다보리 - 백엔드 Quick Start

- 백엔드 내부 흐름도
- 백엔드 main_api 설치해보기
- MAIN_API와 프론트엔드 연결하기
- Golang 변경사항 *
    - 개선된 Limit 사용 방식
- DB에서 ListToken 활용
- 대용량 데이터 처리를 위한 기술 활용 (Kafka, ES)
- Echo에서 HTTP 응답 반환

백엔드 내부 흐름도

다보리 백엔드에서는 프론트엔드가 요청을 하게되면

Router -> Middleware -> Controller -> Handler(Controller도 Model도 되는 애매모호한 함수들이 서로 호출하며 결과값 도출) -> Mo
보낸다.

백엔드 main_api 설치해보기

MAIN_API 설치 방법!

1. WSL을 설치합니다.

2. 아이디와 비밀번호를 지정하여 서버에 접속합니다.

   `예) sudo adduser <username>.`

3. home 경로에 새로운 백엔드를 설치할 디렉토리를 생성합니다.

   `cd /home`

   `mkdir main_api`

4. 생성이 됐으면 main_a`cd main_api)`□. (

5. wget http://dbr02-wget.daboryhost.com/main_api.tar.gz 를 입력

   (wget은 웹페이지에서 파일을 다운로드하기 위한 도구)

설치가 완료되었으면 아래 디렉토리들이 존재하는지 확인해주세요.

(1). cache-key-pair

(2). weberp-go

(3). weberp-queries

(4). mybin

6. 파일들이 전부 존재하면 디렉토리 접근 후 실행파일 가동하여 서버 시작합니다

```
1. cd weberp-go
```

```
2. ./weberp-go
```

MAIN_API와 프론트엔드 연결하기

Docker(프론트엔드)와 로컬(백엔드)에 각각 띄웠다는 가정하에 설명을 시작하겠습니다

.env.dabory
MAIN_API_URL='http://host.docker.internal:18080' <- URL만 수정해주면 됩니다.

MAIN_API_URL = host.docker.internal:(포트)

프론트엔드 애플리케이션이 **Docker** 컨테이너에서 실행되고, 백엔드 서버는 로컬(호스트 머신)에서 실행될 때

- Docker 컨 `localhost` 를 사용하면 컨테이너 자기 자신 을 참조하게 됩니다.
- 프론트엔드가 Docker 컨테이너에서 실행되고, 백엔드 서버는 로컬(호스트 머신)에서 실행될 때.

MAIN_API_URL= localhost:(포트)

Docker를 사용하지 않는 순수 로컬 개발 환경이라면 이렇게 사용

ThunderClient로 통신 테스트

url = http://localhost:18080/gate-token-get
body에 키 값 넣고 (ClientId, Base64) 게이트 토큰 받기 -> 게이트토큰 발급 후 헤더에 넣어서 엔드포인트 요청 -> 요청이 돌아오면 성공

Golang □□□ *

Golang 변경사항 *

개선된 Limit 사용 방식

1. 기존 방식 : **Limit**와 **IsntPagination**

기존의 백엔드 설계 방식에서는 모든 데이터를 가져오기 위해 다음과 같이 설계되었습니다

- `QueryVars.IsntPagination = true` 를 설정하여 페이지네이션을 비활성화
- `vRet.PageVars.Limit` 값을 매 `1000000000`)로 설정

하지만 이러한 접근 방식은 다음과 같은 문제가 있었습니다

- 코드 가독성 저하
- 불필요한 `Limit` 값은 메모리와 리소스의 낭비를 발생 시킬 수 있습니다.

2. 개선된 방식

`IsntPagination` ListType1 파라 `Limit = 0` 으로 설정하면 모든 데이터를 한 번에 가져오도록 설계되었습니다

(Limit > 500 이상이면 **0**으로 처리됩니다**)**

개선 내용 :

- 코드 `IsntPagination` ListType1 파라미터 옵션을 제거하여 설정을 간소화.
- 자원 `Limit = 0` 으로 동작을 제어하므로 메모리 사용이 최적화됨.
- 유지보수 직관적인 코드 구조로 인해 개발자가 쉽게 이해하고 수정 가능.

3. Query File에서의 Limit 사용

페이징 `LIMIT` [`OFFSET`]을 명시적으로 설정하면 됩니다. (페이징 처리가 불필요할 경우 안넣어줘도 됩니다)

```
SELECT
    mx.id as id,
    turbo_thumb as c1,
    item_name as c2,
    item_slug as c3
FROM dbr_item as mx
    INNER JOIN dbr_igroup as mb ON mx.igroup_id = mb.id
```

```
-- @where
limit 1
```

`LIMIT` 은 반환 **`OFFSET`**)와 개수 를 계산하기 때문에 추가적인 연산 작업이 발생하기 때문에 쿼리 속도가 늦어집니다.

DB에서 ListToken 활용

ListToken 필드란?

- `DbtListType1`에 `ListToken` 필드가 포함되어 있음
- 특정 회원을 `DbtListType1`에 당 사용자가 특정된 데이터가
- 조회 `ListToken` 동일한 값을 부여받음

ListToken의 동작 방식

1. 새로 `DbtListType1`의 데이터가 삭제 후 재삽입됨
2. 새로 `ListToken` 값이 생성됨
3. 이를 통해 사용자가 필터한 회원 데이터로 다양한 작업(**Pick, Act, Page** 등)을 수행 가능

활용 예시

동작	설명
회원 목록 필터링	특정 `DbtListType1`을 저장
ListToken 활용	`ListToken`을 통해 특정 필터링된 회원 집합을 유지
데이터 재사용	한 `Pick`, `Act`, `Page` 등의 작업에 활용

