HSE Buisness Club Telegram-bot Documentation

Release v3.0.0

HSE IT Team

May 27, 2025

CONTENTS

CHAPTER

ONE

CONFIG

1.1 texts

1.1.1 admin

Constants

config.texts.admin.EVENT_END_FEEDBACK

config.texts.admin.THEMES

1.1.2 commands

Constants

config.texts.commands.WELCOME_MESSAGE

config.texts.commands.INFO_ADMIN

config.texts.commands.INFO_USER

1.1.3 months

Constants

config.texts.months.MONTHS

config.texts.months.MONTH_MAP

1.1.4 quest

Constants

config.texts.quest.QUEST_START

config.texts.quest.QUEST_FIRST_STEP

1.2 settings

Settings module Application settings and configuration for bot and database.

config.settings.SQL_URL = 'sqlite+aiosqlite:///./database/database.db'

Development database configuration using SQLite.

1.2.1 Constants

config.settings.TOKEN

config.settings.ENV

Default to development if not set

config.settings.NETWORKING_THEMES

config.settings.LOG_LEVEL

config.settings.LOG_FORMAT

CHAPTER

TWO

DATABASE

2.1 req

2.1.1 event

DB Operations

CRUD operations for users, club_events, vacancies, and networking.

async database.req.event.get_event(name: str)

Retrieve an event by name.

Parameters
name (str) – Event name.

Returns
Event object or “not created”.

Return type
Event or str

async database.req.event.create_event(name: str, data: dict)

Create a new event.

Parameters

- name (str) – Event name.
- data (dict) – Event data fields.

Returns
Confirmation message.

Return type
str

async database.req.event.update_event(name: str, data: dict)

Update existing event data.

Parameters

- name (str) – Event name.
- data (dict) – Fields to update.

Returns
None

async database.req.event.get_all_events_in_p()

Fetch names of club_events in progress.

Returns
: List of event names.

Return type
: list[str]

async database.req.event.get_all_events()

Fetch all event names.

Returns
: List of event names.

Return type
: list[str]

2.1.2 face_control

DB Operations

CRUD operations for users, club_events, vacancies, and networking.

async database.req.face_control.add_face_control(user_id: int, admin_id: int, username: str = None, full_name: str = None)

Add a new face control user.

Parameters

- user_id (int) – Telegram ID of the user to add as face control
- admin_id (int) – Telegram ID of the admin who is adding the user
- username (str, optional) – Telegram username of the user
- full_name (str, optional) – Full name of the user

Returns
: Created face control instance

Return type
: FaceControl

Raises
: Error409 – If user is already a face control

async database.req.face_control.remove_face_control(user_id: int)

Remove a user from face control.

Parameters
: user_id (int) – Telegram ID of the user to remove

Returns
: True if user was removed, False if user wasn't a face control

Return type
: bool

async database.req.face_control.get_face_control(user_id: int)

Get face control user by Telegram ID.

Parameters
: user_id (int) – Telegram ID of the user

Returns
: Face control instance or “not found”

Return type
: FaceControl or str

async database.req.face_control.list_face_control()

List all face control users.

Returns
: List of all face control users

Return type
: list[FaceControl]

2.1.3 networking

DB Operations

CRUD operations for users, club_events, vacancies, and networking.

async database.req.networking.add_user_to_networking(tg_id: int)

Add a user to networking table.

Parameters
: tg_id (int) – Telegram user ID.

Returns
: Confirmation message ‘ok’.

Return type
: str

Raises
: Error409 – If user already in networking.

async database.req.networking.get_all_for_networking()

Retrieve all user IDs from networking.

Returns
: List of networking user IDs.

Return type
: list[int]

Raises
: Error404 – If no networking data.

async database.req.networking.delete_all_from_networking()

Delete all entries from networking table.

2.1.4 qr

DB Operations

CRUD operations for users, club_events, vacancies, and networking.

async database.req.qr.create_qr_code(user_id: int, event_name: str)

Create a new QR code for a user and event.

Parameters

- user_id (int) – Telegram user identifier.
- event_name (str) – Event name.

Returns

Created QR code instance.

Return type

QRCode

async database.req.qr.get_latest_qr_code(user_id: int)

Get the latest QR code for a user.

Parameters

user_id (int) – Telegram user identifier.

Returns

Latest QR code instance or None if not found.

Return type

QRCode or None

async database.req.qr.mark_qr_code_used(qr_code_id: int)

Mark a QR code as used.

Parameters

qr_code_id (int) – QR code identifier.

Returns

None

async database.req.qr.record_attendance(user_id: int, event_name: str, verified_by: int)

Record user attendance at an event.

Parameters

- user_id (int) – Telegram user identifier.
- event_name (str) – Event name.
- verified_by (int) – Telegram ID of the superuser who verified.

Returns

Created attendance record.

Return type

EventAttendance

async database.req.qr.get_user_attendance(user_id: int, event_name: str)

Check if a user has attended an event.

Parameters

- user_id (int) – Telegram user identifier.
- event_name (str) – Event name.

Returns

Attendance record if found, None otherwise.

Return type
EventAttendance or None

2.1.5 questionary

DB Operations

CRUD operations for users, club_events, vacancies, and networking.

async database.req.questionary.get_questionary(tg_id: int)

Retrieve a questionary by user ID.

Parameters
tg_id (int) – Telegram user identifier.

Returns
Questionary object or "not created".

Return type
Questionary or str

async database.req.questionary.create_questionary(tg_id: int)

Create a new questionary for a user.

Parameters
tg_id (int) – Telegram user identifier.

Returns
None

async database.req.questionary.update_questionary(tg_id: int, data: dict)

Update existing questionary data.

Parameters

- tg_id (int) – Telegram user identifier.
- data (dict) – Fields to update with values.

Returns
None

async database.req.questionary.get_all_quests()

Fetch all questionaries.

Returns
List of Questionary objects.

Return type
list[Questionary]

2.1.6 reg_event

DB Operations

CRUD operations for users, club_events, vacancies, and networking.

async database.req.reg_event.get_reg_event(tg_id: int)

Retrieve registration event data by Telegram ID.

Parameters
tg_id (int) – Telegram user ID for registration.

Returns
: Registration event data.

Return type
: RegEvent

async database.req.reg_event.create_reg_event(tg_id: int)
: Create a new registration event entry.

  Parameters
  : tg_id (int) – Telegram user ID for registration.

  Returns
  : None

async database.req.reg_event.update_reg_event(tg_id: int, data: dict)
: Update fields of an existing registration event.

  Parameters
  - tg_id (int) – Telegram user ID for registration.
  - data (dict) – Fields to update with values.

  Returns
  : None

async database.req.reg_event.check_completly_reg_event(tg_id: int)
: Check if all registration fields are filled.

  Parameters
  : tg_id (int) – Telegram user ID for registration.

  Returns
  : True if all fields non-empty, False otherwise.

  Return type
  : bool

2.1.7 reg_give_away

DB Operations

CRUD operations for users, club_events, vacancies, and networking.

async database.req.reg_give_away.get_ref_give_away(tg_id: int, event_name: str)
: Retrieve a RefGiveAway record for a user in an event.

  Parameters
  - tg_id (int) – Telegram user ID.
  - event_name (str) – Name of the event.

  Returns
  : Referral give-away data.

  Return type
  : RefGiveAway

async database.req.reg_give_away.delete_ref_give_away_row(user_id: int, event_name: str)
: Delete a RefGiveAway entry by user and event.

  Parameters

- user_id (int) – Telegram user ID.
- event_name (str) – Name of the event.

Returns
None

async database.req.reg_give_away.create_ref_give_away(tg_id: int, event_name: str, host_id: int)

Create a new RefGiveAway entry for a user.

Parameters

- tg_id (int) – Telegram user ID.
- event_name (str) – Name of the event.
- host_id (int) – Host user ID.

Returns
None

async database.req.reg_give_away.get_all_from_give_away(user_id: int, event_name: str)

Retrieve all referrals given away by a host in an event.

Parameters

- user_id (int) – Host Telegram user ID.
- event_name (str) – Name of the event.

Returns
Tuples of RefGiveAway and user handler.

Return type
list[tuple]

async database.req.reg_give_away.get_reg_users(event_name: str)

Retrieve registered users and handlers for an event.

Parameters
event_name (str) – Name of the event.

Returns
Tuples of RegEvent and user handler.

Return type
list[tuple]

async database.req.reg_give_away.get_reg_users_stat(event_name: str)

Retrieve user registration statistics for an event.

