UCube

MujyKun

Jul 29, 2021

CONTENTS:

1 UCubeClient **1**

2 Clients **5**
- 2.1 UCubeClientSync . . . 5
- 2.2 UCubeClientAsync . . . 7

3 Models **11**
- 3.1 BaseModel . . . 11
- 3.2 Club . . . 12
- 3.3 Board . . . 13
- 3.4 Post . . . 14
- 3.5 User . . . 15
- 3.6 Notification . . . 16
- 3.7 Comment . . . 17
- 3.8 Image . . . 18
- 3.9 Video . . . 19

4 Model Creation **21**

5 Exceptions **23**
- 5.1 Invalid Token . . . 23
- 5.2 InvalidCredentials . . . 23
- 5.3 Something Went Wrong . . . 23
- 5.4 LoginFailed . . . 23
- 5.5 Page Not Found . . . 23
- 5.6 Being Rate Limited . . . 24
- 5.7 NoHookFound . . . 24

6 Get Account Token **25**

7 Asynchronous Usage **27**

8 Synchronous Usage **29**

9 Indices and tables **31**

Python Module Index **33**

Index **35**

CHAPTER

ONE

UCUBECLIENT

class UCube.**UCubeClient**(*username: Optional[str] = None*, *password: Optional[str] = None*, *token=None*, *web_session=None*, *verbose: bool = False*, *hook=None*)

Abstract & Parent Client for connecting to UCube and creating the internal cache.

> **Warning:** Do not create an object directly from this class. Instead, create a *UCube.UCubeClientSync* or *UCube.UCubeClientAsync* object since those are concrete.

Parameters

- **username** (*str*) – Email or Username to log in with.
- **password** (*str*) – Password to log in with.
- **verbose** (*bool*) – Whether to print out verbose messages.
- **web_session** – An aiohttp or requests client session.
- **token** – The account token to connect to the UCube API. In order to find your token, please refer to *Get Account Token*
- **hook** – A passed in method that will be called every time there is a new notification. This method must take in a list of *models.Notification* objects.

verbose

Whether to print out verbose messages.

Type bool

web_session

An aiohttp or requests client session.

cache_loaded

Whether the Internal UCube Cache is fully loaded. This will change for a split moment when grabbing a new post.

Type bool

clubs

A dict of all Clubs in cache with the slug as the key.

Type Dict[str, *models.Club*]

boards

A dict of all Boards in cache with the slug as the key.

Type Dict[str, *models.Board*]

posts
A dict of all Posts in cache with the slug as the key.

Type Dict[str, *models.Post*]

users
A dict of all Users in cache with the slug as the key.

Type Dict[str, *models.User*]

notifications
A dict of all Notifications in cache with the slug as the key.

Type Dict[str, *models.Notification*]

get_board(*board_slug: str*) → Optional[*UCube.models.board.Board*]
Get a Board if it exists.

Parameters **board_slug** (*str*) – The unique identifier of the Board.

Returns **The Board associated with the slug if it exists.**

Return type *models.Board*

get_club(*club_slug: str*) → Optional[*UCube.models.club.Club*]
Get a Club if it exists.

Parameters **club_slug** (*str*) – The unique identifier of the Club.

Returns **The Club associated with the slug if it exists.**

Return type *models.Club*

get_comment(*comment_slug: str*) → Optional[*UCube.models.comment.Comment*]
Get a Comment if it exists.

Parameters **comment_slug** (*str*) – The unique identifier of the Comment.

Returns **The Comment associated with the slug if it exists.**

Return type *models.Comment*

get_notification(*notification_slug: str*) → Optional[*UCube.models.notification.Notification*]
Get a Notification if it exists.

Parameters **notification_slug** (*str*) – The unique identifier of the Notification.

Returns **The Notification associated with the slug if it exists.**

Return type *models.Notification*

get_post(*post_slug: str*) → Optional[*UCube.models.post.Post*]
Get a Post if it exists.

Parameters **post_slug** (*str*) – The unique identifier of the Post.

Returns **The Post associated with the slug if it exists.**

Return type *models.Post*

get_user(*user_slug: str*) → Optional[*UCube.models.user.User*]
Get a User if it exists.

Parameters **user_slug** (*str*) – The unique identifier of the User.

Returns **The User associated with the slug if it exists.**

Return type *models.User*

static replace(*url*, ***kwargs*) → str
Will replace the args in an endpoint url.

Parameters

- **url** (*str*) – The URL to replace args for.
- **kwargs** (*dict*) – The args that need to be replaced followed by what they need to be replaced with.

