

# 데이터베이스시스템및응용 최종 과제

## - Personalized Calendar powered by Java and PostgreSQL -

한양대학교 데이터사이언스학부

안준영, 2020047029

### Summary

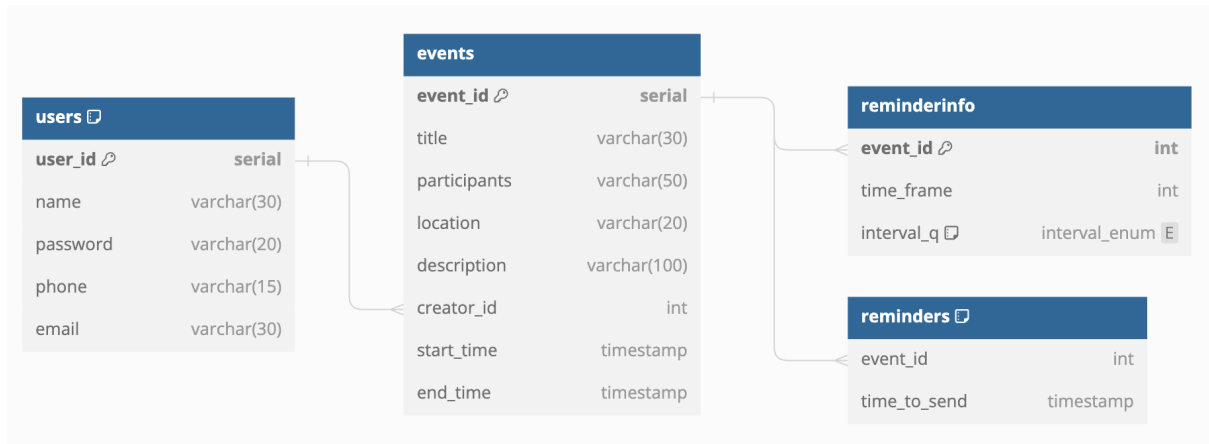
Java 언어의 GUI 라이브러리 Swing과, PostgreSQL의 DB를 활용한 개인 캘린더 프로그램이다. 주어진 requirements를 모두 만족하는 기능을 제공하고 있다. 자주 사용할 최소한의 테이블을 2차 정규화로 구현하여 cost-information trade-off를 맞추었다. 이벤트 생성, 수정 시 예외 처리를 신경 썼다. 반대로 실습 상황에서 예외가 없을 상황은 효율을 위해 생략했다. 랩탑 고장 이슈로 2차 과제 제출 직후까지 VM에선 잘 돌아가던 GUI 코드와 현재 환경 간의 missing link가 있어서 기능하지 않는 부분은 후술하였다.

### Specification

하나의 캘린더 PostgreSQL DB를 통해, 이용자별로 캘린더를 구성하는 것이 아닌 ID에 따라 필터링하는 방식의 월/주/일별 캘린더를 Java언어로 제공한다. Accessibility를 최대화하기 위하여, Swing 외의 라이브러리나 추가 .jar 파일을 사용하지 않는다. 사용자는 각 날짜에 대해 일정을 상세 내용과 함께 예약/수정/삭제할 수 있고, 타인의 일정에는 접근 권한이 없다. 일정 생성 시 1) 겹치는 시간에는 불가하도록 하고 2) time frame & interval만 추가로 입력하면 reminder가 자동 생성되며 3) reminder를 보낸 이후에는 해당 정보를 DB에서 삭제, 4) 캘린더가 실행 중인 동안 1분마다 reminder list check, 5) 계정 정보 수정 시 verification 등 세부적인 컨트롤이 지원된다.

