

# **CPSC 304 Project Cover Page**

Milestone #: 4

Date: Dec 1

Group Number: 119

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Stuart Chen	48414957	r4v4j	stuartcc6@gmail.com
Allan Xing	28532901	p7i4r	allanx01@students.cs.ubc.ca
Eric Fu	57440844	y2c8w	ericfu55@student.ubc.ca

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

# University of British Columbia, Vancouver

Department of Computer Science

## Description:

We made a social media platform similar to applications like Instagram, Twitter/X, and Facebook, using PostgreSQL to store all of our information. An essential feature we created was allowing users to register an account with us and login with their credentials. They are able to communicate with other users in real time using a chat room feature, and they can choose to save and delete chat groups. Users can also create posts, or delete posts that they have already created. We have created the ability to interact with different kinds of posts through viewing and liking. Users are also able to follow and unfollow each other, and if a user is following someone then they can see that person's posts on the home page.

## Difference between old and new schema:

There are some small differences between the old and new schema. In the new schema, we decided it would be easiest to turn the chatId into a SERIAL field because in PostgreSQL it would automatically increment and use a value, serving as the id for the chat. This allows us to automatically generate new ChatIds without any additional work. Also, we added a new field called directedTo in messages because one of the requirements for the project needed us to be able to update 2 non-primary key fields.

## Copy of the Schema

- Account(username: string, email: string, URL: string)

Table		
username	email	url
user1	test1@gmail.com	http://www.example1.com
user2	test2@gmail.com	http://www.example2.com
user3	test3@gmail.com	http://www.example3.com
user4	test4@gmail.com	http://www.example4.com
user5	test5@gmail.com	http://www.example5.com
test	a	a
test2	test2	test2

- Relationship: hasPermissions(user: string, perms: string)

Table	
username	perms
user1	normalUser
user2	premiumUser
user3	admin
user4	editor
user5	viewer

# University of British Columbia, Vancouver

## Department of Computer Science

- Relationship: follow(follower: string, following: string)

follow

Selected columns: follower, following

follower  
following

Submit

Table	
follower	following
user1	user2
user2	user3
user3	user4
user4	user5
user5	user1
test	user1

- Login(Username: string, Password: string)

login

Selected columns: username, password

username  
password

Submit

Table	
username	password
user1	pass1
user2	pass2
user3	pass3
user4	pass4
user5	pass5
test	\$2b\$10\$O0u7e875hwBfQtVVf/SY5uDy29d2Eqhw9N7BY2wsTZMb91US/LcJO
test2	\$2b\$10\$oARNgdqW.2KaMxBoolZUVerhFV54EjEUx.jm0/6MSHsm/Q6kXJ0U2

- Relationship: validateUser(accountUser: string, loginUser: string)

validateuser

Selected columns: accountuser, loginuser

accountuser  
loginuser

Submit

Table	
accountuser	loginuser
user1	user1
user2	user2
user3	user3
user4	user4
user5	user5

# University of British Columbia, Vancouver

## Department of Computer Science

- Permissions(type: string, cost: number)

type	cost
normalUser	0
premiumUser	10
admin	20
editor	5
viewer	0

- Hashtag(text: string, color: string)

text	color
Hashtag1	FF0000
Hashtag2	00FF00
Hashtag3	0000FF
Hashtag4	FFFF00
Hashtag5	FF00FF

- Relationship: associateHashtag(**postId: number**, hashTag: string)

postid	hashtag
1	Hashtag1
2	Hashtag2
3	Hashtag3
4	Hashtag4
5	Hashtag5
1	Hashtag2
1	Hashtag3

# University of British Columbia, Vancouver

## Department of Computer Science

- Post(postID: number, URL: string, caption: string, advertisement?: number, **createdBy: string**, type: number)

