...Continuando

- Para quem teve problemas na última aula:

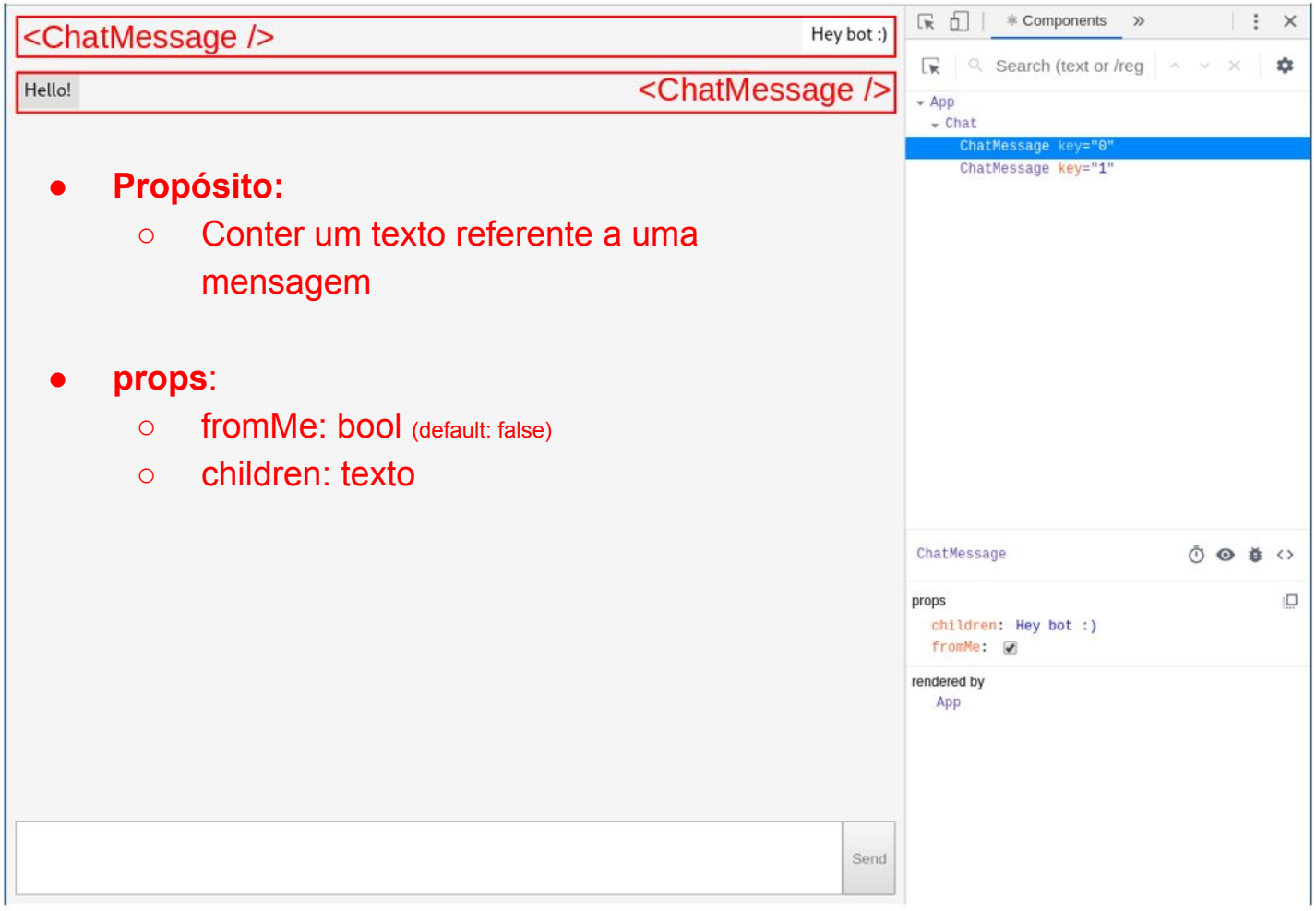
```
git clone https://github.com/filipealvesdef/tutorial-react-chat.git
cd tutorial-react-chat
yarn install
```