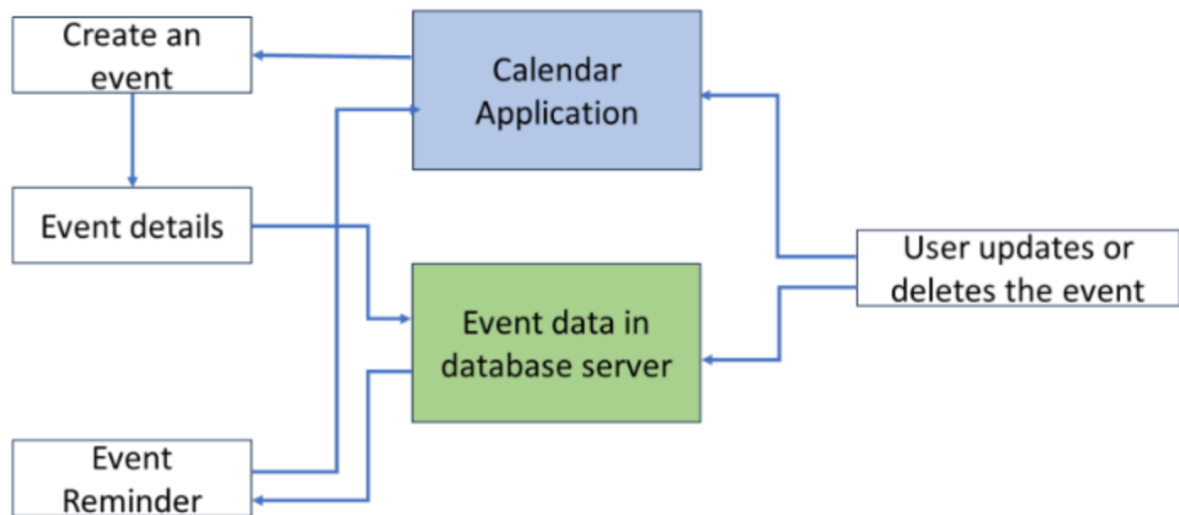
### Design and Implementation

#### <Design>



※ 각 테이블의 역할과 그 attributes들의 특성이 명확하므로, ERD와 통합하여 제시한다.

### <Data Flow Diagram>



**Users** : 유저의 정보를 담고 있는 테이블

**Events** : 일정의 정보를 담고 있는 테이블

**Reminderinfo** : 리마인더의 정보를 담고 있는 테이블

**Reminders** : 하나의 일정에 대해 리마인더를 보낼 시간들의 정보를 담고 있는 테이블

- 1) Reminder를 보낼 때마다 해당 일정의 정보와 join해야 하지만, events&reminders/info를 하나로 합칠 경우 데이터의 중복이 늘어나게 된다. 동시에 레코드의 중복 방지를 위해 더 정규화를 진행하면 테이블의 명시적인 역할이 무너져 유지보수 측면에서 좋지 않다. 이러한 trade-off를 고려하여 events&reminders/info 모두 2차 정규화까지만 진행하였다.
- 2) user\_id와 event\_id를 추가로 유지함으로써, 각각 (name - password)가 unique하지만 코

















드 내에서 이용자를 구분하는 데에 불편함이 있는 것을 해소하고, reminders/info 테이블이 events table을 참조하기 용이하도록 하였다. 이러한 사유가 없는 reminders에서는 불필요한 id를 유지하지 않고, composite key를 곧바로 p-key로 사용한다. 유지 보수 및 효율성 양 측면을 모두 고려하였다.

- 3) Interval은 값을 0부터 60까지 15 단위로만 가질 수 있으므로, enum 자료형을 만들어 사용한다.
- 4) 일정 하나 당 reminder가 여러 개 생기지만, 공통된 information이 있으므로 중간 단계인 reminderinfo를 maintain한다. 이로써 event-reminder 간의 join/not join 비율에 따라 크게 달라질 수 있는 cost가 mid-level인 reminderinfo를 통해 상향 조정된 평균으로 수렴한다.

전반적으로 명세서에 주어진 requirements와 interface를 재현하기 위한 디자인이므로, 코드의 흐름은 생략한다.

### <Implementation>

모든 기능에 대해 SQL문을 작성하였고, 이를 서포트하는 GUI의 우선 순위(효율성 및 구현 정도)를 두 번째로 두었다. 각 기능들은 다음과 같이 java file 단위로 분리되어 있다.

 create_tables.sql	
 DateBox.java	9
 DeleteEvent.java	
 DisplayCalendarDaily.java	3, M
 DisplayCalendarMonthly.java	9+
 DisplayCalendarWeekly.java	9+, M
 DisplayLoginPanel.java	9
 Event.java	3
 EventDetailsPanel.java	U
 Helpers.java	3
 PersonalCalendar.java	9+
 postgresql-42.6.0.jar	
 ReminderPopup.java	9+, M
 StringManager.java	
 UpdateEvent.java	
 ViewEvents.java	M

