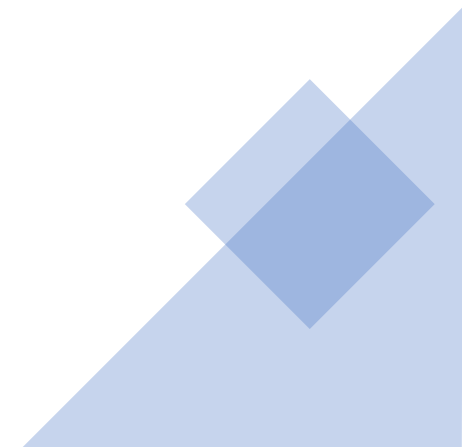


5월 9일 발표자료 - iOS

1871069 김진웅

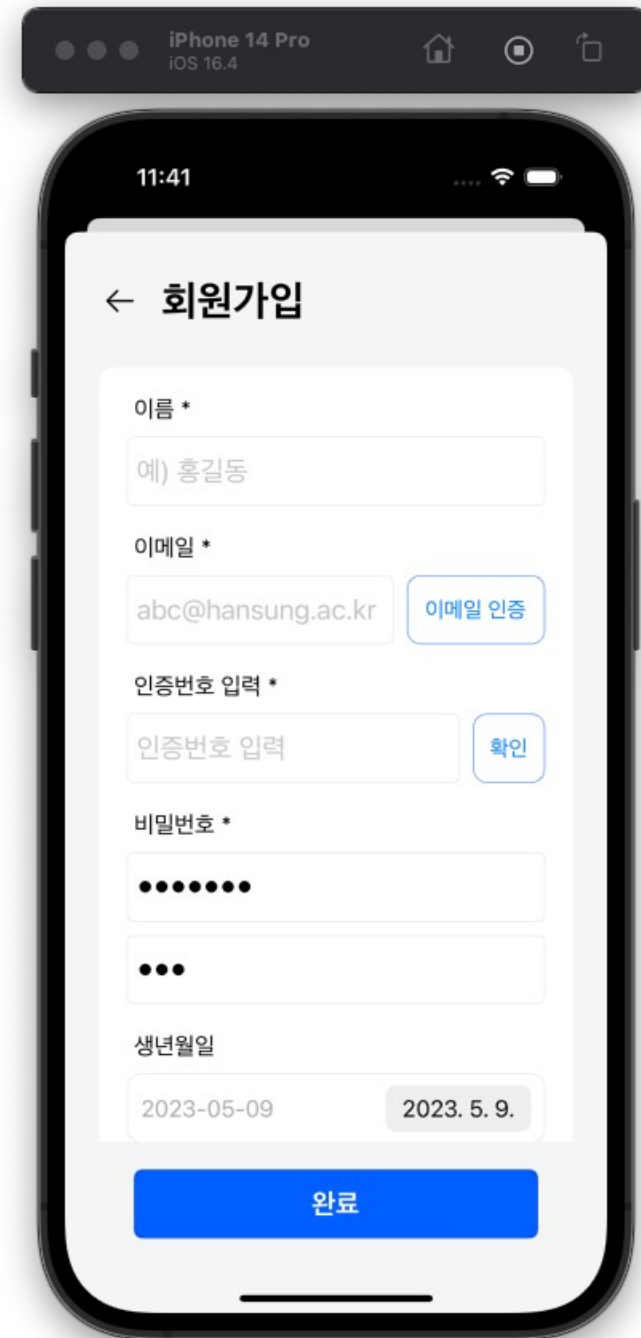


목차

- 1. 한 주간의 진행 상황
 - 회원가입 UI 재구성 및 로직 구현
 - 일정 UI 재구성 및 로직 구현
 - 2. 앞으로의 계획
- 

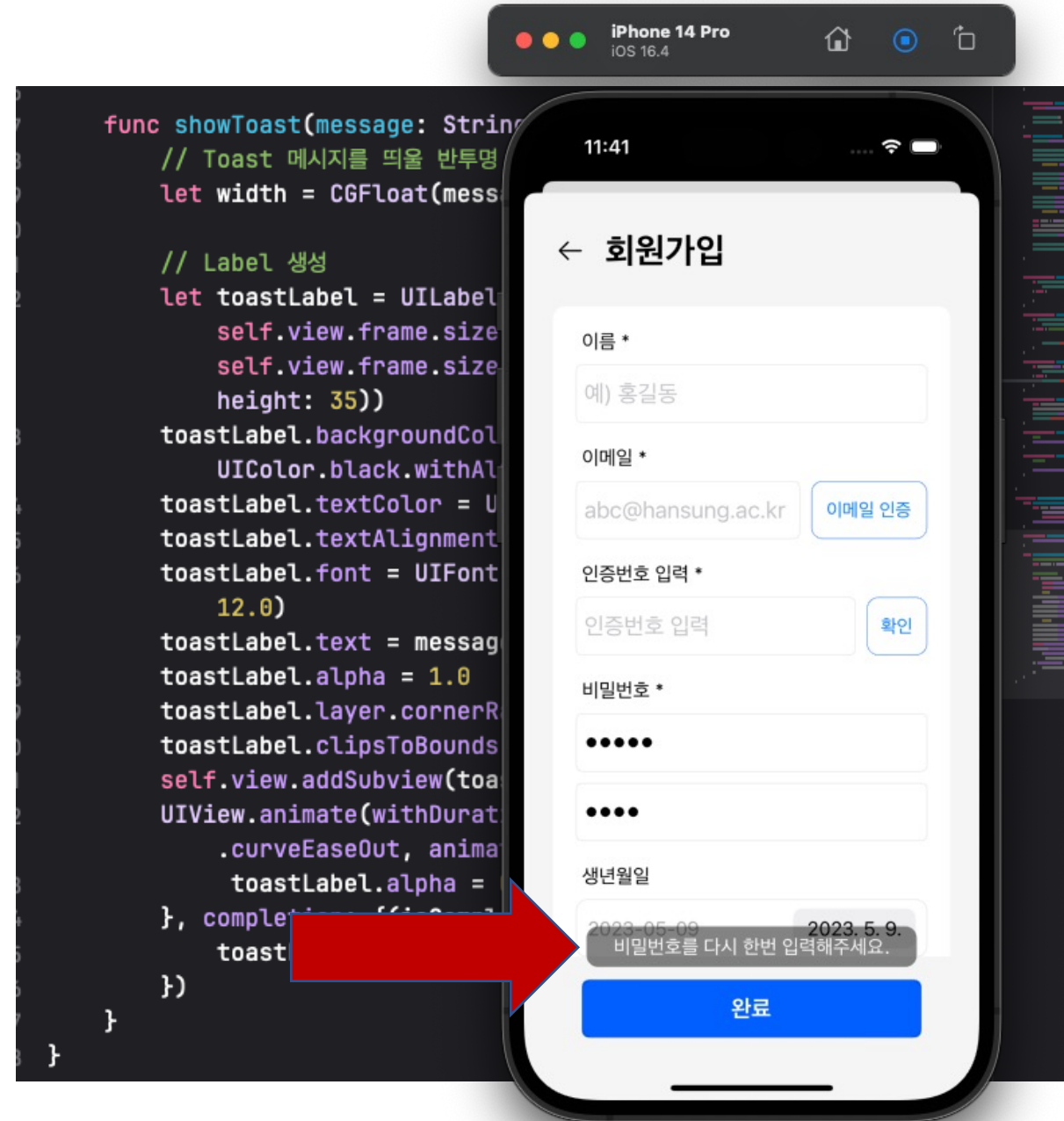
1. 한 주간의 진행상황

- 회원가입 UI 및 로직
 - 회원가입 UI를 Web과 비슷하게 함과 동시에
 - 각 버튼들과 TextField간 데이터 비교와 저장할 User 클래스를 선언하여
 - 서버에 nil값이 전달되지 않도록 하였음.



1. 한 주간의 진행상황

- 안드로이드의 Toast 메시지를 구현
 - 안드로이드에서 사용되는 토스트 메시지는 사용자에게 간단한 알림을 띄워주기 위해 사용하는데,
 - 앱을 사용하는 사용자들에게 있어 발생할 수 있는 입력 상에 오류 등을 알리기 위해 구현하였음.



