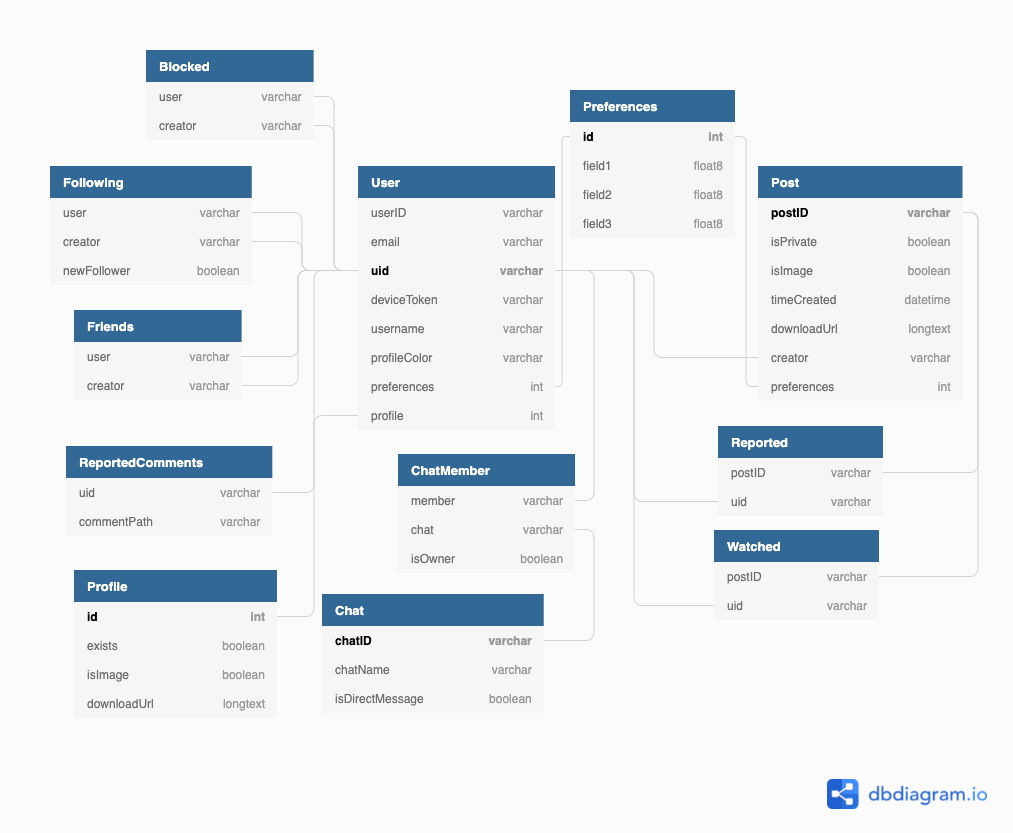
**Database:**

The main database is a SQL database. Some of the tables in this database contain “pointers” to other databases, such as Google Firestore and Google Storage.



*Figure 1.1.1 The SQL database schema*

Pointers to other databases and storages:

* “ChatID” is used to identify the collection in Firestore that contains the list of chat items.
* “PostID” is used to identify the collection in Firestore that contains the comments for the post also identified by “PostID”.
* The field “downloadUrl” found in the Profile and Post is a link that downloads a file associated with the Profile or Post entity (this file would either be an .mp4 or .png).

Notes:

* “field1”, “field2”, and “field3” are not in the actual database, they just represent the list of preference fields that are found in the database. The actual fields would be categories, such as “comedy”, “sports”, “outdoors”, etc.
* The boolean field “isImage” keeps track of whether the file found with “downloadUrl” is a video or an image. If “isImage” is set to true, then the file is an image, if it is false, then the file is a video.

**API**

This will list the urls for the new REST API. A list of operations will be listed for each endpoint (these operations are based on CRUD operations). Assume each url starts with ‘v2’

/access?token=“token” - GET

/blocked/uid - GET, POST

/blocked/uid/uid1 - DELETE

/chats/uid - GET, POST

/chats/uid/chatID - GET, POST

/chats/uid/chatID/updated - POST

/comments/postID - GET, POST

/feeds/recommendations?uid=“uid” - GET

/feeds/following?uid=“uid” - GET

/followers/uid - GET

/followers/uid/new - GET

/followings/uid - GET, POST

/followings/uid/uid1 - GET, PUT, DELETE

/posts/uid - GET, POST

/posts/uid/profile - GET, POST, PUT, DELETE

/posts/uid/postID - GET, PUT, DELETE

/preferences - GET

/preferences/uid - GET, POST

/reports/uid/profile - POST

/reports/uid/post - POST

/reports/uid/postID/comment - POST

/users - POST

/users?contains=“searchString” - GET

/users?uid=“uid”&contains=“searchString” - GET

/users/uid - GET, PUT, DELETE

/users/uid/activity - GET, PUT

/watched/postID - POST

**Notes:**

* If a client sends a GET request and there is nothing that matches the GET request, then an empty list is returned
* Only two status codes will be returned: 200 for success, 500 for errors
* For POST and PUT requests, if the request isn’t allowed for whatever reason, then a JSON object with a key “denied” and a unique value is returned. If the request is allowed, then whatever resource was created or updated is returned.
* For DELETE requests, {deleted: True} is returned for successful deletions.
* This is so that on the front end, a response with a null value indicates that the request is failed
* For each endpoint, the methods should be orders as such: POST, GET, PUT, DELETE

**Backend Structure:**

an\_app\_has\_no\_name/

app\_backend/

env/

functions/

models/

v1/

migrations/

tests/

test1.py

test2.py

test3.py

views/

view1.py

view2.py

view3.py

\_\_init\_\_.py

admin.py

apps.py

urls.py

**Notes:**

* Each API version will be it’s own app, and they will be called ‘v1’, ‘v2’, etc.
* In each app, there will be ‘views’ directory and ‘tests’ directory. Each test file corresponds to one view file.
* The directory ‘functions’ will contain all helper functions, no functions are than views will be in the ‘views’ directory