Notification 받기

구현할 알림 기능

여러 채팅 방중에서
내가 현재 있는 채팅방이 아닌 곳에서
다른 사람이 채팅하면
하나의 쪽지당 하나의 Count가 올라가는 알림

Notification UI

```
<li
    key={room.id}
    onClick={() => changeChatRoom(room)}
    style={{
        backgroundColor: room.id === activeChatRoomId && "#ffffff45"
    }}
>
    # {room.name}
    <Badge style={{ float: 'right', marginTop: '4px' }} variant="danger">
        0
    </Badge>
</li>
```

알림 정보 만들기

먼저 notifications를
Array로 State를 만들기

```
notifications: []
```

채팅룸 정보를 가져올 때
알림 정보 리스너도 추가

```
AddChatRoomsListeners = () => {
    let chatRommsArray = []
    // chatRoomsRef.on(eventType, callback)
    this.state.chatRoomsRef.on("child_added", DataSnapshot => {
        chatRommsArray.push(DataSnapshot.val());
        this.setState({ chatRooms: chatRommsArray }, () => { this
        // DataSnapshot.key === chatRoomId
        this.addNotificationListener(DataSnapshot.key);
    })
}
```

채팅룸 아이디

messages Collection에서 해당 ChatRoom Id에 해당하는 정보를 가져오기

```
addNotificationListener = chatRoomId => {
    this.state.messagesRef.child(chatRoomId).on("value", DataSnapshot => {
        if (this.props.chatRoom) {

            this.handleNotifications(
                chatRoomId,
                this.props.chatRoom.id,  // 현재 채팅룸 아이디
                this.state.notifications,
                DataSnapshot
            )
        }
    })
}
```

https://firebase.google.com/docs/database/admin/retrieve-data#node.js_8

Read Event Types in Java and Node.js

Value

The `value` event is used to read a static snapshot of the contents at a given database path, as they existed at the time of the read event. It is triggered once with the initial data and again every time the data changes. The event callback is passed a snapshot containing all data at that location, including child data. In the code example above, `value` returned all of the blog posts in your app. Everytime a new blog post is added, the callback function will return all of the posts.

가져온 정보를 이용해서 notification 정보 생성.

먼저는 notifications 데이터가 없을 경우 !!!
=>
채팅방 하나 하나에 맞는 해당 알림 정보를 생성

```
if (index === -1) {
    notifications.push({
        id: chatRoomId,
        total: DataSnapshot.numChildren(),
        lastKnownTotal: DataSnapshot.numChildren(),
        count: 0
    })
}
```

id: 채팅방 아이디,
total: 해당 채팅방 전체 메시지 개수,
lastKnownTotal: 이전에 확인한 전체 메시지 개수,
count: 알림으로 사용 될 숫자 ,
DataSnapshot.numChildren(): 전체 children 개수
전체 메시지 개수

3 + 3 = 6 total

3 = lastKnownTotal

notifications state안에 해당 chatRoom 알림 데이터가

```
//이미 notifications안에 해당 chatRoom 데이터가 있을 때
else {
    //상대방이 채팅 보내는 그 해당 채팅방에 있지 않을 때
    if (chatRoomId !== currentChatRoomId) {
        //현재까지 유저가 확인한 총 메시지 개수
        lastTotal = notifications[index].lastKnownTotal;

        //count (알림으로 보여줄 숫자)를 구하기
        //현재 총 메시지 개수 - 이전에 확인한 총 메시지 개수 > 0
        //현재 총 메시지 개수가 10개이고 이전에 확인한 게 8개였다면 2개를 알림으로 보여줘야 함.
```

이미 들어 있을 때

```
        if (DataSnapshot.numChildren() - lastTotal > 0) {
            notifications[index].count = DataSnapshot.numChildren() - lastTotal;
        }
    }
    //total property에 현재 전체 메시지 개수를 넣어주기
    notifications[index].total = DataSnapshot.numChildren();
}
```

notification 숫자를 보여주기

```
(<Badge style={{ float: 'right', marginTop: '4px' }} variant="danger">
    {this.getNotificationCount(room)}
</Badge>)}
```

```
getNotificationCount = chatRoom => {
    let count = 0;

    this.state.notifications.forEach(notification => {
        if (notification.id === chatRoom.id) {
            count = notification.count;
        }
    });

    if (count > 0) return count;
};
```

notification 클릭해서 없애주기

```
changeChatRoom = (chatRoom) => {
    this.props.dispatch(setCurrentChatRoom(chatRoom));
    this.setState({ activeChatRoomId: chatRoom.id });
    this.props.dispatch(setPrivateChatRoom(false));
    this.clearNotifications();
}
```

```
clearNotifications = () => {
    let index = this.state.notifications.findIndex(
        notification => notification.id === this.props.chatRoom.id
    );

    if (index !== -1) {
        let updatedNotifications = [...this.state.notifications];
        updatedNotifications[index].lastKnownTotal = this.state.notifications[
```

```
            index
        ].total;
        updatedNotifications[index].count = 0;
        this.setState({ notifications: updatedNotifications });
    }
};
```

messageRef 정리해주기

```
componentWillUnmount() {
    this.state.chatRoomsRef.off();

    this.state.chatRooms.forEach(chatRoom => {
        this.state.messagesRef.child(chatRoom.id).off();
    });
}
```

