Largo

Generated by Doxygen 1.12.0

1 Documentation for "Largo" Auction Backend

1.0.1 Overview

Auction platform backend built with Flask, Flask-SocketIO, SQLAlchemy, and MySQL. Provides REST API for auction management and real-time WebSocket updates for live bidding.

1.0.2 Features

1.0.2.1 Auction Management

- **Full Auction Lifecycle**: Create, list, bid, close auctions automatically
- **Bidding System**: Real-time bids with 1.0 minimum increment and overtime extension
- **Multi-Photo Support**: Main photo + gallery per auction with upload API
- **Categories**: Multiple categories per auction via junction table

1.0.2.2 User Management

- **Authentication**: JWT-based registration, login, password management
- **Seller Dashboard**: View own auctions (active/archived), user bids
- **Profile**: User info retrieval with account creation timestamps

1.0.2.3 Real-Time Features

- **Live Bidding**: SocketIO room per auction with bid updates
- **Auction Status**: Automatic opening/closing via scheduler
- **Presence**: Join/leave auction rooms with user notifications

1.0.2.4 Scheduling System

- **Smart Scheduler**: Background jobs for auction open/close timing
- **Overtime Handling**: Dynamic extension (60s added if bid within last minute)
- **Concurrency**: Per-auction locks prevent race conditions

1.0.3 Architecture

1.0.3.1 Database Models (`db_objects.py`)

Seven core models:

- **Users**: Authentication + profiles
- **Auctions**: Core auction data with status enum (at_auction/sold/not_issued)
- **AuctionPriceHistory**: Bid history with timestamps for conflict resolution
- **Categories**: Auction categories lookup
- **CategoriesAuction**: Many-to-many auction-category junction
- **PhotosItem**: Auction images with main photo flag

1.0.3.2 REST API Routes

Authentication**:

- `/register` - Create account + JWT
- `/login` - Authenticate + JWT

- `/change_password` - Update password
- `/get_user_info` - Profile data
  Auctions**:
- `/create_auction` - New auction with photos/categories
- `/get_all_auctions` - List all (optimized subqueries)
- `/get_auction_details` - Single auction full data
- `/get_user_own_auctions` - Seller's auctions
- `/get_user_auctions` - User's active bids
- `/archived_auctions` - Seller's ended auctions
- `/place_bid` - Submit bid (locked, validated)
- `/delete_auction` - Seller deletes own auction
  Categories**:
- `/get_all_categories` - Category listing
  File Upload**:
- `/api/upload_image` - Secure image upload (10MB, png/jpg/webp)
- `/uploads/<filename>` - Serve uploaded images

1.0.3.3 WebSocket Events

- `join` - Enter auction room
- `leave` - Exit auction room

- `auction_updated` - New bid broadcast (price, bidder, overtime)
- `auction_closed` - Auction ended notification (winner)

1.0.3.4 Application Factory (`create_app()`)

- **Config**: JWT, MySQLAlchemy, upload limits from .env
- **Extensions**: SocketIO, JWTManager, CORS, blueprints
- **Blueprints**: Auctions/Users/Uploads/Categories modular routing

1.0.4 Technology Stack

- **Flask 2.x**: Core framework
- **Flask-SocketIO**: Real-time bidding
- **SQLAlchemy + MySQL**: ORM/database
- **Flask-JWT-Extended**: Token authentication
- **APScheduler**: Auction lifecycle automation
- **Flask-CORS**: Frontend compatibility
- **Werkzeug**: File security utilities

1.0.5 Environment Variables

- **DB_USER, DB_PASSWORD, DB_HOST, DB_PORT, DB_NAME
- **JWT_SECRET_KEY
- **UPLOAD_DIRECTORY (default: 'uploads')

1.0.6 Getting Started

1. Configure `.env` with database/JWT settings
2. `pip install -r requirements.txt`
3. `flask run` or production WSGI server
4. Auctions auto-schedule via BackgroundScheduler

Authors

Paweł Dyczek, Paweł Herzyk, Mikołaj Całus

2 Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

3 Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

6 Namespace Documentation

6.1 app_state Namespace Reference

Variables

- socketio = SocketIO(cors_allowed_origins="∗")

6.1.1 Variable Documentation

socketio

```
app_state.socketio = SocketIO(cors_allowed_origins="*")
```

6.2 Auctions Namespace Reference

Functions

- get_all_auctions ()
- get_auction_details ()
- create_auction ()
- place_bid ()
- get_user_own_auctions ()
- get_user_auctions ()
- archived_auctions ()
- delete_auction ()

