Largo

Generated by Doxygen 1.12.0

1 Documentation for "Largo" Auction Backend **2**
1.0.1 Overview . . . 2
1.0.2 Features . . . 2
1.0.3 Architecture . . . 3
1.0.4 Technology Stack . . . 4
1.0.5 Environment Variables . . . 4
1.0.6 Getting Started . . . 4

2 Namespace Index **5**
2.1 Namespace List . . . 5

3 Hierarchical Index **5**
3.1 Class Hierarchy . . . 5

4 Class Index **5**
4.1 Class List . . . 5

5 File Index **6**
5.1 File List . . . 6

6 Namespace Documentation **6**
6.1 app_state Namespace Reference . . . 6
6.1.1 Variable Documentation . . . 6
6.2 Auctions Namespace Reference . . . 7
6.2.1 Function Documentation . . . 7
6.2.2 Variable Documentation . . . 11
6.3 auctions Namespace Reference . . . 11
6.3.1 Function Documentation . . . 12
6.3.2 Variable Documentation . . . 17
6.4 Categories Namespace Reference . . . 17
6.4.1 Function Documentation . . . 18
6.4.2 Variable Documentation . . . 18
6.5 db_objects Namespace Reference . . . 18
6.5.1 Variable Documentation . . . 18
6.6 main Namespace Reference . . . 19
6.6.1 Function Documentation . . . 19
6.6.2 Variable Documentation . . . 19
6.7 Uploads Namespace Reference . . . 20
6.7.1 Function Documentation . . . 20
6.7.2 Variable Documentation . . . 21
6.8 Users Namespace Reference . . . 21
6.8.1 Function Documentation . . . 22
6.8.2 Variable Documentation . . . 24

7 Class Documentation 24
7.1 db_objects.AuctionPriceHistory Class Reference . . . 24
7.1.1 Detailed Description . . . 25
7.1.2 Member Function Documentation . . . 25
7.1.3 Member Data Documentation . . . 25
7.2 db_objects.Auctions Class Reference . . . 26
7.2.1 Detailed Description . . . 26
7.2.2 Member Function Documentation . . . 27
7.2.3 Member Data Documentation . . . 27
7.3 db_objects.Categories Class Reference . . . 28
7.3.1 Detailed Description . . . 29
7.3.2 Member Function Documentation . . . 29
7.3.3 Member Data Documentation . . . 29
7.4 db_objects.CategoriesAuction Class Reference . . . 29
7.4.1 Detailed Description . . . 30
7.4.2 Member Function Documentation . . . 30
7.4.3 Member Data Documentation . . . 30
7.5 db_objects.PhotosItem Class Reference . . . 31
7.5.1 Detailed Description . . . 31
7.5.2 Member Function Documentation . . . 32
7.5.3 Member Data Documentation . . . 32
7.6 db_objects.Users Class Reference . . . 33
7.6.1 Detailed Description . . . 33
7.6.2 Member Function Documentation . . . 33
7.6.3 Member Data Documentation . . . 34

8 File Documentation 35
8.1 app_state.py File Reference . . . 35
8.2 auctions.py File Reference . . . 35
8.3 routes/Auctions.py File Reference . . . 36
8.4 db_objects.py File Reference . . . 37
8.5 docs/mainpage.dox File Reference . . . 38
8.6 main.py File Reference . . . 38
8.7 routes/Categories.py File Reference . . . 38
8.8 routes/Uploads.py File Reference . . . 38
8.9 routes/Users.py File Reference . . . 39

Index 41

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:

app_state 6

Auctions 7

auctions 11

Categories 17

db_objects 18

main 19

Uploads 20

Users 21

3 Hierarchical Index

3.1 Class Hierarchy

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

db.Model

db_objects.AuctionPriceHistory 24

db_objects.Auctions 26

db_objects.Categories 28

db_objects.CategoriesAuction 29

db_objects.PhotosItem 31

db_objects.Users 33

4 Class Index

4.1 Class List

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

db_objects.AuctionPriceHistory
Bid history tracking per auction/user with timestamps 24

db_objects.Auctions
Auction model for buy-now auctions with bidding and overtime 26

db_objects.Categories
Auction category lookup table 28

db_objects.CategoriesAuction
Many-to-many junction between Auctions and Categories 29

db_objects.PhotosItem
Auction photos with main photo flag 31

db_objects.Users
User model for authentication and profiles 33

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

app_state.py 35

auctions.py 35

db_objects.py 37

main.py 38

routes/Auctions.py 36

routes/Categories.py 38

routes/Uploads.py 38

routes/Users.py 39

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 ()

