otaku-info

Release 0.3.2

Hermann Krumrey

Aug 26, 2020

CONTENTS

Contents:

CHAPTER

ONE

OTAKU_INFO

1.1 otaku_info package

1.1.1 Subpackages

otaku_info.background package

Submodules

otaku_info.background.anilist module

otaku_info.background.anilist.**update_anilist_data**(*usernames: Optional[List[*otaku_info.db.ServiceUsername.ServiceUsernam *= None*)

Retrieves all entries on the anilists of all users that provided an anilist username :param usernames: Can be used to override the usernames to use :return: None

otaku_info.background.ln_releases module

otaku_info.background.ln_releases.**create_media_item_from_reddit_ln_release**(*ln_release:* otaku_info.external.ent → Optional[*otaku_info.db.M*

Creates a media item based on a reddit ln release Fills any missing entries in the database :param ln_release: The ln release :return: The media item

otaku_info.background.ln_releases.**update_ln_releases**()

Updates the light novel releases :return: None

otaku_info.background.manga_chapter_guesses module

otaku_info.background.manga_chapter_guesses.**update_manga_chapter_guesses**()
Updates the manga chapter guesses :return: None

otaku_info.background.mangadex module

otaku_info.background.mangadex.**update_mangadex_data**(*start: int = 1*, *end: Optional[int] = None*, *refresh: bool = False*)
Goes through mangadex IDs sequentially and stores ID mappings for these entries if found. Stops once 100 consecutive entries didn't return any results :param start: Optionally specifies a starting index :param end: Optionally specifies an ending index :param refresh: If true, will update existing mangadex info :return: None

otaku_info.background.notifications module

otaku_info.background.notifications.**handle_notification**(*media_user_state: otaku_info.db.MediaUserState.MediaUserState*, *settings: otaku_info.db.NotificationSetting.NotificationSettin*
Handles a single notification :param media_user_state: The user state for which to notify :param settings: The notification settings :return: None

otaku_info.background.notifications.**send_new_update_notifications**()
Sends out telegram notifications for media updates :return: None

Module contents

otaku_info.background.**bg_tasks: Dict[str, Tuple[int, Callable]] = {'anilist_update': (300**
A dictionary containing background tasks for the flask application

otaku_info.db package

Submodules

otaku_info.db.LnRelease module

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

Database model that keeps track of light novel releases

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

digital
Whether this is a digital release

get_ids() → List[*otaku_info.db.MediaId.MediaId*]

Returns Any related Media IDs

id

property identifier_tuple

Returns A tuple that uniquely identifies this database entry

media_item
The media item referenced by this release

media_item_id
The ID of the media item referenced by this release

physical
Whether this is a physical release

publisher
The publisher

purchase_link
Link to a store page

property release_date

Returns The release date as a datetime object

release_date_string
The release date as a ISO-8601 string

series_name
The series name

update(*new_data:* otaku_info.db.LnRelease.LnRelease)
Updates the data in this record based on another object :param new_data: The object from which to use the new values :return: None

volume
The volume identifier

property volume_number

Returns The volume number as an integer

otaku_info.db.MangaChapterGuess module

class otaku_info.db.MangaChapterGuess.**MangaChapterGuess**(**args*, ***kwargs*)
Bases: *otaku_info.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

property identifier_tuple

Returns A tuple that uniquely identifies this database entry

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(*new_data:* otaku_info.db.MangaChapterGuess.MangaChapterGuess)
Updates the data in this record based on another object :param new_data: The object from which to use the new values :return: None

