AI chatbots with shinychat (R) : : CHEAT SHEET

Create a basic chatbot

1. Set up the chatbot UI

Create a **chat UI element** with `chat_ui()`.

3. Handle communication between the user and chatbot

Set up a **reactive listener** that waits for the user to submit a message.

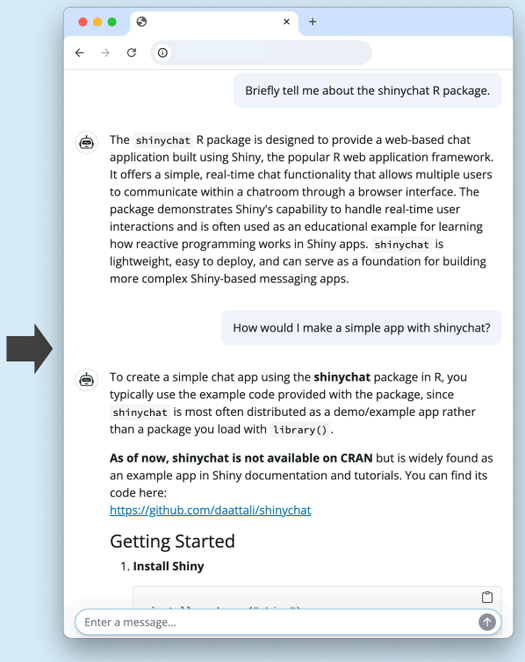
Submit input to the chatbot and **return** asynchronously streaming results.

Append the response to the UI element so the user can see the model's reply.

2. Create a chat

Initialize a **chat** with a `chat_*()` function, like `chat_openai()`.

```
# app.R
library(shiny)
library(shinychat)

ui <- bslib::page_fluid(
  chat_ui("chat")
)

server <- function(input, output, session) {
  chat <- ellmer::chat_openai()

  observeEvent(input$chat_user_input, {
    stream <- chat$stream_async(input$chat_user_input)
    chat_append("chat", stream)
  })
}

shinyApp(ui, server)
```



Save and restore chat state

Bookmark the chat state so the user can return to the conversation if they reload the app.

```
server <- function(input, output, session) {
  chat_client <- ellmer::chat_anthropic()
  chat_restore("chat", chat_client)
  # rest of server function ...
}

shinyApp(ui, server, enableBookmarking = "url")
```

Update the user input

Programmatically update the content in the user-input box and optionally submit it.

```
observeEvent(input$update_value, {
  update_chat_user_input(
    "chat",
    value = "New input"
  )
})
```

To update the placeholder text, use the `placeholder` argument.

To submit the input, use `submit = TRUE`.

Setup

CHOOSE A MODEL PROVIDER

Initialize a chat and specificy a model provider with an `ellmer::chat_*()` function.

Anthropic Claude: **`chat_anthropic()`**
AWS Bedrock: **`chat_aws_bedrock()`**
Azure OpenAI: **`chat_azure_openai()`**
Databricks: **`chat_databricks()`**
DeepSeek: **`chat_deepseek()`**
Google Gemini: **`chat_google_gemini()`**
Groq: **`chat_groq()`**
Hugging Face: **`chat_huggingface()`**
Ollama: **`chat_ollama()`**
OpenAI: **`chat_openai()`**
perplexity.ai: **`chat_perplexity()`**

Use the model argument to specify a model.

e.g.,

```
chat_openai(
  model = "o3"
)
```

Visit https://ellmer.tidyverse.org/#providers to see **all available providers.**

SPECIFY CREDENTIALS

Authenticate with your LLM provider. Many providers require an API key, which you can specify in your **.Renviron** file.

```
#.Renviron

OPENAI_API_KEY=your-key-123
ANTHROPIC_API_KEY=your-key-456
GOOGLE_API_KEY=your-key-789
```

Open .Renviron with `usethis::edit_r_environ()`.

Prompts and messages

ADD A SYSTEM PROMPT

Use a system prompt to tell the chatbot how you'd like it to behave.

```
# prompt.md

You are a helpful assistant who
answers questions briefly…
```

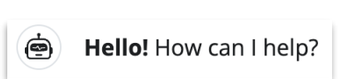
```
chat_*(system_prompt = readLines("prompt.md"))
```

Specify a system prompt using the `system_prompt` argument.

ADD STARTUP MESSAGES AND SUGGESTIONS

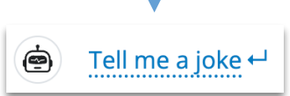
Add startup messages or suggestions to the conversation.

```
chat_ui(id = "chat", messages = "**Hello!** How can I help?")
```



Use `messages` to supply messages that will appear when the chat starts.

```
chat_ui(
  id = "chat",
  messages = '<span class="suggestion submit">Tell me a joke</span>'
)
```



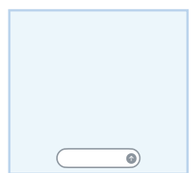
Use the `suggestion` and `submit` CSS classes to suggest inputs to the user and make them clickable.

Layouts

CHOOSE A LAYOUT FOR YOUR CHATBOT

Use functions from the bslib package to create a layout for your chatbot app.

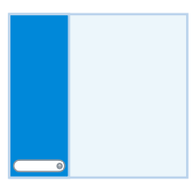
SCREEN-FILLING LAYOUT



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

Chat fills the page.

SIDEBAR LAYOUT



```
ui <- page_sidebar(
  sidebar = sidebar(
    chat_ui("chat", height = "100%"),
    style = "height: 100%;"
  ),
  fillable = TRUE
)
```

Chat fills the sidebar.

CARD LAYOUT

```
ui <- page_fillable(
  card(
    chat_ui(id = "chat")
  ),
  fillable_mobile = TRUE
)
```

Chat appears inside a card.