Returns

Return type str

stop()
Stop the hook loop.

CHAPTER

TWO

CLIENTS

2.1 UCubeClientSync

class UCube.**UCubeClientSync**(***kwargs*)

Synchronous UCube Client that Inherits from *UCubeClient*.

Parameters **kwargs** – Args for *UCubeClient*.

check_new_notifications() → List[*UCube.models.notification.Notification*]

Checks and returns new notifications for every club.

Compares with the already existing notifications. This will also create the posts associated with the notification so they can be used efficiently.

Returns **A list of new Notifications.**

Return type List[*models.Notification*]

check_token_works()

Check if a token is valid and will set the general information about the client if it is.

Returns (bool) True if the token works.

fetch_all_clubs(*clubs_per_page: int = 99999*, *page_number: int = 1*) → List[*UCube.models.club.Club*]

Fetch all Clubs from the UCube API.

Returns **A list of Clubs**

Return type List[*models.Club*]

fetch_board_posts(*board_slug: str*, *feed=False*, *posts_per_page: int = 99999*, *page_number: int = 1*) → List[*UCube.models.post.Post*]

Retrieve a list of Posts from a board.

Parameters

- **board_slug** (*str*) – The slug (unique identifier) of a board to search the feed for.
- **feed** (*bool*) – Whether to use the feeds endpoint. The feeds endpoint would retrieve the user information of a post.
- **posts_per_page** (*int*) – The amount of posts to retrieve per page.
- **page_number** (*int*) – The page number when paginating.

Returns **A list of Posts**

Return type List[*models.Post*]

fetch_club_boards(*club_slug: str*) → List[*UCube.models.board.Board*]
Retrieve a list of Boards from a Club.

Parameters **club_slug** (*str*) – The slug (Unique Identifier) of a Club.

Returns **A list of Boards**

Return type List[*models.Board*]

fetch_club_notifications(*club_slug: str*, *notifications_per_page: int = 99999*, *page_number: int = 1*) → List[*UCube.models.notification.Notification*]
Retrieve a list of Notifications from a club.

Parameters

- **club_slug** (*str*) – The slug (unique identifier) of a club to search the notifications for.
- **notifications_per_page** (*int*) – The amount of notifications to retrieve per page.
- **page_number** (*int*) – The page number when paginating.

Returns **A list of Notifications**

Return type List[*models.Notification*]

fetch_post(*post_slug: str*, *load_comments=False*) → Optional[*UCube.models.post.Post*]
Fetch a Post by it's slug.

Parameters

- **post_slug** (str) – The post's unique identifier.
- **load_comments** (bool) – Whether to load up the comments of the Post.

Returns **The Post Object if there is one.**

Return type *models.Post*

fetch_post_comments(*post_slug: str*, *comments_per_page: int = 99999*, *page_number: int = 1*) → List[*UCube.models.comment.Comment*]
Retrieve a list of Comments from a Post.

Parameters

- **post_slug** (*str*) – The slug (unique identifier) of a Post to search the Comments for.
- **comments_per_page** (*int*) – The amount of notifications to retrieve per page.
- **page_number** (*int*) – The page number when paginating.

Returns **A list of Comments**

Return type List[*models.Comment*]

follow_club(*club_slug: str*) → bool
Follow a club.

Parameters **club_slug** (str) – The unique identifier of the club to follow.

Returns **Whether following the Club was successful.**

Return type bool

start(*load_boards=True*, *load_posts=True*, *load_notices=True*, *load_media=True*, *load_from_artist=True*, *load_to_artist=False*, *load_talk=False*, *load_comments=False*, *follow_all_clubs=True*)
Creates internal cache.

This is the main process that should be run.

Parameters

- **load_boards** (bool) – Whether to load up all of a Club's boards.
- **load_posts** (bool) – Whether to load up all of the Posts of a Board.
- **load_notices** (bool) – Whether to load up all of the Notice posts.
- **load_media** (bool) – Whether to load up all of the Media posts.
- **load_from_artist** (bool) – Whether to load up all of the From Artist posts.
- **load_to_artist** (bool) – Whether to load up all of the To Artist posts.
- **load_talk** (bool) – Whether to load up all of the talk posts.
- **load_comments** (bool) – Whether to load up comments.
- **follow_all_clubs** (bool) – Whether to follow all clubs that are not followed.