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

otaku_info.db.MediaId module

class otaku_info.db.MediaId.**MediaId**(**args*, ***kwargs*)
Bases: *otaku_info.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

chapter_guess
Chapter Guess for this media ID (Only applicable if this is a manga title)

id

property identifier_tuple

Returns A tuple that uniquely identifies this database entry

media_item
The media item referenced by this ID

media_item_id
The ID of the media item referenced by this ID

media_subtype
The media subtype of the list item

media_type
The media type of the list item

media_user_states
Media user states associated with this media ID

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

update(*new_data:* otaku_info.db.MediaId.MediaId)
Updates the data in this record based on another object :param new_data: The object from which to use the new values :return: None

otaku_info.db.MediaItem module

class otaku_info.db.MediaItem.**MediaItem**(**args*, ***kwargs*)
Bases: *otaku_info.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

property identifier_tuple

Returns A tuple that uniquely identifies this database entry

latest_release
The latest release chapter/episode for this media item

latest_volume_release
The latest volume for this media item

ln_releases
Light novel releases associated with this Media item

property media_id_mapping

Returns A dictionary mapping list services to IDs for this media item

media_ids
Media IDs associated with this Media item

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

media_type
The media type of the list item

property own_url

Returns The URL to the item's page on the otaku-info site

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.

update(*new_data:* otaku_info.db.MediaItem.MediaItem)
Updates the data in this record based on another object :param new_data: The object from which to use the new values :return: None

otaku_info.db.MediaList module

class otaku_info.db.MediaList.**MediaList**(**args*, ***kwargs*)
Bases: *otaku_info.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

property identifier_tuple

Returns A tuple that uniquely identifies this database entry

media_list_items
Media List Items that are a part of this media list

media_type
The media type for this list

name
The name of this list

service
The service for which this list applies to

update(*new_data:* otaku_info.db.MediaList.MediaList)
Updates the data in this record based on another object :param new_data: The object from which to use the new values :return: None

user
The user associated with this list

user_id
The ID of the user associated with this list

otaku_info.db.MediaListItem module

class otaku_info.db.MediaListItem.**MediaListItem**(**args*, ***kwargs*)
Bases: *otaku_info.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

property identifier_tuple

Returns A tuple that uniquely identifies this database entry

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

update(*new_data:* otaku_info.db.MediaListItem.MediaListItem)
Updates the data in this record based on another object :param new_data: The object from which to use the new values :return: None

otaku_info.db.MediaNotification module

class otaku_info.db.MediaNotification.**MediaNotification**(**args*, ***kwargs*)
Bases: *otaku_info.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

property identifier_tuple

Returns A tuple that uniquely identifies this database entry

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

update(*new_data:* otaku_info.db.MediaNotification.MediaNotification)
Updates the data in this record based on another object :param new_data: The object from which to use the new values :return: None

otaku_info.db.MediaUserState module

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

Bases: *otaku_info.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

property identifier_tuple

Returns A tuple that uniquely identifies this database entry

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_notification

Notification object for this user state

progress

The user's current progress consuming the media item

score

The user's score for the references media item

update(*new_data:* otaku_info.db.MediaUserState.MediaUserState)

Updates the data in this record based on another object :param new_data: The object from which to use the new values :return: None

user

The user associated with this user state

user_id

The ID of the user associated with this user state

volume_progress

The user's current 'volume' progress.

otaku_info.db.ModelMixin module

class otaku_info.db.ModelMixin.**ModelMixin**
Bases: puffotter.flask.db.ModelMixin.ModelMixin

Class that define methods that greatly ease working with existing database entries

property identifier_tuple

Returns A tuple that's unique to this database entry

update(*new_data:* otaku_info.db.ModelMixin.ModelMixin)
Updates the data in this record based on another object :param new_data: The object from which to use the new values :return: None

otaku_info.db.NotificationSetting module

class otaku_info.db.NotificationSetting.**NotificationSetting**(**args*, ***kwargs*)
Bases: *otaku_info.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

property identifier_tuple

Returns A tuple that uniquely identifies this database entry

minimum_score
The minimum score for notification items

notification_type
The notification type

update(*new_data:* otaku_info.db.NotificationSetting.NotificationSetting)
Updates the data in this record based on another object :param new_data: The object from which to use the new values :return: None

