구글 API를 이용한 기능 구현

1) 구글 인증

OAuth.2 이용.

2) 캘린더 리스트 받아오기

**public** **static** ArrayList<CalendarDTO> getCalendarList() **throws** IOException{

ArrayList<CalendarDTO> result = **new** ArrayList<CalendarDTO>();

com.google.api.services.calendar.Calendar service = *getCalendarService*();

String pageToken = **null**;

**do** {

CalendarList calendarList = service.calendarList().list().setPageToken(pageToken).execute();

List<CalendarListEntry> items = calendarList.getItems();

**for** (CalendarListEntry calendarListEntry : items) {

//System.out.println(calendarListEntry.getSummary());

CalendarDTO tempDTO = **new** CalendarDTO();

tempDTO.setId(calendarListEntry.getId());

tempDTO.setSummary(calendarListEntry.getSummary());

tempDTO.setCheck(**true**);

tempDTO.setColorId(calendarListEntry.getColorId());

**boolean** primary = **true**;

**if**(calendarListEntry.getPrimary() == **null**) {

primary = **false**;

}

tempDTO.setPrimary(primary);

tempDTO.setAccessRole(calendarListEntry.getAccessRole());

result.add(tempDTO);

}

pageToken = calendarList.getNextPageToken();

} **while** (pageToken != **null**);

**return** result;

}

CalendarDTO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 자료명 | 자료형 | 자료형 크기 | 설명 | 값 |
| id | String |  | 사용자의 캘린더 아이디 | CalendarListEntry.getId() |
| summary | String |  | 사용자의 캘린더 제목 | CalendarListEntry.getSummary() |
| check | boolean |  | 사용자가 캘린더의 체크박스 체크 여부 | Calendar의 id를 name으로 갖는 세션의 value  (초기는 true) |
| colorId | String |  | 사용자의 캘린더 색 아이디 | CalendarListEntry.getColorId() |
| primary | Boolean |  | 캘린더가 사용자의 메인 캘린더인지에 대한 여부 | calendarListEntry.getPrimary() |
| accessRole | String |  | 캘린더의 Access Role(일정에 대한 접근 권한 정의) | calendarListEntry.getAccessRole() |

3) 이벤트 리스트 받아오기

**public** **static** ArrayList<EventDTO> getEvent\_Month(ArrayList<CalendarDTO> calendarList, **int** year, **int** month) **throws** IOException{

com.google.api.services.calendar.Calendar service =

*getCalendarService*();

// List the next 10 events from the primary calendar.

//DateTime now = new DateTime(System.currentTimeMillis());

Date cur = **new** ~~Date~~(year-1900, month-1, 1);

ArrayList<EventDTO> dtoList = **new** ArrayList<EventDTO>();

Date nextDate;

**if**(month == 12) {

nextDate = **new** ~~Date~~(year-1900 + 1,0,1);

}**else** {

nextDate = **new** ~~Date~~(year-1900,month,1);

}

DateTime now = **new** DateTime(cur);

DateTime next = **new** DateTime(nextDate);

**int** size = calendarList.size();

**for**(**int** i=0;i<size;i++) {

//System.out.println(checkedCalId.get(i));

String id = calendarList.get(i).getId();

Events events = service.events().list(id)

.setTimeMin(now)

.setTimeMax(next)

.setOrderBy("startTime")

.setSingleEvents(**true**)

.execute();

List<Event> items = events.getItems();

**if** (items.size() == 0) {

System.***out***.println("No upcoming events found.");

} **else** {

// System.out.println(now.toString());

System.***out***.println("Upcoming events");

**for** (Event event : items) {

DateTime start = event.getStart().getDateTime();

**if** (start == **null**) {

start = event.getStart().getDate();

}

DateTime end = event.getEnd().getDateTime();

**if**(end == **null**) {

end = event.getEnd().getDate();

}

// System.out.printf("%s (%s)\n", event.getSummary(), start.toString());

EventDTO tempDTO = **new** EventDTO();

tempDTO.setCalendarID(id);

tempDTO.setSummary(event.getSummary());

tempDTO.setStart(start.getValue(),start.isDateOnly());

tempDTO.setEnd(end.getValue(),end.isDateOnly());

tempDTO.setEventID(event.getId());

dtoList.add(tempDTO);

