otaku-info-web

Release 0.1.0

Hermann Krumrey

Apr 23, 2020

CONTENTS

Contents:

CHAPTER

ONE

OTAKU_INFO_WEB

1.1 otaku_info_web package

1.1.1 Subpackages

otaku_info_web.background package

Submodules

otaku_info_web.background.anilist module

otaku_info_web.background.anilist.**fetch_anilist_data**()
Retrieves all entries on the anilists of all users that provided an anilist username :return: None

otaku_info_web.background.anilist.**fetch_media_id**(*anilist_entry:* otaku_info_web.utils.anilist.AnilistItem.AnilistUserItem, *media_items:* *Dict[Tuple[str,* otaku_info_web.utils.enums.MediaType, otaku_info_web.utils.enums.MediaSubType, *str],* otaku_info_web.db.MediaItem.MediaItem*]*, *media_ids:* *Dict[Tuple[*otaku_info_web.utils.enums.ListService, *int],* otaku_info_web.db.MediaId.MediaId*]*, *media_item:* *Optional[*otaku_info_web.db.MediaId.MediaId*]* = *None*) → Tuple[Optional[Tuple[*otaku_info_web.utils.enums.ListService*, int]], Optional[*otaku_info_web.db.MediaItem.MediaItem*]]
Retrieves an existing media ID based on anilist data :param anilist_entry: The anilist entry to use :param media_items: The preloaded media items :param media_ids: The preloaded media IDs :param media_item: Optional media item associated with the ID.

If not provided, will figure out using anilist data

Returns The media ID, or None if none exists

otaku_info_web.background.anilist.**fetch_media_item**(*anilist_entry:* otaku_info_web.utils.anilist.AnilistItem.AnilistUserItem, *media_items:* *Dict[Tuple[str,* otaku_info_web.utils.enums.MediaType, otaku_info_web.utils.enums.MediaSubType, *str],* otaku_info_web.db.MediaItem.MediaItem*]*) → Tuple[Tuple[str, *otaku_info_web.utils.enums.MediaType*, *otaku_info_web.utils.enums.MediaSubType*, str], Optional[*otaku_info_web.db.MediaItem.MediaItem*]]

Retrieves an existing media item based on anilist data :param anilist_entry: The anilist entry to use :param media_items: The preloaded media items :return: The media item, or None if none exists

otaku_info_web.background.anilist.**load_existing**() → Tuple[Dict[Tuple[str, *otaku_info_web.utils.enums.MediaType*, *otaku_info_web.utils.enums.MediaSubType*, str], *otaku_info_web.db.MediaItem.MediaItem*], Dict[Tuple[*otaku_info_web.utils.enums.ListService*, int], *otaku_info_web.db.MediaId.MediaId*], Dict[Tuple[int, int], *otaku_info_web.db.MediaUserState.MediaUserState*], Dict[Tuple[str, int, *otaku_info_web.utils.enums.ListService*, *otaku_info_web.utils.enums.MediaType*], *otaku_info_web.db.MediaList.MediaList*], Dict[Tuple[int, int], *otaku_info_web.db.MediaListItem.MediaListItem*]]

Loads current database contents, mapped to unique identifer tuples :return: The database contents

otaku_info_web.background.anilist.**update_media_entries**(*anilist_data:* *Dict[*otaku_info_web.db.ServiceUsername.ServiceUs *Dict[*otaku_info_web.utils.enums.MediaType, *List[*otaku_info_web.utils.anilist.AnilistItem.AnilistU *media_items:* *Dict[Tuple[str,* otaku_info_web.utils.enums.MediaType, otaku_info_web.utils.enums.MediaSubType, *str],* otaku_info_web.db.MediaItem.MediaItem*],* *media_ids:* *Dict[Tuple[*otaku_info_web.utils.enums.ListService, *int],* otaku_info_web.db.MediaId.MediaId*]*)

Updates the media entries and anilist IDs :param anilist_data: The anilist data to store :param media_items: The preloaded media items :param media_ids: The preloaded media IDs :return: None

otaku_info_web.background.anilist.**update_media_id**(*new_data:* otaku_info_web.utils.anilist.AnilistItem.AnilistUserItem, *media_item:* otaku_info_web.db.MediaItem.MediaItem, *existing:* *Optional[*otaku_info_web.db.MediaId.MediaId*]*) → *otaku_info_web.db.MediaId.MediaId*

Updates/Creates a MediaId database entry based on anilist data :param new_data: The anilist data to use :param

media_item: The media item associated with the ID :param existing: The existing database entry. If None, will be created :return: The updated/created MediaId object

otaku_info_web.background.anilist.**update_media_item**(*new_data:* otaku_info_web.utils.anilist.AnilistItem.AnilistUserItem, *existing:* *Optional[*otaku_info_web.db.MediaItem.MediaItem*]*) → *otaku_info_web.db.MediaItem.MediaItem*

