Voice Dev 2

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 fully grasp the asynchronous nature of JavaScript.
2. Be able to keep the context of the conversation.
3. Be able to delegate to subagents.
4. Be able to appreciate other neat features, such as traits, speech-to-text translation, and text-to-speech synthesis.

Last Time...

Let's create a Wit.AI comedian that can understand...

`why_chicken` e.g. "why did the chicken cross the road"

- We will reply with "to get to the other side!"

`tell_joke` e.g. "tell me a joke"

- We will reply with jokes fetched from the Jokes API!

```
const createChatAgent = () => {
    const CS571_WITAI_ACCESS_TOKEN = "...";

    // Define variables here!
    let jokeNum;

    const handleInitialize = async () => {
        return "Welcome to BadgerChat Jokes!";
    }

    // ...
}
export default createChatAgent;
```

Closures allow us to share some *state*. Should we be concerned about sharing this?

Asynchronous Code

❗ Two `async` functions sharing state? JavaScript is actually single-threaded!

❗ ❗ JavaScript is single-threaded? Things like `fetch` and `setTimeout` are run by the browser *outside of JavaScript* on a seperate thread.

Enter the **event loop**.

Event Loop

1. When an asynchronous function is invoked, like *fetch* or *setTimeout*, begin work outside of JavaScript.
2. When the work is complete add the callback function to the task queue.
3. When the stack is empty, pull and execute the next callback function from the task queue*.

Learn more about the event loop... (1) (2)