Variables

- bp = Blueprint('auctions', __name__, url_prefix='/api')

6.2.1 Function Documentation

archived_auctions()

```
Auctions.archived_auctions ()
```

```
@brief Retrieves archived (ended) auctions owned by the authenticated seller.

Filters seller's auctions where end_date + overtime < now(). Includes main photo,
highest bid (or start_price as final_price), winner details, and categories.

@return JSON array of archived auctions with: id_auction, description, starting_price,
final_price, winner_id, winner_name, end_date, title, main_photo, categories.

@retval 200 Success: List of ended auctions (may be empty).
@retval 404 User not found.
```

create_auction()

```
Auctions.create_auction ()
```

```
@brief Creates a new auction for the authenticated seller with photos and categories.

Validates required fields, sets status based on start_date vs now(). Adds auction,
bulk-inserts PhotosItem (with main flag) and CategoriesAuction links post-flush.

@param title String: Auction title (required).
@param description String: Description (required).
@param start_price Float: Starting price (required).
@param start_date ISO datetime: Auction start (required).
@param end_date ISO datetime: Auction end (required).
@param photos Array optional: [{"url": str, "is_main": bool}].
@param categories Array optional: [category IDs].

@return JSON with new auction ID.

@retval 201 Success: {"message": "...", "id_auction": ID}
@retval 400 Missing required fields
@retval 404 User not found
```

delete_auction()

```
Auctions.delete_auction ()
```

```
@brief Deletes an auction. Only the seller can delete an auction.

@detail Requires JWT authentication. Parses id_auction from JSON body, verifies user ownership, deletes associ

@param id_auction: The id of the auction to be deleted.

@return A JSON object with a message indicating whether the auction was deleted successfully.

@retval 200: If the auction was deleted successfully.
@retval 400: If the id_auction parameter is missing.
@retval 404: If the user or the auction is not found.
@retval 403: If the user is not authorized to delete the auction.
```

get_all_auctions()

```
Auctions.get_all_auctions ()
```

```
@brief Retrieves all auctions with aggregated max prices and categories.

Complex query uses subqueries for max bid per auction (or start_price),
GROUP_CONCAT categories, LEFT JOIN main photo. Public endpoint for auction list.

@return JSON array of auctions: id_auction, title, description, id_seller,
start_price, current_price, dates, overtime, status, id_winner, main_photo, categories array.

@retval 200 Success: Complete auctions list
```

get_auction_details()

```
Auctions.get_auction_details ()
```

```
@brief Retrieves complete details for a specific auction by ID.

Fetches auction, seller/winner info, all photos (main separate, others in array),
highest bid/current price, categories. Public endpoint (no JWT required).

@param id_auction Query param: Integer auction ID (required).

@return Full auction JSON with seller/winner names, main_photo, photos array
(non-main), current_price, highest_bidder ID, categories.

@retval 200 Success: Detailed auction object
@retval 400 Missing id_auction
@retval 404 Auction not found
```

get_user_auctions()

```
Auctions.get_user_auctions ()
```

```
@brief Retrieves active auctions for the authenticated user.

Fetches user's bids from AuctionPriceHistory, gets distinct auctions not yet ended
(considering end_date + overtime). Enriches with main photo, highest bid (or start_price),
and joined categories for each.

@return JSON array of user auctions with fields: id_auction, description, starting_price,
current_price, end_date, overtime, title, main_photo, status, categories.

@retval 200 Success: List of active auctions (may be empty).
@retval 404 User not found.
```

get_user_own_auctions()

```
Auctions.get_user_own_auctions ()
```

```
@brief Retrieves all auctions owned by the authenticated seller.

Fetches seller's auctions with main photo, highest bid (or start_price),
categories, dates, status, and winner ID. No end-date filtering applied.

@return JSON array of seller auctions with: id_auction, title, description,
start_price, current_price, start_date, end_date, overtime, status, id_winner,
main_photo, categories.

@retval 200 Success: Complete list (may be empty).
@retval 404 User not found.
```

place_bid()

```
Auctions.place_bid ()
```

```
@brief Places a bid on an active auction with validation and overtime extension.

Validates bid > current + 1.0, not after end, not duplicate timestamp. Uses lock for concurrency.
Adds to AuctionPriceHistory, extends overtime by 60s if <60s remain. Emits SocketIO update.

@param id_auction JSON body: Integer auction ID (required).
@param new_price JSON body: Float bid amount > current + 1.0 (required).

@return JSON success or error message.

@retval 200 Success: {"message": "Bid placed successfully"}
@retval 400 Missing params, inactive auction, too low bid, ended, or duplicate timestamp
@retval 404 User or auction not found
```

