def donwloadprofile():
    if request.method =='GET':
        df_profile = db.session.query(USERGAMESPLAYED.ID_USER,
USERGAMESPLAYED.NM_GAME, V_GAMES.DT_YEAROFRELEASE, V_GAMES.NM_GENRE,
V_GAMES.NR_CRITICSCORE).select_from(USERGAMESPLAYED).join(V_GAMES,
V_GAMES.ID_GAME ==
USERGAMESPLAYED.ID_GAME).filter(USERGAMESPLAYED.ID_USER==current_user.id).count()
        if df_checkgamesplayed is None or df_profile == 0:
            return None
        else:
            profile_table = db.session.query(USERGAMESPLAYED.ID_USER,
USERGAMESPLAYED.NM_GAME, V_GAMES.DT_YEAROFRELEASE, V_GAMES.NM_GENRE,
V_GAMES.NR_CRITICSCORE).select_from(USERGAMESPLAYED).join(V_GAMES,
V_GAMES.ID_GAME ==
USERGAMESPLAYED.ID_GAME).filter(USERGAMESPLAYED.ID_USER==current_user.id)
            profile_df = pd.DataFrame(profile_table, columns=['ID_USER', 'NM_GAME',
'DT_YEAROFRELEASE', 'NM_GENRE', 'NR_CRITICSCORE'])
            profile_df.to_csv(r'profile.csv', index=False)
            profile = (r'profile.csv')
            return send_file(profile, as_attachment=True)
```

```
@main.route('/donwloadrecommendation', methods=('GET', 'POST'))
def donwloadrecommendation():
    if request.method =='GET':
        df_profile = db.session.query(USERGAMESPLAYED.ID_USER, USERGAMESPLAYED.NM_GAME, V_GAMES.DT_YEAROFRELEASE, V_GAMES.NM_GENRE, V_GAMES.NR_CRITICSCORE).select_from(USERGAMESPLAYED).join(V_GAMES, V_GAMES.ID_GAME == USERGAMESPLAYED.ID_GAME).filter(USERGAMESPLAYED.ID_USER==current_user.id).count()
        if df_checkgamesplayed is None or df_profile == 0:
            return None
        else:
            recommendations_df = get_recommendation()
            recommendations_df.to_csv(r'recommendations.csv', index=False)
            recommendations_file = (r'recommendations.csv')
            return send_file(recommendations_file, as_attachment=True)

@main.route('/donwloadprofile', methods=('GET', 'POST'))
def donwloadprofile():
    if request.method =='GET':
        df_profile = db.session.query(USERGAMESPLAYED.ID_USER, USERGAMESPLAYED.NM_GAME, V_GAMES.DT_YEAROFRELEASE, V_GAMES.NM_GENRE, V_GAMES.NR_CRITICSCORE).select_from(USERGAMESPLAYED).join(V_GAMES, V_GAMES.ID_GAME == USERGAMESPLAYED.ID_GAME).filter(USERGAMESPLAYED.ID_USER==current_user.id).count()
        if df_checkgamesplayed is None or df_profile == 0:
            return None
        else:
            profile_table = db.session.query(USERGAMESPLAYED.ID_USER, USERGAMESPLAYED.NM_GAME, V_GAMES.DT_YEAROFRELEASE, V_GAMES.NM_GENRE, V_GAMES.NR_CRITICSCORE).select_from(USERGAMESPLAYED).join(V_GAMES, V_GAMES.ID_GAME == USERGAMESPLAYED.ID_GAME).filter(USERGAMESPLAYED.ID_USER==current_user.id)
            profile_df = pd.DataFrame(profile_table, columns=['ID_USER', 'NM_GAME', 'DT_YEAROFRELEASE', 'NM_GENRE', 'NR_CRITICSCORE'])
            profile_df.to_csv(r'profile.csv', index=False)
            profile = (r'profile.csv')
            return send_file(profile, as_attachment=True)

@main.route('/profile/<int:page_num>', methods=('GET', 'POST'))
@login_required
def profile(page_num):
    profile = db.session.query(USERGAMESPLAYED.ID_USER, USERGAMESPLAYED.NM_GAME,
```

```
V_GAMES.DT_YEAROFRELEASE, V_GAMES.NM_GENRE, V_GAMES.NR_CRITICSCORE).select_from(USERGAMESPLAYED).join(V_GAMES, V_GAMES.ID_GAME == USERGAMESPLAYED.ID_GAME).filter(USERGAMESPLAYED.ID_USER==current_user.id).paginate(per_page=len(qtd_rows), page=page_num, error_out=True)
    df_profile = db.session.query(USERGAMESPLAYED.ID_USER, USERGAMESPLAYED.NM_GAME, V_GAMES.DT_YEAROFRELEASE, V_GAMES.NM_GENRE, V_GAMES.NR_CRITICSCORE).select_from(USERGAMESPLAYED).join(V_GAMES, V_GAMES.ID_GAME == USERGAMESPLAYED.ID_GAME).filter(USERGAMESPLAYED.ID_USER==current_user.id).count()

    if " " not in current_user.name:
        first_name = current_user.name.rsplit(' ', 1)[0]
        last_name = " "
    else:
        first_name = current_user.name.rsplit(' ', 1)[0]
        last_name = current_user.name.rsplit(' ', 1)[1]

    df_checkgamesplayed = check_gamesplayed()
    if df_checkgamesplayed is None or df_profile == 0:
        recommendations_df = pd.DataFrame()
        recommendations_df[["ID_USER", "ID_GAME", "NM_GAME", "NM_PUBLISHER", "NM_GENRE", "QT_GAMES", "NR_CRITICSCORE", "DT_YEAROFRELEASE", "IC_PLAYED", "Score"]] = ""
        disable = True
    else:
        disable = False
        recommendations_df = get_recommendation()

    if request.method =='POST':
        if request.form.getlist('delete_checkbox'):
            for id in request.form.getlist('delete_checkbox'):
                # Delete the games that were checked and commit in the database
                #USERGAMESPLAYED.query.filter_by(NM_GAME=id, ID_USER=current_user.id).delete(synchronize_session='fetch')

                cursor = conn.cursor()
                cursor.execute('DELETE FROM "USERGAMESPLAYED" WHERE "NM_GAME" = %s AND "ID_USER" = %s', (id, current_user.id))
                conn.commit()
                #db.session.commit()
            recommendations_df = get_recommendation()
```