Warning: All Clubs and Notifications are always created no matter what. The params are dependent on each other. Attempting to load `notices/media/from artist/to artist/talk/comments` will not work without `load_posts` set to `True`. Attempting to load any posts will not work without `load_boards` set to `True`.

Raises *UCube.error.InvalidToken* If the token was invalid.

Raises *UCube.error.InvalidCredentials* If the user credentials were invalid or not provided.

Raises *UCube.error.BeingRateLimited* If the client is being rate-limited.

Raises *UCube.error.LoginFailed* Login process had failed.

2.2 UCubeClientAsync

class UCube.**UCubeClientAsync**(*loop=<_UnixSelectorEventLoop running=False closed=False debug=False>*, ***kwargs*)

Asynchronous UCube Client that Inherits from *UCubeClient*.

Parameters

- **loop** – Asyncio Event Loop
- **kwargs** – Args for *UCubeClient*.

loop

Asyncio Event Loop

async check_new_notifications() → List[*UCube.models.notification.Notification*]

Checks and returns new notifications for every club.

Compares with the already existing notifications. This will also create the posts associated with the notification so they can be used efficiently.

This is a coroutine and must be awaited.

Returns **A list of new Notifications.**

Return type List[*models.Notification*]

async check_token_works() → bool
Check if a token is valid and will set the general information about the client if it is.

This is a coroutine and must be awaited.

Returns (bool) True if the token works.

async fetch_all_clubs(*clubs_per_page: int = 99999*, *page_number: int = 1*) → List[*UCube.models.club.Club*]
Fetch all Clubs from the UCube API.

This is a coroutine and must be awaited.

Returns A list of Clubs

Return type List[*models.Club*]

async fetch_board_posts(*board_slug: str*, *feed=False*, *posts_per_page: int = 99999*, *page_number: int = 1*) → List[*UCube.models.post.Post*]
Retrieve a list of Posts from a board.

This is a coroutine and must be awaited.

Parameters

- **board_slug** (*str*) – The slug (unique identifier) of a board to search the feed for.
- **feed** (*bool*) – Whether to use the feeds endpoint. The feeds endpoint would retrieve the user information of a post.
- **posts_per_page** (*int*) – The amount of posts to retrieve per page.
- **page_number** (*int*) – The page number when paginating.

Returns A list of Posts

Return type List[*models.Post*]

async fetch_club_boards(*club_slug: str*) → List[*UCube.models.board.Board*]
Retrieve a list of Boards from a Club.

This is a coroutine and must be awaited.

Parameters **club_slug** (*str*) – The slug (Unique Identifier) of a Club.

Returns A list of Boards

Return type List[*models.Board*]

async fetch_club_notifications(*club_slug: str*, *notifications_per_page: int = 99999*, *page_number: int = 1*) → List[*UCube.models.notification.Notification*]
Retrieve a list of Notifications from a club.

This is a coroutine and must be awaited.

Parameters

- **club_slug** (*str*) – The slug (unique identifier) of a club to search the notifications for.
- **notifications_per_page** (*int*) – The amount of notifications to retrieve per page.
- **page_number** (*int*) – The page number when paginating.

Returns A list of Notifications

Return type List[*models.Notification*]

async fetch_post(*post_slug: str*, *load_comments=False*) → Optional[*UCube.models.post.Post*]
Fetch a Post by it's slug.

This is a coroutine and must be awaited.

Parameters

- **post_slug** (str) – The post's unique identifier.
- **load_comments** (bool) – Whether to load up the comments of the Post.

Returns **The Post Object if there is one.**

Return type *models.Post*

async fetch_post_comments(*post_slug: str*, *comments_per_page: int = 99999*, *page_number: int = 1*) → List[*UCube.models.comment.Comment*]
Retrieve a list of Comments from a Post.

This is a coroutine and must be awaited.

Parameters

- **post_slug** (*str*) – The slug (unique identifier) of a Post to search the Comments for.
- **comments_per_page** (*int*) – The amount of notifications to retrieve per page.
- **page_number** (*int*) – The page number when paginating.

Returns **A list of Comments**

Return type List[*models.Comment*]

async follow_club(*club_slug: str*) → bool
Follow a club.

This is a coroutine and must be awaited.

Parameters **club_slug** (str) – The unique identifier of the club to follow.

Returns **Whether following the Club was successful.**

Return type bool

async start(*load_boards=True*, *load_posts=True*, *load_notices=True*, *load_media=True*, *load_from_artist=True*, *load_to_artist=False*, *load_talk=False*, *load_comments=False*, *follow_all_clubs=True*)
Creates internal cache.

