jerrycan

Release 0.1.0

Hermann Krumrey

Sep 29, 2020

CONTENTS

Contents:

CHAPTER

ONE

JERRYCAN PACKAGE

1.1 Subpackages

1.1.1 jerrycan.background package

Submodules

jerrycan.background.telegram module

jerrycan.background.telegram.**telegram_whoami**()
Specifies the background behaviour of the telegram bot By default, the bot listens to /whoami messages and answers with the telegram chat ID :return: None

Module contents

1.1.2 jerrycan.db package

Submodules

jerrycan.db.ApiKey module

class jerrycan.db.ApiKey.**ApiKey**(**args*, ***kwargs*)
Bases: *jerrycan.db.ModelMixin.ModelMixin*, sqlalchemy.ext.declarative.api.Model

Model that describes the 'api_keys' SQL table An ApiKey is used for API access using HTTP basic auth

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

creation_time: int
The time at which this API key was created as a UNIX timestamp

has_expired() → bool
Checks if the API key has expired. API Keys expire after 30 days :return: True if the key has expired, False otherwise

id

key_hash: str
The hash of the API key

user: *jerrycan.db.User.User*
The user associated with this API key

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

verify_key(*key: str*) → bool
Checks if a given key is valid :param key: The key to check :return: True if the key is valid, False otherwise

jerrycan.db.ModelMixin module

class jerrycan.db.ModelMixin.**ModelMixin**
Bases: object

A mixin class that specifies a couple of methods all database models should implement

id = Column(None, Integer(), table=None, primary_key=True, nullable=False)
The ID is the primary key of the table and increments automatically

jerrycan.db.TelegramChatId module

class jerrycan.db.TelegramChatId.**TelegramChatId**(**args*, ***kwargs*)
Bases: *jerrycan.db.ModelMixin.ModelMixin*, sqlalchemy.ext.declarative.api.Model

Model that describes the 'telegram_chat_ids' SQL table Maps telegram chat ids to users

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

chat_id: **str**
The telegram chat ID

id

send_message(*message_text: str*)
Sends a message to the telegram chat :param message_text: The message text to send :return: None

user: *jerrycan.db.User.User*
The user associated with this telegram chat ID

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

jerrycan.db.User module

class jerrycan.db.User.**User**(**args*, ***kwargs*)
Bases: *jerrycan.db.ModelMixin.ModelMixin*, sqlalchemy.ext.declarative.api.Model

Model that describes the 'users' SQL table A User stores a user's information, including their email address, username and password hash

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

api_keys: List[ApiKey]
API keys for this user

confirmation_hash: str
The account's confirmation hash. This is the hash of a key emailed to the user. Only once the user follows the link in the email containing the key will their account be activated

confirmed: bool
The account's confirmation status. Logins should be impossible as long as this value is False.

email: str
The user's email address

get_id() → str
Method required by flask-login :return: The user's ID as a unicode string

id

property is_active
Property required by flask-login :return: True

property is_anonymous
Property required by flask-login :return: True if the user is not confirmed, False otherwise

property is_authenticated
Property required by flask-login :return: True if the user is confirmed, False otherwise

password_hash: str
The user's hashed password, salted and hashed.

telegram_chat_id: Optional[TelegramChatId]
Telegram chat ID for the user if set up

username: str
The user's username

verify_confirmation(*confirmation_key: str*) → bool
Verifies a confirmation key against the confirmation hash :param confirmation_key: The key to check :return: True if the key matches, False otherwise

verify_password(*password: str*) → bool
Verifies a password against the password hash :param password: The password to check :return: True if the password matches, False otherwise

Module contents

1.1.3 jerrycan.routes package

Subpackages

jerrycan.routes.api package

Submodules

jerrycan.routes.api.user_management module

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

Module contents

Submodules

jerrycan.routes.decorators module

jerrycan.routes.decorators.**api**(*func: Callable*) → Callable
Decorator that handles common API patterns and ensures that the JSON response will always follow a certain pattern :param func: The function to wrap :return: The wrapper function

jerrycan.routes.decorators.**api_login_required**(*func: Callable*) → Callable
Decorator to make unauthorized API calls respond with JSON properly :param func: The function to wrap :return: The wrapped function

jerrycan.routes.static module

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

jerrycan.routes.user_management module

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

Module contents

jerrycan.routes.**blueprint_generators: List[Tuple[Callable[[str], flask.blueprints.Bluepri**
Defines the functions used to create the various blueprints as well as their names

