MIS Bot Documentation

Release 4.0.0

Kanishk Singh

May 21, 2020

Installation

CHAPTER 1

Installation

1.1 Pre-requisites

1.1.1 Telegram

You need a Telegram Bot Token, which you can get by messaging Botfather on Telegram. You can get your bot token by asking BotFather anytime, but for now, save your bot token somewhere because we will require it in the next step.

Since you're gonna run the bot, you are the administrator of the bot. You require your telegram chat ID to use admin commands like `/push`, `/revert`, etc. To get your chat ID, message FalconGate with the command `/get_my_id` and save the 9-digit number that comes back, we will require this in the next step.

1.1.2 Docker

Linux

Learn how to install docker here.

Learn how to install docker-compose here.

Windows

On Windows 10 Home, I found using Docker-Toolbox to be less than ideal since the VM docker-toolbox uses doesn't expose the container's ports to the host machine, which wouldn't let my bot's container talk to the Splash container (you'll know what it is in next step).

So I chose to run docker inside a Ubuntu VM which I run through Vagrant. If you're on Windows 10 Pro or Enterprise, just using Docker Desktop will work out fine. Download it here.

However, if you're on Windows 10 Home edition, follow below steps.

Download & install Vagrant from here.

After installation is done run this command in the folder where you'll be downloading or cloning this repository (e.g. `Documents`): .. code-block:: none vagrant init kwilczynski/ubuntu-16.04-docker vagrant up

A new `Vagrantfile` will be created inside the directory where you run the above command (`Documents` in our case). Open it and add this line before the last line (which says `end`) .. code-block:: none config.vm.synced_folder "MIS-Bot/", "/srv/MIS-Bot"

This will sync the cloned/downloaded project folder with the ubuntu virtual machine.

Now you can run this command to get inside your Ubuntu VM with docker installed: .. code-block:: none vagrant ssh

When you're done doing docker stuff, you can type `exit` to come out of the Ubuntu VM and then run `vagrant halt` to shutdown the VM.

1.2 Development Setup

1.2.1 Cloning the repository

Clone the repository by running (in `Documents` or some other directory, whichever you created the vagrant VM in the previous step):

git clone https://github.com/ArionMiles/MIS-Bot/

Open the project directory in terminal with

cd MIS-Bot/

Run the below command to create folders which will be required by our bot:

mkdir -p files/captcha/

1.2.2 Setting environment variables

Rename `example.env` to `.env`:

`mv example.env .env`

Edit `.env` and enter your `Bot Token`, your `Chat ID` (See *Pre-requisites*), and if you are gonna run it with webhooks, you'll need to enter your server address as `URL` (without HTTP:// or trailing /, like this: `X.X.X.X`).

Enter `SPLASH_INSTANCE` as `http://splash:8050`

1.2.3 Running with Long Polling

If you're testing this locally, change DEBUG value to `True` in this file which allows you to use long-polling instead of webhooks and makes development easier.

1.2.4 Running with Webhooks

If you're gonna deploy this on a remote server, and are expecting lots of users, it's better to use webhooks rather than long-polling.

You need SSL certificates in order to use webhooks. Telegram servers communicate only via HTTPS, with long polling, the telegram servers take care of it, but since we're using webhooks, we need to take care of it. We'll be using a self-signed certificate. To create a self-signed SSL certificate using openssl, run the following command:

```
openssl req -newkey rsa:2048 -sha256 -nodes -keyout private.key -x509 -days
3650 -out cert.pem
```

The openssl utility will ask you a few details. Make sure you enter the correct FQDN! If your server has a domain, enter the full domain name here (eg. sub.example.com). If your server only has an IP address, enter that instead. If you enter an invalid FQDN (Fully Qualified Domain Name), you won't receive any updates from Telegram but also won't see any errors!

-Source

Move the `private.key` and `cert.pem` generated to the `files/` directory so that they're picked up by `telegram_bot.py`: .. code-block:: none mv private.key cert.pem files/

1.2.5 Running the docker container

On the first run, docker will build an image for our container, it can take significant amount of time depending on your internet connection, so wait while docker downloads the python, splash images and installs all the dependencies.