- Caso não tenha o **git**, baixe o zip
  - https://github.com/filipealvesdef/tutorial-react-chat/archive/master.zip/

    OU

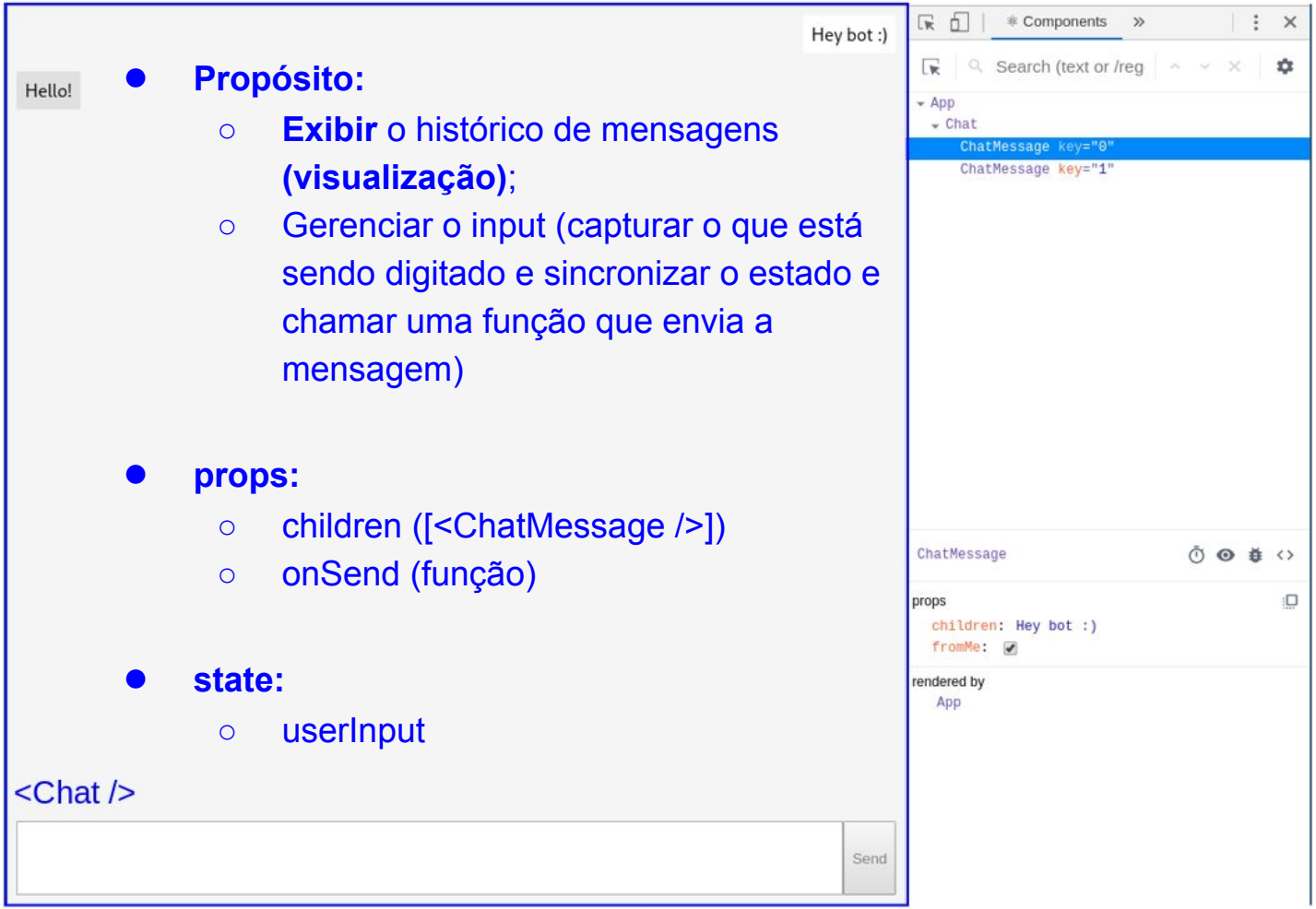
  - https://bit.ly/2kPiZBX

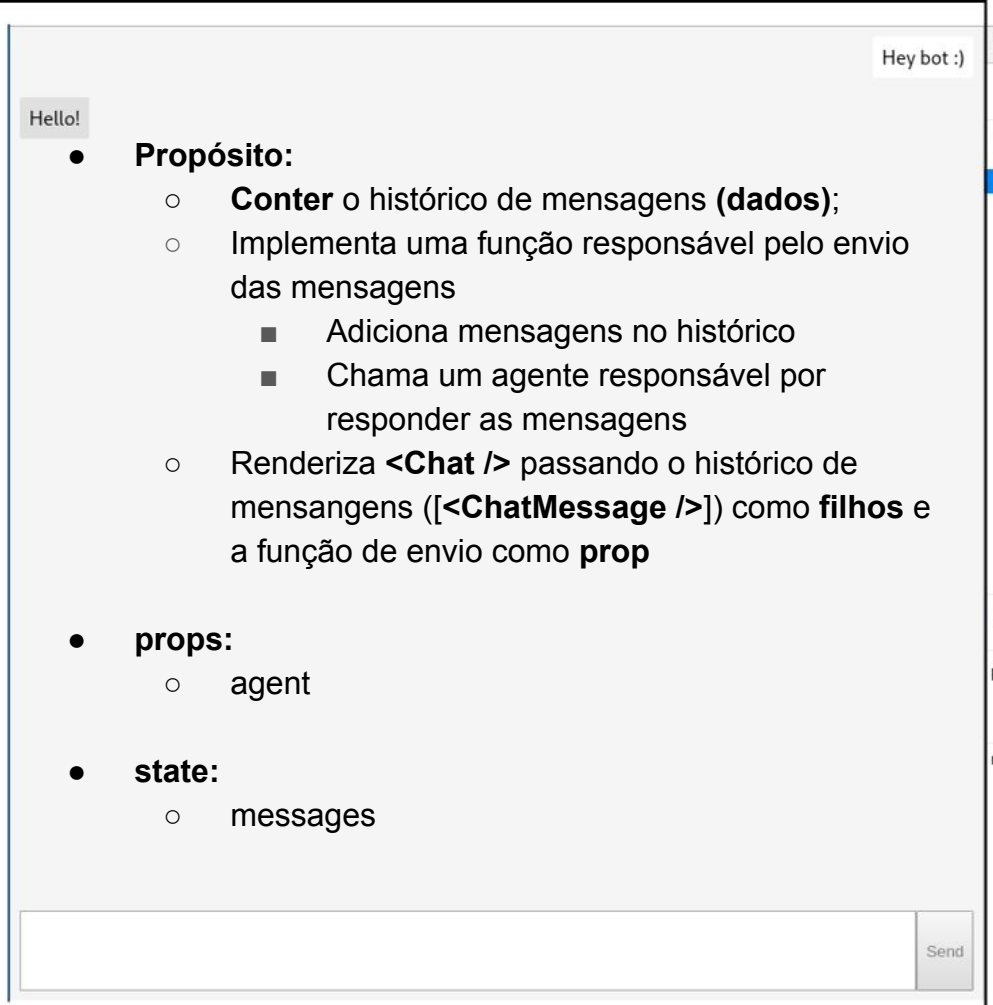
Recaptulando...



