bundesliga-tippspiel

Release 1.7.1

Hermann Krumrey

Sep 19, 2020

CONTENTS

Contents:

CHAPTER

ONE

BUNDESLIGA_TIPPSPIEL PACKAGE

1.1 Subpackages

1.1.1 bundesliga_tippspiel.actions package

Submodules

bundesliga_tippspiel.actions.Action module

class bundesliga_tippspiel.actions.Action.**Action**
Bases: object

A framework class for actions that can be used by normal site requests as well as API calls to achieve stuff™.

static check_id_or_filters(*_id: Optional[int]*, *filters: List[Optional[Any]]*)
Checks that no filters are applied if a specific ID was provided :param _id: The specific ID :param filters: A list of filters :return: None

execute() → Dict[str, Any]
Executes the action after validating user-provided data :return: A JSON-compatible dictionary containing the response :raises ActionException: if anything went wrong

execute_with_redirects(*success_url: str*, *success_msg: str*, *failure_url: str*) → werkzeug.wrappers.response.Response
Executes the action and subsequently redirects accordingly :param success_url: The URL to which to redirect upon success :param success_msg: The message to flash on success :param failure_url: The URl to which to redirect upon failure :return: The redirect

classmethod from_dict(*data: Dict[str, Any]*)
Generates an action from a dictionary :param data: The dictionary containing the relevant data :return: The generated Action object

classmethod from_site_request()
Generates an Action object from a site request :return: The generated Action object

static get_current_matchday() → int

Returns The current matchday

static handle_id_fetch(*_id: int*, *query_cls: Type[sqlalchemy.ext.declarative.api.Model]*) → sqlalchemy.ext.declarative.api.Model
Handles fetching a single object by using it's ID Raises an ActionException if an ID does not exist :return: The object identified by the ID :raises ActionException: Without fail

prepare_get_response(*result: List[puffotter.flask.db.ModelMixin.ModelMixin]*, *keyword: str*) → Dict[str, Any]
Prepares a GetAction response by :param result: The result to wrap in a response dictionary :param keyword: The keyword to use, e.g: bet|match|player etc. :return: The wrapped response dictionary

static resolve_and_check_matchday(*matchday: Optional[int]*) → Optional[int]
Checks the bound of a matchday :param matchday: The matchday to check :return: None :raises ActionException: If the matchday is invalid

validate_data()
Validates user-provided data :return: None :raises ActionException: if any data discrepancies are found

class bundesliga_tippspiel.actions.Action.**GetAction**(*_id: Optional[int]*)
Bases: *bundesliga_tippspiel.actions.Action.Action*

Special Action class for 'Get' Actions

__init__(*_id: Optional[int]*)

Parameters **_id** – The ID to get

bundesliga_tippspiel.actions.ChangeSettingsAction module

class bundesliga_tippspiel.actions.ChangeSettingsAction.**ChangeSettingsAction**(*display_bots: bool*)
Bases: *bundesliga_tippspiel.actions.Action.Action*

Action that allows users to change miscellaneous settings

__init__(*display_bots: bool*)
Initializes the SetReminderSettingsAction object :param display_bots: Whether to display bots

validate_data()
Validates user-provided data :return: None :raises ActionException: if any data discrepancies are found

bundesliga_tippspiel.actions.GetBetAction module

class bundesliga_tippspiel.actions.GetBetAction.**GetBetAction**(*_id: Optional[int] = None*, *user_id: Optional[int] = None*, *match_id: Optional[int] = None*, *matchday: Optional[int] = None*, *include_other_users_bets: bool = False*)
Bases: *bundesliga_tippspiel.actions.Action.GetAction*

Action that allows retrieving bets from the database

__init__(*_id: Optional[int] = None*, *user_id: Optional[int] = None*, *match_id: Optional[int] = None*, *matchday: Optional[int] = None*, *include_other_users_bets: bool = False*)
Initializes the GetBetAction object :param _id: If provided, returns the bet with the specified ID :param user_id: If provided, will only provide bets for the

specified user

Parameters

- **match_id** – If provided, will only provide bets for the specified match
- **matchday** – If provided, will only provide bets for the specified matchday
- **include_other_users_bets** – If True, will include bets from other users even if the match hasn't started yet.

Raises ActionException if any problems occur

validate_data()
Validates user-provided data :return: None :raises ActionException: if any data discrepancies are found

bundesliga_tippspiel.actions.GetGoalAction module

class bundesliga_tippspiel.actions.GetGoalAction.**GetGoalAction**(*_id: Optional[int] = None*, *matchday: Optional[int] = None*, *match_id: Optional[int] = None*, *player_id: Optional[int] = None*, *team_id: Optional[int] = None*)

Bases: *bundesliga_tippspiel.actions.Action.GetAction*

Action that allows retrieving goals from the database

__init__(*_id: Optional[int] = None*, *matchday: Optional[int] = None*, *match_id: Optional[int] = None*, *player_id: Optional[int] = None*, *team_id: Optional[int] = None*)
Initializes the GetGoalAction object :param _id: If provided, returns the goal with the specified ID :param matchday: If provided, will only fetch goals

on the specified matchday

Parameters

- **match_id** – If provided, will only fetch goals that occured during the specified match
- **player_id** – If provided, will only fetch goals from the specified player
- **team_id** – If provided, will only fetch goals from the specified team

Raises ActionException if any problems occur

validate_data()
Validates user-provided data :return: None :raises ActionException: if any data discrepancies are found

bundesliga_tippspiel.actions.GetMatchAction module

class bundesliga_tippspiel.actions.GetMatchAction.**GetMatchAction**(*_id: Optional[int] = None*, *matchday: Optional[int] = None*, *team_id: Optional[int] = None*)

Bases: *bundesliga_tippspiel.actions.Action.GetAction*

Action that enables getting Matches

__init__(*_id: Optional[int] = None*, *matchday: Optional[int] = None*, *team_id: Optional[int] = None*)

Initializes the GetMatchAction object :param _id: If provided, returns the match with that ID :param matchday: If provided, will return all matches on that matchday :param team_id: If provided, will return all matches of a team :raises: ActionException if any problems occur

validate_data()

Validates user-provided data :return: None :raises ActionException: if any data discrepancies are found

bundesliga_tippspiel.actions.GetPlayerAction module

class bundesliga_tippspiel.actions.GetPlayerAction.**GetPlayerAction**(*_id: Optional[int] = None*, *team_id: Optional[int] = None*)

Bases: *bundesliga_tippspiel.actions.Action.GetAction*

Action that allows retrieving players from the database

__init__(*_id: Optional[int] = None*, *team_id: Optional[int] = None*)

Initializes the GetPlayerAction object :param _id: If provided, will only fetch the selected ID :param team_id: If provided, will only fetch players

in the selected team

Raises ActionException if any problems occur

validate_data()

Validates user-provided data :return: None :raises ActionException: if any data discrepancies are found

bundesliga_tippspiel.actions.GetReminderSettingsAction module

class bundesliga_tippspiel.actions.GetReminderSettingsAction.**GetReminderSettingsAction**
Bases: *bundesliga_tippspiel.actions.Action.Action*

Action that allows retrieving the user's reminder settings

validate_data()
Validates user-provided data :return: None :raises ActionException: if any data discrepancies are found

bundesliga_tippspiel.actions.GetTeamAction module

class bundesliga_tippspiel.actions.GetTeamAction.**GetTeamAction**(*_id: Optional[int] = None*)
Bases: *bundesliga_tippspiel.actions.Action.GetAction*

Action that allows retrieving teams from the database

__init__(*_id: Optional[int] = None*)
Initializes the GetTeamAction object :param _id: If provided, will only fetch the selected ID :raises: ActionException if any problems occur

validate_data()
Validates user-provided data :return: None :raises ActionException: if any data discrepancies are found

bundesliga_tippspiel.actions.LeaderboardAction module

class bundesliga_tippspiel.actions.LeaderboardAction.**LeaderboardAction**(*matchday: Optional[int] = None*, *count: bool = False*, *include_bots: bool = False*, *bets: Optional[List[*bundesliga_tippsp
= None)
Bases: *bundesliga_tippspiel.actions.Action.Action*

Action that allows fetching a sorted leaderboard

__init__(*matchday: Optional[int] = None*, *count: bool = False*, *include_bots: bool = False*, *bets: Optional[List[*bundesliga_tippspiel.db.user_generated.Bet.Bet*]] = None*)
Initializes the LeaderboardAction object :param matchday: The matchday for which to generate the leaderboard.