Start the container by running this from the root directory of the project: .. code-block:: none docker-compose up

and after everything is installed, the bot should be up.

Cool! Now you've got the bot running, start experimenting, create new features, the possibilities are endless!

ON GCP: Switch to your project, go to Compute Instance > VPC Network > Firewall Rules and change "default-http" rule's Protocol/ports value from "tcp:80" to all to allow tg webhook to work

CHAPTER 2

Admin

misbot.admin.**push_notification**(*bot*, *update*)
Starts Push notification conversation. Asks for message and transfers control to *notification_message()*

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

Returns NOTIF_MESSAGE

Return type int

misbot.admin.**notification_message**(*bot*, *update*, *user_data*)
Ask for confirmation, stores the message in user_data dictionary, transfer control to *notification_confirm()*

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object
- **user_data** (*dict*) – User data dictionary

Returns NOTIF_CONFIRM

Return type int

misbot.admin.**notification_confirm**(*bot*, *update*, *user_data*)
Sends message if "Yes" is sent. Aborts if "No" is sent. Sends a message with statistics like users reached, time taken after sending push notification.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object
- **user_data** (*dict*) – User data dictionary

Returns ConversationHandler.END

Return type int

misbot.admin.**revert_notification**(*bot*, *update*)
Delete a previously sent push notification. Ask for `uuid` of message to delete and pass control to *ask_uuid()*

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

misbot.admin.**ask_uuid**(*bot*, *update*, *user_data*)
Store the `uuid`, send confirmation message and pass control to *confirm_revert()*

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object
- **user_data** (*dict*) – Conversation data

misbot.admin.**confirm_revert**(*bot*, *update*, *user_data*)
If Yes, revert the specified message for all users and send a summary of the operation to admin.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object
- **user_data** (*dict*) – Conversation data

misbot.admin.**clean_all_attendance_records**(*bot*, *update*)
Activated by `/clean` command. See *misbot.mis_utils.clean_attendance_records()* to see what this does.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

CHAPTER 3

Attendance Target

misbot.attendance_target.**attendance_target**(*bot*, *update*)

Like `until_eighty()`, but with user specified target attendance percentage which is stored in the `Misc` table. If target isn't set, asks users whether they'd like to and passes control to *select_yn()*

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

Returns SELECT_YN

Return type int

misbot.attendance_target.**select_yn**(*bot*, *update*)

If user replies no, ends the conversation, otherwise transfers control to *input_target()*.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

Returns INPUT_TARGET

Return type int

misbot.attendance_target.**input_target**(*bot*, *update*)

If the user reply is a int/float and between 1-99, stores the figure as the new attendance target.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

Returns ConversationHandler.END

Return type int

misbot.attendance_target.**edit_attendance_target**(*bot*, *update*)
Edit existing attendance target. Shows current target and transfers control to *update_target()*

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

Returns UPDATE_TARGET

Return type int

misbot.attendance_target.**update_target**(*bot*, *update*)
Takes the sent figure and sets it as new attendance target.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

Returns ConversationHandler.END

Return type int

CHAPTER 4

Bunk Command

misbot.bunk.**bunk**(*bot*, *update*)

Starting point of bunk_handler. Sends a KeyboardMarkup (https://core.telegram.org/bots#keyboards) Passes control to *bunk_choose()*

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

Returns CHOOSING

Return type int

misbot.bunk.**bunk_choose**(*bot*, *update*, *user_data*)

Removes keyboardMarkup sent in previous handler.

Stores the response (for Lectures/Practicals message sent in previous handler) in a `user_data` dictionary with the key *"stype"*. `user_data` is a user relative dictionary which holds data between different handlers/functions in a ConversationHandler.

Selects the appropriate table (Lecture or Practical) based on `stype` value. Checks if records exist in the table for a user and sends a warning message or proceeds to list names of all subjects in the table.