  Retrieves all auctions with aggregated max prices and categories.
- get_auction_details ()

  Retrieves complete details for a specific auction by ID.
- create_auction ()

  Creates a new auction for the authenticated seller with photos and categories.
- place_bid ()

  Places a bid on an active auction with validation and overtime extension.
- get_user_own_auctions ()

  Retrieves all auctions owned by the authenticated seller.
- get_user_auctions ()

  Retrieves active auctions for the authenticated user.
- archived_auctions ()

  Retrieves archived (ended) auctions owned by the authenticated seller.
- delete_auction ()

  Deletes an auction.

Variables

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

6.2.1 Function Documentation

archived_auctions()

```
Auctions.archived_auctions ()
```

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.

Returns

JSON array of archived auctions with: id_auction, description, starting_price, final_price, winner_id, winner↩ _name, end_date, title, main_photo, categories.

Return values

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

create_auction()

```
Auctions.create_auction ()
```

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.

Parameters

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

Returns

JSON with new auction ID.

Return values

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

delete_auction()

```
Auctions.delete_auction ()
```

Deletes an auction.

Only the seller can delete an auction.

@detail Requires JWT authentication. Parses id_auction from JSON body, verifies user ownership, deletes associated photos and bids first, then the auction. Calls on_auction_update() post-deletion.

Parameters

id_auction	The id of the auction to be deleted.

Returns

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

Return values

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

get_all_auctions()

```
Auctions.get_all_auctions ()
```

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.

Returns

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

Return values

	200	Success: Complete auctions list

get_auction_details()

```
Auctions.get_auction_details ()
```

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).

Parameters

	id_auction	Query param: Integer auction ID (required).

Returns

Full auction JSON with seller/winner names, main_photo, photos array (non-main), current_price, highest_↩ bidder ID, categories.

Return values

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

get_user_auctions()

```
Auctions.get_user_auctions ()
```

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.

Returns

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

Return values

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

get_user_own_auctions()

```
Auctions.get_user_own_auctions ()
```

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.

Returns

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.

Return values

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

place_bid()

```
Auctions.place_bid ()
```

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 Auction↩ PriceHistory, extends overtime by 60s if <60s remain. Emits SocketIO update.

Parameters

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

Returns

JSON success or error message.

Return values

	200	Success: {"message": "Bid placed successfully"}
	400	Missing params, inactive auction, too low bid, ended, or duplicate timestamp
	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)

  Handles client joining auction SocketIO room.
- handle_leave (data)

  Handles client leaving auction SocketIO room.
- get_auction_lock (auction_id)

  Returns the lock object for auction-specific synchronization.
- get_next_auction_to_close ()

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

- get_next_auction_to_open ()

  Opens 'not_issued' auction to 'at_auction' status.
- close_auction_if_ended (auction_id, expected_overtime=0)

  Closes auction if ended, handling concurrent overtime changes.
- open_auction (auction_id)

  Opens 'not_issued' auction to 'at_auction' status.
- schedule_auction_closure (auction)

  Schedules auction closure job at end_date + overtime.
- schedule_auction_opening (auction)

  Schedules auction opening job at start_date (or immediate if past).
- schedule_next_auction ()

  Schedules the next auction closure or reschedules self in 1 minute.
- schedule_open_next_auction ()

  Schedules next 'not_issued' auction opening or reschedules self.
- start_scheduler (app)

  Initializes scheduler with app context binding.
- start_scheduler ()

  Initializes and starts BackgroundScheduler with initial auction scheduling.
- on_auction_update ()

  Triggers auction scheduling refresh after auction changes.