}

}

}

Collections.*sort*(dtoList,**new** comparator());

dtoList = **new** EventProcessing().arrangeOrder(dtoList, year, month);

**return** dtoList;

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 자료명 | 자료형 | 자료형 크기 | 설명 | 값 |
| calendarID | String |  | 사용자의 캘린더 아이디 | CalendarListEntry.getID() |
| eventID | String |  | 일정 아이디 | Event.getId() |
| summary | String |  | 일정 제목 | Event.getSummary() |
| start | long |  | 일정 시작 날짜 | Event.getStart().getDateTime().getValue() |
| startTime | Int | 5 | 일정 시작 날짜를 년, 월, 일, 시, 분으로 나눔. | start를 년, 월, 일, 시, 분으로 나눔. |
| end | long |  | 일정 끝 날짜 | Event.getEnd().getDateTime().getValue() 여기서 종일 일정인 경우는 하루 빼줘야 함. |
| endTime | Int | 5 | 일정 끝 날짜를 년, 월, 일, 시, 분으로 나눔. | end를 년, 월, 일, 시, 분으로 나눔. |

4) 이벤트 하나 정보 요청

**public** **static** EventDetailDTO getEventDetail(String calendarId, String eventId) **throws** IOException {

EventDetailDTO result = **new** EventDetailDTO();

com.google.api.services.calendar.Calendar service = *getCalendarService*();

Event event = service.events().get(calendarId, eventId).execute();

result.setSummary(event.getSummary());

DateTime start = event.getStart().getDateTime();

**if** (start == **null**) {

start = event.getStart().getDate();

}

DateTime end = event.getEnd().getDateTime();

**if**(end == **null**) {

end = event.getEnd().getDate();

}

result.setStart(start.getValue(), start.isDateOnly());

result.setEnd(end.getValue(), end.isDateOnly());

result.setLocation(event.getLocation());

result.setDescription(event.getDescription());

result.setRecurrence(event.getRecurrence());

**return** result;

}

**public** **static** Event getEvent(String calendarId, String eventId) {

Event event = **new** Event();

com.google.api.services.calendar.Calendar service;

**try** {

service = *getCalendarService*();

event = service.events().get(calendarId, eventId).execute();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** event;

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 자료명 | 자료형 | 자료형 크기 | 설명 | 값 |
| summary | String |  | 일정 제목 | Event.getSummary() |
| startTime | Int | 5 | 일정 시작 날짜를 년, 월, 일, 시, 분으로 나눔. | 시작 날짜를 년, 월, 일, 시, 분으로 나눔. |
| endTime | Int | 5 | 일정 끝 날짜를 년, 월, 일, 시, 분으로 나눔. | 끝 날짜를 년, 월, 일, 시, 분으로 나눔. |
| description | String |  | 일정 상세 내용 | Event.getDescription() |
| Location | String |  | 일정 장소 | Event.getLocation() |
| Recurrence | List<String> |  | 일정 반복 규칙 | Event.getRecurrence() |

+ 구글의 이벤트 객체 정리

5) 이벤트 생성

6) 이벤트 수정

**public** String updateEvent(HttpServletRequest request, Model model) {

GoogleCalendarService gcs = **new** GoogleCalendarService();

Calendar service;

String calendarId = request.getParameter("calendarId");

String eventId = request.getParameter("eventId");

EventDateTime start = **new** EventDateTime();

EventDateTime end = **new** EventDateTime();

String[] strStartDate = request.getParameter("startDate").split("-");

String[] strEndDate = request.getParameter("endDate").split("-");

System.***out***.println(request.getParameter("allDay"));

**if**(request.getParameter("allDay") != **null**) {

Date startD;

startD = **new** ~~Date~~(Integer.*parseInt*(strStartDate[0])-1900, Integer.*parseInt*(strStartDate[1])-1, Integer.*parseInt*(strStartDate[2]),9,0); //timezone만큼 시간 설정해야함.