Passes control to *bunk_input()*

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object
- **user_data** (*dict*) – User data dictionary

Returns ConversationHandler.END if no records else INPUT

Return type int

misbot.bunk.**bunk_input**(*bot*, *update*, *user_data*)

Stores index of the chosen subject in `user_data['index']` from `update.message.text`. Passes control to *bunk_calc()*

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object
- **user_data** (*dict*) – User data dictionary

Returns ConversationHandler.END if message is "Cancel" else CALCULATING

Return type int

misbot.bunk.**bunk_calc**(*bot*, *update*, *user_data*)

Calculate the % drop/rise with the previously given values from the user and send the response to the user.

`user_data` contains: `type`, `index`, `figures`.

Incorrect no. of arguments resets the state and user is asked for input again.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object
- **user_data** (*dict*) – User data dictionary

Returns None if incorrect values else INPUT

Return type None or int

CHAPTER 5

Decorators

misbot.decorators.**signed_up**(*func*)

Checks if user is signed up or not. If user isn't signed up, sends a message asking them to register.

Parameters **func** (*func*) – A telegram bot function

Returns None if user isn't registered or the function which decorator is applied on.

Return type None or func

misbot.decorators.**admin**(*func*)

Checks if the user sending the command/message is an admin or not. If user isn't an admin, their username (or chat ID, if unregistered) is logged and a message is sent saying that the incident has been reported.

Parameters **func** (*func*) – A telegram bot function

Returns None if user isn't registered or the function which decorator is applied on.

Return type None or func

misbot.decorators.**premium**(*tier=1*)

Checks if the user sending the command/message is a premium member or not.

Parameters **tier** (*int*) – The tier level of the user permitted to use the feature

Returns None if user doesn't satisfy tier level or the function which decorator is applied on.

Return type None or func

CHAPTER 6

General Functions

misbot.general.**start**(*bot*, *update*)
Initial message sent to all users. Starts registration conversation, passes control to *credentials()*

misbot.general.**register**(*bot*, *update*, *user_data*)
Let all users register with their credentials. Similar to *start()* but this function can be invoked by /register command.

If user's chatID & DOB are already present in database then ends the conversation. Otherwise, if only chatID is present, then stores StudentID (PID) in user_data dict & gives control to *parent_login()* function.

If both conditions are false, then asks user to input Student details (PID & Password) and gives control to *credentials()*

misbot.general.**credentials**(*bot*, *update*, *user_data*)
Store user credentials in a database. Takes student info (PID & password) from update.message.text and splits it into Student_ID & Password and checks if they are correct with *misbot.mis_utils.check_login()* and stores them in the Chat table. Finally, sends message asking users to enter DOB and gives control to *parent_login()* after storing Student_ID (PID) in user_data dict.

misbot.general.**parent_login**(*bot*, *update*, *user_data*)
user_data dict contains Student_ID key from *credentials()*. Extracts DOB from update.message.text and checks validity using *misbot.mis_utils.check_parent_login()* before adding it to database. Finally, sends a message to the user requesting them to start using /attendance or /itinerary commands.

misbot.general.**delete**(*bot*, *update*)
Delete a user's credentials if they wish to stop using the bot or update them.

misbot.general.**cancel**(*bot*, *update*)
Cancel registration operation (terminates conv_handler)

misbot.general.**unknown**(*bot*, *update*)
Respond to incomprehensible messages/commands with some canned responses.

misbot.general.**help_text**(*bot*, *update*)
Display help text.

misbot.general.**tips**(*bot*, *update*)
Send a random tip about the bot.

misbot.general.**error_callback**(*bot*, *update*, *error*)
Simple error handling function. Handles PTB lib errors

misbot.general.**subscription**(*bot*, *update*)
Sends text detailing the subscription model

CHAPTER 7

MIS Functions

misbot.mis_utils.**bunk_lecture**(*n*, *tot_lec*, *chatID*, *stype*, *index*)
Calculates % drop/rise if one chooses to bunk certain lectures.

