Unity Backend

뒤끝 팁과 최신 기능

Created in 2023-08-09

Last Updated 2023-08-09

Unity Version 2022.2.2f1

Index

- 뒤끝 팁
- 새로운 기능

뒤끝 팁

- 뒤끝 요금 최적화 팁
- DB 이용량 가이드
- DB 이용이 발생하는 기능
- DB 최적화 가이드

뒤끝 팁

뒤끝 요금 최적화 팁

공통

- 게임 점검, 다른 기기 로그인 체크는 핸들러 이용
  - 뒤끝에서는 주기적으로 체크가 필요한 에러에 대해서는 핸들러를 제공
  - 해당 에러를 체크하기 위해 뒤끝 함수를 1초 등의 단위로 반복 호출해 체크하는 것은 지양

	Backend.Utils.GetServerStatus() 함수를 1초마다 호출해 프로젝트 상태 값 체크
	Backend.ErrorHandler.OnMaintenanceError 핸들러에서 응답 시 점검 관련 UI 출력

- 뒤끝 함수 에러 발생 시, 자동 재호출자제
  - 뒤끝 에러 로직 처리 중 예외 혹은 특정하지 않은 에러가 발생하였을 때 성공할 때까지 재호출을 하는 로직일 경우, 서버 점검이나 액세스토큰 소멸과 같은 케이스에서는 영구적으로 에러가 발생되며, 기타 항목의 호출 비용이 발생

	if 문으로 개발자 문서에 기재된 문서에 따라 에러 처리 후, 나머지 에러를 위한 else 문에서 반복처리
	if 문으로 개발자 문서에 기재된 문서에 따라 에러 처리 후, 나머지 에러를 위한 else 문에서는 UI를 통해 에러 표시 후 게임 진행 중단

뒤끝 팁

- 차트, 내 게임 정보 등 고정된 데이터는 최초 1회만 호출
  - 차트 불러오기(Backend.Chart.GetChartContents)나
    내 게임 정보 불러오기(Backend.GameData.GetMyData)와 같이
    변경될 일이 적은 데이터는 데이터의 크기로 인해
    DB 읽기량이 높게 발생할 수 있기 때문에 최소한으로 호출

	저장할 때마다 내 게임 정보를 불러와 클라이언트 데이터를 더한 후 저장
	클라이언트의 데이터를 바로 저장
	상점에 들어갈 때마다 상점 아이템 관련 차트 불러오기 함수 호출
	로그인 이후 최초 상점 아이템 관련 차트 불러오기 함수를 호출한 후 캐싱하여 사용

뒤끝 팁

- 게임정보관리
  - 데이터 갱신 방식은 Insert가 아닌 Update를 이용
    - 뒤끝의 한 테이블에는 유저 한 명당 하나의 데이터(row)를 가지는 것을 추천
    - 데이터 백업 및 운영을 위해 한 데이터를 덮어씌우는 형태(Update)가 아닌 계속해서 생성하는 로직(Insert)일 경우, 스토리지/백업 요금이 증가
    - 데이터 백업은 자동 데이터 삭제 기능이 존재하는 게임 로그 기능을 이용하는 것을 추천

	5분 주기로 Backend.GameData.Insert("Table", param)을 통해 데이터 새로 생성
	inDate 저장 후 Backend.GameData.UpdateV2("Table", param, inDate, Backend.UserInDate) 함수를 호출해 데이터 덮어씌우기

  - 데이터 저장 주기는 초 단위 보다는 분 단위로 설정
    - 뒤끝에서는 자동 저장 주기를 5~20분 정도로 추천
    - 주기는 게임의 운영 방식에 따라 알맞게 추천 주기로 적용

	10초마다 저장 코루틴 호출
	10분마다 저장 코루틴 호출 (호출이 꼭 필요한 것이 아니라면 저장 주기를 더 긴 시간으로 설정)