post	Table				
	postid	url	caption	createdby	timestamp
Selected columns: postid, url, caption, createdby, timestamp, type	1	http://example1.com	This is caption 1	user1	2023-10-10T07:00:00.000Z
<button>postid</button>	2	http://example2.com	This is caption 2	user2	2023-10-10T07:00:00.000Z
<button>url</button>	3	http://example3.com	This is caption 3	user3	2023-10-10T07:00:00.000Z
<button>caption</button>	4	http://example4.com	This is caption 4	user4	2023-10-10T07:00:00.000Z
<button>createdby</button>	5	http://example5.com	This is caption 5	user5	2023-10-10T07:00:00.000Z
<button>timestamp</button>	6	http://example6.com	This is caption 1	user1	2023-10-10T07:00:00.000Z
<button>type</button>	7	http://example7.com	This is caption 2	user2	2023-10-10T07:00:00.000Z
<button>Submit</button>	8	http://example8.com	This is caption 3	user3	2023-10-10T07:00:00.000Z
	9	http://example9.com	This is caption 4	user4	2023-10-10T07:00:00.000Z
	10	http://example10.com	This is caption 5	user5	2023-10-10T07:00:00.000Z
	11	http://example11.com	This is caption 1	user1	2023-10-10T07:00:00.000Z
	12	http://example12.com	This is caption 2	user2	2023-10-10T07:00:00.000Z
	13	http://example13.com	This is caption 3	user3	2023-10-10T07:00:00.000Z
	14	http://example14.com	This is caption 4	user4	2023-10-10T07:00:00.000Z
	15	http://example15.com	This is caption 5	user5	2023-10-10T07:00:00.000Z
	16	http://example1.com	This is caption 16	user1	2023-10-10T07:00:00.000Z
	17	http://example1.com	This is caption 17	user1	2023-10-10T07:00:00.000Z
	18	http://example1.com	This is caption 18	user1	2023-10-10T07:00:00.000Z

# University of British Columbia, Vancouver

## Department of Computer Science

- Relationship: likePost(postID: string, acc: string)

The screenshot shows a database interface with a table named 'likepost'. The table has two columns: 'postid' and 'acc'. The data is as follows:

postid	acc
1	user1
2	user2
3	user3
4	user4
5	user5

- Caption(caption: string, advertisement: boolean, postID: number)

The screenshot shows a database interface with a table named 'caption'. The table has three columns: 'caption', 'postid', and 'advertisement'. The data is as follows:

caption	postid	advertisement
This is caption 1	1	true
This is caption 2	2	false
This is caption 3	3	true
This is caption 4	4	false
This is caption 5	5	true

- IsA: TextPost(postID: number, content: string)

The screenshot shows a database interface with a table named 'textpost'. The table has two columns: 'postid' and 'content'. The data is as follows:

postid	content
1	This is caption 1
2	This is caption 2
3	This is caption 3
4	This is caption 4
5	This is caption 5

- IsA: VideoPost(postID: number, content: BLOB)

The screenshot shows a database interface with a table named 'videopost'. The table has two columns: 'postid' and 'content'. The data is as follows:

postid	content
11	[object Object]
12	[object Object]
13	[object Object]
14	[object Object]
15	[object Object]

# University of British Columbia, Vancouver

## Department of Computer Science

- IsA: ImagePost(postID: number, content: BLOB)

imagepost

Selected columns: postid, content

**Table**

postid	content
6	[object Object]
7	[object Object]
8	[object Object]
9	[object Object]
10	[object Object]

- Weak entity: Comment(commentsOn: number, commenter: string, timeStamp: timestamp, contents: string)

comment

Selected columns: commentpost, commenter, timestamp, contents

**Table**

commentpost	commenter	timestamp	contents
1	user1	2023-10-10T07:00:00.000Z	This is a comment 1
1	user2	2023-10-10T07:00:01.000Z	This is a comment 2
1	user3	2023-10-10T07:00:02.000Z	This is a comment 3
1	user4	2023-10-10T07:00:03.000Z	This is a comment 4
1	user5	2023-10-10T07:00:04.000Z	This is a comment 5
1	user1	2023-10-10T07:00:05.000Z	This is a comment 6

- Relationship: reply(post: number, repliedTo: string, reply: string)

reply

Selected columns: post, repliedto, reply

**Table**

post	repliedto	reply
1	2023-10-10T07:00:00.000Z	2023-10-10T07:00:01.000Z
1	2023-10-10T07:00:01.000Z	2023-10-10T07:00:02.000Z
1	2023-10-10T07:00:02.000Z	2023-10-10T07:00:03.000Z
1	2023-10-10T07:00:03.000Z	2023-10-10T07:00:04.000Z
1	2023-10-10T07:00:04.000Z	2023-10-10T07:00:05.000Z