1. 한 주간의 진행상황

- 토스트 메시지를 사용하여

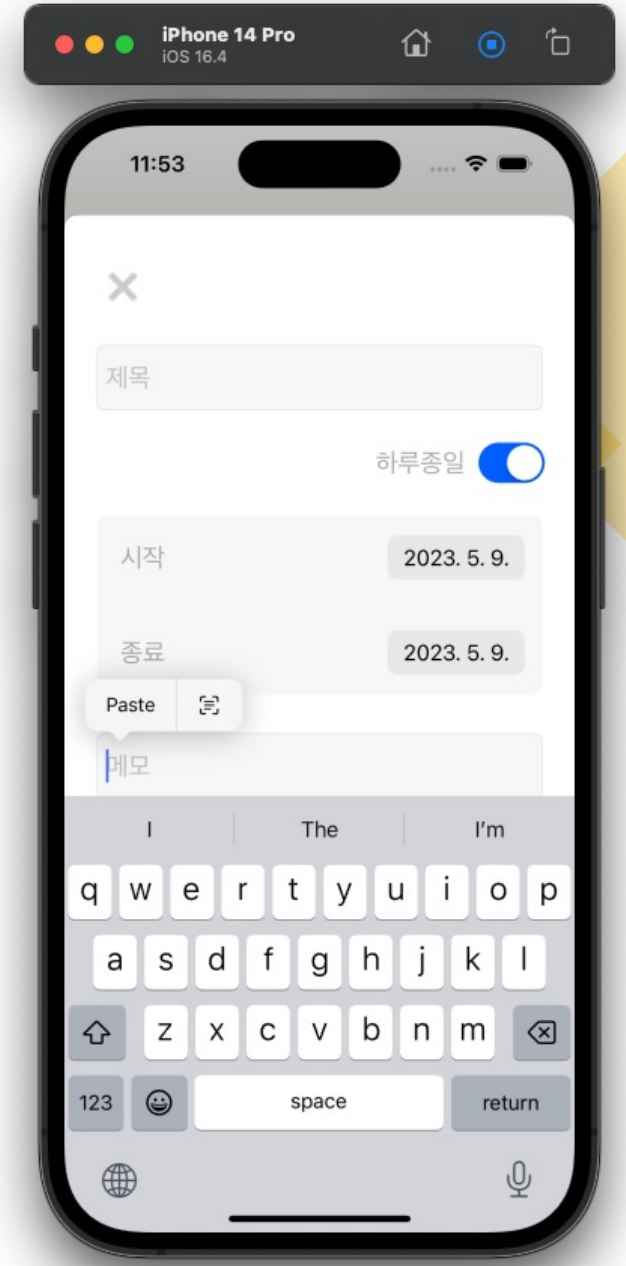
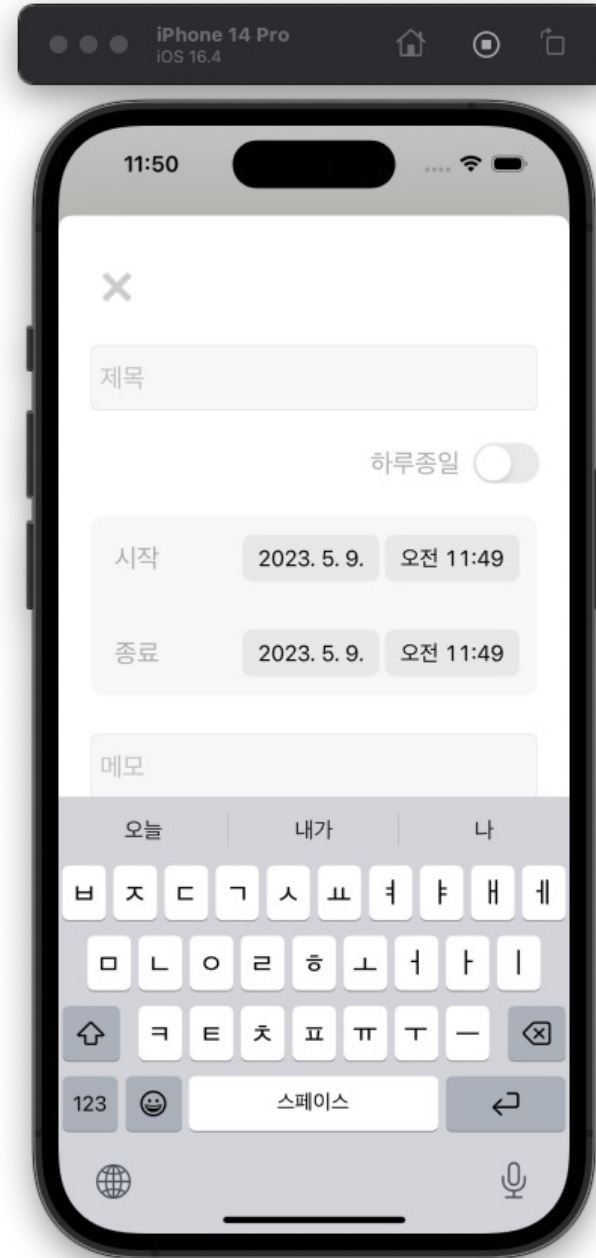
- 회원가입 창에서 사용자가 필수적으로 입력해야 할 (이름, 이메일, 인증번호, 비밀번호 등) 필드에 대해 알림을 준다.

- 일정, 그룹 등 앱 내의 전반적인 필수적인 부분에서 해당 함수를 통해 toast 메시지를 간접적으로 구현할 수 있기 때문에 활용 가능성이 높다.

