GLaDOS

Release 0.0.1.dev24

Jun 04, 2020

Contents

1 Installation **1**

2 GLaDOS Structure **3**
2.1 GLaDOS 3
2.2 Bot 3
2.3 Plugin 3
2.4 Routing 3
2.4.1 Route Types 4
2.4.2 Route 4

3 GLaDOS Plugin **5**

4 glados package **7**
4.1 Subpackages 14
4.1.1 glados.slack_classes package 14
4.1.1.1 Submodules 14
4.1.1.2 glados.slack_classes.views module 14
4.2 Submodules 14
4.3 glados.bot module 14
4.4 glados.configs module 15
4.5 glados.core module 16
4.6 glados.datastore module 17
4.7 glados.errors module 20
4.8 glados.message_blocks module 21
4.9 glados.plugin module 23
4.10 glados.request module 26
4.11 glados.route_type module 28
4.12 glados.router module 29
4.13 glados.utils module 30

5 Indices and tables **33**

Python Module Index **35**

Index **37**

CHAPTER 1

Installation

The easiest way to install GLaDOS is by using pip

```
$ pip3 install glados
```

CHAPTER 2

GLaDOS Structure

There are a few major components of GLaDOS as shows below.

2.1 GLaDOS

2.2 Bot

GLaDOS can support multiple Slack bots and applications at once. When you initialize a plugin you pass the Glados-Bot object of the bot you would like to be responsible for the actions of that plugin.

2.3 Plugin

Every GLaDOS plugin is its own module that is imported into your main handler application. This allows GLaDOS plugins to be community based and installed as wanted just by a simple import statement.

Note: V2: Each GLaDOS Plug-in has to have the register_plugin decorator and match the Plugin Interface. The easiest way to do this is to use the Plugin Class that is provided with GLaDOS. In the main GLaDOS function you will have to import all of your plugin modules and kick off a scan for plugins. This same method applies for GLaDOS Bots.

2.4 Routing

GLaDOS event routing is done in two different parts: *Route Type* and *Route*. These two parts and sometimes the bot name combined together control how events sent to GLaDOS are routed to the respective plugins

2.4.1 Route Types

Route Types represent the type of action that is being requested. One way to look at Route Types is that they would be the first path in your API structure. For example you would configure Slack to send *Slash* commands to `https://slack.glados.wtf/Slash/myCommand`

- **SendMessage**: Sample action to send a message as the bot.
- **Slash**: This is the route called when a slash command is called.

Note: The Route Types below use the bot name as a prefix of the route (see blow).

- **Event**: This route type is used to receive messages from subscribed events in Slack.
- **Interaction**: This route type is used to receive messages from user interactions.

2.4.2 Route

Routes represent the action that is being requested. Routes are not always included in the URL configured in Slack.

Interactive component callbacks are an example of this. The *Request URL* of for interactive components should be set to something like `https://slack.glados.wtf/Interaction/myBot` and lets say that *action_id* is *myBottonPress* then the full route would be `myBot_myButtonPress`

Depending on the type of action sometimes the routes will be automatically prefixed with the bot name that is responsible for handling the request.

CHAPTER 3

GLaDOS Plugin

The GLaDOS Plugin has only a few requirements for setup and installation.

CHAPTER 4

glados package

class glados.**Glados**(*config_file=None*, *plugins_folder=None*, *bots_config_dir=None*, *plugins_config_dir=None*)

Bases: `object`

Glados is the core of the GLaDOS package.

Parameters

- **config_file** (`Optional[str]`) – path to config file
- **plugins_folder** (`Optional[str]`) – path to plugins folder
- **bots_config_dir** (`Optional[str]`) – path to bots config folder
- **plugins_config_dir** (`Optional[str]`) – path to plugin config folder.

Notes

If `config_file` is passed in and the file has `plugins_folder`, `bots_config_dir`, `plugins_config_dir` in it , then the other parameters are not required

add_bot(*bot*)

Add a new bot to GLaDOS.

Parameters **bot** (`GladosBot`) – the bot to be added to GLaDOS

Return type `NoReturn`

add_plugin(*plugin*)

Add a plugin to GLaDOS

Parameters **plugin** (`GladosPlugin`) – the plugin to be added to GLaDOS

Return type `NoReturn`

has_datastore()

Returns True if there is a datastore else False

Return type `bool`

import_bots()
Import all discovered bots

Return type `NoReturn`

import_plugins(*bot_name=None*)
Import all discovered plugins and add them to the plugin list.

Parameters **bot_name** (`Optional[str]`) – If set GLaDOS will only import the bot name that is provided here.

Return type `NoReturn`

read_config(*bot_name=None*)
Read the GLaDOS config file. If a bot name is provided it will only install that bot. Else it will install all bots.

Parameters **bot_name** (`Optional[str]`) – If provided, install only the bot with this name.

Return type `NoReturn`

request(*request*)
Send a request to GLaDOS. This returns whatever the plugin returns.

This function will also set the datastore session for the request, try to find the interaction in the datastore and fetch it. This info is available in the request.

Parameters **request** (`GladosRequest`) – the request to be sent to GLaDOS

class glados.**GladosBot**(*token*, *name*, *signing_secret=None*, ***kwargs*)
Bases: `object`

GLaDOS Bot represents all the required data and functions for a Slack bot.

Notes

All Slack Web API functions can be called from MyBot.client.*

Parameters

- **name** (`str`) – The name of the bot (URL Safe)
- **token** (`Union[str, Dict[str, str]]`) – The bot token
- **signing_secret** (`Union[str, Dict[str, str], None]`) – The bot signing secret.

name
The name of the bot (URL Safe)

Type str

token
The bot token

Type str

client
A Slack client generated for that bot

Type WebClient

signing_secret
The bots signing secret.

Type str

delete_message(*channel*, *ts*)

Deletes a message that was sent by a bot

Parameters

- **channel** (`str`) –
- **ts** (`str`) –

Return type `MockObject`

send_message(*channel*, *message*)

Send a message as the bot

Parameters

- **channel** (`str`) – channel to send the message to
- **message** (`MockObject`) – message object to send

Return type `MockObject`

update_message(*channel*, *ts*, *message*)

Updates a message that was sent by the bot

Parameters

- **channel** (`str`) –
- **ts** (`str`) –
- **message** (`MockObject`) –

Return type `MockObject`

validate_slack_signature(*request*)

Parameters **request** (`GladosRequest`) –

class glados.**GladosRequest**(*route_type*, *route=None*, *slack_verify=None*, *bot_name=None*, *json=None*, *data=None*, ***kwargs*)

Bases: `object`

GLaDOS Request Object. This holds all the data required to process the request.