* since 2015, there is technically a task (js) *and* microtask (us) queue

Image Source, Slightly Modified

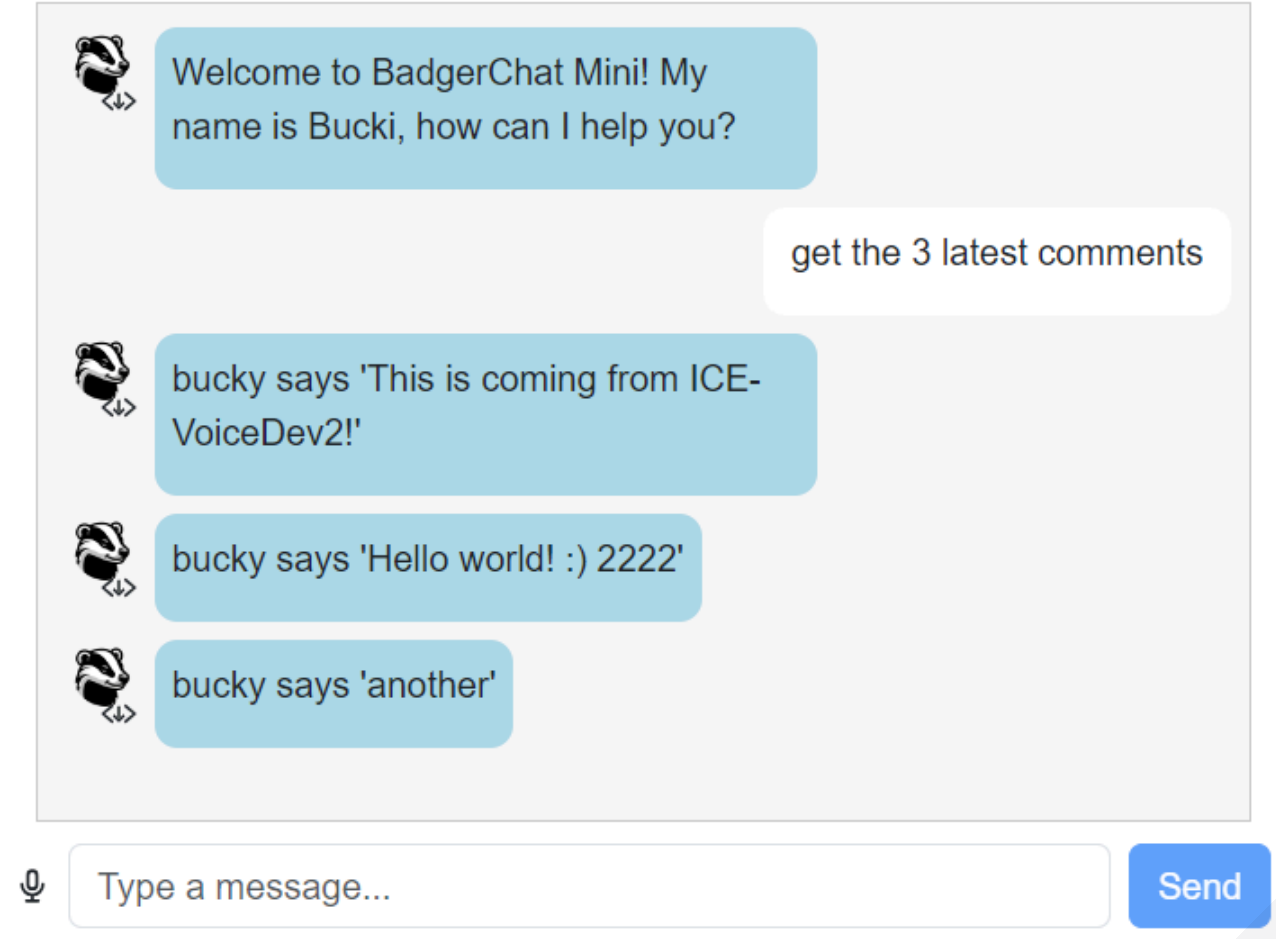
A Deeper Understanding

Let's deconstruct the React app surrounding our agent.

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 **bucky124** and password **myp@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"`

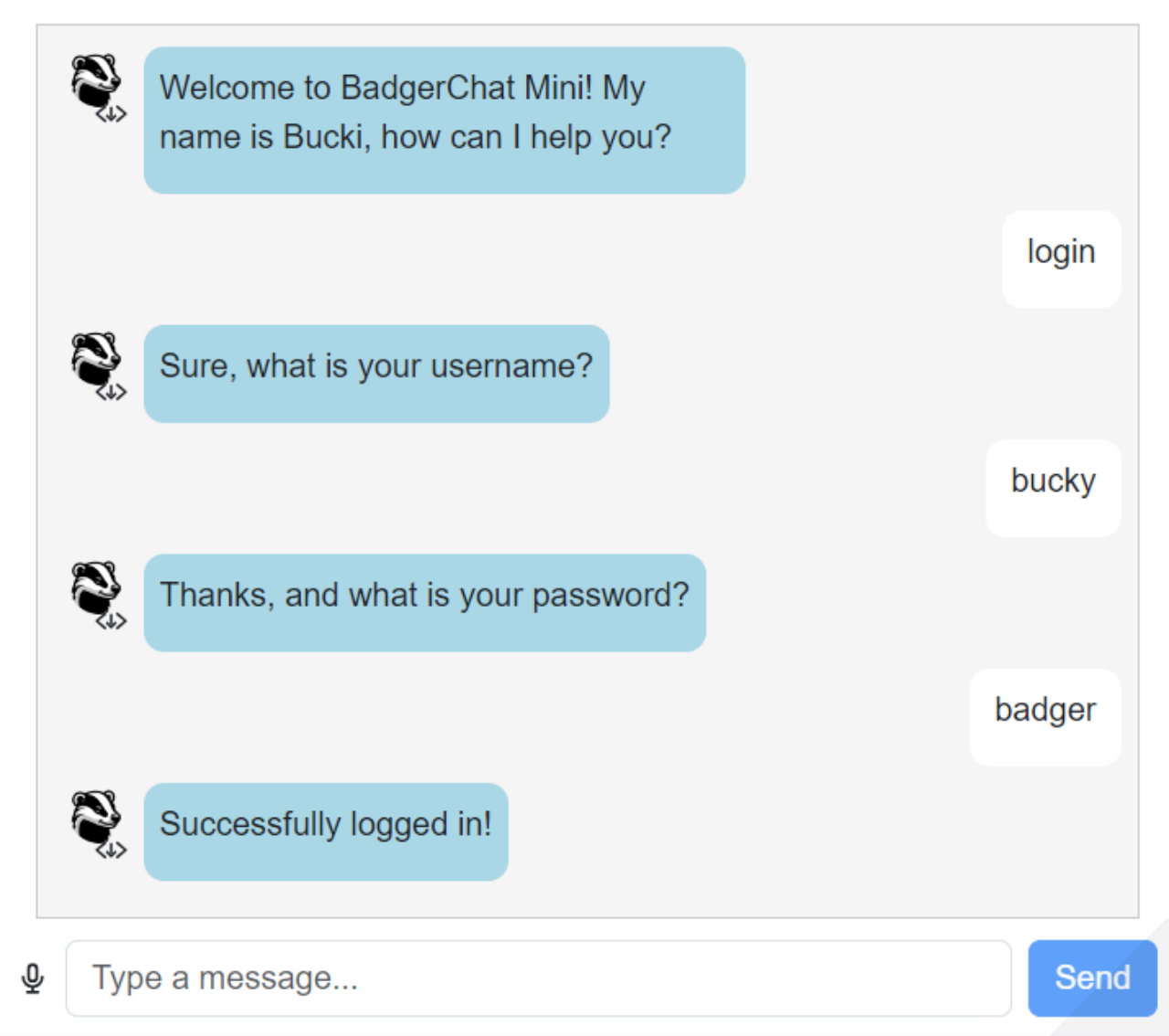
On the next `handleReceive`, ask for their password and set the stage to `"FOLLOWUP_PASSWORD"`