- **Propósito:**
  - Conter um texto referente a uma mensagem

- **props**:
  - fromMe: bool (default: false)
  - children: texto

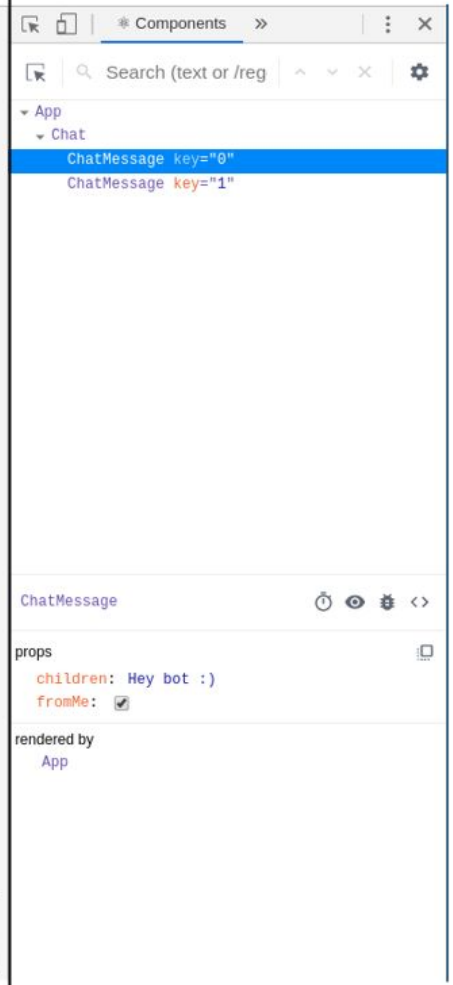

Hey bot :)
Hello!
Propósito:
Exibir o histórico de mensagens (visualização);
Gerenciar o input (capturar o que está sendo digitado e sincronizar o estado e chamar uma função que envia a mensagem)
props:
children ([<ChatMessage />])
onSend (função)
state:
userInput
<Chat />
Send
Components
Search (text or /reg
App
Chat
ChatMessage key="0"
ChatMessage key="1"
ChatMessage
props
children: Hey bot :)
fromMe:
rendered by
App



- **Propósito:**
  - **Conter** o histórico de mensagens **(dados)**;
  - Implementa uma função responsável pelo envio das mensagens
    - Adiciona mensagens no histórico
    - Chama um agente responsável por responder as mensagens
  - Renderiza **<Chat />** passando o histórico de mensangens ([**<ChatMessage />**]) como **filhos** e a função de envio como **prop**

- **props:**
  - agent

- **state:**
  - messages



<App />

Onde paramos?

- Adicionamos os componentes <ChatMessage /> como filhos do <Chat />
  - **Por que?** Permitir que sejam adicionados vários <ChatMesssage /> à medida que a conversa for progredindo;

  - **Problemas**: Mensagens estáticas e hardcoded

Objetivos

1. Adicionar as mensagens de forma dinâmica na interface
    a. Separar os **dados** (app.js) da **visualização** (chat.js)
    b. Criar função responsável pelo envio das mensagens
    c. Versão monólogo (apenas as mensagens digitadas pelo usuário são mostradas no histórico)

2. Criar um agente responsável por responder quando alguma mensagem é enviada
    a. Agente com respostas estáticas (defaultProps do app.js)
    b. Integrar com o serviço de *chatbot* rodando na nossa infraestrutura (Y-Bot)

1) Adicionar mensagens dinamicamente

a) Separar os dados da visualização

1. Criar um estado no **<App />** referente ao histórico de mensagens **(dados)**
    a. Vamos usar o **useState**!
        i. Importar usando **{ useState }** porque não é um export default do módulo ‘react’
        ii. **Parâmetros**: Estado inicial
        iii. **Retorno:** o **estado atual** e **função** responsável por atualizar o estado

a) Separar os dados da visualização

2. Criar as **<ChatMessages />** com base nas mensagens contidas no estado atual do **<App />**
    a. Iterar em *messages*, obtendo os valores de *from* e *text*
    b. Criar uma lista de **<ChatMessages />** com os valores de *text* e *from* obtidos a partir de *messages*
    c. Passar a lista de **<ChatMessage />** como filha do **<Chat />**

a) Separar os dados da visualização

2. Criar as **<ChatMessages />** com base nas mensagens contidas no estado atual do **<App />**
    a. Iterar em *messages*, obtendo os valores de *from* e *text*
    b. Criar uma lista de **<ChatMessages />** com os valores de *text* e *from* obtidos a partir de *messages*
    c. Passar a lista de **<ChatMessage />** como filha do **<Chat />**

