

# Voice Dev 3

## CS571: Building User Interfaces

**Cole Nelson**

# Today's Warmup

- Clone [today's code](#) to your machine.
  - Run the command `npm install` inside of the `starter` and `solution` folders.
- Create and train the following intents in [wit.ai](#)...
  - `get_comments` e.g. "get 4 comments"
  - `login` e.g. "log me in"
  - `logout` e.g. "log me out"
  - `create_comment` e.g. "create a comment"

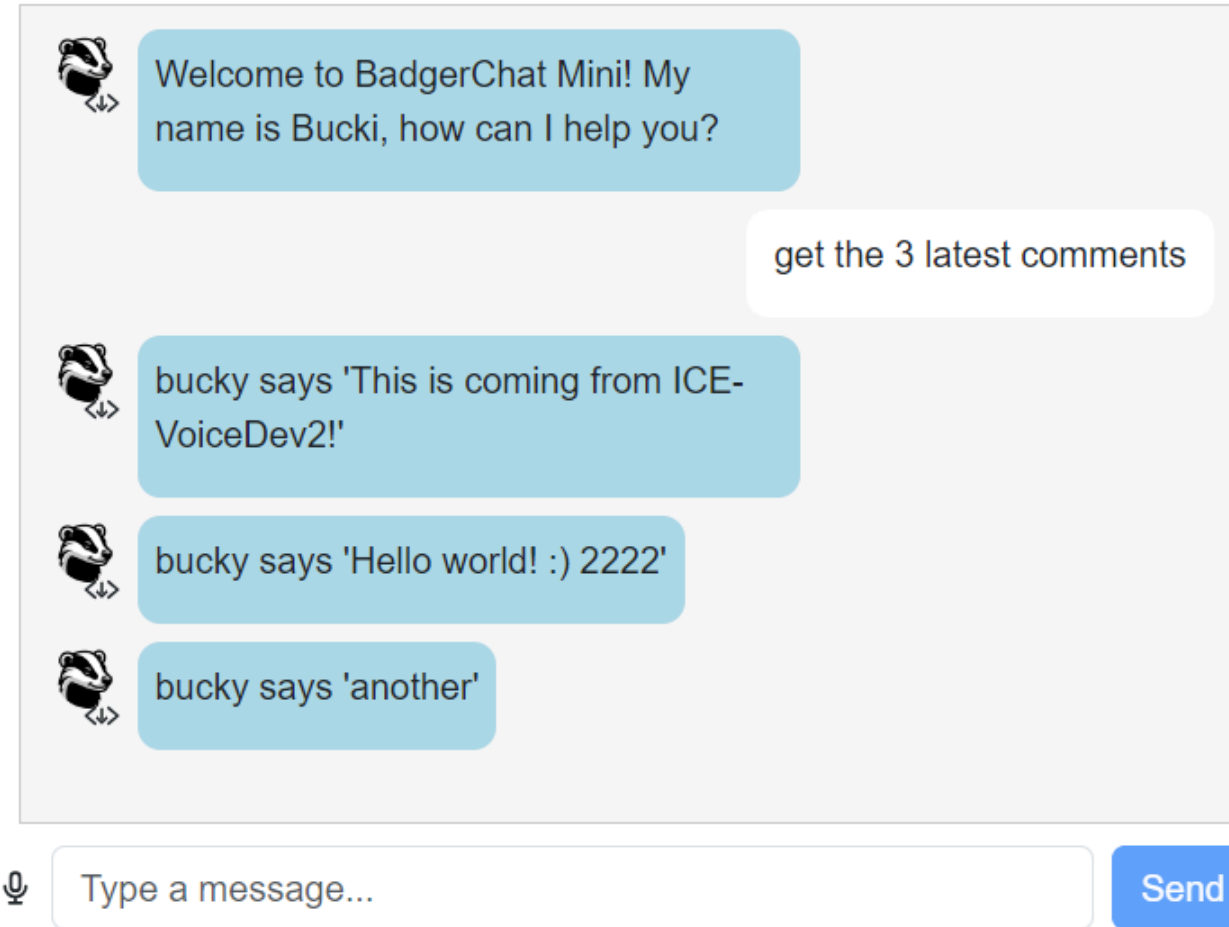
# Learning Objectives

1. Be able to keep the context of the conversation.
2. Be able to delegate to subagents.
3. Be able to appreciate other neat features, such as traits, speech-to-text translation, and text-to-speech synthesis.