1.1.4 jerrycan.test package

Subpackages

jerrycan.test.db package

Submodules

jerrycan.test.db.TestApiKey module

class jerrycan.test.db.TestApiKey.**TestApiKey**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests the ApiKey database model

test_equality()
Tests checking equality for model objects :return: None

test_expiration()
Tests if the expiration of API keys works correctly :return: None

test_hashing()
Tests using the model objects as keys in a dictionary :return: None

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

test_repr()
Tests the __repr__ method of the model class :return: None

test_string_representation()
Tests the string representation of the model :return: None

test_verifying_key()
Tests verifying an api key :return: None

jerrycan.test.db.TestModelMixin module

class jerrycan.test.db.TestModelMixin.**TestModelMixin**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests the ModelMixin class

test_enum_attributes()
Tests if enum attributes are handled correctly :return: None

test_json_representation()
Tests the JSOn representation of a ModelMixin :return: None

jerrycan.test.db.TestTelegramChatId module

class jerrycan.test.db.TestTelegramChatId.**TestTelegramChatId**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests the TelegramChatId database model

test_equality()
Tests checking equality for model objects :return: None

test_hashing()
Tests using the model objects as keys in a dictionary :return: None

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

test_repr()
Tests the __repr__ method of the model class :return: None

test_sending_message()
Tests sending a message :return: None

test_string_representation()
Tests the string representation of the model :return: None

jerrycan.test.db.TestUser module

class jerrycan.test.db.TestUser.**TestUser**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests the User database model

test_equality()
Tests checking equality for model objects :return: None

test_flask_properties()
Tests if the flask_login properties work as expected :return: None

test_hashing()
Tests using the model objects as keys in a dictionary :return: None

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

test_json_with_telegram_chat_id()
Tests whether or not including children can cause recursion errors :return: None

test_repr()
Tests the __repr__ method of the model class :return: None

test_string_representation()
Tests the string representation of the model :return: None

test_verifying_password()
Tests verifying the password of a user :return: None

Module contents

jerrycan.test.misc package

Submodules

jerrycan.test.misc.TestApiCalls module

class jerrycan.test.misc.TestApiCalls.**TestApiCalls**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests various API calls

test_exception_in_api_route()
Tests if an exeption on an API route is correctly wrapped in a JSON response :return: None

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

test_non_base64_header()
Tests using a header that's not base64 encoded :return: None

test_random_exception()
Tests that the API routes catch any Exceptions without issue :return: None

test_unauthorized_call()
Tests and unauthorized API call :return: None

test_using_non_json_data()
Tests sending the data as something that's not JSON :return: None

jerrycan.test.misc.TestConfig module

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

Class that tests the config class

test_building_base_url()
Tests building the base URL :return: None

test_db_config()
Tests the database configuration :return: None

test_dumping_environment_variables()
Tests dumping environment variables :return: None

test_environment_variables_definitions()
Tests the definition of the environent variables :return: None

test_initializing_telegram_bot_connection()
'Tests' initializing the telegram bot connection :return: None

test_version()
Tests if the version is fetched correctly :return: None

jerrycan.test.misc.TestErrorHandling module

class jerrycan.test.misc.TestErrorHandling.**TestErrorHandling**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests the flask error handling

test_404()
Tests if a 404 error is handled correctly :return: None

test_api_exception()
Tests if ApiExceptions in the API are handled correctly :return: None

test_exception()
Tests if unexpected exceptions are caught correctly :return: None

jerrycan.test.misc.TestInitialization module

class jerrycan.test.misc.TestInitialization.**TestInitialization**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Tests the initialization of the flask application

test_db_connection_error()
Tests if a database connection error is handled correctly :return: None

test_missing_environment_variables()
Tests if missing environment variables are detected correctly :return: None

test_missing_required_template()
Tests if missing template files are detected correctly :return: None

test_no_extra_jinja()
Tests if not providing any additional jinja variables works as intended :return: None

jerrycan.test.misc.TestTelegramBackground module

class jerrycan.test.misc.TestTelegramBackground.**TestTelegramBackground**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Tests the telegram background functions

test_whoami()
Tests the whoami functionality :return: None

jerrycan.test.misc.TestWsgi module

class jerrycan.test.misc.TestWsgi.**TestWsgi**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Tests the WSGI server

test_starting_and_stopping_wsgi_server()
Tests starting and stopping the WSGI server :return: None

test_starting_background_tasks()
Tests starting background tasks :return: None

Module contents

jerrycan.test.routes package

Subpackages

jerrycan.test.routes.api package

Submodules

jerrycan.test.routes.api.TestApiKeyRoute module

class jerrycan.test.routes.api.TestApiKeyRoute.**TestApiKeyRoute**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests API-key related features

test_requesting_api_key()
Tests requesting an API key :return: None

test_requesting_invalid_api_keys()
Tests requesting API keys with invalid data :return: None

test_revoking_api_key()
Tests revoking an API key :return: None

test_unsuccessfully_revoking_api_key()
Tests unsuccessfully revoking an API key :return: None

