

design and implementation of calendar schema, its data structures and UI

2018079116 정진성

Contents

1. Summary
2. Specification
3. Design and Implementation
4. Opinion

1 Summary

Assignment3의 목적은 Family Calendar의 기능을 수행할 수 있도록 사용자 UI와 Database가 통신하는 API 설계하고, 적용하여 Family Calendar를 완성하는 것입니다.

2 Specification

- JAVA 17
- JDBC postgresql-42.7.0

3 Design and Implementation

□ Schema

User

```
CREATE TABLE "user" (  
    user_id serial PRIMARY KEY,  
    id VARCHAR(100) NOT NULL,  
    username VARCHAR(50) NOT NULL,  
    email VARCHAR(100) NOT NULL,  
    password VARCHAR(100) NOT NULL,  
    channel INT NOT NULL);
```

user라는 단어는 postgresql의 예약어이기 때문에 큰따옴표를 붙여 사용했습니다.

user_id를 PK키로 설정, 아이디와 유저의 이름, 이메일, 패스워드 및 채널을 저장하도록 했습니다.

Event

```
CREATE TABLE event (  
    event_id serial PRIMARY KEY,  
    event_name VARCHAR(50) NOT NULL,  
    event_stime Timestamp NOT NULL,  
    event_etime Timestamp NOT NULL,  
    event_detail VARCHAR(100),  
    event_creator_id int NOT NULL);
```

Event_id를 PK키로 설정하고 일정의 이름을 VARCHAR(50)으로 설정했습니다. 일정의 시작과 끝나는 시간을 날짜 및 시간으로 기록하기 위해 Timestamp를 활용하였습니다. 일정 설명, 일정의 주최자를 판단하기 위해 event_creator_id 또한 저장하였습니다.

EventMember

```
CREATE TABLE eventmember (  
    eventmember_id serial PRIMARY KEY,  
    user_id int NOT NULL,  
    event_id int references event(event_id) on delete cascade,  
    foreign key (user_id) references "user"(user_id) on delete cascade);
```

일정과 유저의 다대다 관계를 설정하기 위해 중간의 eventmember 테이블을 만들었습니다. user_id와 event_id를 FK키로 가지는 테이블을 구성하였습니다. 또한 on delete cascade를 활용해 user_id나 event_id를 가진 릴레이션이 삭제되었을 때 eventmember테이블의 해당 릴레이션 또한 삭제되도록 하였습니다.

Reminder

```
CREATE TABLE reminder (  
  reminder_id serial PRIMARY KEY,  
  event_id int references event(event_id) on delete cascade,  
  reminder_start_time TIMESTAMP NOT NULL,  
  interval int NOT NULL);
```

일정의 reminder를 설정하기 위한 table입니다. Event_id를 FK키로 설정하였고, reminder_start_time을 지정하여 reminder가 설정된 시간에 시작하도록 하였습니다. 또한 interval을 integer 형식으로 지정하여 계산할 수 있도록 하였습니다.

RSVP


```
CREATE TABLE rsvp (  
  rsvp_id serial PRIMARY KEY,  
  event_id int references event(event_id) on delete cascade,  
  host_user_id int references "user"(user_id) on delete cascade,  
  guest_user_id int references "user"(user_id) on delete cascade,  
  request_time TIMESTAMP NOT NULL,  
  rsvp_status int NOT NULL);
```

Rsvp는 일정에 대한 rsvp이기 때문에 event_id를 FK로 설정하였고, host 및 guest의 관계 설정을 위해 user_id 또한 FK키로 설정하여 관리하였습니다. 10분이내를 판단하기 위해 request_time을 저장하도록 했고 rsvp를 수락했는지 거절했는지, 보류인지를 저장하기 위해 Integer로 rsvp_status를 저장하였습니다.

▣ Implementations of requirement

- 계정 관리

1) 유저 계정 생성



JB Family Calendar - 회원가입

이름

아이디

패스워드

이메일

Notification 채널 ☐ Pop-up message

뒤로가기 등록

회원가입 화면에 이름과 아이디 패스워드 및 이메일, Notification 채널을 입력할 수 있도록 하였습니다. 입력을 마친 사용자가 등록 버튼을 누르면

```

// 등록 버튼 클릭시
registerBtn.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent actionEvent) {
        // 유저 db 등록
        // 빈 문자열 체크
        if (nameField.getText().isEmpty() ||
            idField.getText().isEmpty() ||
                String.valueOf(pwField.getPassword()).isEmpty() ||
            emailField.getText().isEmpty()
                || !chBtn.isSelected()) {
            JOptionPane.showMessageDialog(jFrame, "모든 필드를
            입력해주세요");
        } else {
            String name = nameField.getText();
            if (!UserAPI.isUserAlreadyExists(name)) {
                User user = UserAPI.registerUser(name,
                idField.getText(), String.valueOf(pwField.getPassword()),
                emailField.getText(), 1);
                if (user != null) {
                    JOptionPane.showMessageDialog(jFrame, "회원가입
                    성공!");
                    jFrame.dispose();
                } else {
                    JOptionPane.showMessageDialog(jFrame, "회원가입
                    실패! 입력을 다시 한번 확인해주세요");
                }
            } else {
                // 사용자 이미 존재 메시지
                JOptionPane.showMessageDialog(jFrame, "동일한 이름의
                사용자가 존재합니다.");
            }
        }
    }
});

