**Evidence of time:**

**https://docs.google.com/document/d/1udSUKYzf88SMSymsr0m1SWHhcDrAmEKpv4mj45Ei\_uQ/edit?usp=sharing**

**Meeting 1**

22/09/2021 | 1:30pm - 2:50pm | Wednesday

Next Meeting: 28/092021|7:00pm - ??? |Tuesday

Discussion Topics:

What is meant by assumptions?

* Guesses about how things should work. Things which aren’t clarified by the specifications.
* Examples: word limits for user inputs (on the spot suggestion)

What is a standup?

* May be addressed in week 2 lectures

Anyone know how to use GitLab Issue Boards?

* May be addressed in week 2 lectures

How will storing data work?

* Use of the data\_store variable
* Focus is on how to use data\_store rather than how it works
* initial\_object in data\_store.py needs further work

Attaining skills required for Project?

* Completing lab exercises. Week 2 lab exercises will help with testing
* Following leads in the forum
* Asking questions in the forum
* Not being afraid to ask for help from group members

Order in which functions need to be implemented?

* Theory: maybe the order of the functions set out in the table indicates the order in which they should be completed.
* After some talk:

1. data\_store and clear\_vl
2. auth\_register\_vl
3. auth\_login\_vl
4. channels\_create\_vl
5. channel\_ join and channel\_invite
6. channel \_details\_vl, channel\_listall\_vl, channel\_list\_vl
7. channel\_messages\_vl

What’s API?

* Might be important
* API stands for application programming interface. It allows two applications to communicate with one another to access data. Every action you take on your phone, like sending a direct message or checking the score of the baseball game, uses an API to access and deliver that information.

Function Allocation?

* Draft 1:

|  |  |  |  |
| --- | --- | --- | --- |
| Jeremy | Darren | Leon and Ishan | Jimmy |
| auth\_register\_vl  channel\_details\_vl | auth\_register\_vl  channel\_details\_vl | channels\_create\_vl  clear\_vl  channels\_list\_vl  Channels\_listall\_vl | data\_store  channel\_join\_vl  channel\_invite\_vl |

* Draft 2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Jeremy | Darren | Leon | Jimmy | Ishan |
| auth\_register\_vl  channel\_details\_vl | auth\_login\_vl  channel\_messages\_vl | channels\_create\_vl  channel\_join\_vl | data\_store  clear\_vl  channel\_invite\_vl | channels\_list\_vl  channels\_listall\_vl |

**Meeting 2**

28/09/2021 | 7:00pm - 8:00pm | Tuesday

Next Meeting: 5/09/2021|7:00pm - ??? |Tuesday

Discussion Topics:

Progress check:

* Leon has finished create channel
* Jeremy has finished channel details
* Ishan has done good progress on channel\_list\_v1

Discussed deadlines:

Channel invite should be done asap (due: **29/09 or latest 30/09**)

* No update on channel invite yet? Jimmy?

Channel list:(due 30/09)

* Ishan has made some progress

Channel listall: (due 30/09)

* Same as before

Channel details: (due: 29/09)

* Jeremy has made some progress

Channel message (due: 30/09)

* Darren has read the specs required

Monday 19th is due

Channel join:

issues - how to format channel global owner

**Meeting 3**

5/10/2021 | 7:00 - 9:00| Tuesday

Discussion Topics:

Went through how to demonstrate the information to the tutors

Explained each other’s codes to make sure we were aware of exactly how the functions ran

Tried to think of things we could improve in retrospect

* Possibly used the wrong error type
* Better variable names (according to pylint regulations)
* Incorrect application of permission\_type
* Using helper functions for lines of code everyone uses

Planned to go through the spec on our own and then assign tasks and resolve any questions on our tute on Friday

Explained the tests we ran for our functions and why we ran them

**Meeting 4**

09/10/2021 | 3:00pm - 4:00pm | Saturday

Next Meeting: 12/10/2021 |7:00pm - ??? |Tuesday

HTTP Wrapping

Discussing what we know about wrapping the functions

Grouping functions together

1.

* channel/leave/v1 - Leon

2.

* channel/addowner/v1 - Leon
* channel/removeowner/v1 - Leon

3.

* message/send/v1 - Jimmy
* message/edit/v1 - Jimmy
* message/remove/v1 - Jimmy

4.