Parameters

- **route_type** (`RouteType`) – what type of route is this
- **route** (`Optional[str]`) – what is the route to be called
- **slack_verify** (`Optional[SlackVerification]`) – slack data used for verifying the request came from Slack
- **bot_name** (`Optional[str]`) – The name of the bot to send the request to. This is used for select RouteTypes
- **json** (`Union[str, dict, None]`) – the json paylod of the request
- **data** (`Optional[dict]`) – data to send with the request. This should be from a database
- **kwargs** –

Examples

```
>>> request = GladosRequest(RouteType.SendMessage, "send_mock", json={"message":
↪"my message"})
>>> print(request.json.message)
my message
>>> try:
...     print(request.json.other_param)
... except AttributeError:
...     print("ERROR")
ERROR
```

add_interaction_to_datastore(*interaction*)
Add an interaction to the datastore and return the updated interaction.

Notes

The interaction_id can be retrieved by doing interaction.interaction_id

Parameters **interaction** (`DataStoreInteraction`) – the interaction to be added

Return type `Optional[DataStoreInteraction]`

close_session()
Close session for request

Return type `NoReturn`

data
Returns the data object of the request

Return type `PyJSON`

data_blob
Returns the raw dict of the data object

Return type `dict`

gen_new_interaction(*, *followup_action=None*, *followup_ts=None*, *ttl=None*, *data=None*, *auto_link=True*, *auto_set=True*)
Generate a new interaction object and set it as new_interaction.

Parameters

- **followup_action** –
- **followup_ts** –
- **ttl** –
- **data** –
- **auto_link** (`bool`) – set this request to auto-link using the return payload. The return payload must be the response from sending a slack message.
- **auto_set** (`bool`) – set this new interaction object as the request new_interaction

Return type `DataStoreInteraction`

has_interaction()
Check if request has interaction.

Return type `bool`

`has_new_interaction``()`
check if request has a new interaction object.

Return type `bool`

`interaction`
Returns the interaction for the request

Return type `Optional[DataStoreInteraction]`

`interaction_id`
Returns the interaction_id of request.interaction

Return type `Optional[str]`

`link_interaction_to_message``(`*interaction_id*, *channel*, *message_ts*`)`
Link interaction to message

Parameters

- **`interaction_id`** (`str`) – interaction ID to link
- **`channel`** (`str`) – channel to be linked to
- **`message_ts`** (`datetime`) – ts to be linked to

Return type `NoReturn`

`link_interaction_to_message_response``(`*interaction_id*, *message_response*`)`
Link interaction to message response

Parameters

- **`interaction_id`** (`str`) – interaction ID to be linked
- **`message_response`** (`dict`) – JSON payload response from sending message on slack.

Return type `NoReturn`

`rollback_session``()`
Rollback the session.

Return type `NoReturn`

`route`
the actual route

If the route automatically prefixed the route with the bot name, it will return the route with the prefix

Return type `str`

`set_datastore``(`*datastore*`)`
Set the Datastore and session for the request.

Parameters **`datastore`** (`DataStore`) – Datastore to use. This datastore will be used to create the session.

Return type `NoReturn`

`set_interaction_from_datastore``()`
Get the interaction object from the datastore.

Return type `NoReturn`

`set_session``(`*session*`)`
Set the session for this request.

Parameters **session** (`Session`) – session to use for this request.

Raises `ConnectionError` – If the session is not active raise a ConnectionError

Return type `NoReturn`

class `glados.`**RouteType**

Bases: `enum.Enum`

An enumeration.

Callback = 3

Events = 5

Interaction = 6

Menu = 7

Response = 2

Slash = 4

Webhook = 1

class `glados.`**EventRoutes**

Bases: `enum.Enum`

An enumeration.

app_home_opened = 1

message = 2

class `glados.`**GladosPlugin**(*config*, *bot*, ***kwargs*)

Bases: `object`

Parent class for a GLaDOS Plugin

Parameters

- **config** (`PluginConfig`) – PluginConfig object for the plugin.
- **bot** (`GladosBot`) – the GLaDOS bot that this plugin will use

add_route(*route_type*, *route*, *function*)

Add a new route to the plugin

Parameters

- **route_type** (`RouteType`) – what type of route this is this
- **route** (`Union`[`EventRoutes`, `str`]) – what is the route to be added
- **function** (`Callable`) – the function to be executed when this route runs

Return type `NoReturn`

has_route(*route*)

See if route exists.

Parameters **route** (*`route to check`*) –

Returns

Return type True if route exists else false

respond_to_url(*request*, *text*, ***kwargs*)

When you click on a link that was sent via slack it sends a callback, This is to handle that

Parameters

- **request** (GladosRequest) –
- **text** (str) –

routes

List all routes for the plugin.

Return type List[GladosRoute]

send_request (*request*, ***kwargs*)

This is the function to be called when sending a request to a plugin.

This function is responsible for validating the slack signature if needed. It also returns and empty string if the function called returns None.

Parameters

- **request** (GladosRequest) – the request object to be sent
- **kwargs** –

Return type Any

class glados.**GladosConfig** (*config_file*)

Bases: object

Parameters **config_file** (str) –

read_config ()

Read the config file into a config object

sections

what sections are there in the config file

Returns

Return type sorted list of sections in the yaml file

glados.**read_config** (*config_file*)

Parameters **config_file** (str) –

glados.**check_for_env_vars** (*value*)

Check an input value to see if it is an env_var or enc_env_var and get the value.

Parameters **value** (Union[str, dict]) – input to check.

Returns Returns the value of the var from either the passed in value, or the env var value.

Return type Any

Raises KeyError if the env var is not set for what youre tying to get.

4.1 Subpackages

4.1.1 glados.slack_classes package

4.1.1.1 Submodules

4.1.1.2 glados.slack_classes.views module

class `glados.slack_classes.views.`**`Home`**`(*, blocks=None)`
Bases: `sphinx.ext.autodoc.importer._MockObject`

`add_block``(block)`

Parameters **`block`** (`MockObject`) –

`attributes = {'home'}`

`to_dict``(*args)`

Return type `dict`

4.2 Submodules

4.3 glados.bot module

class `glados.bot.`**`BotImporter`**`(bots_dir)`
Bases: `object`

Parameters **`bots_dir`** (`str`) –

`import_bots``()`
Import all bots in the bots config folder

class `glados.bot.`**`GladosBot`**`(token, name, signing_secret=None, **kwargs)`
Bases: `object`

GLaDOS Bot represents all the required data and functions for a Slack bot.