```

입력값을 확인하여 적절한 입력이 들어왔는지 확인하고, isUserAlreadyExists함수로 동일한 이름이 있는지 체크합니다. 모든 입력이 Valid하다면 registerUser함수를 통해 user의 정보를 table에 insert 하고 결과를 확인합니다.

```

// 사용자 이름이 이미 데이터베이스에 존재하는지 확인
public static boolean isUserAlreadyExists(String username) {

    String checkUserQuery = "SELECT * FROM \"user\" WHERE username = ?";

    try (Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
, "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
connection.prepareStatement(checkUserQuery)) {

        preparedStatement.setString(1, username);

        ResultSet resultSet = preparedStatement.executeQuery();
        return resultSet.next(); // 결과가 있으면 이미 존재하는 사용자
    } catch (SQLException e) {
        e.printStackTrace();
        return true;
    }
}

```

user 테이블에 조건을 username으로 쿼리를 보내 결과가 있으면 true, 없다면 false를 반환하게 하였습니다.

```

// 사용자를 데이터베이스에 등록
public static User registerUser(String username, String id, String
password, String email, int channel) {
    String signupQuery = "INSERT INTO \"user\" (username, id, password,
email, channel) VALUES (?, ?, ?, ?, ?)";

    try (Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
, "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
connection.prepareStatement(signupQuery, Statement.RETURN_GENERATED_KEYS)) {

        preparedStatement.setString(1, username);
        preparedStatement.setString(2, id);
        preparedStatement.setString(3, password);
        preparedStatement.setString(4, email);
        preparedStatement.setInt(5, channel);

        int rowsAffected = preparedStatement.executeUpdate();

        if (rowsAffected > 0) {
            ResultSet generatedKeys =
preparedStatement.getGeneratedKeys();
            if (generatedKeys.next()) {
                int userId = generatedKeys.getInt(1);
                return new User(userId, username, id, password, email,
channel);
            }
            return null;
        }
        else {
            return null;
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

```

registerUser는 입력받은 값으로 user 테이블에 insert 하고, User객체를 반환하여 결과를 알 수 있도록 하였습니다.

2) 유저 계정 업데이트

캘린더 화면에서 회원정보수정 버튼을 클릭하면 접속한 user의 user_id를 통해 이름 및 아이디 등을 가져오는 getUserByUserId 함수를 통해 현재 user의 정보들이 textfield에 이미 담겨있는 화면을 출력합니다.



A dialog box titled "회원정보 수정" (Modify Member Information) with a close button (X). It contains five text input fields and a radio button selection. The fields are labeled: "새로운 사용자 이름:" (New Username), "새로운 아이디:" (New ID), "새로운 비밀번호:" (New Password), and "새로운 이메일:" (New Email). The radio button is labeled "새로운 Notification 채널:" (New Notification Channel) and is currently selected for "Pop-up message". At the bottom are two buttons: "확인" (Confirm) and "취소" (Cancel).

새로운 사용자 이름:	정진성
새로운 아이디:	정진성
새로운 비밀번호:
새로운 이메일:	정진성
새로운 Notification 채널:	<input checked="" type="radio"/> Pop-up message

확인 취소

변경할 사항을 입력하고 확인 버튼을 누르면 updateUser 함수를 통해 현재 user의 정보를 업데이트 합니다.

```
/ userId 로 user 찾기
public static User getUserByuserId(int userId) {
    String query = "SELECT * FROM \"user\" WHERE user_id = ?";

    try (Connection connection =
        DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
        , "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
        connection.prepareStatement(query)) {

        preparedStatement.setInt(1, userId);

        ResultSet resultSet = preparedStatement.executeQuery();
        if (resultSet.next()) {
            return new User(resultSet.getInt("user_id"),
                resultSet.getString("username"),
                resultSet.getString("id"),
                resultSet.getString("password"),
                resultSet.getString("email"),
                resultSet.getInt("channel"));
        } else {
            return null;
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}
```



```

// userId 에 해당하는 user 정보 업데이트
public static User updateUser(int oldUserId, String newName, String
newId, String newPw, String newEmail, int newCh) {
    String query = "UPDATE \"user\" SET username = ?, id = ?, password
= ?, email = ?, channel = ? WHERE user_id = ?";

    try (Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
, "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

        preparedStatement.setString(1, newName);
        preparedStatement.setString(2, newId);
        preparedStatement.setString(3, newPw);
        preparedStatement.setString(4, newEmail);
        preparedStatement.setInt(5, newCh);
        preparedStatement.setInt(6, oldUserId);

        int rowsAffected = preparedStatement.executeUpdate();
        if (rowsAffected > 0) {
            return getUserById(oldUserId);
        } else {
            return null;
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