```
            profile = db.session.query(USERGAMESPLAYED.ID_USER,
USERGAMESPLAYED.NM_GAME, V_GAMES.DT_YEAROFRELEASE, V_GAMES.NM_GENRE,
V_GAMES.NR_CRITICSCORE).select_from(USERGAMESPLAYED).join(V_GAMES,
V_GAMES.ID_GAME ==
USERGAMESPLAYED.ID_GAME).filter(USERGAMESPLAYED.ID_USER==current_user.id).paginate(pe
r_page=len(qtd_rows), page=page_num, error_out=True)
            flash('Games have been successfully deleted from your profile.')
            return render_template('profile.html', name=current_user.name, profile=profile,
first_name=first_name, last_name=last_name, len = len(recommendations_df),
recommendations_profile=recommendations_df, disable=disable)

    try:
                    user_id = re.search("₩"steamid₩":₩"(₩d+)₩"", line).group(1)
                    if user_id != None:
                        user_ids.append(user_id)
                        break
                except:
                    continue

def get_online_users(member_list_no, user_ids):
    url = 'https://steamcommunity.com/games/steam/members?p=' + str(member_list_no)
    resp = requests.get(url)
    soup = bs(resp.text, 'html.parser')
    all_users = soup.find_all("div", ₩
                                onclick =
re.compile("top₩.location₩.href='https:₩/₩/steamcommunity₩.com₩/id₩/(₩w+)'"), ₩
                                class_ = re.compile("online|in-game"))

    for user in all_users:
        user_profile = user.div.div.div.a['href'].encode("ascii")
        # print user_profile
        get_user_id(user_profile, user_ids)

def dump_user_id(user_ids, user_out_file):
    with open(user_out_file, 'w') as f:
        for idx in range(0, len(user_ids)):
            user_id_idx = {'user_idx': idx, 'user_id': user_ids[idx]}
            json.dump(user_id_idx, f)
            f.write('₩n')

def process_json_obj(resp, user_out_file, user_id):
    if 'user_summary' in user_out_file:
```

```
            try:
                obj = resp.json()['response']['players'][0]
            except:
                obj = {'steamid' : user_id}
        elif 'user_owned_games' in user_out_file:
            obj = resp.json()['response']
            obj = {'steamid' : user_id, 'game_count' : obj['game_count'], 'games' : obj['games']}
        elif 'user_friend_list' in user_out_file:
            obj = resp.json()['friendslist']
            obj = {'steamid' : user_id, 'friends' : obj['friends']}
        elif 'user_recently_played_games' in user_out_file:
            obj = resp.json()['response']
            try:
                obj = {'steamid' : user_id, 'total_count' : obj['total_count'], 'games' :
obj['games']}
            except:
                # corner case: total_count is zero
                obj = {'steamid' : user_id, 'total_count' : obj['total_count'], 'games' : []}
        return obj

def dump_user_info(url, user_ids, user_out_file):
        with open(user_out_file, 'w') as f:
            for user_id in user_ids:
                url_temp = url + str(user_id)
                resp = requests.get(url_temp)
                # resp = requests.head(url_temp)
                obj = process_json_obj(resp, user_out_file, user_id)
                json.dump(obj, f)
                f.write('₩n')


if __name__ == '__main__':
        member_list_page_no = 5
        key = '0D37BB1ABB4C0AFAF76379F37EDB4C8D'

        user_ids = []
        for idx in range(1, member_list_page_no + 1):
            print "Member List " + str(idx)
            get_online_users(idx, user_ids)
            print "Total online users found:"
            print len(user_ids)
```

```
    # dump user id
    dump_user_id(user_ids, 'data/user_idx_sample.json')



    # dump player summaries
    url = 'http://api.steampowered.com/ISteamUser/GetPlayerSummaries/v0002/?key=' + key
+ '&steamids='
    dump_user_info(url, user_ids, 'data/user_summary_sample.json')

    # dump owned games
    url = 'http://api.steampowered.com/IPlayerService/GetOwnedGames/v0001/?key=' + key
+ '&steamid='
    dump_user_info(url, user_ids, 'data/user_owned_games_sample.json')

    # dump friendList
    url = 'http://api.steampowered.com/ISteamUser/GetFriendList/v0001/?key=' + key +
'&steamid='
    dump_user_info(url, user_ids, 'data/user_friend_list_sample.json')

    # dump recently Played games
    url = 'http://api.steampowered.com/IPlayerService/GetRecentlyPlayedGames/v0001/?key='
+ key + '&steamid='
    dump_user_info(url, user_ids, 'data/user_recently_played_games_sample.json')
```

auth.py

```
from flask import Blueprint, render_template, redirect, url_for, request, flash
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import login_user, logout_user, login_required
from models import User
from __init__ import db
from games import save_ml_models
import datetime

auth = Blueprint('auth', __name__)

@auth.route('/login')
def login():
    return render_template('login.html')

@auth.route('/login', methods=['POST'])
def login_post():
```

```
    email = request.form.get('email')
    password = request.form.get('password')
    remember = True if request.form.get('remember') else False

    user = User.query.filter_by(email=email).first()

    # check if user actually exists
    # take the user supplied password, hash it, and compare it to the hashed password in database
    if not user or not check_password_hash(user.password, password):
        flash('Please check your login details and try again.')
        return redirect(url_for('auth.login')) # if user doesn't exist or password is wrong, reload the page

    # if the above check passes, then we know the user has the right credentials
    login_user(user, remember=remember)
    return redirect(url_for('main.index'))

@auth.route('/signup')
def signup():
    return render_template('signup.html')

@auth.route('/signup', methods=['POST'])
def signup_post():

    email = request.form.get('email')
    name = request.form.get('name')
    password = request.form.get('password')
    sysdate = datetime.datetime.now()

    user = User.query.filter_by(email=email).first() # if this returns a user, then the email already exists in database

    if user: # if a user is found, we want to redirect back to signup page so user can try again
        flash('Email address already exists')
        return redirect(url_for('auth.signup'))

    # create new user with the form data. Hash the password so plaintext version isn't saved.
    new_user = User(email=email, name=name, password=generate_password_hash(password, method='sha256'), created=sysdate)
```

```
    # add the new user to the database
    db.session.add(new_user)
    db.session.commit()

    return redirect(url_for('auth.login'))

@auth.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('main.index'))
```

models.py

```
import pandas as pd
import sqlite3 as sql
import os
import psycopg2
from flask_login import UserMixin, current_user
from __init__ import db
from sqlalchemy import create_engine

DATABASE_URL = os.environ['DATABASE_URL']
conn = psycopg2.connect(DATABASE_URL, sslmode='require')