뒤끝 팁

- 데이터의 중요도에 따라 저장 로직 설정
  - 아이템을 얻거나, 적을 처치하거나, 스테이지를 넘어갈 때마다 저장을 하는 로직일 경우, 유저별 게임 속도에 따라 저장 호출이 잦아질 수 있음
  - 따라서 데이터 중요도에 따라 저장 방식을 다르게 설정
  - 우편을 수령할 경우, 해당 우편은 제거되기 때문에 즉시 저장하는 것을 추천
  - 인앱 결제를 통한 아이템 수령의 경우, 데이터 저장이 이루어지지 않을 경우 재결제를 해야 하기 때문에 즉시 저장하는 것을 추천
  - 인 게임 내 재화로 아이템 구매, 재화 사용 등은 데이터가 롤백 되어도 구매하기 전 상태로 돌아가며 다시 구매가 가능하므로, 모든 변경된 정보가 같이 저장되는 자동 저장 주기 때 저장 추천

	아이템 하나, 경험치를 얻거나 적을 처치할 때마다 저장
	아이템, 경험치, 적 처치는 캐싱 된 값으로 사용하다가 특정 주기마다 저장. 우편, 영수증 등 호출이 제한되어 있는 경우는 바로 저장.

뒤끝 팁

- 다수의 데이터를 저장 혹은 불러올 때는 트랜잭션 기능 이용
  - 한 테이블당 하나의 데이터 저장(Update), 조회(Get) 함수를 호출할 경우 호출 수가 많이 발생
  - 데이터를 한 번에 많이 저장하고자 할 경우에는 최대 10개까지 한 개의 호출로 데이터를 저장하는 트랜잭션 기능 이용 추천

	10분마다 저장하는 로직에서 테이블 개수만큼 Backend.GameData.Update 함수 호출.
	10분마다 저장하는 로직에서 테이블당 트랜잭션 리스트에 추가하여 해당 리스트를 Backend.GameData.TransactionWriteV2() 함수에 넣어 호출 1회로 저장

뒤끝 팁

- 랭킹
  - 랭킹 UI를 띄울 때마다 랭킹 불러오기 함수 호출 배제
    - 랭킹 UI를 띄울 때마다 랭킹 불러오기 함수를 호출하는 로직일 경우, 유저에 따라 랭킹 불러오기 함수 호출의 주기가 다르며 많은 호출을 불러올 수 있음
      랭킹을 보여줄 때마다 호출하는 로직보다는 캐싱하여 5~20분 주기로 갱신하는 로직을 사용해 호출 횟수를 줄이는 것을 추천

	랭킹 UI를 띄울 때마다 Backend.URank.User.GetRankList(), Backend.URank.User.UpdateUserScore() 호출하여 UI에 적용
	랭킹 UI를 띄울 때 Backend.URank.User.GetRankList(), Backend.URank.User.UpdateUserScore()를 호출한 시간 체크. 10분이 지나면 새로 호출하고 데이터를 캐싱한 후 캐싱된 데이터를 UI에 적용 10분이 지나지 않을 경우 캐싱된 데이터를 UI에 적용

뒤끝 팁

- 우편
  - 구버전 우편 함수(Backend.Social.Post.GetPostListV2) 사용 배제
    - 구버전 우편 관리 함수인 Backend.Social.Post.GetPostListV2의 경우
      호출할 때마다 최대 100개의 우편을 가져와 제외하는 형식의 로직이기 때문에
      우편이 많아질수록 DB의 읽기량이 증가할 것
    - 우편을 구현하고자 할 경우에는 신버전 우편 함수 사용 추천
      - 가이드 문서 : https://developer.thebackend.io/unity3d/guide/upost/info/
  - 우편 알림을 구현하기 위해 주기적으로 짧게 우편 불러오는 로직 배제
    - 뒤끝 콘솔에서 우편을 보내자마자 유저들에게 우편이 발송되었다는 알림을 띄우고자
      1~60초 사이의 텀을 두고 반복적으로 우편 불러오기 함수를 호출하는 경우가 있음
      뒤끝에서는 콘솔과 클라이언트 간의 실시간 알림을 지원하고 있지 않기 때문에
      구현을 할 경우에 초 단위가 아닌 20~30분 주기로 호출하는 것을 추천