Parameters
event_name (str) – Name of the event.

Returns
Tuples of UserXEvent and user handler.

Return type
list[tuple]

async database.req.reg_give_away.get_add_winner(host_id: int, event_name: str)

Select a random winner from a host's referrals who attended an event.

Parameters

- host_id (int) – Host Telegram user ID.
- event_name (str) – Name of the event.

Returns
: Telegram user ID of the winner.

Return type
: int

async database.req.reg_give_away.get_users_unreg_tg_id(event_name: str)

Retrieve IDs of users not registered in an event.

Parameters
: event_name (str) – Name of the event.

Returns
: List of unregistered Telegram user IDs.

Return type
: list[int]

async database.req.reg_give_away.get_host(user_id: int, event_name: str)

Retrieve a GiveAwayHost entry by user and event.

Parameters

- user_id (int) – Telegram user ID of the host.
- event_name (str) – Name of the event.

Returns
: Host data for the event.

Return type
: GiveAwayHost

async database.req.reg_give_away.get_host_by_org_name(org_name: str, event_name: str)

Retrieve host by organization and event names.

Parameters

- org_name (str) – Organization name.
- event_name (str) – Event name.

Returns
: Host instance.

Return type
: GiveAwayHost

Raises
: Error404 – If host not found.

async database.req.reg_give_away.create_host(user_id: int, event_name: str, org_name: str)

Create a new host record if none exists.

Parameters

- user_id (int) – User identifier.
- event_name (str) – Event name.
- org_name (str) – Organization name.

Raises
: Error409 – If host already exists.

async database.req.reg_give_away.get_all_hosts_in_event_ids(event_name: str)

Retrieve all host user IDs for an event.

Parameters
: event_name (str) – Event name.

Returns
: List of user IDs.

Return type
: list[int]

Raises
: Error404 – If no hosts found.

async database.req.reg_give_away.get_all_hosts_in_event_orgs(event_name: str)

Retrieve all host organization names for an event.

Parameters
: event_name (str) – Event name.

Returns
: List of organization names.

Return type
: list[str]

Raises
: Error404 – If no hosts found.

2.1.8 user

DB Operations

CRUD operations for users, club_events, vacancies, and networking.

async database.req.user.get_user(tg_id: int)

Retrieve a user by Telegram ID.

Parameters
: tg_id (int) – Telegram user identifier.

Returns
: User object or “not created”.

Return type
: User or str

async database.req.user.create_user(tg_id: int, data: dict)

Create a new user.

Parameters

- tg_id (int) – Telegram user identifier.
- data (dict) – User data fields.

Returns
: Newly created User object.

Return type
User

async database.req.user.update_user(tg_id: int, data: dict)

Update existing user data.

Parameters

- tg_id (int) – Telegram user identifier.
- data (dict) – Fields to update with values.

Returns
None

async database.req.user.get_users_tg_id()

Fetch all distinct Telegram IDs of users.

Returns
List of user Telegram IDs.

Return type
list[int]

async database.req.user.get_all_users()

Fetch all user records.

Returns
List of User objects.

Return type
list[User]

async database.req.user.add_money(tg_id: int, cnt: int)

Increment user's money balance.

Parameters

- tg_id (int) – Telegram user ID.
- cnt (int) – Amount to add.

Raises
Error404 – If user not created.

async database.req.user.one_more_event(tg_id: int)

Increment user's event count by one.

Parameters
tg_id (int) – Telegram user ID.

Raises
Error404 – If user not created.

async database.req.user.add_referal_cnt(tg_id: int)

Increment user's referral count by one.

Parameters
tg_id (int) – Telegram user ID.

Raises
Error404 – If user not created.

async database.req.user.update_strick(tg_id: int, cnt: int = 1)

Update user's strike count.

Parameters

- tg_id (int) – Telegram user ID.
- cnt (int) – Strike increment or reset flag.

Raises

Error404 – If user not created.

async database.req.user.get_user_rank_by_money(specific_user_id: int) → int

Get ranking of a user by money.

Parameters

specific_user_id (int) – User identifier.

Returns

User rank by descending money.

Return type

int

Raises

Error404 – If user not found in ranking.

async database.req.user.get_top_10_users_by_money() → list[User]

Retrieve top ten users by money.

Returns

List of top users.

Return type

list[User]

2.1.9 user_x_event

DB Operations

CRUD operations for users, club_events, vacancies, and networking.

async database.req.user_x_event.get_user_x_event_row(user_id: int, event_name: str)

Retrieve a UserXEvent row.

Parameters

- user_id (int) – Telegram user identifier.
- event_name (str) – Event name.

Returns

Row object or "not created".

Return type

UserXEvent or str

async database.req.user_x_event.delete_user_x_event_row(user_id: int, event_name: str)

Delete a UserXEvent row.

Parameters

- user_id (int) – Telegram user identifier.

- event_name (str) – Event name.

Returns
None

async database.req.user_x_event.create_user_x_event_row(user_id: int, event_name: str, first_contact: str)

Create a new UserXEvent row.

Parameters

- user_id (int) – Telegram user identifier.
- event_name (str) – Event name.
- first_contact (str) – Initial contact detail.

Returns
None

async database.req.user_x_event.update_user_x_event_row_status(user_id: int, event_name: str, new_status: str) → UserXEvent

Update status of a UserXEvent row.

Parameters

- user_id (int) – Telegram user identifier.
- event_name (str) – Event name.
- new_status (str) – New status value.

Returns
Updated row object.

Return type
UserXEvent

async database.req.user_x_event.get_random_user_from_event(event_name: str)

Get random user ID from event with status 'been'.

Parameters
event_name (str) – Event name.

Returns
Random user ID.

Return type
int

async database.req.user_x_event.get_random_user_from_event_wth_bad(event_name: str, bad_ids: list[int]) → int

Get random user ID excluding bad IDs.

Parameters

- event_name (str) – Event name.
- bad_ids (list[int]) – IDs to exclude.

Returns
Random user ID.

Return type
int

async database.req.user_x_event.get_users_tg_id_in_event(event_name: str)

Fetch IDs of users in event with status ‘been’.

Parameters
: event_name (str) – Event name.

Returns
: List of user IDs.

Return type
: list[int]

async database.req.user_x_event.get_users_tg_id_in_event_bad(event_name: str)

Retrieve distinct Telegram user IDs registered in an event.

Parameters
: event_name (str) – Name of the event.

Returns
: List of Telegram user IDs.

Return type
: list[int]

async database.req.user_x_event.get_all_users_in_event(event_name: str)

Retrieve all users marked as ‘been’ for a given event.

Parameters
: event_name (str) – Name of the event.

Returns
: Tuples of UserXEvent and user handler.

Return type
: list[tuple]

async database.req.user_x_event.get_all_user_events(user_id: int)

Retrieve all in-progress club_events for a specific user.

Parameters
: user_id (int) – Telegram user ID.

Returns
: List of Event objects in progress.

Return type
: list[Event]

2.1.10 vacancy

DB Operations

CRUD operations for users, club_events, vacancies, and networking.

async database.req.vacancy.get_vacancy(name: str)

Retrieve a vacancy by name.

Parameters
: name (str) – Vacancy name.

Returns
: Vacancy object or “not created”.

Return type
: Vacancy or str

async database.req.vacancy.add_vacancy(name: str)

Add a new vacancy.

Parameters
: name (str) – Vacancy name.

Returns
: Confirmation message.

Return type
: str

async database.req.vacancy.delete_vacancy(name: str)

Delete a vacancy by name.

Parameters
: name (str) – Vacancy name.

Returns
: Confirmation message.

Return type
: str

async database.req.vacancy.get_all_vacancy_names()

Fetch all distinct vacancy names.

Returns
: List of vacancy names.

Return type
: list[str]

2.2 models

Database Models SQLAlchemy models for users, club_events, vacancies, and registrations.

async database.models.async_main()

Initialize database schema.

Connects to the database and creates all tables.

Returns
: None

class database.models.Base(**kwargs: Any)

Bases: AsyncAttrs, DeclarativeBase

Base class for SQLAlchemy models.

metadata: ClassVar[MetaData] = MetaData()

Refers to the _schema.MetaData collection that will be used for new _schema.Table objects.

See also

orm_declarative_metadata

registry: ClassVar[_RegistryType] = <sqlalchemy.orm.decl_api.registry object>

Refers to the _orm.registry in use where new _orm.Mapper objects will be associated.

class database.models.Event(**kwargs)

Bases: Base

Event model representing a bot event.

