Package: gptstudio (via r-universe)

October 10, 2026

Type Package

Title Use Large Language Models Directly in your Development Environment

Version 0.4.0.9018

Maintainer James Wade <github@jameshwade.com>

Description Large language models are readily accessible via API. This package lowers the barrier to use the API inside of your development environment. For more on the API, see <https://platform.openai.com/docs/introduction>.

License MIT + file LICENSE

URL https://github.com/MichelNivard/gptstudio, https://michelnivard.github.io/gptstudio/

BugReports https://github.com/MichelNivard/gptstudio/issues

Depends R (>= 4.1)

Imports bsicons (>= 0.1.2), bslib (>= 0.8.0), cli, colorspace, coro, ellmer (>= 0.1.1.9000), glue, htmltools, htmlwidgets, httr2 (>= 1.0.3), ids, jsonlite, purrr, R6 (>= 2.0), rlang, rstudioapi (>= 0.12), shiny (>= 1.9.0), shiny.i18n, stringr (>= 1.5.0), utils, yaml

Suggests AzureGraph, future, grDevices, knitr, miniUI, mockr, promises, rmarkdown, rvest, shinytest2, spelling, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Remotes tidyverse/ellmer

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev zlib1g-dev

Repository https://michelnivard.r-universe.dev

Date/Publication 2026-01-03 18:11:37 UTC

RemoteUrl https://github.com/michelnivard/gptstudio

RemoteRef HEAD

RemoteSha c6cb15258c5e786dcc1bcf86711cac585ec85741

Contents

Buffer . . . 3
chat . . . 4
chat_create_system_prompt . . . 6
chat_history_append . . . 7
chat_message_default . . . 7
check_api_connection . . . 8
create_chat_app_theme . . . 8
create_completion_azure_openai . . . 9
create_ide_matching_colors . . . 10
create_translator . . . 10
ellmer_chat . . . 11
get_all_available_services . . . 11
get_available_models . . . 12
get_ide_theme_info . . . 12
gptstudio_cache_directory . . . 13
gptstudio_chat . . . 13
gptstudio_chat_in_source_addin . . . 14
gptstudio_comment_code . . . 15
gptstudio_create_skeleton . . . 15
gptstudio_request_perform . . . 16
gptstudio_run_chat_app . . . 17
gptstudio_sitrep . . . 18
gptstudio_spelling_grammar . . . 18
input_audio_clip . . . 19
mod_app_server . . . 20
mod_app_ui . . . 20
mod_chat_server . . . 21
mod_chat_ui . . . 21
multimodal_dep . . . 22
parse_data_uri . . . 22
prepare_chat_history . . . 22
rgb_str_to_hex . . . 23
set_allowed_models . . . 24
streamingMessage . . . 24
streamingMessage-shiny . . . 25
style_chat_history . . . 25

style_chat_message 26
text_area_input_wrapper 26
transcribe_audio 28
welcomeMessage 28
welcomeMessage-shiny 29

Index **30**

`Buffer` *Message Buffer*

Description

A message buffer

Public fields

`value` Placeholder for the message

Methods

Public methods:

- `Buffer$add_chunk()`
- `Buffer$clone()`

Method `add_chunk()`**:** Add a message chunk

Usage:

```
Buffer$add_chunk(chunk)
```

Arguments:

`chunk` A message chunk

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Buffer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

chat *Chat Interface for gptstudio*

Description

This function provides a high-level interface for communicating with various services and models supported by gptstudio. It orchestrates the creation, configuration, and execution of a request based on user inputs and options set for gptstudio. The function supports a range of tasks from text generation to code synthesis and can be customized according to skill level and coding style preferences.

Usage

```
chat(
  prompt,
  service = getOption("gptstudio.service"),
  history = list(list(role = "system", content = "You are an R chat assistant")),
  stream = FALSE,
  model = getOption("gptstudio.model"),
  skill = getOption("gptstudio.skill"),
  style = getOption("gptstudio.code_style", "no preference"),
  task = getOption("gptstudio.task", "coding"),
  custom_prompt = NULL,
  process_response = FALSE,
  session = NULL,
  ...
)
```

Arguments

`prompt`	A string containing the initial prompt or question to be sent to the model. This is a required parameter.
`service`	The AI service to be used for the request. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.service")`. If the option is not set, make sure to provide this parameter to avoid errors.
`history`	An optional parameter that can be used to include previous interactions or context for the current session. Defaults to a system message indicating "You are an R chat assistant".
`stream`	A logical value indicating whether the interaction should be treated as a stream for continuous interactions. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.stream")`.
`model`	The specific model to use for the request. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.model")`.
`skill`	A character string indicating the skill or capability level of the user. This parameter allows for customizing the behavior of the model to the user. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.skill")`.