* dm/create/v1 - Leon
* dm/list/v1 - Ishan
* dm/remove/v1 - Jimmy
* dm/details/v1 - Darren
* dm/leave/v1 - Darren
* dm/message/v1 - Darren
* message/senddm/v1 - Darren

5.

* users/all/v1 - Ishan
* user/profile/v1 - Jeremy
* user/profile/setname/v1 -Jeremy
* user/profile/setemail/v1 - Jeremy
* user/profile/sethandle/v1 - Jeremy

6.

* admin/user/remove/v1 - Ishan
* admin/user/permissionchange/v1 - Ishan

7.

* auth/logout/v1 -Darren

auth/login/v2 - Darren

auth/register/v2 -Jeremy

channels/create/v2 - Leon

channels/list/v2 - Ishan

channels/listall/v2 - Ishan

channel/details/v2 -Jeremy

channel/join/v2 - Leon

channel/invite/v2 -Jimmy

channel/messages/v2 - Darren

clear/v1 - Jimmy

data\_store - Jimmy

Priority list

1. **Wrap functions with HTTP**
2. Message create/send
3. Add/remove owners
4. Message details/other requirements

**Meeting 5**

12/10/2021 |10:30am - 12:30pm |Tuesday

‘How to use Tokens and Session id’ ideas draft:

Login

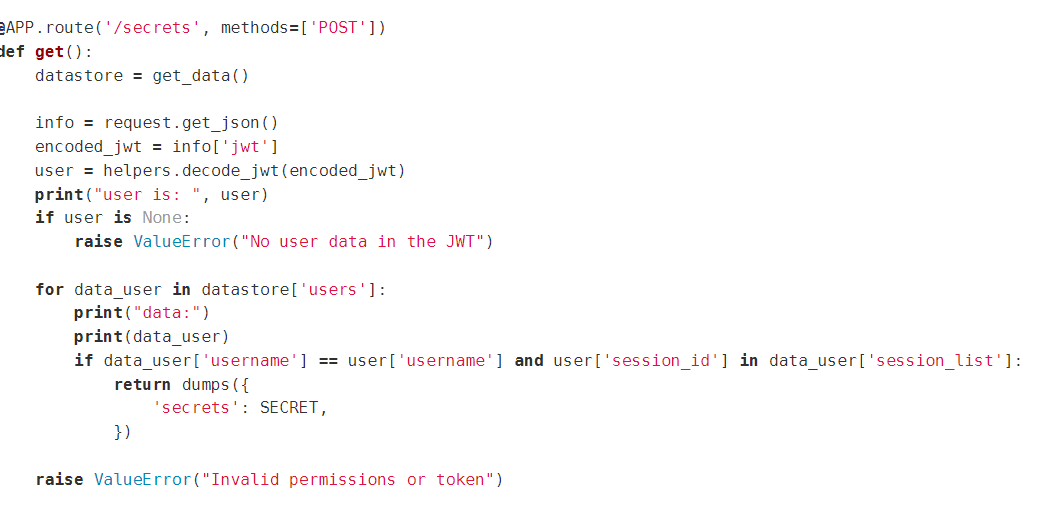
* generate a new session id using helper function.
* Find user in the database using email and password arguments
* Append session\_id to a list of sessions in the user dictionary
* Pass their handle string and the generated session\_id into the generate token function helper function
* Return generated token and auth\_user\_id

Logout

* Decode token using helper function. The helper function will return a handlestring and a session id
* Use handlestring to find the user in the database
* Look through list of sessions in the user dictionary and invalidate the session id that is the same as the one returned by the decode token helper function

For every function that has a token as an argument

* Can refer to lecture 5.2 code



* Check if the token is valid by:

- decoding the token using helper function. Helper function will return handlestring and session\_id

- then use handlestring to find user and check if the session id has been invalidated

Update:

Change of plans lol. We generating tokens with u\_id instead of handle string. Replace every mention of handle string above with u\_id

**Updated schedule**

**Darren**

* **auth/login/v2 - Darren 19/10 TODAY**
* channel/messages/v2 - Darren
* auth/logout/v1 -Darren
* dm/details/v1 - Darren
* dm/leave/v1 - Darren
* dm/message/v1 - Darren