	Backend.UPost.GetPostList(PostType.Admin, 10)를 1~10초 간격으로 호출
	Backend.UPost.GetPostList(PostType.Admin, 10)를 20분 간격으로 호출

뒤끝 팁

- 길드
  - 길드 UI 띄울 때마다 길드 함수, 길드 랭킹 함수 호출 자제
    - 길드 정보 관련 UI를 생성할 때마다
      내 길드 정보 불러오기(Backend.Guild.GetMyGuildInfoV3), 길드 랭킹 불러오기
      (Backend.URank.Guild.GetRankList)를 할 경우 요금이 크게 증가할 수 있음
    - 랭킹과 마찬가지로 불러온 값을 캐싱하여 일정 주기가 지났거나
      길드에서 탈퇴할 경우에 재호출하는 로직 추천

	길드 UI 생성 시 관련 길드 함수 호출
	길드 UI 생성 시 최근에 길드 랭킹 정보를 불러온 시간과 대조하고, 5분이 지나지 않았다면 캐싱된 값으로 UI 구성. **5분이 지났을 경우에는 재호출 및 캐싱, 그리고 캐싱된 값으로 UI 구성**

뒤끝 팁

- 게임 로그
  - 장시간 보존할 필요가 없는 데이터는 보관 기간 조정
    - 게임 로그 삽입 시 별도의 유예기간(graceDays)을 입력하지 않을 경우 90일로 자동 설정됨
    - 서버에는 90일 동안 존재하기에 스토리지 비용에 측정이 되며, 90일까지의 보관이 필요 없다고 판단될 경우에는 7, 15, 30일 등으로 조정하는 것을 추천

	Backend.GameLog.InsertLog("logType", param) graceDays 설정 없이 호출
	Backend.GameLog.InsertLog("logType", param, 15) graceDays 설정 후 호출

  - 일정 시간마다 자동저장을 이용하고 있는 경우, 스토리지 및 DB 쓰기 요금 주의
    - 게임 로그 관리의 경우 데이터 삽입만 제공하고 있기에 호출을 많이 할수록 스토리지 및 DB 쓰기 요금이 상승
    - 해당 부분을 고려하여 에러, 인앱 결제 등의 필요한 정보만 로그로 저장하는 것을 추천

뒤끝 팁

- 랜덤 조회
  - 구버전 랜덤 조회 함수 (Backend.Social.GetRandomUserInfo) 사용 배제
    - 구버전 랜덤 조회 함수인 Backend.Social.GetRandomUserInfo, Backend.Guild.GetRandomGuildInfoV3의 경우 DB 데이터에서 조건이 만족할 때까지 검색을 계속하기 때문에 데이터가 많아질수록 응답시간과 DB량이 높게 측정됨
    - 랜덤 조회를 구현하고자 할 경우에는 해당 문제점을 개선한 랜덤 조회 (Backend.RandomInfo.GetRandomData) 함수 사용 추천

	랜덤 조회 기능으로 Backend.Social.GetRandomUserInfo 함수 사용
	랜덤 조회 기능으로 Backend.RandomInfo.GetRandomData 함수 사용

뒤끝 팁

DB 이용량 가이드

- 뒤끝 DB는 "읽기", "쓰기" 두 가지 항목으로 구성

- 일반 읽기/쓰기 처리 기준
  - 읽기/쓰기 작업을 수행할 때는 아래 표를 기준으로 읽기 처리량을 산정함
  - 단, 사용하는 기능의 API 작동 방식 및 인덱스 사용량에 따라 발생하는 실제 처리량은 상이할 수 있음

  - 읽기 작업

데이터 크기	처리량
4KB 미만	0.5
4KB	1

예시

10KB를 불러올 때 사용하는 읽기 처리량은 1처리(4KB) + 1처리(4KB) + 0.5처리(2KB) = 2.5처리

뒤끝 팁

쓰기 작업

데이터 크기	처리량
1KB 이하	1

예시 1

96KB를 저장, 수정, 삭제할 때 사용하는 쓰기 처리량은 96

예시 2

10KB의 데이터를 1KB로 수정할 때 사용하는 쓰기 처리량은 수정 전, 후를 비교하여 큰 데이터를 기준으로 처리량을 계산
수정 전 10KB → 수정 후 1KB, 따라서 사용하는 쓰기 처리량은 10