유저 상태 정보

UI 만들기

```jsx
<Badge variant="danger" style={{
    height: '10px',
    width: '10px',
    float: 'right',
    marginTop: '8px'
}}> {" "}</Badge>
```

유저가 online인지 아닌지 아는 법

```
firebase.database().ref(".info/connected")
```

이것의 value가 true이면
온라인 상태입니다.

```jsx
this.state.connectedRef.on("value", DataSnapshot => {
    if (DataSnapshot.val() === true) {      You, a few seconds ag
        const ref = this.state.presenceRef.child(currentUserUid);
        ref.set(true);
        ref.onDisconnect().remove(err => {
            if (err !== null) {
                console.error(err);
            }
        });
    }
});
```

유저 상태 정보 저장

```
this.state.connectedRef.on("value", DataSnapshot => {
    if (DataSnapshot.val() === true) {
        const ref = this.state.presenceRef.child(currentUserUid);
        ref.set(true);
        ref.onDisconnect().remove(err => {
            if (err !== null) {
                console.error(err);
            }
        });
    }
});

this.state.presenceRef.on("child_added", DataSnapshot => {
    if (currentUserUid !== DataSnapshot.key) {
        this.addStatusToUser(DataSnapshot.key);
    }
});

this.state.presenceRef.on("child_removed", DataSnapshot => {
    if (currentUserUid !== DataSnapshot.key) {
        this.addStatusToUser(DataSnapshot.key, false);
    }
});
```

```
addStatusToUser = (userId, connected = true) => {
    const updatedUsers = this.state.users.reduce((acc, user) => {
        if (user.uid === userId) {
            user["status"] = `${connected ? "online" : "offline"}`;
        }
        return acc.concat(user);
    }, []);
    this.setState({ users: updatedUsers });
};
```

유저 상태 보여주기

```
isUserOnline = user => user.status === "online";
```

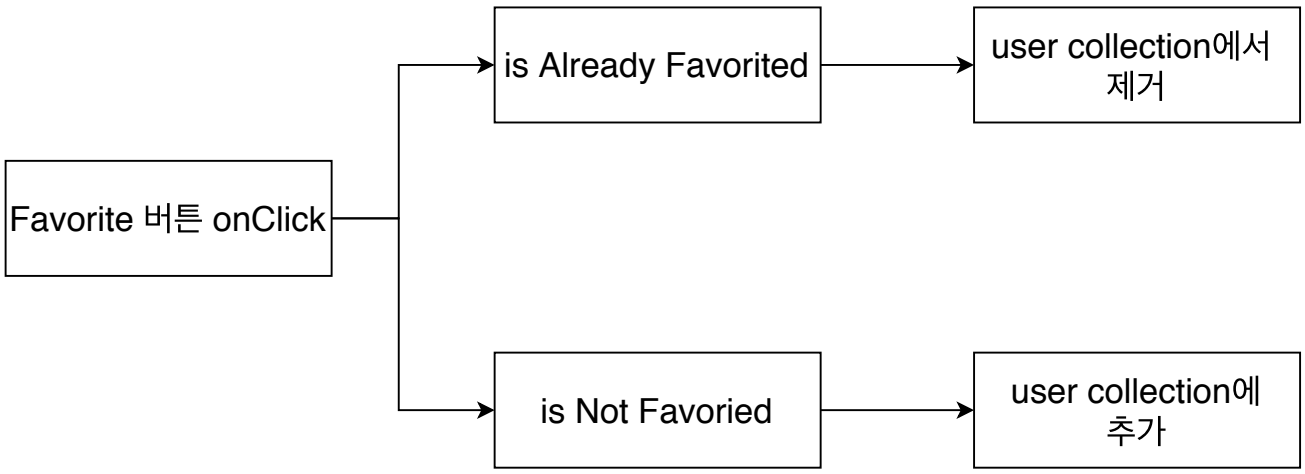
```
<Badge variant={this.isUserOnline(user) ? "info" : "danger"}
    style={{
        height: '10px',
        width: '10px',
```

```
        float: 'right',
        marginTop: '8px'
    }}
>
    {" "}
</Badge>
```