```

3) 유저 인증

로그인 화면을 구성해 유저 인증을 구현하였습니다. 이름과 아이디, 패스워드를 입력하고 로그인 버튼을 누르면

```
// 로그인 버튼 클릭시
loginBtn.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent actionEvent) {
        // 사용자 인증
        // 성공시 로그인 창 닫고 캘린더 앱(monthly) 실행
        User user = UserAPI.isUserExists(nameField.getText(),
idField.getText(), String.valueOf(pwField.getPassword()));
        if (user != null) {
            JOptionPane.showMessageDialog(jFrame, "로그인 성공!");
            jFrame.dispose();
            new CalendarApp(user);
        } else {
            JOptionPane.showMessageDialog(jFrame, "로그인 실패!\n 다시
입력해 주세요");
        }
    }
});
```

isUserExists 함수를 통해 이름과 아이디, 패스워드를 비교해 결과를 반환합니다.

```

// 사용자 이름, 아이디, 비밀번호가 데이터베이스에 일치하는 튜플이 있는지 확인
public static User isUserExists(String username, String id, String
password) {
    String checkUserQuery = "SELECT * FROM \"user\" WHERE username = ?
AND id = ? AND password = ?";

    try (Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
, "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
connection.prepareStatement(checkUserQuery)) {

        preparedStatement.setString(1, username);
        preparedStatement.setString(2, id);
        preparedStatement.setString(3, password);

        ResultSet resultSet = preparedStatement.executeQuery();
        if (resultSet.next()) {
            return new User(resultSet.getInt("user_id"),
resultSet.getString("username"),
resultSet.getString("id"),
resultSet.getString("password"),
resultSet.getString("email"),
resultSet.getInt("channel"));
        } else{
            return null;
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

```

이후 로그인에 성공하면

Family Calendar						
<div> <div>< ></div> <div>2023 12</div> <div>검색</div> <div> <input checked="" type="radio"/> 월 <input type="radio"/> 주 <input type="radio"/> 일 </div> <div>일정 추가</div> <div>참석자 확인</div> <div>받은 RSVP</div> <div>회원정보수정</div> <div>Reminder List</div> <div>Refresh</div> </div>						
일	월	화	수	목	금	토
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15 데이터베이스	16
17	18	19	20	21	22 데이터베이스 시험	23
24	25	26	27	28	29	30
31						

Monthly Calendar 가 default 로 설정되어 있어 화면이 구성되고, Event 들을 db 에서 불러와 표시합니다.

```
// 날짜 받아서 날짜에 대한 일정 제목 만 최대 3 개 추가 4 개 때는 ...넣고 무시
ArrayList<String> eventNameList =
EventAPI.getEventNamesByDateAndUserId(java.sql.Date.valueOf(date),
user.getUserId());
int i = 0;
for (String eventName : eventNameList) {
    i += 1;
    if (i > 3) {
        JLabel event = new JLabel("...", SwingConstants.CENTER);
        packPanel.add(event);
        break;
    }
    JLabel event = new JLabel(eventName, SwingConstants.CENTER);
    packPanel.add(event);
}
```

핵심로직은 date 와 현재 접속한 user_id 를 통해 user_id 가 참여하고 있는 event 중 현재달의 일마다 일정을 가져오는 것입니다. 이를 위해 getEventNamesByDateAndUserId 함수를 구성해 사용하였습니다.

```
// date, userId 로 eventname 목록 반환
public static ArrayList<String>
getEventNamesByDateAndUserId(java.sql.Date date, int userId) {

    ArrayList<String> eventNames = new ArrayList<>();
    String query = "select distinct event.event_name from event join
eventmember on event.event_id = eventmember.event_id " +
        "where eventmember.user_id = ? and (DATE(event.event_stime)
<= ? AND DATE(event.event_etime) >= ?)";

    try (Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
, "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

        preparedStatement.setInt(1, userId);
        preparedStatement.setDate(2, date);
        preparedStatement.setDate(3, date);

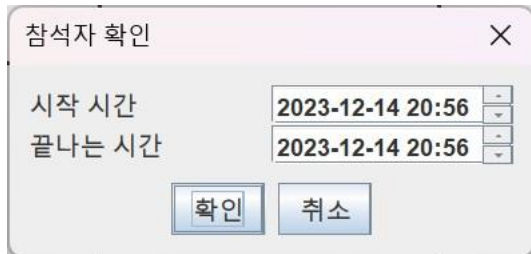
        ResultSet resultSet = preparedStatement.executeQuery();
        while (resultSet.next()) {
            eventNames.add(resultSet.getString("event_name"));
        }
        return eventNames;
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}
```

시간이 아닌 날짜로 비교하기 위해 Date 로 감싸 비교하였고 join 을 활용해 eventmember 테이블의 user_id 를 비교하였습니다. 이로써 해당 날짜에 user 가 참여하고 있는 일정들을 불러와 표시할 수 있었습니다.