6.2.2 Variable Documentation

bp

```
Auctions.bp = Blueprint('auctions', __name__, url_prefix='/api')
```

6.3 auctions Namespace Reference

Functions

- handle_join (data)
- handle_leave (data)
- get_auction_lock (auction_id)
- get_next_auction_to_close ()
- get_next_auction_to_open ()
- close_auction_if_ended (auction_id, expected_overtime=0)
- open_auction (auction_id)
- schedule_auction_closure (auction)
- schedule_auction_opening (auction)
- schedule_next_auction ()
- schedule_open_next_auction ()
- start_scheduler (app)
- start_scheduler ()
- on_auction_update ()

Variables

- level
- logger = logging.getLogger("auctions_scheduler_test")
- _auction_locks = defaultdict(Lock)
- SCHEDULER = None
- APP = None

6.3.1 Function Documentation

close_auction_if_ended()

```
auctions.close_auction_if_ended (
             auction_id,
             expected_overtime = 0)
```

```
@brief Closes auction if ended, handling concurrent overtime changes.

App context check: refreshes auction under lock, verifies end time vs expected_overtime.
Sets status='sold', highest bidder as winner. Emits SocketIO, reschedules next.

@param auction_id Integer: Auction to potentially close.
@param expected_overtime Integer: Expected overtime at scheduling (default 0).

@note Idempotent: reschedules if not ended or overtime changed.

@return None (DB update + emit side-effects).
```

get_auction_lock()

```
auctions.get_auction_lock (
             auction_id)
```

```
@brief Returns the lock object for auction-specific synchronization.

Assumes _auction_locks dict populated elsewhere (e.g., on-demand RLock).

@param auction_id Integer/String: Auction identifier.

@note Global/shared _auction_locks; ensure initialized before use.

@return Lock instance for auction_id (e.g., threading.RLock).
```

get_next_auction_to_close()

```
auctions.get_next_auction_to_close ()
```

```
@brief Finds the active auction ending soonest (end_date + overtime).

Queries 'at_auction' status auctions, returns min by computed end time.

@return Auction instance closest to ending, or None if none active.
```

get_next_auction_to_open()

```
auctions.get_next_auction_to_open ()
```

```
@brief Opens 'not_issued' auction to 'at_auction' status.

Updates status, commits, schedules closure. Reschedules next open if invalid.

@param auction_id Integer: Auction to open.

@note Idempotent: skips if wrong status, chains to closure scheduling.

@return None (DB update + scheduling).
```

handle_join()

```
auctions.handle_join (
              data)
```

```
@brief Handles client joining auction SocketIO room.

Validates 'auction' field, calls join_room(). Broadcasts user_joined confirmation.

@param data Dict: {'auction': room_id str/int}

@note Emits 'error' code 0/1 on fail; 'user_joined' success to room.

@return None (room ops + emit).
```

handle_leave()

```
auctions.handle_leave (
              data)
```

```
@brief Handles client leaving auction SocketIO room.

Validates 'auction' in data, calls leave_room(). Broadcasts user_left to room.

@param data Dict: {'auction': room_id str/int}

@note Emits 'error' on validation fail; 'user_left' success to room.

@return None (room ops + emit).
```

on_auction_update()

```
auctions.on_auction_update ()
```

```
@brief Triggers auction scheduling refresh after auction changes.

Recreates app context and calls schedule_next_auction() to handle updates
like new bids extending overtime.

@note Call after auction modifications (bids, status changes).

@return None (scheduling side-effect).
```

open_auction()

```
auctions.open_auction (
             auction_id)
```

```
@brief Opens 'not_issued' auction to 'at_auction' status.

Updates status, commits, schedules closure. Reschedules next open if invalid.

@param auction_id Integer: Auction to open.

@note Idempotent: skips wrong status, chains to closure scheduling.

@return None (DB update + scheduling).
```

schedule_auction_closure()

```
auctions.schedule_auction_closure (
             auction)
```

```
@brief Schedules auction closure job at end_date + overtime.

Removes existing job if present, adds new date-trigger job for close_auction_if_ended.
Passes id_auction as arg, overtime as kwarg.

@param auction Auction instance with end_date, overtime, id_auction.

@note Job ID: 'close_auction_{id_auction}' for uniqueness.

@return None (scheduling side-effect).
```

schedule_auction_opening()

```
auctions.schedule_auction_opening (
             auction)
```

```
@brief Schedules auction opening job at start_date (or immediate if past).

Removes existing job, adds date-trigger for open_auction(id_auction).

@param auction Auction instance with start_date, id_auction.

@note Job ID: 'open_auction_{id_auction}'. Runs now+1s if start_date past.

@return None (scheduling side-effect).
```