This is the main process that should be run.

This is a coroutine and must be awaited.

Parameters

- **load_boards** (bool) – Whether to load up all of a Club's boards.
- **load_posts** (bool) – Whether to load up all of the Posts of a Board.
- **load_notices** (bool) – Whether to load up all of the Notice posts.
- **load_media** (bool) – Whether to load up all of the Media posts.
- **load_from_artist** (bool) – Whether to load up all of the From Artist posts.
- **load_to_artist** (bool) – Whether to load up all of the To Artist posts.
- **load_talk** (bool) – Whether to load up all of the talk posts.

- **load_comments** (bool) – Whether to load up comments.
- **follow_all_clubs** (bool) – Whether to follow all clubs that are not followed.

Warning: All Clubs and Notifications are always created no matter what. The params are dependent on each other. Attempting to load `notices/media/from artist/to artist/talk/comments` will not work without `load_posts` set to `True`. Attempting to load any posts will not work without `load_boards` set to `True`.

Raises *UCube.error.InvalidToken* If the token was invalid.

Raises *UCube.error.InvalidCredentials* If the user credentials were invalid or not provided.

Raises *UCube.error.BeingRateLimited* If the client is being rate-limited.

Raises *UCube.error.LoginFailed* Login process had failed.

Raises `asyncio.exceptions.TimeoutError` Waited too long for a login.

CHAPTER

THREE

MODELS

3.1 BaseModel

class UCube.models.**BaseModel**(*slug: str*, *name: Optional[str] = None*)

The Base Class for Model objects.

x == y

Checks if two models have the same slug.

x != y

Checks if two models do not have the same slug.

str(x)

Returns the model's name.

Parameters

- **slug** (str) – The unique identifier of the model.
- **name** (Optional[str]) – The name of the object.

slug

The unique identifier.

Type str

name

The name of the object.

Type Optional[str]

static remove_html(*content: str*) → str

Removes HTML tags of the html content and returns a cleansed version.

Parameters **content** (str) – The raw html content to remove html tags from.

Returns **A cleansed string with no HTML.**

Return type str

3.2 Club

class UCube.models.**Club**(*artist_name: str*, *create_image*, ***options*)

A Club object that represents a UCube Club.

Inherits from *BaseModel*

> **Warning:** It is not suggested to create a Club manually, but rather through the following method: *UCube.objects.create_club*

`The information retrieved on a Club is directly from the UCube API and altered to fit this class.`

x == y

Checks if two Clubs have the same slug.

x != y

Checks if two Clubs do not have the same slug.

str(x)

Returns the Club's name.

Parameters

- **slug** (str) – The unique identifier of the Club.
- **artist_name** (str) – The artist's name. Could also be a group.
- **create_image** (UCube.create_image) – The method to call for creating an image.
- **color_1** (str) – The first color hex code.
- **color_2** (str) – The second color hex code.
- **artist_logo_file** (dict) – The raw information about the artist logo image.
- **thumbnail_file** (dict) – The raw information about the thumbnail.
- **thumbnail_small_file** (dict) – The raw information about the smaller version of the thumbnail.
- **external_url** (str) – Any external url to the Club.
- **register_datetime** (str) – The datetime for when the club was first registered.

artist_name

The artist's name. Could also be a group.

Type str

artist_logo

The artist logo as an Image. May be None.

Type Optional[*Image*]

color_one

The first color hex code.

Type str

color_two

The second color hex code.

Type str

thumbnail_image
The thumbnail Image for the Club.

Type Optional[*Image*]

small_thumbnail_image
The smaller version of the thumbnail Image for the Club.

Type Optional[*Image*]

external_url
Any external url to the Club.

Type str

registered_time
The datetime for when the club was first registered.

Type str

boards
A Dict of Boards that belong to the Club with the slug as the key.

Type Dict[str, *Board*]

notifications
A list of Notifications that belong to the Club.

Type List[*Notification*]

3.3 Board

class UCube.models.**Board**(***options*)
A Board object that represents a UCube Board.

Inherits from *BaseModel*

> **Warning:** It is not suggested to create a Board manually, but rather through the following method: *UCube.objects.create_board*

The information retrieved on a Board is directly from the UCube API and altered to fit this class.

x == y
Checks if two Boards have the same slug.

x != y
Checks if two Boards do not have the same slug.

str(x)
Returns the Board's name.

Parameters

- **slug** (str) – The unique identifier of the Board.
- **active_flag** (bool) – Whether the board is active.