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)
```

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.

Parameters

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

Note

Idempotent: reschedules if not ended or overtime changed.

Returns

None (DB update + emit side-effects).

get_auction_lock()

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

Returns the lock object for auction-specific synchronization.

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

Parameters

	auction_id	Integer/String: Auction identifier.

Note

Global/shared _auction_locks; ensure initialized before use.

Returns

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

get_next_auction_to_close()

```
auctions.get_next_auction_to_close ()
```

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

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

Returns

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

get_next_auction_to_open()

```
auctions.get_next_auction_to_open ()
```

Opens 'not_issued' auction to 'at_auction' status.

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

Parameters

	auction_id	Integer: Auction to open.

Note

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

Returns

None (DB update + scheduling).

handle_join()

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

Handles client joining auction SocketIO room.

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

Parameters

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

Note

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

Returns

None (room ops + emit).

handle_leave()

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

Handles client leaving auction SocketIO room.

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

Parameters

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

Note

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

Returns

None (room ops + emit).

on_auction_update()

```
auctions.on_auction_update ()
```

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).

Returns

None (scheduling side-effect).

open_auction()

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

Opens 'not_issued' auction to 'at_auction' status.

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

Parameters

	auction_id	Integer: Auction to open.

Note

Idempotent: skips wrong status, chains to closure scheduling.

Returns

None (DB update + scheduling).

schedule_auction_closure()

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

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.

Parameters

	auction	Auction instance with end_date, overtime, id_auction.

Note

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

Returns

None (scheduling side-effect).

schedule_auction_opening()

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

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

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

Parameters

	auction	Auction instance with start_date, id_auction.

Note

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

Returns

None (scheduling side-effect).

schedule_next_auction()

```
auctions.schedule_next_auction ()
```

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.

Returns

None (scheduling side-effect).

schedule_open_next_auction()

```
auctions.schedule_open_next_auction ()
```

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'.

Returns

None (scheduling side-effect).

start_scheduler() [1/2]

```
auctions.start_scheduler ()
```

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.

Returns

Active scheduler instance.

start_scheduler() [2/2]

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

Initializes scheduler with app context binding.

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

Parameters

	app	Flask app instance.

Note

Call before scheduling jobs; globals used in scheduled funcs.

Returns

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 ()

  Returns a list of all categories.

Variables

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

6.4.1 Function Documentation

get_all_categories()

```
Categories.get_all_categories ()
```

Returns a list of all categories.

Returns

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

  Bid history tracking per auction/user with timestamps.
- class Auctions

  Auction model for buy-now auctions with bidding and overtime.
- class Categories

  Auction category lookup table.
- class CategoriesAuction

  Many-to-many junction between Auctions and Categories.
- class PhotosItem

  Auction photos with main photo flag.
- class Users

  User model for authentication and profiles.

Variables

- db = SQLAlchemy()

6.5.1 Variable Documentation

db

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

6.6 main Namespace Reference

Functions

- create_app ()

  Creates and configures the Flask application instance.

Variables

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

6.6.1 Function Documentation

create_app()

```
main.create_app ()
```

Creates and configures the Flask application instance.

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

Returns

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)

  Validates file extension against app config ALLOWED_EXTENSIONS.
- upload_image ()

  Uploads image file, validates type/size, saves with UUID name.
- serve_image (filename)

  Serves uploaded image file by filename.

Variables

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

6.7.1 Function Documentation

allowed_file()

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

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.

Parameters

	filename	String: File name to validate.

Returns

True if allowed extension, False otherwise.

serve_image()

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

Serves uploaded image file by filename.

Uses send_from_directory with secure_filename for path traversal protection.

Parameters

	filename	Path param: Uploaded image filename (e.g., "uuid.webp").

Returns

Image file stream.

upload_image()

```
Uploads.upload_image ()
```

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.

Returns

Image URL on success.

Return values

	201	Success: {"image_url": "/uuid.ext"}
	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 ()
  Retrieves profile info for the authenticated user.
- change_password ()
  Changes authenticated user's password after old password verification.
- login ()
  Authenticates user and returns JWT access token.
- register ()
  Registers a new user and returns JWT access token.
- check_email ()
  Checks if an email is already registered.

Variables

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

6.8.1 Function Documentation

change_password()

```
Users.change_password ()
```

Changes authenticated user's password after old password verification.

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

Parameters

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

Returns

Success message.

Return values

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

check_email()

```
Users.check_email ()
```

Checks if an email is already registered.

Simple existence check for email uniqueness before registration.

Parameters

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

Returns

Boolean existence flag.

Return values

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

get_user_info()

```
Users.get_user_info ()
```

Retrieves profile info for the authenticated user.

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

Returns

User profile JSON.

Return values

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

login()