schedule_next_auction()

```
auctions.schedule_next_auction ()
```

```
@brief Schedules the next auction closure or reschedules self in 1 minute.

Creates app context, finds next auction via get_next_auction_to_close().
Calls schedule_auction_closure() if found, else recurses via scheduler.

@note Uses app factory and scheduler; logs when no auctions.

@return None (scheduling side-effect).
```

schedule_open_next_auction()

```
auctions.schedule_open_next_auction ()
```

```
@brief Schedules next 'not_issued' auction opening or reschedules self.

App context finds get_next_auction_to_open(), calls schedule_auction_opening().
Recurses every 1min if none pending via SCHEDULER job.

@note Uses global APP/SCHEDULER; fixed job ID 'schedule_open_next_auction'.

@return None (scheduling side-effect).
```

start_scheduler() [1/2]

```
auctions.start_scheduler ()
```

```
@brief Initializes and starts BackgroundScheduler with initial auction scheduling.

Sets global scheduler, starts it, creates app context for schedule_next_auction().

@note Modifies global 'scheduler' variable.

@return Active scheduler instance.
```

start_scheduler() [2/2]

```
auctions.start_scheduler (
              app)
```

```
@brief Initializes scheduler with app context binding.

Sets global APP for context usage, creates/starts BackgroundScheduler.

@param app Flask app instance.

@note Call before scheduling jobs; globals used in scheduled funcs.

@return None (modifies globals SCHEDULER, APP).
```

6.3.2 Variable Documentation

_auction_locks

```
auctions._auction_locks = defaultdict(Lock)  [protected]
```

APP

```
auctions.APP = None
```

level

```
auctions.level
```

logger

```
auctions.logger = logging.getLogger("auctions_scheduler_test")
```

SCHEDULER

```
auctions.SCHEDULER = None
```

6.4 Categories Namespace Reference

Functions

- get_all_categories ()

Variables

- bp = Blueprint('categories', __name__, url_prefix='/api')

6.4.1 Function Documentation

get_all_categories()

```
Categories.get_all_categories ()
```

```
@brief Returns a list of all categories.

@return list: A list of dictionaries where each dictionary contains the id and name of a category.
```

6.4.2 Variable Documentation

bp

```
Categories.bp = Blueprint('categories', __name__, url_prefix='/api')
```

6.5 db_objects Namespace Reference

Classes

- class AuctionPriceHistory
- class Auctions
- class Categories
- class CategoriesAuction
- class PhotosItem
- class Users

Variables

- db = SQLAlchemy()

6.5.1 Variable Documentation

db

```
db_objects.db = SQLAlchemy()
```

6.6 main Namespace Reference

Functions

- create_app ()

Variables

- app = create_app()
- scheduler = start_scheduler(app)
- debug
- True
- host
- port

6.6.1 Function Documentation

create_app()

```
main.create_app ()
```

```
@brief Creates and configures the Flask application instance.

Loads .env, sets JWT/SQLAlchemy configs, CORS, SocketIO, blueprints.
Creates upload directory.

@return Configured Flask app instance.
```

6.6.2 Variable Documentation

app

```
main.app = create_app()
```

debug

```
main.debug
```

host

```
main.host
```

port

```
main.port
```

scheduler

```
main.scheduler = start_scheduler(app)
```

True

```
main.True
```

6.7 Uploads Namespace Reference

Functions

- allowed_file (filename)
- upload_image ()
- serve_image (filename)

Variables

- bp = Blueprint('uploads', __name__, url_prefix='')

6.7.1 Function Documentation

allowed_file()

```
Uploads.allowed_file (
              filename)
```

```
@brief Validates file extension against app config ALLOWED_EXTENSIONS.

Case-insensitive check using rsplit for final extension. Defaults to
{'png', 'jpg', 'jpeg', 'webp'} if config missing.

@param filename String: File name to validate.

@return True if allowed extension, False otherwise.
```

serve_image()

```
Uploads.serve_image (
              filename)
```

```
@brief Serves uploaded image file by filename.

Uses send_from_directory with secure_filename for path traversal protection.

@param filename Path param: Uploaded image filename (e.g., "uuid.webp").
@return Image file stream.
```

upload_image()

```
Uploads.upload_image ()
```

```
@brief Uploads image file, validates type/size, saves with UUID name.

Requires JWT. Checks 'image' file field, allowed_file(), size <= MAX_FILE_SIZE.
Saves to UPLOAD_DIRECTORY as UUID.ext, returns relative URL.

@return Image URL on success.

@retval 201 Success: {"image_url": "/uuid.ext"}
@retval 400 No file, empty name, invalid type, oversized
```