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True) # primary keys are required by SQLAlchemy
    email = db.Column(db.String(100), unique=True)
    password = db.Column(db.String(100))
    name = db.Column(db.String(1000))
    created = db.Column(db.Date)

class USERGAMESPLAYED(db.Model):
    ID_PLAYED = db.Column(db.Integer, primary_key=True)
    ID_USER = db.Column(db.Integer)
    ID_GAME = db.Column(db.Integer)
    NM_GAME = db.Column(db.String(100))
    IC_PLAYED = db.Column(db.String(20))
    DT_PLAYED = db.Column(db.Date)

class V_GAMES(db.Model):
```

```
    ID_GAME = db.Column(db.Integer)
    NM_GAME = db.Column(db.String(100), primary_key=True)
    NM_GENRE = db.Column(db.String(100))
    NR_CRITICSCORE = db.Column(db.Integer)
    DT_YEAROFRELEASE = db.Column(db.String(100))

try:
    qtd_rows = pd.read_sql_query('SELECT * FROM "V_GAMES"', conn)
except:
    qtd_rows = 0

def check_gamesplayed():
    DATABASE_URL = os.environ['DATABASE_URL']
    conn = psycopg2.connect(DATABASE_URL, sslmode='require')
    df_checkgamesplayed = pd.read_sql_query('SELECT * FROM "USERGAMESPLAYED"', conn)
    df_checkgamesplayed.columns = map(lambda x: str(x).upper(),
df_checkgamesplayed.columns)
    df_checkgamesplayed.loc[df_checkgamesplayed['ID_USER'] == current_user.id]
    conn.close()
    return df_checkgamesplayed

def gamesunplayed():

    userid = current_user.id
    params = (str(userid), str(userid), str(userid))

    DATABASE_URL = os.environ['DATABASE_URL']
    conn = psycopg2.connect(DATABASE_URL, sslmode='require')

    # Create dataframe
    df_gamesunplayed = pd.read_sql_query('''select
        x.*,
        nm_game || '-' || nm_publisher || '-' || nm_genre as IMPORTANT_FEATURES
        from(
        select distinct
        coalesce(f."ID_USER",%s) as id_user,
        b.id_game AS ID_GAME,
        b.nm_game AS NM_GAME,
        max(e.nm_publisher) as NM_PUBLISHER,
        max(c.nm_genre) as NM_GENRE,
        max(1) as qt_games,
        max(a.nr_criticscore) as NR_CRITICSCORE,
```

```
            max(a.nr_userscore) as NR_USERSCORE,
            min(d.dt_year) as DT_YEAROFRELEASE,
            max(case
            when f."IC_PLAYED" = 'NO'  then 0
            when f."IC_PLAYED" = 'YES' then 1
            else 0
            end) IC_PLAYED
            from f_gamesbyplatform a
            left join d_games b
            on b.id_game = a.id_game
            left join d_genre c
            on c.id_genre = a.id_genre
            left join d_date d
            on d.id_date = a.id_date
            left join d_publisher e
            on e.id_publisher = a.id_publisher
            left join "USERGAMESPLAYED" AS f
            on f."ID_GAME" = a.id_game
            and f."ID_USER" = %s
            where b.linsource <> 'CARGA MANUAL'
            and d.dt_year > 0
            group by b.id_game, b.nm_game, coalesce(f."ID_USER",%s)) x
            where ic_played = 0''', conn, params = params)

    #print(df_gamesunplayed)

    conn.close()

    return df_gamesunplayed

def gamesplayed():

    userid = current_user.id
    params = (str(userid), str(userid), str(userid))

    DATABASE_URL = os.environ['DATABASE_URL']
    conn = psycopg2.connect(DATABASE_URL, sslmode='require')

    # Create dataframe
    df_gamesplayed = pd.read_sql_query('''select
        x.*,
        nm_game || '-' || nm_publisher || '-' || nm_genre as IMPORTANT_FEATURES
```

```
        from(
        select distinct
        coalesce(f."ID_USER",%s) as id_user,
        b.id_game AS ID_GAME,
        b.nm_game AS NM_GAME,
        max(e.nm_publisher) as NM_PUBLISHER,
        max(c.nm_genre) as NM_GENRE,
        max(1) as qt_games,
        max(a.nr_criticscore) as NR_CRITICSCORE,
        max(a.nr_userscore) as NR_USERSCORE,
        min(d.dt_year) as DT_YEAROFRELEASE,
        max(case
        when f."IC_PLAYED" = 'NO'  then 0
        when f."IC_PLAYED" = 'YES' then 1
        else 0
        end) IC_PLAYED
        from f_gamesbyplatform a
        left join d_games b
        on b.id_game = a.id_game
        left join d_genre c
        on c.id_genre = a.id_genre
        left join d_date d
        on d.id_date = a.id_date
        left join d_publisher e
        on e.id_publisher = a.id_publisher
        left join "USERGAMESPLAYED" AS f
        on f."ID_GAME" = a.id_game
        and f."ID_USER" = %s
        where b.linsource <> 'CARGA MANUAL'
        and d.dt_year > 0
        group by b.id_game, b.nm_game, coalesce(f."ID_USER",%s)) x
    WHERE ic_played IS NOT NUll''', conn, params = params)

    #print(df_gamesplayed)
    conn.close()

    return df_gamesplayed
```

games.py

```
from flask_login import current_user
from flask import Blueprint
from models import gamesplayed
```

```
from os.path import join
from statistics import median
import pandas as pd
import numpy as np
import os
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack, vstack
from lightgbm import LGBMClassifier

games = Blueprint('games', __name__)

def save_ml_models():

        df_gamesplayed = gamesplayed()
        df_gamesplayed.columns = map(lambda x: str(x).upper(), df_gamesplayed.columns)
        print(sorted(df_gamesplayed))



        features = pd.DataFrame(index=df_gamesplayed.index)

        y = df_gamesplayed["IC_PLAYED"].copy()
        features["NR_CRITICSCORE"] = df_gamesplayed["NR_CRITICSCORE"]
        features["DT_YEAROFRELEASE"] = df_gamesplayed["DT_YEAROFRELEASE"]

        mask_train = df_gamesplayed['DT_YEAROFRELEASE'] <
median(features["DT_YEAROFRELEASE"])
        mask_val = (df_gamesplayed['DT_YEAROFRELEASE'] >=
median(features["DT_YEAROFRELEASE"]))

        Xtrain, Xval = features[mask_train], features[mask_val]
        ytrain, yval = y[mask_train], y[mask_val]

        title_train = df_gamesplayed[mask_train]['IMPORTANT_FEATURES']
        title_val = df_gamesplayed[mask_val]['IMPORTANT_FEATURES']

        title_vec = TfidfVectorizer(min_df=4, ngram_range=(1,3))
        title_bow_train = title_vec.fit_transform(title_train)
        title_bow_val = title_vec.transform(title_val)

        from scipy.sparse import hstack, vstack
```

```
        Xtrain_wtitle = hstack([Xtrain, title_bow_train])
        Xval_wtitle = hstack([Xval, title_bow_val])

        mdl_lgbm = LGBMClassifier(random_state=0, class_weight="balanced", n_jobs=6)
        mdl_lgbm.fit(Xtrain_wtitle, ytrain)

        from scipy.sparse import hstack, vstack

        Xtrain_wtitle = hstack([Xtrain, title_bow_train])
        Xval_wtitle = hstack([Xval, title_bow_val])