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.db.ServiceUsername module

class otaku_info.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

property identifier_tuple

Returns A tuple that uniquely identifies this database entry

service
The external service this item is a username for

update(*new_data:* otaku_info.db.ServiceUsername.ServiceUsername)
Updates the data in this record based on another object :param new_data: The object from which to use the new values :return: None

user
The user associated with this service username

user_id
The ID of the user associated with this service username

username
The service username

Module contents

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

otaku_info.external package

Subpackages

otaku_info.external.entities package

Submodules

otaku_info.external.entities.AnilistItem module

class otaku_info.external.entities.AnilistItem.**AnilistItem**(*_id: int*, *service:* otaku_info.enums.ListService, *extra_ids: Dict[*otaku_info.enums.ListService*, str]*, *media_type:* otaku_info.enums.MediaType, *media_subtype:* otaku_info.enums.MediaSubType, *english_title: Optional[str]*, *romaji_title: str*, *cover_url: str*, *chapters: Optional[int]*, *volumes: Optional[int]*, *episodes: Optional[int]*, *releasing_state:* otaku_info.enums.ReleasingState, *relations: Dict[Tuple[*otaku_info.enums.MediaType*, int]*, otaku_info.enums.MediaRelationType*]*)

Bases: *otaku_info.external.entities.AnimeListItem.AnimeListItem*

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

classmethod from_query(*media_type:* otaku_info.enums.MediaType, *data: Dict[str, Any]*) → *otaku_info.external.entities.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 myanimelist_id

Returns The myanimelist ID

otaku_info.external.entities.AnilistUserItem module

class otaku_info.external.entities.AnilistUserItem.**AnilistUserItem**(*_id: int*, *service: otaku_info.enums.ListService*, *extra_ids: Dict[otaku_info.enums.ListService*, *str]*, *media_type: otaku_info.enums.MediaType*, *media_subtype: otaku_info.enums.MediaSubType*, *english_title: Optional[str]*, *romaji_title: str*, *cover_url: str*, *chapters: Optional[int]*, *volumes: Optional[int]*, *episodes: Optional[int]*, *releasing_state: otaku_info.enums.ReleasingState*, *relations: Dict[Tuple[otaku_info.enums.Med*, *int]*, *otaku_info.enums.MediaRelationT*, *score: Optional[int]*, *progress: Optional[int]*, *volume_progress: Optional[int]*, *consuming_state: otaku_info.enums.ConsumingState*, *list_name: str*)

Bases: *otaku_info.external.entities.AnilistItem.AnilistItem*

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

__init__(*_id: int*, *service:* otaku_info.enums.ListService, *extra_ids: Dict[*otaku_info.enums.ListService*, str]*, *media_type:* otaku_info.enums.MediaType, *media_subtype:* otaku_info.enums.MediaSubType, *english_title: Optional[str]*, *romaji_title: str*, *cover_url: str*, *chapters: Optional[int]*, *volumes: Optional[int]*, *episodes: Optional[int]*, *releasing_state:* otaku_info.enums.ReleasingState, *relations: Dict[Tuple[*otaku_info.enums.MediaType*, int]*, otaku_info.enums.MediaRelationType*]*, *score: Optional[int]*, *progress: Optional[int]*, *volume_progress: Optional[int]*, *consuming_state:* otaku_info.enums.ConsumingState, *list_name: str*)

Initializes the AnilistItem object :param _id: The anilist ID :param service: Anilist :param extra_ids: The myanimelist 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 volumes: The total amount of known manga/ln volumes :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 volume_progress: The user's volume progress :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.enums.MediaType, *data: Dict[str, Any]*) → *otaku_info.external.entities.AnilistUserItem.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.external.entities.AnimeListItem module

class otaku_info.external.entities.AnimeListItem.**AnimeListItem**(*_id: int*, *service:* otaku_info.enums.ListService, *extra_ids: Dict[*otaku_info.enums.ListService*, str]*, *media_type:* otaku_info.enums.MediaType, *media_subtype:* otaku_info.enums.MediaSubType, *english_title: Optional[str]*, *romaji_title: str*, *cover_url: str*, *chapters: Optional[int]*, *volumes: Optional[int]*, *episodes: Optional[int]*, *releasing_state:* otaku_info.enums.ReleasingState, *relations: Dict[Tuple[*otaku_info.enums.MediaType*, int]*, otaku_info.enums.MediaRelationType*]*)