1. 한 주간의 진행상황

- 일정 UI

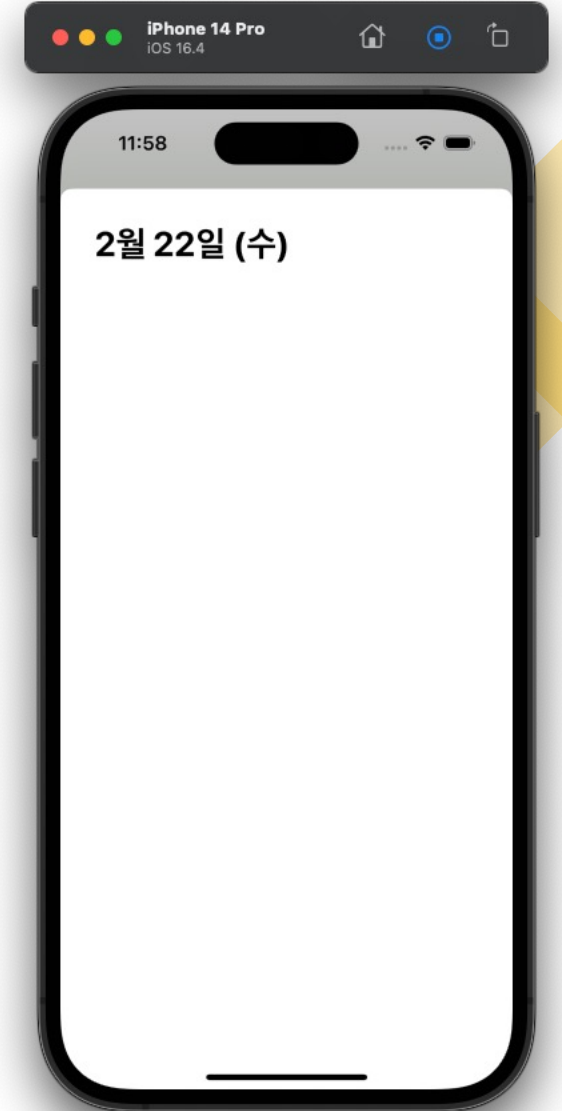
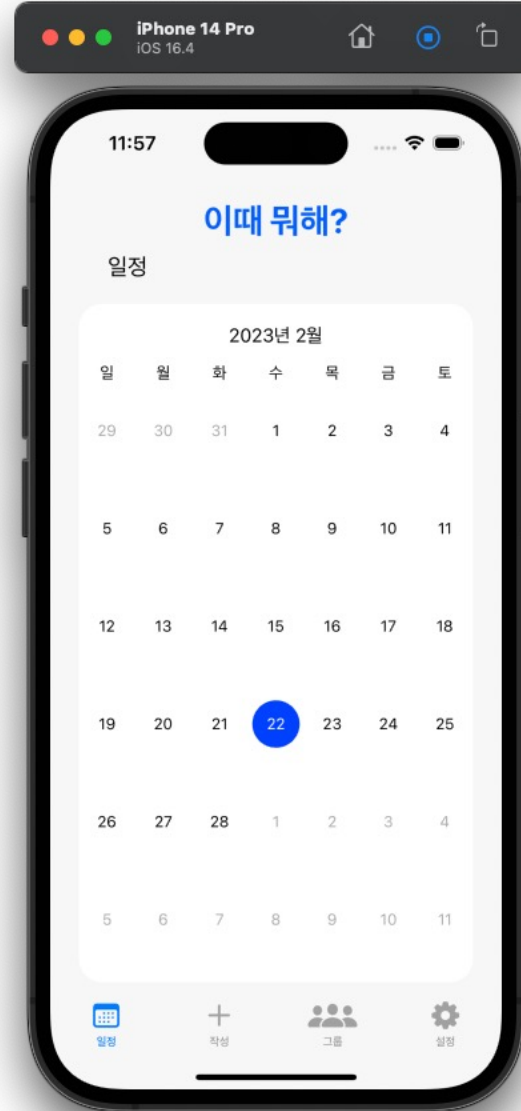
- 기존 시작 및 종료의 날짜 선택 부분을 Table View로 구현하여 UI 구현에 있어 난항을 겪고 있었으나,
- 해당을 StackView로 두번 묶고, DatePicker의 경우
- 하루 종일 토글에 대해 mode를 변경하는 식으로 구현하였음.



1. 한 주간의 진행상황

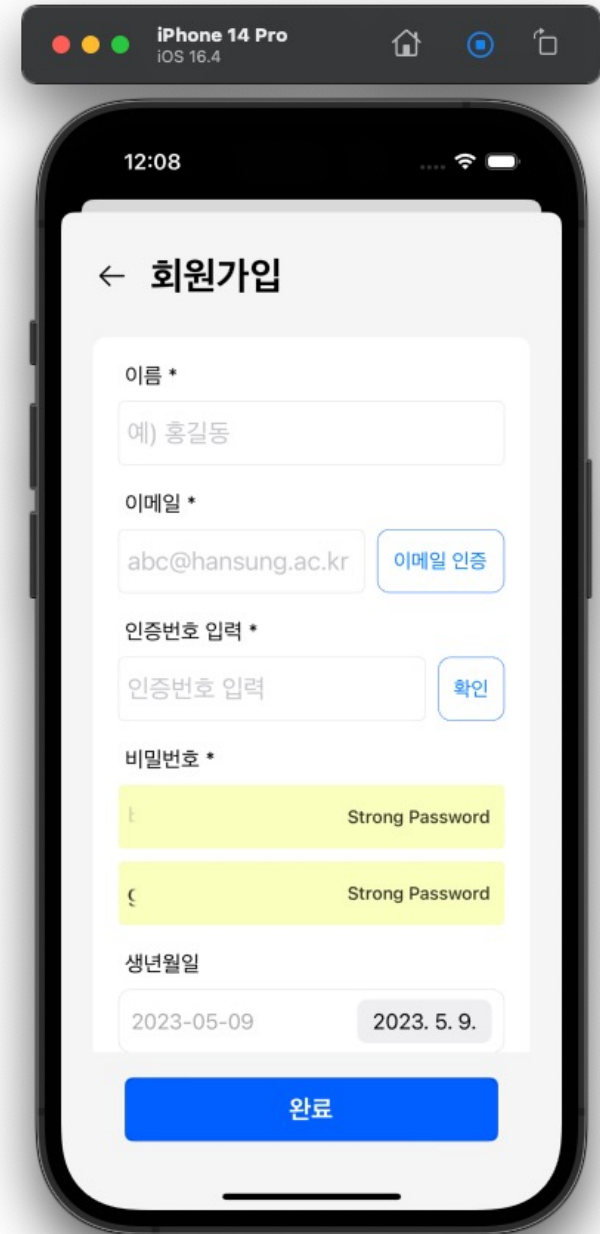
- 캘린더

- FSCalendar로 구현
 - 메인 뷰에서 세부적인 일정은 확인이 힘들기 때문에
- 날짜를 클릭 시에 올라오는 모달을 통해 해당 날짜의 일정을 확인하는 방법으로 구현
- 일정 리스트를 보여줄 View 구성을 아직 구현하지 못했음.