```
Users.login ()
```

Authenticates user and returns JWT access token.

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

Parameters

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

Returns

JWT token.

Return values

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

register()

```
Users.register ()
```

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.

Parameters

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

Returns

JWT token on success.

Return values

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

6.8.2 Variable Documentation

bp

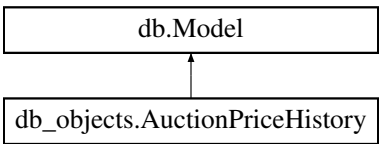
```
Users.bp = Blueprint('users', __name__, url_prefix='/api')
```

7 Class Documentation

7.1 db_objects.AuctionPriceHistory Class Reference

Bid history tracking per auction/user with timestamps.

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

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]
```

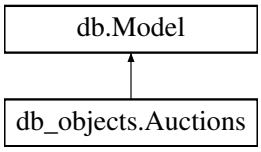
The documentation for this class was generated from the following file:

- db_objects.py

7.2 db_objects.Auctions Class Reference

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

Inheritance diagram for db_objects.Auctions:



Public Member Functions

- to_dict (self)

  Serializes auction to JSON-compatible dict.

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

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)
```

Serializes auction to JSON-compatible dict.

Returns

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]
```

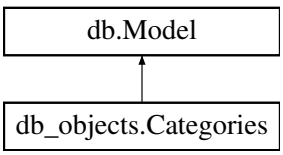
The documentation for this class was generated from the following file:

- db_objects.py

7.3 db_objects.Categories Class Reference

Auction category lookup table.

Inheritance diagram for db_objects.Categories:



Public Member Functions

- to_dict (self)

  Serializes category to basic dict.

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

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)
```

Serializes category to basic dict.

Returns

{'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]
```

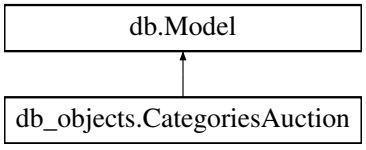
The documentation for this class was generated from the following file:

- db_objects.py

7.4 db_objects.CategoriesAuction Class Reference

Many-to-many junction between Auctions and Categories.

Inheritance diagram for db_objects.CategoriesAuction:



Public Member Functions

- to_dict (self)

  Serializes junction record.

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

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)
```

Serializes junction record.

Returns

{'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]
```

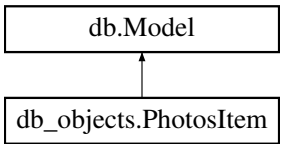
The documentation for this class was generated from the following file:

- db_objects.py

7.5 db_objects.PhotosItem Class Reference

Auction photos with main photo flag.

Inheritance diagram for db_objects.PhotosItem:



Public Member Functions

- to_dict (self)

  Serializes photo record.

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

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)
```

Serializes photo record.

Returns

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]
```

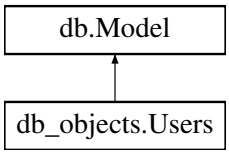
The documentation for this class was generated from the following file:

- db_objects.py

7.6 db_objects.Users Class Reference

User model for authentication and profiles.

Inheritance diagram for db_objects.Users:



Public Member Functions

- check_password (self, password)

  Compares provided password to stored (plain text).
- to_dict (self)

  Serializes user excluding password.