Bases: object

Class that models a general anime list item

__init__(*_id: int*, *service: otaku_info.enums.ListService*, *extra_ids: Dict[otaku_info.enums.ListService, str]*, *media_type: otaku_info.enums.MediaType*, *media_subtype: otaku_info.enums.MediaSubType*, *english_title: Optional[str]*, *romaji_title: str*, *cover_url: str*, *chapters: Optional[int]*, *volumes: Optional[int]*, *episodes: Optional[int]*, *releasing_state: otaku_info.enums.ReleasingState*, *relations: Dict[Tuple[otaku_info.enums.MediaType, int], otaku_info.enums.MediaRelationType]*)

Initializes the AnimeListItem object :param _id: The anime list ID :param service: The anime list service :param extra_ids: Any extra IDs for other services :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 volumes: The total amount of known manga/ln volumes :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.enums.MediaType*, *data: Dict[str, Any]*) → *otaku_info.external.entities.AnimeListItem.AnimeListItem*

Generates an AnimeListItem 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 AnimeListItem

property latest_release

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

otaku_info.external.entities.MangadexItem module

class otaku_info.external.entities.MangadexItem.**MangadexItem**(*mangadex_id: int*, *external_ids: Dict[otaku_info.enums.ListService, str]*, *title: str*, *cover_url: str*, *total_chapters: Optional[int]*, *latest_chapter: Optional[int]*, *releasing_state: otaku_info.enums.ReleasingState*)

Bases: object

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

__init__(*mangadex_id: int*, *external_ids: Dict[otaku_info.enums.ListService, str]*, *title: str*, *cover_url: str*, *total_chapters: Optional[int]*, *latest_chapter: Optional[int]*, *releasing_state: otaku_info.enums.ReleasingState*)

Initializes the MangadexItem object :param mangadex_id: The mangadex ID of the series :param external_ids: IDs for other services :param title: The title of the series :param cover_url: URL to a cover image for this series :param total_chapters: The total amount of chapters :param latest_chapter: The latest chapter :param releasing_state: The releasing state

classmethod from_json(*mangadex_id: int*, *data: Dict[str, Any]*) → *otaku_info.external.entities.MangadexItem.MangadexItem*

Generates a MangadexItem from a dictionary generated by an API query :param mangadex_id: The mangadex ID :param data: The data to use :return: The generated MangadexItem

static parse_chapter(*chapter_string: str*) → Optional[int]

Converts a mangadex chapter string into an integer :param chapter_string: The chapter string to convert

:return: The converted integer

static resolve_releasing_state(*state_code: int*) → *otaku_info.enums.ReleasingState*
Translates mangadex status codes to releasing states :param state_code: The status code to translate :return: The releasing state

otaku_info.external.entities.MyanimelistItem module

class otaku_info.external.entities.MyanimelistItem.**MyanimelistItem**(*_id: int*, *service:* otaku_info.enums.ListService, *extra_ids:* *Dict[*otaku_info.enums.ListService *str]*, *media_type:* otaku_info.enums.MediaType, *media_subtype:* otaku_info.enums.MediaSubType, *english_title:* *Optional[str]*, *romaji_title:* *str*, *cover_url:* *str*, *chapters:* *Optional[int]*, *volumes:* *Optional[int]*, *episodes:* *Optional[int]*, *releasing_state:* otaku_info.enums.ReleasingState, *relations:* *Dict[Tuple[*otaku_info.enums.Med *int]*, otaku_info.enums.MediaRelationT

Bases: *otaku_info.external.entities.AnimeListItem.AnimeListItem*

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

classmethod from_query(*media_type:* otaku_info.enums.MediaType, *data: Dict[str, Any]*) → *otaku_info.external.entities.MyanimelistItem.MyanimelistItem*
Generates an MyanimelistItem 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 MyanimelistItem

static resolve_media_subtype(*subtype_string: str*) → *otaku_info.enums.MediaSubType*
Resolves myanimelist media subtype :param subtype_string: The string to resolve :return: The resolved