`style` The coding style preferred by the user for code generation tasks. This parameter is particularly useful when the task involves generating code snippets or scripts. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.code_style")`.

`task` The specific type of task to be performed, ranging from text generation to code synthesis, depending on the capabilities of the model. If not explicitly provided, this defaults to the value set in `getOption("gptstudio.task")`.

`custom_prompt` An optional parameter that provides a way to extend or customize the initial prompt with additional instructions or context.

`process_response` A logical indicating whether to process the model's response.

`session` An optional parameter for a shiny session object.

`...` Reserved for future use.

Value

Depending on the task and processing, the function returns the response from the model, which could be text, code, or any other structured output defined by the task and model capabilities. The precise format and content of the output depend on the specified options and the capabilities of the selected model.

Examples

```
## Not run:
# Basic usage with a text prompt:
result <- chat("What is the weather like today?")

# Advanced usage with custom settings, assuming appropriate global options are set:
result <- chat(
  prompt = "Write a simple function in R",
  skill = "advanced",
  style = "tidyverse",
  task = "coding"
)

# Usage with explicit service and model specification:
result <- chat(
  prompt = "Explain the concept of tidy data in R",
  service = "openai",
  model = "gpt-4-turbo-preview",
  skill = "intermediate",
  task = "general"
)

## End(Not run)
```

chat_create_system_prompt

Create system prompt

Description

This function creates a customizable system prompt based on user-defined parameters such as coding style, skill level, and task. It supports customization for specific use cases through a custom prompt option.

Usage

```
chat_create_system_prompt(
  style = getOption("gptstudio.code_style"),
  skill = getOption("gptstudio.skill"),
  task = getOption("gptstudio.task"),
  custom_prompt = getOption("gptstudio.custom_prompt"),
  in_source = FALSE
)
```

Arguments

`style`	A character string indicating the preferred coding style. Valid values are "tidyverse", "base", "no preference". Defaults to `getOption(gptstudio.code_style)`.
`skill`	The self-described skill level of the programmer. Valid values are "beginner", "intermediate", "advanced", "genius". Defaults to `getOption(gptstudio.skill)`.
`task`	The specific task to be performed: "coding", "general", "advanced developer", or "custom". This influences the generated system prompt. Defaults to "coding".
`custom_prompt`	An optional custom prompt string to be utilized when `task` is set to "custom". Default is NULL.
`in_source`	A logical indicating whether the instructions are intended for use in a source script. This parameter is required and must be explicitly set to TRUE or FALSE. Default is FALSE.

Value

Returns a character string that forms a system prompt tailored to the specified parameters. The string provides guidance or instructions based on the user's coding style, skill level, and task.

Examples

```
## Not run:
chat_create_system_prompt(in_source = TRUE)
chat_create_system_prompt(
  style = "tidyverse",
  skill = "advanced",
```

```
  task = "coding",
  in_source = FALSE
)

## End(Not run)
```

`chat_history_append` *Append to chat history*

Description

This appends a new response to the chat history

Usage

```
chat_history_append(history, role, content, name = NULL)
```

Arguments

`history`	List containing previous responses.
`role`	Author of the message. One of `c("user", "assistant")`
`content`	Content of the message. If it is from the user most probably comes from an interactive input.
`name`	Name for the author of the message. Currently used to support rendering of help pages

Value

list of chat messages

`chat_message_default` *Default chat message*

Description

Default chat message

Usage

```
chat_message_default(translator = create_translator())
```

Arguments

`translator`	A Translator from `shiny.i18n::Translator`

Value

A default chat message for welcoming users.

check_api_connection *Check API Connection*

Description

This generic function checks the API connection for a specified service by dispatching to related methods.

Usage

```
check_api_connection(service, api_key = "", model = NULL)
```

Arguments

service	The name of the API service for which the connection is being checked.
api_key	The API key used for authentication.
model	The service's model to check

Value

A logical value indicating whether the connection was successful.

create_chat_app_theme *Chat App Theme*

Description

Create a bslib theme that matches the user's RStudio IDE theme.