Updates or creates MediaItem database entries based on anilist data :param new_data: The new anilist data :param existing: The existing database entry. If None, will be created :return: The updated/created MediaItem object

otaku_info_web.background.anilist.**update_media_lists**(*anilist_data:* *Dict[*otaku_info_web.db.ServiceUsername.ServiceUser*Dict[*otaku_info_web.utils.enums.MediaType*, List[*otaku_info_web.utils.anilist.AnilistItem.AnilistUse*media_items: Dict[Tuple[str,* otaku_info_web.utils.enums.MediaType*,* otaku_info_web.utils.enums.MediaSubType*, str],* otaku_info_web.db.MediaItem.MediaItem*], media_ids:* *Dict[Tuple[*otaku_info_web.utils.enums.ListService*, int],* otaku_info_web.db.MediaId.MediaId*], media_user_states:* *Dict[Tuple[int, int],* otaku_info_web.db.MediaUserState.MediaUserState*], media_lists: Dict[Tuple[str, int,* otaku_info_web.utils.enums.ListService*,* otaku_info_web.utils.enums.MediaType*],* otaku_info_web.db.MediaList.MediaList*], media_list_items:* *Dict[Tuple[int, int],* otaku_info_web.db.MediaListItem.MediaListItem*]*)

Updates the database for anilist user lists. This includes custom anilist lists. :param anilist_data: The anilist data to enter into the database :param media_items: Preloaded media items :param media_ids: Preloaded media IDs :param media_user_states: The current media user states in the database :param media_lists: The media lists currently in the database :param media_list_items: The media list items currently in the database :return: None

otaku_info_web.background.anilist.**update_media_user_entries**(*anilist_data: Dict[otaku_info_web.db.ServiceUsername.Se Dict[otaku_info_web.utils.enums.MediaType List[otaku_info_web.utils.anilist.AnilistItem.A media_items: Dict[Tuple[str, otaku_info_web.utils.enums.MediaType, otaku_info_web.utils.enums.MediaSubType, str], otaku_info_web.db.MediaItem.MediaItem], media_ids: Dict[Tuple[otaku_info_web.utils.enums.ListS int], otaku_info_web.db.MediaId.MediaId], media_user_states: Dict[Tuple[int, int], otaku_info_web.db.MediaUserState.MediaUs*)

Updates the individual users' current state for media items in thei ranilist account. :param anilist_data: The anilist data to enter into the database :param media_items: Preloaded media items :param media_ids: Preloaded media IDs :param media_user_states: Preloaded media user states :return: None

otaku_info_web.background.anilist.**update_media_user_state**(*new_data: otaku_info_web.utils.anilist.AnilistItem.AnilistU media_id: otaku_info_web.db.MediaId.MediaId, user: puffotter.flask.db.User.User, existing: Optional[otaku_info_web.db.MediaUserState.Medi* → *otaku_info_web.db.MediaUserState.MediaUserS*)

Updates or creates a MediaUserState entry in the database :param new_data: The new anilist data :param media_id: The media ID of the anilist media item :param user: The user associated with the data :param existing: The existing database entry. If None, will be created :return: The updated/created MediaUserState object

otaku_info_web.background.manga_chapters module

otaku_info_web.background.manga_chapters.**update_manga_chapter_guesses**()

Updates the manga chapter guesses :return: None

otaku_info_web.background.mangadex module

otaku_info_web.background.mangadex.**create_anilist_media_item**(*anilist_id: int*) → Optional[*otaku_info_web.db.MediaItem.Media*

Creates an anilist media item using an anilist ID, fetching the data using the anilist API :param anilist_id: The anilist ID of the media :return: The generated Media Item

otaku_info_web.background.mangadex.**load_db_content**() → Tuple[Dict[str, *otaku_info_web.db.MediaId.MediaId*], Dict[int, List[*otaku_info_web.utils.enums.ListService*]]]

Loads the existing data from the database. By doing this as few times as possible, we can greatly improve performance :return: The anilist IDs, The mangadex IDs mapped to other existing IDs

otaku_info_web.background.mangadex.**load_id_mappings**()
Goes through mangadex IDs sequentially and stores ID mappings for these entries if found :return: None

otaku_info_web.background.mangadex.**store_ids**(*existing_ids: Dict[int, List[otaku_info_web.utils.enums.ListService]]*, *anilist_ids: Dict[str, otaku_info_web.db.MediaId.MediaId]*, *mangadex_id: int*, *other_ids: Dict[otaku_info_web.utils.enums.ListService, str]*)
Stores the fetched IDs in the database :param existing_ids: A dictionary mapping mangadex IDs to existing list service types