Parameters

- **n** (*int*) – number of lectures for a subject to bunk
- **tot_lec** (*int*) – total lectures conducted for that subject
- **chatID** (*int | str*) – user's unique 9-digit ChatID from telegram
- **stype** (*str*) – Subject type (Lectures or Practicals)
- **index** (*int*) – Index of the user-selected subject from list of subjects

Returns Percentage drop/rise

Return type float

misbot.mis_utils.**until_x**(*chatID*, *target*)
Calculates the no. of lectures user must attend in order to get overall attendance to their specified target.

Parameters

- **chatID** (*int | str*) – user's unique 9-digit ChatID from telegram
- **target** (*float*) – attendance percentage target

Returns Number of lectures to attend

Return type int

misbot.mis_utils.**check_login**(*username*, *password*)
Checks if user input for their credentials is correct for the student portal.

Parameters

- **username** (*str*) – student's PID (format: XXXNameXXXX) where X - integers
- **password** (*str*) – student's password for student portal

Returns True for correct credentials, false otherwise.

Return type bool

misbot.mis_utils.**check_parent_login**(*username*, *dob*)
Checks if user input for their credentials is correct for parent's portal.

Parameters

- **username** (*str*) – student's PID (format: XXXNameXXXX) where X - integers
- **dob** (*str*) – User's Date of Birth

Returns True for correct credentials, false otherwise.

Return type bool

misbot.mis_utils.**crop_image**(*path*)
Crop image if the image height is > 800px.

Parameters **path** (*str*) – image file path

Returns True for image height larger than 800px in length.

Return type bool

misbot.mis_utils.**clean_attendance_records**()
Delete all lectures and practical records from the DB. To be used at the beginning of a new semester so that `/bunk` command doesn't display lectures of previous semester(s).

Returns Number of records deleted from Lecture and Practical tables.

Return type tuple

misbot.mis_utils.**get_user_info**(*chat_id*)
Give user data.

Parameters **chat_id** (*str*) – 9-Digit unique user ID

Returns Dictionary of all user data

Return type dict

misbot.mis_utils.**solve_captcha**(*session_id*)
Solve captcha using `securimage_solver` library. Downloads the image from the securimage_endpoint and feeds it to securimage_solver lib.

Parameters **session_id** (*str*) – Session cookie

Returns Captcha answer

Return type str

misbot.mis_utils.**rate_limited**(*bot*, *chat_id*, *command*)
Checks if user has made a request in the past 5 minutes.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **chat_id** (*str*) – 9-Digit unique user ID
- **command** (*str*) – Telegram command

Returns True if user has made a request in past 5 mins, else False

Return type bool

misbot.mis_utils.**get_subject_name**(*chat_id*, *index*, *stype*)
Return name of subject for a given user

Parameters

- **chat_id** (*str*) – 9-Digit unique user ID
- **index** (*int*) – Index of the user-selected subject from list of subjects
- **stype** (*str*) – Subject type (Lectures or Practicals)

Returns Subject name

Return type str

misbot.mis_utils.**build_menu**(*buttons*, *n_cols*, *header_buttons=None*, *footer_buttons=None*)

misbot.mis_utils.**get_misc_record**(*chat_id*)

Returns Misc record for a user

Parameters **chat_id** (*str*) – 9-Digit unique user ID

CHAPTER 8

Push Notifications

misbot.push_notifications.**get_user_list**()
Retrieves the chatID of all users from Chat table. A tuple is returned which is then unpacked into a list by iterating over the tuple.

Returns List of user chat IDs

Return type list

misbot.push_notifications.**get_bot**()
Create an instance of the telegram.bot.Bot object.

Returns Telegram bot object.

Return type telegram.bot.Bot

misbot.push_notifications.**push_message_threaded**(*message*, *user_list*)
Use ThreadPoolExecutor to send notification message asynchronously to all users. Before sending the message, we create a record of the sent message in the PushMessage DB model.

We pass the message_uuid generated from created the record of the message previously and pass it to the *push_t()*.

After messages have been sent, we bulk commit all the PushNotification records created within *push_t()* to our database.