6.7.2 Variable Documentation

bp

```
Uploads.bp = Blueprint('uploads', __name__, url_prefix='')
```

6.8 Users Namespace Reference

Functions

- get_user_info ()
- change_password ()
- login ()
- register ()
- check_email ()

Variables

- bp = Blueprint('users', __name__, url_prefix='/api')

6.8.1 Function Documentation

change_password()

```
Users.change_password ()
```

```
@brief Changes authenticated user's password after old password verification.

Requires JWT. Validates old_password via check_password(), sets new_password directly.

@param old_password JSON body: Current password (required).
@param new_password JSON body: New password (required).

@return Success message.

@retval 200 Success: {"message": "Password changed successfully"}
@retval 400 Missing passwords or incorrect old password
@retval 404 User not found
```

check_email()

```
Users.check_email ()
```

```
@brief Checks if an email is already registered.

Simple existence check for email uniqueness before registration.

@param email Query param: Email address to verify (required).

@return Boolean existence flag.

@retval 200 Success: {"exists": true/false}
@retval 400 Missing email param
```

get_user_info()

```
Users.get_user_info ()
```

```
@brief Retrieves profile info for the authenticated user.

Returns basic user details from JWT identity. No input parameters needed.

@return User profile JSON.

@retval 200 Success: {"first_name", "last_name", "email", "phone_number", "create_account_date"}
@retval 404 User not found
```

login()

```
Users.login ()
```

```
@brief Authenticates user and returns JWT access token.

Validates credentials via user.check_password() method, generates JWT
with identity=user.id_user on success.

@param email JSON body: User email (required).
@param password JSON body: User password (required).

@return JWT token.

@retval 200 Success: {"access_token": "..."}
@retval 400 Missing credentials
@retval 401 Invalid email/password
```

register()

```
Users.register ()
```

```
@brief Registers a new user and returns JWT access token.

Validates required fields, checks email uniqueness (no hash--consider bcrypt).
Creates user with create_account_date, generates JWT for identity=user.id_user.

@param first_name String: Required.
@param last_name String: Required.
@param email String: Required, must be unique.
@param password String: Required (plain text stored).
@param phone_number String: Optional.

@return JWT token on success.

@retval 201 Success: {"access_token": "..."}
@retval 400 Missing required fields
@retval 400 Email already exists
```

6.8.2 Variable Documentation

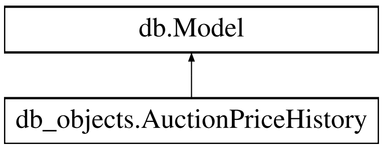
bp

```
Users.bp = Blueprint('users', __name__, url_prefix='/api')
```

7 Class Documentation

7.1 db_objects.AuctionPriceHistory Class Reference

Inheritance diagram for db_objects.AuctionPriceHistory:



Public Member Functions

- to_dict (self)

Static Public Attributes

- id_price_history = db.Column(db.Integer, primary_key=True, autoincrement=True)
- id_auction = db.Column(db.Integer, ForeignKey('auctions.id_auction'), nullable=False)
- id_user = db.Column(db.Integer, ForeignKey('users.id_user'), nullable=False)
- new_price = db.Column(db.Float(10, 2), nullable=False)
- price_reprint_date = db.Column(db.DateTime, nullable=False, default=func.now())

Static Private Attributes

- str __tablename__ = 'auction_price_history'

7.1.1 Detailed Description

```
@brief Bid history tracking per auction/user with timestamps.

Records each bid with new_price > previous, price_reprint_date for ordering/conflict resolution.
Used for highest bid lookup and user bid lists.
```

7.1.2 Member Function Documentation

to_dict()

```
db_objects.AuctionPriceHistory.to_dict (
              self)
```

```
@brief Serializes bid record to dict.

@return Dict with all fields.
```