**PersonalCalendar.java** : 최초 로그인 창 실행을 위한 main method class

**DateBox.java** : 월간 캘린더의 각 칸에 해당하는 JPanel-based class

**DisplayCalendarDaily/Monthly/Weekly.java** : 일/주/월별 캘린더 frame class

**DisplayLoginPanel.java** : 로그인을 받고 verification을 진행하는 class

**DeleteEvent.java** : 일정 삭제 창과 기능이 구체적으로 구현된 class

**UpdateEvent.java** : 일정 수정 창과 기능이 구체적으로 구현된 class

**ViewEvents.java** : 생성된 모든 일정을 확인하는 기능이 구현된 class

**EventDetailsPanel.java** : ViewEvents에서 추가로 각 일정의 상세 정보를 보여주는 class

**ReminderPopup.java** : Reminder를 보내야 할 일정을 탐색하고, 팝업을 띄우는 class

```

public class Helpers {

    public static String dbURL = "jdbc:postgresql://127.0.0.1:5432/dbms_practice";
    public static String dbUser = "dbms_practice";
    public static String dbPasswd = "dbms_practice";
    public static Map<String, List<String>> eventDataMap = new HashMap<>();

    // After clicking create an event button
    public static void createEvent() { ...

    // After clicking view all events button
    public static void viewEvents() { ...

    // After clicking update an event button
    public static void updateEvent() { ...

    // After clicking delete an event button
    public static void deleteEvent() { ...

    // After clicking search events button
    public static void eventList() { ...

    // For changing mode between monthly/weekly/daily
    public static void modeChange(JFrame window) { ...

    // After clicking create an user button
    public static void createUser() { ...

    // After clicking update an user button
    public static void updateUser(String myName) { ...

    // For checking event availability when creating
    private static boolean isEventAvailable(Date newStartTime, Date newEndTime) { ...

    // Convert given information into a SQL query
    private static String converter(String title, String participant, String location) { ...

```

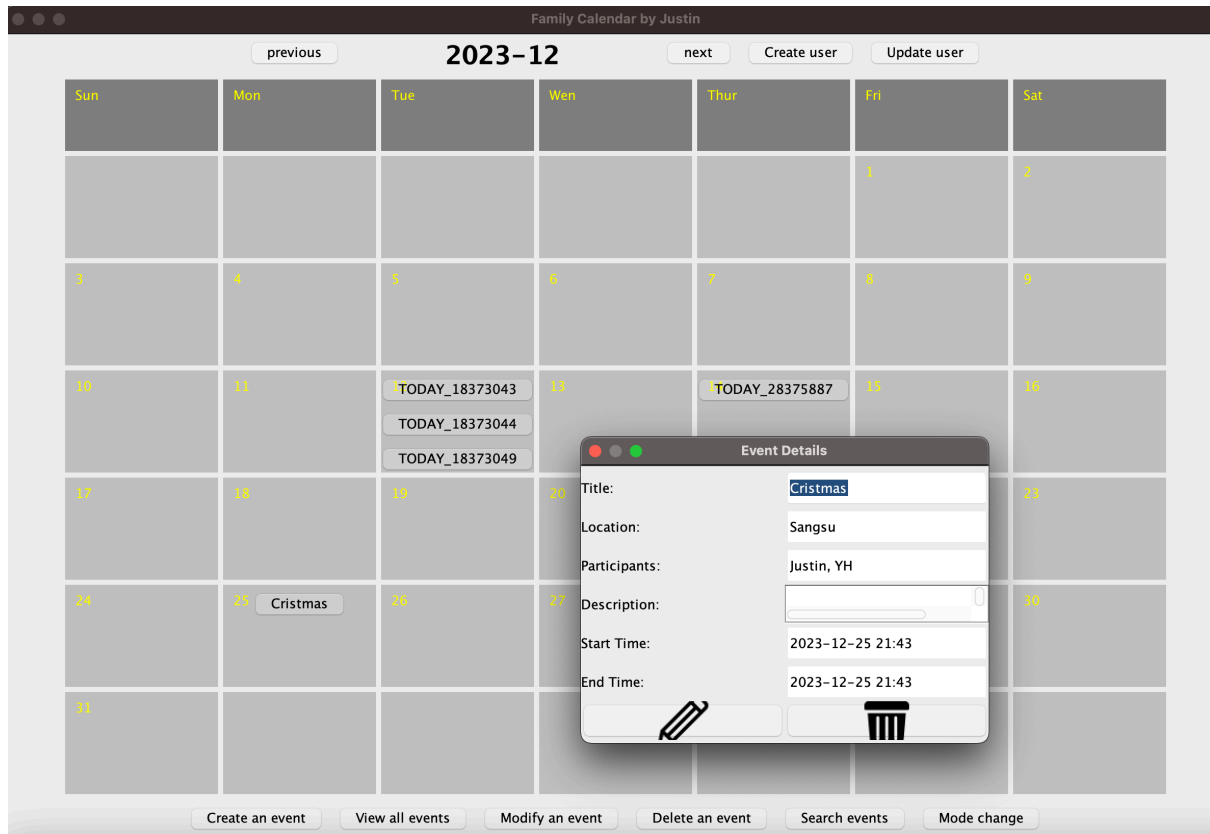
**Helpers.java** : 3종의 캘린더의 버튼에서 공통으로 지원하는 기능들을 static method로 모으고 있으며, 코드 전역에서 사용되는 JDBC configuration도 모두 Helpers의 static variable을 사용하고 있으므로 공통 수정 사항을 하나의 모듈에서 관리할 수 있다.