MediaSubType

static resolve_relation_type(*relation_string: str*) → *otaku_info.enums.MediaRelationType*
Resolves myanimelist relation types :param relation_string: The string to resolve :return: The resolved MediaRelationType

static resolve_releasing_state(*releasing_string: str*) → *otaku_info.enums.ReleasingState*
Resolves myanimelist releasing state :param releasing_string: The string to resolve :return: The resolved ReleasingState

otaku_info.external.entities.RedditLnRelease module

class otaku_info.external.entities.RedditLnRelease.**RedditLnRelease**(*series_name: str*, *year: int*, *release_date_string: str*, *volume: str*, *publisher: Optional[str]*, *purchase_link: Optional[str]*, *info_link: Optional[str]*, *digital: bool*, *physical: bool*)

Bases: object

Object that acts as a wrapper around a light novel release on reddit.com

__init__(*series_name: str*, *year: int*, *release_date_string: str*, *volume: str*, *publisher: Optional[str]*, *purchase_link: Optional[str]*, *info_link: Optional[str]*, *digital: bool*, *physical: bool*)
Initializes the object :param series_name: The name of the series :param year: The year of release :param release_date_string: The month and day of release :param volume: The volume number :param publisher: The publisher of the release :param purchase_link: A purchase link for the release :param info_link: Link to information for the release :param digital: Whether or not the release is digital :param physical: Whether or not the release is physical

property anilist_id

Returns The anilist ID, if available

classmethod from_parts(*year: int*, *parts: List[bs4.element.Tag]*) → *otaku_info.external.entities.RedditLnRelease.RedditLnRelease*
Generates a reddit LN release from BeautifulSoup td tags :param year: The year of the release :param parts: The td tags :return: The reddit ln release

property myanimelist_id

Returns The myanimelist ID, if available

property release_date

Returns The release date as a datetime object

property release_date_string

Returns The release date as a string (ISO 8601)

Module contents

Submodules

otaku_info.external.anilist module

otaku_info.external.anilist.**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.external.anilist.**load_anilist**(*username: str*, *media_type:* otaku_info.enums.MediaType) → List[*otaku_info.external.entities.AnilistUserItem.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.external.anilist.**load_anilist_info**(*service_id: int*, *media_type: otaku_info.enums.MediaType*, *service: otaku_info.enums.ListService = <ListService.ANILIST: 'anilist'>*) → Optional[*otaku_info.external.entities.AnilistItem.AnilistItem*]

Loads information for a single anilist media item :param service_id: The anilist or myanimelist media ID :param media_type: The media type :param service: The service the ID belongs to

(either anilist or myanimelist)

Returns The fetched AnilistItem

otaku_info.external.mangadex module

otaku_info.external.mangadex.**fetch_mangadex_item**(*mangadex_id: int*) → Optional[*otaku_info.external.entities.MangadexItem.MangadexIt*

Fetches information for a mangadex

otaku_info.external.myanimelist module

otaku_info.external.myanimelist.**load_myanimelist_item**(*myanimelist_id: int*, *media_type:* otaku_info.enums.MediaType) → Optional[*otaku_info.external.entities.MyanimelistItem.My*

Loads myanimelist data using the jikan API :param myanimelist_id: The myanimelist ID :param media_type: The media type :return: The myanimelist item

otaku_info.external.reddit module

otaku_info.external.reddit.**load_ln_releases**(*year: Optional[int] = None*) → List[*otaku_info.external.entities.RedditLnRelease.RedditLnRelease*]

Loads the light novel releases

otaku_info.external.reddit.**load_tables**(*year: int*) → List[bs4.BeautifulSoup]

Loads the tables containing the release data

Module contents

otaku_info.routes package

Subpackages

otaku_info.routes.api package

Submodules

otaku_info.routes.api.media_api module

otaku_info.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.routes.external_service module

otaku_info.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.routes.ln module

otaku_info.routes.ln.**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.routes.media module

otaku_info.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.routes.notifications module

otaku_info.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

otaku_info.routes.updates module

otaku_info.routes.updates.**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

otaku_info.routes.**blueprint_generators: List[Tuple[Callable[str, flask.blueprints.Bluepri**

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

otaku_info.test package

Subpackages

otaku_info.test.db package

Submodules

otaku_info.test.db.TestApiKey module

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

Bases: otaku_info.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.test.db.TestMangaChapterGuess module