        mdl_rf = RandomForestClassifier(n_estimators=1000, random_state=0,
min_samples_leaf=2, class_weight="balanced", n_jobs=6)
        mdl_rf.fit(Xtrain_wtitle, ytrain)

        return mdl_rf, mdl_lgbm, title_vec
```

utils.py

```
from flask import Blueprint, render_template, request, flash, send_file, send_from_directory
import pandas as pd
from models import gamesunplayed
from flask_login import current_user
from models import gamesplayed
from os.path import join
from statistics import median
import numpy as np
import os
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack, vstack
from lightgbm import LGBMClassifier

ml_utils = Blueprint('ml_utils', __name__)

def predict_api():

    df_gamesplayed = gamesplayed()
    df_gamesplayed.columns = map(lambda x: str(x).upper(), df_gamesplayed.columns)
    #print(sorted(df_gamesplayed))



    features = pd.DataFrame(index=df_gamesplayed.index)
```

```
    y = df_gamesplayed["IC_PLAYED"].copy()

    features["NR_CRITICSCORE"] = df_gamesplayed["NR_CRITICSCORE"]
    features["DT_YEAROFRELEASE"] = df_gamesplayed["DT_YEAROFRELEASE"]

    mask_train = df_gamesplayed['DT_YEAROFRELEASE'] < median(features["DT_YEAROFRELEASE"])
    mask_val = (df_gamesplayed['DT_YEAROFRELEASE'] >= median(features["DT_YEAROFRELEASE"]))

    Xtrain, Xval = features[mask_train], features[mask_val]
    ytrain, yval = y[mask_train], y[mask_val]

    title_train = df_gamesplayed[mask_train]['IMPORTANT_FEATURES']
    title_val = df_gamesplayed[mask_val]['IMPORTANT_FEATURES']

    title_vec = TfidfVectorizer(min_df=4, ngram_range=(1,3))
    title_bow_train = title_vec.fit_transform(title_train)
    title_bow_val = title_vec.transform(title_val)

    from scipy.sparse import hstack, vstack

    Xtrain_wtitle = hstack([Xtrain, title_bow_train])
    Xval_wtitle = hstack([Xval, title_bow_val])

    mdl_lgbm = LGBMClassifier(random_state=0, class_weight="balanced", n_jobs=6)
    mdl_lgbm.fit(Xtrain_wtitle, ytrain)

    from scipy.sparse import hstack, vstack

    Xtrain_wtitle = hstack([Xtrain, title_bow_train])
    Xval_wtitle = hstack([Xval, title_bow_val])


    mdl_rf = RandomForestClassifier(n_estimators=1000, random_state=0, min_samples_leaf=2, class_weight="balanced", n_jobs=6)
    mdl_rf.fit(Xtrain_wtitle, ytrain)

    df_gamesunplayed = gamesunplayed()
    df_gamesunplayed.columns = map(lambda x: str(x).upper(), df_gamesunplayed.columns)
    #print("df_gamesplayed columns:" + str(sorted(df_gamesunplayed)))
```

```
    title = df_gamesunplayed['IMPORTANT_FEATURES']

    features = pd.DataFrame(index=df_gamesunplayed.index)
    features['NR_CRITICSCORE'] = df_gamesunplayed["NR_CRITICSCORE"]
    features['DT_YEAROFRELEASE'] = df_gamesunplayed["DT_YEAROFRELEASE"]

    vectorized_title = title_vec.transform(title)
    feature_array = hstack([features, vectorized_title])

    try:
        p_rf = mdl_rf.predict_proba(feature_array)[:, 1]
    except:
        p_rf = [0] * feature_array.shape[0]

    try:
        p_lgbm = mdl_lgbm.predict_proba(feature_array)[:, 1]
    except:
        p_lgbm = [0] * feature_array.shape[0]

    try:
        one = 0.5*p_rf
    except:
        one = 0.5

    try:
        two = 0.5*p_lgbm
    except:
        two = 0.5

    p = one + two

    return p
```

webcrawler.py

```
import sys
import requests
import json

from bs4 import BeautifulSoup as bs
from contextlib import closing
import urllib
```

```
import re


def get_user_id(user_profile, user_ids):
    url = user_profile
    with closing(urllib.urlopen(url)) as page:
        for line in page:
            if "steamid" in line:
                try:
                    user_id = re.search("₩"steamid₩":₩"(₩d+)₩"", line).group(1)
                    # print user_id + ' ' + user_profile
                    if user_id != None:
                        user_ids.append(user_id)
                        break
                except:
                    continue

def get_online_users(member_list_no, user_ids):
    url = 'https://steamcommunity.com/games/steam/members?p=' + str(member_list_no)
    resp = requests.get(url)
    soup = bs(resp.text, 'html.parser')
    # print(soup.prettify())
    # search profile of users who are online/in-game
    all_users = soup.find_all("div", ₩
                                    onclick                                                                    =
re.compile("top₩.location₩.href='https:₩/₩/steamcommunity₩.com₩/id₩/(₩w+)'"), ₩
                                    class_ = re.compile("online|in-game"))
    for user in all_users:
        user_profile = user.div.div.div.a['href'].encode("ascii")
        # print user_profile
        get_user_id(user_profile, user_ids)

def dump_user_id(user_ids, user_out_file):
    with open(user_out_file, 'w') as f:
        for idx in range(0, len(user_ids)):
            user_id_idx = {'user_idx': idx, 'user_id': user_ids[idx]}
            json.dump(user_id_idx, f)
            f.write('₩n')

# dump player summaries
def process_json_obj(resp, user_out_file, user_id):
        # corner case: list index out of range
```

```
            try:
                obj = resp.json()['response']['players'][0]
            except:
                obj = {'steamid' : user_id}
        elif 'user_owned_games' in user_out_file:
            obj = resp.json()['response']
            obj = {'steamid' : user_id, 'game_count' : obj['game_count'], 'games' : obj['games']}
        elif 'user_friend_list' in user_out_file:
            obj = resp.json()['friendslist']
            obj = {'steamid' : user_id, 'friends' : obj['friends']}
        elif 'user_recently_played_games' in user_out_file:
            obj = resp.json()['response']
            try:
                obj = {'steamid' : user_id, 'total_count' : obj['total_count'], 'games' :
obj['games']}
            except:
                # corner case: total_count is zero
                obj = {'steamid' : user_id, 'total_count' : obj['total_count'], 'games' : []}
        return obj

def dump_user_info(url, user_ids, user_out_file):
        with open(user_out_file, 'w') as f:
            for user_id in user_ids:
                url_temp = url + str(user_id)
                resp = requests.get(url_temp)
                # resp = requests.head(url_temp)
                obj = process_json_obj(resp, user_out_file, user_id)
                json.dump(obj, f)
                f.write('₩n')


if __name__ == '__main__':
        member_list_page_no = 5
        key = '0D37BB1ABB4C0AFAF76379F37EDB4C8D'

        user_ids = []
        for idx in range(1, member_list_page_no + 1):
            print "Member List " + str(idx)
            get_online_users(idx, user_ids)
            print "Total online users found:"
            print len(user_ids)
```