Notes

All Slack Web API functions can be called from MyBot.client.*

Parameters

- **`name`** (`str`) – The name of the bot (URL Safe)
- **`token`** (`Union[str, Dict[str, str]]`) – The bot token
- **`signing_secret`** (`Union[str, Dict[str, str], None]`) – The bot signing secret.

`name`
The name of the bot (URL Safe)

Type str

`token`
The bot token

Type str

client
A Slack client generated for that bot

Type WebClient

signing_secret
The bots signing secret.

Type str

delete_message(*channel*, *ts*)
Deletes a message that was sent by a bot

Parameters

- **channel** (`str`) –
- **ts** (`str`) –

Return type `MockObject`

send_message(*channel*, *message*)
Send a message as the bot

Parameters

- **channel** (`str`) – channel to send the message to
- **message** (`MockObject`) – message object to send

Return type `MockObject`

update_message(*channel*, *ts*, *message*)
Updates a message that was sent by the bot

Parameters

- **channel** (`str`) –
- **ts** (`str`) –
- **message** (`MockObject`) –

Return type `MockObject`

validate_slack_signature(*request*)

Parameters **request** (`GladosRequest`) –

4.4 glados.configs module

class `glados.configs.`**GladosConfig**(*config_file*)
Bases: `object`

Parameters **config_file** (`str`) –

read_config()
Read the config file into a config object

sections
what sections are there in the config file

Returns

Return type sorted list of sections in the yaml file

`glados.configs.`**`read_config`**(*config_file*)

Parameters **`config_file`** (`str`) –

4.5 glados.core module

class `glados.core.`**`Glados`**(*config_file=None*, *plugins_folder=None*, *bots_config_dir=None*, *plugins_config_dir=None*)

Bases: `object`

Glados is the core of the GLaDOS package.

Parameters

- **`config_file`** (`Optional[str]`) – path to config file
- **`plugins_folder`** (`Optional[str]`) – path to plugins folder
- **`bots_config_dir`** (`Optional[str]`) – path to bots config folder
- **`plugins_config_dir`** (`Optional[str]`) – path to plugin config folder.

Notes

If `config_file` is passed in and the file has `plugins_folder`, `bots_config_dir`, `plugins_config_dir` in it , then the other parameters are not required

`add_bot`(*bot*)

Add a new bot to GLaDOS.

Parameters **`bot`** (`GladosBot`) – the bot to be added to GLaDOS

Return type `NoReturn`

`add_plugin`(*plugin*)

Add a plugin to GLaDOS

Parameters **`plugin`** (`GladosPlugin`) – the plugin to be added to GLaDOS

Return type `NoReturn`

`has_datastore`()

Returns True if there is a datastore else False

Return type `bool`

`import_bots`()

Import all discovered bots

Return type `NoReturn`

`import_plugins`(*bot_name=None*)

Import all discovered plugins and add them to the plugin list.

Parameters **`bot_name`** (`Optional[str]`) – If set GLaDOS will only import the bot name that is provided here.

Return type `NoReturn`

read_config(*bot_name=None*)
Read the GLaDOS config file. If a bot name is provided it will only install that bot. Else it will install all bots.

Parameters **bot_name** (`Optional[str]`) – If provided, install only the bot with this name.

Return type `NoReturn`

request(*request*)
Send a request to GLaDOS. This returns whatever the plugin returns.

This function will also set the datastore session for the request, try to find the interaction in the datastore and fetch it. This info is available in the request.

Parameters **request** (`GladosRequest`) – the request to be sent to GLaDOS

4.6 glados.datastore module

class `glados.datastore.`**DataStore**(*host*, *username*, *password*, *port=5432*, *database='glados'*)
Bases: `object`

DataStore is how GLaDOS stores async data.

Parameters

- **host** (`str`) – postgres host.
- **username** (`str`) – postgres username.
- **password** (`str`) – postgres password.
- **port** (`int`) – postgres port.
- **database** (`str`) – postgres database to use.

create_session()
Generate a new session with the existing connection.

Return type `Session`

create_table(*tables=None*, *force=False*)
Create the table.

If you set force to True then it will drop the existing tables and then recreate them. ALL DATA WILL BE LOST

Parameters

- **tables** (`Optional[List[str]]`) – only take action on these tables. If None, then take action on all tables
- **force** (`bool`) – drop existing tables and rebuild. (default: False)

Return type `NoReturn`

drop_table(*table='interactions'*, *force=False*)
Drop the GLaDOS table so that it can be re-created.

Parameters

- **table** (`str`) – table name to use.
- **force** (`bool`) – if True will fill force drop the table without checks.

Return type `NoReturn`

`find_by_id`(*interaction_id*, *session*)
Find an interaction by interaction_id.

Parameters

- **`interaction_id`** (`str`) – interaction ID to find
- **`session`** (`Session`) – session to be used

Return type `DataStoreInteraction`

`find_interaction_by_channel_ts`(*channel*, *ts*, *session*)
Find the interaction in the datastore by channel and message ts.

Parameters

- **`channel`** (`str`) – channel of the interaction youre looking for
- **`ts`** (`datetime`) – ts of the interaction you are looking for
- **`session`** (`Session`) – session to be used

Raises `ReferenceError` – There were more than one interaction that matched the channel and message_ts

Return type `Optional[DataStoreInteraction]`

`insert_interaction`(*interaction*, *session*)
Insert an interaction object into the database.

Parameters

- **`interaction`** (`DataStoreInteraction`) – The row to be inserted
- **`session`** (`Session`) – session to be used

Return type `NoReturn`

`link_to_message`(*interaction_id*, *channel*, *ts*, *session*)
Link to message by setting message ts and channel.

Parameters

- **`interaction_id`** (`str`) – interaction ID to link
- **`channel`** (`str`) – channel to link interaction to
- **`ts`** (`datetime`) – ts to link interaction to
- **`session`** (`Session`) – session to be used

Return type `NoReturn`

`link_to_message_response`(*interaction_id*, *message_response*, *session*)
Add info from the Slack message into the database for the interaction.