## Testing

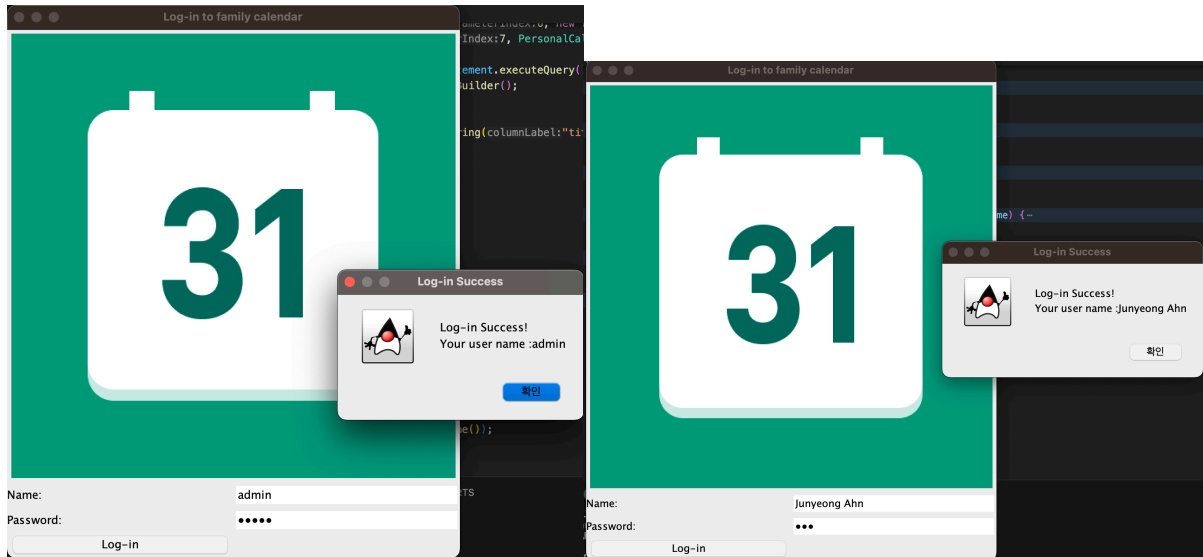
**Test plan:** 파일들을 [Appendix](#) 에서와 같이 배치하고, 1) terminal에서 `javac -cp './src/postgresql-42.6.0.jar:./src/' src/PersonalCalendar.java` 명령어에 이어 2) `java -cp './src/postgresql-42.6.0.jar:./src/' PersonalCalendar` 를 실행함으로써 로그인 인터페이스를 발생시키고, 3) 최초에 psql에서 `INSERT INTO users (name, password) VALUES ('admin', 'admin');`를 통하여 생성한 계정을 사용한다.

1) 로그인한 뒤 월/주/일별 캘린더로 각각 모드를 변경, 2) 다른 계정 생성 후 재 로그인, 3) 일정 생성 및 리마인더 확인, 4) 기타 기능 테스트의 절차를 통해 주요 기능들의 작동을 빠르게 파악한다.

## Result:



월별 캘린더에서 일별 버튼은 총 3개까지 보인다(앱 TimeBlocks에서 착안). 이를 초과해도 View all events나 Search events에서 일정의 creator\_id이 자신의 id와 같도록 predicate로 추가함으로써 '자신의' 모든 일정을 확인할 수 있다. Requirements 2 – 2)에서 요구한 것처럼, 해당 버튼 자체를 클릭하면 상세 일정이 뜨고, 하단의 두 버튼을 통해 재확인 후 수정/삭제가 가능하다.



새로 생성한 Junyeong Ahn 계정으로 재로그인에 성공했다. 계정을 업데이트 할 때는 아래의 코드와 같이 verification을 먼저 진행하고 확인된 사용자에게 대해서만 권한을 부여한다.