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

Your Turn!

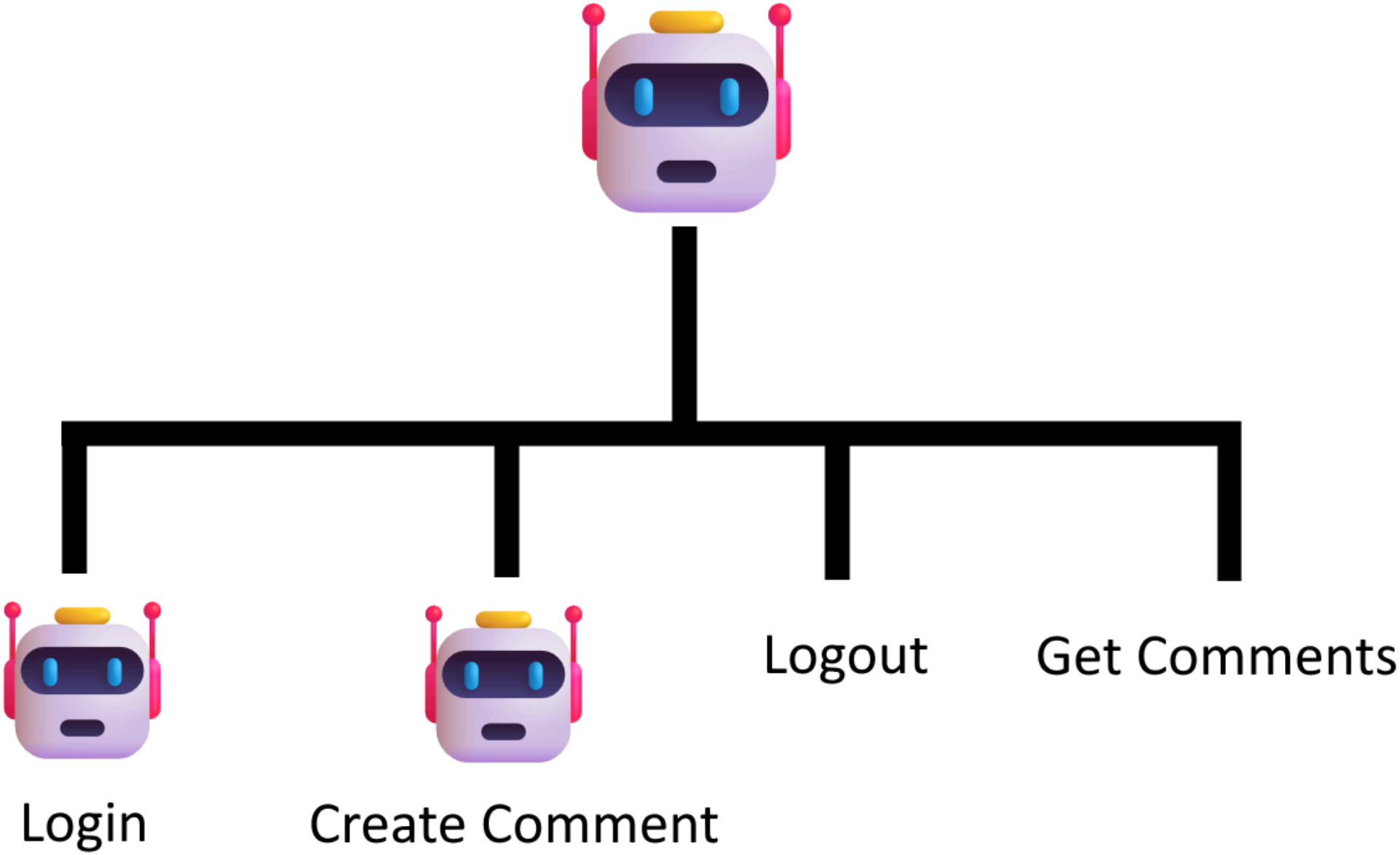
Implement `login` for BadgerChat Mini!