Parameters

- **anilist_ids** – Dictionary mapping anilist IDs to media IDs
- **mangadex_id** – The mangadex ID
- **other_ids** – The other IDs

Returns None

otaku_info_web.background.notifications module

otaku_info_web.background.notifications.**send_new_manga_chapter_notifications**()
Sends out telegram notifications for manga chapter updates :return: None

otaku_info_web.background.telegram module

otaku_info_web.background.telegram.**handle_whoami_requests**()
Handles whoami requests to the telegram bot :return: None

Module contents

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

otaku_info_web.db package

Submodules

otaku_info_web.db.MangaChapterGuess module

class otaku_info_web.db.MangaChapterGuess.**MangaChapterGuess**(**args*, ***kwargs*)
Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Database model that keeps track of manga chapter guesses.

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

guess
The actual guess for the most current chapter of the manga series

id

last_update
Timestamp from when the guess was last updated

media_id
The media ID referenced by this manga chapter guess

media_id_id
The ID of the media ID referenced by this manga chapter guess

update()
Updates the manga chapter guess (if the latest guess is older than an hour) :return: None

otaku_info_web.db.MediaId module

class otaku_info_web.db.MediaId.**MediaId**(**args*, ***kwargs*)
Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Database model for media IDs. These are used to map media items to their corresponding external IDS on external sites.

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

id

manga_chapter_guesses

media_item
The media item referenced by this ID

media_item_id
The ID of the media item referenced by this ID

media_user_states

service
The service for which this object represents an ID

property service_icon

Returns The path to the service's icon file

service_id
The ID of the media item on the external service

property service_url

Returns The URL to the series for the given service

otaku_info_web.db.MediaItem module

class otaku_info_web.db.MediaItem.**MediaItem**(**args*, ***kwargs*)

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

Database model for media items. These model a generic, site-agnostic representation of a series.

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

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

cover_url

An URL to a cover image of the media item

english_title

The English title of the media item

id

latest_release

The latest release chapter/episode for this media item

media_ids

media_subtype

The subtype (for example, TV short, movie oneshot etc)

media_type

The media type of the list item

releasing_state

The current releasing state of the media item

romaji_title

The Japanese title of the media item written in Romaji

property title

Returns The default title for the media item.

otaku_info_web.db.MediaList module

class otaku_info_web.db.MediaList.**MediaList**(**args*, ***kwargs*)

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

Database model for user-specific media lists.

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

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

id

media_list_items

media_type

The media type for this list

name

The name of this list

service
The service for which this list applies to

user
The user associated with this list

user_id
The ID of the user associated with this list

otaku_info_web.db.MediaListItem module

class otaku_info_web.db.MediaListItem.**MediaListItem**(**args*, ***kwargs*)
Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Database model for media list items. This model maps MediaLists and MediaUserStates

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

id

media_list
The media list this list item is a part of

media_list_id
The ID of the media list this list item is a part of

media_user_state
The media user state this list item references

media_user_state_id
The ID of the media user state this list item references

otaku_info_web.db.MediaNotification module

class otaku_info_web.db.MediaNotification.**MediaNotification**(**args*, ***kwargs*)
Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Database model that stores a media notification for a user

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

id

last_update
The last update value sent to the user

media_user_state
The media user state this notification references

media_user_state_id
The ID of the media user state this notification references

otaku_info_web.db.MediaRelation module

otaku_info_web.db.MediaUserState module

class otaku_info_web.db.MediaUserState.**MediaUserState**(**args*, ***kwargs*)

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

Database model that keeps track of a user's entries on external services for a media item

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

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

consuming_state

The current consuming state of the user for this media item

id

media_id

The media ID referenced by this user state

media_id_id

The ID of the media ID referenced by this user state

media_list_items

media_notifications

progress

The user's current progress consuming the media item

score

The user's score for the references media item

user

The user associated with this user state

user_id

The ID of the user associated with this user state

otaku_info_web.db.NotificationSetting module

class otaku_info_web.db.NotificationSetting.**NotificationSetting**(**args*, ***kwargs*)

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

Database model that stores notification settings for a user

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

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

id

notification_type

The notification type

user

The user associated with this notification setting

user_id
The ID of the user associated with this notification setting

value
Whether or not the notification is active or not

otaku_info_web.db.ServiceUsername module

class otaku_info_web.db.ServiceUsername.**ServiceUsername**(**args*, ***kwargs*)
Bases: puffotter.flask.db.ModelMixin.ModelMixin, sqlalchemy.ext.declarative.api.Model