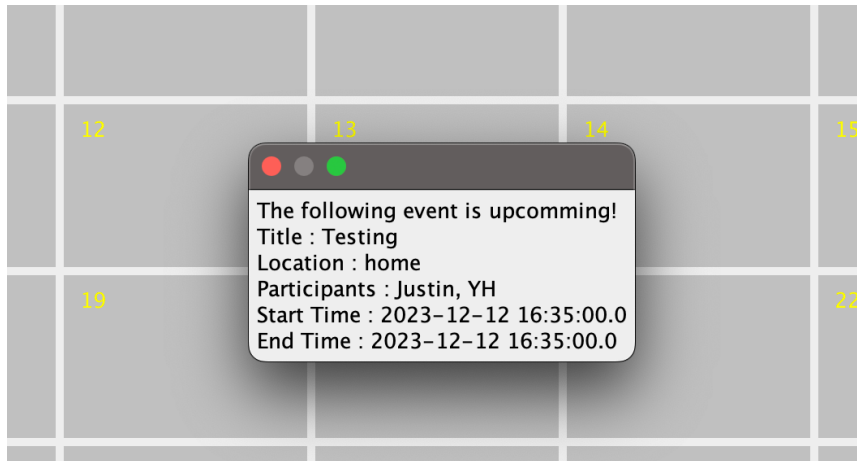
```
verifyButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String passwdToVerify = new String(passwd_verify.getPassword());

        String SQL_SELECT = String.format(
            format:"SELECT * FROM users WHERE name='%s' AND password='%s';", myName, passwdToVerify);

        try (Connection conn = DriverManager.getConnection(Helpers.dbURL, Helpers.dbUser, Helpers.dbPasswd);
            PreparedStatement preparedStatement = conn.prepareStatement(SQL_SELECT)) {
            ResultSet resultSet = preparedStatement.executeQuery();
        }
    }
});
```

```
updateButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (flag.get()) {
            String nameCreated = name.getText();
            String passwdCreated = new String(password.getPassword());
            String phoneCreated = phone.getText();
            String emailCreated = email.getText();

            String SQL_UPDATE = String.format(
                /// SQL query 바꾸고, and flag==true로 확인 후 업데이트
                format:"UPDATE users SET name='%s', password='%s', phone='%s', email='%s' WHERE name='%s' AND password='%s';",
                nameCreated,
                passwdCreated, phoneCreated, emailCreated, myName, password_res.get());
        }
    }
});
```



Event가 잘 생성되고, reminder도 calendar mode에 상관없이 pop up을 제대로 띄워준다. `Helpers.createEvent()`에 event creation과 reminders/info creation이 entangle되어있다. 전송을 완료한 reminders record는 즉시 삭제한다.

