인턴 중간 발표 1차 보고서

장연수

0) 진행 경과

1주차(3/20~3/23) -> Google Calendar API 조사. 월별 캘린더 출력. 하루 일정 출력. Java Spring 공부

2주차(3/26~3/30) -> 프로젝트 구조 설계. 모델링. AJAX를 이용한 비동기 전송 방식 이용. 이어지는 일정 출력. 체크된 여러 캘린더들의 일정 출력. 일정의 이름을 클릭하면 일정의 정보가 화면에 나오는 것을 구현.

3주차(4/2~4/6) -> 일정이 날짜 칸을 넘어가는 경우 더보기 기능 구현. 일정 상세 정보 페이지 구현. 일정 생성, 수정, 삭제 구현. UI 개선. 목록 캘린더(한달 동안의 일정 목록) 출력.

1) Google Calendar API 소개

Google Calendar API를 쉽게 사용할 수 있도록 라이브러리를 제공

구글 계정 내에 있는 캘린더를 관리할 수 있게 해 줌.

OAuth 2.0 인증 이용

주로 사용한 기능

* Google OAuth2 API v2
  + Google API Client Libraries를 이용해 인증
* Calendar API v3
  + CalendarList (Resources)
    - list : 사용자가 가지고 있는 캘린더들의 정보를 가져옴.
  + Events (Resources)
    - list : 특정 캘린더의 일정들의 정보를 가져옴.
    - get : 특정 캘린더의 특정 일정 정보를 가져옴.
    - insert : 특정 캘린더에 일정 생성.
    - update : 일정 정보 수정.
    - delete : 일정 삭제.

2) 현재까지 구현한 기능

월 뷰 캘린더에 여러 캘린더의 일정 표시

목록 캘린더에 여러 캘린더의 일정 표시

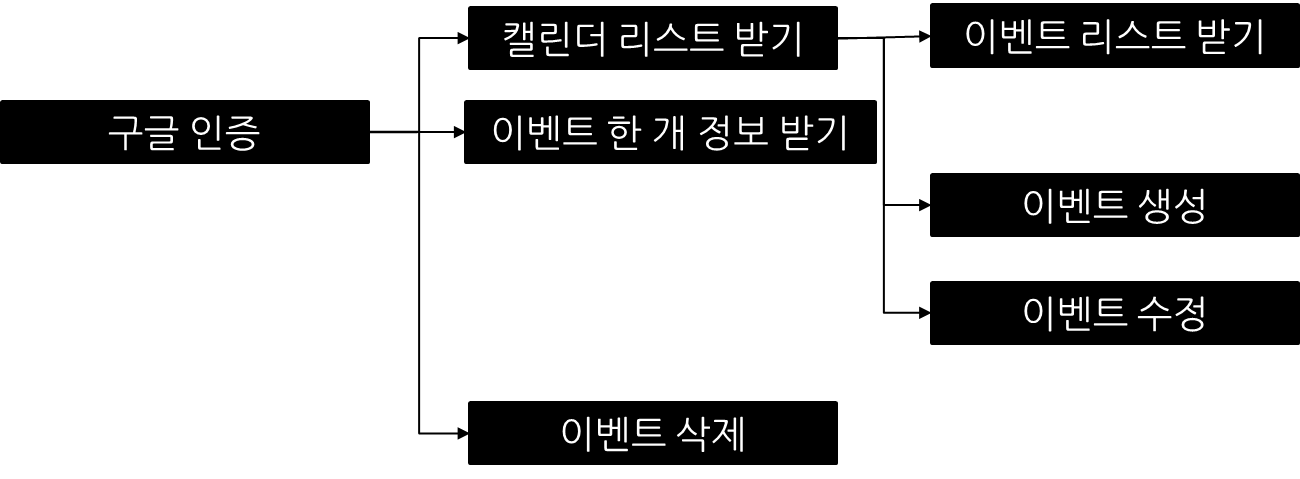
일정 생성, 수정, 삭제

추가로 구현할 기능

* 일 뷰 캘린더, 주 뷰 캘린더
* 일정 생성, 수정 부분에 참가자와 반복 일정 추가 기능.

