

FOLDER STRUCTURE

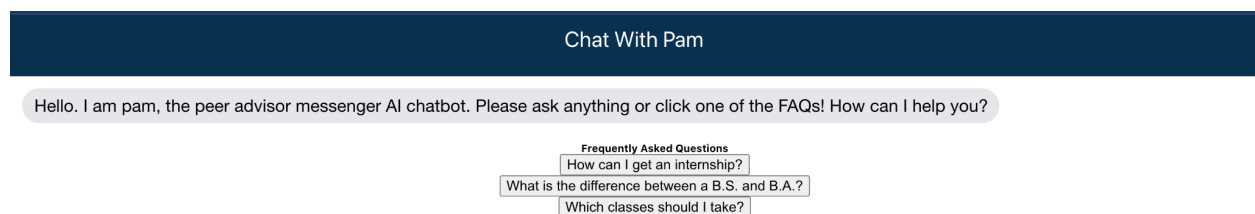
In the src folder of the repo, there's two subdirectories: frontend and backend. The frontend folder uses React and the backend folder uses Django. We currently only use the frontend folder, but the backend folder can be used by feature contributors as needed.

FRONTEND FOLDER

The file that we've used the most is the **chatbot_component.js** file. This file was adapted off of the react-lex-plus npm package (<https://www.npmjs.com/package/react-lex-plus>, <https://github.com/AlexWang-16/react-lex-plus>) which itself is adapted off of the react-lex npm package (<https://www.npmjs.com/package/react-lex/v/1.0.0>, <https://github.com/promediacorp/react-lex>). This file handles displaying the user's message on the frontend and sending it to AWS Lex and retrieving the response from AWS Lex and displaying it on the frontend. There are many functions in this file, but we will only cover the most important ones to what we were doing.

pushChat - this function takes the user's message and displays it on the screen (`this.showRequest(inputField);`) and sends it to AWS Lex (`this.lexruntime.postText(params, a.bind(this));`). We haven't edited this in any way from the react-lex-plus package.

componentDidMount() - this function is in charge of greeting message with a response card of showing some of the FAQs that the users can click. The users are also welcomed to ask any questions if their questions are not shown on the FAQs.



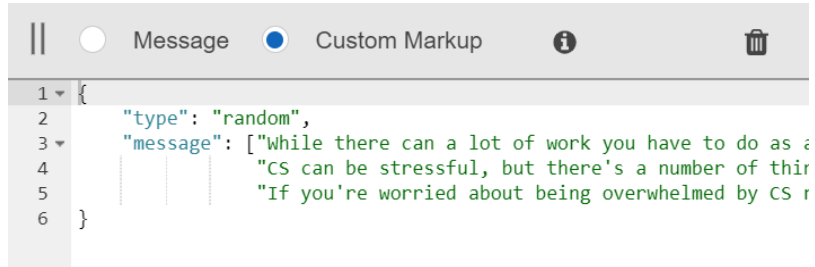
1. The FAQs can be edited in the buttons array: add or delete any questions if necessary

```
this.greetingMsgRef.current = faqNode;
faqNode.className = "lexResponse";
let title = "Frequently Asked Questions";
let buttons = ["How can I get an internship?",
               "What is the difference between a B.S. and B.A.?",
               "Which classes should I take?"];
let responseCardDiv = document.createElement("div");
responseCardDiv.style.textAlign = "center";
```

showResponse() - this function takes the response from Lex and displays it on the webpage. Nearly all of our contributions to the code come in this function

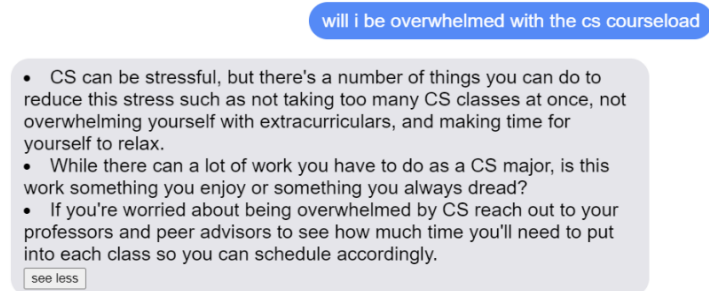
1. If we get a response from Lex
 - a. Then we try to parse out the message attribute of Lex's response as a JSON
 - i. If the type attribute of this JSON is "random" it means that the JSON has provided 3 messages and one of them will be displayed randomly, with a see more button being provided to see all of the responses if the user desired.