- 이벤트 관리

1) 멤버들의 availability 확인

캘린더 화면의 참석자 확인 버튼을 누르면



알고싶은 시간 및 날짜의 시작 시간 및 끝나는 시간을 입력하여 해당 시간에 다른 유저들의 availability 를 확인할 수 있습니다.

```
if (result == JOptionPane.OK_OPTION) {  
    // 시간 겹치는 일정을 가져온다음에  
    ArrayList<Integer> eventIdList =  
    EventAPI.getEventIdsBytime((Date)spinner.getValue(),  
    (Date)spinner.getValue());  
    HashSet<String> noMembers = new HashSet<>();  
    for (int eventId : eventIdList) {  
        noMembers.addAll(EventAPI.getMemberNamesByEventId(eventId));  
    }  
    ArrayList<String> yesMembers = new  
    ArrayList<>(Arrays.asList(UserAPI.getAllUsers()));  
    for (String noUser : noMembers) {  
        yesMembers.remove(noUser);  
    }  
}
```

이는 해당 시간의 겹치는 일정을 가져오고, 일정의 참석자들을 불가능, 그 외의 유저를 가능으로 구분합니다.

```

/ 해당 시간 term 에 겹치는 eventIds 반환
public static ArrayList<Integer> getEventIdsBytime(Date sTime, Date
eTime) {
    ArrayList<Integer> eventList = new ArrayList<>();

    String query = "SELECT event_id FROM event WHERE (event_stime <= ?
and event_etime >= ?) or (event_stime <= ? and event_etime >= ?)";

    try (Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
, "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
connection.prepareStatement(query, Statement.RETURN_GENERATED_KEYS)) {

        preparedStatement.setTimestamp(1, new
Timestamp(sTime.getTime()));
        preparedStatement.setTimestamp(2, new
Timestamp(sTime.getTime()));
        preparedStatement.setTimestamp(3, new
Timestamp(eTime.getTime()));
        preparedStatement.setTimestamp(4, new
Timestamp(eTime.getTime()));

        ResultSet resultSet = preparedStatement.executeQuery();
        while (resultSet.next()) {
            eventList.add(resultSet.getInt("event_id"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
    return eventList;
}

```

```

// event 에 포함된 usernames 반환
public static ArrayList<String> getMemberNamesByEventId(int eventId) {
    ArrayList<Integer> userIds = new ArrayList<>();
    ArrayList<String> usernames = new ArrayList<>();

    String query = "SELECT user_id FROM eventmember WHERE event_id = ?";

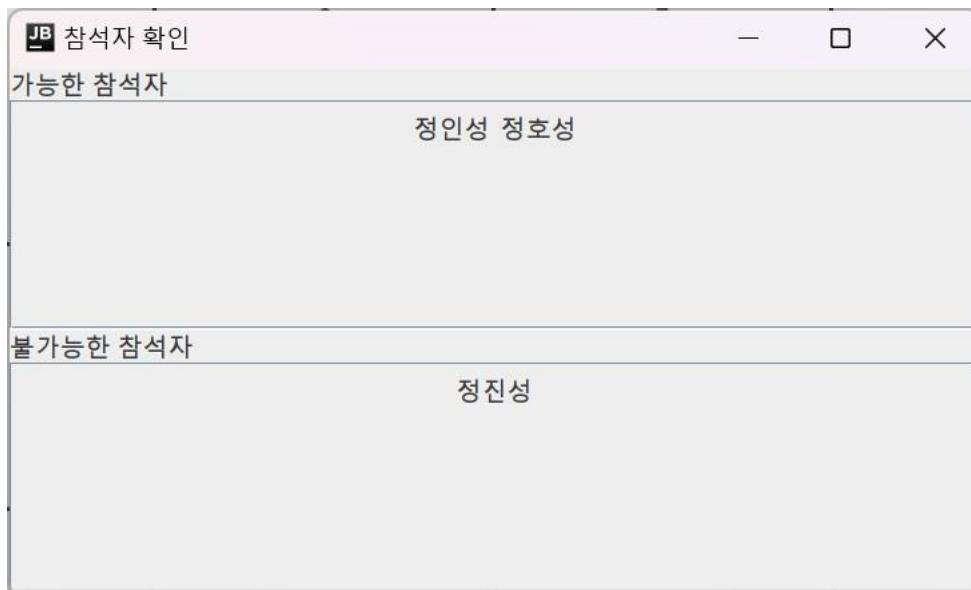
    try (Connection connection =
        DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
        , "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
        connection.prepareStatement(query, Statement.RETURN_GENERATED_KEYS)) {

        preparedStatement.setInt(1, eventId);

        ResultSet resultSet = preparedStatement.executeQuery();
        while (resultSet.next()) {
            userIds.add(resultSet.getInt("user_id"));
        }

        for (int userId : userIds) {
            String userName = UserAPI.getUserNameByUserId(userId);
            if (userName != null) {
                usernames.add(userName);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
    return usernames;
}

```



참석자 확인 결과 화면 예시입니다.

2) RSVP 요청

캘린더의 일정을 더블클릭하면 일정의 주최자를 확인한 후 RSVP 요청이 활성화된 일정 상세 정보화면이 나타나게 되고 이곳에서 RSVP 요청을 통해 일정에 대한 RSVP 를 다른 유저들에게 요청할 수 있습니다.