# Your Turn!

Implement  
`get_messages` for  
BadgerChat Mini!

Here's the web version  
you built...



# Looking Ahead...

We still need to implement `login` and `create_comment`, but how would these be expressed?

"Log me in with username **ucky124** and password **mp@ssw0rd**." ← seems silly!

There is no `Wit.AI entity` to handle this, usernames and passwords are unpredictable!

# Possible Solution

## Step out of Wit.AI for a moment!

When the `login` intent has been triggered, ask the user for their username and set the `stage` to `"FOLLOWUP_USERNAME"`

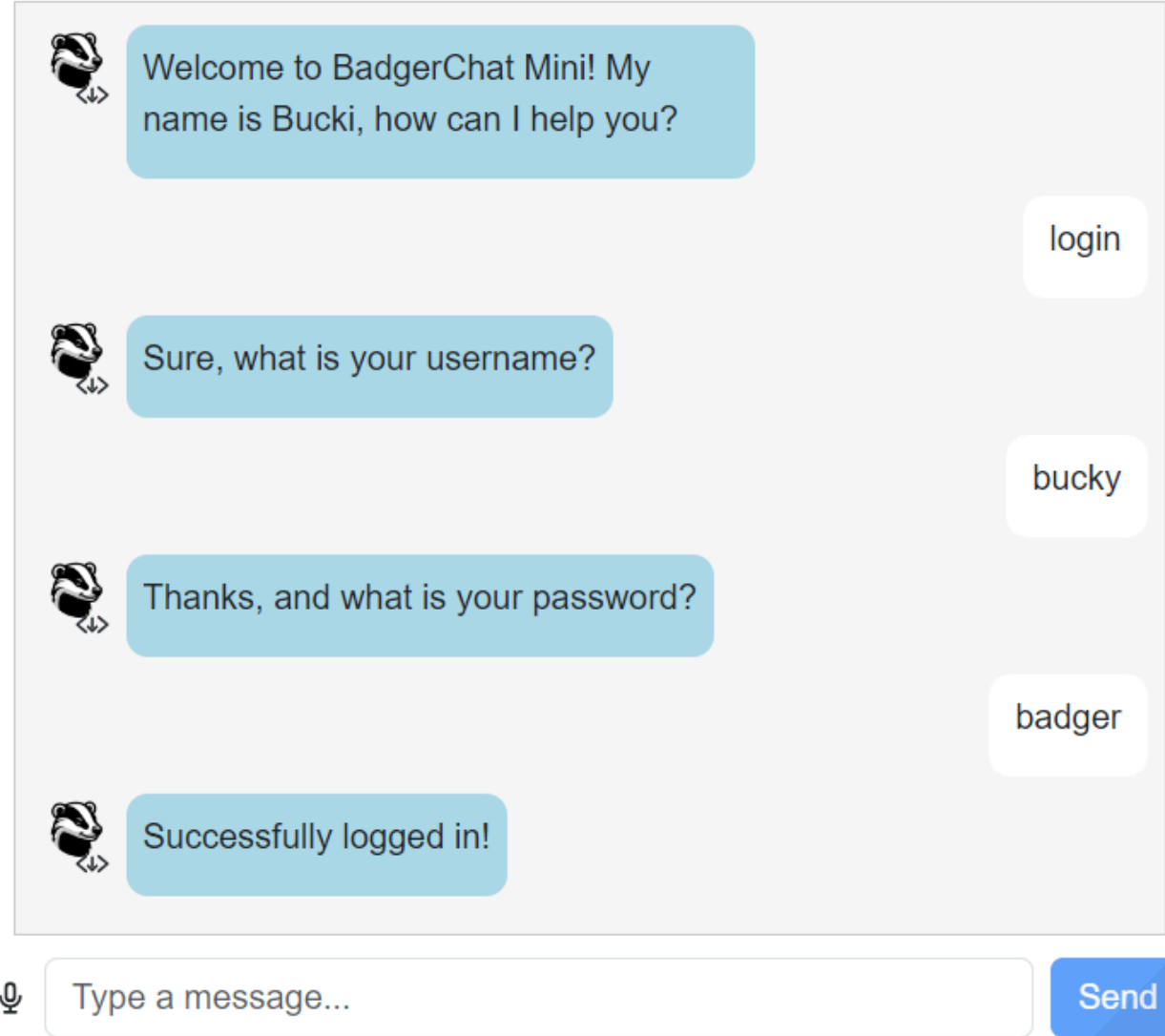
On the next `handleReceive`, ask for their password and set the stage to `"FOLLOWUP_PASSWORD"`

Finally, on the next `handleReceive`, attempt to login.