```
    # dump user id
    dump_user_id(user_ids, 'data/user_idx_sample.json')



    # dump player summaries
    url = 'http://api.steampowered.com/ISteamUser/GetPlayerSummaries/v0002/?key=' + key
+ '&steamids='
    dump_user_info(url, user_ids, 'data/user_summary_sample.json')

    # dump owned games
    url = 'http://api.steampowered.com/IPlayerService/GetOwnedGames/v0001/?key=' + key
+ '&steamid='
    dump_user_info(url, user_ids, 'data/user_owned_games_sample.json')

    # dump friendList
    url = 'http://api.steampowered.com/ISteamUser/GetFriendList/v0001/?key=' + key +
'&steamid='
    dump_user_info(url, user_ids, 'data/user_friend_list_sample.json')

    # dump recently Played games
    url = 'http://api.steampowered.com/IPlayerService/GetRecentlyPlayedGames/v0001/?key='
+ key + '&steamid='
    dump_user_info(url, user_ids, 'data/user_recently_played_games_sample.json')
```

gamedetali.py

```
import sys
import requests
import json

from bs4 import BeautifulSoup as bs
from contextlib import closing
import urllib
import re
import time



def get_app_id_list():
    url                                                                                =
'https://steamcommunity.com/linkfilter/https://api.steampowered.com/ISteamApps/GetAppList/v
2/'
    header = {
        'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36
```

```
(KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36',
            'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
            'Accept-Encoding':'gzip, deflate, sdch',
            'Accept-Language':'en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4,zh-TW;q=0.2'}
      resp = requests.get(url,header)

      app_id_objs = resp.json()['applist']['apps']
      app_id_list = []

      for app in app_id_objs:
            app_id_list.append(app['appid'])

      return app_id_list

def get_game_detail(app_id_list, num, game_detail_out_file):
      url = 'https://store.steampowered.com/api/appdetails?appids='
      header = {
            'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36',
            'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
            'Accept-Encoding':'gzip, deflate, sdch',
            'Accept-Language':'en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4,zh-TW;q=0.2'}
      with open(game_detail_out_file, 'w') as f:
            for idx in xrange(num):
                  url_temp = url + str(app_id_list[idx])
                  time.sleep(.100) # sleep 100ms
                  resp = requests.get(url_temp, header)

                  obj = resp.json()
                  for key in obj:
                        if obj[key]["success"] is True :
                              json.dump(obj[key]["data"], f)
                              f.write('₩n')

if __name__ == '__main__':
      app_id_list = get_app_id_list()
      print "total apps: " + str(len(app_id_list))

      get_game_detail(app_id_list,10,"data/game_detail.json")
```

alg.py

```
import os
```

```
import csv
import sqlite3

BASEDIR = "C:/Users/82103/Desktop/추천/"
CSVFilename = "GamesCSV.csv"
DBFilename = "Games.db"

con = sqlite3.connect(BASEDIR + DBFilename)
cur = con.cursor()

csvfile = open(BASEDIR + CSVFilename)
rows = csv.reader(csvfile)

for line in rows:
    cur.execute("SELECT * FROM Games WHERE GameName='" + line[0] + "'")
    result = cur.fetchall()

    if(len(result) == 0):
        cur.execute("INSERT INTO Games VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", line)
    else:
        cur.execute("DELETE FROM Games WHERE GameName='" + line[0] + "'")
        cur.execute("INSERT INTO Games VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", line)

gamename = input("Input 2 Game : ").split()
cur.execute("SELECT * FROM Games WHERE GameName='" + gamename[0] + "'")
result1 = list((cur.fetchall())[0])
cur.execute("SELECT * FROM Games WHERE GameName='" + gamename[1] + "'")
result2 = list((cur.fetchall())[0])

result1 = list(filter(None, result1))
result2 = list(filter(None, result2))

print(result1)
print(result2)

common = list(set(result1) & set(result2))
print(common)

cur.execute("SELECT * FROM Games")
result = cur.fetchall()

recommend = []
```

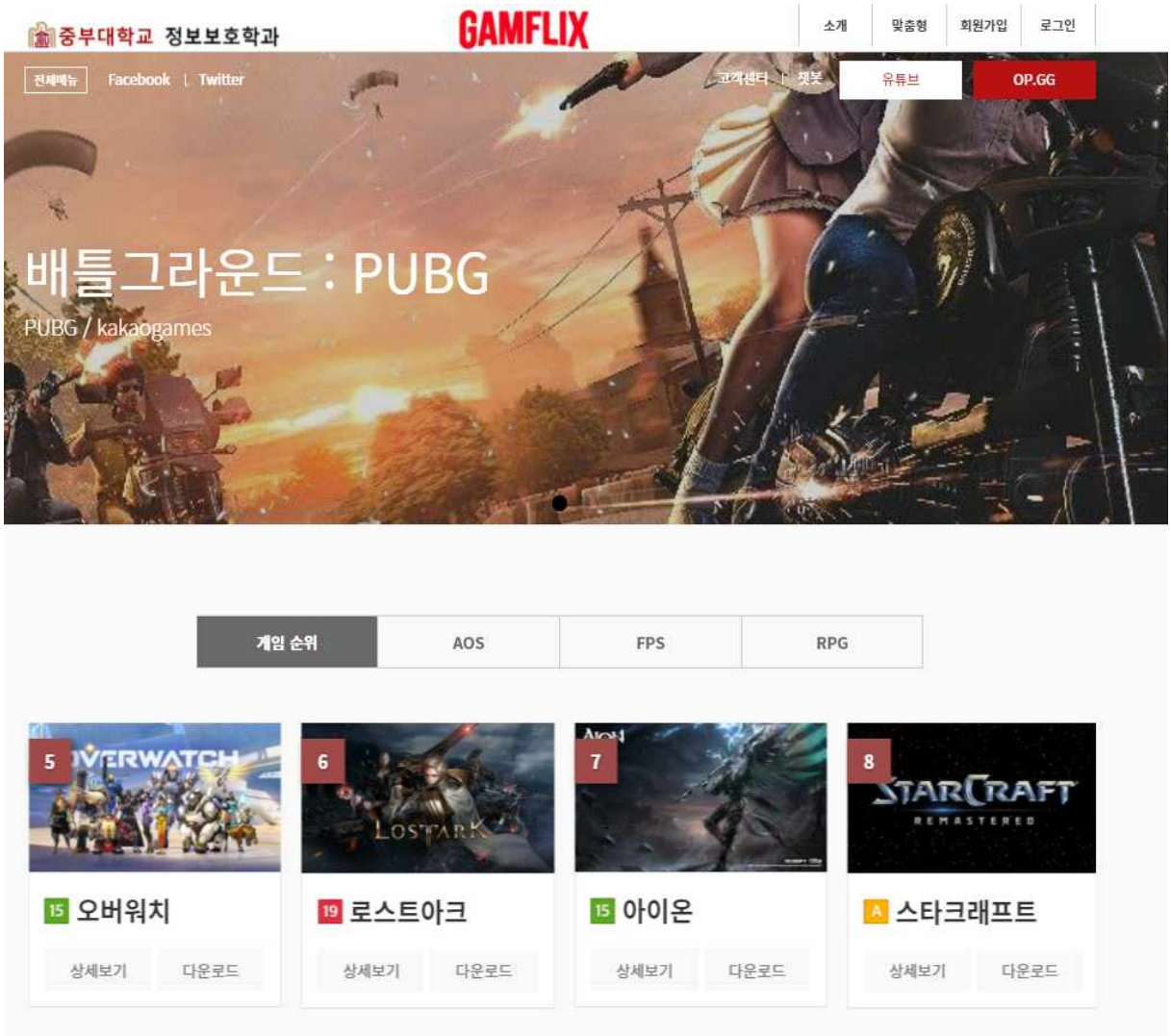
```
for game in result:
    count = 0
    for atr in common:
        if(atr in game):
            count = count + 1
    if(count >= 4):
        recommend.append(game)