RSVP 요청 버튼을 누르면

다른 유저들을 선택할 수 있습니다. 보낼 유저의 목록은 모든 유저를 가져와 현재 접속한 유저를 뺀 나머지입니다. RSVP 요청은 registerRSVP 함수로 수행합니다.

```

public static RSVP registerRSVP(int eventId, int hostUserId, int
guestUserId, Date requestTime) {
    String query = "INSERT INTO rsvp (event_id, host_user_id,
guest_user_id, request_time, rsvp_status) VALUES (?, ?, ?, ?, ?)";

    try (Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
, "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
connection.prepareStatement(query, Statement.RETURN_GENERATED_KEYS)) {

        preparedStatement.setInt(1, eventId);
        preparedStatement.setInt(2, hostUserId);
        preparedStatement.setInt(3, guestUserId);
        preparedStatement.setTimestamp(4,
Timestamp.from(requestTime.toInstant().truncatedTo(ChronoUnit.MINUTES)));
        preparedStatement.setInt(5, 0);

        int rowsAffected = preparedStatement.executeUpdate();

        if (rowsAffected > 0) {
            ResultSet generatedKeys =
preparedStatement.getGeneratedKeys();
            if (generatedKeys.next()) {
                int RSVPId = generatedKeys.getInt(1);
                return new RSVP(RSVPIId, eventId, hostUserId, guestUserId,
requestTime, 0);
            }
            return null;
        }
        else {
            return null;
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

```

Rsvp_status 의 값은 -1, 0 ,1 을 가지며, 각각 거절, 보류, 수락을 의미합니다. 따라서 요청을 보낼 때 보류인 상태이기 때문에 0 을 넣어주었습니다.

```

JPanel bodyPanel = new JPanel(new GridLayout(1, 2));
    ArrayList<RSVP> recRSVPList =
RSVPAPI.getRSVPListByGuestUserIdAndStatus(user.getUserId(), 0);
    ArrayList<RSVP> RSVPListIn10 = new ArrayList<>();
    Date cDate = new Date();
    for (RSVP rsvp : recRSVPList) {
        // 현재 시간과의 차이가 10 분 이내인지
        if ((cDate.getTime() - rsvp.getRequestTime().getTime()) /
(60 * 1000) <= 10) {
            RSVPListIn10.add(rsvp);
        } else {
            // rsvp 객체 상태 업데이트
            RSVPAPI.updateStatusByrsvpId(rsvp.getRSVPId(), -1);
        }
    }
    JList<RSVP> recRSVP = new JList<>(RSVPListIn10.toArray(new
RSVP[0]));
    // 보낸거 확인할 때 일단 RSVP 다 가져오고
    // 상태가 0 인것중 현재시간차이 10 분이상인건 -1 로 바꿔
    ArrayList<HostRSVP> reqRSVPList =
RSVPAPI.getRSVPListByHostUserId(user.getUserId());
    for (HostRSVP rsvp : reqRSVPList) {
        // 현재 시간과의 차이가 10 분 이내인지
        if (!((cDate.getTime() - rsvp.getRequestTime().getTime())
/ (60 * 1000) <= 10)) {
            // rsvp 객체 상태 업데이트
            RSVPAPI.updateStatusByrsvpId(rsvp.getRSVPId(), -1);
            rsvp.setStatus(-1);
        }
    }
    JList<HostRSVP> reqRSVP = new JList<>(reqRSVPList.toArray(new
HostRSVP[0]));

```

Rsvp 버튼을 눌렀을 때 받은 RSVP, 보낸 RSVP 가 각각 왼쪽 오른쪽에 보낸 사람, 받은 사람으로 구별되어 나타납니다. 10 분 이상이 지난 RSVP 라면 상태를 -1 로 바꾸어 거절을 의미하도록 하였습니다.

```
// rsvp 상태 업데이트
public static void updateStatusByrsvpId(int rsvpId, int status) {
    String query = "UPDATE rsvp SET rsvp_status = ? WHERE rsvp_id = ?";

    try (Connection connection =
        DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
        , "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
        connection.prepareStatement(query)) {

        preparedStatement.setInt(1, status);
        preparedStatement.setInt(2, rsvpId);

        preparedStatement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

수락 버튼이나 거절 버튼을 누르면 updateStatusByrsvpId 함수를 통해 rsvp 의 상태가 업데이트 되도록 하였습니다.

3) 일정 생성

캘린더의 상단에 일정 추가 버튼을 누르면

✕

일정 이름		
참석자(아무도 선택하지 않으면 개인일정)	정인성 정호성	
시작 시간	2023-12-15 01:14	▲ ▼
끝나는 시간	2023-12-15 01:14	▲ ▼
일정 설명		
Reminder Interval	<input checked="" type="radio"/> 0 <input type="radio"/> 15 <input type="radio"/> 30 <input type="radio"/> 45 <input type="radio"/> 60	
Reminder Time(15~60)		

확인
취소

입력할 수 있는 화면이 나타납니다. 참석자, 시작시간, 끝나는 시간, 설명 및 Reminder 를 설정할 수 있도록 하였습니다.