1. 한 주간의 진행상황

- Strong Password?
 - 회원가입 UI의 경우,
 - Secure text entry(비밀번호 창의 텍스트 입력을 보이지 않게 하는 것)을
 - 체크 시에 Strong Password로 변하면서
 - 어떠한 입력도 허용되지 않는 문제가 발생하였음.
 - 허나 이 부분은 textfield의 content type을 new password로 변경함으로써 문제를 해결하였음.



3. 앞으로의 계획

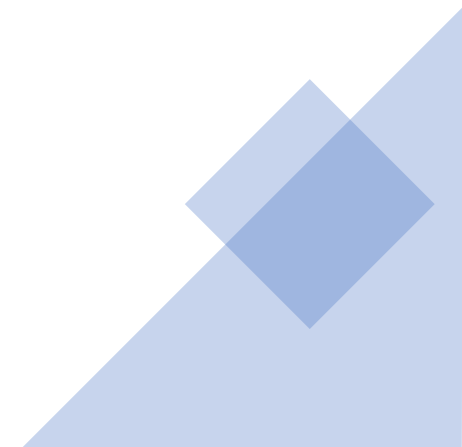
- User 인증 및 생성 프로토콜을 구현하고 확인하여 DB에 정확히 생성되는지 확인하기
- 일정 생성 및 수정을 구현하기

5월 9일 발표자료 - Web

1871062 김정한



목차

- 1. 한 주간의 진행 상황
 - 1-1. 어떠한 것을 공부했는가
 - 1-2. 부딪힌 이슈 & 해결사항
 - 2. 고민하고 있는 issue
 - 3. 앞으로의 계획
 - 4. 기타
- 

1. 한 주간의 진행 상황

- 1-1. 어떠한 것을 공부했는가

- FullCalendar Library를 사용해 캘린더 UI 구현

2023년 5월

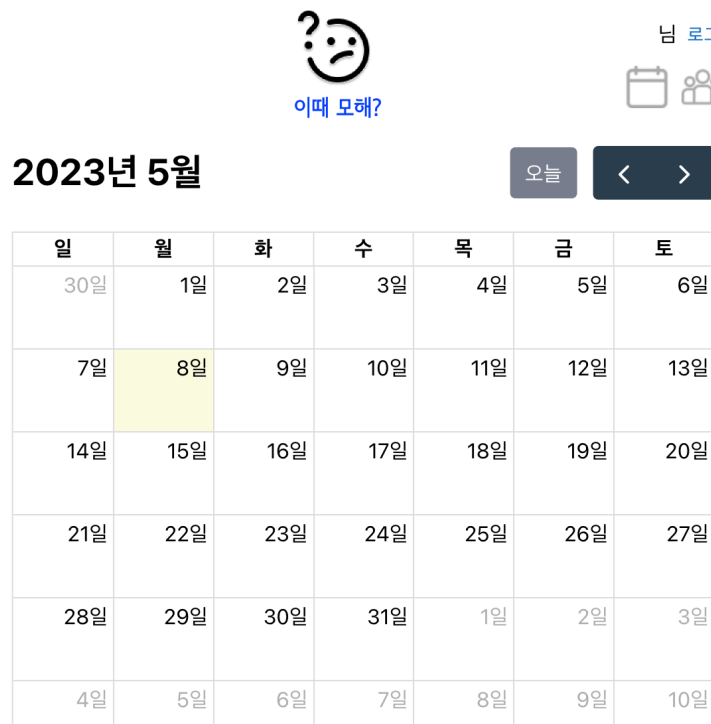
오늘 < >

일	월	화	수	목	금	토
30일	1일	2일	3일	4일	5일	6일
7일	8일	9일	10일	11일	12일	13일
14일	15일	16일	17일	18일	19일	20일
21일	22일	23일	24일	25일	26일	27일
28일	29일	30일	31일	1일	2일	3일

FullCalendar Library

1. 한 주간의 진행 상황

- 1-1. 어떠한 것을 공부했는가
 - 캘린더 헤더 구현중



님 로그아웃

Username 정보 띄우기,
로그아웃 기능 구현 완료

아이콘 클릭시 각 아이콘에 해당하는 페이지 렌더링 기능 구현중

1. 한 주간의 진행 상황

• 1-1. 어떠한 것을 공부했는가

- 로그인 시 캘린더에 사용자 이름 띄우기 구현 완료

```
// id,pw가 맞으면 로그인 성공, 토큰을 붙여서 다시 넘기기
{
  "userId": "terry8408@gmail.com",
  "password": null,
  "userName": null,
  "birth": null,
  "createdAt": null,
  "lastModifiedAt": null,
}
```