Vamos usar programação funcional! (Arrow functions e map) :)

Arrow functions

```
(param1, param2, ..., paramN) => {
    do something;
}
```

```
param1 => {
    do something with single param;
}
```

```
() => {console.log ('Hello Filipe')}
```

Arrow functions

```
1 param1 => param1 ** 2
```

```
1 () => ({a: 32, b: 23})
```

Map function

```
arr.map((v, i) => {
    // i is an optional parameter that refers to index
    // of each element in the arr
    do something with v and i
})
```

b) Função responsável pelo envio das mensagens

1. Criar função **onSend** no **<App />** cuja responsabilidade será adicionar novas mensagens no estado do **<App />** (apenas no próximo commit)

2. Passar a prop **onSend** para o **<Chat />**

3. **<Chat />** chama a função passada como prop e reinicia o estado do input quando o botão de enviar for pressionado

c) Versão monólogo

1. Alterar o estado inicial das mensagens do **<App />** para uma lista vazia;

2. Adicionar novas mensagens na lista quando **onSend** for chamada (**<Chat />** chama esta função quando ocorre o clique no botão pois foi passada como prop)
   a. Não é uma boa prática atualizar o estado diretamente. Por isso chamamos **setMessages**
   b. Usar a função **concat** ao invés de **push**

2) Agente que responde às mensagens

a) Agente com respostas prontas

- Possui 2 estados: **idle** e **recording.** Muda o estado e envia a mensagem referente ao estado atual
  - **Idle:** Hello I'm an answering machine. Leave your message after the beep.
    BEEEEEEP…
  - **recording:** Message recorded… (not really)

- **Problema:** Quando utilizarmos um serviço para receber as mensagens (Y-Bot), haverá latência. Se o envio das mensagens é for feito de forma síncrona, a interface trava até o recebimento da resposta.

a) Agente com respostas prontas

- Possui 2 estados: **idle** e **recording.** Muda o estado e envia a mensagem referente ao estado atual
  - **Idle:** Hello I'm an answering machine. Leave your message after the beep.
    BEEEEEEP…
  - **recording:** Message recorded… (not really)

- **Problema:** Quando utilizarmos um serviço para receber as mensagens (Y-Bot), haverá latência. Se o envio das mensagens é for feito de forma síncrona, a interface trava até o recebimento da resposta.

Vamos fazer chamadas assíncronas! (Promises) :)

Promises

- Objeto que representa conclusão ou falha de operação assíncrona
- Permite adicionar callbacks quando a conclusão ou falha ocorrer
- Criação de promises

```
1 const p = new Promise(function(resolve, reject) { ... } )
```

resolve e **reject** são funções

Promises

- Atalho para criação de promises resolvidas

```
1 Promise.resolve(value)
```

- Consumo de promises

```
1 doSomething().then(successCallback, failCallback)
```

b) Integração com Y-Bot

- Disponível em: https://learning.veslasoft.com:3535/y-bot/api/rest/v1.0/ask
  - **question:** String codificada (Espaços e caracteres são convertidos para %20...)
  - **userid:** Cada interação com o *chat bot* cria uma nova sessão, responsável por guardar informações obtidas de mensagens prévias (*e.g.* nome)
- Pacote **uuid** para gerar a string que identifica a sessão
- Implementação do **YBotAgent**
  - **atributos:** endpoint, userid
  - **método:** sendMessage

Exportar variável de ambiente

- Pause a aplicação **(Ctrl + C)**

- **No Linux**

```
export REACT_APP_YBOT_ENDPOINT=https://learning.veslasoft.com:3535/y-bot/api/rest/v1.0/ask
```

- **No Windows**

```
SET "REACT_APP_YBOT_ENDPOINT=https://learning.veslasoft.com:3535/y-bot/api/rest/v1.0/ask"
```

Obrigado!