3) 기능 구현 방법

기능 동작 구조



* 여러 캘린더들에 대해 이벤트를 받거나 생성, 수정이 가능하기 때문에 캘린더 리스트를 받고 이벤트 리스트를 받거나 생성, 수정할 수 있음.
* 이벤트 한 개 정보를 받거나 이벤트를 삭제할 때도 캘린더의 정보(캘린더 아이디)가 필요로 하지만 이벤트 한 개에 대한 캘린더 정보는 바뀌지 않기 때문에 태그에 속성 값으로 저장하여 사용함.

API 사용을 위한 사전 준비

Pom.xml에 Google Calendar API 라이브러리를 쓰기 위한 라이브러리들을 추가함.

* com.google.api-client -> Google api를 더 쉽게 사용하기 위해 제공. 인증을 쉽게 만들어 줌. 구글 로그인 기능 제공.
* org.springframework.social -> Spring Social extension. 소셜 로그인을 해줌.
* com.google.oauth-client -> OAuth 인증 방식을 이용한 구글 인증.
* com.google.apis -> Calendar API v3-rev235-1.22.0 Calendar API를 사용하기 위한 라이브러리 제공.
* org.apache.httpcomponents -> Apache HttpComponents Client HTTP 관련 low level의 java components를 생성하고 유지할 수 있도록 함.

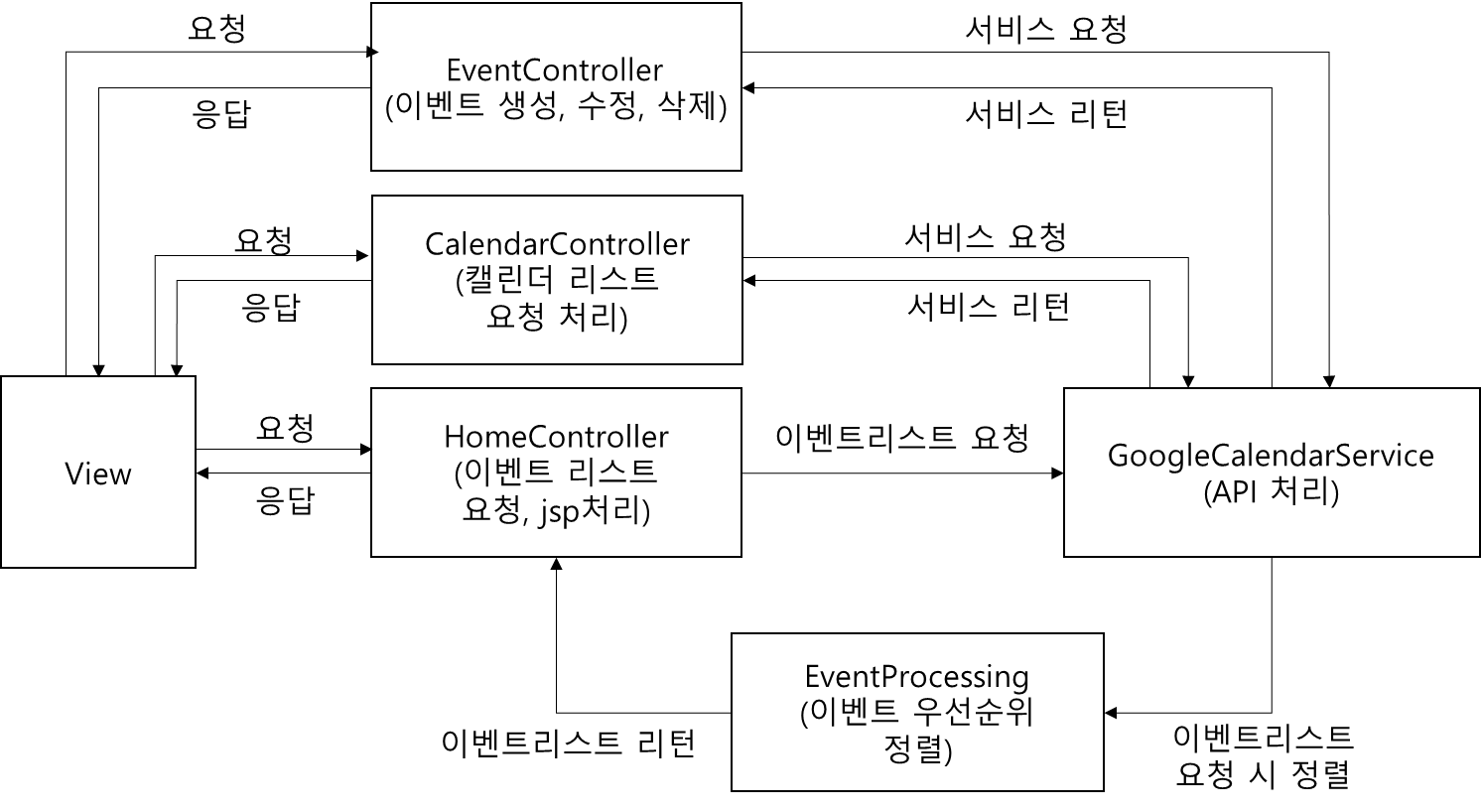
Google Cloud Platform에 접속.