Parameters

- name (str) – Event name.
- desc (str) – Event description.
- date (str) – Event date.
- status (str) – Event status.
- time (str) – Event time.
- place (str) – Event location.
- winner (BigInteger) – Winner user ID.

Returns

SQLAlchemy event model instance.

Return type

Event

name

desc

date

status

time

place

winner

class database.models.EventAttendance(**kwargs)

Bases: Base

EventAttendance model for tracking user attendance at club_events.

Parameters

- id (Integer) – Primary key.
- user_id (BigInteger) – Foreign key to user.
- event_name (String) – Foreign key to event.
- attended_at (String) – Timestamp of attendance.
- verified_by (BigInteger) – Foreign key to user (superuser who verified).

Returns

SQLAlchemy attendance model instance.

Return type

EventAttendance

attended_at

Store as ISO format string

id

user_id

event_name

verified_by

class database.models.FaceControl(**kwargs)

Bases: Base

FaceControl model for storing face control users who can verify QR codes.

Parameters

- id (Integer) – Primary key.
- user_id (BigInteger) – Foreign key to user who has face control permissions.
- added_by (BigInteger) – Foreign key to user (admin) who granted the permission.
- added_at (String) – Timestamp when permission was granted.
- username (String) – Telegram username of the face control user.
- full_name (String) – Full name of the face control user.

Returns

SQLAlchemy face control model instance.

Return type

FaceControl

added_at

Store as ISO format string

username

Telegram username

full_name

Full name from Telegram

id

user_id

added_by

class database.models.GiveAwayHost(**kwargs)

Bases: Base

GiveAwayHost model representing event hosts.

Parameters

- user_id (BigInteger) – Host user ID.
- event_name (str) – Event name.
- org_name (str) – Organization name.

Returns
SQLAlchemy host model instance.

Return type
GiveAwayHost

id

user_id

event_name

org_name

class database.models.Networking(**kwargs)

Bases: Base

Networking model placeholder.

Parameters
id (BigInteger) – Primary key.

Returns
SQLAlchemy networking model instance.

Return type
Networking

id

class database.models.QRCode(**kwargs)

Bases: Base

QRCode model for storing QR code data.

Parameters

- id (Integer) – Primary key.
- user_id (BigInteger) – Foreign key to user.
- event_name (String) – Foreign key to event.
- created_at (String) – Timestamp of creation.
- is_used (Boolean) – Whether the QR code has been used.

Returns
SQLAlchemy QR code model instance.

Return type
QRCode

created_at
Store as ISO format string

id

user_id

event_name

is_used

class database.models.Questionary(**kwargs)

Bases: Base

Questionary model storing user's application info.

Parameters

- user_id (BigInteger) – Foreign key to user.
- full_name (str) – Full name.
- degree (str) – Academic degree.
- course (str) – Study course.
- program (str) – Study program.
- email (str) – Email address.
- vacancy (str) – Vacancy applied for.
- motivation (str) – Motivation text.
- plans (str) – Future plans.
- strengths (str) – User strengths.
- career_goals (str) – Career goals.
- team_motivation (str) – Team motivation.
- role_in_team (str) – Role in a team.
- events (str) – Events attended.
- found_info (str) – How user found out.
- resume (str) – Resume link or text.

Returns

SQLAlchemy questionary model instance.

Return type

Questionary

id

user_id

full_name

degree

course

program

email

vacancy

motivation

plans

strengths

career_goals

team_motivation

role_in_team

events

found_info

resume

class database.models.RefGiveAway(**kwargs)

Bases: Base

RefGiveAway model for referrals in giveaways.

Parameters

- user_id (BigInteger) – User ID.
- event_name (str) – Event name.
- host_id (BigInteger) – Host user ID.

Returns

SQLAlchemy referral model instance.

Return type

RefGiveAway

id

user_id

event_name

host_id

class database.models.RegEvent(**kwargs)

Bases: Base

RegEvent model for generic event registrations.

Parameters

- id (BigInteger) – Primary key.
- name (str) – First name.
- surname (str) – Surname.
- fathername (str) – Middle name.
- mail (str) – Email address.
- phone (str) – Phone number.
- org (str) – Organization name.

Returns

SQLAlchemy registration model instance.

Return type

RegEvent

id

name

surname

fathername

mail

phone

org

class database.models.User(**kwargs)

Bases: Base

User model representing a bot user.

Parameters

- id (BigInteger) – Primary key.
- handler (str) – User's handler.
- is_superuser (bool) – Indicates superuser status.
- event_cnt (int) – Number of club_events the user has.
- strick (int) – User's streak.
- first_contact (str) – First contact value.
- money (int) – Amount of money.
- ref_cnt (int) – Referral count.

Returns

SQLAlchemy user model instance.

Return type

User

id

handler

is_superuser

event_cnt

strick

first_contact

money

ref_cnt

class database.models.UserXEvent(**kwargs)

Bases: Base

UserXEvent model representing user registration to an event.

Parameters

- user_id (BigInteger) – Foreign key to user.
- event_name (str) – Foreign key to event.
- status (str) – Registration status.
- first_contact (str) – First contact value.

Returns
SQLAlchemy user-event relation model.

Return type
UserXEvent

id

user_id

event_name

status

first_contact

class database.models.Vacancy(**kwargs)

Bases: Base

Vacancy model representing available positions.

Parameters
name (str) – Name of the vacancy.

Returns
SQLAlchemy vacancy model instance.

Return type
Vacancy

name

CHAPTER

THREE

ERRORS

3.1 errors

Custom Errors Custom exceptions for database and event/vacancy validations.

class errors.errors.CustomError

Bases: Exception

Base class for custom exceptions.

Parameters
message (str) – Error message.

class errors.errors.DatabaseConnectionError(message: str = 'Error with connection to db')

Bases: CustomError

Exception raised for database connection errors.

Parameters
message (str) – Description of the connection error.

class errors.errors.Error404(message: str = 'Error with status code 404')

Bases: CustomError

Exception raised for HTTP 404 not found errors.

Parameters
message (str) – Description of the 404 error.

class errors.errors.Error409(message: str = 'Error with status code 409')

Bases: CustomError

Exception raised for HTTP 409 conflict errors.

Parameters
message (str) – Description of the 409 error.

class errors.errors.EventNameError

Bases: CustomError

Exception raised when an event name already exists.

Parameters
message (str) – Description of the event name error.

class errors.errors.VacancyNameError

Bases: CustomError

Exception raised when a vacancy name already exists.

Parameters
message (str) – Description of the vacancy name error.

3.2 handlers

Error Decorators Wrappers for database and statistic error handling.

errors.handlers.db_error_handler(func)

Decorator to handle database related exceptions.

Parameters
func (Callable) – Asynchronous function to wrap.

Returns
Wrapped function that returns original result or None.

Return type
Callable

errors.handlers.stat_error_handler(func)

Decorator to handle statistic generation exceptions.

Parameters
func (Callable) – Asynchronous function to wrap.

Returns
Wrapped function that notifies user on error.

Return type
Callable

CHAPTER

FOUR

HANDLERS

4.1 admin

Admin Routers

List of all routers used in admin panel routing.

4.1.1 club_events

Admin Routers

List of all routers used in admin panel routing.

events

Event Management Bot Handles event creation, linking, and participant selection in Telegram.

async handlers.admin.club_events.events.cmd_add_event(message: Message, state: FSMContext)

Starts the event creation process.

Parameters

- message (Message) – The command message.
- state (FSMContext) – The state context.

Returns

None

async handlers.admin.club_events.events.add_event_part_2(message: Message, state: FSMContext)

Saves event description and requests event date.

Parameters

- message (Message) – The user message.
- state (FSMContext) – The state context.

Returns

None

async handlers.admin.club_events.events.add_event_part_3(message: Message, state: FSMContext)

Saves event date, creates event, and requests time.

Parameters

- message (Message) – The user message.
- state (FSMContext) – The state context.

Returns
: None

async handlers.admin.club_events.events.add_event_part_4(message: Message, state: FSMContext)
: Saves event time and requests place.

  Parameters
  - message (Message) – The user message.
  - state (FSMContext) – The state context.

  Returns
  : None

async handlers.admin.club_events.events.add_event_part_5(message: Message, state: FSMContext)
: Saves event place and requests link count.

  Parameters
  - message (Message) – The user message.
  - state (FSMContext) – The state context.