예시 3

3KB의 데이터를 20KB로 수정할 때 사용하는 쓰기 처리량은 수정 전, 후를 비교하여 큰 데이터를 기준으로 처리량을 계산
수정 전 3KB → 수정 후 20KB, 따라서 사용하는 쓰기 처리량은 20

예시 4

15KB의 데이터를 삭제할 때 쓰기 처리량은 15

뒤끝 팁

- 트랜잭션 처리 기준
  - 트랜잭션 작업의 경우 읽기/쓰기 관계없이 최대 4KB의 데이터만 작업수행 가능
  - 트랜잭션 작업을 수행할 때는 아래 표를 기준으로 읽기/쓰기 처리량 산정

  - 트랜잭션 읽기 작업

데이터 크기	처리량
4KB 이하	2

예시

3.5KB를 불러올 때 사용하는 읽기 처리량은 2

  - 트랜잭션 쓰기 작업

데이터 크기	처리량
1KB 이하	2

예시

3.5KB를 저장, 수정, 삭제할 때 사용하는 쓰기 처리량은 2처리(1KB) + 2처리(1KB) + 2처리(1KB) + 2처리(0.5KB) = 8처리

뒤끝 팁

- 데이터 타입 / 크기
  - 뒤끝 DB에 데이터를 저장할 때 저장한 데이터 타입에 맞게 저장됨
  - 뒤끝 DB에 저장하는 모든 데이터는 인덱스와 인증 정보 등이 추가되기 때문에 row 당 약 200바이트가 증가

뒤끝 팁

DB 이용이 발생하는 기능

- 뒤끝 베이스
  - 뒤끝 베이스 기능의 경우 기능의 종류 관계없이 모든 기능을 호출 시 유저의 인증을 실시하고, 이때 0.5~2 읽기 처리량, 0.5~2 쓰기 처리량이 발생함
  - 즉, DB 이용이 발생하는 경우 해당 기능의 DB 이용량(읽기, 쓰기 처리량) + 유저 인증 처리량이 발생하게 됨

서버 시간 조회, 서버 상태 조회, 서버 버전 조회 이 3가지 함수의 경우 유저 인증으로 인한 읽기, 쓰기 처리량이 발생하지 않음

기능	설명
게임 운영	공지사항, 이벤트, 정책 기능을 이용 시 DB 이용 발생
1대1 문의	1대1 문의 기능을 이용 시 DB 이용 발생
게임 정보	게임 정보를 삽입, 수정, 조회, 삭제할 때 이용한 데이터의 크기만큼 DB 이용 발생
게임 유저	게임 유저 관리에 존재하는 기능을 이용 시 DB 이용 발생

뒤끝 팁

기능	설명
(구) 일반 랭킹	일반 랭킹 기능을 이용 시 DB 이용 발생 유저가 많아질 경우 랭킹 갱신 시 유저의 수에 비례하게 DB 이용량이 증가할 수 있음
(구) 실시간 랭킹	실시간 랭킹 기능을 이용 시 DB 이용 발생 1회 갱신, 조회의 DB 이용량은 유저 수와 관계없이 거의 동일
랭킹	랭킹 기능을 이용 시 DB 이용 발생 1회 갱신, 조회의 DB 이용량은 유저 수와 관계없이 거의 동일 랭킹 초기화 시 컬럼 초기화 기능을 사용할 경우 랭킹에 참여한 유저의 수 만큼 DB 이용 발생
쿠폰 관리	유저 인증 과정에서 발생하는 DB 이용을 제외하고, DB 이용이 발생하지 않음
푸시 설정	푸시 토큰을 저장, 삭제하면서 DB 이용 발생
게임 로그	게임 로그 저장/조회 시 DB 이용 발생
차트 관리	유저 인증 과정에서 발생하는 DB 이용을 제외하고, DB 이용이 발생하지 않음 차트의 데이터 크기와 DB 이용량은 무관