* 사용자 인증 정보를 만들어서 Oauth 클라이언트 ID(특정 사용자의 구글 안에서의 정보를 가져오는 경우)를 받음.
* client\_secret.json 파일을 다운 받아 프로젝트에 저장.
* 프로젝트를 생성하고 사용할 API 설정.

캘린더 공유 설정

* 캘린더 공개 설정 체크

전체 클래스 구조 설계



* 전체적인 동작은 View에서 정보를 요청하거나 생성, 수정, 삭제를 요청하면 그에 맞는 Controller에서 GoogleCalendarService를 통해 서비스(인증과 연결 정보가 있는 Calendar 객체)를 받아 API를 사용하여 처리한 후 응답해 줌.
* View의 jsp 파일 처리는 HomeController에서 해 줌.
* 이벤트 리스트를 요청 받으면 이벤트를 그리기 위한 편의성을 위해 정해 놓은 우선 순위에 맞게 정렬하여 이벤트를 보냄.

구글 API 사용을 위한 인증의 구조

Your application sends a token request to the Google Authorization Server, receives an authorization code,
exchanges the code for a token, and uses the token to call a Google API endpoint.

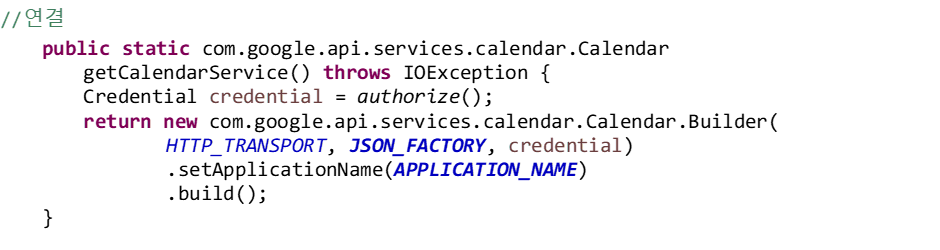
[출처: https://developers.google.com/identity/protocols/OAuth2]

* Google API Console에서 Oauth 2.0 사용을 위한 client id, client secret(일종의 패스워드) 받기
* Token 요청
* 사용자가 SCOPE(권한. 여기서는 캘린더의 읽기/쓰기) 확인 후 로그인을 하고 승인 버튼을 누름.
* Authorization code(사용자가 SCOPE 허락에 대한 비밀 번호)를 서버에서부터 받기
* Authorization code, client id, client secret으로 인증
* Access token을 받아 Google API 사용