  Returns
  : None

async handlers.admin.club_events.events.add_event_part_6(message: Message, state: FSMContext)
: Generates registration and confirmation links.

  Parameters
  - message (Message) – The user message.
  - state (FSMContext) – The state context.

  Returns
  : None

async handlers.admin.club_events.events.cmd_end_event(message: Message, state: FSMContext)
: Begins event completion process.

  Parameters
  - message (Message) – The command message.
  - state (FSMContext) – The state context.

  Returns
  : None

async handlers.admin.club_events.events.process_end_event(message: Message, state: FSMContext)
: Selects a winner and starts verification.

  Parameters
  - message (Message) – The user message.
  - state (FSMContext) – The state context.

  Returns
  : None

async handlers.admin.club_events.events.reroll_end_event(callback: CallbackQuery, state: FSMContext)

Selects another winner if previous is invalid.

Parameters

- callback (CallbackQuery) – The callback query.
- state (FSMContext) – The state context.

Returns

None

async handlers.admin.club_events.events.confirm_end_event(callback: CallbackQuery, state: FSMContext)

Confirms winner and notifies participants.

Parameters

- callback (CallbackQuery) – The callback query.
- state (FSMContext) – The state context.

Returns

None

async handlers.admin.club_events.events.get_link(message: Message, state: FSMContext)

Starts process to generate new registration links.

Parameters

- message (Message) – The command message.
- state (FSMContext) – The state context.

Returns

None

async handlers.admin.club_events.events.make_link_05(message: Message, state: FSMContext)

Saves selected event name and requests link count.

Parameters

- message (Message) – The user message.
- state (FSMContext) – The state context.

Returns

None

async handlers.admin.club_events.events.make_link(message: Message, state: FSMContext)

Generates and sends registration and confirmation links.

Parameters

- message (Message) – The user message.
- state (FSMContext) – The state context.

Returns

None

face_control

Face Control Management Bot handlers for assigning, removing, and listing face control users.

async handlers.admin.club_events.face_control.cmd_face_control(message: Message)

Show face control management menu.

Parameters
message (Message) – The incoming command message.

Returns
None

async handlers.admin.club_events.face_control.face_control_menu(callback: CallbackQuery)

Show face control management menu via callback.

Parameters
callback (CallbackQuery) – The callback query.

Returns
None

async handlers.admin.club_events.face_control.face_control_add(callback: CallbackQuery, state: FSMContext)

Start process of adding a face control user.

Parameters

- callback (CallbackQuery) – The callback query.
- state (FSMContext) – The FSM context.

Returns
None

async handlers.admin.club_events.face_control.face_control_remove(callback: CallbackQuery)

Show list of face control users to remove.

Parameters
callback (CallbackQuery) – The callback query.

Returns
None

async handlers.admin.club_events.face_control.face_control_list(callback: CallbackQuery)

List all face control users.

Parameters
callback (CallbackQuery) – The callback query.

Returns
None

async handlers.admin.club_events.face_control.face_control_remove_confirm(callback: CallbackQuery, state: FSMContext)

Confirm removal of face control user.

Parameters

- callback (CallbackQuery) – The callback query.
- state (FSMContext) – The FSM context.

Returns
None

async handlers.admin.club_events.face_control.face_control_remove_execute(callback: CallbackQuery)

Execute removal of face control user.

Parameters
callback (CallbackQuery) – The callback query.

Returns
None

async handlers.admin.club_events.face_control.face_control_cancel_remove(callback: CallbackQuery)

Cancel removal of face control user.

Parameters
callback (CallbackQuery) – The callback query.

Returns
None

async handlers.admin.club_events.face_control.face_control_add_process(message: Message, state: FSMContext)

Process adding a new face control user.

Parameters

- message (Message) – The user message with ID.
- state (FSMContext) – The FSM context.

Returns
None

give_away

Giveaway and Networking Handlers for creating giveaways and assigning themes in networking club_events.

async handlers.admin.club_events.give_away.cmd_create_give_away(message: Message, state: FSMContext)

Start giveaway creation process by selecting an event.

Parameters

- message (Message) – Incoming command message.
- state (FSMContext) – FSM state context.

Returns
None

async handlers.admin.club_events.give_away.cmd_create_give_away2(message: Message, state: FSMContext)

Ask for organization name after event is selected.

Parameters

- message (Message) – User message with event name.
- state (FSMContext) – FSM state context.

Returns
None

async handlers.admin.club_events.give_away.cmd_create_give_away3(message: Message, state: FSMContext)

Ask for host ID after organization name.

Parameters

- message (Message) – User message with organization name.
- state (FSMContext) – FSM state context.

Returns
None

async handlers.admin.club_events.give_away.cmd_create_give_away4(message: Message, state: FSMContext)

Finalize giveaway by creating the host.

Parameters

- message (Message) – User message with ID or "я".
- state (FSMContext) – FSM state context.

Returns
None

async handlers.admin.club_events.give_away.give_colors(message: Message)

Assign random themes to networking participants.

Parameters
message (Message) – Incoming command message.

Returns
None

posts

Send Post Router Handles admin post distribution to Telegram users.

async handlers.admin.club_events.posts.cmd_send_post(message: Message)

Entry point to start the post sending process.

Parameters
message (Message) – Incoming message from admin.

Returns
None

async handlers.admin.club_events.posts.cancel(callback: CallbackQuery, state: FSMContext)

Cancels current FSM context and deletes previous messages.

Parameters

- callback (CallbackQuery) – Callback query from user.
- state (FSMContext) – Current FSM context.

Returns
None

async handlers.admin.club_events.posts.choose_event(callback: CallbackQuery, state: FSMContext)

Starts process of choosing an event for unregistered users.

Parameters

- callback (CallbackQuery) – Callback query from user.
- state (FSMContext) – FSM context to store selected event.

Returns

None

async handlers.admin.club_events.posts.mb_add_link_unreg(message: Message, state: FSMContext)

Asks whether to add a link to the post.

Parameters

- message (Message) – Incoming message with event name.
- state (FSMContext) – FSM context to store event name.

Returns

None

async handlers.admin.club_events.posts.link_no_unreg(callback: CallbackQuery, state: FSMContext)

Continues posting without link.

Parameters

- callback (CallbackQuery) – Callback query from user.
- state (FSMContext) – FSM context to update flag.

Returns

None

async handlers.admin.club_events.posts.link_yes_unreg(callback: CallbackQuery, state: FSMContext)

Prompts user to send a link.

Parameters

- callback (CallbackQuery) – Callback query from user.
- state (FSMContext) – FSM context to transition.

Returns

None

async handlers.admin.club_events.posts.process_post_to_all_media_unreg(message: Message, state: FSMContext)

Processes and sends post to unregistered users.

Parameters

- message (Message) – Message containing post content.
- state (FSMContext) – FSM context with event and post data.

Returns

None

async handlers.admin.club_events.posts.mb_add_link(callback: CallbackQuery, state: FSMContext)

Asks whether to add a link before posting to all users.

Parameters

- callback (CallbackQuery) – Callback query from admin.
- state (FSMContext) – FSM context to store user choice.

Returns
None

async handlers.admin.club_events.posts.link_no(callback: CallbackQuery, state: FSMContext)

Continues posting without a link.

Parameters

- callback (CallbackQuery) – User interaction callback.
- state (FSMContext) – State machine context.

Returns
None

async handlers.admin.club_events.posts.link_yes(callback: CallbackQuery, state: FSMContext)

Starts flow to add a link button.

Parameters

- callback (CallbackQuery) – User interaction callback.
- state (FSMContext) – State machine context.

Returns
None

async handlers.admin.club_events.posts.process_post_to_all_media(message: Message, state: FSMContext)

Sends the prepared post to all users.

Parameters

- message (Message) – Message with content.
- state (FSMContext) – Context with link flag and data.

Returns
None

async handlers.admin.club_events.posts.cmd_post_to_all(message: Message, state: FSMContext)

Receives button text and moves to media collection.

Parameters

- message (Message) – Message with button text.
- state (FSMContext) – Context to update button label.

Returns
None

async handlers.admin.club_events.posts.process_post_to_all(message: Message, state: FSMContext)

Sends post with or without button to all users.

Parameters

- message (Message) – Post content.
- state (FSMContext) – FSM state with post options.

Returns
None

async handlers.admin.club_events.posts.process_post_to_ev_media(message: Message, state: FSMContext)

Sends a post to users registered in a specific event.

Parameters

- message (Message) – Incoming post message.
- state (FSMContext) – FSM context with event data.