for game in recommend:
    print(game[0])

con.commit()
con.close()
```

5.2 웹사이트

5.2.1 메인화면



새로운 게임

붉은사막

2021년 공식 예정

★★★½☆ 펄어비스

플레이스테이션4, 윈도우, 엑스박스

액션 롤플레잉, 미선정 게임플레이

펄어비스에서 개발중인 오픈월드 액션 어드벤처 게임 / 근접과 PC로 출시 예정

2021년 4월 25일 | 오전 0:00

주소 또는 도로명을 입력해주세요.

찾는 곳 확인하기 | 예약하기

공지사항

전체 공지 | 게임 공지

전체 2021.04.15
[공지사항] 안내말씀

롤 2021.04.15
[신규 챔피언] 신규 챔피언 등록

배틀그라운드 2021.04.15
[업데이트] 3.21 패치

오버워치 2021.04.15
[공지] 2021년 4월 미션 이벤트

자유게시판

안녕하세요
안녕하세요

이번 패치 정리
이번 패치 정리

공지사항에 관하여
공지사항에 관하여

고객센터

1:1 문의

게임추가요청

FAQ
자주묻는 질문

광고 문의

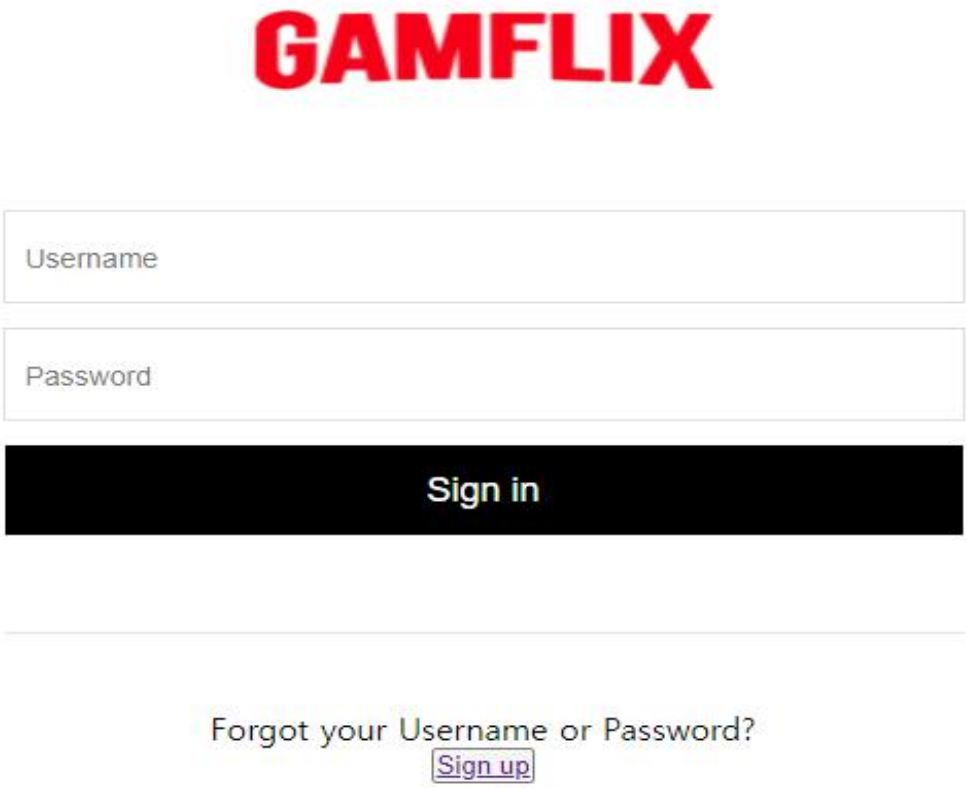
GAMFLIX

경기도 고양시 덕양구 대자동 동헌로 305
대표자 송무대학교 정보보호학과 윤다예 / 오예지

Copyright 2021 by Joongbu. All right reserved

GAMFLIX

5.2.2 로그인 및 회원가입 화면



GAMFLIX

아이디

비밀번호

비밀번호 재확인

이름

생년월일

년(4자) 월 일

성별

성별

이메일

선택입력

휴대전화

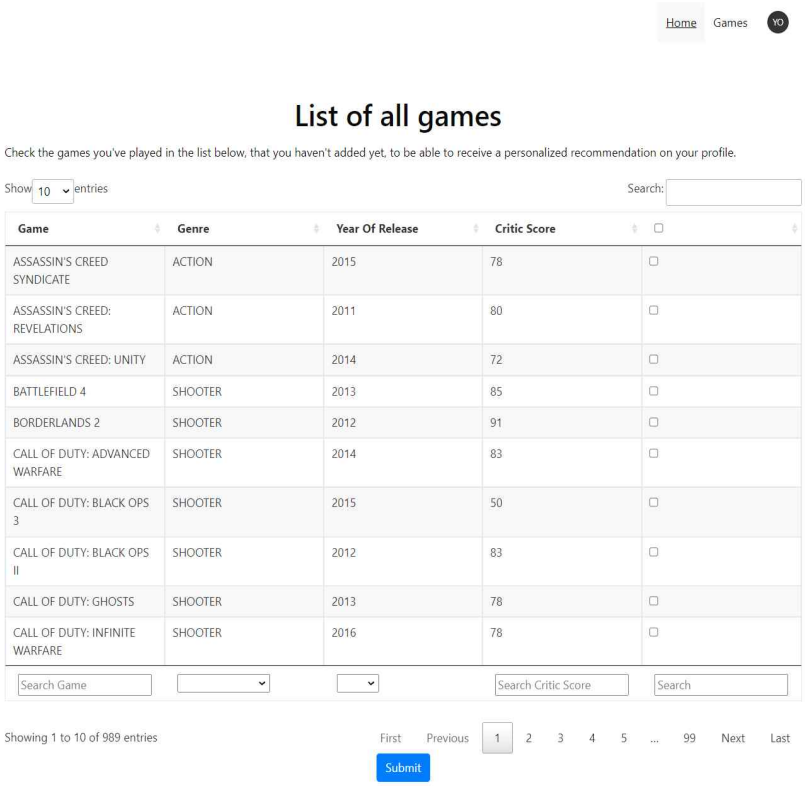
전화번호 입력

가입하기

5.2.3 사이트 및 개발자 소개 화면

5.2.4 선호 콘텐츠 선택 화면

5.2.5 맞춤형 콘텐츠 결과 화면



5.3 발표자료

웹크롤링과 인공지능을 활용한 개인화 추천 반응형 웹사이트 개발

2021.11

지도교수 : 양환석 교수님

TEAM : GAMFILX
윤다예, 오예지

조원편성

이름	역할
윤다예	프론트엔드 구축 및 개발, 이메일 인증 시스템 구현, 콘텐츠 기반 추천 웹크롤링 개발, PPT 작성, 보고서 작성
오예지	API 및 DB 설계, 백엔드 서버 환경 구축, 챗봇 개발, 협업 기반 추천 웹크롤링 개발, PPT 작성, 보고서 작성, 발표