```

// 참석자들 중에 현재 설정된 시간에 이벤트가 있는지 체크
    for (String member : eventMembers) {
        for (Event event :
UserAPI.getEventListByuserId(UserAPI.getUserIdByUsername(member))) {
            Date mEventStartTime = event.getEventSTime();
            Date mEventEndTime = event.getEventETime();

            if ((eventSTime.getTime() <=
mEventEndTime.getTime() && eventSTime.getTime() >= mEventStartTime.getTime()
) || ( eventETime.getTime() <=
mEventEndTime.getTime() && eventETime.getTime() >=
mEventStartTime.getTime())) {
                JOptionPane.showMessageDialog(null, member +
"의 " + event.getEventName() + "일정으로 인해 설정된 시간이 불가능합니다");
                return;
            }
        }
    }
}

```

확인 버튼을 누르면 입력받은 값을 바탕으로 참석자들의 다른 일정들의 시간을 체크하고 겹치는 일정이 있으면 어떤 일정 때문에 불가능 한지 나타내고 종료하도록 했습니다.

```

public static Event registerEvent(String eventName, Date sTime, Date eTime,
String detail, int eventCreatorId) {
    String query = "INSERT INTO event (event_name, event_stime,
event_etime, event_detail, event_creator_id) VALUES (?, ?, ?, ?, ?)";

    try (Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
, "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
connection.prepareStatement(query, Statement.RETURN_GENERATED_KEYS)) {

        preparedStatement.setString(1, eventName);
        preparedStatement.setTimestamp(2, new
Timestamp(sTime.getTime()));
        preparedStatement.setTimestamp(3, new
Timestamp(eTime.getTime()));
        preparedStatement.setString(4, detail);
        preparedStatement.setInt(5, eventCreatorId);

        int rowsAffected = preparedStatement.executeUpdate();

        if (rowsAffected > 0) {
            ResultSet generatedKeys =
preparedStatement.getGeneratedKeys();
            if (generatedKeys.next()) {
                int eventId = generatedKeys.getInt(1);
                return new Event(eventId, eventName, sTime, eTime,
detail, eventCreatorId);
            }
            return null;
        } else {
            return null;
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

```

입력받은 값이 valid 하다면 새로운 Event 릴레이션을 추가하는 쿼리를 보내 데이터베이스에 등록하였습니다.

4) 이벤트 참석자 초대

참석자(아무도 선택하지않으면 개인일정)	정인성
	정호성

일정 추가시에 참석자들을 추가하면 참석자들의 일정을 확인하고 해당하는 시간에 일정이 없을 시에 일정을 추가하게 됩니다. 이때 eventMember 테이블에 event_id 와 user_id 을 저장하여 일정의 참석자를 저장할 수 있도록 하였습니다.

```
public static int registerMembers(ArrayList<String> members, int eventId) {
    for (String member : members) {
        int userId = UserAPI.getUserIdByUsername(member);

        String query = "INSERT INTO eventmember (event_id, user_id)
VALUES (?, ?)";

        try (Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
, "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

            preparedStatement.setInt(1, eventId);
            preparedStatement.setInt(2,userId);

            int rowsAffected = preparedStatement.executeUpdate();

            if (rowsAffected <= 0) {
                return -1;
            }
        } catch (SQLException e) {
            e.printStackTrace();
            return -1;
        }
    }
    return 0;
}
```

5) 이벤트 상세 보기

초대 받거나 생성한 이벤트를 캘린더 화면에서 접근하여 상세정보를 볼 수 있습니다. 월간 캘린더에서는 날짜를 누르면 날짜에 대한 유저의 일정 목록이 나오고 이를 더블클릭하면 일정의 상세 정보를 볼 수 있습니다. 주간 달력과 일간 달력은 일정을 클릭하면 상세 정보를 볼 수 있습니다.

JB 데이터베이스상세

일정 삭제

일정 수정

RSVP 요청

일정 이름	데이터베이스
주최자	정진성
참석자	[정진성, 정인성]
일정 시작 시간	2023-12-15 13:00
일정 끝나는 시간	2023-12-15 16:00
일정 설명	강의

```
// eventName 으로 Event 객체 반환
public static Event getEventByEventName(String eventName) {
    String query = "SELECT * FROM event WHERE event_name = ?";

    try (Connection connection =
        DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice",
            "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
            connection.prepareStatement(query)) {

        preparedStatement.setString(1, eventName);

