MyProject Documentation

Contents

404 Client Error: Not Found for url: http://172.17.0.1:3000/api/v1/metrics/tickets
404 Client Error: Not Found for url: http://172.17.0.1:3000/api/v1/metrics/ticketsp
— description: | API documentation for modules: dashboard, dashboard.main, dashboard.src, dashboard.src.Chatlog, dashboard.src.components, dashboard.src.plots, dashboard.src.settings, dashboard.src.utils, dashboard.views, dashboard.views.chatlog, dashboard.views.feedback, dashboard.views.graphs, dashboard.views.home.

lang: en

classoption: oneside geometry: margin=1in papersize: a4

linkcolor: blue links-as-notes: true …

Module `dashboard`

Sub-modules

- dashboard.main
- dashboard.src
- dashboard.views

Module `dashboard.main`

Module `dashboard.src`

Sub-modules

- dashboard.src.Chatlog
- dashboard.src.components
- dashboard.src.plots
- dashboard.src.settings
- dashboard.src.utils

Module `dashboard.src.Chatlog`

Classes

Class `Chatlog`

```
class Chatlog(
    session_id: str,
    chats: list[dict],
    session_data: dict
)
```

Chatlog Class for Managing Chat Session Data.

This class represents a chatlog containing chat messages and session data for a specific session.

Args —–= - session_id (str): The unique identifier for the chat session. - chats (list[dict]): A list of chat messages, each represented as a dictionary. - session_data (dict): A dictionary containing session-related data. Example Usage: chatlog = Chatlog(session_id="12345", chats=[…], session_data={…})

Attributes —–= - _session_id (str): The private attribute storing the session ID. - _chats (list[dict]): The private attribute storing chat messages. - _session_data (dict): The private attribute storing session-related data.

Methods —-= - get_chatlogs(session_id: str) -> List[dict]: Retrieves chat messages for the specified session. - get_session_info(session_id: str) -> dict: Retrieves session information for the specified session. - from_session_id(cls, session_id: str) -> Chatlog: Creates a Chatlog instance from a session ID.

Example Usage: chatlog = Chatlog(session_id="12345") chatlog.get_chatlogs("12345") chatlog.get_session_info("12345") chatlog = Chatlog.from_session_id("12345")

Static methods

`Method from_session_id`

```
def from_session_id(
    session_id: str
) -> dashboard.src.Chatlog.Chatlog
```

Create a Chatlog instance from a session ID.

Args —-= `session_id` : str : The unique identifier for the chat session.

Returns —-= Chatlog : An instance of the Chatlog class.

Methods

Method `get_chatlogs`

```
def get_chatlogs(
    self,
    session_id: str
) -> List[dict]
```

Retrieve chat messages for the specified session.

Args —-= `session_id` : str : The unique identifier for the chat session.

Returns —-= List[dict] : A list of chat messages, each represented as a dictionary.

Method `get_session_info`

```
def get_session_info(
    self,
    session_id: str
)
```

Retrieve session information for the specified session.

Args —-= `session_id` : str : The unique identifier for the chat session.

Returns —-= dict : A dictionary containing session-related data.

Module `dashboard.src.components`

Functions

Function `calendar`

```
def calendar()
```

Function `predefined_time`

```
def predefined_time()
```

Module `dashboard.src.plots`

Functions

Function `plot_timeseries`

```
def plot_timeseries(
    df: pandas.core.frame.DataFrame,
    metric_title: str,
    metric_name: str
)
```

Plot a time series graph for a metric.

Args —-= `df` : DataFrame : The DataFrame containing the data.

`metric_title` **: str** The title for the plot.
`metric_name` **: str** The name of the metric.

Returns —-= go.Figure : The Plotly figure object.

Function `plot_wordcloud`

```
def plot_wordcloud(
    dict: Dict,
    metric_title: str
)
```

Generate and plot a word cloud from a word frequency dictionary.

Args —–= `word_freq_dict` : dict : The word frequency dictionary.

`metric_title` **: str** The title for the plot.

Returns —–= plt.Figure : The matplotlib figure object containing the word cloud.

Module `dashboard.src.settings`

Classes

Class `APISettings`

```
class APISettings(
    **values: Any
)
```

API Settings Configuration.

This class defines the API settings configuration, including API URI, API token, resources path, and model configuration.

Attributes —–= - API_URI (str): The base URI for the API endpoint. - API_TOKEN (str): The API authentication token. - RESOURCES_PATH (str): The path to the resources directory. - model_config (SettingsConfigDict): Configuration for loading environment variables from a .env file. Example Usage: api_settings = APISettings() print(api_settings.API_URI)

Create a new model by parsing and validating input data from keyword arguments.

Raises [`ValidationError`][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses __pydantic_self__ instead of the more common self for the first arg to allow self as a field name.

Ancestors (in MRO)

- pydantic_settings.main.BaseSettings
- pydantic.main.BaseModel

Class variables

Variable `API_TOKEN` Type: `str`

Variable `API_URI`   Type: `str`

Variable `RESOURCES_PATH`   Type: `str`

Variable `model_config`   Type: `ClassVar[pydantic_settings.main.SettingsConfigDict]`

Variable `model_fields`

Class `DefaultMetrics`

```
class DefaultMetrics(
    **values: Any
)
```

Default Metrics Configuration.

This class defines the default metrics configuration, including a list of metric types that can be used in the application.