Returns
None

async handlers.admin.club_events.posts.cmd_post_to_ev(callback: CallbackQuery, state: FSMContext)

Starts flow to send a post with feedback form link.

Parameters

- callback (CallbackQuery) – Callback from admin.
- state (FSMContext) – FSM context for storing event.

Returns
None

async handlers.admin.club_events.posts.pre_process_post_to_ev(message: Message, state: FSMContext)

Receives event name and requests Google Form link.

Parameters

- message (Message) – Message with event name.
- state (FSMContext) – FSM context to store event.

Returns
None

async handlers.admin.club_events.posts.process_post_to_wth_op_to_ev(message: Message, state: FSMContext)

Sends feedback form link to users in selected event.

Parameters

- message (Message) – Message containing Google Form link.
- state (FSMContext) – FSM context containing event.

Returns
None

winner

Giveaway Result Retrieval Handles additional giveaway winner selection for club_events.

async handlers.admin.club_events.winner.cmd_get_result(message: Message, state: FSMContext)

Starts the giveaway result retrieval process.

Parameters

- message (Message) – Admin command message.
- state (FSMContext) – FSM context for state transitions.

Returns
None

async handlers.admin.club_events.winner.get_result(message: Message, state: FSMContext)

Requests organizer name after receiving the event name.

Parameters

- message (Message) – Message with selected event.
- state (FSMContext) – FSM context to store event name.

Returns
None

async handlers.admin.club_events.winner.get_result2(message: Message, state: FSMContext)

Retrieves and sends the winner of the selected giveaway.

Parameters

- message (Message) – Message with organizer name.
- state (FSMContext) – FSM context with event data.

Returns
None

4.1.2 statistics

base

Statistics Handlers

Handlers for sending various types of statistics to superusers.

async handlers.admin.statistics.base.cmd_send_stat(message: Message)

Send the initial statistics selection menu to the superuser.

Parameters
message (Message) – Incoming message from the user.

Returns
None

async handlers.admin.statistics.base.cmd_stat_all(callback: CallbackQuery)

Send overall statistics to the user.

Parameters
callback (CallbackQuery) – Incoming callback from button press.

Returns
None

async handlers.admin.statistics.base.process_post_to_all(message: Message, state: FSMContext)

Handle event selection for statistics.

Parameters

- message (Message) – Incoming message with event name.
- state (FSMContext) – Current FSM context.

Returns
None

async handlers.admin.statistics.base.cmd_stat_ev(callback: CallbackQuery)

Send statistics on questions.

Parameters
callback (CallbackQuery) – Incoming callback from button press.

Returns
None

async handlers.admin.statistics.base.cmd_stat_give_away(callback: CallbackQuery, state: FSMContext)

Start additional giveaway statistics process.

Parameters

- callback (CallbackQuery) – Incoming callback from button press.
- state (FSMContext) – Current FSM context.

Returns
None

async handlers.admin.statistics.base.cmd_stat_give_away2(message: Message, state: FSMContext)

Handle event name for giveaway statistics.

Parameters

- message (Message) – Message containing event name or 'quit'.
- state (FSMContext) – Current FSM context.

Returns
None

async handlers.admin.statistics.base.cmd_stat_give_away3(message: Message, state: FSMContext)

Handle organizer ID for giveaway statistics.

Parameters

- message (Message) – Message containing user ID or 'quit'.
- state (FSMContext) – Current FSM context.

Returns
None

async handlers.admin.statistics.base.cmd_stat_reg(callback: CallbackQuery, state: FSMContext)

Start process for registration statistics.

Parameters

- callback (CallbackQuery) – Incoming callback from button press.
- state (FSMContext) – Current FSM context.

Returns
None

async handlers.admin.statistics.base.cmd_stat_reg2(message: Message, state: FSMContext)

Handle event name for registration statistics.

Parameters

- message (Message) – Message with event name.
- state (FSMContext) – Current FSM context.

Returns
None

utils

Statistics Handlers Excel export of user statistics for bot.

async handlers.admin.statistics.utils.get_stat_all(user_id: int) → None

Retrieve all users and send statistics as Excel.

Parameters
user_id (int) – Telegram user ID to send report.

Returns
None

async handlers.admin.statistics.utils.get_stat_all_in_ev(user_id: int, event_name: str) → None

Retrieve users in event and send Excel plus summary.

Parameters

- user_id (int) – Telegram user ID to send report.
- event_name (str) – Event identifier.

Returns
None

async handlers.admin.statistics.utils.get_stat_quest(user_id: int) → None

Retrieve questionnaire submissions and send as Excel.

Parameters
user_id (int) – Telegram user ID to send report.

Returns
None

async handlers.admin.statistics.utils.get_stat_ad_give_away(user_id: int, host_id: int, event_name: str) → None

Retrieve giveaway participants and send as Excel.

Parameters

- user_id (int) – Telegram user ID to send report.
- host_id (int) – Host identifier.
- event_name (str) – Event identifier.

Returns
None

async handlers.admin.statistics.utils.get_stat_reg_out(user_id: int, event_name: str) → None

Retrieve external registrations and send as Excel.

Parameters

- user_id (int) – Telegram user ID to send report.
- event_name (str) – Event identifier.

Returns
None

async handlers.admin.statistics.utils.get_stat_reg(user_id: int, event_name: str) → None

Retrieve registration statistics and send Excel plus summary.

Parameters

- user_id (int) – Telegram user ID to send report.
- event_name (str) – Event identifier.

Returns

None

4.1.3 vacancies

Vacancy Management Handles creation, viewing and deletion of vacancies.

async handlers.admin.vacancies.cmd_all_vacancies(message: Message)

Sends a list of all active vacancies to the admin.

Parameters

message (Message) – Incoming command message.

Returns

None

async handlers.admin.vacancies.cmd_add_vacancy(message: Message, state: FSMContext)

Starts the process of adding a new vacancy.

Parameters

- message (Message) – Incoming command message.
- state (FSMContext) – FSM context to track input.

Returns

None

async handlers.admin.vacancies.process_vacancy_name(message: Message, state: FSMContext)

Saves the new vacancy or handles duplicates.

Parameters

- message (Message) – Message with vacancy name.
- state (FSMContext) – FSM context for state cleanup.

Returns

None

async handlers.admin.vacancies.cmd_dell_vacancy(message: Message, state: FSMContext)

Starts the process to delete a vacancy.

Parameters

- message (Message) – Incoming command message.
- state (FSMContext) – FSM context to store deletion state.

Returns

None

async handlers.admin.vacancies.process_vacancy_name_to_delete(message: Message, state: FSMContext)

Deletes the specified vacancy.

Parameters

- message (Message) – Message with the vacancy name.
- state (FSMContext) – FSM context for cleanup.

Returns
None

4.2 public

Public Routers List Imports and aggregates all public routers for registration.

4.2.1 club_events

Public Routers List Imports and aggregates all public routers for registration.

face_control

QR Verification Handler Handles QR verification and user admission to club_events.

async handlers.public.club_events.face_control.process_verification(callback: CallbackQuery)

Handle QR code verification by admin.

Parameters
callback (CallbackQuery) – Callback query triggered by admin verification.

Returns
None

qr

QR Code Handlers Handlers for verifying and retrieving QR codes for club_events.

async handlers.public.club_events.qr.cmd_check_qr(message: Message, command: CommandObject)

Handle QR code verification via command.

Parameters

- message (Message) – Incoming Telegram message object.
- command (CommandObject) – Parsed command with arguments.

Returns
None

async handlers.public.club_events.qr.cmd_my_qr(message: Message)

Handle /my_qr command to get QR code for event registration.

Parameters
message (Message) – Incoming Telegram message object.

Returns
None

async handlers.public.club_events.qr.handle_qr_button(message: Message)

Handle QR code button press.

Parameters
message (Message) – Incoming Telegram message object.

Returns
None

async handlers.public.club_events.qr.handle_qr_request(message: Message)

Common handler for QR code requests from both command and button.

Parameters
message (Message) – Incoming Telegram message object.

Returns
None

async handlers.public.club_events.qr.process_qr_event_selection(callback: CallbackQuery)

Handle event selection for QR code generation.

Parameters
callback (CallbackQuery) – Telegram callback with event selection.

Returns
None

async handlers.public.club_events.qr.process_reg_yes(callback: CallbackQuery, state: FSMContext)

Handle event registration confirmation.