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

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)
```

Compares provided password to stored (plain text).

Parameters

	password	String: Plain password to check.

Returns

True if matches.

to_dict()

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

Serializes user excluding password.

Returns

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)

  Handles client joining auction SocketIO room.
- auctions.handle_leave (data)

  Handles client leaving auction SocketIO room.
- auctions.get_auction_lock (auction_id)

  Returns the lock object for auction-specific synchronization.
- auctions.get_next_auction_to_close ()

  Finds the active auction ending soonest (end_date + overtime).
- auctions.get_next_auction_to_open ()

  Opens 'not_issued' auction to 'at_auction' status.
- auctions.close_auction_if_ended (auction_id, expected_overtime=0)

  Closes auction if ended, handling concurrent overtime changes.
- auctions.open_auction (auction_id)

  Opens 'not_issued' auction to 'at_auction' status.
- auctions.schedule_auction_closure (auction)

  Schedules auction closure job at end_date + overtime.
- auctions.schedule_auction_opening (auction)

  Schedules auction opening job at start_date (or immediate if past).
- auctions.schedule_next_auction ()

  Schedules the next auction closure or reschedules self in 1 minute.
- auctions.schedule_open_next_auction ()

  Schedules next 'not_issued' auction opening or reschedules self.
- auctions.start_scheduler (app)

  Initializes scheduler with app context binding.
- auctions.start_scheduler ()

  Initializes and starts BackgroundScheduler with initial auction scheduling.
- auctions.on_auction_update ()

  Triggers auction scheduling refresh after auction changes.

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 ()

  Retrieves all auctions with aggregated max prices and categories.
- Auctions.get_auction_details ()

  Retrieves complete details for a specific auction by ID.
- Auctions.create_auction ()

  Creates a new auction for the authenticated seller with photos and categories.
- Auctions.place_bid ()

  Places a bid on an active auction with validation and overtime extension.
- Auctions.get_user_own_auctions ()

  Retrieves all auctions owned by the authenticated seller.
- Auctions.get_user_auctions ()

  Retrieves active auctions for the authenticated user.
- Auctions.archived_auctions ()

  Retrieves archived (ended) auctions owned by the authenticated seller.
- Auctions.delete_auction ()

  Deletes an auction.

Variables

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

8.4 db_objects.py File Reference

Classes

- class db_objects.Auctions

  Auction model for buy-now auctions with bidding and overtime.
- class db_objects.AuctionPriceHistory

  Bid history tracking per auction/user with timestamps.
- class db_objects.Categories

  Auction category lookup table.
- class db_objects.CategoriesAuction

  Many-to-many junction between Auctions and Categories.
- class db_objects.PhotosItem

  Auction photos with main photo flag.
- class db_objects.Users

  User model for authentication and profiles.

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 ()

  Creates and configures the Flask application instance.

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 ()

  Returns a list of 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)

  Validates file extension against app config ALLOWED_EXTENSIONS.
- Uploads.upload_image ()

  Uploads image file, validates type/size, saves with UUID name.
- Uploads.serve_image (filename)

  Serves uploaded image file by filename.

Variables

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

8.9 routes/Users.py File Reference

Namespaces

- namespace Users

Functions

- Users.get_user_info ()

  Retrieves profile info for the authenticated user.
- Users.change_password ()

  Changes authenticated user's password after old password verification.
- Users.login ()

  Authenticates user and returns JWT access token.
- Users.register ()

  Registers a new user and returns JWT access token.
- Users.check_email ()

  Checks if an email is already registered.

Variables

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

Index

__tablename__
    db_objects.AuctionPriceHistory, 25
    db_objects.Auctions, 27
    db_objects.Categories, 29
    db_objects.CategoriesAuction, 30
    db_objects.PhotosItem, 32
    db_objects.Users, 34
_auction_locks
    auctions, 17

allowed_file
    Uploads, 20
APP
    auctions, 17
app
    main, 19