Usage

```
create_chat_app_theme(ide_colors = get_ide_theme_info())
```

Arguments

ide_colors	List containing the colors of the IDE theme.

Value

A bslib theme

create_completion_azure_openai

Generate text using Azure OpenAI's API

Description

Use this function to generate text completions using OpenAI's API.

Usage

```
create_completion_azure_openai(
  prompt,
  task = Sys.getenv("AZURE_OPENAI_TASK"),
  base_url = Sys.getenv("AZURE_OPENAI_ENDPOINT"),
  deployment_name = Sys.getenv("AZURE_OPENAI_DEPLOYMENT_NAME"),
  api_key = Sys.getenv("AZURE_OPENAI_API_KEY"),
  api_version = Sys.getenv("AZURE_OPENAI_API_VERSION")
)
```

Arguments

`prompt`	a list to use as the prompt for generating completions
`task`	a character string for the API task (e.g. "completions"). Defaults to the Azure OpenAI task from environment variables if not specified.
`base_url`	a character string for the base url. It defaults to the Azure OpenAI endpoint from environment variables if not specified.
`deployment_name`	a character string for the deployment name. It will default to the Azure OpenAI deployment name from environment variables if not specified.
`api_key`	a character string for the API key. It will default to the Azure OpenAI API key from your environment variables if not specified.
`api_version`	a character string for the API version. It will default to the Azure OpenAI API version from your environment variables if not specified.

Value

a list with the generated completions and other information returned by the API

create_ide_matching_colors

Chat message colors in RStudio

Description

This returns a list of color properties for a chat message

Usage

```
create_ide_matching_colors(
  role = c("user", "assistant"),
  ide_colors = get_ide_theme_info()
)
```

Arguments

role	The role of the message author
ide_colors	List containing the colors of the IDE theme.

Value

list

create_translator

Internationalization for the ChatGPT addin

Description

The language can be set via `options("gptstudio.language" = "<language>")` (defaults to "en").

Usage

```
create_translator(language = getOption("gptstudio.language"))
```

Arguments

language	The language to be found in the translation JSON file.

Value

A Translator from `shiny.i18n::Translator`

ellmer_chat *Create Chat Client for Different API Providers*

Description

This function provides a generic interface for creating chat clients for different API providers (e.g., OpenAI, HuggingFace, Google AI Studio). It dispatches the actual client creation to the relevant method based on the `class` of the `skeleton` argument.

Usage

```
ellmer_chat(skeleton, all_turns)
```

Arguments

`skeleton`	A `gptstudio_request_skeleton` object containing API configuration
`all_turns`	A list of conversation turns formatted for the ellmer package

Value

An ellmer chat client object for the specific API provider

get_all_available_services *List supported service providers*

Description

List the names of all providers/vendors supported by the gptstudio chat app.

Usage

```
get_all_available_services()
```

Value

A character vector

Examples

```
get_all_available_services()
```

get_available_models *List supported models*

Description

Get a list of the models supported by the selected service.

Usage

```
get_available_models(service)
```

Arguments

service	The API service

Value

A character vector

See Also

get_all_available_services()

Examples

```
## Not run:
get_available_models()

## End(Not run)
```

get_ide_theme_info *Get IDE Theme Information*

Description

Retrieves the current RStudio IDE theme information including whether it is a dark theme, and the background and foreground colors in hexadecimal format.

Usage

```
get_ide_theme_info()
```

Value

A list with the following components:

- `is_dark` A logical indicating whether the current IDE theme is dark.
- `bg` A character string representing the background color of the IDE theme in hex format.
- `fg` A character string representing the foreground color of the IDE theme in hex format.

If RStudio is unavailable, returns the fallback theme details.

Examples

```
theme_info <- get_ide_theme_info()
print(theme_info)
```

---

gptstudio_cache_directory *a function that determines the appropriate directory to cache a token*

Description

a function that determines the appropriate directory to cache a token

Usage

```
gptstudio_cache_directory()
```

---

gptstudio_chat *Run GPTStudio Chat App*

Description

This function initializes and runs the Chat GPT Shiny App as a background job in RStudio and opens it in the viewer pane or browser window.