System.***out***.println(startD.toString());

start.setDate(**new** DateTime(**true**,startD.getTime(),startD.~~getTimezoneOffset~~())).setTimeZone("Asia/Seoul");

}**else** {

String[] strStartDateTime = request.getParameter("startDateTime").split(":");

System.***out***.println(request.getParameter("startDateTime"));

Date startD = **new** ~~Date~~(Integer.*parseInt*(strStartDate[0])-1900, Integer.*parseInt*(strStartDate[1])-1, Integer.*parseInt*(strStartDate[2]),

Integer.*parseInt*(strStartDateTime[0]), Integer.*parseInt*(strStartDateTime[1]));

start.setDateTime(**new** DateTime(startD)).setTimeZone("Asia/Seoul");

}

**if**(request.getParameter("allDay") != **null**) {

Date endD;

endD = **new** ~~Date~~(Integer.*parseInt*(strEndDate[0])-1900, Integer.*parseInt*(strEndDate[1])-1, Integer.*parseInt*(strEndDate[2])+1,9,0);

DateTime endDate = **new** DateTime(endD);

end.setDate(**new** DateTime(**true**,endD.getTime(),endD.~~getTimezoneOffset~~())).setTimeZone("Asia/Seoul");

}**else** {

String[] strEndDateTime = request.getParameter("endDateTime").split(":");

Date endD = **new** ~~Date~~(Integer.*parseInt*(strEndDate[0])-1900, Integer.*parseInt*(strEndDate[1])-1, Integer.*parseInt*(strEndDate[2]),

Integer.*parseInt*(strEndDateTime[0]), Integer.*parseInt*(strEndDateTime[1]));

end.setDateTime(**new** DateTime(endD)).setTimeZone("Asia/Seoul");

}

System.***out***.println(start.toString());

System.***out***.println(end.toString());

//System.out.println(startDate.toString());

//String[] strStartDateTime = request.getParameter("startDateTime").split(":");

**if**(calendarId.equals(request.getParameter("calendars"))) {

**try** {

service = gcs.*getCalendarService*();

Event event = service.events().get(calendarId, eventId).execute();

event.setSummary(request.getParameter("summary"))

.setLocation(request.getParameter("location"))

.setDescription(request.getParameter("description"))

.setStart(start)

.setEnd(end)

;

Event updatedEvent = service.events().update(calendarId, event.getId(), event).execute();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}**else** **if**(!calendarId.equals(request.getParameter("calendars"))){//calendar Id를 수정한 경우나 일정을 입력한 경우

**try** {

service = gcs.*getCalendarService*();

**if**(!calendarId.equals("addEvent")) {

service.events().delete(calendarId, eventId).execute();

}

Event event = **new** Event()

.setSummary(request.getParameter("summary"))

.setLocation(request.getParameter("location"))

.setDescription(request.getParameter("description"))

.setStart(start)

.setEnd(end)

;

String newCalendarId = request.getParameter("calendars");

service.events().insert(newCalendarId, event).execute();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**return** "redirect:/m/"+request.getParameter("startDate");

}

7) 이벤트 삭제

@RequestMapping(value = "/deleteEvent", method = RequestMethod.***GET***)

**public** @ResponseBody **boolean** deleteEvent(CalendarAndEventIdDTO dto){

**boolean** result = **false**;

GoogleCalendarService gcs = **new** GoogleCalendarService();

com.google.api.services.calendar.Calendar service;

**try** {

service = gcs.*getCalendarService*();

service.events().delete(dto.getCalendarId(), dto.getEventId()).execute();

result = **true**;

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** result;

}

우선순위

* 일정이 여러 날에 거쳐서 나타날 경우
  + End Date가 늦은 순서 > 시간이 있는 일정 > 종일 일정
* 시작 날짜와 끝 날짜가 같은 일정인 경우
  + 종일 일정 > 시간이 있는 일정.
  + 시간이 있는 일정인 경우는 시작 시간이 빠른 순으로 정렬.
* 현재 보고 있는 월보다 시작 날짜가 전인 경우
  + 끝 날짜가 늦은 순서
* 그 외의 경우
  + 시작 날짜와 끝 날짜가 같은 일정인 경우
    - 종일 일정 > 시간이 있는 일정 순서
    - 시간이 있는 일정인 경우에는 시작 시간이 빠른 순으로 정렬.