- **club_slug** (str) – The club slug that the board belongs to.

active_flag
Whether the board is active.

Type bool

club_slug
The club slug that the board belongs to.

Type str

posts

Type Dict[str, *Post*]

3.4 Post

class UCube.models.**Post**(*create_image*, *create_video*, *create_user*, ***options*)
A Post object that represents a UCube Post.

Inherits from *BaseModel*

> **Warning:** It is not suggested to create a Post manually, but rather through the following method: *UCube.objects.create_post*

The information retrieved on a Post is directly from the UCube API and altered to fit this class.

x == y
Checks if two Posts have the same slug.

x != y
Checks if two Posts have the same slug.

str(x)
Returns the Post's content.

Parameters

- **slug** (str) – The unique identifier of the Post.
- **create_image** (UCube.create_image) – The method to call for creating an image. You can also use a custom method.
- **create_video** (UCube.create_video) – The method to call for creating a video. You can also use a custom method.
- **create_user** (UCube.create_user) – The method to call for creating a user. You can also use a custom method.
- **content** (str) – The body content of the post with HTML.
- **board_slug** (str) – The board slug that the post belongs to.
- **media** (List[dict]) – Media that belongs to a post.
- **base_url** (str) – The Base URL of the image site. This is especially useful if there are several base urls for an image if UCube is using an external image host.

slug
The unique identifier of the Post.

Type str

content
The post content (without HTML).

Type str

board_slug
The Post Board slug.

Type str

videos
A list of videos that belong to the post.

Type List[*Video*]

images
A list of images that belong to the Post.

Type List[*Image*]

comment_count
The amount of comments.

Type int

posted_at
When the post was created.

Type str

user
The user that created the Post.

Type Optional[*User*]

comments
A list of comments that belong to the Post.

Type List[*Comment*]

3.5 User

class UCube.models.**User**(*slug: str*, *base_url: str*, ***options*)
A User object that represents a UCube User.

Inherits from *BaseModel*

> **Warning:** It is not suggested to create a User manually, but rather through the following method: *UCube.objects.create_user*

The information retrieved on a User is directly from the UCube API and altered to fit this class.

x == y
Checks if two Users have the same slug.

x != y
Checks if two Users do not have the same slug.

str(x)
Returns the User's name.

Parameters

- **slug** (str) – The unique identifier of the User..
- **nick_name** (str) – The name of the user.
- **base_url** (str) – The Base URL of the image site. This is especially useful if there are several base urls for an image if UCube is using an external image host.
- **profile_path** (str) – The path to the profile photo of the user.

profile_image
The URL to the profile photo of the user.

Type Optional[str]

3.6 Notification

class UCube.models.**Notification**(*uid*, ***options*)
A Notification object that represents a UCube Notification.

Inherits from *BaseModel*

> **Warning:** It is not suggested to create a Notification manually, but rather through the following method: *UCube.objects.create_notification*

The information retrieved on a Notification is directly from the UCube API and altered to fit this class.

x == y
Checks if two Notifications have the same slug.

x != y
Checks if two Notifications do not have the same slug.

str(x)
Returns the Notification's title.

Parameters **uid** (str) – The unique identifier (basically a Slug) of the Notification.

body
The content of the notification.

Type str

topic_slug
The slug of the topic.

Type str

channel_type
The channel type of the Notification.

Type str

created_at
The timestamp of when the Notification was created.

Type str

direct_link
Direct link to access the content.

Type str

data_type
The type of the data.

Type str

club_name
The club's name.

Type str

club_slug
The club's slug.

Type str

post_slug
The slug of the Post.

Type str

board_name
The name of the Board.

Type str

board_slug
The slug (unique identifier) of the board.

Type str

board_type
The type of the Board.

Type str

3.7 Comment

class UCube.models.**Comment**(*uid*, *create_user*, ***options*)
A Comment object that represents a UCube Comment.

Inherits from *BaseModel*

> **Warning:** It is not suggested to create a Comment manually, but rather through the following method: *UCube.objects.create_comment*

The information retrieved on a Comment is directly from the UCube API and altered to fit this class.

`x == y`
Checks if two Comments have the same slug.

`x != y`
Checks if two Comments do not have the same slug.

`str(x)`
Returns the Comment's content.

Parameters

- **uid** – The unique identifier (basically a Slug) of the Comment.
- **create_user** (UCube.create_user) – The method to call for creating a user. You can also use a custom method.

`comment_count`
Amount of comments that belong to the comment.