app_state, 6
    socketio, 6
app_state.py, 35
archived_auctions
    Auctions, 7
Auctions, 7
    archived_auctions, 7
    bp, 11
    create_auction, 7
    delete_auction, 8
    get_all_auctions, 8
    get_auction_details, 9
    get_user_auctions, 9
    get_user_own_auctions, 9
    place_bid, 11
auctions, 11
    _auction_locks, 17
    APP, 17
    close_auction_if_ended, 12
    get_auction_lock, 12
    get_next_auction_to_close, 13
    get_next_auction_to_open, 13
    handle_join, 13
    handle_leave, 14
    level, 17
    logger, 17
    on_auction_update, 14
    open_auction, 14
    schedule_auction_closure, 15
    schedule_auction_opening, 15
    schedule_next_auction, 15
    schedule_open_next_auction, 16
    SCHEDULER, 17
    start_scheduler, 16
auctions.py, 35

bp
    Auctions, 11
    Categories, 18
    Uploads, 21
    Users, 24

Categories, 17
    bp, 18
    get_all_categories, 18
category_name
    db_objects.Categories, 29
change_password
    Users, 22
check_email
    Users, 22
check_password
    db_objects.Users, 33
close_auction_if_ended
    auctions, 12
create_account_date
    db_objects.Users, 34
create_app
    main, 19
create_auction
    Auctions, 7

db
    db_objects, 18
db_objects, 18
    db, 18
db_objects.AuctionPriceHistory, 24
    __tablename__, 25
    id_auction, 25
    id_price_history, 25
    id_user, 25
    new_price, 25
    price_reprint_date, 25
    to_dict, 25
db_objects.Auctions, 26
    __tablename__, 27
    description, 27
    end_date, 27
    id_auction, 27
    id_seller, 27
    id_winner, 27
    overtime, 27
    start_date, 27
    start_price, 28
    status, 28
    title, 28
    to_dict, 27
db_objects.Categories, 28
    __tablename__, 29
    category_name, 29
    id_category, 29
    to_dict, 29
db_objects.CategoriesAuction, 29
    __tablename__, 30
    id_auction, 30
    id_category, 30

to_dict, 30
db_objects.PhotosItem, 31
__tablename__, 32
id_auction, 32
id_photo, 32
is_main_photo, 32
photo, 32
to_dict, 32
db_objects.py, 37
db_objects.Users, 33
__tablename__, 34
check_password, 33
create_account_date, 34
email, 34
first_name, 34
id_user, 34
last_name, 34
password, 35
phone_number, 35
to_dict, 34
debug
main, 19
delete_auction
Auctions, 8
description
db_objects.Auctions, 27
docs/mainpage.dox, 38

email
db_objects.Users, 34
end_date
db_objects.Auctions, 27

first_name
db_objects.Users, 34

get_all_auctions
Auctions, 8
get_all_categories
Categories, 18
get_auction_details
Auctions, 9
get_auction_lock
auctions, 12
get_next_auction_to_close
auctions, 13
get_next_auction_to_open
auctions, 13
get_user_auctions
Auctions, 9
get_user_info
Users, 22
get_user_own_auctions
Auctions, 9

handle_join
auctions, 13
handle_leave
auctions, 14
host
main, 19

id_auction
db_objects.AuctionPriceHistory, 25
db_objects.Auctions, 27
db_objects.CategoriesAuction, 30
db_objects.PhotosItem, 32
id_category
db_objects.Categories, 29
db_objects.CategoriesAuction, 30
id_photo
db_objects.PhotosItem, 32
id_price_history
db_objects.AuctionPriceHistory, 25
id_seller
db_objects.Auctions, 27
id_user
db_objects.AuctionPriceHistory, 25
db_objects.Users, 34
id_winner
db_objects.Auctions, 27
is_main_photo
db_objects.PhotosItem, 32

last_name
db_objects.Users, 34
level
auctions, 17
logger
auctions, 17
login
Users, 23

main, 19
app, 19
create_app, 19
debug, 19
host, 19
port, 19
scheduler, 19
True, 20
main.py, 38

new_price
db_objects.AuctionPriceHistory, 25

on_auction_update
auctions, 14
open_auction
auctions, 14
overtime
db_objects.Auctions, 27

password
db_objects.Users, 35
phone_number
db_objects.Users, 35
photo
db_objects.PhotosItem, 32

place_bid
Auctions, 11
port
main, 19
price_reprint_date
db_objects.AuctionPriceHistory, 25

register
Users, 23
routes/Auctions.py, 36
routes/Categories.py, 38
routes/Uploads.py, 38
routes/Users.py, 39

schedule_auction_closure
auctions, 15
schedule_auction_opening
auctions, 15
schedule_next_auction
auctions, 15
schedule_open_next_auction
auctions, 16
SCHEDULER
auctions, 17
scheduler
main, 19
serve_image
Uploads, 20
socketio
app_state, 6
start_date
db_objects.Auctions, 27
start_price
db_objects.Auctions, 28
start_scheduler
auctions, 16
status
db_objects.Auctions, 28

title
db_objects.Auctions, 28
to_dict
db_objects.AuctionPriceHistory, 25
db_objects.Auctions, 27
db_objects.Categories, 29
db_objects.CategoriesAuction, 30
db_objects.PhotosItem, 32
db_objects.Users, 34
True
main, 20

upload_image
Uploads, 21
Uploads, 20
allowed_file, 20
bp, 21
serve_image, 20
upload_image, 21
Users, 21
bp, 24
change_password, 22
check_email, 22
get_user_info, 22
login, 23
register, 23