Database model that stores an external service username for a user

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

id

service
The external service this item is a username for

user
The user associated with this service username

user_id
The ID of the user associated with this service username

username
The service username

otaku_info_web.db.TelegramChatId module

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

Database model that stores a telegram chat ID for a user

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

chat_id
The telegram chat ID

id

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

user
The user associated with this telegram chat ID

user_id
The ID of the user associated with this telegram chat ID

Module contents

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

otaku_info_web.routes package

Subpackages

otaku_info_web.routes.api package

Submodules

otaku_info_web.routes.api.media_api module

otaku_info_web.routes.api.media_api.**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

otaku_info_web.routes.external_service module

otaku_info_web.routes.external_service.**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

otaku_info_web.routes.manga module

otaku_info_web.routes.manga.**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

otaku_info_web.routes.media module

otaku_info_web.routes.media.**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

otaku_info_web.routes.notifications module

otaku_info_web.routes.notifications.**define_blueprint**(*blueprint_name: str*) → flask.blueprints.Blueprint

Defines a Blueprint that handles notifications for users :param blueprint_name: The name of the blueprint :return: The blueprint

Module contents

otaku_info_web.routes.**blueprint_generators: List[Tuple[Callable[str, flask.blueprints.Blue**

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

otaku_info_web.test package

Subpackages

otaku_info_web.test.db package

Submodules

otaku_info_web.test.db.TestApiKey module

class otaku_info_web.test.db.TestApiKey.**TestApiKey**(*methodName='runTest'*)

Bases: otaku_info_web.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

otaku_info_web.test.db.TestMangaChapterGuess module

class otaku_info_web.test.db.TestMangaChapterGuess.**TestMangaChapterGuess**(*methodName='runTest'*)
Bases: otaku_info_web.test.TestFramework._TestFramework

Class that tests the MangaChapterGuess database model

static generate_sample_guess() → Tuple[*otaku_info_web.db.MediaItem.MediaItem*, *otaku_info_web.db.MediaId.MediaId*, *otaku_info_web.db.MangaChapterGuess.MangaChapterGuess*]
Generates a sample chapter guess :return: The media item, media id and chapter guess

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

test_guessing_latest_chapter()
Tests guessing the latest chapter :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_uniqueness()
Tests if the uniqueness of the model is handled properly :return: None

otaku_info_web.test.db.TestMediaId module

class otaku_info_web.test.db.TestMediaId.**TestMediaId**(*methodName='runTest'*)
Bases: otaku_info_web.test.TestFramework._TestFramework

Class that tests the MediaId database model

static generate_sample_media_id() → Tuple[*otaku_info_web.db.MediaItem.MediaItem*, *otaku_info_web.db.MediaId.MediaId*]
Generates a media id :return: The media item and media id

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

test_generating_service_url()
Tests generating the generating of service URLs :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_uniqueness()
Tests if the uniqueness of the model is handled properly :return: None

otaku_info_web.test.db.TestMediaItem module

class otaku_info_web.test.db.TestMediaItem.**TestMediaItem**(*methodName='runTest'*)
Bases: otaku_info_web.test.TestFramework._TestFramework

Class that tests the MediaItem database model

static generate_sample_media_item() → *otaku_info_web.db.MediaItem.MediaItem*
Generates a sample media item :return: The media item

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

test_title()
Tests the title attribute of the media item :return: None

test_uniqueness()
Tests that a mediaitem can only exist once :return: None

otaku_info_web.test.db.TestMediaList module

class otaku_info_web.test.db.TestMediaList.**TestMediaList**(*methodName='runTest'*)
Bases: otaku_info_web.test.TestFramework._TestFramework

Class that tests the MediaList database model

generate_sample_media_list() → Tuple[*otaku_info_web.db.MediaList.MediaList*, puffotter.flask.db.User.User]
Generates a sample media item :return: The media list and the associated user

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

test_uniqueness()
Tests if the uniqueness of the model is handled properly :return: None

otaku_info_web.test.db.TestMediaListItem module

class otaku_info_web.test.db.TestMediaListItem.**TestMediaListItem**(*methodName='runTest'*)
Bases: otaku_info_web.test.TestFramework._TestFramework

Class that tests the MediaListItem database model

generate_sample_media_list_item() → Tuple[*otaku_info_web.db.MediaListItem.MediaListItem*, *otaku_info_web.db.MediaList.MediaList*, *otaku_info_web.db.MediaUserState.MediaUserState*, puffotter.flask.db.User.User, *otaku_info_web.db.MediaItem.MediaItem*, *otaku_info_web.db.MediaId.MediaId*]
Generates a media list item :return: The media list item, media list, media user state, user, media item and media id