Parameters

- **message** (*str*) – Notification message
- **user_list** (*list*) – List of all bot users

Returns Time taken to send messages and the message_uuid

Return type tuple

misbot.push_notifications.**push_t**(*bot*, *message*, *message_uuid*, *chat_id*)
Sends the message to the specified chat_id. Records the message_id received into PushNotification DB Model. Appends the record object to list_of_objs to be bulk committed after all messages have been sent through ThreadPoolExecutor

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **message** (*str*) – Notification message
- **message_uuid** (*str*) – UUID of the notification message created in `push_message_threaded`
- **chat_id** (*int | str*) – 9-Digit unique user ID

`misbot.push_notifications.`**delete_threaded**(*message_id_list*, *user_list*)

Use `ThreadPoolExecutor` to delete notification message asynchronously for all users.

Parameters

- **message_id_list** (*list*) – List of message ids stored in our DB.
- **user_list** (*list*) – List of all bot users

Returns Time taken to send all delete requests

Return type float

`misbot.push_notifications.`**delete**(*bot*, *message_id*, *chat_id*)

Sends a delete request for a particular `chat_id` and `message_id`

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **message_id** (*int*) – The message ID for the notification message
- **chat_id** (*int | str*) – 9-Digit unique user ID

CHAPTER 9

Attendance, Itinerary, Result Functions

misbot.spider_functions.**attendance**(*bot*, *update*)

Core function. Fetch attendance figures from Aldel's MIS. Runs AttendanceSpider for registered users and passes it their `Student_ID` (PID), `Password`, & `ChatID` (necessary for `AttendancePipeline`)

`AttendanceSpider` creates a image file of the format: `<Student_ID>_attendance.png` File is deleted after being sent to the user. If the file is unavailable, error message is sent to the user.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

misbot.spider_functions.**results**(*bot*, *update*)

Fetch Unit Test results from the Aldel MIS. Core function. Fetch Test Reports from Aldel's MIS. Runs `ResultsSpider` for registered users and passes it their `Student_ID` (PID) & `Password`.

ResultsSpider creates a image file of the format: `<Student_ID>_tests.png` File is deleted after being sent to the user. If the file is unavailable, error message is sent to the user.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

misbot.spider_functions.**itinerary**(*bot*, *update*, *args*)

Core function. Fetch detailed attendance reports from Aldel's MIS (Parent's Portal). Runs `ItinerarySpider` for registered users and passes it their `Student_ID` (PID) & `Password`.

`AttendanceSpider` creates a image file of the format: `<Student_ID>_itinerary.png` If `args` are present, full report is sent in the form of a document. Otherwise, it is cropped to the past 7 days using *misbot.mis_utils.crop_image()* and this function stores the resultant image as: `<Student_ID>_itinerary_cropped.png` and returns True.

File is deleted after sent to the user. If the file is unavailable, error message is sent to the user.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object
- **args** (*tuple*) – User supplied arguments

misbot.spider_functions.**profile**(*bot*, *update*)

Fetch profile info from the Aldel MIS. Core function. Runs `ProfileSpider` for registered users and passes it their `Student_ID` (PID) & `Password`.

ProfileSpider creates a image file of the format: `<Student_ID>_profile.png` File is deleted after being sent to the user. If the file is unavailable, error message is sent to the user.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

CHAPTER 10

Until80 and similar functions

misbot.until_func.**until_eighty**(*bot*, *update*)
Calculate number of lectures you must consecutively attend before you attendance is 80%

If *misbot.mis_utils.until_x()* returns a negative number, attendance is already over 80%

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object

misbot.until_func.**until**(*bot*, *update*, *args*)
Like *until_eighty()* but user supplies the number.