뒤끝 팁

기능	설명
확률 관리	유저 인증 과정에서 발생하는 DB 이용을 제외하고, DB 이용이 발생하지 않음 확률 차트의 데이터 크기와 DB 이용량은 무관
유저 검색	유저 검색 시 DB 이용 발생
우편 기능	우편 기능 사용 시 DB 이용 발생
쪽지 기능	쪽지 기능 사용 시 DB 이용 발생
길드 기능	길드 기능 사용 시 DB 이용 발생
실시간 알림	실시간 알림 자체는 DB 이용이 발생하지 않음
뒤끝 펑션	유저 인증 과정에서 발생하는 DB 이용을 제외하고, DB 이용이 발생하지 않음 다만 뒤끝펑션 내에서 뒤끝베이스의 기능 호출 시 DB 이용이 발생할 수 있음

- 콘솔
  - 콘솔에서 데이터를 조회할 때 SDK와 동일하게 DB 요금 발생함

뒤끝 팁

DB 최적화 가이드

- 뒤끝 DB는 사용한만큼 요금이 부과되는 종량제 서비스
- 게임 데이터를 저장하고 불러오는 방법을 최적화하면
  읽기/쓰기 요금을 효율적으로 관리할 수 있음

- 테이블 설계 최적화
  - 관련이 있는 정보들을 테이블로 묶어 관리하면 읽기/쓰기 처리를
    효율적으로 사용할 수 있음

  - 모든 데이터를 한 번에 저장 (처리량을 많이 사용하는 구조)
    - 캐주얼 게임에서 사용하는 구조. 1개의 테이블에 모든 정보를 관리, 저장
    - 개발이 단순해지는 장점이 있지만 읽기/쓰기 사용량을 필요 이상으로 많이 사용하게 됨
    - item, character, stage 중 하나의 값이 변경될 때 매번 모든 데이터를 다시 쓰고 읽어야 함
    - DB 최적화를 위해 테이블을 나누는 것을 권장

뒤끝 팁

- userData 테이블 1개 사용

```
{
    // userData 내부에서 관리되는 아이템 정보
    "item":
    [
        "보유 아이템1", "보유 아이템2", "보유 아이템3", "보유 아이템4"
    ],
    // userData 내부에서 관리되는 캐릭터 정보
    "character":
    [
        { "nickname": "보유캐릭터1", "level": 5 },
        { "nickname": "보유캐릭터2", "level": 99 }
    ],
    // userData 내부에서 관리되는 스테이지 클리어 정보
    "stage":
    [
        { "stage1": "clear", "clearTime": 55 },
        { "stage2": "clear", "clearTime": 120 }
    ]
}
```

뒤끝 팁

- 테이블 별로 데이터를 저장 (개선된 구조)
  - userData 테이블 안에서 통합 관리한 정보들을 각각의 테이블로 분리한 구조
  - item 테이블의 정보가 변경될 때는 item 테이블을 character 테이블의 정보가 변경될 때는 character 테이블만 쓰고 읽을 수 있음

  - item, character, stage로 나누어진 3개의 테이블 사용
    - item 테이블

```
[ "보유 아이템1", "보유 아이템2", "보유 아이템3", "보유 아이템4" ]
```

    - character 테이블

```
[
    { "nickname": "보유캐릭터1", "level": 5 },
    { "nickname": "보유캐릭터2", "level": 99 }
]
```

    - stage 테이블

```
[
    { "stage1": "clear", "clearTime": 55 },
    { "stage2": "clear", "clearTime": 120 }
]
```

뒤끝 팁

- 데이터 저장 시점 최적화
  - 데이터를 서버에 저장하는 시점을 조정하면 읽기/쓰기 처리량을 효율적으로 관리 가능

  - 핵심 데이터와 비핵심 데이터 분류
    - 핵심 데이터는 캐릭터 레벨업, 스테이지 클리어, 유료 재화(다이아) 구매 등 사라졌을 때의 영향이 큰 정보
    - 비핵심 데이터는 하급 몬스터 사냥 후 얻게 되는 일반 아이템(잡템), 클리커 게임의 클릭 보상 골드 등 빈번하게 변경되는 정보