Parameters

- **`interaction_id`** (`str`) – The interaction ID that was returned on adding the message to the database.
- **`message_response`** (`dict`) – The raw message response from slack. The channel and ts will be pulled from this.
- **`session`** (`Session`) – session to be used

Return type `NoReturn`

table_exists(*table='interactions'*)
Check to see if the GLaDOS table is found in postgres.

Parameters **table** (`str`) – table name to use.

Return type `bool`

update_interaction(*interaction_id*, *session*, ***kwargs*)
Find and update an interaction with the provided values.

Parameters

- **interaction_id** (`str`) – interaction ID to update
- **session** (`Session`) – session to be used
- **kwargs** – fields and new values to update

Return type `DataStoreInteraction`

class `glados.datastore.`**DataStoreInteraction**(***kwargs*)
Bases: `sqlalchemy.ext.declarative.api.Base`

DataStoreInteraction represents a row in the datastore. This is used to update data in the datastore.

interaction_id
This is the primary key of the datastore. This is the ID of the entry in the datastore.

Type `str`

ts
This is the time the row was put into the database.

Type `datetime`

bot
This is the name of the bot it should use when completing followup actions.

Type `str`

data
Any extra data stored with the interaction. This is a JSON blob.

Type `dict`

message_channel
The channel that this interaction was sent to.

Type `str`

message_ts
The message timestamp when this interaction was sent.

Type `datetime`

ttl
How long this interaction should live for.

Type `int`

followup_ts
When should the follow up action happen.

Type `datetime`

followup_action
The action name to execute when following up. If None then no action will happen.

Type `str`

`cron_followup_action`
The action name to execute on a normal cron schedule like every 5 min. If None then no action will happen.

Type `str`

`followed_up`
This is the time when the action was followed up at. If it has not happened yet this value will be None.

Type `datetime`

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

`bot`

`cron_followup_action`

`data`

`followed_up`

`followup_action`

`followup_ts`

`interaction_id`

`message_channel`

`message_ts`

`ts`

`ttl`

`update`(***kwargs*)
Update the object dropping any arguments that are not valid

4.7 glados.errors module

exception `glados.errors.`**`GladosBotNotFoundError`**
Bases: `glados.errors.GladosError`

Error raised when GladosBot is not found

exception `glados.errors.`**`GladosError`**
Bases: `Exception`

Base class for Client errors

exception `glados.errors.`**`GladosPathExistsError`**
Bases: `glados.errors.GladosError`

Error raised when trying to add a path that already exists

exception `glados.errors.`**`GladosRouteNotFoundError`**

Bases: `glados.errors.GladosError`

Error raised when the requested path is not found

4.8 glados.message_blocks module

class `glados.message_blocks.`**`ModalBuilder`**

Bases: `sphinx.ext.autodoc.importer._MockObject`

The ModalBuilder enables you to more easily construct the JSON required to create a modal in Slack.

Modals are a focused surface to collect data from users or display dynamic and interactive information.

To learn how modals are invoked, how to compose their contents, and how to enable and handle complex interactivity read this guide:

https://api.slack.com/block-kit/surfaces/modals

`actions`(*, *elements*, *block_id=None*)

A block that is used to hold interactive elements.

https://api.slack.com/reference/block-kit/blocks#actions

Parameters

- **`elements`** (`List[MockObject]`) – Up to 5 InteractiveElement objects - buttons, date pickers, etc
- **`block_id`** (`Optional[str]`) – ID to be used for this block - autogenerated if left blank. Cannot exceed 255 characters.

`attributes = {'_blocks', '_callback_id', '_clear_on_close', '_close', '_external_id',`

`blocks_length`()

`blocks_max_length = 100`

`callback_id`(*callback_id*)

An identifier to recognize interactions and submissions of this particular modal. Don't use this to store sensitive information (use private_metadata instead).

Parameters **`callback_id`** (`str`) – must not exceed 255 characters

Return type `ModalBuilder`

`callback_id_max_length`()

`clear_on_close`(*clear_on_close*)

When set to true, clicking on the close button will clear all views in a modal and close it.

Parameters **`clear_on_close`** (`bool`) – Default is false.

Return type `ModalBuilder`

`close`(*close*)

Specify the text displayed in the close button at the bottom-right of the view.

Max length of 24 characters.

Parameters **`close`** (`str`) – must not exceed 24 characters

Return type `ModalBuilder`

`close_length`()

close_max_length = 24

context (*, *elements*, *block_id=None*)
Displays message context, which can include both images and text.

https://api.slack.com/reference/block-kit/blocks#context

Parameters

- **elements** (`List[MockObject]`) – Up to 10 ImageElements and TextObjects
- **block_id** (`Optional[str]`) – ID to be used for this block - autogenerated if left blank. Cannot exceed 255 characters.

divider (*, *block_id=None*)
A content divider, like an <hr>, to split up different blocks inside of a message.

Parameters **block_id** (`Optional[str]`) – A string acting as a unique identifier for a block. You can use this block_id when you receive an interaction payload to identify the docs-src of the action. If not specified, one will be generated. Maximum length for this field is 255 characters. block_id should be unique for each message and each iteration of a message. If a message is updated, use a new block_id.

https://api.slack.com/reference/block-kit/blocks#divider

external_id (*external_id*)
A custom identifier that must be unique for all views on a per-team basis.

Parameters **external_id** (`str`) – A unique identifier.

Return type `ModalBuilder`

image (*, *image_url*, *alt_text*, *title=None*, *block_id=None*)
A simple image block, designed to make those cat photos really pop.

https://api.slack.com/reference/block-kit/blocks#image

Parameters

- **image_url** (`str`) – Publicly hosted URL to be displayed. Cannot exceed 3000 characters.
- **alt_text** (`str`) – Plain text summary of image. Cannot exceed 2000 characters.
- **title** (`Optional[str]`) – A title to be displayed above the image. Cannot exceed 2000 characters.
- **block_id** (`Optional[str]`) – ID to be used for this block - autogenerated if left blank. Cannot exceed 255 characters.

notify_on_close (*notify_on_close*)
Indicates whether Slack will send your request URL a view_closed event when a user clicks the close button.

Parameters **notify_on_close** (`bool`) – Default is false.

Return type `ModalBuilder`

private_metadata (*private_metadata*)
An optional string that will be sent to your app in view_submission and block_actions events.