Attributes —–= - metric_list (list[str]): A list of metric types including "messages," "sessions," "tickets," "ticketsp," "avg_messages," "keywords," and "topics." Example Usage: metrics = DefaultMetrics() print(metrics.metric_list)

Create a new model by parsing and validating input data from keyword arguments.

Raises [`ValidationError`][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses __pydantic_self__ instead of the more common self for the first arg to allow self as a field name.

Ancestors (in MRO)

- pydantic_settings.main.BaseSettings
- pydantic.main.BaseModel

Class variables

Variable `metric_list`   Type: `list[str]`

Variable `model_config`   Type: `ClassVar[pydantic_settings.main.SettingsConfigDict]`

Variable `model_fields`

Class `Settings`

```
class Settings(
    **values: Any
)
```

Application Settings Configuration.

This class defines the application settings configuration, including API settings and default metrics.

Attributes —–= - api (APISettings): An instance of APISettings containing API-related configuration. - metrics (DefaultMetrics): An instance of DefaultMetrics containing default metrics configuration. Example Usage: settings = Settings() print(settings.api.API_URI)

Create a new model by parsing and validating input data from keyword arguments.

Raises [`ValidationError`][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses __pydantic_self__ instead of the more common self for the first arg to allow self as a field name.

Ancestors (in MRO)

- pydantic_settings.main.BaseSettings
- pydantic.main.BaseModel

Class variables

Variable `api` Type: `dashboard.src.settings.APISettings`

Variable `metrics` Type: `dashboard.src.settings.DefaultMetrics`

Variable `model_config` Type: `ClassVar[pydantic_settings.main.SettingsConfigDict]`

Variable `model_fields`

Module `dashboard.src.utils`

Functions

Function `get_all_data`

```
def get_all_data() -> dict[str, pandas.core.frame.DataFrame | dict]
```

Get all available data for configured metrics.

Returns —-= dict[str, Union[pd.DataFrame, dict]] : Processed data for each metric.

Function `get_data`

```
def get_data(
    metric: str
) -> Any
```

Get metric data from the API.

Args —-= `metric` : str : The name of the metric to retrieve.

Returns —-= Any : The data for the specified metric.

Function `get_data_by_date`

```
def get_data_by_date(
    data: dict,
    start_date,
    end_date
)
```

Filter data within a specified date range for all available metrics.

Args —-= `data` : dict : Processed data for all metrics.

`start_date` **: datetime** The start date for the date range.
`end_date` **: datetime** The end date for the date range.

Returns —-= dict : Filtered data for each metric.

Function `get_data_by_sampling`

```
def get_data_by_sampling(
    data: dict,
    sampling: str
)
```

Resample data to a specified sampling rate for all available metrics.

Args —-= `data` : dict : Processed data for all metrics.

`sampling` **: str** The desired sampling rate.

Returns —-= Union[None, dict] : Resampled data for each metric, or None if the sampling rate is invalid.

Function `get_feedbacks`

```
def get_feedbacks(
    start_date,
    end_date
) -> pandas.core.frame.DataFrame
```

Get feedback data within a specified date range.

Args —-= `start_date` : datetime : The start date for the date range.

`end_date` **: datetime** The end date for the date range.

Returns —-= Union[pd.DataFrame, None] : DataFrame containing feedback data or None if an error occurs.

Function `get_sessions`

```
def get_sessions(
    start_date,
    end_date
) -> List[str]
```

Get a list of session IDs within a specified date range.

Args —-= `start_date` : datetime : The start date for the date range.

`end_date` **: datetime** The end date for the date range.

Returns —-= List[str] : A list of session IDs within the specified date range.

Function `save_csv_data`

```
def save_csv_data(
    metric: str
)
```

Save data for a metric to a CSV file.

Args —-= `metric` : str : The name of the metric.

Returns —–= bool : True if data was successfully saved, False otherwise.

Module `dashboard.views`

Sub-modules

- dashboard.views.chatlog
- dashboard.views.feedback
- dashboard.views.graphs
- dashboard.views.home

Module `dashboard.views.chatlog`

Functions

Function `ChatlogView`

```
def ChatlogView(
    data: list
)
```

Module `dashboard.views.feedback`

Functions

Function `FeedbackView`

```
def FeedbackView(
    data
)
```

Module `dashboard.views.graphs`

Functions

Function `GraphsView`

```
def GraphsView(
    data
)
```

Module `dashboard.views.home`

Functions

Function `HomeView`

```
def HomeView(
    data,
    start_date,
    end_date,
    sammpling_freq
)
```

HomeView is a Streamlit application for displaying data, analytics, and providing data download options.

Args —–= `data` : pandas.DataFrame : The input data to be displayed and analyzed.

`start_date` **: str** The start date for filtering the data.
`end_date` **: str** The end date for filtering the data.
`sammpling_freq` **: str** The sampling frequency for data.

This function creates a Streamlit application to display data based on specified date ranges and sampling frequency. It allows users to select a metric to display, download data as a CSV file, and provides summary analytics such as total messages, total sessions, total tickets, average tickets per session, and average messages per session.

Parameters —–= - data (pandas.DataFrame): The input data to be displayed and analyzed. - start_date (str): The start date for filtering the data. - end_date (str): The end date for filtering the data. - sammpling_freq (str): The sampling frequency for data.

Returns —–= pandas.DataFrame : The filtered and sampled data.