7.1.3 Member Data Documentation

__tablename__

```
str db_objects.AuctionPriceHistory.__tablename__ = 'auction_price_history'  [static], [private]
```

id_auction

```
db_objects.AuctionPriceHistory.id_auction = db.Column(db.Integer, ForeignKey('auctions.id_↩
auction'), nullable=False)  [static]
```

id_price_history

```
db_objects.AuctionPriceHistory.id_price_history = db.Column(db.Integer, primary_key=True,
autoincrement=True)  [static]
```

id_user

```
db_objects.AuctionPriceHistory.id_user = db.Column(db.Integer, ForeignKey('users.id_user'),
nullable=False)  [static]
```

new_price

```
db_objects.AuctionPriceHistory.new_price = db.Column(db.Float(10, 2), nullable=False)  [static]
```

price_reprint_date

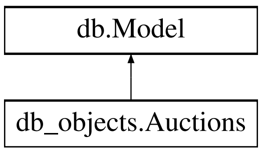
```
db_objects.AuctionPriceHistory.price_reprint_date = db.Column(db.DateTime, nullable=False,
default=func.now()) [static]
```

The documentation for this class was generated from the following file:

- db_objects.py

7.2 db_objects.Auctions Class Reference

Inheritance diagram for db_objects.Auctions:



Public Member Functions

- to_dict (self)

Static Public Attributes

- id_auction = db.Column(db.Integer, primary_key=True, autoincrement=True)
- title = db.Column(db.String(255), nullable=False)
- description = db.Column(db.String(2048), nullable=True)
- id_seller = db.Column(db.Integer, ForeignKey('users.id_user'), nullable=False)
- start_price = db.Column(db.Float(10, 2), nullable=False)
- start_date = db.Column(db.DateTime, nullable=False)
- end_date = db.Column(db.DateTime, nullable=False)
- overtime = db.Column(db.Integer, nullable=False, default=0)
- status = db.Column(db.Enum('at_auction','sold','not_issued'), nullable=False)
- id_winner = db.Column(db.Integer, ForeignKey('users.id_user'), nullable=True)

Static Private Attributes

- str __tablename__ = 'auctions'

7.2.1 Detailed Description

```
@brief Auction model for buy-now auctions with bidding and overtime.

Stores auction details, seller/winner FKs to Users, status enum.
Supports bidding history via separate AuctionPriceHistory table.
```

7.2.2 Member Function Documentation

to_dict()

```
db_objects.Auctions.to_dict (
              self)
```

```
@brief Serializes auction to JSON-compatible dict.

@return Dict with core fields (excludes description).
```

7.2.3 Member Data Documentation

__tablename__

```
str db_objects.Auctions.__tablename__ = 'auctions'  [static], [private]
```

description

```
db_objects.Auctions.description = db.Column(db.String(2048), nullable=True)  [static]
```

end_date

```
db_objects.Auctions.end_date = db.Column(db.DateTime, nullable=False)  [static]
```

id_auction

```
db_objects.Auctions.id_auction = db.Column(db.Integer, primary_key=True, autoincrement=True)
[static]
```

id_seller

```
db_objects.Auctions.id_seller = db.Column(db.Integer, ForeignKey('users.id_user'), nullable=False)
[static]
```

id_winner

```
db_objects.Auctions.id_winner = db.Column(db.Integer, ForeignKey('users.id_user'), nullable=True)
[static]
```

overtime

```
db_objects.Auctions.overtime = db.Column(db.Integer, nullable=False, default=0)  [static]
```

start_date

```
db_objects.Auctions.start_date = db.Column(db.DateTime, nullable=False)  [static]
```

start_price

```
db_objects.Auctions.start_price = db.Column(db.Float(10, 2), nullable=False)  [static]
```

status

```
db_objects.Auctions.status = db.Column(db.Enum('at_auction','sold','not_issued'), nullable=False)
[static]
```

title

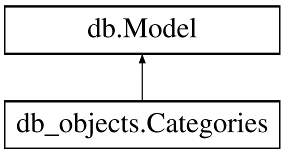
```
db_objects.Auctions.title = db.Column(db.String(255), nullable=False)  [static]
```

The documentation for this class was generated from the following file:

- db_objects.py

7.3 db_objects.Categories Class Reference

Inheritance diagram for db_objects.Categories:



Public Member Functions

- to_dict (self)

Static Public Attributes

- id_category = db.Column(db.Integer, primary_key=True, autoincrement=True)
- category_name = db.Column(db.String(255), nullable=False)

Static Private Attributes

- str __tablename__ = 'categories'

7.3.1 Detailed Description

```
@brief Auction category lookup table.

Simple id/name pairs, many-to-many with Auctions via CategoriesAuction junction.
```

7.3.2 Member Function Documentation

to_dict()

```
db_objects.Categories.to_dict (
              self)
```

```
@brief Serializes category to basic dict.

@return {'id_category': int, 'category_name': str}
```