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

test_uniqueness()
Tests if the uniqueness of the model is handled properly :return: None

otaku_info_web.test.db.TestMediaRelation module

otaku_info_web.test.db.TestMediaUserState module

class otaku_info_web.test.db.TestMediaUserState.**TestMediaUserState**(*methodName='runTest'*)
Bases: otaku_info_web.test.TestFramework._TestFramework

Class that tests the MediaUserState database model

generate_sample_media_user_state() → Tuple[*otaku_info_web.db.MediaUserState.MediaUserState*, puffotter.flask.db.User.User, *otaku_info_web.db.MediaItem.MediaItem*, *otaku_info_web.db.MediaId.MediaId*]
Generates a media user state :return: The media user state, user, media item and media id

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

test_uniqueness()
Tests if the uniqueness of the model is handled properly :return: None

otaku_info_web.test.db.TestServiceUsername module

class otaku_info_web.test.db.TestServiceUsername.**TestServiceUsername**(*methodName='runTest'*)
Bases: otaku_info_web.test.TestFramework._TestFramework

Class that tests the ServiceUsername database model

generate_sample_service_username() → Tuple[*otaku_info_web.db.ServiceUsername.ServiceUsername*, puffotter.flask.db.User.User]
Generates a sample service username :return: The service username and the user

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

test_uniqueness()
Tests if the uniqueness of the model is handled properly :return: None

otaku_info_web.test.db.TestUser module

class otaku_info_web.test.db.TestUser.**TestUser**(*methodName='runTest'*)
Bases: otaku_info_web.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_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

otaku_info_web.test.misc package

Submodules

otaku_info_web.test.misc.TestApiCalls module

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

Class that tests varios API calls

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

otaku_info_web.test.misc.TestConfig module

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

Class that tests the config class

test_db_config()
Tests the database configuration :return: None

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

otaku_info_web.test.misc.TestErrorHandling module

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

Class that tests the flask error handling

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

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

otaku_info_web.test.misc.TestMappings module

class otaku_info_web.test.misc.TestMappings.**TestMappings**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Class that tests enum mappings

test_completeness()
Tests that mappings include all possible enum types :return: None

otaku_info_web.test.misc.TestServer module

class otaku_info_web.test.misc.TestServer.**TestServer**(*methodName='runTest'*)
Bases: otaku_info_web.test.TestFramework._TestFramework

Class that tests starting the server

test_starting_server()
Tests starting the server :return: None

Module contents

otaku_info_web.test.routes package

Subpackages

otaku_info_web.test.routes.api package

Submodules

otaku_info_web.test.routes.api.TestApiKeyRoute module

class otaku_info_web.test.routes.api.TestApiKeyRoute.**TestApiKeyRoute**(*methodName='runTest'*)
Bases: otaku_info_web.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

otaku_info_web.test.routes.TestForgotRoute module

class otaku_info_web.test.routes.TestForgotRoute.**TestForgotRoute**(*methodName='runTest'*)
Bases: otaku_info_web.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

otaku_info_web.test.routes.TestLoginRoute module

class otaku_info_web.test.routes.TestLoginRoute.**TestLoginRoute**(*methodName='runTest'*)
Bases: otaku_info_web.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_page_get()
Tests getting the page :return: None

otaku_info_web.test.routes.TestProfileRoute module

class otaku_info_web.test.routes.TestProfileRoute.**TestProfileRoute**(*methodName='runTest'*)
Bases: otaku_info_web.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

otaku_info_web.test.routes.TestRegisterRoute module

class otaku_info_web.test.routes.TestRegisterRoute.**TestRegisterRoute**(*methodName='runTest'*)
Bases: otaku_info_web.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

otaku_info_web.test.routes.TestStaticRoutes module

class otaku_info_web.test.routes.TestStaticRoutes.**TestStaticRoutes**(*methodName='runTest'*)
Bases: otaku_info_web.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

otaku_info_web.test.utils package

Subpackages

otaku_info_web.test.utils.anilist package

Submodules

otaku_info_web.test.utils.anilist.TestApi module

class otaku_info_web.test.utils.anilist.TestApi.**TestApi**(*methodName='runTest'*)
Bases: otaku_info_web.test.TestFramework._TestFramework