Parameters **private_metadata** (`str`) – must not exceed 3000 characters

Return type `ModalBuilder`

private_metadata_max_length ()

section(*, *text=None*, *block_id=None*, *fields=None*, *accessory=None*)

A section is one of the most flexible blocks available. It can be used as a simple text block, in combination with text fields, or side-by-side with any of the available block elements.

https://api.slack.com/reference/block-kit/blocks#section

Parameters

- **text** (`Union[str, MockObject, None]`) – The text for the block, in the form of string or a text object. Maximum length for the text in this field is 3000 characters.
- **block_id** (`Optional[str]`) – A string acting as a unique identifier for a block. You can use this block_id when you receive an interaction payload to identify the docs-src of the action. If not specified, one will be generated. Maximum length for this field is 255 characters. block_id should be unique for each message and each iteration of a message. If a message is updated, use a new block_id.
- **fields** (`Optional[List[str]]`) – optional: a sequence of strings that will be rendered using MarkdownTextObjects. Any strings included with fields will be rendered in a compact format that allows for 2 columns of side-by-side text. Maximum number of items is 10. Maximum length for the text in each item is 2000 characters.
- **accessory** (`Optional[MockObject]`) – an optional BlockElement to attach to this SectionBlock as secondary content

Return type `ModalBuilder`

submit(*submit*)

Specify the text displayed in the submit button at the bottom-right of the view.

Important Note: submit is required when an input block is within the blocks array.

Max length of 24 characters.

Parameters **submit** (`str`) – must not exceed 24 characters

Return type `ModalBuilder`

submit_length()

submit_max_length = 24

title(*title*)

Specify a title for this modal

Parameters **title** (`str`) – must not exceed 24 characters

Return type `ModalBuilder`

title_length()

title_max_length = 24

to_dict()

Return type `dict`

4.9 glados.plugin module

class glados.plugin.**GladosPlugin**(*config*, *bot*, ***kwargs*)

Bases: `object`

Parent class for a GLaDOS Plugin

Parameters

- **`config`** (`PluginConfig`) – PluginConfig object for the plugin.
- **`bot`** (`GladosBot`) – the GLaDOS bot that this plugin will use

`add_route`(*route_type*, *route*, *function*)
Add a new route to the plugin

Parameters

- **`route_type`** (`RouteType`) – what type of route this is this
- **`route`** (`Union[EventRoutes, str]`) – what is the route to be added
- **`function`** (`Callable`) – the function to be executed when this route runs

Return type `NoReturn`

`has_route`(*route*)
See if route exists.

Parameters **`route`** (*`route to check`*) –

Returns

Return type True if route exists else false

`respond_to_url`(*request*, *text*, ***kwargs*)
When you click on a link that was sent via slack it sends a callback, This is to handle that

Parameters

- **`request`** (`GladosRequest`) –
- **`text`** (`str`) –

`routes`
List all routes for the plugin.

Return type `List[GladosRoute]`

`send_request`(*request*, ***kwargs*)
This is the function to be called when sending a request to a plugin.

This function is responsible for validating the slack signature if needed. It also returns and empty string if the function called returns None.

Parameters

- **`request`** (`GladosRequest`) – the request object to be sent
- **`kwargs`** –

Return type `Any`

`class` `glados.plugin.`**`PluginBotConfig`**(*name='NOT SET'*)
Bases: `object`

`to_dict`()

`class` `glados.plugin.`**`PluginConfig`**(*name*, *config_file*, *module=None*, *enabled=False*, *bot=None*, ***kwargs*)
Bases: `object`

Plugin Config Object.

Parameters

- **name** (str) – Plugin Name
- **config_file** (str) – Path to config file for plugin
- **module** – plugin python module name
- **enabled** – enable this plugin
- **bot** – what bot does this plugin use
- **kwargs** –

to_dict (*user_config_only=True*)
Return config as dict

Parameters **user_config_only** – if True only get get waht is in the config file and not the running config.

Return type dict

to_yaml (*user_config_only=True*)

update (*config*, *use_base_module=True*)
Update a config object using the default values from the config object passed in.

Parameters

- **config** (*PluginConfig*) – the config object to use as the base. By default the module property will be set from the base config object only
- **use_base_module** (*bool*) – if set true use the value of module and package from the base config object only.

Return type NoReturn

class glados.plugin.**PluginImporter** (*plugins_folder*, *plugins_config_folder*)
Bases: object

Create the PluginImporter object.

Parameters

- **plugins_folder** (str) – plugin folder
- **plugins_config_folder** (str) – plugin config folder

discover_plugins ()
Discover all plugin config files in the plugins folder

Return type NoReturn

import_discovered_plugins (*bots*)
Import all discovered plugins and store them in self.plugins.

Parameters **bots** (Dict[str, GladosBot]) – dict of all the imported bots

Returns the results are updated in self.plugins

Return type obj: *NoReturn*:

load_discovered_plugins_config (*write_to_user_config=True*)
Load all the yaml configs for the plugins

Parameters **write_to_user_config** (bool) –

Return type NoReturn

4.10 glados.request module

class glados.request.**GladosRequest**(*route_type*, *route=None*, *slack_verify=None*, *bot_name=None*, *json=None*, *data=None*, ***kwargs*)

Bases: object

GLaDOS Request Object. This holds all the data required to process the request.

Parameters

- **route_type** (RouteType) – what type of route is this
- **route** (Optional[str]) – what is the route to be called
- **slack_verify** (Optional[SlackVerification]) – slack data used for verifying the request came from Slack
- **bot_name** (Optional[str]) – The name of the bot to send the request to. This is used for select RouteTypes
- **json** (Union[str, dict, None]) – the json paylod of the request
- **data** (Optional[dict]) – data to send with the request. This should be from a database
- **kwargs** –

Examples

```
>>> request = GladosRequest(RouteType.SendMessage, "send_mock", json={"message":
↪"my message"})
>>> print(request.json.message)
my message
>>> try:
...     print(request.json.other_param)
... except AttributeError:
...     print("ERROR")
ERROR
```

add_interaction_to_datastore(*interaction*)

Add an interaction to the datastore and return the updated interaction.