Module contents

Submodules

jerrycan.test.routes.TestForgotRoute module

class jerrycan.test.routes.TestForgotRoute.**TestForgotRoute**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests password reset features

test_invalid_recaptcha()
Tests that invalid ReCaptcha responses are handled correctly :return: None

test_page_get()
Tests getting the page :return: None

test_resetting_password()
Tests successfully resetting a password :return: None

test_unsuccessfully_resetting_password()
Tests unsuccessfully resetting a password :return: None

jerrycan.test.routes.TestLoginRoute module

class jerrycan.test.routes.TestLoginRoute.**TestLoginRoute**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests log-in features

test_invalid_login_attempts()
Tests trying to log in with invalid credentials etc :return: None

test_logging_in_and_out()
Tests logging in successfully, then once more, then logging out :return: None

test_logging_in_with_email()
Tests logging in with an email address :return: None

test_page_get()
Tests getting the page :return: None

jerrycan.test.routes.TestProfileRoute module

class jerrycan.test.routes.TestProfileRoute.**TestProfileRoute**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests profile features

test_changing_password()
Tests changing a password :return: None

test_page_get()
Tests getting the page :return: None

test_unsuccessful_password_change()
Tests unsuccessfully changing a password :return: None

test_unsuccessful_user_delete()
Tests unsuccessfully deleting a user :return: None

test_user_delete()
Tests deleting a user :return: None

jerrycan.test.routes.TestRegisterRoute module

class jerrycan.test.routes.TestRegisterRoute.**TestRegisterRoute**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests registration features

test_confirming()
Tests confirming a user :return: None

test_invalid_confirm()
Tests invalid confirmations :return: None

test_invalid_recaptcha()
Tests that invalid ReCaptcha responses are handled correctly :return: None

test_invalid_registrations()
Tests registering using invalid parameters :return: None

test_page_get()
Tests getting the page :return: None

test_registering_user()
Tests registering a new user :return: None

jerrycan.test.routes.TestRegisterTelegramRoute module

class jerrycan.test.routes.TestRegisterTelegramRoute.**TestRegisterTelegramRoute**(*methodName='ru*
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests the route used to register telegram chat IDs

test_registering()
Tests registering a telegram chat ID :return: None

jerrycan.test.routes.TestStaticRoutes module

class jerrycan.test.routes.TestStaticRoutes.**TestStaticRoutes**(*methodName='runTest'*)
Bases: jerrycan.test.TestFramework._TestFramework

Class that tests static pages

test_get_about()
Tests getting the about page :return: None

test_get_index()
Tests getting the index page :return: None

test_get_privacy()
Tests getting the privacy page :return: None

Module contents

Submodules

jerrycan.test.TestFramework module

jerrycan.test.mocks module

Module contents

1.2 Submodules

1.3 jerrycan.Config module

class jerrycan.Config.**Config**
Bases: object

Class that keeps track of configuration data The class attributes should only be called after running load_config

API_VERSION: str = '1'
The API Version

BEHIND_PROXY: bool
Whether or not the site is being served by a reverse proxy like nginx.

DB_MODE: str
The database mode (for example 'sqlite' or 'mysql')

DB_URI: str
The database URI to use for database operations

DEBUG_LOGGING_PATH: str
The path to the debug logging path

DOMAIN_NAME: str
The domain name of the website

FLASK_SECRET: str
The flask secret key

HTTP_PORT: int
The port to use when serving the flask application

LOGGING_PATH: str
The path to the logging file

MAX_API_KEY_AGE: int = 2592000
The maximum age for API keys

MAX_USERNAME_LENGTH: int = 12
The maximum length of usernames

MIN_USERNAME_LENGTH: int = 1
The Minimum length for usernames

RECAPTCHA_SECRET_KEY: str
Google ReCaptcha secret key for bot detection

RECAPTCHA_SITE_KEY: str
Google ReCaptcha site key for bot detection

REQUIRED_TEMPLATES: Dict[str, str] = {'about': 'static/about.html', 'error_page': 'st
Specifies required template files

SENTRY_DSN: str
The sentry DSN used for error logging

SMTP_ADDRESS: str
The SMPT Address used for sending emails

SMTP_HOST: str
The SMPT Host used for sending emails

SMTP_PASSWORD: str
The SMPT Password used for sending emails

SMTP_PORT: int
The SMPT Port used for sending emails

STRINGS: Dict[str, str] = {'401_message': 'You are not logged in', '500_message': 'T
Dictionary that defines various strings used in the application. Makes it easier to use custom phrases.