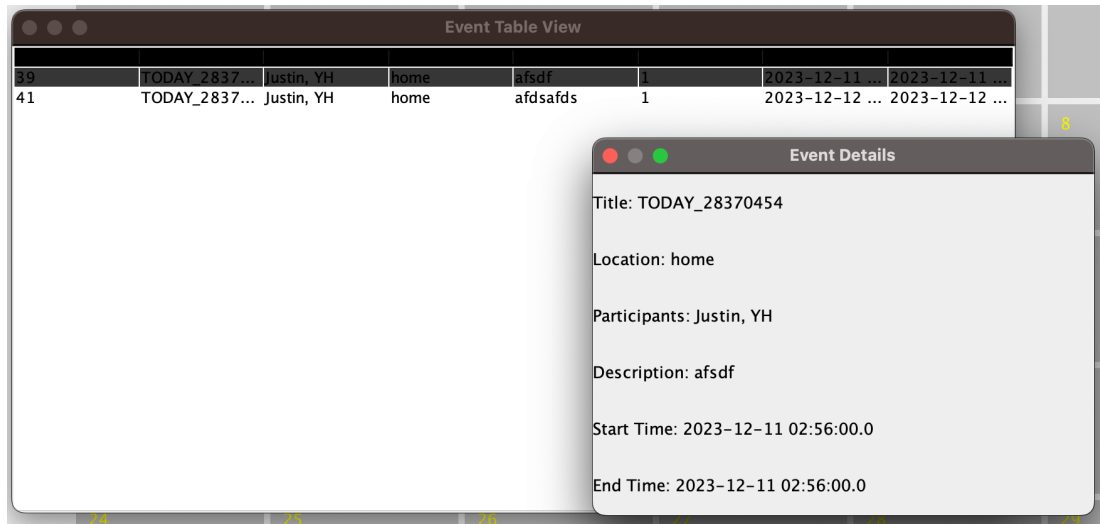
```
try {
    if (isEventAvailable(new java.sql.Timestamp(dateFormat.parse(startCreated).getTime()),
        new java.sql.Timestamp(dateFormat.parse(endCreated).getTime()))) {
        String SQL_INSERT1 = "INSERT INTO events (title, start_time, end_time, description, creator_id, location, participants) VALUES (?, ?, ?, ?, ?, ?, ?)";
        String SQL_INSERT2 = "INSERT INTO reminderinfo (event_id, time_frame, interval_q) VALUES (?, ?, ?::interval_enum)"; // for
        // reminder_info

        try (Connection conn = DriverManager.getConnection(dbURL, dbUser, dbPasswd);
            PreparedStatement stmt1 = conn.prepareStatement(SQL_INSERT1);
            PreparedStatement stmt2 = conn.prepareStatement(SQL_INSERT2);) {
            System.out.println(String.format(format:"endTime : %s", endTime));
            System.out.println(String.format(format:"startTime : %s", startTime));
            while (endTime < startTime) {
                System.out.println(String.format(format:"endTime : %s", endTime));
                System.out.println(String.format(format:"startTime : %s", startTime));
                // reminder 생성
                if (endTime > currentMillis) { // reminder에 넣을 시간이 지금보다 나중일 때만
                    Timestamp reminderTime = new Timestamp(endTime.getTime());

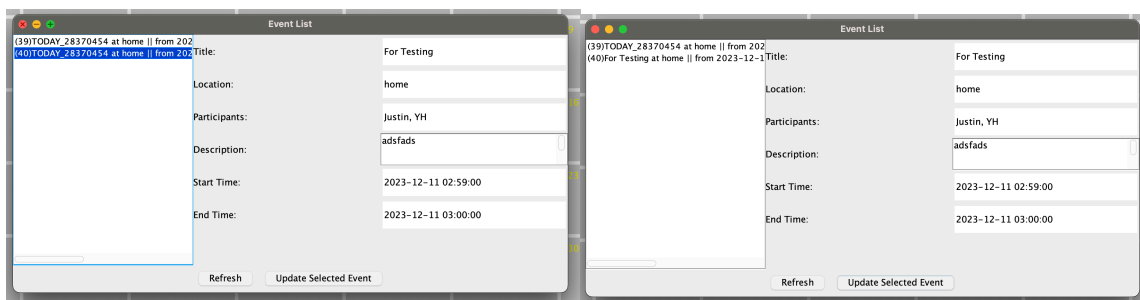
                    // 데이터베이스에 추가
                    String SQL_INSERT3 = "INSERT INTO reminders (event_id, time_to_send) VALUES (?, ?)"; // for
                    // reminders

                    try (Connection conn2 = DriverManager.getConnection(dbURL, dbUser, dbPasswd);
                        PreparedStatement preparedStatement3 = conn2.prepareStatement(SQL_INSERT3);) {
```

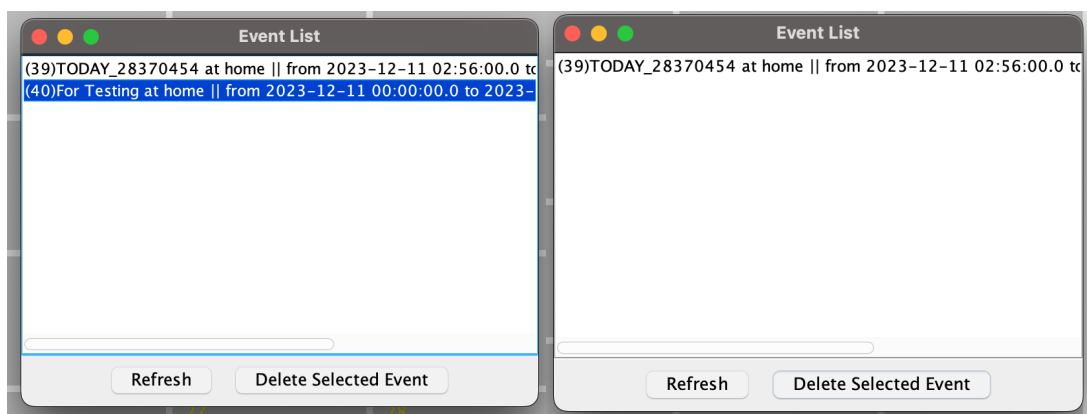




View all events를 누르면 해당 사용자의 모든 이벤트를 볼 수 있고, 이벤트를 클릭하면 디테일을 볼 수 있다. 이외에도 search events를 누르면, requirements의 2 - 5)와 동일하게 구현된 일정 검색 기능을 지원한다.