Notes

The interaction_id can be retrieved by doing interaction.interaction_id

Parameters **interaction** (DataStoreInteraction) – the interaction to be added

Return type Optional[DataStoreInteraction]

close_session()

Close session for request

Return type NoReturn

data

Returns the data object of the request

Return type PyJSON

data_blob

Returns the raw dict of the data object

Return type `dict`

`gen_new_interaction``(*,` *followup_action=None*, *followup_ts=None*, *ttl=None*, *data=None*, *auto_link=True*, *auto_set=True*`)`

Generate a new interaction object and set it as new_interaction.

Parameters

- **`followup_action`** –
- **`followup_ts`** –
- **`ttl`** –
- **`data`** –
- **`auto_link`** (`bool`) – set this request to auto-link using the return payload. The return payload must be the response from sending a slack message.
- **`auto_set`** (`bool`) – set this new interaction object as the request new_interaction

Return type `DataStoreInteraction`

`has_interaction``()`

Check if request has interaction.

Return type `bool`

`has_new_interaction``()`

check if request has a new interaction object.

Return type `bool`

`interaction`

Returns the interaction for the request

Return type `Optional[DataStoreInteraction]`

`interaction_id`

Returns the interaction_id of request.interaction

Return type `Optional[str]`

`link_interaction_to_message``(`*interaction_id*, *channel*, *message_ts*`)`

Link interaction to message

Parameters

- **`interaction_id`** (`str`) – interaction ID to link
- **`channel`** (`str`) – channel to be linked to
- **`message_ts`** (`datetime`) – ts to be linked to

Return type `NoReturn`

`link_interaction_to_message_response``(`*interaction_id*, *message_response*`)`

Link interaction to message response

Parameters

- **`interaction_id`** (`str`) – interaction ID to be linked
- **`message_response`** (`dict`) – JSON payload response from sending message on slack.

Return type `NoReturn`

rollback_session()
Rollback the session.

Return type `NoReturn`

route
the actual route

If the route automatically prefixed the route with the bot name, it will return the route with the prefix

Return type `str`

set_datastore(*datastore*)
Set the Datastore and session for the request.

Parameters **datastore** (`DataStore`) – Datastore to use. This datastore will be used to create the session.

Return type `NoReturn`

set_interaction_from_datastore()
Get the interaction object from the datastore.

Return type `NoReturn`

set_session(*session*)
Set the session for this request.

Parameters **session** (`Session`) – session to use for this request.

Raises `ConnectionError` – If the session is not active raise a ConnectionError

Return type `NoReturn`

class glados.request.**SlackVerification**(*data*, *timestamp=None*, *signature=None*)
Bases: `object`

An object to hold slack verification data

Parameters

- **data** (`str`) – raw request body. This is used to verify the message is from slack.
- **timestamp** (`Optional[str]`) – The X-Slack-Request-Timestamp from the headers of the request. This is used to verify the message is from slack.
- **signature** (`Optional[str]`) – The X-Slack-Signature from the headers of the request. This is used to verify the message is from slack.

json
Returns the dict of the SlackVerification

Return type `dict`

4.11 glados.route_type module

class glados.route_type.**EventRoutes**
Bases: `enum.Enum`

An enumeration.

app_home_opened = 1

message = 2

class glados.route_type.**RouteType**

Bases: `enum.Enum`

An enumeration.

Callback = 3

Events = 5

Interaction = 6

Menu = 7

Response = 2

Slash = 4

Webhook = 1

4.12 glados.router module

class glados.router.**GladosRoute**(*route_type*, *route*, *function*)

Bases: `object`

Represents a single route

Parameters

- **route_type** (`RouteType`) –
- **route** (`str`) –
- **function** (`Callable`) –

class glados.router.**GladosRouter**(***kwargs*)

Bases: `object`

add_route(*plugin*, *route*)

Add a route to the router

Parameters

- **plugin** – the plugin the route belongs to
- **route** (`GladosRoute`) – the route to be added

Raises `KeyError` – a route with the same type and same name already exists

Return type `NoReturn`

add_routes(*plugin*)

Add multiple routes to the router.

Parameters **plugin** – the plugin to add routes from

Return type `NoReturn`

exec_route(*request*)

Execute a route function directly

Parameters **request** (`GladosRequest`) – the GLaDOS request

Returns

Return type the data returned by the plugin

Examples

```
>>> def mock_function(request: GladosRequest):
...     print(f"Mock Function: {request.params.message}")
...     return True
>>> router = GladosRouter()
>>> route = GladosRoute(RouteType.SendMessage, "send_mock", mock_function)
>>> router.add_route(route)
>>> request = GladosRequest(RouteType.SendMessage, "send_mock", message=
↪"Hello World!")
>>> successful = router.exec_route(request)
Mock Function: Hello World!
>>> print(successful)
True
```

```
>>> def mock_function(request: GladosRequest):
...     print(f"Mock Function: {request.params.message}")
...     return True
>>> router = GladosRouter()
>>> route = GladosRoute(RouteType.SendMessage, "send_mock", mock_function)
>>> router.add_route(route)
>>> request = GladosRequest(RouteType.SendMessage, "send_mock_fail", message=
↪"Hello World!")
>>> successful = router.exec_route(request)
>>> print(successful)
False
```

get_route(*route_type*, *route*)

Get a GladosRoute object for the requested route.

Parameters

- **route_type** (`RouteType`) – the type of route to get
- **route** (`str`) – the route to get

Raises `GladosRouteNotFoundError` – the requested route is not found

Return type `Callable`

route_function(*route_type*, *route*)

Return only the callable function for the requested GladosRoute.

Parameters

- **route_type** (`RouteType`) – the type of route to get
- **route** (`str`) – the route to get

Returns return the requested routes callable function

Return type Callable

4.13 glados.utils module

class glados.utils.**PyJSON**(*d*)

Bases: `object`

from_dict(*d*)

get(*key*, *default=None*)

to_dict()

Return type `dict`

`glados.utils.`**check_for_env_vars**(*value*)

Check an input value to see if it is an env_var or enc_env_var and get the value.

Parameters **value** (`Union[str, dict]`) – input to check.

Returns Returns the value of the var from either the passed in value, or the env var value.

Return type Any

Raises KeyError if the env var is not set for what youre tying to get.

`glados.utils.`**decode_kms**(*ciphertext_blob*)

Decode a secret using the IAM role of the lambda function.