  - 데이터 저장 시점 분리
    - 핵심 데이터는 변경이 일어난 시점에 바로 저장하는 것을 권장
      예외 : 연속 강화, 연속 레벨업 등으로 핵심 데이터의 변경이 짧은 시간 동안 여러 번 발생하는 게임 → 핵심 데이터의 변경이 완료된 후에 데이터를 저장
    - 비핵심 데이터는 스테이지 클리어 등 특정 이벤트 발생 시점에만 저장하는 것을 권장
    - 중요하지 않은 정보는 꼭 필요할 때에만 저장해 읽기/쓰기 처리량을 절약할 수 있음

뒤끝 팁

- 로그 저장 최적화
  - 뒤끝의 로그 저장 기능은 뒤끝 DB 쓰기 처리를 진행함.
  - 빈번하게 로그를 저장할 경우 쓰기 처리량이 급증할 수 있음

  - 핵심 로그와 비핵심 로그의 분리
    - 핵심 로그는 결제, 인앱 아이템 구매, 닉네임 변경 등
      회원의 계정 정보 또는 매출과 관련된 로그
    - 비핵심 로그는 게임 플레이 시간, 플레이 유형 등 일부가 유실되더라도
      게임 운영에 지장을 주지 않는 로그

  - 로그 저장 시점 분리
    - 핵심 로그는 행위가 일어난 시점에 바로 저장하는 것을 권장
      예외 : 핵심 로그 저장 행위가 짧은 시간 동안 여러 번 발생하는 게임
      → 행위가 완료된 후에 로그를 저장
    - 비핵심 로그는 게임 종료, 스테이지 클리어, 정해진 주기(예:30분) 등
      최대한 지연하여 특정 시점에 저장하는 것을 권장

뒤끝 팁

- 콘솔 조회 최적화
  - 뒤끝 콘솔의 게임 유저 관리, 게임 정보 관리 메뉴는 조회 시 읽기 처리를 수행
  - 게임 정보 관리 읽기 최적화
    - 검색 조건 없이 검색하기 버튼을 클릭하면 모든 회원을 대상으로 조회 시도 → 많은 읽기 처리량을 사용함
    - 대상 검색 조건(회원번호, 회원 아이디, 닉네임)을 지정하면 특정 회원을 대상으로 조회 시도 → 읽기 처리량을 적게 사용할 수 있음
    - 회원 번호, 회원 아이디를 알 수 없는 경우 → 게임 유저 관리 메뉴에서 닉네임으로 회원 정보를 조회하면 회원 번호를 확인할 수 있음

뒤끝 최신 기능

- 멀티 캐릭터

뒤끝 최신 기능

멀티 캐릭터

- 멀티 캐릭터란 (뒤끝 SDK 5.10.x 이상 사용 가능)
  - 한 계정 안에 여러 개의 커스텀 로그인 유저를 가질 수 있는 기능으로
  - RPG 장르에서 로그인 후 캐릭터 선택화면에서 캐릭터 하나를 선택해 로그인 하는 것과 같은 게임을 만들 때 사용