주제선정(1/2)

■ 빅데이터를 활용

'주간 뉴스 키워드가 한눈에'…통계청, 통계 자동 검색

특구재단, 기업의 AI 도입 확산 위해 맞춤형 기업 지원 실시

현장에 바로 적용할 수 있는 '진단-교육-파일럿 제작' 맞춤형 지원

있는 기능을 제공한다.

여러 분야에서 빅데이터를 활용한 검색 기반 통계 시스템인 웹크롤링 사용

주제선정(2/2)

■ 많은 분야에서 개인화 데이터 활용

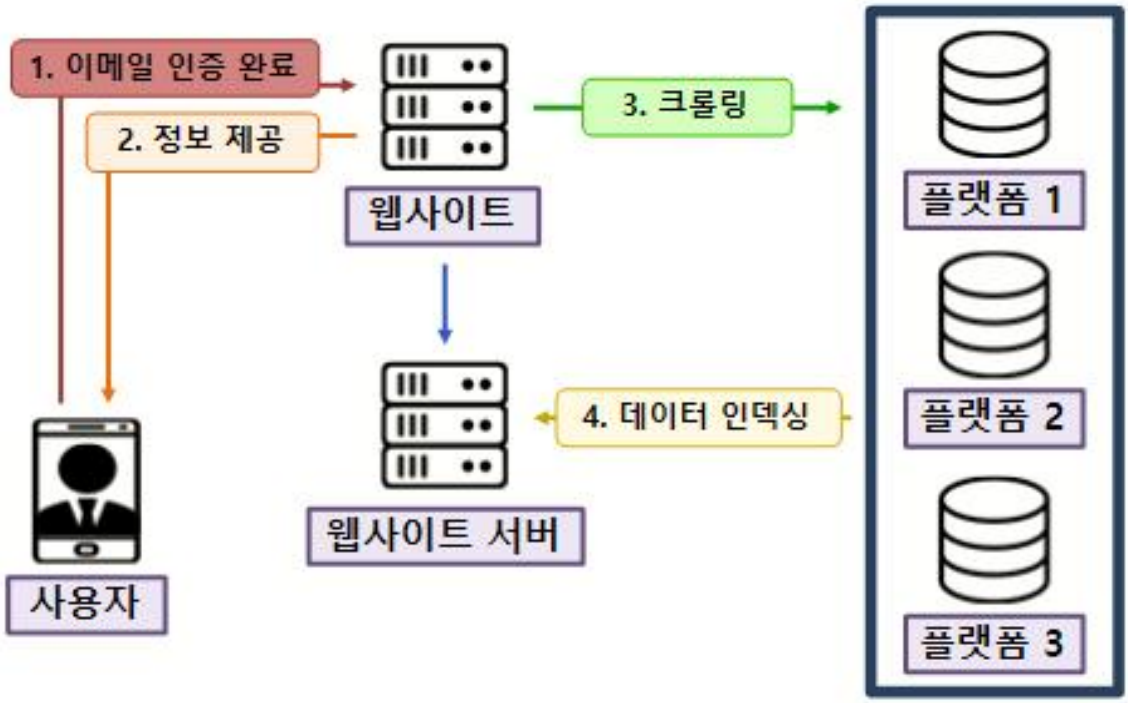
개인화 데이터 활용으로 고객 로열티 프로그램에 변화 추구하는 항공업계

데이터와 인공지능에 의한 초(超) 개인화 시대

초개인화 시대의 전망과 우리가 알아야 할 것들

웹크롤링을 이용하여 데이터를 수집한 후
수집한 데이터로 맞춤형 서비스 제공 웹사이트 개발 추진

구상도



추진경과

	3월	4월	5월	6월	7월	8월	9월	10월
자료조사 및 분석								
프론트엔드 개발								
연동								
백엔드 개발 및 DB 구축								
홈페이지 검토 및 수정								

개발환경

WEB Framework

Flask

Software

SQLAlchemy

Development Language

HTML CSS JS

개발내용(1/12)

■ 프론트엔드 개발

개발내용(2/12)

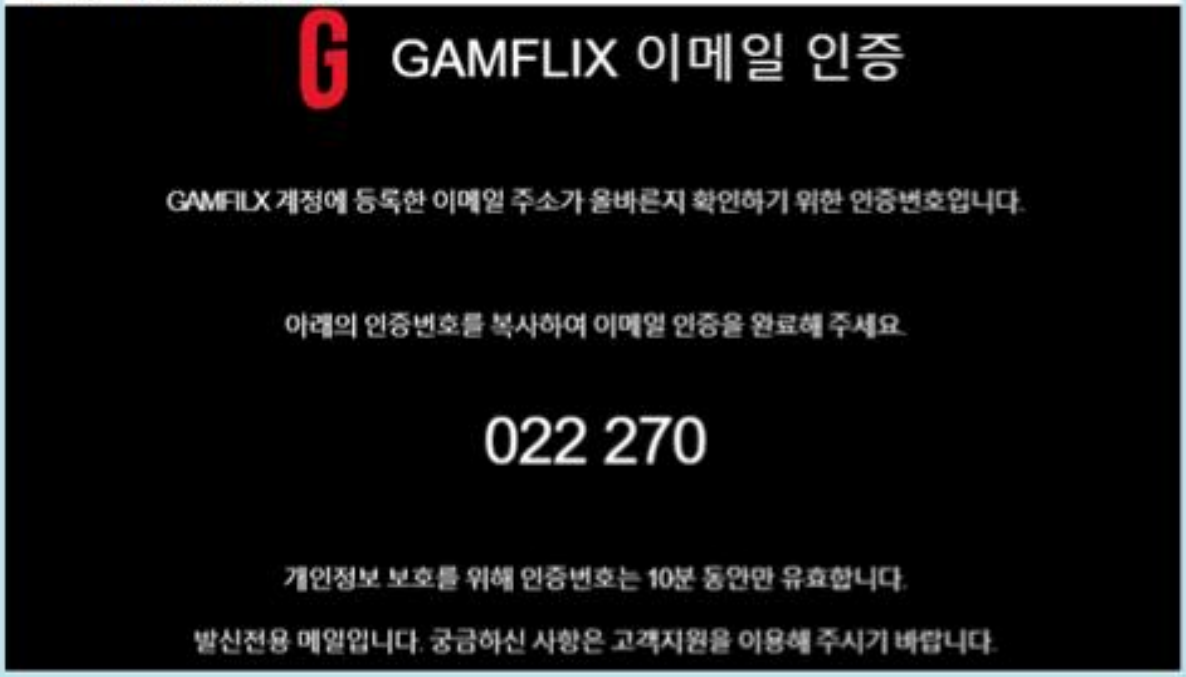
■ 프론트엔드 개발

개발내용(3/12)

■ 프론트엔드 개발

개발내용(4/12)

■ 이메일인증 구현



개발내용(5/12)

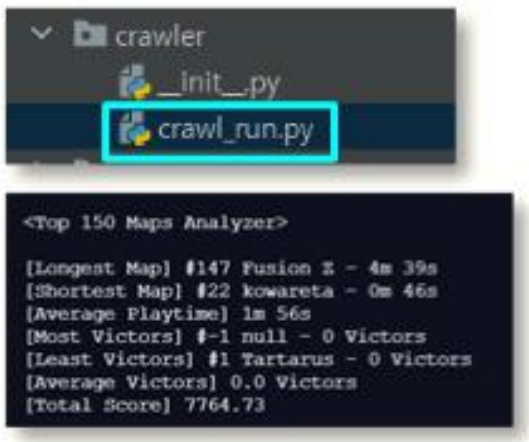
■ 콘텐츠 기반 추천

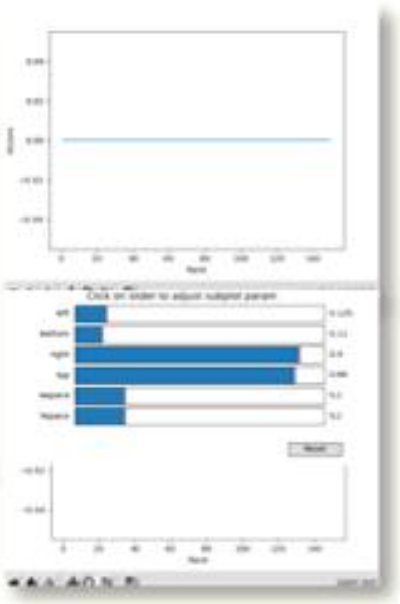
개발내용(6/12)

■ 콘텐츠 기반 추천

개발내용(7/12)

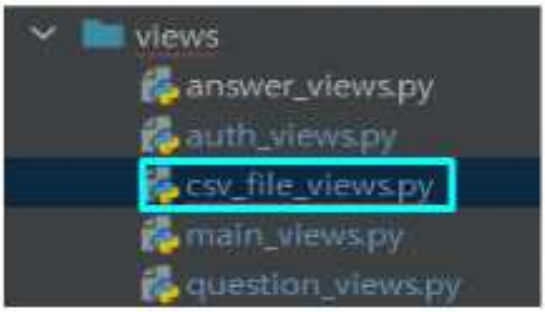
■ 협업 기반 추천





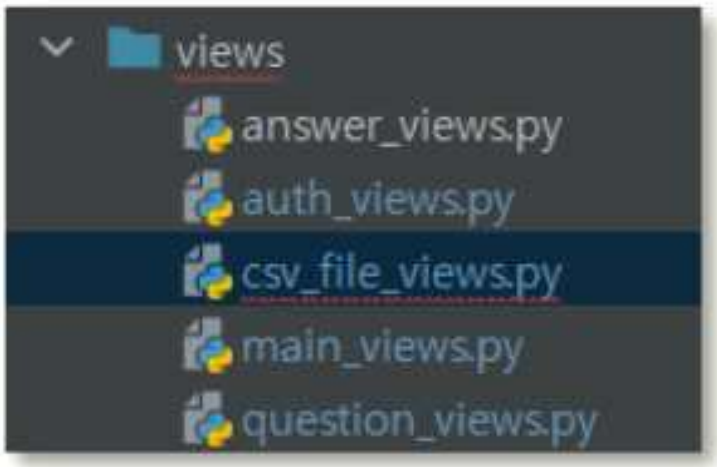
개발내용(8/12)

- Crawler as responsive HTML



개발내용(9/12)

- VIEWS



개발내용(10/12)

- question_view