        ResultSet resultSet = preparedStatement.executeQuery();
        if (resultSet.next()) {
            return new Event(resultSet.getInt("event_id"), eventName,
                resultSet.getTimestamp("event_stime"),
                resultSet.getTimestamp("event_etime"),
                resultSet.getString("event_detail"),
                resultSet.getInt("event_creator_id"));
        } else {
            return null;
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}
```

일정이름으로 데이터베이스에 저장되어있는 일정 릴레이션을 찾고, 이를 통해 다른 칼럼 값을 가져와 보여주는 방식입니다.

6) 일정 수정

일정 수정 기능은 일정에 주최자를 저장하여 주최자의 계정으로 접속했을 경우에만 일정상세 화면에 일정 수정 버튼이 활성화 되도록 하였습니다.

일정 수정

일정 이름

데이터베이스

참석자

정진성

정인성

정호성

시작 시간

2023-12-15 13:00

끝나는 시간

2023-12-15 16:00

일정 설명

강의

확인

취소

일정 수정 화면에는 현재의 일정의 정보를 받아와 미리 입력으로 입력해두고 사용자가 변경할 사항만 변경할 수 있도록하고 확인 버튼을 누르면 업데이트 할 수 있도록 하였습니다.

7) 일정 취소(삭제)

일정 취소는 사용자 계정과 일정의 주최자를 확인하여 주최자가 일정을 삭제할 경우 모든 사용자의 캘린더에서 일정이 삭제되고, 주최자가 아닌 참가자가 일정을 삭제할 경우 자신의 캘린더에서만 일정이 삭제되도록 eventMember 테이블에서 참가자의 user_id 를 삭제하여 구현하였습니다.

```
@Override
public void actionPerformed(ActionEvent actionEvent) {
    // 주최자면 이벤트 아예 삭제
    // 주최자 아니면 이벤트 멤버에서만 삭제
    if (user.getUserId() == event.getEventCreatorId()) {
        if (EventAPI.deleteEventById(event.getId()) > 0) {
            JOptionPane.showMessageDialog(null, "모든 캘린더에서 일정이
            삭제되었습니다. 새로고침 해보세요");
            eventDetailFrame.dispose();
        } else {
            JOptionPane.showMessageDialog(null, "일정 삭제 실패", "오류",
            JOptionPane.ERROR_MESSAGE);
        }
    }
    else {
        if
        (EventMemberAPI.deleteMemberByEventIdAndUserId(event.getId(),
        user.getUserId()) > 0) {
            JOptionPane.showMessageDialog(null, "나의 캘린더에서 일정이
            삭제되었습니다. 새로고침 해보세요");
        } else {
            JOptionPane.showMessageDialog(null, "일정 삭제 실패", "오류",
            JOptionPane.ERROR_MESSAGE);
            eventDetailFrame.dispose();
        }
    }
}
```

8) 일정 검색

캘린더 상단의 검색 버튼을 누르면 일정 이름이나 일정 설명으로 자신이 멤버로 있는 일정을 검색할 수 있도록 구현하였습니다.

```
// 이벤트 검색
public static ArrayList<Event> getEventListByEventNameOrEventDetail(String
eventName, String eventDetail, int userId) {
    ArrayList<Event> eventArrayList = new ArrayList<>();

    if (eventName.isEmpty() && !eventDetail.isEmpty()) {
        String query = "select * from event join eventmember on
event.event_id = eventmember.event_id where eventmember.user_id = ? and " +
"event.event_detail like ?";
        ...
    }

    } else if (!eventName.isEmpty() && eventDetail.isEmpty()) {
        String query = "select * from event join eventmember on
event.event_id = eventmember.event_id where eventmember.user_id = ? and " +
"event.event_name like ?";
        ...
    }
    } else if (!eventName.isEmpty() && !eventDetail.isEmpty()) {
        String query = "select * from event join eventmember on
event.event_id = eventmember.event_id where eventmember.user_id = ? and " +
"event.event_name like ? and event.event_datail like ?";
        ...
    }
    return null;
}
```

일정 이름과 설명을 각각 경우의 수로 나누어 sql 쿼리를 다르게 구성하여 해당 릴레이션을 찾을 수 있도록 구성하였습니다.

9) 월간, 주간, 일간 달력

월간 달력은 해당 월의 첫번째 날부터 마지막날까지 차례로 date 와 user_id 를 이용해 event 를 찾아 넣었고, 주간 달력은 해당하는 주의 첫날부터 마지막 날까지, 일간 달력은 해당 날짜의 date 와 user_id 를 이용해 event 를 찾아 달력에 이름을 넣어 구현하였습니다.

```
public static ArrayList<String> getEventNamesByDateAndUserId(java.sql.Date
date, int userId) {

    ArrayList<String> eventNames = new ArrayList<>();
    String query = "select distinct event.event_name from event join
eventmember on event.event_id = eventmember.event_id " +
"where eventmember.user_id = ? and (DATE(event.event_stime) <= ?
AND DATE(event.event_etime) >= ?)";

    try (Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
, "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
connection.prepareStatement(query)) {

        preparedStatement.setInt(1, userId);
        preparedStatement.setDate(2, date);
        preparedStatement.setDate(3, date);

        ResultSet resultSet = preparedStatement.executeQuery();
        while (resultSet.next()) {
            eventNames.add(resultSet.getString("event_name"));
        }
        return eventNames;
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}
```

- 리마인더 관리

1) 리마인더 설정

리마인더는 일정을 생성할 때 선택할 수 있도록 일정 생성화면에서 설정하였습니다. Interval 이 0 이 아닐경우에만 timeframefield 를 활성화 시켜 구현하였고, reminder 테이블을 정의하여 interval, reminder_start_time 을 저장해 리마인더가 활용될 수 있도록 하였습니다. 또한 설정된 첫 리마인더 시작이 현재시간의 1 시간 이내의 일정 알림이, 캘린더 상단의 Reminder 버튼을 누르면 확인할 수 있도록 하였습니다.