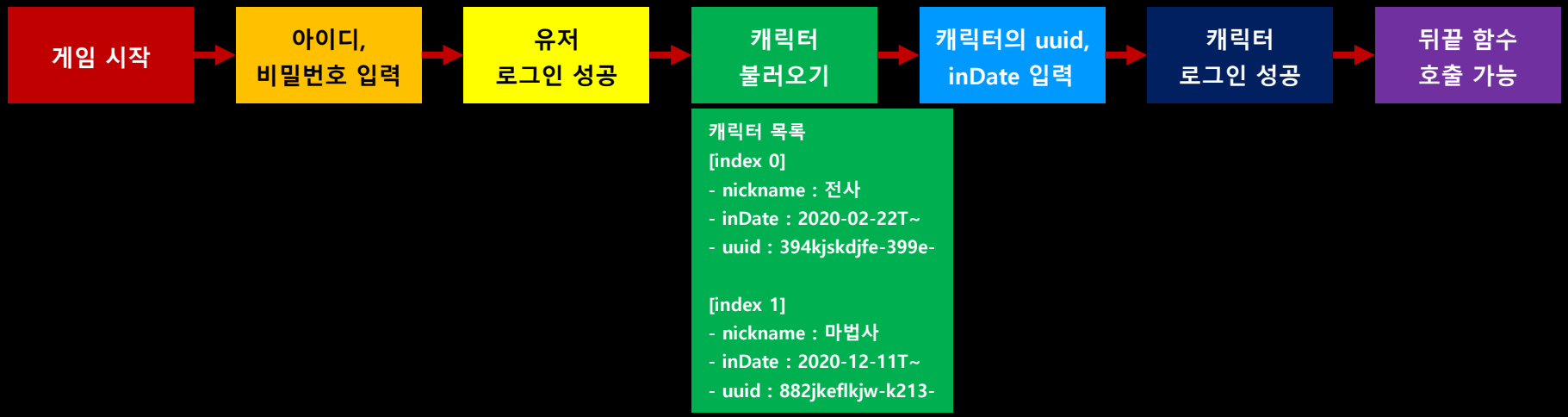
- 멀티 캐릭터 기능 사용시 제한사항
  - 캐릭터의 총 생성 개수 제한은 20개
  - 푸시 알람은 제일 최근에 등록한 유저 한명에게만 발송
  - 내 캐릭터 리스트를 불러올 때 같이 불러올 수 있는 테이블은 1개
  - 현재 멀티 캐릭터의 아이디 혹은 비밀번호를 찾기 위한 수단은 존재하지 않음
  - 계정 로그인에서는 뒤끝 베이스의 호출이 불가능
  - 따라서 소유중인 캐릭터들의 데이터를 종합하여 랭킹에 등록하거나 DB에 저장하는 식의 구현 불가능

뒤끝 최신 기능

■ 멀티 캐릭터 로그인 프로세스

커스텀 로그인을 통한 로그인 프로세스

멀티 캐릭터 계정을 통한 로그인 프로세스



뒤끝 최신 기능

멀티 캐릭터 로그인 프로세스 (계속)

1. 멀티 캐릭터 계정 로그인

먼저 커스텀 로그인과 동일하게 아이디, 비밀번호를 입력해 멀티 캐릭터의 계정 로그인을 호출

2. 캐릭터 불러오기

계정 로그인이 성공하면 해당 계정이 가지고 있는 모든 캐릭터의 리스트를 불러옵니다.

이 때 캐릭터의 프로필 또는 스탯을 표시하기 위해 하나의 테이블을 지정하여 가져올 수 있다.