Type int

`content`
Content of the comment

Type str

`parent_slug`
The parent slug/uid if there was one.

Type str

`created_at`
The timestamp for when the comment was created.

Type str

`user`
The User that created the comment.

Type Optional[*User*]

3.8 Image

`class` UCube.models.`Image`(*path*, *base_url*, ***options*)
An Image object that represents a UCube Image/Logo.

Inherits from *BaseModel*

> **Warning:** It is not suggested to create an Image manually, but rather through the following method: *UCube.objects.create_image*

The information retrieved on an Image is directly from the UCube API and altered to fit this class.

`x == y`
Checks if two Images have the same path.

`x != y`
Checks if two Images have the same path.

str(x)
Returns the Image's name.

int(x)
Returns the Image's size.

Parameters

- **slug** (str) – The unique identifier of the image. This can be the full link to the image.
- **path** (str) – The path to the direct link of the image.
- **base_url** (str) – The Base URL of the image site. This is especially useful if there are several base urls for an image if UCube is using an external image host.
- **size** (int) – The size of the Image. This may be set to 0 if it is unknown.
- **width** (int) – The width of the Image. This may be set to 0 if it is unknown.
- **height** (int) – The height of the Image. This may be set to 0 if it is unknown.

slug
The unique identifier of the image. This can be the full link to the image.

Type str

base_url
The Base URL of the image site.

Type str

path
The path to the direct link of the image.

Type str

size
The size of the Image.

Type int

width
The width of the Image. This may be set to 0 at times.

Type int

height
The height of the Image. This may be set to 0 at times.

Type int

3.9 Video

class UCube.models.**Video**(*url*, ***options*)
A Video object that represents a UCube Video.

Inherits from *BaseModel*

> **Warning:** It is not suggested to create a Video manually, but rather through the following method: *UCube.objects.create_video*

The information retrieved on a Video is directly from the UCube API and altered to fit this class.

x == y
Checks if two Videos have the same url (slug for a video).

x != y
Checks if two Videos have the same url (slug for a video).

str(x)
Returns the Video's slug (url).

Parameters

- **slug** (str) – The unique identifier of the video. This can be the full link to the video.
- **url** (str) – The URL of the video.
- **name** (str) – The title of the video. This can also be passed in as title.
- **image** (str) – The link to the thumbnail.

slug
The unique identifier of the video. This can be the full link to the video.

Type str

name
The title of the video.

Type str

url
The URL to the Video.

Type str

thumbnail
The thumbnail of the video.

Type str

CHAPTER

FOUR

MODEL CREATION

UCube.objects.**create_board**(*raw_board: dict*) → *UCube.models.board.Board*

Parameters **raw_board** (*dict*) – The raw information about a board directly from a UCube API endpoint.

Returns **A Board Model**

Return type *UCube.models.Board*

UCube.objects.**create_club**(*raw_club: dict*) → *UCube.models.club.Club*

Parameters **raw_club** (*dict*) – The raw information about a club directly from a UCube API endpoint.

Returns **A Club Model**

Return type *UCube.models.Club*

UCube.objects.**create_comment**(*raw_comment*) → *UCube.models.comment.Comment*

Parameters **raw_comment** (*dict*) – The raw information about a comment directly from a UCube API endpoint.

Returns **A Comment Model**

Return type *UCube.models.Comment*

UCube.objects.**create_image**(*raw_image*) → *UCube.models.image.Image*

Parameters **raw_image** (*dict*) – The raw information about an image directly from a UCube API endpoint.

Returns **An Image Model**

Return type *UCube.models.Image*

UCube.objects.**create_notification**(*raw_notification*) → *UCube.models.notification.Notification*

Parameters **raw_notification** (*dict*) – The raw information about a notification directly from a UCube API endpoint.

Returns **A Notification Model**

Return type *UCube.models.Notification*

UCube.objects.**create_post**(*raw_post*) → *UCube.models.post.Post*

Parameters **raw_post** (*dict*) – The raw information about a post directly from a UCube API endpoint.

Returns A Post Model

Return type *UCube.models.Post*

UCube.objects.**create_user**(*raw_user*) → *UCube.models.user.User*

Parameters **raw_user** (*dict*) – The raw information about a user directly from a UCube API endpoint.

Returns A User Model

Return type *UCube.models.User*

UCube.objects.**create_video**(*raw_video*) → *UCube.models.video.Video*

Parameters **raw_video** (*dict*) – The raw information about a video directly from a UCube API endpoint.