Usage

```
gptstudio_chat(host = getOption("shiny.host", "127.0.0.1"))
```

Arguments

- `host` A character string specifying the host on which to run the app. Defaults to the value of `getOption("shiny.host", "127.0.0.1")`.

Details

The function performs the following steps:

1. Verifies that RStudio API is available.
2. Finds an available port for the Shiny app.
3. Creates a temporary directory for the app files.
4. Runs the app as a background job in RStudio.
5. Opens the app in the RStudio viewer pane or browser window.

Value

This function does not return a value. It runs the Shiny app as a side effect.

Note

This function is designed to work within the RStudio IDE and requires the rstudioapi package.

Examples

```
## Not run:
gptstudio_chat()

## End(Not run)
```

---

gptstudio_chat_in_source_addin *ChatGPT in Source*

Description

Call this function as a Rstudio addin to ask GPT to improve spelling and grammar of selected text.

Usage

```
gptstudio_chat_in_source_addin()
```

Value

This function has no return value.

Examples

```
# Select some text in a source file
# Then call the function as an RStudio addin
## Not run:
gptstudio_chat_in_source()

## End(Not run)
```

gptstudio_comment_code

Comment Code Addin

Description

Call this function as a Rstudio addin to ask GPT to add comments to your code

Usage

```
gptstudio_comment_code()
```

Value

This function has no return value.

Examples

```
# Open a R file in Rstudio
# Then call the function as an RStudio addin
## Not run:
gptstudio_comment_code()

## End(Not run)
```

gptstudio_create_skeleton

Create a Request Skeleton

Description

This function dynamically creates a request skeleton for different AI text generation services.

Usage

```
gptstudio_create_skeleton(
  service = "openai",
  prompt = "Name the top 5 packages in R.",
  history = list(list(role = "system", content = "You are an R chat assistant")),
  stream = TRUE,
  model = "gpt-4o-mini",
  ...
)
```

Arguments

`service`	The text generation service to use. Currently supports "openai", "huggingface", "anthropic", "google", "azure_openai", "ollama", and "perplexity".
`prompt`	The initial prompt or question to pass to the text generation service.
`history`	A list indicating the conversation history, where each element is a list with elements "role" (who is speaking; e.g., "system", "user") and "content" (what was said).
`stream`	Logical; indicates if streaming responses should be used. Currently, this option is not supported across all services.
`model`	The specific model to use for generating responses. Defaults to "gpt-3.5-turbo".
`...`	Additional arguments passed to the service-specific skeleton creation function.

Value

Depending on the selected service, returns a list that represents the configured request ready to be passed to the corresponding API.

Examples

```
## Not run:
request_skeleton <- gptstudio_create_skeleton(
  service = "openai",
  prompt = "Name the top 5 packages in R.",
  history = list(list(role = "system", content = "You are an R assistant")),
  stream = TRUE,
  model = "gpt-3.5-turbo"
)

## End(Not run)
```

---

gptstudio_request_perform

Perform API Request

Description

This function provides a wrapper for calling different APIs (e.g., OpenAI, HuggingFace, Google AI Studio). It dispatches the actual API calls to the relevant ellmer chat.

Usage

```
gptstudio_request_perform(skeleton, shiny_session = NULL)
```

Arguments

skeleton	A gptstudio_request_skeleton object
shiny_session	Shiny session to send messages to. Only relevant if skeleton$stream is TRUE.

Value

A list with a skeleton and and the last response

---

gptstudio_run_chat_app *Run the ChatGPT app*

Description

This starts the chatgpt app. It is exported to be able to run it from an R script.

Usage

```
gptstudio_run_chat_app(
  ide_colors = get_ide_theme_info(),
  code_theme_url = get_highlightjs_theme(),
  host = getOption("shiny.host", "127.0.0.1"),
  port = getOption("shiny.port")
)
```

Arguments

ide_colors	List containing the colors of the IDE theme.
code_theme_url	URL to the highlight.js theme
host	The IPv4 address that the application should listen on. Defaults to the shiny.host option, if set, or "127.0.0.1" if not. See Details.
port	The TCP port that the application should listen on. If the port is not specified, and the shiny.port option is set (with options(shiny.port = XX)), then that port will be used. Otherwise, use a random port between 3000:8000, excluding ports that are blocked by Google Chrome for being considered unsafe: 3659, 4045, 5060, 5061, 6000, 6566, 6665:6669 and 6697. Up to twenty random ports will be tried.