If None, will use the most current matchday

Parameters

- **count** – If set to true, will count the amount of bets instead of evaluating their points
- **include_bots** – Whether or not to include bots. Bots are identified by a robot or brain emoji
- **bets** – Allows using a specific set of bets to calculate the leaderboard

validate_data()
Validates user-provided data :return: None :raises ActionException: if any data discrepancies are found

bundesliga_tippspiel.actions.LoadSettingsAction module

class bundesliga_tippspiel.actions.LoadSettingsAction.**LoadSettingsAction**
Bases: *bundesliga_tippspiel.actions.Action.Action*

Action that retrieves miscellaneous settings for the current user

__init__()
Initializes the LoadSettingsAction object :raises: ActionException if any problems occur

validate_data()
Validates data :return: None :raises ActionException: if any data discrepancies are found

bundesliga_tippspiel.actions.PlaceBetsAction module

class bundesliga_tippspiel.actions.PlaceBetsAction.**PlaceBetsAction**(*bets: Dict[Union[str, int], Tuple[Optional[Union[str, int]], Optional[Union[str, int]]]]*)
Bases: *bundesliga_tippspiel.actions.Action.Action*

Action that allows placing bets

__init__(*bets: Dict[Union[str, int], Tuple[Optional[Union[str, int]], Optional[Union[str, int]]]]*)
Initializes the PlaceBetsAction object :param bets: a dictionary mapping match IDs to tuples containing the

home and away scores

Raises ActionException if any problems occur

validate_data()
Validates user-provided data :return: None :raises ActionException: if any data discrepancies are found

bundesliga_tippspiel.actions.SetReminderSettingsAction module

class bundesliga_tippspiel.actions.SetReminderSettingsAction.**SetReminderSettingsAction**(*hours: int*, *reminder_states: Dict[bool]*)

Bases: *bundesliga_tippspiel.actions.Action.Action*

Action that allows setting reminder settings

__init__(*hours: int*, *reminder_states: Dict[*bundesliga_tippspiel.enums.ReminderType*, bool]*)
Initializes the SetReminderSettingsAction object :param hours: How many hours before the match the user wants a reminder :param reminder_states: specifies the states for each of the reminder

types

validate_data()
Validates user-provided data :return: None :raises ActionException: if any data discrepancies are found

Module contents

1.1.2 bundesliga_tippspiel.background package

Submodules

bundesliga_tippspiel.background.match_data module

bundesliga_tippspiel.background.match_data.**get_team_data**(*team_name: str*) → Tuple[str, str, str, Tuple[str, str]]
Generates team short_names, abbreviations and icon URLs for teams :param team_name: The team's full name as specified by OpenLigaDB :return: A tuple containing the

name, short_name, abbreviation, svg URL, png URL

bundesliga_tippspiel.background.match_data.**parse_goal**(*goal_data: Dict[str, Any]*, *match_id: int*) → Optional[*bundesliga_tippspiel.db.match_data.Goal.Goal*]
Parses a goal JSON object and generates a Goal object :param match_id: The match ID of the match in which the goal was scored :param goal_data: The goal data to parse :return: The generated Goal object

bundesliga_tippspiel.background.match_data.**parse_match**(*match_data: Dict[str, Any]*) → *bundesliga_tippspiel.db.match_data.Match.Match*
Parses a Match object from JSON match data :param match_data: The match data to parse :return: The generated Match object

bundesliga_tippspiel.background.match_data.**parse_player**(*goal_data: Dict[str, Any]*, *team_id: int*) → *bundesliga_tippspiel.db.match_data.Player.Player*
Parses a Player object from a Goal JSON data object :param goal_data: The data of a goal the player scored :param team_id: The Team of the player :return: The generated Player object

bundesliga_tippspiel.background.match_data.**parse_team**(*team_data: Dict[str, Any]*) → *bundesliga_tippspiel.db.match_data.Team.Team*

Parses team-related JSON data and generates a Team object from that :param team_data: The team data to parse :return: The generated Team object

bundesliga_tippspiel.background.match_data.**store_in_db**(*objects: List[sqlalchemy.ext.declarative.api.Model]*, *model_cls: Type[sqlalchemy.ext.declarative.api.Model]*)

Stores a list of objects in the database. While doing so, will either update existing objects or create new one if they don't exist :param objects: The objects to add :param model_cls: The model class of these objects :return: None

bundesliga_tippspiel.background.match_data.**update_match_data**(*league: Optional[str] = None*, *season: Optional[str] = None*)

Updates the database with the match data for the specified league and season :param league: The league for which to update the data :param season: The season for which to update the data :return: None

bundesliga_tippspiel.background.match_data.**wikimedia_icon_urls**(*path: str*, *png_size: int = 500*) → Tuple[str, str]

Generates URL paths to wikimedia-hosted SVG and PNG files :param path: The URL path to the SVG file (without the wikimedia part) :param png_size: The size of the PNG file :return: The URL path to the SVG File, PNG file

bundesliga_tippspiel.background.reminders module

bundesliga_tippspiel.background.reminders.**send_due_reminders**()

Sends all email reminders that are due :return: None

bundesliga_tippspiel.background.season_events module

bundesliga_tippspiel.background.season_events.**handle_midseason_reminder**() → bool

Handles sending out midseason reminders :return: Whether the reminder was sent or not

bundesliga_tippspiel.background.season_events.**handle_postseason_wrapup**() → bool

Handles the post-season wrapup :return: None

bundesliga_tippspiel.background.season_events.**handle_preseason_reminder**() → bool

Sends a reminder to existing users a week before the start of the season :return: Whether or not the reminder was sent

bundesliga_tippspiel.background.season_events.**handle_season_events**()

Handles any events that happen once every season :return: None

bundesliga_tippspiel.background.season_events.**load_season_events**() → List[*bundesliga_tippspiel.db.SeasonE*

Loads all event states from the database :return: The event states

Module contents

bundesliga_tippspiel.background.**bg_tasks: Dict[str, Tuple[int, Callable]] = {'handle_seas**
A dictionary containing background tasks for the flask application

1.1.3 bundesliga_tippspiel.db package

Subpackages

bundesliga_tippspiel.db.match_data package

Submodules

bundesliga_tippspiel.db.match_data.Goal module

class bundesliga_tippspiel.db.match_data.Goal.**Goal**(**args*, ***kwargs*)
Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Model that describes the "goals" SQL table

__init__(**args*, ***kwargs*)
Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

away_score: int
The away team's score after the goal was scored

home_score: int
The home team's score after the goal was scored

id

match: *bundesliga_tippspiel.db.match_data.Match.Match*
The match in which this goal was scored.

match_id: int
The ID of the match in which this goal was scored. Acts as a foreign key.

minute: int
The minute in which the goal was scored

minute_et: int
This keeps track in which minute of extra time a goal was scored.

own_goal: bool
Indicates whether or not this goal was an own goal

penalty: bool
Indicates whether or not this goal was a penalty

player: *bundesliga_tippspiel.db.match_data.Player.Player*
The player that scored this goal.

player_id: int
The ID of the player that scored this goal. Acts as a foreign key.

bundesliga_tippspiel.db.match_data.Match module

class bundesliga_tippspiel.db.match_data.Match.**Match**(**args*, ***kwargs*)

Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Model that describes the 'matches' SQL table

__init__(**args*, ***kwargs*)

Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

away_current_score: int

The current score of the away team.

away_ft_score: int

The final score of the away team

away_ht_score: int

The score of the away team at half time

away_team: *bundesliga_tippspiel.db.match_data.Team.Team*

The away team.

away_team_id: int

The ID of the away team. Acts as a foreign key

bets: List[Bet]

Bets placed on this match

property current_score

Returns The current score formatted as a string

finished: bool

Indicates whether or not the match has finished yet

property ft_score

Returns The full time score formatted as a string

goals: List[Goal]

The goals scored during this match

home_current_score: int

The current score of the home team.

home_ft_score: int

The final score of the home team

home_ht_score: int

The score of the home team at half time

home_team: *bundesliga_tippspiel.db.match_data.Team.Team*

The home team.

