Package 'shinychat'

December 18, 2024

Title Chat UI Component for 'shiny'

Version 0.1.1

Description Provides a scrolling chat interface with multiline input, suitable for creating chatbot apps based on Large Language Models (LLMs). Designed to work particularly well with the 'elmer' R package for calling LLMs.

License MIT + file LICENSE

URL https://github.com/jcheng5/shinychat, https://jcheng5.github.io/shinychat/

BugReports https://github.com/jcheng5/shinychat/issues

Imports bslib, coro, htmltools, promises (>= 1.3.2), rlang, shiny (>= 1.10.0)

Suggests testthat (>= 3.0.0)

Encoding UTF-8

RoxygenNote 7.3.2

Config/testthat/edition 3

Config/Needs/website tidyverse/tidytemplate

NeedsCompilation no

Author Joe Cheng [aut, cre], Carson Sievert [aut], Posit Software, PBC [cph, fnd]

Maintainer Joe Cheng <joe@posit.co>

Repository CRAN

Date/Publication 2024-12-18 15:50:10 UTC

Contents

chat_append *Append an assistant response (or user message) to a chat control*

Description

The chat_append function appends a message to an existing chat control. The response can be a string, string generator, string promise, or string promise generator (as returned by the 'elmer' package's chat, stream, chat_async, and stream_async methods, respectively).

This function should be called from a Shiny app's server. It is generally used to append the model's response to the chat, while user messages are added to the chat UI automatically by the front-end. You'd only need to use chat_append(role="user") if you are programmatically generating queries from the server and sending them on behalf of the user, and want them to be reflected in the UI.

Usage

```
chat_append(
  id,
  response,
  role = c("assistant", "user"),
  session = getDefaultReactiveDomain()
)
```

Arguments

id	The ID of the chat element
response	The message or message stream to append to the chat element
role	The role of the message (either "assistant" or "user"). Defaults to "assistant".
session	The Shiny session object

Value

Returns a promise. This promise resolves when the message has been successfully sent to the client; note that it does not guarantee that the message was actually received or rendered by the client. The promise rejects if an error occurs while processing the response (see the "Error handling" section).

Error handling

If the response argument is a generator, promise, or promise generator, and an error occurs while producing the message (e.g., an iteration in stream_async fails), the promise returned by chat_append will reject with the error. If the chat_append call is the last expression in a Shiny observer, Shiny will see that the observer failed, and end the user session. If you prefer to handle the error gracefully, use promises::catch() on the promise returned by chat_append.

Examples

```
library(shiny)
library(coro)
library(bslib)
library(shinychat)

# Dumbest chatbot in the world: ignores user input and chooses
# a random, vague response.
fake_chatbot <- async_generator(function(input) {
  responses <- c(
    "What does that suggest to you?",
    "I see.",
    "I'm not sure I understand you fully.",
    "What do you think?",
    "Can you elaborate on that?",
    "Interesting question! Let's examine thi... **See more**"
  )

  await(async_sleep(1))
  for (chunk in strsplit(sample(responses, 1), "")[[1]]) {
    yield(chunk)
    await(async_sleep(0.02))
  }
})

ui <- page_fillable(
  chat_ui("chat", fill = TRUE)
)

server <- function(input, output, session) {
  observeEvent(input$chat_user_input, {
    response <- fake_chatbot(input$chat_user_input)
    chat_append("chat", response)
  })
}

shinyApp(ui, server)
```

---

chat_append_message *Low-level function to append a message to a chat control*

Description

For advanced users who want to control the message chunking behavior. Most users should use `chat_append()` instead.

Usage

```
chat_append_message(
  id,
  msg,
  chunk = TRUE,
  operation = c("append", "replace"),
  session = getDefaultReactiveDomain()
)
```

Arguments

id	The ID of the chat element
msg	The message to append. Should be a named list with `role` and `content` fields. The `role` field should be either "user" or "assistant". The `content` field should be a string containing the message content, in Markdown format.
chunk	Whether `msg` is just a chunk of a message, and if so, what type. If `FALSE`, then `msg` is a complete message. If `"start"`, then `msg` is the first chunk of a multi-chunk message. If `"end"`, then `msg` is the last chunk of a multi-chunk message. If `TRUE`, then `msg` is an intermediate chunk of a multi-chunk message. Default is `FALSE`.
operation	The operation to perform on the message. If `"append"`, then the new content is appended to the existing message content. If `"replace"`, then the existing message content is replaced by the new content. Ignored if `chunk` is `FALSE`.
session	The Shiny session object

Value

Returns nothing (`invisible(NULL)`).

Examples

```
library(shiny)
library(coro)
library(bslib)
library(shinychat)

# Dumbest chatbot in the world: ignores user input and chooses
# a random, vague response.
fake_chatbot <- async_generator(function(id, input) {
  responses <- c(
    "What does that suggest to you?",
    "I see.",
    "I'm not sure I understand you fully.",
    "What do you think?",
    "Can you elaborate on that?",
    "Interesting question! Let's examine thi... **See more**"
  )

  # Use low-level chat_append_message() to temporarily set a progress message
```

```
  chat_append_message(id, list(role = "assistant", content = "_Thinking..._ "))
  await(async_sleep(1))
  # Clear the progress message
  chat_append_message(id, list(role = "assistant", content = ""), operation = "replace")

  for (chunk in strsplit(sample(responses, 1), "")[[1]]) {
    yield(chunk)
    await(async_sleep(0.02))
  }
})

ui <- page_fillable(
  chat_ui("chat", fill = TRUE)
)

server <- function(input, output, session) {
  observeEvent(input$chat_user_input, {
    response <- fake_chatbot("chat", input$chat_user_input)
    chat_append("chat", response)
  })
}

shinyApp(ui, server)
```

`chat_ui` *Create a chat UI element*

Description

Inserts a chat UI element into a Shiny UI, which includes a scrollable section for displaying chat messages, and an input field for the user to enter new messages.

To respond to user input, listen for `input$ID_user_input` (for example, if `id="my_chat"`, user input will be at `input$my_chat_user_input`), and use `chat_append()` to append messages to the chat.

Usage

```
chat_ui(
  id,
  ...,
  messages = NULL,
  placeholder = "Enter a message...",
  width = "min(680px, 100%)",
  height = "auto",
  fill = TRUE
)
```

Arguments

id	The ID of the chat element
...	Extra HTML attributes to include on the chat element
messages	A list of messages to prepopulate the chat with. Each message can be a string or a named list with content and role fields.
placeholder	The placeholder text for the chat's user input field
width	The CSS width of the chat element
height	The CSS height of the chat element
fill	Whether the chat element should try to vertically fill its container, if the container is fillable

Value

A Shiny tag object, suitable for inclusion in a Shiny UI

Examples

```
library(shiny)
library(bslib)
library(shinychat)

ui <- page_fillable(
  chat_ui("chat", fill = TRUE)
)

server <- function(input, output, session) {
  observeEvent(input$chat_user_input, {
    # In a real app, this would call out to a chat model or API,
    # perhaps using the 'elmer' package.
    response <- paste0(
      "You said:\n\n",
      "<blockquote>",
      htmltools::htmlEscape(input$chat_user_input),
      "</blockquote>"
    )
    chat_append("chat", response)
  })
}

shinyApp(ui, server)
```

Index