구글 API 사용을 위한 인증



* HttpTransport는 GET, POST, PUT, DELETE만 지원해 줌.
* 프로젝트에 저장된 client\_secret.json 파일을 통해 client secret 값을 얻음.
* SCOPES = 캘린더의 읽기/쓰기 권환
* Access type = offline으로 해서 access token이 만료 되어 사용하지 못하게 되면 리소스 서버에 refresh token으로 다시 요청해 access token을 받을 수 있음.
* LocalServerReceiver에서 redirectUri 생성. Host(localhost), port는 -1로 해서 사용하지 않은 port 선정 verification code receiver
* authorize(“user”) 이 부분에서 브라우저를 열어서 승인을 받고 code를 받아서 access token을 받음. 그리고 credential 저장.

JsonFactory -> JSON 파싱 기능 제공.(input stream에서 파싱)

* builder안에 캘린더 정보 요청할 때 사용하는 url 가지고 있음.
* 기본 url: https://www.googleapis.com/calendar/v3
* 만약 캘린더 리스트에 대한 요청을 했다면 위의 url 뒤에
  + - /users/me/calendarList 붙어서 요청.

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) {

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;

}

* 서버에서 받아올 자원들의 개수를 max로 지정해두어도 보장받지 못할 수 있음. 그럴 때 page token을 사용해서 page token이 null이 나올 때까지 받아오면 모두 받을 수 있음. 한 페이지 안에 서버가 보낼 수 있는 최대의 자원들이 들어가 있음.
* 한번 요청에 최대 250개를 받아 올 수 있음. Default = 100
* 캘린더 개수가 많지 않으므로 빼도 상관없지만 page token 사용법을 익히기 위해 기록.
* CalendarList calendarList = service.calendarList().list().setPageToken(pageToken).execute();
  + 이 방법으로 리스트를 얻어 옴.
* 그 후 아래의 DTO에 맞게 저장함.
* DTO를 사용하는 이유는 전송하는 데이터의 사이즈를 작게 해서 전송 시간을 줄이기 위함.
  + API에 요청하고 받아오는 시간이 오래 걸림.
* 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() |

* Access Role = freeBusyReader < reader < writer < owner
* writer, owner만이 일정을 쓸 수 있는 권한이 있음.

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;

}

* calendarList에는 현재 체크 된 캘린더 만 입력 되어 있음.

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

.setTimeMin(now)

.setTimeMax(next)

.setOrderBy("startTime")

.setSingleEvents(**true**)

.execute();

* + id는 calendar Id.
  + setTimeMin과 Max로 이벤트를 받아올 날짜의 범위를 지정. 그 안에 포함된 이벤트들을 모두 가져온다.
    - setTimeMin은 이벤트의 끝 날짜를 기준으로 파라미터 날짜 보다 늦은 이벤트들만 뽑는다. inclusive
    - setTimeMax는 이벤트의 시작 날짜를 기준으로 파라미터 날짜 보다 빠른 이벤트들만 뽑는다. exclusive
    - 둘다 RFC3339 형태. 밀리초 단위는 무시.
  + setOrderBy(“startTime)을 사용하려면 setSingleEvents(true)로 해야함.
    - Default = false
    - 반복 이벤트를 각각 하나의 이벤트로 볼지에 대한 여부
* 이벤트의 시작과 끝 날짜가 종일 일정인 경우(Date)와 아닌 경우(DateTime)로 나뉘므로 다르게 값을 받음.
* 아래의 DTO 형태로 저장.
* 선택된 모든 캘린더들의 이벤트들을 저장하고 우선순위에 맞게 정렬하여 리턴.
* 이벤트 한번 요청 시 받아올 수 있는 최대 개수는 250개.(default = 250) 만약 더 가져오고 싶으면 pageToken 사용.
* EventDTO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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. |