Value

Nothing.

gptstudio_sitrep *Current Configuration for gptstudio*

Description

This function prints out the current configuration settings for gptstudio and checks API connections if verbose is TRUE.

Usage

```
gptstudio_sitrep(verbose = TRUE)
```

Arguments

verbose	Logical value indicating whether to output additional information, such as API connection checks. Defaults to TRUE.

Value

Invisibly returns NULL, as the primary purpose of this function is to print to the console.

Examples

```
## Not run:
gptstudio_sitrep(verbose = FALSE) # Print basic settings, no API checks
gptstudio_sitrep() # Print settings and check API connections

## End(Not run)
```

gptstudio_spelling_grammar *Spelling and Grammar Addin*

Description

Call this function as a Rstudio addin to ask GPT to improve spelling and grammar of selected text.

Usage

```
gptstudio_spelling_grammar()
```

Value

This function has no return value.

Examples

```
# Select some text in Rstudio
# Then call the function as an RStudio addin
## Not run:
gptstudio_spelling_grammar()

## End(Not run)
```

---

input_audio_clip — *An audio clip input control that records short audio clips from the microphone*

Description

An audio clip input control that records short audio clips from the microphone

Usage

```
input_audio_clip(
  id,
  record_label = "Record",
  stop_label = "Stop",
  reset_on_record = TRUE,
  mime_type = NULL,
  audio_bits_per_second = NULL,
  show_mic_settings = TRUE,
  ...
)
```

Arguments

`id`	The input slot that will be used to access the value.
`record_label`	Display label for the "record" control, or NULL for no label. Default is 'Record'.
`stop_label`	Display label for the "stop" control, or NULL for no label. Default is 'Record'.
`reset_on_record`	Whether to reset the audio clip input value when recording starts. If TRUE, the audio clip input value will become NULL at the moment the Record button is pressed; if FALSE, the value will not change until the user stops recording. Default is TRUE.
`mime_type`	The MIME type of the audio clip to record. By default, this is NULL, which means the browser will choose a suitable MIME type for audio recording. Common MIME types include 'audio/webm' and 'audio/mp4'.
`audio_bits_per_second`	The target audio bitrate in bits per second. By default, this is NULL, which means the browser will choose a suitable bitrate for audio recording. This is only a suggestion; the browser may choose a different bitrate.

show_mic_settings	Whether to show the microphone settings in the settings menu. Default is TRUE.
...	Additional parameters to pass to the underlying HTML tag.

Value

An audio clip input control that can be added to a UI definition.

`mod_app_server` *App Server*

Description

App Server

Usage

```
mod_app_server(id, ide_colors = get_ide_theme_info())
```

Arguments

id	id of the module
ide_colors	List containing the colors of the IDE theme.

`mod_app_ui` *App UI*

Description

App UI

Usage

```
mod_app_ui(
  id,
  ide_colors = get_ide_theme_info(),
  code_theme_url = get_highlightjs_theme()
)
```

Arguments

id	id of the module
ide_colors	List containing the colors of the IDE theme.
code_theme_url	URL to the highlight.js theme

mod_chat_server *Chat server*

Description

Chat server

Usage

```
mod_chat_server(
  id,
  ide_colors = get_ide_theme_info(),
  translator = create_translator(),
  settings,
  history
)
```

Arguments

id	id of the module
ide_colors	List containing the colors of the IDE theme.
translator	Translator from `shiny.i18n::Translator`
settings, history	Reactive values from the settings and history module

mod_chat_ui *Chat UI*

Description

Chat UI

Usage

```
mod_chat_ui(
  id,
  translator = create_translator(),
  code_theme_url = get_highlightjs_theme()
)
```

Arguments

id	id of the module
translator	A Translator from `shiny.i18n::Translator`
code_theme_url	URL to the highlight.js theme

multimodal_dep *Create HTML dependency for multimodal component*

Description

Create HTML dependency for multimodal component

Usage

```
multimodal_dep()
```

parse_data_uri *Parse a Data URI*

Description

This function parses a data URI and returns the MIME type and decoded data.

Usage

```
parse_data_uri(data_uri)
```

Arguments

data_uri	A string. The data URI to parse.

Value

A list with two elements: 'mime_type' and 'data'.

prepare_chat_history *Prepare chat completion prompt*

Description

This function prepares the chat completion prompt to be sent to the OpenAI API. It also generates a system message according to the given parameters and inserts it at the beginning of the conversation.