Parameters

- callback (CallbackQuery) – User's confirmation callback.
- state (FSMContext) – FSM context with event data.

Returns
None

async handlers.public.club_events.qr.send_event_qr_code(user_id: int, event_name: str, message: Message | CallbackQuery, state: FSMContext)

Send QR code for event registration.

Parameters

- user_id (int) – Telegram user ID.
- event_name (str) – Name of the event.
- message (Union[Message, CallbackQuery]) – Incoming message or callback.
- state (FSMContext) – FSM context.

Returns
None

ref

User commands and profile Handles info, profile, top and referral link generation.

async handlers.public.club_events.ref.get_ref_v2_part1(message: Message)

Start referral link generation for user.

Parameters
message (Message) – Incoming user message.

Returns
None

async handlers.public.club_events.ref.get_ref_v2_part2(callback: CallbackQuery)

Handle event selection for referral link generation.

Parameters
: callback (CallbackQuery) – Callback query from Telegram.

Returns
: None

registration

Start command handler Handles /start command and hash-based user scenarios.

async handlers.public.club_events.registration.handle_networking_hash(user_id: int, username: str, message: Message)

Handle the networking registration hash.

Parameters

- user_id (int) – Telegram user ID.
- username (str) – Telegram username.
- message (Message) – Incoming message object.

Returns
: None

async handlers.public.club_events.registration.handle_reg_hash(user_id: int, username: str, hash_value: str, message: Message, state: FSMContext)

Handle registration hash for event sign-up.

Parameters

- user_id (int) – Telegram user ID.
- username (str) – Telegram username.
- hash_value (str) – Registration hash string.
- message (Message) – Incoming message object.
- state (FSMContext) – FSM context for user.

Returns
: None

async handlers.public.club_events.registration.handle_ref_hash(user_id: int, username: str, hash_value: str, message: Message, state: FSMContext)

Handle referral hash and event registration.

Parameters

- user_id (int) – Telegram user ID.
- username (str) – Telegram username.
- hash_value (str) – Referral hash string.
- message (Message) – Incoming message object.
- state (FSMContext) – FSM context for user.

Returns
None

async handlers.public.club_events.registration.handle_otbor_hash(user_id: int, username: str, message: Message)

Handle 'otbor' hash.

Parameters

- user_id (int) – Telegram user ID.
- username (str) – Telegram username.
- message (Message) – Incoming message object.

Returns
None

async handlers.public.club_events.registration.handle_default_hash(user_id: int, username: str, hash_value: str, message: Message)

Handle unknown/default hash behavior.

Parameters

- user_id (int) – Telegram user ID.
- username (str) – Telegram username.
- hash_value (str) – Default hash string.
- message (Message) – Incoming message object.

Returns
None

async handlers.public.club_events.registration.reg_event_part0_5(callback: CallbackQuery, state: FSMContext)

Handle user declining event registration.

Parameters

- callback (CallbackQuery) – Callback query from Telegram.
- state (FSMContext) – FSM context for user.

Returns
None

async handlers.public.club_events.registration.reg_event_part1(callback: CallbackQuery, state: FSMContext)

Ask user if they are an HSE student/employee.

Parameters

- callback (CallbackQuery) – Callback query from Telegram.
- state (FSMContext) – FSM context for user.

Returns
None

async handlers.public.club_events.registration.reg_event_part1_5(callback: CallbackQuery, state: FSMContext)

Process HSE student/employee event QR code.

Parameters

- callback (CallbackQuery) – Callback query from Telegram.
- state (FSMContext) – FSM context for user.

Returns
None

async handlers.public.club_events.registration.reg_event_part2(callback: CallbackQuery, state: FSMContext)

Handle registration for non-HSE users.

Parameters

- callback (CallbackQuery) – Callback query from Telegram.
- state (FSMContext) – FSM context for user.

Returns
None

async handlers.public.club_events.registration.reg_event_part3(message: Message, state: FSMContext)

Store user name and ask for surname.

Parameters

- message (Message) – Incoming user message.
- state (FSMContext) – FSM context for user.

Returns
None

async handlers.public.club_events.registration.reg_event_part4(message: Message, state: FSMContext)

Store user surname and ask for fathername.

Parameters

- message (Message) – Incoming user message.
- state (FSMContext) – FSM context for user.

Returns
None

async handlers.public.club_events.registration.reg_event_part5(message: Message, state: FSMContext)

Store user fathername and ask for phone.

Parameters

- message (Message) – Incoming user message.
- state (FSMContext) – FSM context for user.

Returns
None

async handlers.public.club_events.registration.reg_event_part6(message: Message, state: FSMContext)

Store user phone and ask for email.

Parameters

- message (Message) – Incoming user message.
- state (FSMContext) – FSM context for user.

Returns
None

async handlers.public.club_events.registration.reg_event_part7(message: Message, state: FSMContext)

Store user email and ask for organization.

Parameters

- message (Message) – Incoming user message.
- state (FSMContext) – FSM context for user.

Returns
None

async handlers.public.club_events.registration.reg_event_part8(message: Message, state: FSMContext)

Store user organization and finish registration.

Parameters

- message (Message) – Incoming user message.
- state (FSMContext) – FSM context for user.

Returns
None

4.2.2 utils

base

Handlers utils

async handlers.public.utils.base.create_user_if_not_exists(user_id: int, username: str, first_contact: str = None) → str

Create a user if not exists.

Parameters

- user_id (int) – Telegram user ID.
- username (str) – Telegram username.
- first_contact (str, optional) – Referrer or first hash source.

Returns
User status or existing user object.

Return type
str

4.2.3 quest

Questionnaire Router Handlers for questionnaire FSM and callbacks.

async handlers.public.quest.start2(message: Message)

Redirect text trigger to /quest command.

Parameters
message (types.Message) – Incoming message instance.

async handlers.public.quest.start(message: Message)
Send introduction message and first keyboard.

Parameters
message (types.Message) – Incoming message instance.

async handlers.public.quest.start_2(callback: CallbackQuery)
Send details message and second keyboard on callback.

Parameters
callback (CallbackQuery) – Incoming callback query.

async handlers.public.quest.start_nu(message: Message, state: FSMContext)
Initialize or continue questionnaire based on state.

Parameters

- message (types.Message) – Incoming message or callback.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.start_first_part(message: Message, state: FSMContext)
Begin first part: request full name.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.enter_full_name(message: Message, state: FSMContext)
Handle full name input and proceed to degree.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.enter_degree(message: Message, state: FSMContext)
Handle degree input and proceed to course.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.enter_course(message: Message, state: FSMContext)
Handle course input and proceed to program.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.enter_program(message: Message, state: FSMContext)
Handle program input and proceed to email.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.enter_email(message: Message, state: FSMContext)

Handle email input and request vacancy selection.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.enter_vacancy(message: Message, state: FSMContext)

Handle vacancy selection and ask for another.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.ask_another_vacancy(callback: CallbackQuery, state: FSMContext)

Handle another vacancy choice or continue questionnaire.

Parameters

- callback (CallbackQuery) – Incoming callback query.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.continue_from_second_part(message: Message, state: FSMContext)

Begin second part: ask motivation.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.enter_motivation(message: Message, state: FSMContext)

Handle motivation input and proceed to plans.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.enter_plans(message: Message, state: FSMContext)

Handle plans input and proceed to strengths.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

async handlers.public.quest.enter_strengths(message: Message, state: FSMContext)

Handle strengths input and proceed to career goals.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

Returns

None

async handlers.public.quest.enter_career_goals(message: Message, state: FSMContext)

Handle career goals input and proceed to team motivation.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

Returns
None

async handlers.public.quest.enter_team_motivation(message: Message, state: FSMContext)

Handle team motivation input and proceed to role in team.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

Returns
None

async handlers.public.quest.enter_role_in_team(message: Message, state: FSMContext)

Handle role in team input and proceed to club_events.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

Returns
None

async handlers.public.quest.enter_events(message: Message, state: FSMContext)

Handle club_events input and proceed to found info.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

Returns
None

async handlers.public.quest.enter_found_info(message: Message, state: FSMContext)

Handle found info input and proceed to resume.

Parameters

- message (types.Message) – Incoming message instance.
- state (FSMContext) – FSM context instance.

Returns
None

async handlers.public.quest.enter_resume(message: Message, state: FSMContext)

Handle resume input and end questionnaire.

Parameters

- message (types.Message) – Incoming message instance.

- state (FSMContext) – FSM context instance.