1. Code:



```
1 {  
2   "type": "random",  
3   "message": ["While there can a lot of work you have to do as a CS major, is this  
4               "CS can be stressful, but there's a number of things you can do to  
5               "If you're worried about being overwhelmed by CS reach out to your  
6   }]
```

2. How it looks:



will i be overwhelmed with the cs courseload

- CS can be stressful, but there's a number of things you can do to reduce this stress such as not taking too many CS classes at once, not overwhelming yourself with extracurriculars, and making time for yourself to relax.
- While there can a lot of work you have to do as a CS major, is this work something you enjoy or something you always dread?
- If you're worried about being overwhelmed by CS reach out to your professors and peer advisors to see how much time you'll need to put into each class so you can schedule accordingly.

see less

- b. However, not all of the message attributes of Lex's response are in JSON form. In fact, right now the only one that is is the "random" message type. So we catch all the other message types.
 - i. If the message attribute begins with a "<" then we interpret it to be formatted as HTML. We did originally have the HTML message type in JSON format, with the type attribute being "HTML" and the message attribute being the HTML. However, we decided not to do this because it meant that the person writing the HTML would have to make it a string and also have it indented a couple tabs. Additionally, the majority of our messages were in HTML because they included links, so we wanted to make it as easy as possible to write messages of this type. However if anyone in future development wishes to go back to putting this in JSON form because they have other responses that start with "<" or they prefer it that way, by all means this can be done.

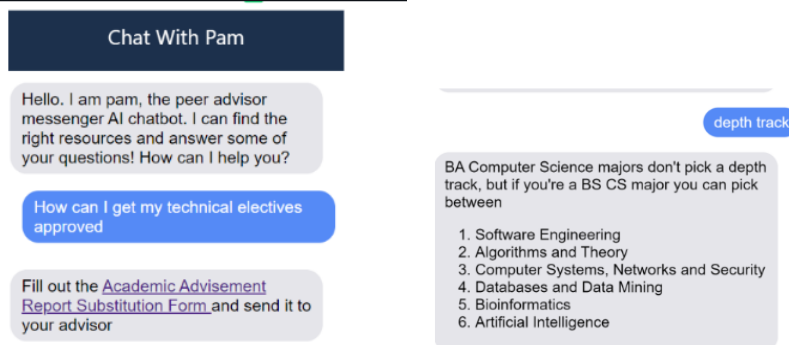
1. Old code:

```
7  {
8    "type": "HTML",
9    "message": '<p>
10 You can find resources to improve your English speaking/
11 writing skills
12 <a href="https://case.edu/studentsuccess/academic-resources/en
13 </p>'
14 }
```

2. Current code:

```
|| ● Message ● Custom Markup ⓘ 🗑️
1 <p>
2 You can find resources to improve your English speaking/
3 writing skills
4 <a href="https://case.edu/studentsuccess/academic-resources/englis
5 </p>
```

3. How it looks:




ii. **else if** the response from Lex includes an attribute called “responseCard” then a special response is sent in the chat called a “Response Card”

1. Code:

Prompt response cards

Card 1 ⓘ Preview as: Facebook 🗑️



[Edit Image Uri](#)

Title* Semester

Subtitle* To Study Abroad

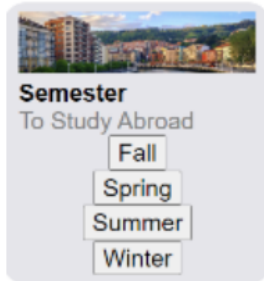
Button title* Fall ⓘ

Button value* Fall ▼

2. How it looks:

Which country would you like to study abroad?

finland



Summer

- iii. **else** the last type of message that's sent is just a plaintext message with no randomness, html, or response cards

1. Code:

||

☒ Message

☐ Custom Markup

i

🗑

One of these messages will be presented at random.

e.g. Thank you. Your {Drink_Name} has been ordered.