Parameters **ciphertext_blob** (`str`) – ciphertext_blob to decode

Returns Decoded KMS data

Return type obj: *str*

`glados.utils.`**get_enc_var**(*var_name*)

Get an encrypted ENV VAR

Parameters **var_name** (`str`) –

Return type `str`

`glados.utils.`**get_var**(*var_name*)

Get an ENV VAR

Parameters **var_name** (`str`) –

`glados.utils.`**read_config**(*config_file*)

Parameters **config_file** (`str`) –

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

glados, 7
glados.slack_classes, 14
glados.slack_classes.views, 14

Index

A

actions() (*glados.message_blocks.ModalBuilder method*), 21
add_block() (*glados.slack_classes.views.Home method*), 14
add_bot() (*glados.core.Glados method*), 16
add_bot() (*glados.Glados method*), 7
add_interaction_to_datastore() (*glados.GladosRequest method*), 10
add_interaction_to_datastore() (*glados.request.GladosRequest method*), 26
add_plugin() (*glados.core.Glados method*), 16
add_plugin() (*glados.Glados method*), 7
add_route() (*glados.GladosPlugin method*), 12
add_route() (*glados.plugin.GladosPlugin method*), 24
add_route() (*glados.router.GladosRouter method*), 29
add_routes() (*glados.router.GladosRouter method*), 29
app_home_opened (*glados.EventRoutes attribute*), 12
app_home_opened (*glados.route_type.EventRoutes attribute*), 28
attributes (*glados.message_blocks.ModalBuilder attribute*), 21
attributes (*glados.slack_classes.views.Home attribute*), 14

B

blocks_length() (*glados.message_blocks.ModalBuilder method*), 21
blocks_max_length (*glados.message_blocks.ModalBuilder attribute*), 21
bot (*glados.datastore.DataStoreInteraction attribute*), 19, 20
BotImporter (*class in glados.bot*), 14

C

Callback (*glados.route_type.RouteType attribute*), 29
Callback (*glados.RouteType attribute*), 12
callback_id() (*glados.message_blocks.ModalBuilder method*), 21
callback_id_max_length() (*glados.message_blocks.ModalBuilder method*), 21
check_for_env_vars() (*in module glados*), 13
check_for_env_vars() (*in module glados.utils*), 31
clear_on_close() (*glados.message_blocks.ModalBuilder method*), 21
client (*glados.bot.GladosBot attribute*), 15
client (*glados.GladosBot attribute*), 8
close() (*glados.message_blocks.ModalBuilder method*), 21
close_length() (*glados.message_blocks.ModalBuilder method*), 21
close_max_length (*glados.message_blocks.ModalBuilder attribute*), 21
close_session() (*glados.GladosRequest method*), 10
close_session() (*glados.request.GladosRequest method*), 26
context() (*glados.message_blocks.ModalBuilder method*), 22
create_session() (*glados.datastore.DataStore method*), 17
create_table() (*glados.datastore.DataStore method*), 17
cron_followup_action (*glados.datastore.DataStoreInteraction attribute*), 20

D

data (*glados.datastore.DataStoreInteraction attribute*), 19, 20
data (*glados.GladosRequest attribute*), 10

data (*glados.request.GladosRequest attribute*), 26
data_blob (*glados.GladosRequest attribute*), 10
data_blob (*glados.request.GladosRequest attribute*), 26
DataStore (*class in glados.datastore*), 17
DataStoreInteraction (*class in glados.datastore*), 19
decode_kms() (*in module glados.utils*), 31
delete_message() (*glados.bot.GladosBot method*), 15
delete_message() (*glados.GladosBot method*), 8
discover_plugins() (*glados.plugin.PluginImporter method*), 25
divider() (*glados.message_blocks.ModalBuilder method*), 22
drop_table() (*glados.datastore.DataStore method*), 17

E

EventRoutes (*class in glados*), 12
EventRoutes (*class in glados.route_type*), 28
Events (*glados.route_type.RouteType attribute*), 29
Events (*glados.RouteType attribute*), 12
exec_route() (*glados.router.GladosRouter method*), 29
external_id() (*glados.message_blocks.ModalBuilder method*), 22

F

find_by_id() (*glados.datastore.DataStore method*), 17
find_interaction_by_channel_ts() (*glados.datastore.DataStore method*), 18
followed_up (*glados.datastore.DataStoreInteraction attribute*), 20
followup_action (*glados.datastore.DataStoreInteraction attribute*), 19, 20
followup_ts (*glados.datastore.DataStoreInteraction attribute*), 19, 20
from_dict() (*glados.utils.PyJSON method*), 30

G

gen_new_interaction() (*glados.GladosRequest method*), 10
gen_new_interaction() (*glados.request.GladosRequest method*), 27
get() (*glados.utils.PyJSON method*), 30
get_enc_var() (*in module glados.utils*), 31
get_route() (*glados.router.GladosRouter method*), 30
get_var() (*in module glados.utils*), 31
Glados (*class in glados*), 7
Glados (*class in glados.core*), 16
glados (*module*), 7
glados.bot (*module*), 14
glados.configs (*module*), 15
glados.core (*module*), 16
glados.datastore (*module*), 17
glados.errors (*module*), 20
glados.message_blocks (*module*), 21
glados.plugin (*module*), 23
glados.request (*module*), 26
glados.route_type (*module*), 28
glados.router (*module*), 29
glados.slack_classes (*module*), 14
glados.slack_classes.views (*module*), 14
glados.utils (*module*), 30
GladosBot (*class in glados*), 8
GladosBot (*class in glados.bot*), 14
GladosBotNotFoundError, 20
GladosConfig (*class in glados*), 13
GladosConfig (*class in glados.configs*), 15
GladosError, 20
GladosPathExistsError, 20
GladosPlugin (*class in glados*), 12
GladosPlugin (*class in glados.plugin*), 23
GladosRequest (*class in glados*), 9
GladosRequest (*class in glados.request*), 26
GladosRoute (*class in glados.router*), 29
GladosRouteNotFoundError, 20
GladosRouter (*class in glados.router*), 29

H

has_datastore() (*glados.core.Glados method*), 16
has_datastore() (*glados.Glados method*), 7
has_interaction() (*glados.GladosRequest method*), 10
has_interaction() (*glados.request.GladosRequest method*), 27
has_new_interaction() (*glados.GladosRequest method*), 10
has_new_interaction() (*glados.request.GladosRequest method*), 27
has_route() (*glados.GladosPlugin method*), 12
has_route() (*glados.plugin.GladosPlugin method*), 24
Home (*class in glados.slack_classes.views*), 14