**Leon**

* channels/create/v2 - Leon
* channel/join/v2 - Leon
* channel/leave/v1 - Leon
* channel/addowner/v1 - Leon
* channel/removeowner/v1 - Leon
* **dm/create/v1 - Leon 20/10 WEDNESDAY**

**Ishan**

* channels/list/v2 - Ishan
* channels/listall/v2 - Ishan
* dm/list/v1 - Ishan
* admin/user/remove/v1 - Ishan -> how messages are being stored
* admin/user/permissionchange/v1 - Ishan
* users/all/v1 - Ishan

**Jeremy**

* auth/register/v2 -Jeremy
* channel/details/v2 -Jeremy
* user/profile/v1 - Jeremy
* user/profile/setname/v1 -Jeremy
* user/profile/setemail/v1 - Jeremy
* user/profile/sethandle/v1 - Jeremy

**Jimmy**

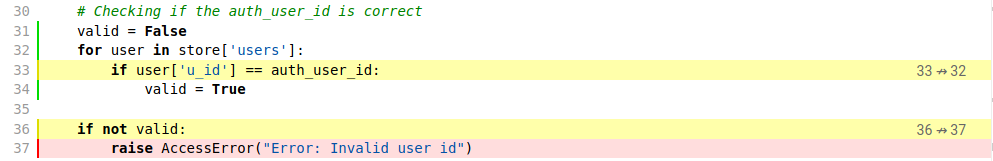
* clear/v1 - Jimmy
* channel/invite/v2 -Jimmy
* data\_store - Jimmy

* **message/send/v1 - Jimmy 20/10 WEDNESDAY**
* message/edit/v1 - Jimmy
* message/remove/v1 - Jimmy
* dm/remove/v1 - Jimmy
* **message/senddm/v1 - Jimmy 20/10 WEDNESDAY**