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

Class that tests the MangaChapterGuess database model

static generate_sample_guess() → Tuple[*otaku_info.db.MediaItem.MediaItem*, *otaku_info.db.MediaId.MediaId*, *otaku_info.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.test.db.TestMediaId module

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

Class that tests the MediaId database model

static generate_sample_media_id() → Tuple[*otaku_info.db.MediaItem.MediaItem*, *otaku_info.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.test.db.TestMediaItem module

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

Class that tests the MediaItem database model

static generate_sample_media_item() → *otaku_info.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.test.db.TestMediaList module

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

Class that tests the MediaList database model

generate_sample_media_list() → Tuple[*otaku_info.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.test.db.TestMediaListItem module

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

Class that tests the MediaListItem database model

generate_sample_media_list_item() → Tuple[*otaku_info.db.MediaListItem.MediaListItem*, *otaku_info.db.MediaList.MediaList*, *otaku_info.db.MediaUserState.MediaUserState*, puffotter.flask.db.User.User, *otaku_info.db.MediaItem.MediaItem*, *otaku_info.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.test.db.TestMediaRelation module

otaku_info.test.db.TestMediaUserState module

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

Class that tests the MediaUserState database model

generate_sample_media_user_state() → Tuple[*otaku_info.db.MediaUserState.MediaUserState*, puffotter.flask.db.User.User, *otaku_info.db.MediaItem.MediaItem*, *otaku_info.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.test.db.TestServiceUsername module

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

Class that tests the ServiceUsername database model

generate_sample_service_username() → Tuple[*otaku_info.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.test.db.TestUser module

class otaku_info.test.db.TestUser.**TestUser**(*methodName='runTest'*)
Bases: otaku_info.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.test.misc package

Submodules

otaku_info.test.misc.TestApiCalls module

class otaku_info.test.misc.TestApiCalls.**TestConfig**(*methodName='runTest'*)
Bases: otaku_info.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.test.misc.TestConfig module

class otaku_info.test.misc.TestConfig.**TestConfig**(*methodName='runTest'*)
Bases: otaku_info.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.test.misc.TestErrorHandling module

class otaku_info.test.misc.TestErrorHandling.**TestErrorHandling**(*methodName='runTest'*)
Bases: otaku_info.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.test.misc.TestMappings module

class otaku_info.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.test.misc.TestServer module

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

Class that tests starting the server

test_starting_server()
Tests starting the server :return: None

Module contents

otaku_info.test.routes package

Subpackages

otaku_info.test.routes.api package

Submodules

otaku_info.test.routes.api.TestApiKeyRoute module

class otaku_info.test.routes.api.TestApiKeyRoute.**TestApiKeyRoute**(*methodName='runTest'*)
Bases: otaku_info.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.test.routes.TestForgotRoute module

class otaku_info.test.routes.TestForgotRoute.**TestForgotRoute**(*methodName='runTest'*)
Bases: otaku_info.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.test.routes.TestLoginRoute module

class otaku_info.test.routes.TestLoginRoute.**TestLoginRoute**(*methodName='runTest'*)
Bases: otaku_info.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.test.routes.TestProfileRoute module

class otaku_info.test.routes.TestProfileRoute.**TestProfileRoute**(*methodName='runTest'*)
Bases: otaku_info.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.test.routes.TestRegisterRoute module

class otaku_info.test.routes.TestRegisterRoute.**TestRegisterRoute**(*methodName='runTest'*)
Bases: otaku_info.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.test.routes.TestStaticRoutes module

class otaku_info.test.routes.TestStaticRoutes.**TestStaticRoutes**(*methodName='runTest'*)
Bases: otaku_info.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.test.utils package