로그인 시 서버 프로토콜에서 보내준 토큰의 내용.
쿠키에 저장되어 있다.

```
const handleSubmit = async (event) => { // await을 사용하기 위해 async 함수로 선언
  event.preventDefault();

  try {
    const response = await axios.post('/auth/signin', { // axios는 항상 await과 함께 사용
      userId: emailInput,
      password: passwordInput
    });
    const { status, data } = response;
    cookie.set('token', data.token) //response의 data에 있는 token값 읽어오기
    cookie.set('username', response.data.userName) //response의 data에 있는 userName 쿠키에 저장
    navigate('/calendar')
  } catch (e) {
    window.alert("로그인에 실패했습니다. 아이디와 비밀번호를 다시 한번 확인해주세요!")
  }
};
```

Signin 시 서버에서 보내준 data
안의 userName 불러오기

```
const [username, setUsername] = useState('')
useEffect(() => {
  setUsername(cookie.get('username'))
}, [])

const navigate = useNavigate()
const handleClickLogout = (event) => {
  // TODO: 서버에 로그아웃 요청
  cookie.remove('token')
  cookie.remove('username')
  window.alert('로그아웃이 완료되었습니다!')
  navigate('/signin')
}
```

쿠키 안에 있는
username 정보 get

<p>{username}</p>

Username 정보 띄우기

1. 한 주간의 진행 상황

- 1-1. 어떠한 것을 공부했는가
 - 로그아웃 기능 구현 완료

localhost:3000 내용:

로그아웃이 완료되었습니다!

확인

로그아웃 버튼 클릭시 로그아웃 alert

```
const [username, setUsername] = useState('')
useEffect(() => {
  setUsername(cookie.get('username'))
}, [])

const navigate = useNavigate()
const handleClickLogout = (event) => {
  // TODO: 서버에 로그아웃 요청
  cookie.remove('token')
  cookie.remove('username')
  window.alert('로그아웃이 완료되었습니다!')
  navigate('/signin')
}
```

로그아웃 버튼 클릭시 실행되는
핸들러.
토큰과 불러왔던 username 정보 삭제.
(토큰만 삭제하는 것이기 때문에
실제 유저 정보는 삭제되지 않음.)

그 후 logout alert 작동, signin 페이지로 이동

1. 한 주간의 진행 상황

• 1-1. 어떠한 것을 공부했는가

- 등록된 일정 클릭시 완료 버튼 및 삭제 버튼 구현

```
const [editMode, setEditMode] = useState(false); //수정모달 띄울지 말지 결정해주는 상태
```

editMode가 true -> 수정모달이 띄워짐
editMode가 false -> 일정추가 모달이 띄워짐

```
{editMode && (  
  <Button  
    variant="contained"  
    fullWidth  
    sx={{ mt: 1, height: 45, fontSize: 16 }}  
    // onClick={handleDelete}  
  >  
    삭제  
  </Button>  
)}
```

editMode의 상태에 따라 삭제 버튼을
띄울지 말지 결정

Mobile app screenshot showing the '일정 등록' (Schedule Registration) modal. The modal is titled '일정 등록 ?' and contains fields for '이름' (Name), '하루종일' (All day) checkbox, '시작' (Start) and '종료' (End) times, '메모' (Memo), '색상' (Color), and '반복' (Repeat). The '저장' (Save) button is visible at the bottom.

빈 일정 클릭시

Mobile app screenshot showing the '일정 등록' (Schedule Registration) modal. The modal is titled '일정 등록 ?' and contains fields for '이름' (Name), '하루종일' (All day) checkbox, '시작' (Start) and '종료' (End) times, '메모' (Memo), '색상' (Color), and '반복' (Repeat). The '저장' (Save) and '삭제' (Delete) buttons are visible at the bottom.

등록된 일정 클릭시

1. 한 주간의 진행 상황

- 1-1. 어떠한 것을 공부했는가
 - 일정 추가 기능 구현

```
const handleSubmit = async (event) => {
  const format = (value) => {
    if (value < 10) {
      return '0' + value
    } else return value
  }

  const { start, end } = input

  const startAt = start.$y + '-' + format(start.$M + 1) + '-' + format(start.$D) + 'T' + format(start.$H) + ':' + format(start.$m)
  const endAt = end.$y + '-' + format(end.$M + 1) + '-' + format(end.$D) + 'T' + format(end.$H) + ':' + format(end.$m)

  const payload = {
    name: input.name,
    startAt: startAt,
    endAt: endAt,
    userId: '',
    memo: input.memo,
    notification: 0,
    allDayToggle: input.allDay === true ? "true" : "false"
  }

  // 수정 모드일 때는 (editMode) PUT
  // 생성할 때는 POST

  let response

  if (editMode) {
    response = await axios.put('/schedule', payload)
  } else {
    response = await axios.post('/schedule', payload)
  }

  if (response.data.status === 'succeed') {
    onSubmitSchedule(response.data.data)
  }
}
```