```
// event 에 대한 리마인더 date 와 비교해서 출력
public static Reminder getReminderByEventIdAndDate(int eventId, Date
date) {

    String query = "select * from reminder join event on event.event_id
= reminder.event_id where event.event_id = ? and " +
        "reminder.reminder_start_time <= ? and" +
        " reminder.reminder_start_time >= ?";

    try (Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/dbms_practice"
, "dbms_practice", "dbms_practice");
        PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        calendar.add(Calendar.MINUTE, 60);
        Date newDate = calendar.getTime();
        preparedStatement.setInt(1, eventId);
        preparedStatement.setTimestamp(2, new
Timestamp(newDate.getTime()));
        preparedStatement.setTimestamp(3, new
Timestamp(date.getTime()));

        ResultSet resultSet = preparedStatement.executeQuery();
        if (resultSet.next()) {
            return new Reminder(resultSet.getInt("reminder_id"),
resultSet.getInt("event_id"),
            new
Date(resultSet.getTimestamp("reminder_start_time").getTime()),
resultSet.getInt("interval"));
        }
        return null;
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}
```

2) 3) 팝업 메시지, 일정 시작 시 리마인더 삭제

Swing 의 Timer 를 활용하여 구현하였습니다. 프로그램을 실행하고 로그인을 마쳐 캘린더 화면이 구성되면, 그때부터 1 분간격으로 타이머를 발생시켜 reminder 를 확인하였습니다. 따라서 초단위 시간은 정확하지 않을 수 있지만 분단위의 reminder 는 잘 작동할 수 있도록 구현하였습니다.

```
private static void checkReminder(User user) {
    Date cTime = new Date();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");
    ArrayList<Integer> eventIdList =
EventMemberAPI.getEventIdListByUserId(user.getUserId());
    ArrayList<Reminder> reminderList = new ArrayList<>();

    for (int eventId : eventIdList) {
        reminderList.add(ReminderAPI.getReminderByEventId(eventId));
    }
    // 이벤트의 시작시간 확인해서 이미 지났으면 리마인더 삭제
    for (Reminder reminder : reminderList) {
        .....
        if (reminderEvent.getEventSTime().getTime() < cTime.getTime()) {
            ReminderAPI.deleteReminderByReminderId(reminder.getReminderId());
            continue;
        }
        if (sdf.format(reminder.getReminderStartTime()).equals(sdf.format(cTime))) {
            // 팝업 메시지 표시하고
            JOptionPane.showMessageDialog(null, "Lunch at " + reminderEvent.getEventName()
+ " is starting at " +
                sdf.format(reminderEvent.getEventSTime()), "Event
reminder",JOptionPane.INFORMATION_MESSAGE );
            // 간격+해서 이벤트 시작시간보다 작은지 비교해서
            Date calculatedTime = new Date(cTime.getTime() + 60 * 1000 *
reminder.getInterval());
            int i = 0;
            while (calculatedTime.getTime() < reminderEvent.getEventSTime().getTime()) {
                i += 1;
                calculatedTime = new Date(calculatedTime.getTime() + 60 * 1000 *
reminder.getInterval());
                Timer timer = new Timer(60 * 1000 * reminder.getInterval() * i, new
ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                        JOptionPane.showMessageDialog(null, "Lunch at " +
reminderEvent.getEventName() + " is starting at " +
                            sdf.format(reminderEvent.getEventSTime()), "Event
reminder",JOptionPane.INFORMATION_MESSAGE );
                    }
                });
                timer.setRepeats(false);
                timer.start();
            }
        }
    }
}
```

4 Opinion

처음 과제를 할당받았을 때에는 쉬운 과제라고 생각했습니다. 어떻게 구현해야할지 바로바로 생각이 났고 몇 번 해보았던 툴과 주제이기 때문입니다. 하지만 생각보다 어려운 부분이 많아 더 공부해야했습니다. 힘들었지만, 이 과정에서 얻은 것들이 많습니다. Swing을 처음 배워 사용해봤는데 생각보다 구현할 수 있는 부분이 많아 재미있었습니다. 특히, 새로운 툴을 배우고 바로바로 활용하며 익히는 것이 즐거웠습니다. 과제를 마무리하며, 앞으로는 데이터베이스를 활용해서 좀 더 신뢰성있고 안정적인 프로덕트를 만들고 운영해보고 싶다는 생각이 들었습니다. 코드도 가독성과 수정을 염두에 두어 잘 작성하기 위해 공부할 것이며, 이를 통해 진짜 내 가족이 사용할 수 있는 캘린더 프로덕트를 직접 만들어 보겠다는 다짐을 했습니다.