```python
from datetime import datetime

from flask import Blueprint, url_for, request, render_template
from werkzeug.utils import redirect

from .. import db
from ..forms import AnswerForm
from ..models import Question, Answer

bp = Blueprint('answer', __name__, url_prefix='/answer')


@bp.route('/create/<int:question_id>', methods=('POST',))
def create(question_id):
    form = AnswerForm()
```

개발내용(11/12)

- Chatbot

개발내용(12/12)

- recommand system

개발화면

- 동영상 시연

결론

- 웹크롤링과 인공지능을 활용한 맞춤형 웹사이트를 제작하고 사이트가 가지고 있는 자료를 보안함
- 사용자들의 편리하고 안전하게 웹사이트를 이용할 수 있도록 도움
- 개인 맞춤형 사이트로써 사용자들의 정보를 안전하게 수집 및 분석하여 원하는 콘텐츠를 선별하고 추천하여 개인에게 맞는 최적의 맞춤형 서비스 제공

기대효과

- 웹크롤링과 인공지능을 활용한 맞춤형 웹사이트 제작이라는 목표를 달성하기 위해 협력하는 과정에서 팀워크의 중요성을 깨달음
- 개인 기술 개발 역량을 발전 시킬 수 있는 계기가 됨
- 웹사이트의 데이터 인증 및 보안을 통해 사용자들이 안전하게 사용할 수 있는 사이트를 제작함으로써 정보보호의 중요성을 인지함