home_team_id: int

The ID of the home team. Acts as a foreign key

property ht_score

Returns The half time score formatted as a string

id

kickoff: str
A string representing the kickoff time in UTC in the following format: YYYY-MM-DD:HH-mm-ss If the kickoff time is not known, it should be set to 'TBD'

property kickoff_datetime

Returns A datetime object representing the kickoff time

matchday: int
The match day of the match

property minute_display
This generates a string for displaying the current match minute. Sadly, since OpenligaDB does not provide information on the current minute, this can only offer an approximation. :return: A formatted string displaying the current match minute

season: int
The season of the match

started: bool
Indicates whether or not the match has started yet

bundesliga_tippspiel.db.match_data.Player module

class bundesliga_tippspiel.db.match_data.Player.**Player**(**args*, ***kwargs*)
Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Model that describes the "players" SQL table

__init__(**args*, ***kwargs*)
Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

goals: List[Goal]
The goals the player scored.

id

name: str
The name of the player

team: *bundesliga_tippspiel.db.match_data.Team.Team*
The team the player is affiliated with.

team_id: int
The ID of the team the player is affiliated with. Acts as a foreign key to the 'teams' table.

bundesliga_tippspiel.db.match_data.Team module

class bundesliga_tippspiel.db.match_data.Team.**Team**(**args*, ***kwargs*)
Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Model that describes the 'teams' SQL table A Team is the most basic data for a match, it relies on no other data, only primitives

__init__(**args*, ***kwargs*)
Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

abbreviation: str
A three-letter abbreviation of the team's name. Has to be unique. Example: FCB

icon_png: str
The URL of an image file representing the team's logo in PNG format

icon_svg: str
The URL of an image file representing the team's logo in SVG format

id

name: str
The full name of the team. Has to be unique. Example: FC Bayern München

players: List[Player]
The players of this team

season_position_bets: List[SeasonPositionBet]
The players of this team

season_team_bets: List[SeasonTeamBet]
The players of this team

short_name: str
The shortened version of the team's name. Has to be unique. Example: Bayern

Module contents

bundesliga_tippspiel.db.settings package

Submodules

bundesliga_tippspiel.db.settings.DisplayBotsSettings module

class bundesliga_tippspiel.db.settings.DisplayBotsSettings.**DisplayBotsSettings**(**args*, ***kwargs*)

Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Database model that specifies whether a user wants to see bots or not

__init__(**args*, ***kwargs*)
Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

display_bots
Whether or not to show bots

id

user: puffotter.flask.db.User.User
The user associated with this setting

user_id: int
The ID of the user associated with this setting

bundesliga_tippspiel.db.settings.ReminderSettings module

class bundesliga_tippspiel.db.settings.ReminderSettings.**ReminderSettings**(**args*, ***kwargs*)

Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Database model that keeps track of reminder settings

__init__(**args*, ***kwargs*)
Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

active
Whether the reminder is active or not

get_due_matches() → List[*bundesliga_tippspiel.db.match_data.Match.Match*]
Checks if the reminder is due and returns a list of matches that the user still needs to bet on. :return: The matches for which the reminder is due

id

last_reminder: str
The time when the last reminder was sent. Format in the form %Y-%m-%d:%H-%M-%S

property last_reminder_datetime

Returns The 'last_reminder' parameter as a datetime object

reminder_time: int
The time before the next unbet match when the reminder message will be sent. Unit: seconds

property reminder_time_delta

Returns The 'reminder_time' parameter as a datetime timedelta

reminder_type
The type of reminder

send_reminder()
Sends a reminder message if it's due :return: None

send_reminder_message(*message: str*)
Sends a reminder message using the appropriate method of delivery :param message: The message to send :return: None

set_reminder_time(*reminder_time: int*)
Sets the reminder time and resets the time stored as the last reminder :param reminder_time: the new reminder time :return: None

user: puffotter.flask.db.User.User
The user associated with this setting

user_id: int
The ID of the user associated with this setting

Module contents

bundesliga_tippspiel.db.user_generated package

Submodules

bundesliga_tippspiel.db.user_generated.Bet module

class bundesliga_tippspiel.db.user_generated.Bet.**Bet**(**args*, ***kwargs*)

Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Model that describes the 'bets' SQL table

__init__(**args*, ***kwargs*)

Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

away_score: int

The score bet on the away team

evaluate(*when_finished: bool = False*) → int

Evaluates the points score on this bet :param when_finished: Only calculate the value

when the match is finished. Otherwise, returns 0

Returns The calculated points

home_score: int

The score bet on the home team

id

match: *bundesliga_tippspiel.db.match_data.Match.Match*

The match that this bet refers to

match_id: int

The ID of the match that this bet refers to.

user: puffotter.flask.db.User.User

The user associated with this bet

user_id: int

The ID of the user associated with this bet

bundesliga_tippspiel.db.user_generated.ChatMessage module

class bundesliga_tippspiel.db.user_generated.ChatMessage.**ChatMessage**(**args*, ***kwargs*)

Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Model that describes the 'chat_messages' SQL table

__init__(**args*, ***kwargs*)

Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

children: List[*bundesliga_tippspiel.db.user_generated.ChatMessage.ChatMessage*]
Any child chat messages

creation_time: float
The timestamp of when the message was created

delete()
Marks the message as deleted :return: None

deleted: bool
Whether the message has been deleted

edit(*new_text: str*)
Edits the message :param new_text: The new message text :return: None

edited: bool
Whether the message has been edited

get_text() → Optional[str]

Returns The text of the chat message if the user still exists and the message has not been deleted

id: int
The ID of the chat message

last_edit: float
The timestamp of when the message was last edited

parent: *bundesliga_tippspiel.db.user_generated.ChatMessage.ChatMessage*
The parent chat message

parent_id: int
ID of the parent chat message

text: str
The text of the message

user: Optional[puffotter.flask.db.User.User]
The user associated with this bet

user_id: int
The ID of the user associated with this bet

bundesliga_tippspiel.db.user_generated.SeasonPositionBet module

class bundesliga_tippspiel.db.user_generated.SeasonPositionBet.**SeasonPositionBet**(**args*, ***kwargs*)

Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Model that describes the 'season_position_bets' SQL table

__init__(**args*, ***kwargs*)
Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

id

position: int
The position of the team in the table

season: int
The season of the season bet

team: *bundesliga_tippspiel.db.match_data.Team.Team*
The team the position bet is for.

team_id: **int**
The ID of the team the position bet is for.

user: **puffotter.flask.db.User.User**
The user associated with this season position bet

user_id: **int**
The ID of the user associated with this season position bet

bundesliga_tippspiel.db.user_generated.SeasonTeamBet module

class bundesliga_tippspiel.db.user_generated.SeasonTeamBet.**SeasonTeamBet**(**args*, ***kwargs*)
Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Model that describes the 'season_team_bets' SQL table

__init__(**args*, ***kwargs*)
Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

bet_type: *bundesliga_tippspiel.db.user_generated.SeasonTeamBet.SeasonTeamBetType*
The type of the bet

id

season: **int**
The season of the season bet

team: *bundesliga_tippspiel.db.match_data.Team.Team*
The team the user bet on.

team_id: **int**
The ID of the team the user bet on.

user: **puffotter.flask.db.User.User**
The user associated with this bet

user_id: **int**
The ID of the user associated with this bet

class bundesliga_tippspiel.db.user_generated.SeasonTeamBet.**SeasonTeamBetType**(*value*)
Bases: enum.Enum

Class that specifies the various season team bets that are possible

LEAST_GOALS_CONCEDED = 'Die wenigsten Tore'

LEAST_GOALS_SCORED = 'Die wenigsten Gegentore'

MOST_GOALS_CONCEDED = 'Die meisten Gegentore'

MOST_GOALS_SCORED = 'Die meisten Tore'

MOST_OWN_GOALS = 'Die meisten Eigentore'

bundesliga_tippspiel.db.user_generated.SeasonWinner module

class bundesliga_tippspiel.db.user_generated.SeasonWinner.**SeasonWinner**(**args*, ***kwargs*)

Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Model that describes the 'season_winners' SQL table

__init__(**args*, ***kwargs*)

Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

id

season: int

The season for which this is the winner

property season_string

Returns The season string, e.g. 2019/20

user: puffotter.flask.db.User.User

The user that won the competition

user_id: int

The ID of the user that won the competition

Module contents

Submodules

bundesliga_tippspiel.db.SeasonEvent module

class bundesliga_tippspiel.db.SeasonEvent.**SeasonEvent**(**args*, ***kwargs*)

Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Model that describes the 'season_events' SQL table

__init__(**args*, ***kwargs*)

Initializes the Model :param args: The constructor arguments :param kwargs: The constructor keyword arguments

event_type: *bundesliga_tippspiel.db.SeasonEvent.SeasonEventType*

The type of event

executed: bool

Whether the event was executed or not

id

season: int

The season this event is for

class bundesliga_tippspiel.db.SeasonEvent.**SeasonEventType**(*value*)

Bases: enum.Enum

Enumeration that describes all the possible season event types

MID_SEASON_REMINDER = 'mid_season_reminder'

POST_SEASON_WRAPUP = 'post_season_wrapup'

PRE_SEASON_MAIL = 'pre_season_mail'

Module contents

bundesliga_tippspiel.db.**models: List[sqlalchemy.ext.declarative.api.Model] = [<class 'bund**
The database models of the application

1.1.4 bundesliga_tippspiel.routes package

Subpackages

bundesliga_tippspiel.routes.api package

Submodules

bundesliga_tippspiel.routes.api.getters module

bundesliga_tippspiel.routes.api.getters.**define_blueprint**(*blueprint_name: str*) → flask.blueprints.Blueprint
Defines the blueprint for this route :param blueprint_name: The name of the blueprint :return: The blueprint

bundesliga_tippspiel.routes.api.getters.**execute_getter**(*_id: Optional[int]*, *action_cls: Type[*bundesliga_tippspiel.actions.Action.GetAction*]*) → Dict[str, Any]
Executes a getter API method using a Getter Action and an optional ID :param _id: Optional ID to use while fetching :param action_cls: The action class to use for fetching :return: The result

bundesliga_tippspiel.routes.api.putters module

bundesliga_tippspiel.routes.api.putters.**define_blueprint**(*blueprint_name: str*) → flask.blueprints.Blueprint
Defines the blueprint for this route :param blueprint_name: The name of the blueprint :return: The blueprint

Module contents

Submodules

bundesliga_tippspiel.routes.betting module

bundesliga_tippspiel.routes.betting.**define_blueprint**(*blueprint_name: str*) → flask.blueprints.Blueprint
Defines the blueprint for this route :param blueprint_name: The name of the blueprint :return: The blueprint

bundesliga_tippspiel.routes.chat module

bundesliga_tippspiel.routes.chat.**define_blueprint**(*blueprint_name: str*) → flask.blueprints.Blueprint

Defines the blueprint for this route :param blueprint_name: The name of the blueprint :return: The blueprint

bundesliga_tippspiel.routes.information module

bundesliga_tippspiel.routes.information.**define_blueprint**(*blueprint_name: str*) → flask.blueprints.Blueprint

Defines the blueprint for this route :param blueprint_name: The name of the blueprint :return: The blueprint

bundesliga_tippspiel.routes.reminder module

bundesliga_tippspiel.routes.reminder.**define_blueprint**(*blueprint_name: str*) → flask.blueprints.Blueprint

Defines the blueprint for this route :param blueprint_name: The name of the blueprint :return: The blueprint

bundesliga_tippspiel.routes.settings module

bundesliga_tippspiel.routes.settings.**define_blueprint**(*blueprint_name: str*) → flask.blueprints.Blueprint

Defines the blueprint for this route :param blueprint_name: The name of the blueprint :return: The blueprint

Module contents

bundesliga_tippspiel.routes.**blueprint_generators: List[Tuple[Callable[str, flask.blueprint**

Defines the functions used to create the various blueprints as well as their names

1.1.5 bundesliga_tippspiel.test package

Subpackages

bundesliga_tippspiel.test.actions package

Submodules

bundesliga_tippspiel.test.actions.ActionTestFramework module

bundesliga_tippspiel.test.actions.GetActionTestFramework module

bundesliga_tippspiel.test.actions.TestAction module

class bundesliga_tippspiel.test.actions.TestAction.**ActionTest**(*methodName='runTest'*)

Bases: bundesliga_tippspiel.test.TestFramework._TestFramework

Class that tests the static helper functions of the Action class

test_getting_current_matchday()

Tests getting the current matchday :return: None

bundesliga_tippspiel.test.actions.TestGetBetAction module

class bundesliga_tippspiel.test.actions.TestGetBetAction.**TestGetBetAction**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.actions.GetActionTestFramework._GetActionTestFramework

Class that tests the GetBet action

property action_cls

Returns The tested Action class

test_fetching()
Tests fetching using different methods :return: None

test_fetching_by_match()
Explicitly search by match ID, since there was a bug once :return: None

test_using_filter_and_id()
Tests that using an ID and an explicit filter does not work :return: None

bundesliga_tippspiel.test.actions.TestGetGoalAction module

class bundesliga_tippspiel.test.actions.TestGetGoalAction.**TestGetGoalAction**(*methodName='runTe*
Bases: bundesliga_tippspiel.test.actions.GetActionTestFramework._GetActionTestFramework

Class that tests the GetGoal action

property action_cls

Returns The tested Action class

test_fetching()
Tests fetching using different methods :return: None

test_using_filter_and_id()
Tests that using an ID and an explicit filter does not work :return: None

bundesliga_tippspiel.test.actions.TestGetMatchAction module

class bundesliga_tippspiel.test.actions.TestGetMatchAction.**TestGetMatchAction**(*methodName='ru*
Bases: bundesliga_tippspiel.test.actions.GetActionTestFramework._GetActionTestFramework

Class that tests the GetMatch action

property action_cls

Returns The tested Action class

test_fetching()
Tests fetching using different methods :return: None

test_using_filter_and_id()
Tests that using an ID and an explicit filter does not work :return: None

bundesliga_tippspiel.test.actions.TestGetPlayerAction module

class bundesliga_tippspiel.test.actions.TestGetPlayerAction.**TestGetPlayerAction**(*methodName='*
Bases: bundesliga_tippspiel.test.actions.GetActionTestFramework._GetActionTestFramework

Class that tests the GetTeam action

property action_cls

Returns The tested Action class

test_fetching()
Tests fetching using different methods :return: None

test_using_filter_and_id()
Tests that using an ID and an explicit filter does not work :return: None

bundesliga_tippspiel.test.actions.TestGetReminderSettingsAction module

class bundesliga_tippspiel.test.actions.TestGetReminderSettingsAction.**TestGetReminderSettin**
Bases: bundesliga_tippspiel.test.actions.ActionTestFramework._ActionTestFramework

Class that tests the GetReminderSettingsAction action

generate_action() → *bundesliga_tippspiel.actions.GetReminderSettingsAction.GetReminderSettingsAction*
Generates a valid SetReminderSettingsAction object :return: The generated SetReminderSettingsAction

setUp()
Sets up a user for testing :return: None

test_from_dict()
Tests the from_dict method :return: None

test_setting_updating_and_deleting_reminder()
Tests getting a reminder :return: None

bundesliga_tippspiel.test.actions.TestGetTeamAction module

class bundesliga_tippspiel.test.actions.TestGetTeamAction.**TestGetTeamAction**(*methodName='runTe*
Bases: bundesliga_tippspiel.test.actions.GetActionTestFramework._GetActionTestFramework

Class that tests the GetTeam action

property action_cls

Returns The tested Action class

test_fetching()
Tests fetching using different methods :return: None

test_using_filter_and_id()
Tests that using an ID and an explicit filter does not work :return: None

bundesliga_tippspiel.test.actions.TestLeaderboardAction module

class bundesliga_tippspiel.test.actions.TestLeaderboardAction.**TestLeaderboardAction**(*methodNa*
Bases: bundesliga_tippspiel.test.actions.ActionTestFramework._ActionTestFramework

Class that tests the Login action

generate_action() → *bundesliga_tippspiel.actions.LeaderboardAction.LeaderboardAction*
Generates a valid LeaderboardAction object :return: The generated LeaderboardAction

setUp()
Sets up users for testing :return: None

test_leaderboard()
Tests that the leaderboard is generated correctly :return: None

test_leaderboard_for_matchday()
Tests generating a leaderboard for a specific matchday :return: None

bundesliga_tippspiel.test.actions.TestPlaceBetsAction module

class bundesliga_tippspiel.test.actions.TestPlaceBetsAction.**TestPlaceBetsAction**(*methodName='*
Bases: bundesliga_tippspiel.test.actions.ActionTestFramework._ActionTestFramework