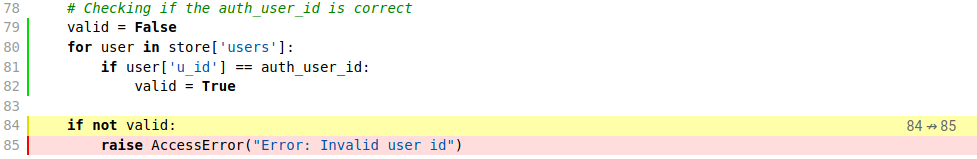
Coverage things

**channels.py: 88% coverage**

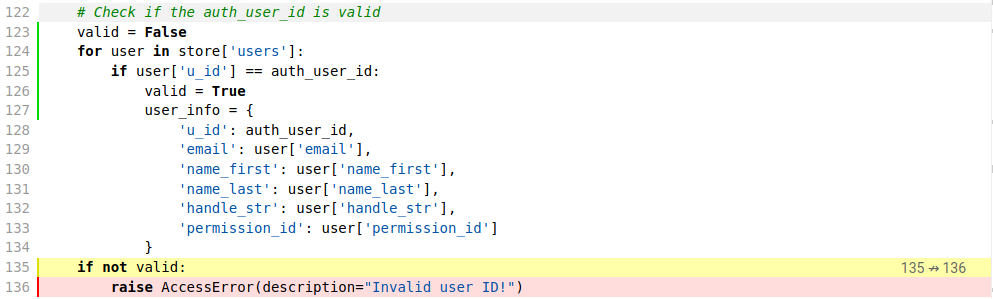
* channels/list/v2



* channels/listall/v2

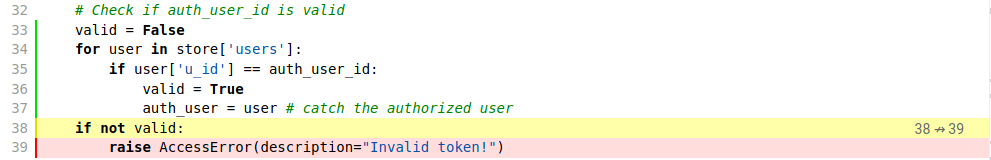


* channels/create/v2

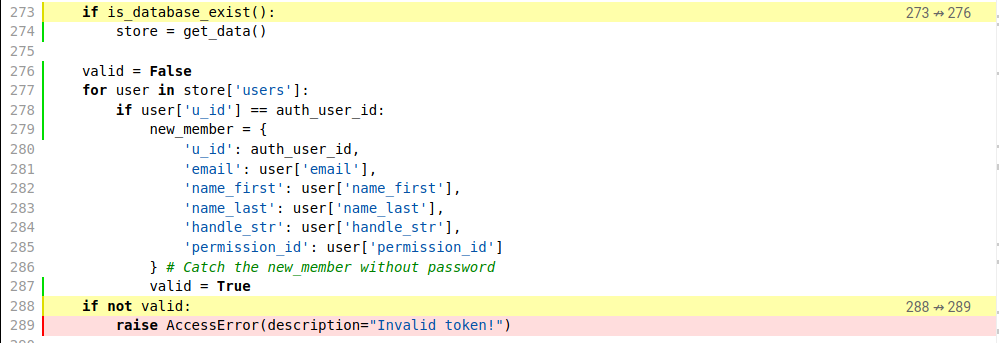


**channel.py: 95% coverage**

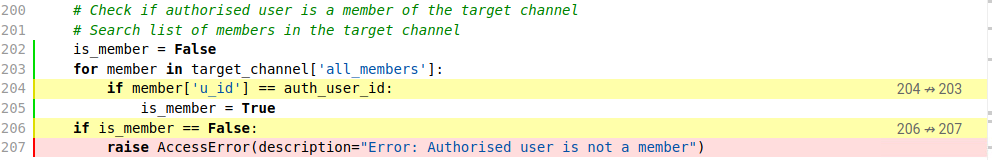
* channel/invite/v2



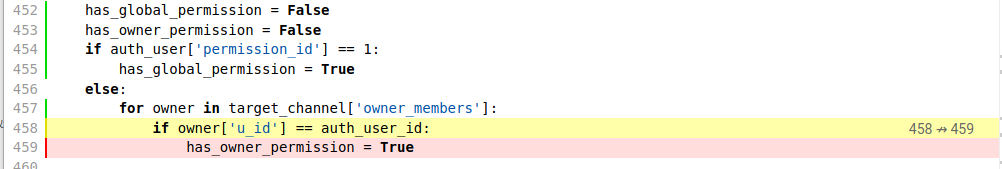
* channel/join/v2



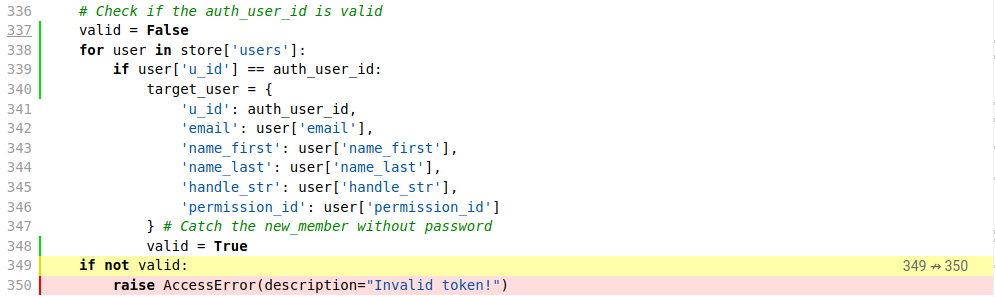
* channel/messages/v2