# Your Turn!

Implement `login` for  
BadgerChat Mini!

- bucky,badger
- pete608,gopioneers!
- gophy77,booooo



A screenshot of the BadgerChat Mini chat interface. It features a light gray background with a series of blue speech bubbles containing messages from a character named Bucki. The messages are: "Welcome to BadgerChat Mini! My name is Bucki, how can I help you?", "Sure, what is your username?", "Thanks, and what is your password?", and "Successfully logged in!". To the right of the chat area, there are three white input fields with rounded corners, each containing a username: "login", "bucky", and "badger". At the bottom of the interface, there is a white input field with a microphone icon on the left and the placeholder text "Type a message...". To the right of this field is a blue button with the text "Send".

Welcome to BadgerChat Mini! My name is Bucki, how can I help you?

login

Sure, what is your username?

bucky

Thanks, and what is your password?

badger

Successfully logged in!

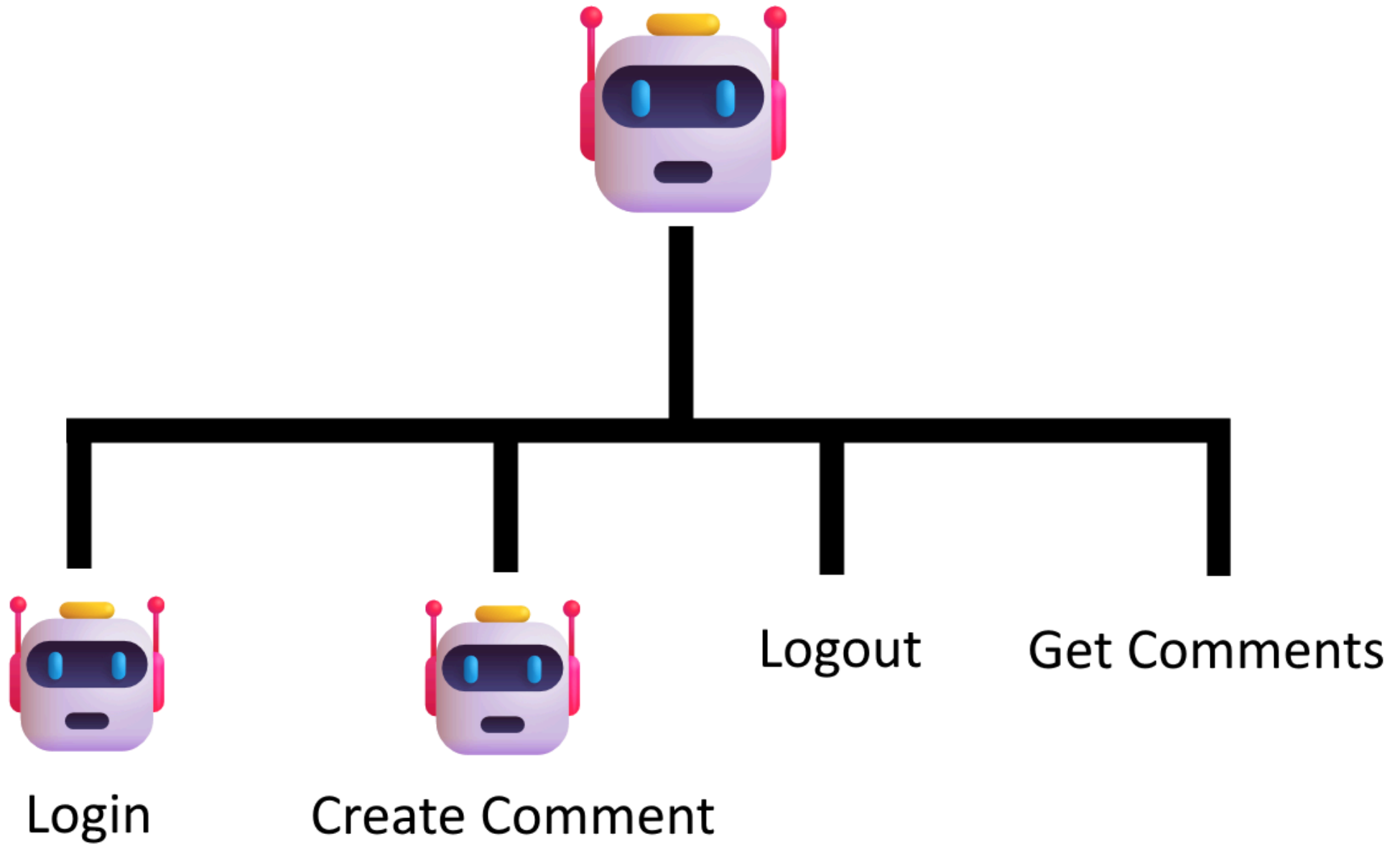
Type a message...