Test class that tests the place bets action

assert_bet(*match_id: int*, *home: int*, *away: int*)
Tests that a stored bet has the specified parameters :param match_id: The match ID of the bet to test :param home: The home score to check for :param away: The away score to check for :return: None

generate_action() → *bundesliga_tippspiel.actions.PlaceBetsAction.PlaceBetsAction*
Generates a new, valid Action object :return: The generated object

setUp()
Sets up a user in the database :return: None

test_bets_out_of_bounds()
Tests using bets that are out of bounds :return: None

test_invalid_bets()
Tests using invalid data to place bets. :return: None

test_mixing_valid_and_invalid_bets()
Tests placing both valid and invalid bets :return: None

test_parsing_form_data()
Tests if parsing the form data works correctly :return: None

test_placing_bets()
Tests placing bets. :return:

test_updating_bets()
Tests placing bets again :return: None

bundesliga_tippspiel.test.actions.TestSetReminderSettingsAction module

class bundesliga_tippspiel.test.actions.TestSetReminderSettingsAction.**TestSetReminderSettin**
Bases: bundesliga_tippspiel.test.actions.ActionTestFramework._ActionTestFramework

Class that tests the SetReminderSettings action

generate_action() → *bundesliga_tippspiel.actions.SetReminderSettingsAction.SetReminderSettingsAction*
Generates a valid SetReminderSettingsAction object :return: The generated SetReminderSettingsAction

setUp()
Sets up a user for testing :return: None

test_setting_and_updating_reminder()
Tests setting a reminder and updating it :return: None

test_setting_invalid_reminder_times()
Tests setting a reminder time that's not 0 < t < 49 :return: None

Module contents

bundesliga_tippspiel.test.background package

Submodules

bundesliga_tippspiel.test.background.TestMatchDataUpdate module

class bundesliga_tippspiel.test.background.TestMatchDataUpdate.**TestMatchDataUpdate**(*methodNa*
Bases: bundesliga_tippspiel.test.TestFramework._TestFramework

Unit test class that tests the match_data_getter script

assert_db_state()
Performs multiple assertions on a filled database :return: None

test_icon_urls()
Tests if all team icon URLs are valid :return: None

test_populating_twice()
Tests populating the database. Twice. :return: None

bundesliga_tippspiel.test.background.TestReminders module

class bundesliga_tippspiel.test.background.TestReminders.**TestReminders**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.TestFramework._TestFramework

Class that tests the sending of due email reminders

setUp()
Sets up a user for testing :return: None

test_with_due_reminder()
Tests running the action with a due reminder :return: None

test_with_non_due_reminder()
Tests running the action with a reminder that's not due yet :return: None

test_without_stored_reminders()
Tests running the action without any stored reminders :return: None

bundesliga_tippspiel.test.background.TestSeasonEvents module

class bundesliga_tippspiel.test.background.TestSeasonEvents.**TestSeasonEvents**(*methodName='runT*
Bases: bundesliga_tippspiel.test.TestFramework._TestFramework

Class that tests the sending of due email reminders

setUp()
Sets up a user for testing :return: None

test_executing()
Tests if the events are executed correctly, and only once. :return: None

test_initializing()
Tests initializing the season-wide database entries :return: None

test_midseason_reminder()
Tests sending the midseason reminder :return: None

test_postseason_wrapup()
Tests doing the postseason wrapup :return: None

test_preseason_reminder()
Tests sending the preseason reminder if it's due :return: None

Module contents

bundesliga_tippspiel.test.models package

Subpackages

bundesliga_tippspiel.test.models.match_data package

Submodules

bundesliga_tippspiel.test.models.match_data.TestGoal module

class bundesliga_tippspiel.test.models.match_data.TestGoal.**TestGoal**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.models.ModelTestFramework._ModelTestFramework

Tests the Goal SQL model

setUp()
Sets up the data needed by the tests :return: None

test_auto_increment()
Tests that auto-incrementing works as expected :return: None

test_cascades()
Tests if cascade deletes work correctly :return: None

test_deleting_from_db()
Tests deleting model objects from the database :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_missing_column_data()
Tests that missing column data is handled correctly :return: None

test_retrieving_from_db()
Tests retrieving model objects from the database :return: None

test_string_representation()
Tests the str and repr methods of the model :return: None

test_uniqueness()
Tests that unique attributes are correctly checked :return: None

bundesliga_tippspiel.test.models.match_data.TestMatch module

class bundesliga_tippspiel.test.models.match_data.TestMatch.**TestMatch**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.models.ModelTestFramework._ModelTestFramework

Tests the Match SQL model

setUp()
Sets up the data needed by the tests :return: None

test_auto_increment()
Tests that auto-incrementing works as expected :return: None

test_cascades()
Tests if cascade deletes work correctly :return: None

test_deleting_from_db()
Tests deleting model objects from the database :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_kickoff_datetime_conversion()
Tests the kickoff_datetime metod :return: None

test_minute_representation()
Tests the representation of the current minute of the match :return: None

test_missing_column_data()
Tests that missing column data is handled correctly :return: None

test_retrieving_from_db()
Tests retrieving model objects from the database :return: None

test_score_representations()
Tests the score representation attributes :return: None

test_string_representation()
Tests the str and repr methods of the model :return: None

test_uniqueness()
Tests that unique attributes are correctly checked :return: None

bundesliga_tippspiel.test.models.match_data.TestPlayer module

class bundesliga_tippspiel.test.models.match_data.TestPlayer.**TestPlayer**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.models.ModelTestFramework._ModelTestFramework

Tests the Player SQL model

setUp()
Sets up the data needed by the tests :return: None

test_auto_increment()
Tests that auto-incrementing works as expected :return: None

test_cascades()
Tests if cascade deletes work correctly :return: None

test_deleting_from_db()
Tests deleting model objects from the database :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_missing_column_data()
Tests that missing column data is handled correctly :return: None

test_retrieving_from_db()
Tests retrieving model objects from the database :return: None

test_string_representation()
Tests the str and repr methods of the model :return: None

test_uniqueness()
Tests that unique attributes are correctly checked :return: None

bundesliga_tippspiel.test.models.match_data.TestTeam module

class bundesliga_tippspiel.test.models.match_data.TestTeam.**TestTeam**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.models.ModelTestFramework._ModelTestFramework

Tests the Team SQL model

setUp()
Sets up the data needed by the tests :return: None

test_auto_increment()
Tests that auto-incrementing works as expected :return: None

test_cascades()
Tests if cascade deletes work correctly :return: None

test_deleting_from_db()
Tests deleting model objects from the database :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_missing_column_data()
Tests that missing column data is handled correctly :return: None

test_retrieving_from_db()
Tests retrieving model objects from the database :return: None

test_string_representation()
Tests the str and repr methods of the model :return: None

test_uniqueness()
Tests that unique attributes are correctly checked :return: None

Module contents

bundesliga_tippspiel.test.models.settings package

Submodules

bundesliga_tippspiel.test.models.settings.TestReminderSettings module

class bundesliga_tippspiel.test.models.settings.TestReminderSettings.**TestReminderSettings**(*
Bases: bundesliga_tippspiel.test.models.ModelTestFramework._ModelTestFramework

Tests the ReminderSettings SQL model

setUp()
Sets up the data needed by the tests :return: None

test_auto_increment()
Tests that auto-incrementing works as expected :return: None

test_cascades()
Tests if cascade deletes work correctly :return: None

test_deleting_from_db()
Tests deleting model objects from the database :return: None

test_due()
Tests if due matches can be found correctly

Dates: Now —— Reminder 1 —— Match 1 —— Reminder 2 —— Match 2 -0————-10min————30min————60min————120min- :return: None

test_due_when_bets_placed()
Tests if the is_due method works correctly when bets are placed :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_missing_column_data()
Tests that missing column data is handled correctly :return: None

test_retrieving_from_db()
Tests retrieving model objects from the database :return: None

test_string_representation()
Tests the str and repr methods of the model :return: None

test_uniqueness()
Tests that unique attributes are correctly checked :return: None

Module contents

bundesliga_tippspiel.test.models.user_generated package

Submodules

bundesliga_tippspiel.test.models.user_generated.TestBet module

class bundesliga_tippspiel.test.models.user_generated.TestBet.**TestBet**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.models.ModelTestFramework._ModelTestFramework