Returns A Video Model

Return type *UCube.models.Video*

CHAPTER

FIVE

EXCEPTIONS

5.1 Invalid Token

exception UCube.**InvalidToken**
An Exception Raised When an Invalid Token was Supplied.

5.2 InvalidCredentials

exception UCube.**InvalidCredentials**(*msg: str = 'The credentials for a token or a username/password could not be found.'*)
An Exception raised when no valid credentials were supplied.

5.3 Something Went Wrong

exception UCube.**SomethingWentWrong**(*msg: str = 'UCube came across an unexpected issue.'*)
An Exception raised when something went wrong.

5.4 LoginFailed

exception UCube.**LoginFailed**(*msg: str = 'The login process for UCube had failed.'*)
An Exception raised when the login failed.

5.5 Page Not Found

exception UCube.**PageNotFound**(*url*)
An Exception Raised When a link was not found.

Parameters **url** (str) – The link that was not found.

5.6 Being Rate Limited

exception UCube.**BeingRateLimited**
An Exception Raised When UCube Is Being Rate-Limited.

5.7 NoHookFound

exception UCube.**NoHookFound**(*msg: str = 'No Hook was passed into the UCube client.'*)
An Exception raised when a loop for the hook was started but did not actually have a hook method.

CHAPTER

SIX

GET ACCOUNT TOKEN

There are two ways to log in. The first way is using a username and password to login which will automatically refresh your token. The second way is getting your account token manually and being logged in for a very short amount of time.

In order to get your account token, go to https://www.united-cube.com/ and Inspect Element (F12). Then go to the *Network* tab and filter by *XHR*. Then refresh your page (F5) and look for `popup` or `clubs` under *XHR*. Under Headers, scroll to the bottom and view the request headers. You want to copy everything past *Authorization: Bearer*.

For example, you may see (This is just an example): `Authorization: Bearer ABCDEFGHIJKLMNOPQRSTUVWXYZ` Then `ABCDEFGHIJKLMNOPQRSTUVWXYZ` would be your auth token for UCube. It is suggested to have the auth token as an environment variable.

The first method to log in (username & password) is the best way and SHOULD be the way that you log in.

CHAPTER

SEVEN

ASYNCHRONOUS USAGE

Example can be found at https://github.com/MujyKun/united-cube/blob/master/examples/asynchronous.py

CHAPTER

EIGHT

SYNCHRONOUS USAGE

Example can be found at https://github.com/MujyKun/united-cube/blob/master/examples/synchronous.py

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

U

UCube.objects, 21

INDEX

A

active_flag (*UCube.models.Board attribute*), 14
artist_logo (*UCube.models.Club attribute*), 12
artist_name (*UCube.models.Club attribute*), 12

B

base_url (*UCube.models.Image attribute*), 19
BaseModel (*class in UCube.models*), 11
BeingRateLimited, 24
Board (*class in UCube.models*), 13
board_name (*UCube.models.Notification attribute*), 17
board_slug (*UCube.models.Notification attribute*), 17
board_slug (*UCube.models.Post attribute*), 15
board_type (*UCube.models.Notification attribute*), 17
boards (*UCube.models.Club attribute*), 13
boards (*UCube.UCubeClient attribute*), 1
body (*UCube.models.Notification attribute*), 16

C

cache_loaded (*UCube.UCubeClient attribute*), 1
channel_type (*UCube.models.Notification attribute*), 16
check_new_notifications() (*UCube.UCubeClientAsync method*), 7
check_new_notifications() (*UCube.UCubeClientSync method*), 5
check_token_works() (*UCube.UCubeClientAsync method*), 7
check_token_works() (*UCube.UCubeClientSync method*), 5
Club (*class in UCube.models*), 12
club_name (*UCube.models.Notification attribute*), 17
club_slug (*UCube.models.Board attribute*), 14
club_slug (*UCube.models.Notification attribute*), 17
clubs (*UCube.UCubeClient attribute*), 1
color_one (*UCube.models.Club attribute*), 12
color_two (*UCube.models.Club attribute*), 12
Comment (*class in UCube.models*), 17
comment_count (*UCube.models.Comment attribute*), 18
comment_count (*UCube.models.Post attribute*), 15
comments (*UCube.models.Post attribute*), 15
content (*UCube.models.Comment attribute*), 18
content (*UCube.models.Post attribute*), 15
create_board() (*in module UCube.objects*), 21
create_club() (*in module UCube.objects*), 21
create_comment() (*in module UCube.objects*), 21
create_image() (*in module UCube.objects*), 21
create_notification() (*in module UCube.objects*), 21
create_post() (*in module UCube.objects*), 21
create_user() (*in module UCube.objects*), 22
create_video() (*in module UCube.objects*), 22
created_at (*UCube.models.Comment attribute*), 18
created_at (*UCube.models.Notification attribute*), 17