I

image() (*glados.message_blocks.ModalBuilder method*), 22
import_bots() (*glados.bot.BotImporter method*), 14
import_bots() (*glados.core.Glados method*), 16
import_bots() (*glados.Glados method*), 8

import_discovered_plugins() (*glados.plugin.PluginImporter method*), 25
import_plugins() (*glados.core.Glados method*), 16
import_plugins() (*glados.Glados method*), 8
insert_interaction() (*glados.datastore.DataStore method*), 18
interaction (*glados.GladosRequest attribute*), 11
interaction (*glados.request.GladosRequest attribute*), 27
Interaction (*glados.route_type.RouteType attribute*), 29
Interaction (*glados.RouteType attribute*), 12
interaction_id (*glados.datastore.DataStoreInteraction attribute*), 19, 20
interaction_id (*glados.GladosRequest attribute*), 11
interaction_id (*glados.request.GladosRequest attribute*), 27

J

json (*glados.request.SlackVerification attribute*), 28

L

link_interaction_to_message() (*glados.GladosRequest method*), 11
link_interaction_to_message() (*glados.request.GladosRequest method*), 27
link_interaction_to_message_response() (*glados.GladosRequest method*), 11
link_interaction_to_message_response() (*glados.request.GladosRequest method*), 27
link_to_message() (*glados.datastore.DataStore method*), 18
link_to_message_response() (*glados.datastore.DataStore method*), 18
load_discovered_plugins_config() (*glados.plugin.PluginImporter method*), 25

M

Menu (*glados.route_type.RouteType attribute*), 29
Menu (*glados.RouteType attribute*), 12
message (*glados.EventRoutes attribute*), 12
message (*glados.route_type.EventRoutes attribute*), 28
message_channel (*glados.datastore.DataStoreInteraction attribute*), 19, 20
message_ts (*glados.datastore.DataStoreInteraction attribute*), 19, 20
ModalBuilder (*class in glados.message_blocks*), 21

N

name (*glados.bot.GladosBot attribute*), 14
name (*glados.GladosBot attribute*), 8
notify_on_close() (*glados.message_blocks.ModalBuilder method*), 22

P

PluginBotConfig (*class in glados.plugin*), 24
PluginConfig (*class in glados.plugin*), 24
PluginImporter (*class in glados.plugin*), 25
private_metadata() (*glados.message_blocks.ModalBuilder method*), 22
private_metadata_max_length() (*glados.message_blocks.ModalBuilder method*), 22
PyJSON (*class in glados.utils*), 30

R

read_config() (*glados.configs.GladosConfig method*), 15
read_config() (*glados.core.Glados method*), 16
read_config() (*glados.Glados method*), 8
read_config() (*glados.GladosConfig method*), 13
read_config() (*in module glados*), 13
read_config() (*in module glados.configs*), 16
read_config() (*in module glados.utils*), 31
request() (*glados.core.Glados method*), 17
request() (*glados.Glados method*), 8
respond_to_url() (*glados.GladosPlugin method*), 12
respond_to_url() (*glados.plugin.GladosPlugin method*), 24
Response (*glados.route_type.RouteType attribute*), 29
Response (*glados.RouteType attribute*), 12
rollback_session() (*glados.GladosRequest method*), 11
rollback_session() (*glados.request.GladosRequest method*), 27
route (*glados.GladosRequest attribute*), 11
route (*glados.request.GladosRequest attribute*), 28
route_function() (*glados.router.GladosRouter method*), 30
routes (*glados.GladosPlugin attribute*), 13
routes (*glados.plugin.GladosPlugin attribute*), 24
RouteType (*class in glados*), 12
RouteType (*class in glados.route_type*), 28

S

section() (*glados.message_blocks.ModalBuilder method*), 22
sections (*glados.configs.GladosConfig attribute*), 15
sections (*glados.GladosConfig attribute*), 13
send_message() (*glados.bot.GladosBot method*), 15

send_message() (*glados.GladosBot method*), 9
send_request() (*glados.GladosPlugin method*), 13
send_request() (*glados.plugin.GladosPlugin method*), 24
set_datastore() (*glados.GladosRequest method*), 11
set_datastore() (*glados.request.GladosRequest method*), 28
set_interaction_from_datastore() (*glados.GladosRequest method*), 11
set_interaction_from_datastore() (*glados.request.GladosRequest method*), 28
set_session() (*glados.GladosRequest method*), 11
set_session() (*glados.request.GladosRequest method*), 28
signing_secret (*glados.bot.GladosBot attribute*), 15
signing_secret (*glados.GladosBot attribute*), 8
SlackVerification (*class in glados.request*), 28
Slash (*glados.route_type.RouteType attribute*), 29
Slash (*glados.RouteType attribute*), 12
submit() (*glados.message_blocks.ModalBuilder method*), 23
submit_length() (*glados.message_blocks.ModalBuilder method*), 23
submit_max_length (*glados.message_blocks.ModalBuilder attribute*), 23

T

table_exists() (*glados.datastore.DataStore method*), 18
title() (*glados.message_blocks.ModalBuilder method*), 23
title_length() (*glados.message_blocks.ModalBuilder method*), 23
title_max_length (*glados.message_blocks.ModalBuilder attribute*), 23
to_dict() (*glados.message_blocks.ModalBuilder method*), 23
to_dict() (*glados.plugin.PluginBotConfig method*), 24
to_dict() (*glados.plugin.PluginConfig method*), 25
to_dict() (*glados.slack_classes.views.Home method*), 14
to_dict() (*glados.utils.PyJSON method*), 31
to_yaml() (*glados.plugin.PluginConfig method*), 25
token (*glados.bot.GladosBot attribute*), 14
token (*glados.GladosBot attribute*), 8
ts (*glados.datastore.DataStoreInteraction attribute*), 19, 20
ttl (*glados.datastore.DataStoreInteraction attribute*), 19, 20

U

update() (*glados.datastore.DataStoreInteraction method*), 20
update() (*glados.plugin.PluginConfig method*), 25
update_interaction() (*glados.datastore.DataStore method*), 19
update_message() (*glados.bot.GladosBot method*), 15
update_message() (*glados.GladosBot method*), 9

V

validate_slack_signature() (*glados.bot.GladosBot method*), 15
validate_slack_signature() (*glados.GladosBot method*), 9

W

Webhook (*glados.route_type.RouteType attribute*), 29
Webhook (*glados.RouteType attribute*), 12