Returns
None

4.2.4 start

Start command handler Handles /start command and hash-based user scenarios.

async handlers.public.start.send_welcome_message(user_id: int, username: str, message: Message)

Send a personalized welcome message.

Parameters

- user_id (int) – Telegram user ID.
- username (str) – Telegram username.
- message (Message) – Incoming message object.

Returns
None

async handlers.public.start.handle_otbor_hash(user_id: int, username: str, message: Message)

Handle 'otbor' hash.

Parameters

- user_id (int) – Telegram user ID.
- username (str) – Telegram username.
- message (Message) – Incoming message object.

Returns
None

async handlers.public.start.cmd_start(message: Message, command: CommandObject, state: FSMContext)

Handle the /start command and route user by hash.

Parameters

- message (Message) – Incoming message object.
- command (CommandObject) – Parsed command data.
- state (FSMContext) – FSM context for user.

Returns
None

4.2.5 user

User commands and profile Handles info, profile, top and referral link generation.

async handlers.public.user.cmd_info(message: Message)

Show help info for user or admin.

Parameters
message (Message) – Incoming user message.

Returns
None

async handlers.public.user.cmd_profile(message: Message)

Display user's personal profile and stats.

Parameters

message (Message) – Incoming user message.

Returns

None

async handlers.public.user.top_inline(message: Message)

Handle inline top leaderboard request.

Parameters

message (Message) – Callback query object.

Returns

None

async handlers.public.user.cmd_top(message: Message)

Display the top-10 users leaderboard.

Parameters

message (Message) – Incoming user message.

Returns

None

4.3 utils

4.3.1 base

Telegram Errors Global error and safe message handlers for bot.

async handlers.utils.base.safe_send_message(bott: Bot, recipient, text: str, reply_markup=ReplyKeyboardMarkup(keyboard=[[KeyboardButton(text=′По
QR-код′, request_users=None, request_chat=None, request_contact=None, request_location=None, request_poll=None, web_app=None, request_user=None)]], is_persistent=None, resize_keyboard=True, one_time_keyboard=None, input_field_placeholder=None, selective=None), retry_attempts: int = 3, delay: int = 5) → Message | None

Send message with retry and error handling.

Parameters

- bott (Bot) – Bot instance to send message.
- recipient (Message|CallbackQuery|int) – Message recipient.
- text (str) – Message text.
- reply_markup – Reply keyboard markup.
- retry_attempts (int) – Max retry attempts.
- delay (int) – Delay between retries in seconds.

Returns

Sent message or None if failed.

Return type
: Message|None

async handlers.utils.base.make_short_link(url: str) → str | None
: Create short link using clck.ru service.

Parameters
: url (str) – URL to shorten.

Returns
: Short link or None if failed.

Return type
: str|None

async handlers.utils.base.get_bot_username()

4.3.2 qr_utils

QR Code Generator Stylish QR code with gradient background and central logo.

handlers.utils.qr_utils.create_styled_qr_code(data: str) → BytesIO
: Generate a stylish QR code with gradient background and embedded logo.

Parameters
: data (str) – Data to encode into the QR code.

Returns
: PNG image of the QR code in memory.

Return type
: BytesIO

4.4 error

Telegram Errors Global error and safe message handlers for bot.

async handlers.error.global_error_handler(event: Update, exception: Exception) → bool
: Handle global bot exceptions.

Parameters

- event (types.Update) – Incoming update instance.
- exception (Exception) – Raised exception.

Returns
: True if exception was handled.

Return type
: bool

4.5 states

Bot States. State machine definitions for various bot workflows and forms.

class handlers.states.EventCreateState
: Bases: StatesGroup

Event creation form states.

Parameters
None

Returns
None

waiting_event_name = <State 'EventCreateState:waiting_event_name'>

waiting_event_date = <State 'EventCreateState:waiting_event_date'>

waiting_event_time = <State 'EventCreateState:waiting_event_time'>

waiting_event_place = <State 'EventCreateState:waiting_event_place'>

waiting_links_count = <State 'EventCreateState:waiting_links_count'>

class handlers.states.EventReg

Bases: StatesGroup

Event registration form states.

Parameters
None

Returns
None

waiting_name = <State 'EventReg:waiting_name'>

waiting_surname = <State 'EventReg:waiting_surname'>

waiting_fathername = <State 'EventReg:waiting_fathername'>

waiting_mail = <State 'EventReg:waiting_mail'>

waiting_phone = <State 'EventReg:waiting_phone'>

waiting_org = <State 'EventReg:waiting_org'>

class handlers.states.EventState

Bases: StatesGroup

Event management states.

Parameters
None

Returns
None

waiting_ev = <State 'EventState:waiting_ev'>

waiting_ev_for_link = <State 'EventState:waiting_ev_for_link'>

waiting_links_count = <State 'EventState:waiting_links_count'>

class handlers.states.FaceControlState

Bases: StatesGroup

States for face control management.

waiting_user_id
Waiting for user ID to add/remove

waiting_confirmation
Waiting for confirmation to remove

class handlers.states.GiveAwayState
Bases: StatesGroup

Giveaway management states.

Parameters
None

Returns
None

waiting_event = <State 'GiveAwayState:waiting_event'>

waiting_org_name = <State 'GiveAwayState:waiting_org_name'>

waiting_id = <State 'GiveAwayState:waiting_id'>

class handlers.states.PostState
Bases: StatesGroup

Post creation and management states.

Parameters
None

Returns
None

waiting_for_post_to_all_text1 = <State 'PostState:waiting_for_post_to_all_text1'>

waiting_for_post_to_all_text05 = <State 'PostState:waiting_for_post_to_all_text05'>

waiting_for_post_to_all_text = <State 'PostState:waiting_for_post_to_all_text'>

waiting_for_post_to_ev_ev_unreg = <State 'PostState:waiting_for_post_to_ev_ev_unreg'>

waiting_for_post_to_all_text1_unreg = <State 'PostState:waiting_for_post_to_all_text1_unreg'>

waiting_for_post_to_all_text05_unreg = <State 'PostState:waiting_for_post_to_all_text05_unreg'>

waiting_for_post_to_all_text_unreg = <State 'PostState:waiting_for_post_to_all_text_unreg'>

waiting_for_post_to_ev_ev = <State 'PostState:waiting_for_post_to_ev_ev'>

waiting_for_post_to_ev_text = <State 'PostState:waiting_for_post_to_ev_text'>

waiting_for_post_wth_op_to_ev_ev = <State 'PostState:waiting_for_post_wth_op_to_ev_ev'>

waiting_for_post_wth_op_to_ev_text = <State 'PostState:waiting_for_post_wth_op_to_ev_text'>

waiting_for_post_to_all_media = <State 'PostState:waiting_for_post_to_all_media'>

waiting_for_post_to_ev_media = <State 'PostState:waiting_for_post_to_ev_media'>

waiting_for_post_to_all_media_unreg = <State 'PostState:waiting_for_post_to_all_media_unreg'>

class handlers.states.Questionnaire

Bases: StatesGroup

Questionnaire form states for user registration.

Parameters
None

Returns
None

full_name = <State 'Questionnaire:full_name'>

degree = <State 'Questionnaire:degree'>

course = <State 'Questionnaire:course'>

program = <State 'Questionnaire:program'>

email = <State 'Questionnaire:email'>

vacancy = <State 'Questionnaire:vacancy'>

motivation = <State 'Questionnaire:motivation'>

plans = <State 'Questionnaire:plans'>

strengths = <State 'Questionnaire:strengths'>

career_goals = <State 'Questionnaire:career_goals'>

team_motivation = <State 'Questionnaire:team_motivation'>

role_in_team = <State 'Questionnaire:role_in_team'>

events = <State 'Questionnaire:events'>

found_info = <State 'Questionnaire:found_info'>

resume = <State 'Questionnaire:resume'>

another_vacancy = <State 'Questionnaire:another_vacancy'>

class handlers.states.StatState

Bases: StatesGroup

Statistics management states.

Parameters
None

Returns
None

waiting_for_ev = <State 'StatState:waiting_for_ev'>

waiting_for_give_away_ev = <State 'StatState:waiting_for_give_away_ev'>

waiting_user_id = <State 'StatState:waiting_user_id'>

waiting_for_ev1 = <State 'StatState:waiting_for_ev1'>

waiting_for_ev2 = <State 'StatState:waiting_for_ev2'>

class handlers.states.VacancyState

Bases: StatesGroup

Vacancy management states.