* start나 end를 저장할 때 startTime, endTime의 값을 함수 내에서 파싱해서 세팅.
* 종일 일정의 경우 end는 하루 뒤의 값을 가지므로 하루치 값을 뺌.
  + 종일 일정의 경우 start는 현재 timezone에 의해 그날 9시의 값을 가지게 됨.
  + 추후 이벤트들을 정렬하기 위해서 종일 일정인 경우 start의 값을 그날 오전 12시에서 1밀리초를 뺀 값을 저장.(추후 날짜 정보는 startTime으로만 다루기 때문에 값이 달라져도 상관 없음)

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());

**return** result;

}

* Event event = service.events().get(calendarId, eventId).execute();
  + 이벤트의 calendarId와 eventId를 가지고 요청 후 아래의 DTO 형태로 저장.

**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;

}

* + EventDetailDTO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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() |

이벤트 생성

Event event = **new** Event() .setSummary(request.getParameter("summary"))

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

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

.setStart(start)

.setEnd(end)

.setReminders(reminders);

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

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

* setStart(시작 날짜 설정), setEnd(끝 날짜 설정)의 파라미터 데이터 타입은 EventDateTime을 가짐.
  + EventDateTime을 생성하기 위해서는 DateTime이 필요.
  + 사용자의 인풋을 받아 Date->DateTime->EventDateTime으로 저장.
  + 종일 일정인 경우 DateTime으로 저장할 때 DateOnly값을 true로 지정.
    - end값에 하루를 더 더해줘야 함.(exclusive)
    - 종일 일정을 Date에서 DateTime으로 변경할 때는 long value값을 사용하는데 timezone 때문에 현재 +9시간이므로 시간을 0으로 세팅하지 않고 9로 세팅해야 함.
  + 월 값은 0~11을 가지므로 1을 빼줘야 함.
* setReminders(알림 설정)의 파라미터 데이터 타입은 Reminders
  + Reminders -> useDefault, List<EventReminders>
  + useDefault == true이면 구글에서 정해놓은 default 값인 알림 30분 전, 이메일 10분 전을 사용하겠다는 의미.
  + EventReminder -> method(popup or email), minutes(0~40320, 몇 분 전인지. 최대 4주 전까지 가능)
* 나머지는 String
* Insert의 파라미터로는 이벤트를 넣을 calendarId와 이벤트 객체.

이벤트 수정

//수정

@RequestMapping(value="/updateEvent",method = RequestMethod.***POST***)

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

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

Calendar service;

String calendarId = dto.getCalendarId();

String eventId = dto.getEventId();

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

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

Reminders reminders = **new** Reminders();

String[] strStartDate = dto.getStartDate().split("-");

String[] strEndDate = dto.getEndDate().split("-");

**if**(dto.getAllDay() != **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 = dto.getStartDateTime().split(":");

System.***out***.println(dto.getStartDateTime());

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**(dto.getAllDay() != **null**) {

Date endD;

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

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

end.setDate(**new** DateTime(**true**,endD.getTime()+86400000l,endD.~~getTimezoneOffset~~())).setTimeZone("Asia/Seoul");

}**else** {

String[] strEndDateTime = dto.getEndDateTime().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(":");

//알람 default 체크

**boolean** useDefault = **false**;

**if**(dto.getOverrides() != **null**) {

**if**(dto.getOverrides().size() == 2) {

EventReminder eventReminder = dto.getOverrides().get(0);

**if**(eventReminder.getMethod().equals("popup") && eventReminder.getMinutes() == 30) {

EventReminder eventReminderTwo = dto.getOverrides().get(1);

**if**(eventReminderTwo.getMethod().equals("email") && eventReminderTwo.getMinutes() == 10) {

useDefault = **true**;