Tests the Bet SQL model

setUp()
Sets up the data needed by the tests :return: None

test_auto_increment()
Tests that auto-incrementing works as expected :return: None

test_cascades()
Tests if cascade deletes work correctly :return: None

test_deleting_from_db()
Tests deleting model objects from the database :return: None

test_evaluating_bet_result()
Tests evaluating the results of a bet :return: None

test_evaluating_unfinished_games()
Tests that evaluating unfinished games behaves correctly :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_missing_column_data()
Tests that missing column data is handled correctly :return: None

test_retrieving_from_db()
Tests retrieving model objects from the database :return: None

test_string_representation()
Tests the str and repr methods of the model :return: None

test_uniqueness()
Tests that unique attributes are correctly checked :return: None

bundesliga_tippspiel.test.models.user_generated.TestChatMessage module

class bundesliga_tippspiel.test.models.user_generated.TestChatMessage.**TestChatMessage**(*metho*
Bases: bundesliga_tippspiel.test.models.ModelTestFramework._ModelTestFramework

Tests the ChatMessage SQL model

setUp()
Sets up the data needed by the tests :return: None

test_auto_increment()
Tests that auto-incrementing works as expected :return: None

test_deleting_from_db()
Tests deleting model objects from the database :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_missing_column_data()
Tests that missing column data is handled correctly :return: None

test_retrieving_from_db()
Tests retrieving model objects from the database :return: None

test_string_representation()
Tests the str and repr methods of the model :return: None

test_using_chat()
Test typical interactions with the chat systems :return: None

bundesliga_tippspiel.test.models.user_generated.TestSeasonPositionBet module

class bundesliga_tippspiel.test.models.user_generated.TestSeasonPositionBet.**TestSeasonPosit**
Bases: bundesliga_tippspiel.test.models.ModelTestFramework._ModelTestFramework

Tests the SeasonPositionBet SQL model

setUp()
Sets up the data needed by the tests :return: None

test_cascades()
Tests if cascade deletes work correctly :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_missing_column_data()
Tests that missing column data is handled correctly :return: None

test_string_representation()
Tests the str and repr methods of the model :return: None

test_uniqueness()
Tests that unique attributes are correctly checked :return: None

bundesliga_tippspiel.test.models.user_generated.TestSeasonTeamBet module

class bundesliga_tippspiel.test.models.user_generated.TestSeasonTeamBet.**TestSeasonTeamBet**(
Bases: bundesliga_tippspiel.test.models.ModelTestFramework._ModelTestFramework

Tests the SeasonTeamBet SQL model

setUp()
Sets up the data needed by the tests :return: None

test_cascades()
Tests if cascade deletes work correctly :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_missing_column_data()
Tests that missing column data is handled correctly :return: None

test_string_representation()
Tests the str and repr methods of the model :return: None

test_uniqueness()
Tests that unique attributes are correctly checked :return: None

bundesliga_tippspiel.test.models.user_generated.TestSeasonWinner module

class bundesliga_tippspiel.test.models.user_generated.TestSeasonWinner.**TestSeasonWinner**(*met*
Bases: bundesliga_tippspiel.test.models.ModelTestFramework._ModelTestFramework

Tests the SeasonWinner SQL model

setUp()
Sets up the data needed by the tests :return: None

test_cascades()
Tests if cascade deletes work correctly :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_missing_column_data()
Tests that missing column data is handled correctly :return: None

test_season_string()
Tests the season string :return: None

test_string_representation()
Tests the str and repr methods of the model :return: None

test_uniqueness()
Tests that unique attributes are correctly checked :return: None

Module contents

Submodules

bundesliga_tippspiel.test.models.ModelTestFramework module

bundesliga_tippspiel.test.models.TestMatchData module

class bundesliga_tippspiel.test.models.TestMatchData.**TestMatchData**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.TestFramework._TestFramework

Class that tests the various match data models

test_model_relations()
Tests the relations between the models :return: None

bundesliga_tippspiel.test.models.TestSeasonEvent module

class bundesliga_tippspiel.test.models.TestSeasonEvent.**TestSeasonEvent**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.models.ModelTestFramework._ModelTestFramework

Tests the SeasonEvent SQL model

setUp()
Sets up the data needed by the tests :return: None

test_cascades()
Tests if cascade deletes work correctly :return: None

test_json_representation()
Tests the JSON representation of the model :return: None

test_missing_column_data()
Tests that missing column data is handled correctly :return: None

test_string_representation()
Tests the str and repr methods of the model :return: None

test_uniqueness()
Tests that unique attributes are correctly checked :return: None

Module contents

bundesliga_tippspiel.test.routes package

Subpackages

bundesliga_tippspiel.test.routes.api package

Submodules

bundesliga_tippspiel.test.routes.api.ApiRouteTestFramework module

bundesliga_tippspiel.test.routes.api.GetterApiRouteTestFramework module

bundesliga_tippspiel.test.routes.api.TestApiKeyApiRoute module

class bundesliga_tippspiel.test.routes.api.TestApiKeyApiRoute.**TestApiKeyApiRoute**(*methodName=*
Bases: bundesliga_tippspiel.test.routes.api.ApiRouteTestFramework._ApiRouteTestFramework

Tests the /api_key API route

property route_info
Provides information about the route :return: The path of the route,

A list of supported methods, Whether or not the API endpoint requires authorization

test_successful_call()
Tests a successful API call :return: None

test_unsuccessful_call()
Tests an unsuccessful API call :return: None

bundesliga_tippspiel.test.routes.api.TestAuthorizeApiRoute module

class bundesliga_tippspiel.test.routes.api.TestAuthorizeApiRoute.**TestAuthorizeApiRoute**(*meth*
Bases: bundesliga_tippspiel.test.routes.api.ApiRouteTestFramework.
_ApiRouteTestFramework

Tests the /authorize API route

property route_info
Provides information about the route :return: The path of the route,

A list of supported methods, Whether or not the API endpoint requires authorization

test_expired_api_key()
Tests using an expired API key :return: None

test_successful_call()
Tests a successful API call :return: None

test_unsuccessful_call()
Tests an unsuccessful API call :return: None

bundesliga_tippspiel.test.routes.api.TestGetBetApiRoute module

class bundesliga_tippspiel.test.routes.api.TestGetBetApiRoute.**TestGetBetApiRoute**(*methodName=*
Bases: bundesliga_tippspiel.test.routes.api.GetterApiRouteTestFramework.
_GetterApiRouteTestFramework

Tests the /bet GET API route

property action_cls

Returns The action class used to fetch data

property keyword

Returns The route keyword

property sample_filters

Returns A sample dictionary of filters with appropriate values

bundesliga_tippspiel.test.routes.api.TestGetGoalApiRoute module

class bundesliga_tippspiel.test.routes.api.TestGetGoalApiRoute.**TestGetGoalApiRoute**(*methodNa*
Bases: bundesliga_tippspiel.test.routes.api.GetterApiRouteTestFramework.
_GetterApiRouteTestFramework

Tests the /goal GET API route

property action_cls

Returns The action class used to fetch data

property keyword

Returns The route keyword.

property sample_filters

Returns A sample dictionary of filters with appropriate values

bundesliga_tippspiel.test.routes.api.TestGetMatchApiRoute module

class bundesliga_tippspiel.test.routes.api.TestGetMatchApiRoute.**TestGetMatchApiRoute**(*methodI*
Bases: bundesliga_tippspiel.test.routes.api.GetterApiRouteTestFramework.
_GetterApiRouteTestFramework

Tests the /match GET API route

property action_cls

Returns The action class used to fetch data

property keyword

Returns The route keyword.

property sample_filters

Returns A sample dictionary of filters with appropriate values

bundesliga_tippspiel.test.routes.api.TestGetPlayerApiRoute module

class bundesliga_tippspiel.test.routes.api.TestGetPlayerApiRoute.**TestGetPlayerApiRoute**(*meth*
Bases: bundesliga_tippspiel.test.routes.api.GetterApiRouteTestFramework.
_GetterApiRouteTestFramework

Tests the /player GET API route

property action_cls

Returns The action class used to fetch data

property keyword

Returns The route keyword.