Module contents

Submodules

otaku_info.test.TestFramework module

Module contents

otaku_info.utils package

Subpackages

otaku_info.utils.db package

Submodules

otaku_info.utils.db.DbCache module

class otaku_info.utils.db.DbCache.**DbCache**
Bases: object

Class that helps identifying existing items by caching all database entries

static cleanup()
Removes any cached items for the current thread :return: None

static get_existing_item(*item:* otaku_info.db.ModelMixin.ModelMixin) → Optional[*otaku_info.db.ModelMixin.ModelMixin*]
Retrieves an existing item from the cache if it exists :param item: The item for which to retrieve the existing item :return: The existing item or None

static load_existing_items(*cls: Type[*otaku_info.db.ModelMixin.ModelMixin*]*, *reload: bool = False*) → Dict[Tuple, *otaku_info.db.ModelMixin.ModelMixin*]
Retrieves all existing items for a database class mapped to their identifier tuples. Automatically separates the results by thread. :param cls: The database class :param reload: Whether to force a reload of database data :return: The existing entries mapped to identifier tuples

static update_item(*item:* otaku_info.db.ModelMixin.ModelMixin)
Updates a cached item :param item: The new item :return: None

otaku_info.utils.db.convert module

otaku_info.utils.db.convert.**anilist_user_item_to_media_list**(*anilist_user_item:* otaku_info.external.entities.AnilistUserItem.A *user:* *puffotter.flask.db.User.User*) → *otaku_info.db.MediaList.MediaList*
Generates a media list from an anilist user item :param anilist_user_item: The anilist user item :param user: The corresponding user :return: The media list object

otaku_info.utils.db.convert.**anilist_user_item_to_media_user_state**(*anilist_user_item:* otaku_info.external.entities.AnilistU *media_id:* otaku_info.db.MediaId.MediaId, *user:* *puffotter.flask.db.User.User*) → *otaku_info.db.MediaUserState.Medi*
Generates a media user state based on an anilist user item :param anilist_user_item: The anilist user item :param media_id: The corresponding media id :param user: The corresponding user :return: The media user state

otaku_info.utils.db.convert.**anime_list_item_to_media_id**(*anime_list_item:* otaku_info.external.entities.AnimeListItem.AnimeL *media_item:* otaku_info.db.MediaItem.MediaItem, *list_service:* *Optional[*otaku_info.enums.ListService*]* = *None*) → *otaku_info.db.MediaId.MediaId*
Converts an anime list item into a media id :param anime_list_item: The anime list item :param media_item: The corresponding media item :param list_service: Uses an extra ID with the specified list service :return: The generated media id

otaku_info.utils.db.convert.**anime_list_item_to_media_item**(*anime_list_item:* otaku_info.external.entities.AnimeListItem.Anim → *otaku_info.db.MediaItem.MediaItem*
Converts an anime list item to a media item object :param anime_list_item: The anime list item :return: The media item

otaku_info.utils.db.convert.**ln_release_from_reddit_item**(*ln_release:* otaku_info.external.entities.RedditLnRelease.Reddi *media_item:* *Optional[*otaku_info.db.MediaItem.MediaItem*]*) → *otaku_info.db.LnRelease.LnRelease*

Creates an LnRelease object based on a RedditLnRelease object :param ln_release: The RedditLnRelease object :param media_item: An associated MediaItem :return: The LnRelease object

otaku_info.utils.db.convert.**mangadex_item_to_media_item**(*mangadex_item:* otaku_info.external.entities.MangadexItem.Mangad → *otaku_info.db.MediaItem.MediaItem*

Converts a mangadex item to a media item object :param mangadex_item: The mangadex item :return: The media item

otaku_info.utils.db.load module

otaku_info.utils.db.load.**load_service_ids**(*limit_media_type_to:* *Optional[*otaku_info.enums.MediaType*]* *= None*) → Dict[*otaku_info.enums.ListService*, Dict[str, *otaku_info.db.MediaId.MediaId*]]

Loads Media IDs from the database and maps them to their service id :param limit_media_type_to: If set, limits the media type :return: The mapped ids

otaku_info.utils.db.updater module

otaku_info.utils.db.updater.**update_or_insert_item**(*to_add:* otaku_info.db.ModelMixin.ModelMixin, *commit_updates:* *bool* *= False*, *reload_cache:* *bool* *= False*) → *otaku_info.db.ModelMixin.ModelMixin*

Updates or creates an item that can be identified based on a unique tuple. Ensures that no duplicates are inserted across threads :param to_add: The item to add :param commit_updates: Whether or not to commit updates :param reload_cache: Reloads the cached data if set to True :return: None

