Typing 시작 시 Typing 정보 데이터베이스에 저장

Typing을 시작할 때
Typing 정보 DB로

```
<Form.Control
    onKeyDown={handleKeyDown}
    value={content}
    onChange={handleChange}
    as="textarea"
    rows={3}
/>
```

```
const handleKeyDown = event => {
    if (event.ctrlKey && event.keyCode === 13) {
        handleSubmit();
    }

    if (content) {
        typingRef
            .child(chatRoom.id)
            .child(user.uid)
            .set(user.displayName);
    } else {
        typingRef
            .child(chatRoom.id)
            .child(user.uid)
            .remove();
    }
};
```

```
typing
    -MInsB0LxVnyIdZni9rY
        NJgOrGQE3SVC4G10QwdWHE2AU8j2: "John Ahn"
```

```
try {
    await messagesRef
        .child(chatRoom.id)
        .push()
        .set(createMessage())
```

채팅을 Submit 하면
Typing 정보
데이터베이스에서 지우기

```
    .set(createMessage());

typingRef
    .child(chatRoom.id)
    .child(user.uid)
    .remove();

setErrors([])
setContent("")
setLoading(false)
```

리스너를 이용하여서 Typing 정보를 가져오기

데이터베이스에
Typing 정보가 들어오면
리스너로 그 정보 가져오기

```
componentDidMount() {
    const { chatRoom } = this.props;
    if (chatRoom) {
        this.addMessagesListeners(chatRoom.id);
        this.addTypingListeners(chatRoom.id);
    }
}
```

가져올 때
나의 Typing은 제외

```
addTypingListeners = (chatRoomId) => {
    let typingUsers = [];
    this.state.typingRef.child(chatRoomId).on("child_added",
        DataSnapshot => {
            if (DataSnapshot.key !== this.props.user.uid) {
                typingUsers = typingUsers.concat({
                    id: DataSnapshot.key,
                    name: DataSnapshot.val()
                });
                this.setState({ typingUsers });
            }
        });
}
```

Typing정보가
데이터베이스에서 제거되면
State에서도 지워주기

```
this.state.typingRef.child(chatRoomId).on("child_removed",
    DataSnapshot => {
        const index = typingUsers.findIndex(user => user.id === DataSnapshot.key);
        if (index !== -1) {
            typingUsers = typingUsers.filter(user => user.id !== DataSnapshot.key);
            this.setState({ typingUsers });
        }
    });
```

```
this.state.connectedRef.on("value", DataSnapshot => {
    if (DataSnapshot.val() === true) {
        this.state.typingRef
```

Typing 하던 유저가
로그아웃하면

```
            .child(chatRoomId)
            .child(this.props.user.uid)
            .onDisconnect()
            .remove(err => {
                if (err !== null) {
                    console.error(err);
                }
            });
    }
});
```

Typing UI 추가하기

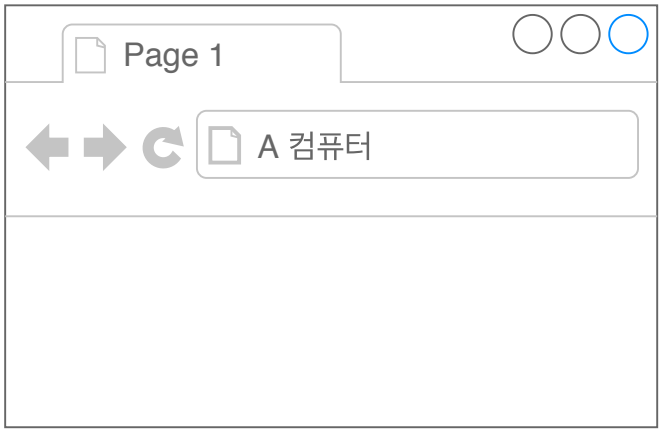
UI

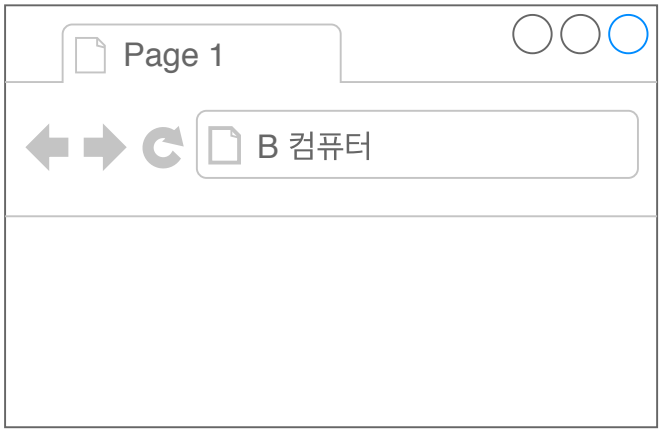
```
<div style={{
    width: '100%',
    height: '450px',
    border: '.2rem solid #ececec',
    borderRadius: '4px',
    padding: '1rem',
    marginBottom: '2rem',
    overflowY: 'auto'
}}>
    {searchTerm
        ? this.displayMessages(searchResults)
        : this.displayMessages(messages)}
    {this.renderTypingUsers(typingUsers)}
</div>
```