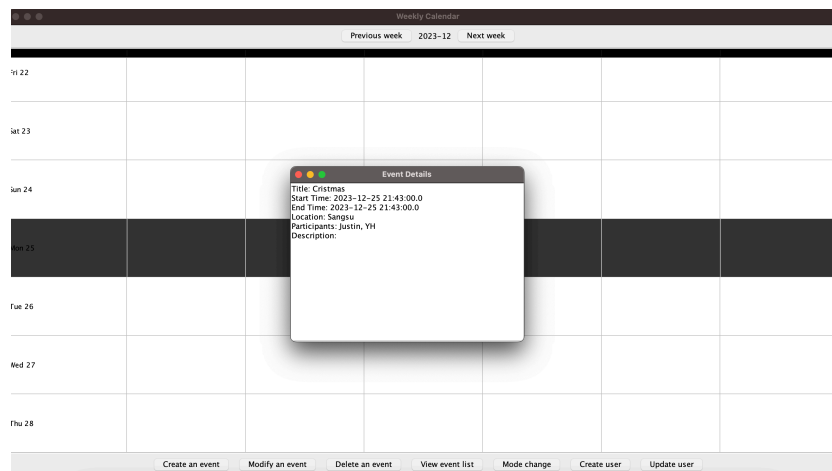


메인 패널에서도 일정을 변경할 수 있지만, 혹시 문제가 생기더라도 Modify an event 버튼을 통해 모든 일정을 수정할 수 있다.