AI chatbots with shinychat (**Python**) : : CHEAT SHEET

Create a basic chatbot

1. Set up the chatbot UI

Create and display the **chat interface.**

3. Handle communication between the user and chatbot

Define a callback that runs **when the user submits a message.**

Submit input to the chatbot and **return** asynchronously streaming results.

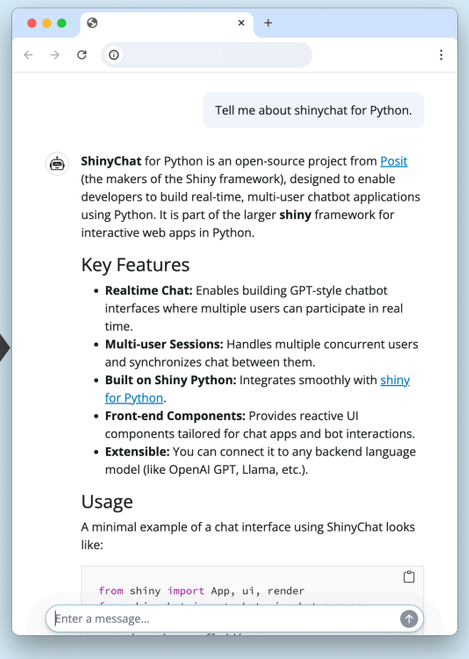
Append the response to the UI element so the user can see the model's reply.

```
# app.py
from chatlas import ChatOpenAI
from shiny.express import ui

chat = ui.Chat(id="my_chat")
chat.ui()

chat_client = ChatOpenAI()

@chat.on_user_submit
async def handle_user_input(user_input: str):
    response = await chat_client.stream_async(
        user_input
    )
    await chat.append_message_stream(
        response
    )
```

2. Create a chat

Initialize a **Chat client** with a `Chat*()` function like `ChatOpenAI()`.

This cheatsheet uses **Shiny Express**. To see examples for Shiny Core, see https://shiny.posit.co/py/docs/genai-chatbots.html.



Save and restore chat state

Bookmark the chat state so the user can return to the conversation if they reload the app.

```
chat_client = ChatOpenAI()
chat = ui.Chat(id="chat")

chat.enable_bookmarking(
  chat_client,
  bookmark_store="url", # or "server"
  bookmark_on="response", # or None
)
```

Update the user input

Programmatically update the content in the user-input box and optionally submit it.

```
@reactive.effect
@reactive.event(input.update_value)
def _():
  chat.update_user_input(
    value="New input"
  )
```

To update the placeholder text, use the `placeholder` parameter.

To submit the input, use `submit=True`.

Setup

CHOOSE A MODEL PROVIDER

Initialize a chat and specificy a model provider with a `chatlas` function.

Anthropic Claude: **`ChatAnthropic()`**
AWS Bedrock: **`ChatBedrockAnthropic()`**
Azure OpenAI: **`ChatAzureOpenAI()`**
Databricks: **`ChatDatabricks()`**
Google Gemini: **`ChatGoogle()`**
Groq: **`ChatGroq()`**
Ollama: **`ChatOllama()`**
OpenAI: **`ChatOpenAI()`**
perplexity.ai: **`ChatPerplexity()`**

You can also use other AI frameworks, like LangChain or Pydantic, to handle model communciation.

Use the `model` parameter to specify a model.
e.g.,
`ChatOpenAI(model="o3")`

Visit https://posit-dev.github.io/chatlas/reference to see **all available providers.**

SPECIFY CREDENTIALS

Authenticate with your LLM provider. Many providers require an API key, which you can specify in a **.env** file and load with **dotenv**.

```
#.env

OPENAI_API_KEY=your-key-123
ANTHROPIC_API_KEY=your-key-456
GOOGLE_API_KEY=your-key-789
```

See the provider's `Chat` function documentation for specifics on authentication.

Prompts and messages

ADD A SYSTEM PROMPT

Use a system prompt to tell the chatbot how you'd like it to behave.

```
# prompt.md

You are a helpful assistant who
answers questions briefly…
```

```
ChatOpenAI(system_prompt=open("prompt.md", "r").read())
```

Specify a system prompt using the `system_prompt` parameter.

ADD STARTUP MESSAGES AND SUGGESTIONS

Add startup messages or suggestions to the conversation.

```
chat.ui(messages=["**Hello!** How can I help?"])
```

Hello! How can I help?

Use `messages` to supply messages that will appear when the chat starts.

```
chat.ui(
  messages=['<span class="suggestion submit">Tell me a joke</span>']
)
```

Tell me a joke ↵

Use the `suggestion` and `submit` CSS classes to suggest inputs to the user and make them clickable.

Layouts

CHOOSE A LAYOUT FOR YOUR CHATBOT

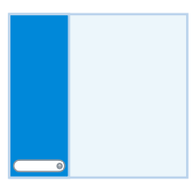
Use functions from the bslib package to create a layout for your chatbot app.

SCREEN-FILLING LAYOUT

```
ui.page_opts(
  fillable=True,
  fillable_mobile=True,
)

chat = ui.Chat(id="chat")
chat.ui()
```

Chat fills the page.

SIDEBAR LAYOUT



```
chat = ui.Chat(id="chat")

with ui.sidebar(width=300, style="height:100%"):
  chat.ui(height="100%")

"Main content"
```

Chat fills the sidebar.

CARD LAYOUT

```
ui.page_opts(
  fillable=True,
  fillable_mobile=True
)

chat = ui.Chat(id="chat")

with ui.card():
    chat.ui()
```

Chat appears inside a card that fills the page.