Send

```
const createChatAgent = () => {  
  const CS571_WITAI_ACCESS_TOKEN = "...";  
  
  // We may have a lot to keep track of!  
  let stage;  
  let comment, commentConfirm;  
  let loginUsername, loginPassword;  
  
  const handleInitialize = async () => {  
    return "Welcome to BadgerChat Mini";  
  }  
  
  // ...  
}  
export default createChatAgent;
```

Having many intents → Having many context variables





## In ChatAgent.js ....

```
const createChatAgent = () => {  
  
  const delegator = createChatDelegator();  
  
  const handleReceive = async (prompt) => {  
    if (delegator.hasDelegate()) { return delegator.handleDelegation(prompt); }  
    // ...  
  }  
  
  // ...  
  
  const handleLogin = async (promptData) => {  
    return await delegator.beginDelegation("LOGIN", promptData);  
  }  
  
  const handleCreateComment = async (promptData) => {  
    return await delegator.beginDelegation("CREATE", promptData);  
  }  
  
  // ...  
}
```

In ChatDelegator.js ....

```
const createChatDelegator = () => {  
  let delegate;  
  
  const DELEGATE_MAP = {  
    "LOGIN": createLoginSubAgent,  
    "CREATE": createCommentSubAgent  
  }  
  
  // ...  
}
```

... with functions to begin and end delegation.