서버 프로토콜 양식에 맞게 날짜 데이터 새로 포맷

서버에 보내는 페이로드

editMode의 상태(일정 추가 상태인지 일정 수정 상태인지)에 따라 서버에 보내는 요청방식이 다름

응답 성공시 서버로부터 받은 데이터를 onSubmitSchedule 함수에 담음.

```
// 일정 생성 요청, POST
// Authorization에 signin시 발급받은 토큰
localhost:8080/schedule
{
  "name" : "밥먹기",
  "startAt" : "2023-03-17-19:11",
  "endAt" : "2023-03-17-19:11",
  "userId" : "abc@naver.com",
  "memo" : "오늘 저녁은 고기!",
  "notification" : 1, // true가 1, false가 0
  "allDayToggle" : "true",
}

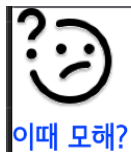
// 응답
{
  "data" : [
    {
      "originKey" : "어쩌구저쩌구",
      "name" : "밥먹기",
      "startAt" : "2023-03-17-19:11",
      "endAt" : "2023-03-17-19:11",
      "userId" : "abc@naver.com",
      "memo" : "오늘 저녁은 고기!",
      "notification" : 1,
      "allDayToggle" : 1,
      "createdAt" : "2023-03-17-19:11",
      "lastModifiedAt" : "2023-03-17-19:11"
    }
  ],
  "status": "succeed",
  "error" : null
}
```

서버 프로토콜 양식

1. 한 주간의 진행 상황

• 1-2. 부딪힌 이슈 & 해결사항

- 이때모해 아이콘이 가운데정렬 되지 않는 이슈



님 로그아웃

아이콘이 가운데정렬되지 않음

```
return (  
  <div style={{ display: 'flex', justifyContent: 'space-between' }}>  
    <img src={MainIcon} style={{ width: '8%' }} />  
  
    <div style={{ display: 'flex', flexDirection: 'column', alignItems: 'center' }}>  
      <div style={{ display: 'flex', alignItems: 'center' }}>  
        <p>{username} 님</p>  
        <Button variant="text" onClick={handleClickLogout}>로그아웃</Button>  
      </div>  
      <div style={{ backgroundColor: 'black', height: '30px', width: '200px' }} />  
    </div>  
  </div>  
)
```

오류 발생 코드

1. 한 주간의 진행 상황

• 1-2. 부딪힌 이슈 & 해결사항

- 이때모해 아이콘이 가운데정렬 되지 않는 이슈

```
return (  
  <div style={{ display: 'flex', justifyContent: 'space-between' }}>  
    <div></div>  
    <div></div>  
    <img src={MainIcon} style={{ width: '8%' }} />  
  
    <div style={{ display: 'flex', flexDirection: 'column', alignItems: 'center' }}>  
      <div style={{ display: 'flex', alignItems: 'center' }}>  
        <p>{username} 님</p>  
        <Button variant="text" onClick={handleClickLogout}>로그아웃</Button>  
      </div>  
      <div>  
        <img src={CalendarIcon} />  
      </div>  
    </div>  
  </div>  
)
```

빈 div 태그를 삽입하여 flex container 안의 영역을 더욱 더 세분화시켜 이미지를 가운데 영역으로 옮김

2. 고민하고 있는 이슈들

- 2-1.

- UI 먼저 구현 vs 기능 먼저 구현

- 기능을 먼저 구현 후 UI를 개선할지, UI부터 짜놓은 후 기능을 개발할지에 대한 고민

3. 앞으로의 계획

- 그룹 추가 기능 구현 예정
- 캘린더 UI 다듬기
- 헤더 아이콘 클릭시 페이지 렌더링 기능 구현 예정

5월 9일 발표자료 - Server

1811072 유영재

1871197 이윤재

The slide features decorative geometric shapes in the corners. The top-left corner has several overlapping yellow squares of various sizes. The bottom-right corner has overlapping blue squares. The main content area is white.