Parameters

- **bot** (*telegram.bot.Bot*) – Telegram Bot object
- **update** (*telegram.update.Update*) – Telegram Update object
- **args** (*tuple*) – User supplied arguments

Returns None

Return type None

CHAPTER 11

Attendance Spider

class scraper.spiders.attendance_spider.**AttendanceSpider**(*username*, *password*, *chatID*, **args*, ***kwargs*)

Scrape attendance figures from `http://report.aldel.org/student/attendance_report.php` and store the figures in database with `scraper.pipelines.LecturePipeline` and `scraper.pipelines.PracticalPipeline`

Parameters **InitSpider** (*Spider*) – Base Spider with initialization facilities

name = 'attendance'

allowed_domains = ['report.aldel.org']

login_page = 'http://report.aldel.org/student_page.php'

start_urls = ['http://report.aldel.org/student/attendance_report.php']

init_request()
This function is called before crawling starts.

login(*response*)
Generate a login request.

check_login_response(*response*)
Check the response returned by a login request to see if we are successfully logged in.

parse(*response*)
Send a SplashRequest and forward the response to *parse_result()*

parse_result(*response*)
Downloads and saves the attendance report in `files/<Student_ID>_attendance.png` format.

Also scrapes every attendance record from the webpage and passes it to `LecturePipeline` and `PracticalPipeline`.

scraper.spiders.attendance_spider.**scrape_attendance**(*username*, *password*, *chatID*)
Run the spider multiple times, without hitting `ReactorNotRestartable` exception. Forks own process.

Parameters

- **username** (*str*) – student's PID (format: XXXNameXXXX) where X - integers
- **password** (*str*) – student's password for student portal
- **chatID** (*str*) – 9-Digit unique user ID

CHAPTER 12

Itinerary Spider

class scraper.spiders.itinerary_spider.**ItinerarySpider**(*username*, *dob*, *chatID*, *uncropped=False*, **args*, ***kwargs*)

Take screenshot of http://report.aldel.org/parent/itinenary_attendance_report.php and send it to the user via scraper.pipelines.ItineraryScreenshotPipeline

Parameters **InitSpider** (*Spider*) – Base Spider with initialization facilities

name = 'itinerary'

allowed_domains = ['report.aldel.org']

login_page = 'http://report.aldel.org/parent_page.php'

start_urls = ['http://report.aldel.org/parent/itinenary_attendance_report.php']

init_request()
This function is called before crawling starts.

login(*response*)
Generate a login request.

check_login_response(*response*)
Check the response returned by a login request to see if we are successfully logged in.

parse(*response*)
Send a SplashRequest and forward the response to *parse_result()*

parse_result(*response*)
Downloads and saves the attendance report in files/<Student_ID>_itinerary.png format.

scraper.spiders.itinerary_spider.**scrape_itinerary**(*username*, *dob*, *chatID*, *uncropped=False*)

Run the spider multiple times, without hitting ReactorNotRestartable exception. Forks own process.

Parameters

- **username** (*str*) – student's PID (format: XXXNameXXXX) where X - integers

- **dob** (*str*) – User's Date of Birth
- **chatID** (*str*) – 9-Digit unique user ID
- **uncropped** (*bool*) – Whether the user wants full report or for last 7-8 days

CHAPTER 13

Results Spider

class scraper.spiders.results_spider.**ResultsSpider**(*username*, *password*, *chatID*, **args*, ***kwargs*)

Take screenshot of `http://report.aldel.org/student/test_marks_report.php` and send it to the user via `scraper.pipelines.ResultsScreenshotPipeline`

Parameters **InitSpider** (*Spider*) – Base Spider with initialization facilities

name = 'results'

allowed_domains = ['report.aldel.org']

login_page = 'http://report.aldel.org/student_page.php'

start_urls = ['http://report.aldel.org/student/test_marks_report.php']

init_request()
This function is called before crawling starts.

login(*response*)
Generate a login request.

check_login_response(*response*)
Check the response returned by a login request to see if we are successfully logged in.

parse(*response*)
Send a SplashRequest and forward the response to *parse_result()*

parse_result(*response*)
Downloads and saves the attendance report in `files/<Student_ID>_tests.png` format.

scraper.spiders.results_spider.**scrape_results**(*username*, *password*, *chatID*)
Run the spider multiple times, without hitting `ReactorNotRestartable` exception. Forks own process.

Parameters

- **username** (*str*) – student's PID (format: XXXNameXXXX) where X - integers
- **password** (*str*) – student's password for student portal
- **chatID** (*str*) – 9-Digit unique user ID

CHAPTER 14

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

s

Index