Usage

```
prepare_chat_history(
  history = list(list(role = "system", content = "You are an R chat assistant")),
  style = getOption("gptstudio.code_style"),
  skill = getOption("gptstudio.skill"),
  task = getOption("gptstudio.task", "coding"),
  custom_prompt = NULL
)
```

Arguments

`history`	A list of previous messages in the conversation. This can include roles such as 'system', 'user', or 'assistant'. System messages are discarded. Default is NULL.i
`style`	The style of code to use. Applicable styles can be retrieved from the "gptstudio.code_style" option. Default is the "gptstudio.code_style" option. Options are "base", "tidyverse", or "no preference".
`skill`	The skill level of the user for the chat conversation. This can be set through the "gptstudio.skill" option. Default is the "gptstudio.skill" option. Options are "beginner", "intermediate", "advanced", and "genius".
`task`	Specifies the task that the assistant will help with. Default is "coding". Others are "general", "advanced developer", and "custom".
`custom_prompt`	This is a custom prompt that may be used to guide the AI in its responses. Default is NULL. It will be the only content provided to the system prompt.

Value

A list where the first entry is an initial system message followed by any non-system entries from the chat history.

`rgb_str_to_hex` *RGB str to hex*

Description

RGB str to hex

Usage

```
rgb_str_to_hex(rgb_string)
```

Arguments

`rgb_string`	The RGB string as returned by `rstudioapi::getThemeInfo()`

Value

hex color

set_allowed_models *Set allowed models by provider*

Description

[Experimental]

Sometimes it is useful to restrict the list of options in the model selection dropdown of the chat app. This function will check against `get_available_models()` to restrict the list to models that are actually available.

Usage

```
set_allowed_models(service, models = NULL)
```

Arguments

`service`	The API service
`models`	A character vector containing the list of allowed models that should be shown in the dropdown selector. If `NULL` (default), all models will be available.

streamingMessage *Streaming message*

Description

Places an invisible empty chat message that will hold a streaming message. It can be reset dynamically inside a shiny app

Usage

```
streamingMessage(
  ide_colors = get_ide_theme_info(),
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`ide_colors`	List containing the colors of the IDE theme.
`width, height`	Must be a valid CSS unit (like `'100%'`, `'400px'`, `'auto'`) or a number, which will be coerced to a string and have `'px'` appended.
`element_id`	The element's id

streamingMessage-shiny

Shiny bindings for streamingMessage

Description

Output and render functions for using streamingMessage within Shiny applications and interactive Rmd documents.

Usage

```
streamingMessageOutput(outputId, width = "100%", height = NULL)

renderStreamingMessage(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`outputId`	output variable to read from
`width, height`	Must be a valid CSS unit (like `'100%'`, `'400px'`, `'auto'`) or a number, which will be coerced to a string and have `'px'` appended.
`expr`	An expression that generates a streamingMessage
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable.

style_chat_history *Style Chat History*

Description

This function processes the chat history, filters out system messages, and formats the remaining messages with appropriate styling.

Usage

```
style_chat_history(history, ide_colors = get_ide_theme_info())
```

Arguments

`history`	A list of chat messages with elements containing 'role' and 'content'.
`ide_colors`	List containing the colors of the IDE theme.

Value

A list of formatted chat messages with styling applied, excluding system messages.

Examples

```
chat_history_example <- list(
  list(role = "user", content = "Hello, World!"),
  list(role = "system", content = "System message"),
  list(role = "assistant", content = "Hi, how can I help?")
)

## Not run:
style_chat_history(chat_history_example)

## End(Not run)
```

---

style_chat_message *Style chat message*

Description

Style a message based on the role of its author.

Usage

```
style_chat_message(message, ide_colors = get_ide_theme_info())
```

Arguments

message	A chat message.
ide_colors	List containing the colors of the IDE theme.

Value

An HTML element.