TELEGRAM_API_KEY: str
API key for a telegram bot

TELEGRAM_BOT_CONNECTION: bokkichat.connection.impl.TelegramBotConnection.TelegramBotCo
Telegram bot connection

TELEGRAM_WHOAMI: bool
Whether or not the telegram WHOAMI background thread will be started

TEMPLATE_EXTRAS: Dict[str, Callable[], Dict[str, Any]]] = {'about': <function Config.
This can be used to provide the template rendering engine additional parameters, which may be necessary when adding UI elements. This is done with functions that don't expect any input and return a dictionary of keys and values to be passed to the template rendering engine

TESTING: bool
Whether or not testing is enabled

VERBOSITY: int
The verbosity level of the logging when printing to the console

VERSION: str
The current version of the application

WARNING_LOGGING_PATH: str
The path to the logging path for WARNING messages

classmethod base_url() → str

Returns The base URL of the website

classmethod dump_env_variables(*path: Optional[str] = None*)
Dumps all environment variables used by this application to a file :param path: The path to the file to which to dump the content.

If this is None, the file contents will be printed.

Returns None

classmethod ensure_environment_variables_present()
Makes sure that all required environment variables have been set. If this is not the case, the app will exit. :return: None

classmethod environment_variables() → Dict[str, List[str]]
Specifies required and optional environment variables :return: The specified environment variables in two lists in

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

classmethod initialize_telegram()
Initializes the telegram bot connection :return: None

classmethod load_config(*root_path: str*, *module_name: str*, *sentry_dsn: str*)
Loads the configuration from environment variables :param root_path: The root path of the application :param module_name: The name of the project's module :param sentry_dsn: The sentry DSN used for error logging :return: None

1.4 jerrycan.base module

jerrycan.base.**app = <Flask 'jerrycan.base'>**
The Flask App

jerrycan.base.**db = <SQLAlchemy engine=None>**
The SQLAlchemy database connection

jerrycan.base.**login_manager = <flask_login.login_manager.LoginManager object>**
The Flask-Login Login Manager

1.5 jerrycan.enums module

class jerrycan.enums.**AlertSeverity**(*value*)
Bases: enum.Enum

Enumeration that defines the various levels of severity an alert can have

DANGER = 'danger'
Translates to a red alert

INFO = 'info'
Translates to a blue alert

SUCCESS = 'success'
Translates to a green alert

WARNING = 'warning'
Translates to a yellow alert

1.6 jerrycan.exceptions module

exception jerrycan.exceptions.**ApiException**(*reason: str*, *status_code: int*)
Bases: Exception

Api raised when an API-related exception occurs

__init__(*reason: str*, *status_code: int*)
Initializes the exception :param reason: The reason the API Exception was raised :param status_code: The status code associated with the exception

1.7 jerrycan.initialize module

jerrycan.initialize.**CREATED_BLUEPRINTS = []**
Keeps track of created blueprint names. This is necessary for unit testing with nose, because duplicate blueprint names will cause errors.

jerrycan.initialize.**init_flask**(*module_name: str*, *sentry_dsn: str*, *root_path: str*, *config: Type[jerrycan.Config.Config]*, *models: List[Type[sqlalchemy.ext.declarative.api.Model]]*, *blueprint_generators: List[Tuple[Callable[[str], flask.blueprints.Blueprint], str]]*, *extra_jinja_vars: Optional[Dict[str, Any]] = None*)
Initializes the flask application :param module_name: The name of the module :param sentry_dsn: The sentry DSN used for error logging :param root_path: The root path of the flask application :param config: The Config class to use for configuration :param models: The database models to create :param blueprint_generators: Tuples that contain a function that generates

a blueprint and the name of the blueprint

Parameters **extra_jinja_vars** – Any extra variables to pass to jinja

Returns None

1.8 jerrycan.wsgi module

jerrycan.wsgi.**start_server**(*config: Type[jerrycan.Config.Config]*, *task_definitions: Dict[str, Tuple[int, Callable]]*)
Starts the flask application using a cheroot WSGI server :param config: The configuration to use :param task_definitions: The background tasks, consisting of:

- the name of the task
- the time to wait before running the task again
- a function that takes no arguments that executes the task

Returns None

1.9 Module contents

CHAPTER

TWO

JERRYCAN

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

j

INDEX

Symbols

A

B

C

D

E

F

G

H

I

P

R

S

T

U

V

W