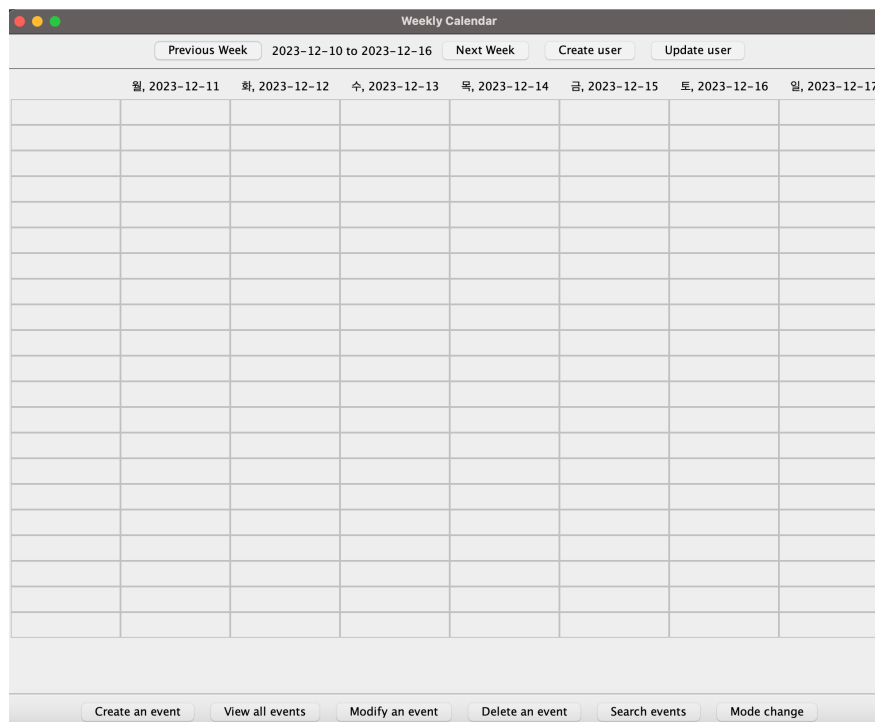


Delete an event 버튼을 통해 일정을 삭제할 수 있다. 위의 기능과 이 기능은 refresh를 클릭하

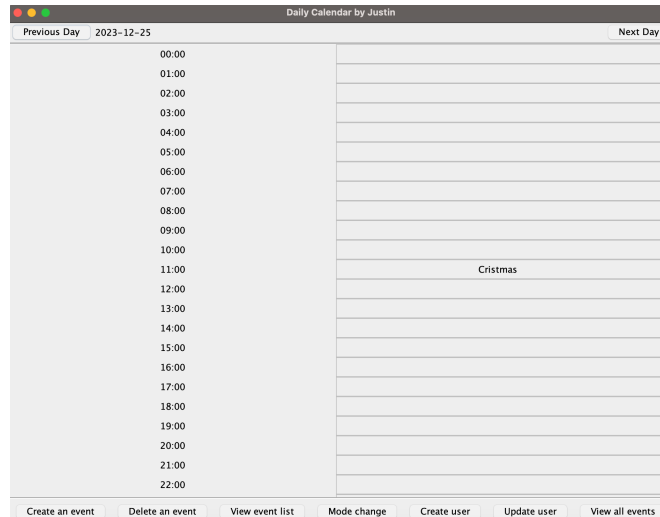
면 up-to-date data를 새로 fetch한다.



### < Weekly calendar GUI (구버전) >



### < Weekly calendar GUI (신버전 - 명세서 재현) >



< Daily calendar GUI >

이외에도 월별/일별 캘린더는 인터페이스에만 차이가 있고, 하단 버튼은 완전히 동일한 기능을 지원한다.

### Known problems

- 2차 과제 직후 고장난 노트북을 Mac(회사 노트북)으로 교체했으나, VM을 지원하지 않고 로컬에 같은 환경으로 자바를 설치하는 것이 보안 프로그램과 많이 충돌하여 11월 초에 겨우 해결하고 곧바로 family calendar 만들기에 돌입했는데, 이마저도 곧 과제 명세서가 많이 변경되면서 급하게 축소/수정하는 과정에서 이전 코드와 missing link가 생겼다. 일례로 명세서에 제시된 '월별 캘린더의 메인 화면에서 각 일정을 클릭할 수 있도록 하는 기능' : 이와 같이 구상한 바 없이 큰 틀을 짚어서 끼워 넣기가 굉장히 어려웠다. 따라서 페이지를 넘길 때 events displaying GUI update가 real time으로 되지 않거나, static할 수 있다. 그러나 이러한 부분에 대해서 SQL문 자체는 완벽하게 작성해 놓았기 때문에, 순수 Java GUI 문제로 해당 DB 과제에서 차치하여 추후 수정할 예정이다.
- 수정된 이벤트의 reminders도 수정해야 하는데, 시간 관계상 아직 구현하지 못하였다. 그러나 event\_id에 해당하는 reminders/reminderinfo를 CASCADE로 지워버리고, event creation에서 구현한 새로운 reminders/reminderinfo 생성 코드를 그대로 이용하면 곧바로 구현 가능하므로 추후 수정할 예정이다.

- 일별 캘린더에서 보여주는 일정의 start\_time과 end\_time이 실제와 조금 차이가 있는 경우가 있다. 이는 통일되지 않은 timestamp 형식의 데이터 삽입으로 생긴 문제로 생각되나, 이 또한 시간 관계상 아직 수정하지 못하였다. 또한 주별 캘린더도 명세서에 급하게 맞추어 renewal하는 과정에서 일정 fetching에 문제가 있다.

## Comment

1분 단위로 fetch하는 reminders를 효율적으로 유지하기 위해, Lecture10 Buffer pool priority hint에서 배운 내용을 활용하여 추후 serial id를 통해 구현하는 것을 최근 구상하여 추후 적용해 볼 생각이다. 사실 DateBox(fancy interface)를 고집하지 않았으면 적당히 panel만 갖다 붙인 캘린더가 되었겠지만, 그 덕분에 이 과제에만 총합 100시간 이상은 써버려서 단순 제출을 넘어 내 개인적인 포트폴리오로 발전시키고자 한다. 시간 관계상 100% 구현이 어려웠으므로, 각 requirement에서의 DB handling을 모두 구현하는 것을 우선으로 전부 진행하고 GUI를 최대한 구현하였다.

## Appendix

### <Directory Structure>

```
.
├── README.md
├── bin
├── calendar.png
├── delete.png
├── lib
├── modify.png
└── src
    ├── DateBox.java
    ├── DeleteEvent.java
    └── DisplayCalendarDaily.java
```

|— DisplayCalendarMonthly.java

|— DisplayCalendarWeekly.java

|— DisplayLoginPanel.java

|— Event.java

|— EventDetailsPanel.java

|— Helpers.java

|— PersonalCalendar.java

|— ReminderPopup.java

|— StringManager.java

|— UpdateEvent.java

|— ViewEvents.java

|— create\_tables.sql

└─ postgresql-42.6.0.jar