D

data_type (*UCube.models.Notification attribute*), 17
direct_link (*UCube.models.Notification attribute*), 17

E

external_url (*UCube.models.Club attribute*), 13

F

fetch_all_clubs() (*UCube.UCubeClientAsync method*), 8
fetch_all_clubs() (*UCube.UCubeClientSync method*), 5
fetch_board_posts() (*UCube.UCubeClientAsync method*), 8
fetch_board_posts() (*UCube.UCubeClientSync method*), 5
fetch_club_boards() (*UCube.UCubeClientAsync method*), 8
fetch_club_boards() (*UCube.UCubeClientSync method*), 5
fetch_club_notifications() (*UCube.UCubeClientAsync method*), 8
fetch_club_notifications() (*UCube.UCubeClientSync method*), 6
fetch_post() (*UCube.UCubeClientAsync method*), 8
fetch_post() (*UCube.UCubeClientSync method*), 6
fetch_post_comments() (*UCube.UCubeClientAsync method*), 9

fetch_post_comments() (*UCube.UCubeClientSync method*), 6
follow_club() (*UCube.UCubeClientAsync method*), 9
follow_club() (*UCube.UCubeClientSync method*), 6

G

get_board() (*UCube.UCubeClient method*), 2
get_club() (*UCube.UCubeClient method*), 2
get_comment() (*UCube.UCubeClient method*), 2
get_notification() (*UCube.UCubeClient method*), 2
get_post() (*UCube.UCubeClient method*), 2
get_user() (*UCube.UCubeClient method*), 2

H

height (*UCube.models.Image attribute*), 19

I

Image (*class in UCube.models*), 18
images (*UCube.models.Post attribute*), 15
InvalidCredentials, 23
InvalidToken, 23

L

LoginFailed, 23
loop (*UCube.UCubeClientAsync attribute*), 7

M

module
    UCube.objects, 21

N

name (*UCube.models.BaseModel attribute*), 11
name (*UCube.models.Video attribute*), 20
NoHookFound, 24
Notification (*class in UCube.models*), 16
notifications (*UCube.models.Club attribute*), 13
notifications (*UCube.UCubeClient attribute*), 2

P

PageNotFound, 23
parent_slug (*UCube.models.Comment attribute*), 18
path (*UCube.models.Image attribute*), 19
Post (*class in UCube.models*), 14
post_slug (*UCube.models.Notification attribute*), 17
posted_at (*UCube.models.Post attribute*), 15
posts (*UCube.models.Board attribute*), 14
posts (*UCube.UCubeClient attribute*), 1
profile_image (*UCube.models.User attribute*), 16

R

registered_time (*UCube.models.Club attribute*), 13
remove_html() (*UCube.models.BaseModel static method*), 11
replace() (*UCube.UCubeClient static method*), 2

S

size (*UCube.models.Image attribute*), 19
slug (*UCube.models.BaseModel attribute*), 11
slug (*UCube.models.Image attribute*), 19
slug (*UCube.models.Post attribute*), 14
slug (*UCube.models.Video attribute*), 20
small_thumbnail_image (*UCube.models.Club attribute*), 13
SomethingWentWrong, 23
start() (*UCube.UCubeClientAsync method*), 9
start() (*UCube.UCubeClientSync method*), 6
stop() (*UCube.UCubeClient method*), 3

T

thumbnail (*UCube.models.Video attribute*), 20
thumbnail_image (*UCube.models.Club attribute*), 13
topic_slug (*UCube.models.Notification attribute*), 16

U

UCube.objects
    module, 21
UCubeClient (*class in UCube*), 1
UCubeClientAsync (*class in UCube*), 7
UCubeClientSync (*class in UCube*), 5
url (*UCube.models.Video attribute*), 20
User (*class in UCube.models*), 15
user (*UCube.models.Comment attribute*), 18
user (*UCube.models.Post attribute*), 15
users (*UCube.UCubeClient attribute*), 2

V

verbose (*UCube.UCubeClient attribute*), 1
Video (*class in UCube.models*), 19
videos (*UCube.models.Post attribute*), 15

W

web_session (*UCube.UCubeClient attribute*), 1
width (*UCube.models.Image attribute*), 19