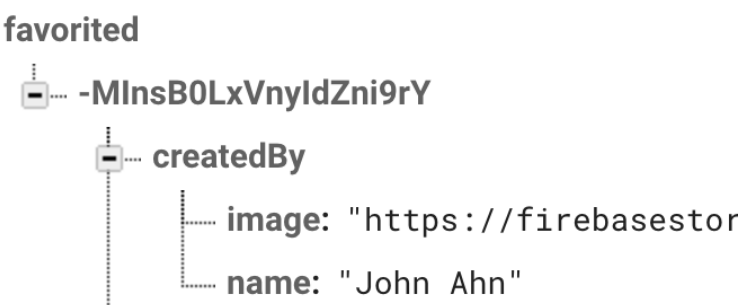
Favorite 버튼 UI & Favorite 정보 데이터베이스에 넣기

Favorite 클릭 버튼 UI

```jsx
{!isPrivateChatRoom &&
    <span style={{ cursor: 'pointer' }} onClick={handleFavorite}>
        {
            isFavorited ?
                <MdFavorite style={{ borderBottom: '10px' }} />
                :
                <MdFavoriteBorder style={{ borderBottom: '10px' }} />
        }
    </span>
}
```




```jsx
const handleFavorite = () => {
    if (isFavorited) {
        usersRef
            .child(`${user.uid}/favorited`)
            .child(chatRoom.id)
            .remove(err => {
                if (err !== null) {
                    console.error(err);
                }
            });
        setIsFavorited(prev => !prev)
    } else {
        usersRef
            .child(`${user.uid}/favorited`).update({
                [chatRoom.id]: {
                    name: chatRoom.name,
```



```
                    description: chatRoom.description,
                    createdBy: {
                        name: chatRoom.createdBy.name,
                        image: chatRoom.createdBy.image
                    }
                }
            });
        setIsFavorited(prev => !prev)
    }
};
```

description: "first chat room"

name: "first room"

Refresh 해도 Favorited가 남아있게하기

```
useEffect(() => {
    if (chatRoom && user) {
        console.log('chatROom user', chatRoom)
        console.log('user user', user)
        addFavoriteListener(chatRoom.id, user.uid)
    }
}, [])

const addFavoriteListener = (chatRoomId, userId) => {
    usersRef
        .child(userId)
        .child("favorited")
        .once("value")
        .then(data => {
            if (data.val() !== null) {
                const chatRoomIds = Object.keys(data.val());
                const isAlreadyFavorited = chatRoomIds.includes(chatRoomId);
                setIsFavorited(isAlreadyFavorited)
            }
        });
};
```

Favorite 리스트에 추가하기

Favorite 리스트 UI

```
return (
    <>
        <span>
            FAVORITED ({favoritedChatRooms.length})
        </span>
        <ul style={{ listStyleType: 'none', padding: '0' }}>
            {renderFavoritedChatRooms(favoritedChatRooms)}
        </ul>
    </>
)
```

```
const renderFavoritedChatRooms = (favoritedChatRooms) =>
    favoritedChatRooms.length > 0 &&
    favoritedChatRooms.map(chatRoom => (
        <li
            key={chatRoom.id}
        >
            # {chatRoom.name}
        </li >
    ));
```

Favorite 리스트
리스너 추가하기

```
useEffect(() => {
    if (currentUser) {
        addListeners(currentUser.uid);
    }
}, [])

const addListeners = userId => {
    usersRef
        .child(userId)
        .child("favorited")
        .on("child_added", DataSnapshot => {
            const favoritedChatRoom = { id: DataSnapshot.key, ...DataSnapshot.val() };
            setFavoritedChatRooms(prev => prev.concat(favoritedChatRoom))
        });

    usersRef
        .child(userId)
        .child("favorited")
        .on("child_removed", DataSnapshot => {
            const chatRoomToRemove = { id: DataSnapshot.key, ...DataSnapshot.val() };
            const filteredChatRooms = favoritedChatRooms.filter(chatRoom => {
```

```
                return chatRoom.id !== chatRoomToRemove.id;
            });
            setFavoritedChatRooms(filteredChatRooms);
        });
}
```

Favorited 채팅방 클릭 시 해당 방으로 이동

chatRoom 컴포넌트에서 한것과 같게 하면 됩니다.

chatRoom.js

```
changeChatRoom = (chatRoom) => {
    this.props.dispatch(setCurrentChatRoom(chatRoom));
    this.setState({ activeChatRoomId: chatRoom.id });
    this.props.dispatch(setPrivateChatRoom(false));
    this.clearNotifications();
}
```

Favorited.js

```
const changeChatRoom = (chatRoom) => {
    dispatch(setCurrentChatRoom(chatRoom));
    setActiveChatRoomId(chatRoom.id);
    dispatch(setPrivateChatRoom(false));
}
```

rendering part

```
<li
    key={chatRoom.id}
    onClick={changeChatRoom(chatRoom)}
    style={{
        backgroundColor: chatRoom.id === activeChatRoomId && "#ffffff4
    }}
>
    # {chatRoom.name}
</li >
```

Favorited를 위한 리스너 제거하기

```
componentWillUnmount() {
    if (this.props.currentUser) {
        this.removeListener();
    }
}

removeListener = () => {
    this.state.usersRef
        .child(`${this.props.currentUser.uid}/favorited`).off();
};
```