## In LoginSubAgent.js ...

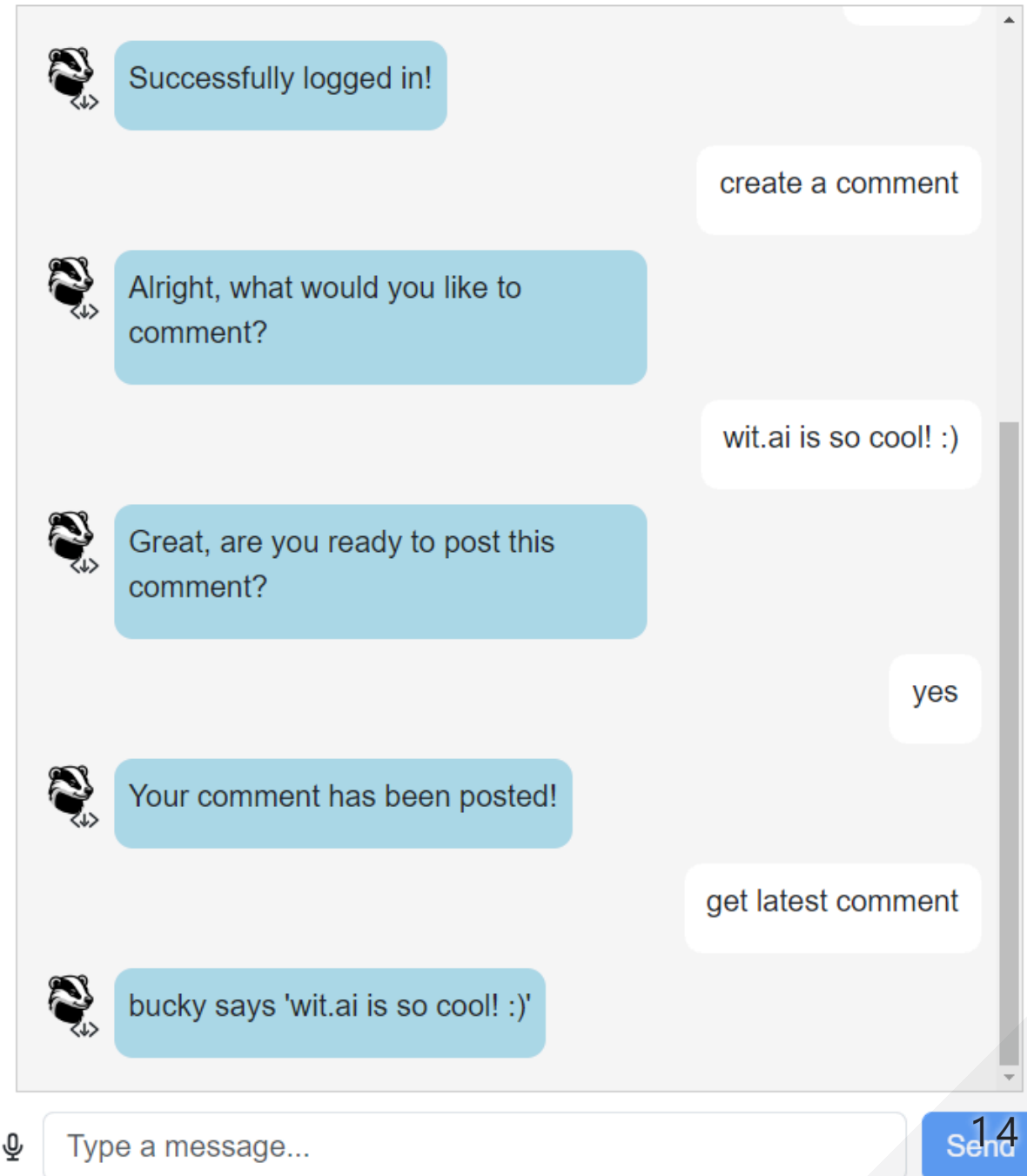
```
const createLoginSubAgent = (end) => {  
  let stage;  
  
  const handleInitialize = async (promptData) => {  
    return "I should handle logging in!";  
  }  
  
  const handleReceive = async (prompt) => {  
    switch(stage) {  
      case "FOLLOWUP_USERNAME": return await handleFollowupUsername(prompt);  
      case "FOLLOWUP_PASSWORD": return await handleFollowupPassword(prompt);  
    }  
  }  
  
  const handleFollowupUsername = async (prompt) => { }  
  
  const handleFollowupPassword = async (prompt) => {  
    // ...  
    end();  
  }  
}
```

## In CreateCommentSubAgent.js ...

```
const createCommentSubAgent = (end) => {  
  let stage;  
  
  const handleInitialize = async (promptData) => {  
    return "I should handle creating a comment!";  
  }  
  
  const handleReceive = async (prompt) => {  
    switch(stage) {  
      case "FOLLOWUP_COMMENT": return await handleFollowupComment(prompt);  
      case "FOLLOWUP_CONFIRM": return await handleFollowupConfirm(prompt);  
    }  
  }  
  
  const handleFollowupComment = async (prompt) => { }  
  
  const handleFollowupConfirm = async (prompt) => {  
    // ...  
    end();  
  }  
}
```

# Your Turn!

*Delegate* and implement  
`create_comment` for  
BadgerChat Mini!



# Other Features

# Text-to-Speech `handleSynthesis`

```
const resp = await fetch(`https://api.wit.ai/synthesize`, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "Accept": "audio/wav",
    "Authorization": `Bearer ...`
  },
  body: JSON.stringify({
    q: "This is the text I would like to transcribe!",
    voice: "Rebecca",
    style: "soft"
  })
})
```

Limited to 60 requests/minute, 280 characters max.



# Speech-to-Text `handleTranscription`

```
const resp = await fetch(`https://api.wit.ai/dictation`, {
  method: "POST",
  headers: {
    "Content-Type": "...",
    "Authorization": `Bearer ...`
  },
  body: rawSound
})
```

`Content-Type` is limited to a range of audio types, *not* what is recorded by default.

# Traits

Used for detecting certain **traits** like user sentiment.



# Questions?