---

text_area_input_wrapper *Custom textAreaInput*

Description

Modified version of `textAreaInput()` that removes the label container. It's used in `mod_prompt_ui()`

Usage

```
text_area_input_wrapper(
  inputId,
  label,
  value = "",
  width = NULL,
  height = NULL,
  cols = NULL,
  rows = NULL,
  placeholder = NULL,
  resize = NULL,
  textarea_class = NULL
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
value	Initial value.
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
height	The height of the input, e.g. '400px', or '100%'; see validateCssUnit().
cols	Value of the visible character columns of the input, e.g. 80. This argument will only take effect if there is not a CSS width rule defined for this element; such a rule could come from the width argument of this function or from a containing page layout such as fluidPage().
rows	The value of the visible character rows of the input, e.g. 6. If the height argument is specified, height will take precedence in the browser's rendering.
placeholder	A character string giving the user a hint as to what can be entered into the control. Internet Explorer 8 and 9 do not support this option.
resize	Which directions the textarea box can be resized. Can be one of "both", "none", "vertical", and "horizontal". The default, NULL, will use the client browser's default setting for resizing textareas.
textarea_class	Class to be applied to the textarea element

Value

A modified textAreaInput

transcribe_audio *Transcribe Audio from Data URI Using OpenAI's Whisper Model*

Description

This function takes an audio file in data URI format, converts it to WAV, and sends it to OpenAI's transcription API to get the transcribed text.

Usage

```
transcribe_audio(audio_input, api_key = Sys.getenv("OPENAI_API_KEY"))
```

Arguments

audio_input	A string. The audio data in data URI format.
api_key	A string. Your OpenAI API key. Defaults to the OPENAI_API_KEY environment variable.

Value

A string containing the transcribed text.

Examples

```
## Not run:
audio_uri <- "data:audio/webm;base64,SGVsbG8gV29ybGQ=" # Example data URI
transcription <- transcribe_audio(audio_uri)
print(transcription)

## End(Not run)
```

welcomeMessage *Welcome message*

Description

HTML widget for showing a welcome message in the chat app. This has been created to be able to bind the message to a shiny event to trigger a new render.

Usage

```
welcomeMessage(
  ide_colors = get_ide_theme_info(),
  translator = create_translator(),
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`ide_colors`	List containing the colors of the IDE theme.
`translator`	A Translator from `shiny.i18n::Translator`
`width, height`	Must be a valid CSS unit (like `'100%'`, `'400px'`, `'auto'`) or a number, which will be coerced to a string and have `'px'` appended.
`element_id`	The element's id

`welcomeMessage-shiny` *Shiny bindings for welcomeMessage*

Description

Output and render functions for using welcomeMessage within Shiny applications and interactive Rmd documents.

Usage

```
welcomeMessageOutput(outputId, width = "100%", height = NULL)

renderWelcomeMessage(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`outputId`	output variable to read from
`width, height`	Must be a valid CSS unit (like `'100%'`, `'400px'`, `'auto'`) or a number, which will be coerced to a string and have `'px'` appended.
`expr`	An expression that generates a welcomeMessage
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable.

Index

Buffer, 3

chat, 4
chat_create_system_prompt, 6
chat_history_append, 7
chat_message_default, 7
check_api_connection, 8
create_chat_app_theme, 8
create_completion_azure_openai, 9
create_ide_matching_colors, 10
create_translator, 10

ellmer_chat, 11

fluidPage(), *27*

get_all_available_services, 11
get_all_available_services(), *12*
get_available_models, 12
get_ide_theme_info, 12
gptstudio_cache_directory, 13
gptstudio_chat, 13
gptstudio_chat_in_source_addin, 14
gptstudio_comment_code, 15
gptstudio_create_skeleton, 15
gptstudio_request_perform, 16
gptstudio_run_chat_app, 17
gptstudio_sitrep, 18
gptstudio_spelling_grammar, 18

input_audio_clip, 19

mod_app_server, 20
mod_app_ui, 20
mod_chat_server, 21
mod_chat_ui, 21
multimodal_dep, 22

parse_data_uri, 22
prepare_chat_history, 22

renderStreamingMessage
(streamingMessage-shiny), 25
renderWelcomeMessage
(welcomeMessage-shiny), 29
rgb_str_to_hex, 23

set_allowed_models, 24
streamingMessage, 24
streamingMessage-shiny, 25
streamingMessageOutput
(streamingMessage-shiny), 25
style_chat_history, 25
style_chat_message, 26

text_area_input_wrapper, 26
transcribe_audio, 28

validateCssUnit(), *27*

welcomeMessage, 28
welcomeMessage-shiny, 29
welcomeMessageOutput
(welcomeMessage-shiny), 29