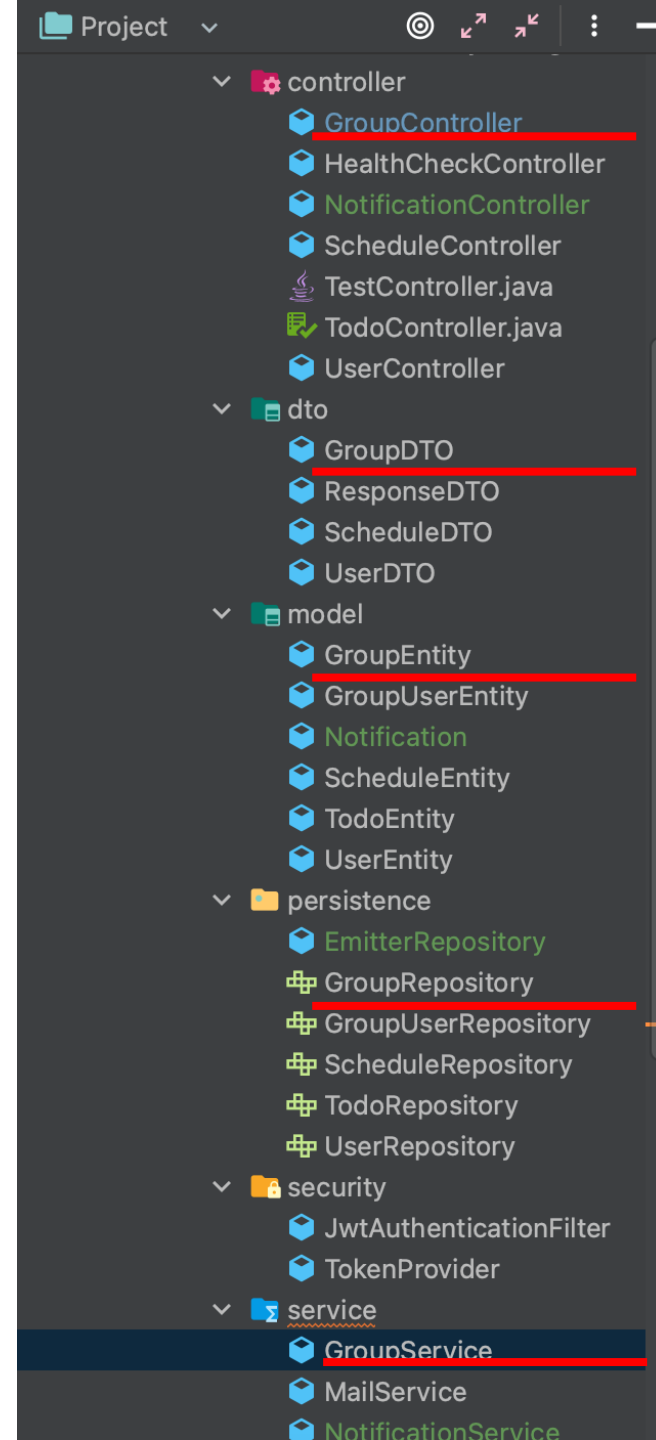
목차

- 1. 한 주간의 진행 상황
 - 1-1. 어떠한 것을 공부했는가
 - 1-2. 부딪힌 이슈 & 해결사항
- 2. 고민하고 있는 issue
- 3. 앞으로의 계획
- 4. 기타

1. 한 주간의 진행상황

• 1-1. 어떠한 것을 공부했는가

- Group CRUD 구현 완료
- CRUD를 구현하기 위해 Controller, Service, Persistence Layer 코딩
- Model을 구현할 Entity, DTO 코딩



1-2. 부딪힌 이슈 & 해결사항

- 1-2. 부딪힌 이슈 - Group/GroupUser Table (해결)

- Group 관련 로직을 구현하던 중, 로직을 정리하다 보니 GroupUser와 Group 테이블이 분리되어야 하는 궁극적인 목적을 생각해보게 됨.
- Group 에서의 CRUD
 - Create : 그룹 생성 - only 그룹장
 - Retrieve : 한명의 사용자가 자신이 속해 있는 모든 그룹을 검색해서 목록으로 출력
 - Update : 그룹 정보 수정 (그룹 이름, 그룹 설명) - only 그룹장
 - Delete : 그룹 삭제 - only 그룹장
- 하지만 이런 로직 말고도 GroupUser CRUD도 존재함.
 - Create : GroupUser 테이블 생성
 - Retrieve : 해당 그룹이 어떤 그룹인지 검색
 - Update : 구성원 변경
 - Delete : 구성원 강퇴
- 결론적으로, Group 테이블과 GroupUser 테이블은 따로 존재해야 함.

1-2. 부딪힌 이슈 & 해결사항

- 1-2. 부딪힌 이슈 - Group/GroupUser Table (해결)
 - 테이블은 따로 존재하지만, Service Layer나 Controller를 따로 만드는 건 괜한 혼란만 가져올 수 있으므로 Entity와 DTO는 따로 존재하되, Service와 Controller는 두 테이블의 메소드들을 합쳐 코딩
 - 또한 GroupUser 테이블은 User 테이블과 비슷하지만, User 테이블과는 별개로 그룹 안의 모든 사람들은 GroupUser 테이블을 가지게 됨.
 - DTO에서는 GroupUser들을 문자열로 받긴 하지만, DB에 넣을 때는 GroupUser로 따로 분리해서 넣음. GroupEntity에는 GroupUser를 String 형태로 집어넣는 코드 제외. -> GroupUser 테이블의 필요성 강조

2. 고민하고 있는 이슈들

2-1. Notification

- 그룹 일정 관련 로직을 구성하려면, 그룹장이 그룹원들에게 요청 알림을 보내는 것부터 시작하기 때문에 그룹 일정 CRUD를 들어가기 전에 Notification을 먼저 구현하기로 결정.
- 사용할 수 있는 방법은 여러가지 ..
 - SSE (Server Sent Event)
 - Web Socket
 - FCM(Firebase Cloud Messaging) .. 등등
- 여기서 SSE 방식 채택할 가능성 제일 높음. 서버가 클라이언트에게 단방향으로만 이벤트를 전달하는 방식.
- 우리는 요청 알림을 보내주기만 하면, 응답은 REST API로 받으면 되기 때문에 아무 상관이 없음.

3. 앞으로의 계획

- 3-1. 계획
 - Notification 이번주 내로 완료 목표
 - GroupUser CRUD, 그룹 일정 CRUD 동시 진행
 - 클라이언트와 지속적인 통신 테스트