Class that tests the anilist API functions

test_guessing_manga_chapter()
Tests guessing a manga chapter :return: None

test_loading_anilist()
Tests loading a user's anilist :return: None

test_loading_media()
Tests loading a media item :return: None

Module contents

Module contents

Submodules

otaku_info_web.test.TestFramework module

Module contents

otaku_info_web.utils package

Subpackages

otaku_info_web.utils.anilist package

Submodules

otaku_info_web.utils.anilist.AnilistItem module

class otaku_info_web.utils.anilist.AnilistItem.**AnilistItem**(*anilist_id: int*, *media_type: otaku_info_web.utils.enums.MediaType*, *media_subtype: otaku_info_web.utils.enums.MediaSubType*, *english_title: Optional[str]*, *romaji_title: str*, *cover_url: str*, *chapters: Optional[int]*, *episodes: Optional[int]*, *releasing_state: otaku_info_web.utils.enums.ReleasingState*, *relations: Dict[Tuple[otaku_info_web.utils.enums.MediaRelationTyp*

Bases: object

Class that models a general anilist list item Represents the information fetched using anilist's API

__init__(*anilist_id: int*, *media_type: otaku_info_web.utils.enums.MediaType*, *media_subtype: otaku_info_web.utils.enums.MediaSubType*, *english_title: Optional[str]*, *romaji_title: str*, *cover_url: str*, *chapters: Optional[int]*, *episodes: Optional[int]*, *releasing_state: otaku_info_web.utils.enums.ReleasingState*, *relations: Dict[Tuple[otaku_info_web.utils.enums.MediaType, int], otaku_info_web.utils.enums.MediaRelationType]*)

Initializes the AnilistItem object :param anilist_id: The anilist ID of the series :param media_type: The media type of the series :param media_subtype: The media subtype of the series :param english_title: The English title of the series :param romaji_title: The Japanes title of the series written in romaji :param cover_url: URL to a cover image for the series :param chapters: The total amount of known manga chapters :param episodes: The total amount of known anime episodes :param releasing_state: The current releasing state of the series :param relations: Related media items identified by IDs

classmethod from_query(*media_type: otaku_info_web.utils.enums.MediaType*, *data: Dict[str, Any]*) → *otaku_info_web.utils.anilist.AnilistItem.AnilistItem*

Generates an AnilistItem from a dictionary generated by an APi query :param media_type: The media type of the item :param data: The data to use :return: The generated AnilistItem

property latest_release

Returns The latest release. Chapters for manga, episodes for anime

class otaku_info_web.utils.anilist.AnilistItem.**AnilistUserItem**(*anilist_id: int*, *media_type: otaku_info_web.utils.enums.MediaType*, *media_subtype: otaku_info_web.utils.enums.MediaSubTy*, *english_title: Optional[str]*, *romaji_title: str*, *cover_url: str*, *chapters: Optional[int]*, *episodes: Optional[int]*, *releasing_state: otaku_info_web.utils.enums.ReleasingSt*, *relations: Dict[Tuple[otaku_info_web.utils.enums.I int], otaku_info_web.utils.enums.MediaRelati*, *score: Optional[int]*, *progress: Optional[int]*, *consuming_state: otaku_info_web.utils.enums.ConsumingS*, *list_name: str*)

Bases: *otaku_info_web.utils.anilist.AnilistItem.AnilistItem*

Class that models an anilist list item for a user Represents the information fetched using anilist's API

__init__(*anilist_id: int*, *media_type: otaku_info_web.utils.enums.MediaType*, *media_subtype: otaku_info_web.utils.enums.MediaSubType*, *english_title: Optional[str]*, *romaji_title: str*, *cover_url: str*, *chapters: Optional[int]*, *episodes: Optional[int]*, *releasing_state: otaku_info_web.utils.enums.ReleasingState*, *relations: Dict[Tuple[otaku_info_web.utils.enums.MediaType, int], otaku_info_web.utils.enums.MediaRelationType]*, *score: Optional[int]*, *progress: Optional[int]*, *consuming_state: otaku_info_web.utils.enums.ConsumingState*, *list_name: str*)

Initializes the AnilistItem object :param anilist_id: The anilist ID of the series :param media_type: The media type of the series :param media_subtype: The media subtype of the series :param english_title: The English title of the series :param romaji_title: The Japanes title of the series written in romaji :param cover_url: URL to a cover image for the series :param chapters: The total amount of known manga chapters :param episodes: The total amount of known anime episodes :param releasing_state: The current releasing state of the series :param relations: Related media items identified by IDs :param score: The user's score for the series :param progress: The user's progress for the series :param consuming_state: The user's consumption state for the series :param list_name: Which of the user's lists this entry belongs to

classmethod from_query(*media_type: otaku_info_web.utils.enums.MediaType*, *data: Dict[str, Any]*) → *otaku_info_web.utils.anilist.AnilistItem.AnilistUserItem*

Generates an AnilistUserItem from a dictionary generated by an APi query :param media_type: The media type of the item :param data: The data to use :return: The generated AnilistItem

otaku_info_web.utils.anilist.api module

otaku_info_web.utils.anilist.api.**guess_latest_manga_chapter**(*anilist_id: int*) → Optional[int]