TypingUser STATE
데이터 처리

```
renderTypingUsers = (typingUsers) =>
    typingUsers.length > 0 &&
    typingUsers.map(user => (
        <span>{user.name}님이 채팅을 입력하고 있습니다...</span>
    ))
```

Connection Typing 리스너 제거





B가 로그아웃하고
A가 글을 쓰면 ?

```
▶Warning: Can't perform a React state                index.js:1
update on an unmounted component. This is a no-op,
but it indicates a memory leak in your application.
To fix, cancel all subscriptions and asynchronous
tasks in the componentWillUnmount method.
    in MainPanel (created by ConnectFunction)
    in ConnectFunction (at ChatPage.js:18)
    in div (at ChatPage.js:17)
    in div (at ChatPage.js:10)
    in ChatPage (created by Context.Consumer)
```

해결방법

```
componentWillUnmount() {
    this.state.messagesRef.off();
    this.state.connectedRef.off();
}
```

B가 글을 쓰다가
로그 아웃하고
A가 글을 쓰면?

```
▶Warning: Can't perform a React state update on an        i
unmounted component. This is a no-op, but it indicates a m
in your application. To fix, cancel all subscriptions and
asynchronous tasks in the componentWillUnmount method.
    in MainPanel (created by ConnectFunction)
    in ConnectFunction (at ChatPage.js:18)
    in div (at ChatPage.js:17)
    in div (at ChatPage.js:10)
    in ChatPage (created by Context.Consumer)
```

해결방법

```
this.addToListenerLists(chatRoomId, this.state.typingRef, "child_added");
```

```
addToListenerLists = (id, ref, event) => {
    const index = this.state.listenerLists.findIndex(listener => {
        return (
            listener.id === id &&
            listener.ref === ref &&
            listener.event === event
        );
    });

    if (index === -1) {
        const newListener = { id, ref, event };
        this.setState({
            listenerLists: this.state.listenerLists.concat(newListener)
        });
    }
};
```

```
componentWillUnmount() {
    this.state.messagesRef.off();
    this.state.connectedRef.off();
    this.removeListeners(this.state.listenerLists);
}
```

```
removeListeners = listeners => {
    listeners.forEach(listener => {
        listener.ref.child(listener.id).off(listener.event);
    });
};
```

```
event: "child_added"
id: "-MInsB0LxVnyIdZni9rY"
ref: Reference
  orderByCalled_: false
  path: Path {pieces_: Array(1), pieceNum_: 0}
```

```
▶ path: Path {pieces_: Array(1), pieceNum_: 0}
▶ queryParams_: QueryParams {limitSet_: false,
▶ repo: Repo {repoInfo_: RepoInfo, app: Firebas
  database: (...)
```

Listens for data changes at a particular location.

This is the primary way to read data from a Database. Your callback will be triggered for the initial data and again whenever the data changes. Use `off( )` to stop receiving updates. See Retrieve Data on the Web for more details.

This event will trigger once with the initial data stored at this location, and then trigger again each time the data changes. The `DataSnapshot` passed to the callback will be for the location at which `on()` was called. It won't trigger until the entire contents has been synchronized. If the location has no data, it will be triggered with an empty `DataSnapshot` (`val()` will return `null`).

This event will be triggered once for each initial child at this location, and it will be triggered again every time a new child is added. The `DataSnapshot` passed into the callback will reflect the data for the relevant child. For ordering purposes, it is passed a second argument which is a string

This event will be triggered once every time a child is removed. The `DataSnapshot` passed into the callback will be the old data for the child that was removed. A child will get removed when either:

- a client explicitly calls `remove()` on that child or one of its ancestors
- a client calls `set(null)` on that child or one of its ancestors
- that child has all of its children removed
- there is a query in effect which now filters out the child (because it's sort order changed or the max limit was hit)

This event will be triggered when the data stored in a child (or any of its descendants) changes. Note that a single `child_changed` event may represent multiple changes to the child. The `DataSnapshot` passed to the callback will contain the new child contents. For ordering purposes, the callback is also passed a second argument which is a string containing the key of the previous sibling child by sort order, or `null` if it is the first child.

```
**Handle a new value:**
```javascript
ref.on('value', function(dataSnapshot) {
...
});
```

@example

Handle a new child:

```
ref.on('child_added', function(childSnapshot, prevChildKey) {
...
});
```

One of the following strings: "value", "child_added", "child_changed", "child_removed", or "child_moved."

@param `callback`
A callback that fires when the specified event occurs. The callback will be passed a DataSnapshot. For ordering purposes, "child_added", "child_changed", and "child_moved" will also be passed a string containing the key of the previous child, by sort order, or `null` if it is the first child.

@param `cancelCallbackOrContext`
An optional callback that will be notified if your event subscription is ever canceled because your client does not have permission to read this data (or it had permission but has now lost it). This callback will be passed an `Error` object indicating why the failure occurred.

@param `context`
If provided, this object will be used as `this` when calling your callback(s).
```