이 방식으로 회원 데이터 조회의 유연성을 높이며, 필터링된 결과를 효율적으로 활용할 수 있도록 설계됨

대용량 데이터 처리를 위한 기술 활용 (Kafka, ES

대용량 데이터 처리 및 Kafka + Bulk 활용

데이터 처리 흐름

1. **DB -> Kafka -> ElasticSearch** 순서로 데이터가 전달됨
2. **Kafka**를 **bucket** 단위로 나누어 데이터 전송 하여 효율적인 처리
3. **ElasticSearch**에는 **Bulk API**를 활용하여 대량의 데이터를 일괄 삽입

주요 개선 사항

1. **Kafka**를 **Bucket** 단→ 데이터 스트리밍 성능 최적화
2. **ElasticSearch**에 **Bulk** → 인덱싱 성능 향상 및 네트워크 부하 감소
3. 대용량 데이터**(150,000**건 이상**)**를 빠른 속도로 처리

Kafka Bucket 처리 예시

```
qryCnt := models.QryCount(y.Db, cntStr)
	fmt.Println("qryCnt:", qryCnt)


	bucketCnt := 0
	if qryCnt == 0 {
		bucketCnt = 0
	} else {
		bucketCnt = 1 + (int(qryCnt) / p.Limit)
	}


    fmt.Println("bucketCnt:", bucketCnt)
	for i := 0; i < bucketCnt; i++ {
		// for i := 0; i < 3; i++ {
		p.Offset = p.Limit * i
		sqlStr, _, _ := models_join.ListType1PlainQryStrGet(y, lt, p, q, "/list")
		if err := t.ProduceaBucketToKafka(y, "kkk", headers, sqlStr, topic); err != nil {
```

```
        return errors.New(e.PageQryErr("vrt452", ": "+err.Error()))
    }
}
```

Bucket □ □□□□ □□ □□ □□!

ElasticSearch Bulk API □□

```
for _, row := range v.Page {
    row.ClientCode = clientCode
    fmt.Printf("BuyerId: %d, LastSorderDate: %s, ClientCode : %s\n", row.Id, row.LastSorderDate, row.ClientCode)
    row.LastVistDate = row.LastSorderDate
    // Meta □□□ □□
    meta := fmt.Sprintf(`{ "index": { "_index": "%s", "_id": "%d" } }%s`, esIndex, row.Id, "\n")
    bulkRequest.WriteString(meta)

    // □□ □□□ □□
    data, _ := json.Marshal(row)
    bulkRequest.WriteString(string(data) + "\n")
    fmt.Println(string(data)) // □□□□ □□□□ □□

    fmt.Printf("Prepared document for BuyerId: %d\n", row.Id)
}

// Bulk □□ □□
res, err := es.Bulk(bytes.NewReader(bulkRequest.Bytes()))
if err != nil {
    return fmt.Errorf("bulk request failed: %w", err)
}
defer res.Body.Close()

// Elasticsearch □□ □□
if res.IsError() {
    log.Printf("Bulk indexing failed: %s", res.String())
    return fmt.Errorf("bulk indexing failed")
}
```

Bulk API□ □□□□ □ □□ □□ □□□□ □□ □□!

□□ □□□ □□□ □□□□ □□□□□ □□□□ □□□ □□□□ □ □□□ □□□.

Echo에서 HTTP 응답 반환

1. 개요

Dabory□**Go** 언어에서는 **Echo** 프레임워크로 프론트엔드 단에 응답데이터로 **Json** □`c.String()`□`c.JSONBlob()`을 사용하여 HTTP 응답을 Json으로 처리하는 □

---

2. `c.String(statusCode int, response string)` 사용법

□ □□

- `c.String()`은 단순 문자열 응답을 반환 하는 함수이다
- `statusCode`는 HTTP □`response`는 프론트엔드로 보낼 응답 본문이다.

□ □□ □□

```
var c echo.Context
return c.String(607, "")
```

□ □□ □□

- `607`이라는 커스텀 **HTTP**□ 응답으로 보낸다.
- 응□`""`□(빈 문자열)이다.
- □**`response.status`** □**`607`**인지 확인 가능

---

3. `c.JSONBlob(statusCode int, jsonData []byte)` 사용법

□ □□

- `c.JSONBlob()`□**JSON** 형식의 응답을 반환 하는 함수이다
- `statusCode`는 HTTP □`jsonData`는 이미 변환된 J`[]byte`□ 데□.

□ □□ □□

```
ret, err := json.Marshal(vRet)
if err != nil {
    return c.String(500, "Failed to encode JSON: "+err.Error())
}
return c.JSONBlob(http.StatusOK, ret)
```

□ □□ □□

1. `json.Marshal(vRet)` □ `ZbaksanSorderEyetestRes` □□□□ JSON □□□□ □□□□.
2. □□□ JS `c.JSONBlob(http.StatusOK, ret)` □ □□ □□□□□□□ □□□□.
3. [**`response.json()`** [**`response.data`** □ **JSON**□ □□ □ □□.

---

4. `c.String()` vs `c.JSONBlob()` □□

□□	**`c.String()`**	**`c.JSONBlob()`**
□□ □□	□□□**(String)**	**JSON (□□□ □□)**
□□ □□	□□ □□□, □□ □□ □□	JSON □□□ □□
□□ □□	`c.String(607, "")`	`c.JSONBlob(http.StatusOK, ret)`
□□□□□ □□	`response.text()`	`response.json()` [`response.data`

---

5. □□

[**`c.String(607, "")`**

- HTTP □ `607` □ □□□□ □□□□□□□□ □□ □□.
- [`response.status` □□□ □□ □ □□ □□ □□.

[**`c.JSONBlob(http.StatusOK, ret)`**

- JSON □□□□ □□□ □ □□.
- `json.Marshal()` □ □□ JSO `response.json()` □□ □□.