Guesses the latest chapter number based on anilist user activity :param anilist_id: The anilist ID to check :return: The latest chapter number

otaku_info_web.utils.anilist.api.**load_anilist**(*username: str*, *media_type:* otaku_info_web.utils.enums.MediaType) → List[*otaku_info_web.utils.anilist.AnilistItem.AnilistUserItem*]

Loads the anilist for a user :param username: The username :param media_type: The media type, either MANGA or ANIME :return: The anilist list items for the user and media type

otaku_info_web.utils.anilist.api.**load_media_info**(*anilist_id: int*, *media_type:* otaku_info_web.utils.enums.MediaType) → Optional[*otaku_info_web.utils.anilist.AnilistItem.AnilistItem*]

Loads information for a single anilist media item :param anilist_id: The anilist media ID :param media_type: The media type :return: The fetched AnilistItem

Module contents

otaku_info_web.utils.manga_updates package

Submodules

otaku_info_web.utils.manga_updates.MangaUpdate module

class otaku_info_web.utils.manga_updates.MangaUpdate.**MangaUpdate**(*media_item_id: int*, *title: str*, *cover_url: str*, *latest_release: int*, *progress: int*, *score: int*, *chapter_guess: Optional[int]*, *related_ids: List[Tuple[*otaku_info_web.utils.enum*str]]*)

Bases: object

Class that encapsulates important data to display for manga updates

__init__(*media_item_id: int*, *title: str*, *cover_url: str*, *latest_release: int*, *progress: int*, *score: int*, *chapter_guess: Optional[int]*, *related_ids: List[Tuple[*otaku_info_web.utils.enums.ListService, *str]]*)

Initializes the MangaUpdate object :param media_item_id: The media item ID :param title: The title of the update :param cover_url: The URL for the media item's cover :param latest_release: The latest known released chapter :param progress: The user's current progress :param score: The user's score for this entry :param chapter_guess: The current chapter guess :param related_ids: Related service IDs

class otaku_info_web.utils.manga_updates.MangaUpdate.**RelatedMangaId**(*service:* otaku_info_web.utils.enums.ListS *service_id: str*)

Bases: object

Class that encapslates attributes for a related manga ID

__init__(*service:* otaku_info_web.utils.enums.ListService, *service_id: str*)

Intializes the RelatedMangaId object :param service: The service of the related manga ID :param service_id: The ID on that service

otaku_info_web.utils.manga_updates.generator module

otaku_info_web.utils.manga_updates.generator.**load_applicable_data**(*user: puffotter.flask.db.User.User*, *service:* otaku_info_web.utils.enums.ListServ *media_list: str*, *include_complete: bool*) → Tuple[Dict[int, Dict[str, Any]], Dict[int, Dict[str, Any]], Dict[int, Dict[str, Any]], Dict[int, Dict[str, Any]], Dict[int, Dict[str, Any]], Dict[int, int]]

Loads the applicable data from the database in an efficient manner. By only loading the data we need and avoiding JOINs, the performance is increased drastically. Since this method is called for every call to a manga/updates page, this should be fast. The return values are mostly database IDs mapped to dictionaries containing the data required for displaying manga updates. A notable exception are the manga chapter guesses, which are simply MediaId IDs mapped to the chapter guess value. :param user: The user requesting the manga updates :param service: The service for which to fetch the updates :param media_list: The media list for which to fetch the updates :param include_complete: Whether or not to include completed series :return: media items, media ids, media user states, media lists,

media list items, manga chapter guesses

otaku_info_web.utils.manga_updates.generator.**prepare_manga_updates**(*user: puffotter.flask.db.User.User*, *service: otaku_info_web.utils.enums.ListSe*, *media_list: str*, *include_complete: bool*, *min_update_count: int*) → List[*otaku_info_web.utils.manga_u*

Prepares easily understandable objects to display for manga updates :param user: The user requesting the manga updates :param service: The service for which to fetch the updates :param media_list: The media list for which to fetch the updates :param include_complete: Whether or not to include completed series :param min_update_count: The minimum amount of new chapters required

for an update to be generated

Returns A list of MangaUpdate objects, sorted by score

Module contents

otaku_info_web.utils.mangadex package

Submodules

otaku_info_web.utils.mangadex.api module

otaku_info_web.utils.mangadex.api.**get_external_ids**(*mangadex_id: int*) → Optional[Dict[*otaku_info_web.utils.enums.ListService*, str]]

Retrieves associated IDs for a mangadex ID :param mangadex_id: The mangadex ID :return: The other IDs, mapped to their list service

