**Authentication {host}/api/auth/**

**login**

pass username and password (POST) get user info + JWT

**signup**

pass username, password and password2 (POST) get user info + JWT

**refresh**

pass users refresh token to get (GET) new access token

**profile**

pass authorization header to get all user info (GET)

pass avatar or author\_pseudo to change either (PATCH)

if you simply pass query param ?user=<int> along with authorization token it will return someones profile info (it works only if request user is authenticated, so basically this shit is useless and probably should not be used)

**writer**

pass author\_pseudo to create writer profile (POST)

**subscribe**

pass author\_id to subscribe to author (POST)

**subscriptions**

pass authorization token to get list of writers you’re subscribed to (GET)

**notify**

pass author\_id to either receive notifications from writer (about him creating new story) or not (POST)

**Stories {host}/api/stories/**

**all\*\***

get all stories (GET), you can also pass query\_params such as: search\_prompt, genres (id’s), tag (id), sort\_by (likes\_count, -likes\_count, -created, created, -views\_counter, views\_counter - use any if needed) those with minus sign return sorted in manner from greater to lesser (if I’m not mistaken)

***{host}/api/stories/all/?search\_prompt=<string>&genres=<list of int’s> e.g. 1, or 1,2,3, &tag=<int>&sort\_by=<string>***

!!!! Filtration is evaluated with AND operator !!!!

**manipulate**

pass genre\* (string), title\* (string), body\* (string), image, tags (string with comma e.g.: tag,tag,tag, - valid; tag,tag,tag – invalid, each tag name should end with comma as I’m splitting by comma on backend) (POST) – to create story

pass the exact same shit to update story (PATCH)

pass story\_id to delete it (DELETE)

\* - must-have

**get\_genres\*\***

fetches all genres, sorted by popularity (most used come first) (GET)

**get\_tags\*\***

fetches all tags, sorted by popularity (most used come first) (GET)

**single**

fetch single story query\_param: story\_id (GET)

**get\_writer\_stories\*\***

fetch stories made by user (if he has writer profile) pass Authorization header (GET)

if u pass author\_id as query param it will return stories made by that user (GET) + auth header

**get\_viewed\*\***

fetch stories viewed by user, pass Authorization header (GET)

**get\_liked\*\***

fetch stories liked by user, pass Authorization header (GET)

**react**

react to story, like or dislike (POST), request body should contain story\_id and type (type can be either like or dislike – idk why I did that, just use it as is)

**Comment, {host}/api/comments/**

**all\*\***

fetch all comments for a story (GET), if you pass query\_param: parent\_comment\_id=<int> it will return all replies for the comment if it exists; story\_id=<int> will return comments for a story.

**manipulate**

create or delete comment (POST or DELETE), gotta pass auth token and post data should be:

story\_id \*=<int>,

comment\_body\*=<string>, to create comment for a story, and:

parent\_comment\_id\*=<int>

comment\_body\*=<string>, to create reply to some comment

\* - must-have

there is a thing about replies to comments, to avoid replies nesting I made it work (at least, made it look like that) like it does on YouTube, when you reply to a reply it doesn’t stack in that tree-type structure, there is main comment (speaking in a language of back end – it means comment whose field “parent\_id” is None) and there are replies to that comment, and all of the replies TO replies on that main comment will be treated as replies to main comment, but on back end I add structure like: @<creator\_of\_comment\_that\_I\_reply\_to> + comment\_body, so that It can be seen which comment is replying to which. That’s just a heads up because you will notice that some replies have that @<username> structure at the beginning.

For DELETE request u just pass comment\_id in request body with auth token. ONLY STORY CREATOR AND COMMENT CREATOR CAN DELETE CERTAIN COMMENT.

**react**

sets user reaction to a comment (POST), auth should be passed and request body should contain:

comment\_id=<int>

type=<string>, same shit as with stories type should be either “like” or “dislike”

**Notifications {host}/api/notifications/**

**all\*\***

fetch all user notifications (GET) requires auth token

**mark\_as\_read**

gotta make (POST) request to mark notification as read, gotta pass auth and request body should be:

type=<string> type for notifications should any of these: “sc”, “scom”, “cr”, “ao”.

note\_id=<int> notification id

This is what type codes mean:

sc = story created

scom = story commented

cr = comment replied

ao = administrative notifications

You will understand which type should be assigned to which notification, when you fetch user notifications they all come with their type specified.

Media is at {host}/api/media/

**story\_fis:** images for story tiles,

**avatars:** user avatars

**logo:** for site logos

at **{host}/api/media/logo/logoipsum-330.svg** resides placeholder logo.

**\*\* - MEANS THAT THERE IS PAGINATION FOR THESE ENDPOINTS (they all share the same query parameter for specifying page number which is - ?page=<int>)**