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



```
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


Login
Create Comment
Logout
Get Comments

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` …

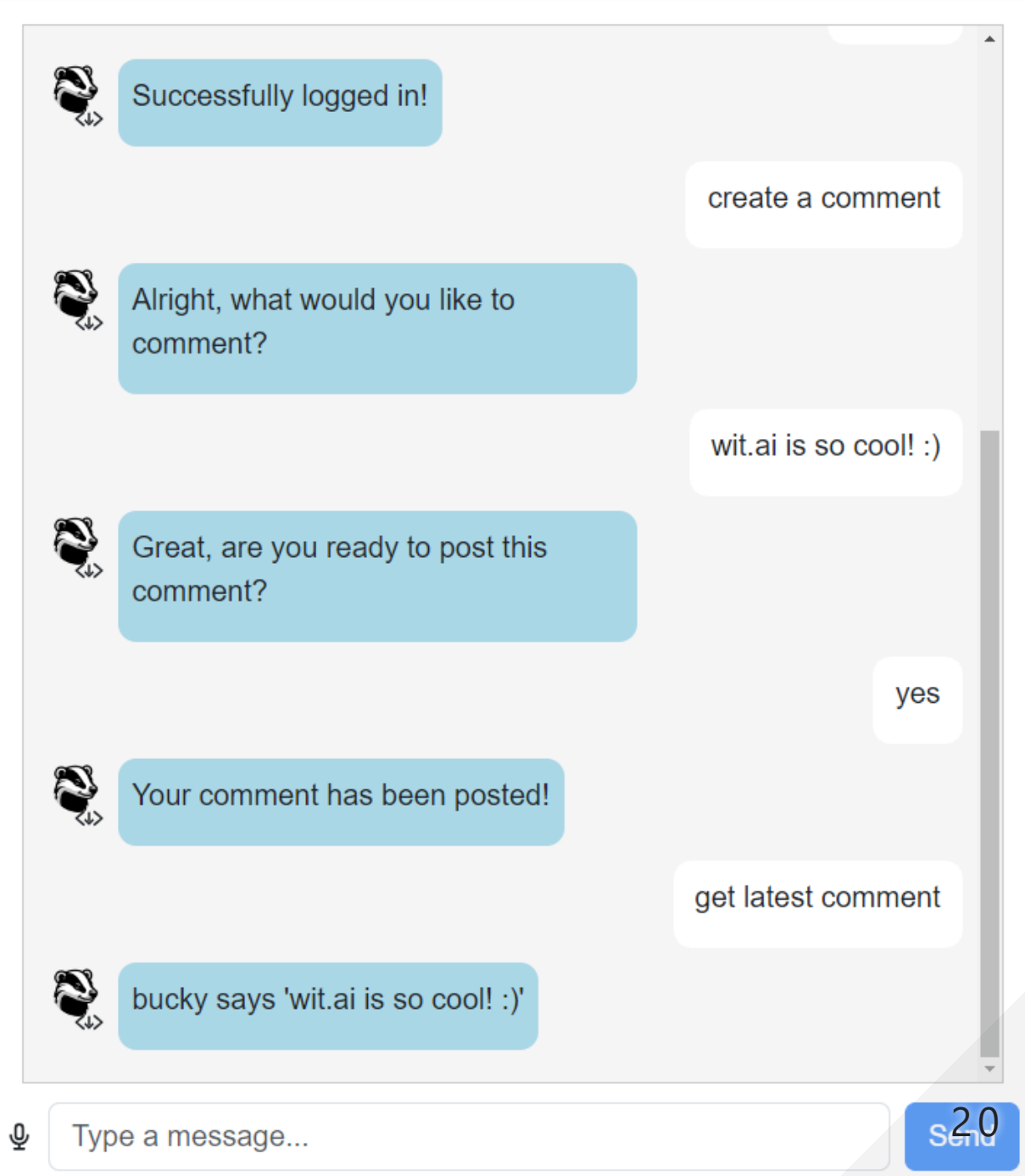
```
const createLoginSubAgent = (end) => {

    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 and Speech-to-Text

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?