property sample_filters

Returns A sample dictionary of filters with appropriate values

bundesliga_tippspiel.test.routes.api.TestGetTeamApiRoute module

class bundesliga_tippspiel.test.routes.api.TestGetTeamApiRoute.**TestGetTeamApiRoute**(*methodNa*
Bases: bundesliga_tippspiel.test.routes.api.GetterApiRouteTestFramework.
_GetterApiRouteTestFramework

Tests the /team GET API route

property action_cls

Returns The action class used to fetch data

property keyword

Returns The route keyword

property sample_filters

Returns A sample dictionary of filters with appropriate values

bundesliga_tippspiel.test.routes.api.TestLeaderboardApiRoute module

class bundesliga_tippspiel.test.routes.api.TestLeaderboardApiRoute.**TestLeaderboardApiRoute**
Bases: bundesliga_tippspiel.test.routes.api.ApiRouteTestFramework._ApiRouteTestFramework

Tests the /leaderboard API route

property route_info
Provides information about the route :return: The path of the route,

A list of supported methods, Whether or not the API endpoint requires authorization

test_successful_call()
Tests a successful API call :return: None

test_unsuccessful_call()
Tests an unsuccessful API call :return: None

bundesliga_tippspiel.test.routes.api.TestPutBetApiRoute module

class bundesliga_tippspiel.test.routes.api.TestPutBetApiRoute.**TestPutBetApiRoute**(*methodName=*
Bases: bundesliga_tippspiel.test.routes.api.ApiRouteTestFramework._ApiRouteTestFramework

Tests the /bet PUT API route

property route_info
Provides information about the route :return: The path of the route,

A list of supported methods, Whether or not the API endpoint requires authorization

setUp()
Sets up data for the tests :return: None

test_successful_call()
Tests a successful API call :return: None

test_unsuccessful_call()
Tests an unsuccessful API call :return: None

Module contents

Submodules

bundesliga_tippspiel.test.routes.RouteTestFramework module

bundesliga_tippspiel.test.routes.TestAboutRoute module

class bundesliga_tippspiel.test.routes.TestAboutRoute.**TestAboutRoute**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /about route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

test_malformed_data()
Tests that malformed data in the request is handled appropriately :return: None

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestBetsRoute module

class bundesliga_tippspiel.test.routes.TestBetsRoute.**TestBetsRoute**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /bets route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

setUp()
Sets up data for the tests :return:

test_getting_specific_matchday()
Tests retrieving a specific matchday :return: None

test_malformed_data()
Tests that malformed data in the request is handled appropriately :return: None

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestChangePasswordRoute module

class bundesliga_tippspiel.test.routes.TestChangePasswordRoute.**TestChangePasswordRoute**(*meth*
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /change_password route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestConfirmRoute module

class bundesliga_tippspiel.test.routes.TestConfirmRoute.**TestConfirmRoute**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /confirm route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestDeleteUserRoute module

class bundesliga_tippspiel.test.routes.TestDeleteUserRoute.**TestDeleteUserRoute**(*methodName='ru*
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /delete_user route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestEmailReminderRoute module

class bundesliga_tippspiel.test.routes.TestEmailReminderRoute.**TestSetReminderRoute**(*methodNam*
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /set_reminder route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestForgotPasswordRoute module

class bundesliga_tippspiel.test.routes.TestForgotPasswordRoute.**TestForgotPasswordRoute**(*meth*
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /forgot route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestIndexRoute module

class bundesliga_tippspiel.test.routes.TestIndexRoute.**TestIndexRoute**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the / route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

test_malformed_data()
Tests that malformed data in the request is handled appropriately :return: None

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestLeaderboardRoute module

class bundesliga_tippspiel.test.routes.TestLeaderboardRoute.**TestLeaderboardRoute**(*methodName=*
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /leaderboard route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

setUp()
Generates sample match data :return: None

test_malformed_data()
Tests that malformed data in the request is handled appropriately :return: None

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestLoginRoute module

class bundesliga_tippspiel.test.routes.TestLoginRoute.**TestLoginRoute**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /login route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestMatchRoute module

class bundesliga_tippspiel.test.routes.TestMatchRoute.**TestMatchRoute**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /match route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

setUp()
Sets up data for the tests :return:

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestPrivacyRoute module

class bundesliga_tippspiel.test.routes.TestPrivacyRoute.**TestPrivacyRoute**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /privacy route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

test_malformed_data()
Tests that malformed data in the request is handled appropriately :return: None

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestProfileRoute module

class bundesliga_tippspiel.test.routes.TestProfileRoute.**TestProfileRoute**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /profile route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

test_malformed_data()
Tests that malformed data in the request is handled appropriately :return: None

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestRegisterRoute module

class bundesliga_tippspiel.test.routes.TestRegisterRoute.**TestRegisterRoute**(*methodName='runTest*
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /register route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestSeasonBetsRoute module

class bundesliga_tippspiel.test.routes.TestSeasonBetsRoute.**TestSeasonBetsRoute**(*methodName='ru*
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /season_bets route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

setUp()
Sets up data for the tests :return: None

test_malformed_data()
Tests that malformed data in the request is handled appropriately :return: None

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestSeasonPositionBetsRoute module

class bundesliga_tippspiel.test.routes.TestSeasonPositionBetsRoute.**TestSeasonPositionBetsRo**
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /season_position_bets route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

setUp()
Sets up data for the tests :return:

test_malformed_data()
Tests that malformed data in the request is handled appropriately :return: None

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestSeasonTeamBetsRoute module

class bundesliga_tippspiel.test.routes.TestSeasonTeamBetsRoute.**TestSeasonTeamBetsRoute**(*meth*
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /season_team_bets route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

setUp()
Sets up data for the tests :return:

test_malformed_data()
Tests that malformed data in the request is handled appropriately :return: None

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestStatsRoute module

class bundesliga_tippspiel.test.routes.TestStatsRoute.**TestStatsRoute**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /stats route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

setUp()
Generates sample match data :return: None

test_malformed_data()
Tests that malformed data in the request is handled appropriately :return: None

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestTeamRoute module

class bundesliga_tippspiel.test.routes.TestTeamRoute.**TestTeamRoute**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /team route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

setUp()
Sets up data for the tests :return:

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

bundesliga_tippspiel.test.routes.TestUserRoute module

class bundesliga_tippspiel.test.routes.TestUserRoute.**TestUserRoute**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.routes.RouteTestFramework._RouteTestFramework

Class that tests the /user route

property route_info
Info about the route to test :return: The route's path,

the route's primary methods, A phrase found on the route's GET page. None if no such page exists, An indicator for if the page requires authentication or not

setUp()
Sets up data for the tests :return:

test_successful_requests()
Tests (a) successful request(s) :return: None

test_unsuccessful_requests()
Tests (an) unsuccessful request(s) :return: None

Module contents

bundesliga_tippspiel.test.utils package

Submodules

bundesliga_tippspiel.test.utils.TestChartData module

class bundesliga_tippspiel.test.utils.TestChartData.**TestChartData**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.TestFramework._TestFramework

Tests chart data generation

test_generating_leaderboard_data()
Tests generating leaderboard data :return: None

bundesliga_tippspiel.test.utils.TestCrypto module

class bundesliga_tippspiel.test.utils.TestCrypto.**TestCrypto**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.TestFramework._TestFramework

Tests cryptographical functions

test_hashing()
Tests that passwords can be hashed successfully :return: None

test_hashing_strings()
Tests that password hashing and verifying works with string as well :return: None

test_salt()
Tests that password hashes are salted :return: None

test_verifying_with_invalid_hash()
Tests that attempting to verify a password with an incorrectly formatted hash will return False :return: None

bundesliga_tippspiel.test.utils.TestDb module

class bundesliga_tippspiel.test.utils.TestDb.**TestDb**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.TestFramework._TestFramework

Class that tests database helper functions

test_user_exists_functions()
Tests functions that check if a user exists or not :return: None

bundesliga_tippspiel.test.utils.TestJson module

class bundesliga_tippspiel.test.utils.TestJson.**TestJson**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.TestFramework._TestFramework

Unit test class that tests the json methods used for the API

test_jsonifying()
Tests jsonifying some objects :return: None

bundesliga_tippspiel.test.utils.TestStats module

class bundesliga_tippspiel.test.utils.TestStats.**TestStats**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.TestFramework._TestFramework

Tests stats generation

test_generating_team_points_data()
Tests generating team points data :return: None

test_rankings_before_season()
Tests the generation of rankings before the season has started :return: None