7.3.3 Member Data Documentation

__tablename__

```
str db_objects.Categories.__tablename__ = 'categories'  [static], [private]
```

category_name

```
db_objects.Categories.category_name = db.Column(db.String(255), nullable=False)  [static]
```

id_category

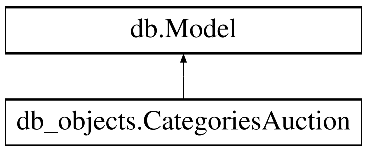
```
db_objects.Categories.id_category = db.Column(db.Integer, primary_key=True, autoincrement=True)
[static]
```

The documentation for this class was generated from the following file:

- db_objects.py

7.4 db_objects.CategoriesAuction Class Reference

Inheritance diagram for db_objects.CategoriesAuction:



Public Member Functions

- to_dict (self)

Static Public Attributes

- id_category = db.Column(db.Integer, ForeignKey('categories.id_category'), primary_key=True, nullable=False)
- id_auction = db.Column(db.Integer, ForeignKey('auctions.id_auction'), primary_key=True, nullable=False)

Static Private Attributes

- str __tablename__ = 'categories_auction'

7.4.1 Detailed Description

```
@brief Many-to-many junction between Auctions and Categories.

Composite PK (id_category, id_auction) FKs to both tables.
```

7.4.2 Member Function Documentation

to_dict()

```
db_objects.CategoriesAuction.to_dict (
              self)
```

```
@brief Serializes junction record.

@return {'id_category': int, 'id_auction': int}
```

7.4.3 Member Data Documentation

__tablename__

```
str db_objects.CategoriesAuction.__tablename__ = 'categories_auction'  [static], [private]
```

id_auction

```
db_objects.CategoriesAuction.id_auction = db.Column(db.Integer, ForeignKey('auctions.id_↩
auction'), primary_key=True, nullable=False)  [static]
```

id_category

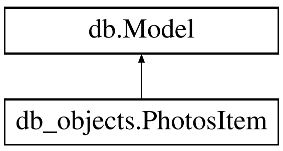
```
db_objects.CategoriesAuction.id_category = db.Column(db.Integer, ForeignKey('categories.id_↩
category'), primary_key=True, nullable=False)  [static]
```

The documentation for this class was generated from the following file:

- db_objects.py

7.5 db_objects.PhotosItem Class Reference

Inheritance diagram for db_objects.PhotosItem:



Public Member Functions

- to_dict (self)

Static Public Attributes

- id_photo = db.Column(db.Integer, primary_key=True, autoincrement=True)
- id_auction = db.Column(db.Integer, ForeignKey('auctions.id_auction'), nullable=False)
- photo = db.Column(db.String(512), nullable=False)
- is_main_photo = db.Column(db.Boolean, nullable=False, default=False)

Static Private Attributes

- str __tablename__ = 'photos_item'

7.5.1 Detailed Description

```
@brief Auction photos with main photo flag.

Stores image paths/URLs per auction. One main photo per auction typical.
```

7.5.2 Member Function Documentation

to_dict()

```
db_objects.PhotosItem.to_dict (
              self)
```

```
@brief Serializes photo record.

@return Dict with id_photo, id_auction, photo path/URL, is_main_photo.
```

7.5.3 Member Data Documentation

__tablename__

```
str db_objects.PhotosItem.__tablename__ = 'photos_item'  [static], [private]
```

id_auction

```
db_objects.PhotosItem.id_auction = db.Column(db.Integer, ForeignKey('auctions.id_auction'),
nullable=False)  [static]
```

id_photo

```
db_objects.PhotosItem.id_photo = db.Column(db.Integer, primary_key=True, autoincrement=True)
[static]
```

is_main_photo

```
db_objects.PhotosItem.is_main_photo = db.Column(db.Boolean, nullable=False, default=False)
[static]
```

photo

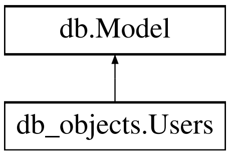
```
db_objects.PhotosItem.photo = db.Column(db.String(512), nullable=False)  [static]
```

The documentation for this class was generated from the following file:

- db_objects.py

7.6 db_objects.Users Class Reference

Inheritance diagram for db_objects.Users:



Public Member Functions

- check_password (self, password)
- to_dict (self)

Static Public Attributes

- id_user = db.Column(db.Integer, primary_key=True, autoincrement=True)
- first_name = db.Column(db.String(64), nullable=False)
- last_name = db.Column(db.String(64), nullable=False)
- email = db.Column(db.String(128), nullable=False, unique=True)
- password = db.Column(db.String(1024), nullable=False)
- phone_number = db.Column(db.String(20), nullable=True)
- create_account_date = db.Column(db.DateTime, nullable=False, default=func.now())