Module contents

Submodules

otaku_info_web.utils.db_model_helper module

otaku_info_web.utils.db_model_helper.**build_service_url**(*media_type: otaku_info_web.utils.enums.MediaType*, *service: otaku_info_web.utils.enums.ListService*, *service_id: str*) → str

Builds an URL for an external service based on an ID :param media_type: The media type for which to generate an URL :param service: The service for which to create the URL :param service_id: The ID of the media item on that service :return: The generated URL

otaku_info_web.utils.db_model_helper.**build_title**(*english_title: Optional[str]*, *romaji_title: str*) → str

Determines the title based on an English and Romaji title :param english_title: The english title :param romaji_title: The romaji title :return: The title

otaku_info_web.utils.enums module

class otaku_info_web.utils.enums.**ConsumingState**

Bases: enum.Enum

Class that defines the possible consuming states for a user and media item

COMPLETED = 'completed'

CURRENT = 'current'

DROPPED = 'dropped'

PAUSED = 'paused'

PLANNING = 'planning'

REPEATING = 'repeating'

class otaku_info_web.utils.enums.**ListService**

Bases: enum.Enum

Class that defines available list services

ANILIST = 'anilist'

ANIMEPLANET = 'animeplanet'

KITSU = 'kitsu'

MANGADEX = 'mangadex'

MANGAUPDATES = 'mangaupdates'

MYANIMELIST = 'myanimelist'

class otaku_info_web.utils.enums.**MediaRelationType**

Bases: enum.Enum

Class that models a media relation type

ADAPTATION = 'adaptation'

ALTERNATIVE = 'alternative'

CHARACTER = 'character'

COMPILATION = 'compilation'

CONTAINS = 'contains'

OTHER = 'other'

PARENT = 'parent'

PREQUEL = 'prequel'

SEQUEL = 'sequel'

SIDE_STORY = 'side_story'

SOURCE = 'source'

SPIN_OFF = 'spin_off'

SUMMARY = 'summary'

class otaku_info_web.utils.enums.**MediaSubType**
Bases: enum.Enum

Class that models a media subtype for media items

MANGA = 'manga'

MOVIE = 'movie'

MUSIC = 'music'

NOVEL = 'novel'

ONA = 'ona'

ONE_SHOT = 'one_shot'

OVA = 'ova'

SPECIAL = 'special'

TV = 'tv'

TV_SHORT = 'tv_short'

UNKNOWN = 'unknown'

class otaku_info_web.utils.enums.**MediaType**
Bases: enum.Enum

Class that models a media type for media items

ANIME = 'anime'

MANGA = 'manga'

class otaku_info_web.utils.enums.**NotificationType**
Bases: enum.Enum

Class that defines the possible notification types

NEW_ANIME_EPISODES = 'new_anime_episodes'

NEW_MANGA_CHAPTERS = 'new_manga_chapters'

class otaku_info_web.utils.enums.**ReleasingState**
Bases: enum.Enum

Class that defines possible releasing states

CANCELLED = 'cancelled'

FINISHED = 'finished'

NOT_YET_RELEASED = 'not_yet_released'

RELEASING = 'releasing'

UNKNOWN = 'unknown'

otaku_info_web.utils.mappings module

otaku_info_web.utils.mappings.**list_service_id_types: Dict[otaku_info_web.utils.enums.ListS**
Which type a list service ID should have

otaku_info_web.utils.mappings.**list_service_url_formats: Dict[otaku_info_web.utils.enums.Li**
Schemas for URLs for external services

otaku_info_web.utils.mappings.**mangadex_external_id_names: Dict[otaku_info_web.utils.enums**
The names used by mangadex to identify IDs from other services

Module contents

1.1.2 Submodules

1.1.3 otaku_info_web.Config module

class otaku_info_web.Config.**Config**
Bases: puffotter.flask.Config.Config

Configuration for the flask application

TELEGRAM_API_KEY: str = None
API Key for the telegram bot used for notification messages

TELEGRAM_BOT_CONNECTION: TelegramBotConnection = None
Single Telegram bot connection used for all telegram communications

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

1.1.4 otaku_info_web.main module

otaku_info_web.main.**main**()
Starts the flask application :return: None

1.1.5 otaku_info_web.template_extras module

otaku_info_web.template_extras.**profile_extras**() → Dict[str, Any]
Makes sure that the profile page displays service usernames :return: The variables to forward to the template

1.1.6 Module contents

otaku_info_web.**root_path: str = '/usr/local/lib/python3.6/dist-packages/otaku_info_web-0.1**
The root path of the application

otaku_info_web.**sentry_dsn = 'https://f899b0c46d324f37b83527a3994afd8d@sentry.namibsun.net/1**
The sentry DSN used for error logging

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

P

R

S

T

U

V