Module contents

Submodules

bundesliga_tippspiel.test.TestConfig module

class bundesliga_tippspiel.test.TestConfig.**TestConfig**(*methodName='runTest'*)
Bases: bundesliga_tippspiel.test.TestFramework._TestFramework

Tests the config class

test_season_calculations()
Tests if the season calculations are done correctly :return: None

bundesliga_tippspiel.test.TestFramework module

Module contents

1.1.6 bundesliga_tippspiel.utils package

Submodules

bundesliga_tippspiel.utils.chart_data module

bundesliga_tippspiel.utils.chart_data.**generate_leaderboard_data**(*current_matchday: Optional[int] = None*, *bets: Optional[List[*bundesliga_tippspiel.db.user = *None*, *include_bots: bool = False*) → Tuple[int, Dict[str, Tuple[str, List[int]]]]

Generates leaderboard data for the rankings chart :param current_matchday: The current matchday :param bets: A list of bets to work on. If not provided, will

load all bets in the database

Parameters **include_bots** – Whether or not to include bots

Returns

A tuple consisting of the matchday to display and the leaderboard data:

username: (colour, list of positions per matchday)

bundesliga_tippspiel.utils.chart_data.**load_leaderboard_history**(*current_matchday: Optional[int] = None*, *bets: Optional[List[*bundesliga_tippspiel.db.user_ = *None*, *include_bots: bool = False*) → List[List[Tuple[puffotter.flask.db.User.Us int]]]

Generates historical leaderboard data for chart generation :param current_matchday: The current matchday :param bets: A list of bets to work on. If not provided, will

load all bets in the database

Parameters **include_bots** – Whether or not to include bots

Returns The list of leaderboard lists

bundesliga_tippspiel.utils.db module

bundesliga_tippspiel.utils.db.**email_exists**(*email: str*) → bool

Checks if an email address already exists in the database :param email: The email to check for :return: True if the email exists in the database, False otherwise

bundesliga_tippspiel.utils.db.**user_exists**(*user_id: int*) → bool

Checks if a username already exists in the database :param user_id: The user's id :return: True if the user exists, False otherwise

bundesliga_tippspiel.utils.db.**username_exists**(*username: str*) → bool
Checks if a username already exists in the database :param username: The username to check :return: True if the username exists in the database, False if not

bundesliga_tippspiel.utils.routes module

bundesliga_tippspiel.utils.routes.**action_route**(*func: Callable*) → Callable
Decorator that catches any ActionExceptions and displays appropriate error messages :param func: The function to wrap :return: The wrapped function

bundesliga_tippspiel.utils.stats module

bundesliga_tippspiel.utils.stats.**create_participation_ranking**(*bets: Optional[List[*bundesliga_tippspiel.db.user_g
= None, include_bots: bool = False) → List[Tuple[puffotter.flask.db.User.User, str]]
Creates a ranking of user's participation percentages :param bets: The bets to analyze. If not provided, will analyze all bets :param include_bots: Whether or not to include bots :return: A sorted list of tuples detailing the participation ranking

bundesliga_tippspiel.utils.stats.**create_point_average_ranking**(*bets: Optional[List[*bundesliga_tippspiel.db.user_g
= None, include_bots: bool = False) → List[Tuple[puffotter.flask.db.User.User, str]]
Creates a ranking of points averages :param bets: The bets to analyze :param include_bots: Whether or not to include bots :return: The ranking

bundesliga_tippspiel.utils.stats.**generate_points_distributions**(*bets: Optional[List[*bundesliga_tippspiel.db.user_
= None) → Dict[puffotter.flask.db.User.User, Dict[int, int]]
Generates a distribution detailing how often a given amount of points a user earned while betting :param bets: The bets to analyze. If not provided, will analyze all bets :return: A dictionary mapping users to point amounts to their

appearance count

bundesliga_tippspiel.utils.stats.**generate_team_points_table**(*team_points: Dict[*bundesliga_tippspiel.db.match_data.Tea
int]) → List[Tuple[*bundesliga_tippspiel.db.match_da
int]]
Generates a sorted list of tuples of teams and their points. :param team_points: The points achieved by the teams :return: The sorted list of tuples

bundesliga_tippspiel.utils.stats.**get_team_points_data**(*bets: Optional[List[*bundesliga_tippspiel.db.user_generated.Be = *None*) → Dict[puffotter.flask.db.User.User, Dict[*bundesliga_tippspiel.db.match_data.Team.Team*, int]]

Generates inforation about the amount of points each user achieved betting on specific teams :param bets: The bets to analyze. If not provided, will analyze all bets :return: A dictionary mapping users to dictionaries mapping teams to points

bundesliga_tippspiel.utils.stats.**get_total_points_per_team**(*bets: Optional[List[*bundesliga_tippspiel.db.user_genera = *None*) → Dict[*bundesliga_tippspiel.db.match_data.Team* int]

Retrieves the total amount of points achieved by betting on individual teams :param bets: The bets to analyze. If not provided, will analyze all bets :return: A dictionary mapping the teams to the points achieved by users

betting on them

Module contents

1.2 Submodules

1.3 bundesliga_tippspiel.Config module

class bundesliga_tippspiel.Config.**Config**

Bases: puffotter.flask.Config.Config

Configuration for the flask application

OPENLIGADB_LEAGUE: str

The openligadb league to use

OPENLIGADB_SEASON: str

The openligadb season to use

classmethod environment_variables() → Dict[str, List[str]]

Specifies required and optional environment variables :return: The specified environment variables in two lists in

a dictionary, grouped by whether the variables are required or optional

classmethod season() → int

Returns The current season

classmethod season_string(*year: Optional[int] = None*) → str

Returns The season string, e.g. 2020/21 for 2020

1.4 bundesliga_tippspiel.enums module

class bundesliga_tippspiel.enums.**ReminderType**(*value*)

Bases: enum.Enum

Class that specifies the different reminder types

EMAIL = 'email'

TELEGRAM = 'telegram'

1.5 bundesliga_tippspiel.exceptions module

exception bundesliga_tippspiel.exceptions.**ActionException**(*reason: str*, *display_message: str*, *status_code: int = 400*, *severity: puffotter.flask.enums.AlertSeverity = <AlertSeverity.DANGER: 'danger'>*)

Bases: Exception

An exception that gets raised whenever an action method fails for whatever reason. The reason should be provided as a parameter

__init__(*reason: str*, *display_message: str*, *status_code: int = 400*, *severity: puffotter.flask.enums.AlertSeverity = <AlertSeverity.DANGER: 'danger'>*)

Initializes the Exception while adding the 'reason' variable :param reason: The reason for the exception :param display_message: The message to display :param status_code: The status code corresponding to the exception

Defaults to 400

Parameters **severity** – The severity of the exception. Defaults to DANGER

flash()

Flashes the message so that it can be displayed on the next redirect :return: None

1.6 bundesliga_tippspiel.jinja_extras module

bundesliga_tippspiel.jinja_extras.**get_matchday_total_pill_class**(*points: int*) → str

Calculates the appropriate pill badge for an amount of points on a matchday :param points: The points for which to get the pill class :return: The appropriate pill badge class

bundesliga_tippspiel.jinja_extras.**get_pill_class**(*points: int*) → str

Calculates the appropriate pill badge for an amount of points :param points: The points for which to get the pill class :return: The appropriate pill badge class

bundesliga_tippspiel.jinja_extras.**jinja_extras**() → Dict[str, Any]

Makes sure that jinja has access to these variables :return: The variables to forward to jinja

1.7 bundesliga_tippspiel.main module

bundesliga_tippspiel.main.**main**()
Initializes and starts the flask application :return: None

1.8 bundesliga_tippspiel.template_extras module

bundesliga_tippspiel.template_extras.**profile_extras**() → Dict[str, Any]
Makes sure that the profile page has access to information on email reminders. :return: The variables to forward to the template

1.9 Module contents

bundesliga_tippspiel.**root_path: str = '/usr/local/lib/python3.6/dist-packages/bundesliga_t**
The root path of the application

bundesliga_tippspiel.**sentry_dsn = 'https://e91e468e84424758bd74e6908af2c565@sentry.namibsu**
The sentry DSN used for exception logging

CHAPTER

TWO

BUNDESLIGA_TIPPSPIEL

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

S

U

V

W