3. 불러온 정보로 캐릭터 로그인

캐릭터에 로그인하려면 2에서 불러온 각 캐릭터의 uuid와 inDate가 필요

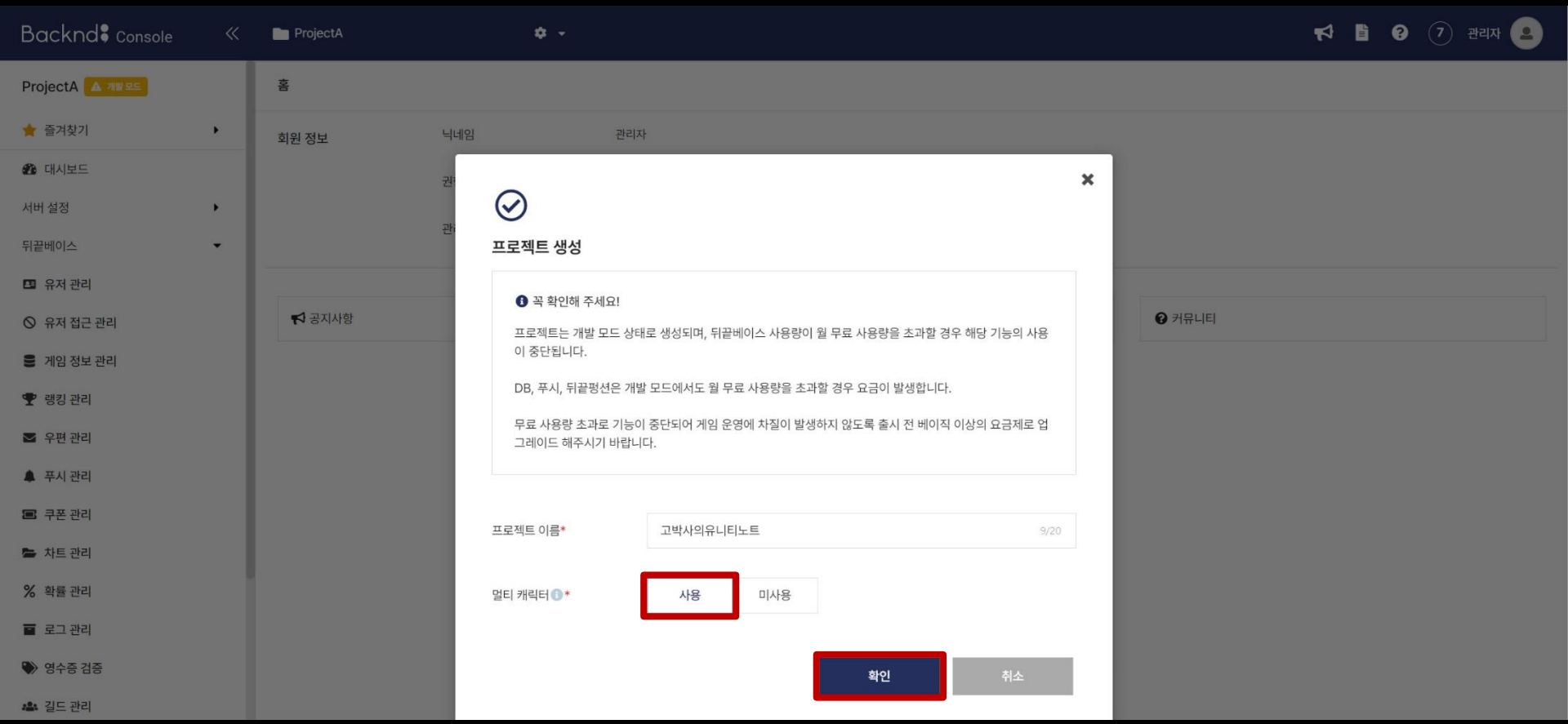
4. 로그인 이후 뒤끝 베이스 기능 이용

캐릭터 선택[SelectCharacter(uuid, inDate)]까지 성공했다면 로그인 성공

이후부터는 커스텀 로그인 성공 이후 로직과 동일하게 뒤끝 베이스 기능을 호출하는 것이 가능

뒤끝 최신 기능

- 멀티 캐릭터 프로젝트 생성
  - Backend Console에서 프로젝트 생성 시 멀티 캐릭터를 "사용"으로 설정
  - 멀티 캐릭터가 "미사용"인 프로젝트에서 멀티 캐릭터를 생성할 경우,
    멀티 캐릭터 전용 UI가 보이지 않아 이용에 불편할 수 있음



뒤끝 최신 기능

멀티 캐릭터 - 계정

1. 계정 생성	CreateAccount(string id, string password);
2. 계정 로그인	LoginAccount(string id, string password);
3. 자동 로그인(토큰 로그인)	AutoLoginAccount();
4. 계정 로그아웃	LogoutAccount();
5. 계정 탈퇴	WithdrawAccount(int expirationHours=0);

가이드 문서 : https://developer.thebackend.io/unity3d/guide/multicharacter/account/함수명/

멀티 캐릭터 - 캐릭터 (계정 로그인 필수)

1. 캐릭터 생성	CreateCharacter(string nickname);
2. 캐릭터 리스트 불러오기	GetCharacterList(string tableName);
3. 캐릭터 로그인	SelectCharacter(string uuid, string inDate);
4. 캐릭터 삭제	DeleteCharacter(string uuid, string inDate);

가이드 문서 : https://developer.thebackend.io/unity3d/guide/multicharacter/character/함수명/