}

}

}

**if**(!useDefault) {

reminders.setOverrides(dto.getOverrides());

//System.out.println(dto.getOverrides().get(1).getMethod()+", "+dto.getOverrides().get(1).getMinutes());

}

}

reminders.setUseDefault(useDefault);

**if**(calendarId.equals(dto.getCalendars())) {

**try** {

service = gcs.*getCalendarService*();

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

event.setSummary(dto.getSummary())

.setLocation(dto.getLocation())

.setDescription(dto.getDescription())

.setStart(start)

.setEnd(end)

.setReminders(reminders)

;

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

} **catch** (IOException e) {

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

e.printStackTrace();

}

}**else** **if**(!calendarId.equals(dto.getCalendars())){//calendar Id를 수정한 경우나 일정을 입력한 경우

**try** {

service = gcs.*getCalendarService*();

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

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

}

Event event = **new** Event()

.setSummary(dto.getSummary())

.setLocation(dto.getLocation())

.setDescription(dto.getDescription())

.setStart(start)

.setEnd(end)

.setReminders(reminders)

;

String newCalendarId = dto.getCalendars();

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

} **catch** (IOException e) {

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

e.printStackTrace();

}

}

**return** "redirect:/m/"+dto.getStartDate();

}

* 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)

.setReminders(reminders)

;

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

* + 생성과 거의 동일. 함수만 다름.
  + 이벤트 객체를 새로 생성하는 것이 아니고 calendarId, eventId로 수정할 이벤트의 객체를 얻어와야 함.
  + update의 파라미터는 현재 이벤트의 calendarId, eventId, 이벤트 객체

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;

}

* service.events().delete(dto.getCalendarId(), dto.getEventId()).execute();
  + Delete를 할 때는 지우고자 하는 이벤트의 calendarId, eventId를 파라미터로 넣어 실행시키면 됨.

동작 방식

이벤트나 캘린더를 가져오거나 이벤트를 생성, 수정, 삭제를 하는 요청을 하면

* http 전송을 위해 인증 정보와 json factory, http\_transport를 가지고(Calendar 객체) 전송 준비.
* 요구하는 파라미터에 null이 들어가 있는지 아닌지 확인하고
* 정보를 가져오는 경우에는 GET 메소드 방식, 생성은 POST, 수정은 PUT, 삭제는 DELETE method 사용.
  + 정보를 가져오거나 생성, 수정, 삭제를 하는 함수를 따라 가면 최종적으로 아래의 함수가 나옴.

**public** CalendarRequest(

Calendar client, String method, String uriTemplate, Object content, Class<T> responseClass) {

**super**(

client, method, uriTemplate, content,responseClass);

}

* + - client에는 getCalendarService()를 통해 얻은 Calendar 객체. 인증 받고 HTTP 연결까지 함. Google client.
    - Method는 GET, POST, PUT, DELETE 중 하나
    - uriTemplate는 API를 통해 요청할 때 요청한 행위에 대한 고유 url
    - content는 이벤트를 생성, 수정할 시 필요한 이벤트 객체가 들어감. JsonHttpContent를 사용하여 Json 형식으로 바꿈.
    - responseClass는 요청 후 받아오는 결과값을 어떤 클래스 객체로 파싱 할지 정해줌.
* Execute()를 하면 http request를 수행하고 response를 받음.

4) 우선순위

**public** **class** comparator **implements** Comparator<EventDTO>{

**public** **int** compare(EventDTO dto1, EventDTO dto2) {

**int** ret = 0;

**if**(dto1.getStart() < dto2.getStart()) {

ret = -1;

}**else** **if**(dto1.getStart() == dto2.getStart()){

**if**(dto1.getEnd() > dto2.getEnd()) {

ret = -1;

}**else** **if**(dto1.getEnd() == dto2.getEnd()) {

ret = 0;

}

**else** **if**(dto1.getEnd() < dto2.getEnd()){

ret = 1;