Static Private Attributes

- str __tablename__ = 'users'

7.6.1 Detailed Description

```
@brief User model for authentication and profiles.

Stores credentials, profile data.
Email unique. check_password() for login (insecure equality).
```

7.6.2 Member Function Documentation

check_password()

```
db_objects.Users.check_password (
              self,
              password)
```

```
@brief Compares provided password to stored (plain text).

@param password String: Plain password to check.

@return True if matches.
```

to_dict()

```
db_objects.Users.to_dict (
              self)
```

```
@brief Serializes user excluding password.

@return Profile dict with dates as datetime objects.
```

7.6.3 Member Data Documentation

__tablename__

```
str db_objects.Users.__tablename__ = 'users'  [static], [private]
```

create_account_date

```
db_objects.Users.create_account_date = db.Column(db.DateTime, nullable=False, default=func.↩
now())  [static]
```

email

```
db_objects.Users.email = db.Column(db.String(128), nullable=False, unique=True)  [static]
```

first_name

```
db_objects.Users.first_name = db.Column(db.String(64), nullable=False)  [static]
```

id_user

```
db_objects.Users.id_user = db.Column(db.Integer, primary_key=True, autoincrement=True)  [static]
```

last_name

```
db_objects.Users.last_name = db.Column(db.String(64), nullable=False)  [static]
```

password

```
db_objects.Users.password = db.Column(db.String(1024), nullable=False)  [static]
```

phone_number

```
db_objects.Users.phone_number = db.Column(db.String(20), nullable=True)  [static]
```

The documentation for this class was generated from the following file:

- db_objects.py

8 File Documentation

8.1 app_state.py File Reference

Namespaces

- namespace app_state

Variables

- app_state.socketio = SocketIO(cors_allowed_origins="*")

8.2 auctions.py File Reference

Namespaces

- namespace auctions

Functions

- auctions.handle_join (data)
- auctions.handle_leave (data)
- auctions.get_auction_lock (auction_id)
- auctions.get_next_auction_to_close ()
- auctions.get_next_auction_to_open ()
- auctions.close_auction_if_ended (auction_id, expected_overtime=0)
- auctions.open_auction (auction_id)
- auctions.schedule_auction_closure (auction)
- auctions.schedule_auction_opening (auction)
- auctions.schedule_next_auction ()
- auctions.schedule_open_next_auction ()
- auctions.start_scheduler (app)
- auctions.start_scheduler ()
- auctions.on_auction_update ()

Variables

- auctions.level
- auctions.logger = logging.getLogger("auctions_scheduler_test")
- auctions._auction_locks = defaultdict(Lock)
- auctions.SCHEDULER = None
- auctions.APP = None

8.3 routes/Auctions.py File Reference

Namespaces

- namespace Auctions

Functions

- Auctions.get_all_auctions ()
- Auctions.get_auction_details ()
- Auctions.create_auction ()
- Auctions.place_bid ()
- Auctions.get_user_own_auctions ()
- Auctions.get_user_auctions ()
- Auctions.archived_auctions ()
- Auctions.delete_auction ()

Variables

- Auctions.bp = Blueprint('auctions', __name__, url_prefix='/api')

8.4 db_objects.py File Reference

Classes

- class db_objects.Auctions
- class db_objects.AuctionPriceHistory
- class db_objects.Categories
- class db_objects.CategoriesAuction
- class db_objects.PhotosItem
- class db_objects.Users

Namespaces

- namespace db_objects

Variables

- db_objects.db = SQLAlchemy()

8.5 docs/mainpage.dox File Reference

8.6 main.py File Reference

Namespaces

- namespace main

Functions

- main.create_app ()

Variables

- main.app = create_app()
- main.scheduler = start_scheduler(app)
- main.debug
- main.True
- main.host
- main.port

8.7 routes/Categories.py File Reference

Namespaces

- namespace Categories

Functions

- Categories.get_all_categories ()

Variables

- Categories.bp = Blueprint('categories', __name__, url_prefix='/api')

8.8 routes/Uploads.py File Reference

Namespaces

- namespace Uploads

Functions

- Uploads.allowed_file (filename)
- Uploads.upload_image ()
- Uploads.serve_image (filename)

Variables

- Uploads.bp = Blueprint('uploads', __name__, url_prefix='')

8.9 routes/Users.py File Reference

Namespaces

- namespace Users

Functions

- Users.get_user_info ()
- Users.change_password ()
- Users.login ()
- Users.register ()
- Users.check_email ()

Variables

- Users.bp = Blueprint('users', __name__, url_prefix='/api')

Index