Parameters
None

Returns
None

waiting_for_vacancy_name = <State 'VacancyState:waiting_for_vacancy_name'>

waiting_for_vacancy_name_to_delete = <State 'VacancyState:waiting_for_vacancy_name_to_delete'>

class handlers.states.WinnerState

Bases: StatesGroup

Winner selection states.

Parameters
None

Returns
None

wait_give_away_event = <State 'WinnerState:wait_give_away_event'>

wait_give_away_id = <State 'WinnerState:wait_give_away_id'>

CHAPTER

FIVE

KEYBOARDS

Keyboard layouts package.

5.1 club_events

5.1.1 admin

Admin-related keyboard layouts.

admin_keyboards

Admin Keyboards Inline and reply keyboards for admin-related actions.

keyboards.club_events.admin.admin_keyboards.post_target() → InlineKeyboardMarkup

Create inline keyboard for post targeting options.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.club_events.admin.admin_keyboards.post_ev_target(events: list[str]) → ReplyKeyboardMarkup

Create reply keyboard for selecting event target.

Parameters
events (list[str]) – List of event identifiers.

Returns
Reply keyboard markup.

Return type
ReplyKeyboardMarkup

keyboards.club_events.admin.admin_keyboards.stat_target() → InlineKeyboardMarkup

Create inline keyboard for statistics targets.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.club_events.admin.admin_keyboards.apply_winner() → InlineKeyboardMarkup

Create inline keyboard for winner confirmation.

Returns
: Inline keyboard markup.

Return type
: InlineKeyboardMarkup

keyboards.club_events.admin.admin_keyboards.top_ikb() → InlineKeyboardMarkup

Create inline keyboard for top users command.

Returns
: Inline keyboard markup.

Return type
: InlineKeyboardMarkup

5.1.2 common

Common keyboard layouts.

common_keyboards

Common Keyboards Reusable inline and reply keyboard layouts for user interaction.

keyboards.club_events.common.common_keyboards.link_ikb(text: str, url: str) → InlineKeyboardMarkup

Create inline keyboard with a URL button.

Parameters

- text (str) – Button text.
- url (str) – Link URL.

Returns
: Inline keyboard markup.

Return type
: InlineKeyboardMarkup

keyboards.club_events.common.common_keyboards.yes_no_ikb() → InlineKeyboardMarkup

Create inline yes/no keyboard.

Returns
: Inline keyboard markup.

Return type
: InlineKeyboardMarkup

keyboards.club_events.common.common_keyboards.yes_no_hse_ikb() → InlineKeyboardMarkup

Create HSE-specific yes/no keyboard.

Returns
: Inline keyboard markup.

Return type
: InlineKeyboardMarkup

keyboards.club_events.common.common_keyboards.yes_no_link_ikb() → InlineKeyboardMarkup

Create yes/no/cancel keyboard with link prompt.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.club_events.common.common_keyboards.unreg_yes_no_link_ikb() → InlineKeyboardMarkup

Create unregistered yes/no/cancel keyboard.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

utils

Keyboard Utilities Helper functions for building keyboard layouts.

keyboards.club_events.common.utils.make_k_from_list(items: list[str]) → list[list[KeyboardButton]]

Build keyboard layout from list of strings.

Parameters
items (list[str]) – List of button labels.

Returns
2D list of KeyboardButton rows.

Return type
list[list[KeyboardButton]]

5.1.3 event

Event-related keyboard layouts.

event_keyboards

Event Keyboards Inline and reply keyboards related to event interactions.

keyboards.club_events.event.event_keyboards.vacancy_selection_keyboard(vacancies: list[str]) → ReplyKeyboardMarkup

Create reply keyboard for selecting vacancies.

Parameters
vacancies (list[str]) – Available vacancy names.

Returns
Reply keyboard markup.

Return type
ReplyKeyboardMarkup

keyboards.club_events.event.event_keyboards.another_vacancy_keyboard() → InlineKeyboardMarkup

Create inline keyboard to ask another vacancy selection.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.club_events.event.event_keyboards.events_ikb(events: list) → InlineKeyboardMarkup

Create inline keyboard from event objects.

Parameters
events (list) – List of event objects with desc and name.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.club_events.event.event_keyboards.get_ref_ikb(event_name: str) → InlineKeyboardMarkup

Create inline keyboard to get referral link.

Parameters
event_name (str) – Event identifier.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.club_events.event.event_keyboards.choose_event_for_qr(events: list) → InlineKeyboardMarkup

Create keyboard for choosing event to get QR code.

Parameters
events (list) – List of event objects.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

5.1.4 face_control

Face control-related keyboard layouts.

face_control_keyboards

Face control-related keyboard layouts.

keyboards.club_events.face_control.face_control_keyboards.face_checkout_kb(user_id: int, event_name: str) → InlineKeyboardMarkup

Create keyboard for face control checkout.

Parameters

- user_id (int) – User ID to verify.
- event_name (str) – Event name.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.club_events.face_control.face_control_keyboards.back_to_face_control() → InlineKeyboardMarkup

Create back button keyboard for face control.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.club_events.face_control.face_control_keyboards.face_control_menu_kb() → InlineKeyboardMarkup

Create main menu keyboard for face control.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.club_events.face_control.face_control_keyboards.face_controls_list(face_controls) → InlineKeyboardMarkup

Create keyboard with list of face control users.

Parameters
face_controls – List of face control objects.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.club_events.face_control.face_control_keyboards.yes_no_face(user_id: int) → InlineKeyboardMarkup

Create confirmation keyboard for face control actions.

Parameters
user_id (int) – User ID to confirm action for.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

5.2 quest

Quest-related keyboard layouts.

5.2.1 quest_keyboards

Quest Keyboards Inline keyboards for quest interaction steps.

keyboards.quest.quest_keyboards.quest_keyboard_1() → InlineKeyboardMarkup

Create inline keyboard for first quest step.

Returns
: Inline keyboard markup.

Return type
: InlineKeyboardMarkup

keyboards.quest.quest_keyboards.quest_keyboard_2() → InlineKeyboardMarkup

Create inline keyboard with external quest link.

Returns
: Inline keyboard markup.

Return type
: InlineKeyboardMarkup

5.3 base

Common Keyboards Reusable inline and reply keyboard layouts for user interaction.

keyboards.base.main_reply_keyboard() → ReplyKeyboardMarkup

Create single-button reply keyboard.

Returns
: Reply keyboard markup.

Return type
: ReplyKeyboardMarkup

keyboards.base.link_ikb(text: str, url: str) → InlineKeyboardMarkup

Create inline keyboard with a URL button.

Parameters

- text (str) – Button text.
- url (str) – Link URL.

Returns
: Inline keyboard markup.

Return type
: InlineKeyboardMarkup

keyboards.base.yes_no_ikb() → InlineKeyboardMarkup

Create inline yes/no keyboard.

Returns
: Inline keyboard markup.

Return type
: InlineKeyboardMarkup

keyboards.base.yes_no_hse_ikb() → InlineKeyboardMarkup

Create HSE-specific yes/no keyboard.

Returns
: Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.base.yes_no_link_ikb() → InlineKeyboardMarkup

Create yes/no/cancel keyboard with link prompt.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

keyboards.base.unreg_yes_no_link_ikb() → InlineKeyboardMarkup

Create unregistered yes/no/cancel keyboard.

Returns
Inline keyboard markup.

Return type
InlineKeyboardMarkup

CHAPTER

SIX

UTILS

6.1 logger

Logger configuration Initializes and returns configured logger instances.

utils.logger.get_logger(name: str) → Logger

Get a logger instance with the specified name.

Parameters
: name (str) – The name of the logger.

Returns
: The logger instance.

Return type
: logging.Logger

6.2 validators

Validation utilities Helpers for number and time format validation.

async utils.validators.is_number_in_range(s: str) → bool

Check if the given string can be converted to a float.

Parameters
: s (str) – Input string to validate.

Returns
: True if string represents a number, False otherwise.

Return type
: bool

utils.validators.is_valid_time_format(time_str: str) → bool

Validate if the time string is in correct HH:MM format.

Parameters
: time_str (str) – Time string to validate.

Returns
: True if time is in valid format, False otherwise.

Return type
: bool

CHAPTER SEVEN

MAIN

Telegram Bot Main. Main entry point for the Telegram bot application with router registration and startup logic.

main.register_routers() → None

Register all routers in the dispatcher.

Parameters
None

Returns
None

async main.main()

Start the bot and handle its lifecycle.

Parameters
None

Returns
None