}

}**else** {

ret = 1;

}

**return** ret;

}

}

* 각 캘린더에서 받아온 이벤트들을 시작 날짜가 빠른 순으로 정렬.
  + 시작 날짜가 같은 경우에는 끝 날짜가 늦은 순으로 정렬

**class** comparatorEndDesc **implements** Comparator<EventDTO>{

**public** **int** compare(EventDTO dto1, EventDTO dto2) {

**int** ret = 0;

**if**(dto1.getEnd() > dto2.getEnd()) {

ret = -1;

}**else** **if**(dto1.getEnd() < dto2.getEnd()) {

ret = 1;

}

**return** ret;

}

}

* 현재 사용자가 보고 있는 월보다 시작 날짜가 전인 이벤트들을 모아
  + 끝 날짜가 늦은 순으로 정렬

**class** comparatorSameDate **implements** Comparator<EventDTO>{

**public** **int** compare(EventDTO dto1, EventDTO dto2) {

**int** ret = 0;

**if**(dto1.getStartTime()[0] == dto2.getStartTime()[0] && dto1.getStartTime()[1] == dto2.getStartTime()[1] && dto1.getStartTime()[2] == dto2.getStartTime()[2]) {

**if**(dto1.getEndTime()[0] == dto2.getEndTime()[0] && dto1.getEndTime()[1] == dto2.getEndTime()[1] && dto1.getEndTime()[2] == dto2.getEndTime()[2]) {//같은날

**if**(dto1.getStart() < dto2.getStart()) {

ret = -1;

}**else** **if**( dto1.getStart() > dto2.getStart()) {

ret = 1;

}**else** {//같은 경우

**if**(dto1.getEnd() > dto2.getEnd()) {

ret = -1;

}**else** {

ret = 1;

}

}

}**else** {

**if**(dto1.getEnd() > dto2.getEnd()) {

ret = -1;

}**else** **if**(dto1.getEnd() < dto2.getEnd()) {

ret = 1;

}

}

}

**return** ret;

}

}

* 위의 경우를 제외하고
* 시작 날짜가 같은 경우
  + 끝 날짜도 같으면 시작 시간이 빠른 순으로 정렬(종일 > 시간)
    - 시작 시간이 일치 하면 끝 시간이 늦은 순으로 정렬.
  + 끝 날짜가 같지 않은 경우
    - 끝 날짜가 늦은 순으로 정렬.
* 이렇게 정렬을 하면 그릴 때 끝 날짜가 늦은 순으로 정렬되어 있기 때문에 즉 칸을 많이 차지하는 순서대로 시작 날짜 순서에 맞게 정렬 되어 있어 미리 자리를 차지하게 만들 수 있음.

5) 배운 점과 향후 계획

3주 동안 배운 점

* Java Spring 사용법과 주의 사항
  + Controller의 사용법과 어떤 방식으로 데이터를 View와 주고 받을 수 있을지에 대해 배울 수 있었음.
  + 주의 사항으로는 클래스에 상태 정보를 저장하면 안된다는 것을 배울 수 있었음.
  + 그 후부터는 클라이언트의 정보에 대한 저장을 클라이언트가 하거나 controller에서 session을 이용하여 저장함.
* Java Script, JQuery 사용법
  + 동적으로 태그를 생성하거나 ajax 방식으로 요청하는 법을 배울 수 있었음.
* Google Calendar API 이해
  + 어떻게 동작하는지에 대해 이해할 수 있었고 사용법에 대해 알 수 있었음.
* 캘린더 구현에 대한 이해
  + 어떤 방식으로 캘린더를 그릴지 이벤트를 어떻게 표시하는게 좋을지에 대해 고민하고 참고해가며 구현을 했음.

향후 계획

* 일, 주 뷰 캘린더
* 일정 생성, 수정 부분에 기능 추가.
* 캘린더에 새로운 아이디어 반영.