Module contents

Submodules

otaku_info.utils.dates module

otaku_info.utils.dates.**map_month_name_to_month_number**(*month_name:* *str*) → Optional[int]

Maps month names to month numbers :param month_name: The name of the month :return: The month number

otaku_info.utils.dates.**map_month_number_to_month_name**(*month_number:* *int*) → Optional[str]

Maps month numbers to month names :param month_number: The month number :return: The name of the month

Module contents

otaku_info.wrappers package

Submodules

otaku_info.wrappers.UpdateWrapper module

class otaku_info.wrappers.UpdateWrapper.**UpdateWrapper**(*media_user_state:* otaku_info.db.MediaUserState.MediaUserState)

Bases: object

Class that encapsulates important data to display for manga updates

__init__(*media_user_state:* otaku_info.db.MediaUserState.MediaUserState)

Initializes the MangaUpdate object :param media_user_state: The media user state contains everything we need. For performance reasons, it's assumed that all relations have been loaded already.

calculate_latest() → int

Returns The latest release number

calculate_progress() → int

Returns The user's current progress

classmethod from_db(*user: puffotter.flask.db.User.User*, *list_name: str*, *service:* otaku_info.enums.ListService, *media_type:* otaku_info.enums.MediaType, *media_subtype: Optional[*otaku_info.enums.MediaSubType*]*, *minimum_diff: int*, *include_complete: bool*)

Generates UpdateWrapper objects based on a couple of parameters and the current database contents :param user: The user for whom to load the updates :param list_name: The list name for which to load the updates :param service: The service for which to load the updates :param media_type: The media type for which to load the updates :param media_subtype: If specified, limits the results to a specific media subtype (example: Light novels)

Parameters

- **minimum_diff** – Specifies a minimum diff value
- **include_complete** – Specifies whether completed items should be included

Returns A list of UpdateWrapper objects

classmethod from_media_lists(*media_lists: List[*otaku_info.db.MediaList.MediaList*]*, *media_subtype: Optional[*otaku_info.enums.MediaSubType*]*, *minimum_diff: int*, *include_complete: bool*) → List[*otaku_info.wrappers.UpdateWrapper.UpdateWrapper*]

Generates UpdateWrapper objects based on media lists To avoid huge performance problems, the relations should be fully loaded beforehand, preferably in a single query. Results can be filtered using the additional parameters :param media_lists: The media lists containing the items to wrap :param media_subtype: If specified, limits the results to a specific media subtype (example: Light novels)

Parameters

- **minimum_diff** – Specifies a minimum diff value

- **include_complete** – Specifies whether completed items should be included

Returns A list of UpdateWrapper objects

Module contents

1.1.2 Submodules

1.1.3 otaku_info.Config module

class otaku_info.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

1.1.4 otaku_info.enums module

class otaku_info.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.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.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.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.enums.**MediaType**
Bases: enum.Enum

Class that models a media type for media items

ANIME = 'anime'

MANGA = 'manga'

class otaku_info.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.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'

1.1.5 otaku_info.main module

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

1.1.6 otaku_info.mappings module

otaku_info.mappings.**list_service_id_types: Dict[otaku_info.enums.ListService, Type] = {<Li**
Which type a list service ID should have

otaku_info.mappings.**list_service_priorities: List[otaku_info.enums.ListService] = [<ListSe**
Specifies the order in which list service data is prioritized

otaku_info.mappings.**list_service_url_formats: Dict[otaku_info.enums.ListService, str] = {<**
Schemas for URLs for external services

otaku_info.mappings.**mangadex_external_id_names: Dict[otaku_info.enums.ListService, str] =**
The names used by mangadex to identify IDs from other services

1.1.7 otaku_info.template_extras module

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

1.1.8 Module contents

otaku_info.**root_path: str = '/usr/local/lib/python3.6/dist-packages/otaku_info-0.3.2-py3.6**
The root path of the application

otaku_info.**sentry_dsn = 'https://f899b0c46d324f37b83527a3994afd8d@sentry.namibsun.net/18'**
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

N

O

P

R

S

T

U

V