+

Please check on SIS > Academics > Academic Progress > View

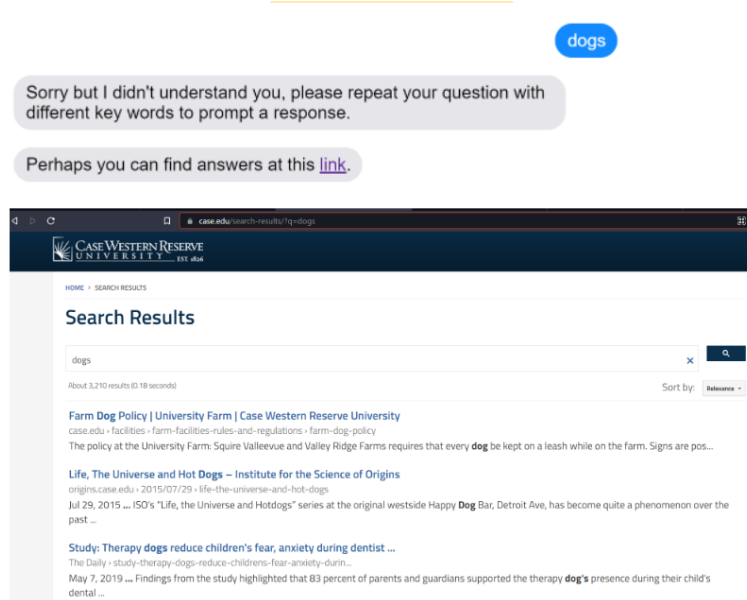
✕

2. How it looks:

How can I graduate early

Please check on SIS > Academics > Academic Progress > View What If Report

- c. After this message is sent, one last message may be sent, which is only for error handling. So if the `intentName` attribute of Lex's response is null (which means the user's question has no corresponding answer), then a message is sent with a link that directs the user to the search results page on the CWRU website

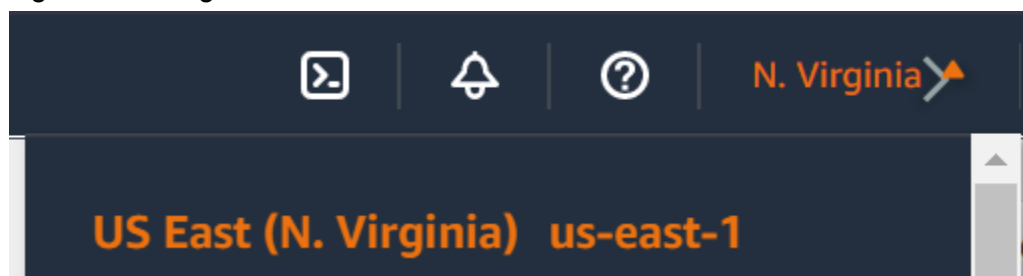


The App.js file is what actually displays the webpage by using the chatbot_component.js component. The props in MyLexChat mean the following:

- botName is the actual name of the bot on Lex (that's why it's WebUiOrderFlowers because that's the name we originally gave this bot when following a tutorial)
- IdentityPoolId is found by searching for "cognito" on aws and going to "identity pools" and clicking on the one for your bot and clicking on the "sample code" section on the side tab
- placeholder is the placeholder text in the user's message box



- region is the region Lex is hosted in



- headerText is the displayed title of the chatbot



- headerStyle is the styling for the header above
- greeting is what Lex will ask before the user asks anything
- width and margin are used to make the webpage responsive