* channel/addowner/v1

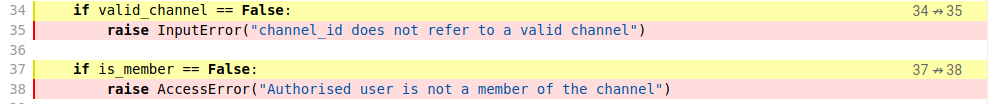


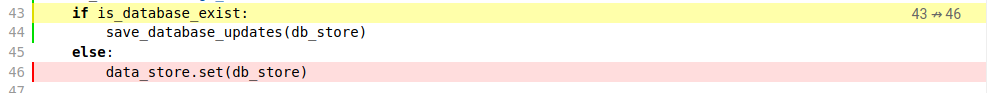
* channel/leave/v1

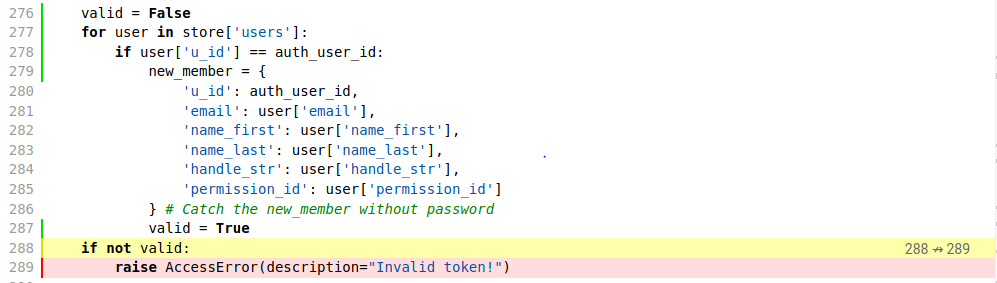


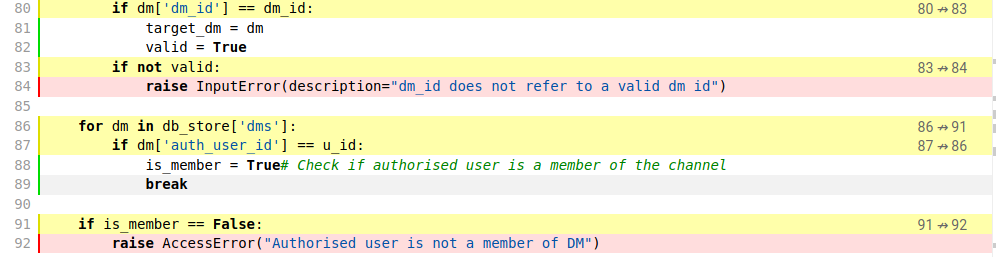
**messages.py: 81% coverage**

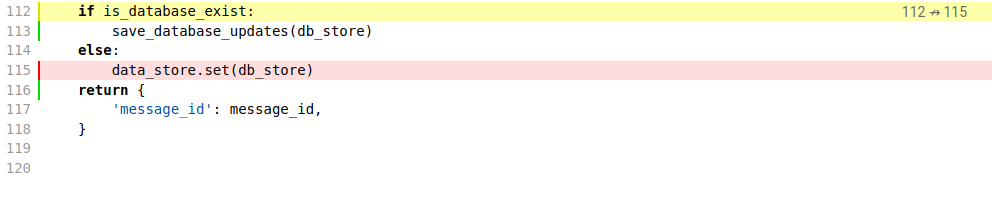
* message/send





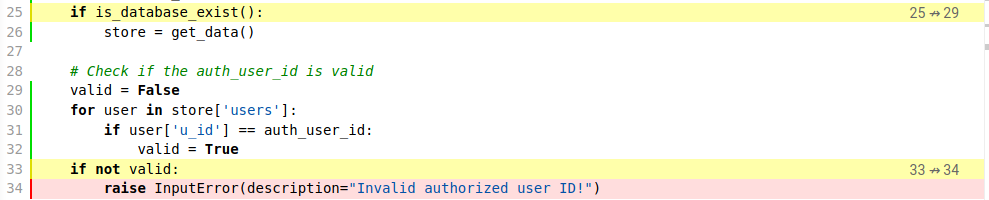


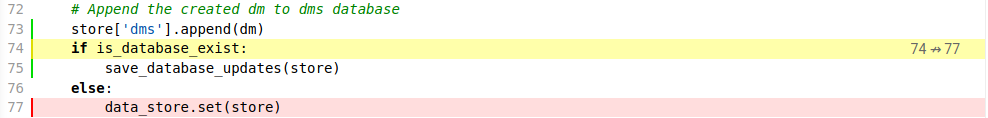




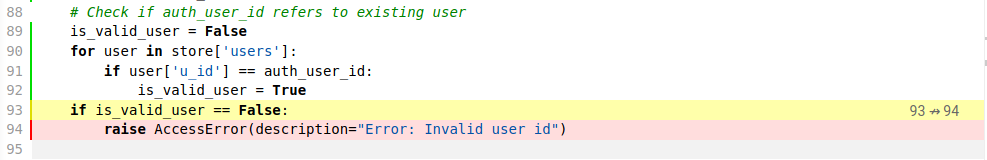
**dm.py: 94% coverage**

* dm/create/v1

****

****

* dm/details/v1



* dm/messages/v1