# University of British Columbia, Vancouver

Department of Computer Science

- Relationship: likeComment(**commentPost:number** , **commentTimeStamp: timestamp**, **acc: string**)

likecomment

Selected columns: commentpost, commenttimestamp, account

commentpost  
commenttimestamp  
account

Submit

Table		
commentpost	commenttimestamp	account
1	2023-10-10T07:00:00.000Z	user1
1	2023-10-10T07:00:01.000Z	user2
1	2023-10-10T07:00:02.000Z	user3
1	2023-10-10T07:00:03.000Z	user4
1	2023-10-10T07:00:04.000Z	user5

- Chat(**chatID: number**)

chat

Selected columns: chatid

chatid

Submit

Table	
chatid	
1	
2	
3	
4	
5	

- Relationship: participates(**chatID: number**, **acc: string**)

participates

Selected columns: chatid, acc

chatid  
acc

Submit

Table	
chatid	acc
1	user1
2	user2
3	user3
4	user4
5	user5

# University of British Columbia, Vancouver

Department of Computer Science

- Weak entity: Message(chatID: number, account: string, timeAndDate: string, contents: string, directedTo: string)

The screenshot shows a user interface for managing messages. On the left, there is a form titled "message" with a dropdown menu set to "Selected columns: chatid, account, timeanddate, contents, directedto". Below the dropdown are five blue buttons labeled "chatid", "account", "timeanddate", "contents", and "directedto". At the bottom of the form is a green "Submit" button. On the right, there is a table with the following data:

Table	chatid	account	timeanddate	contents	directedto
	1	user1	2023-10-10T07:00:00.000Z		
	2	user2	2023-10-10T07:00:01.000Z		
	3	user3	2023-10-10T07:00:02.000Z		
	4	user4	2023-10-10T07:00:03.000Z		
	5	user5	2023-10-10T07:00:04.000Z		

List of all SQL queries used

all the following files in: backend/src/controller

## Account.js

```
16  const dataUser = await db.queryDbValues(`SELECT * FROM login WHERE username=$1`, [
17    |   username,
18  ]);      You, 45 minutes ago • Uncommitted changes
19
20
21
22
23
24  const insertLogin = `INSERT INTO login (username, password) VALUES ($1, $2);`;
25  const insertAccount = `INSERT INTO account (email, username, URL) VALUES ($1, $2, $3);`;
26
27  const dataLogin = await db.queryDbValues(insertLogin, [username, hashedPassword]);
28  const dataAccount = await db.queryDbValues(insertAccount, [
29    |   email,      You, 45 minutes ago • Uncommitted changes
30    |   username,
31    |   `http://localhost/user/${username}`,
32  ]);
```

Above 2 screenshots before and after (Create account and login):

Before:

# University of British Columbia, Vancouver

## Department of Computer Science

Table	
username	password
user1	pass1
user2	pass2
user3	pass3
user4	pass4
user5	pass5
test	\$2b\$10\$O0u7e875hwBfQtVVf/SY5uDy29d2Eqhw9N7BY2wsTZMb91US/LcJO
test2	\$2b\$10\$oARNgdqW.2KaMxBoolZUVerhFV54EjEUx.jm0/6MSHsm/Q6kXJ0U2
new	\$2b\$10\$.uBAykN2aUFo.bJdTaYtf.pYJCNWGHGm.nHVPP.yHRVXjxchf1VEDC
asdf	\$2b\$10\$5HDQZUqgDrqVo7xMvsl/qO0FzulFA.m3OReYQeZS9e7212SHXYUO
fdsa	\$2b\$10\$iNPN3iwtQe3HSjbe2/ezd./.WqfXF61gixwOOWEE6HFQTcp1Owryy

DURING:

### Register

Successfully registered! Login to continue.

AFTER:

# University of British Columbia, Vancouver

## Department of Computer Science

Table	
username	password
user1	pass1
user2	pass2
user3	pass3
user4	pass4
user5	pass5
test	\$2b\$10\$O0u7e875hwBfQtVVf/SY5uDy29d2Eqhw9N7BY2wsTZMb91US/LcJO
test2	\$2b\$10\$oARNgdqW.2KaMxBoolZUVerhFV54EjEUx.jm0/6MSHsm/Q6kXJ0U2
new	\$2b\$10\$.uBAykN2aUFo.bJdTaYtf.pYJCNWGm.nHVPP.yHRVXjxchf1VEDC
asdf	\$2b\$10\$5HDQZUqgDrqVo7xMvsl/q00FzulFA.m3OReYQeZS9e7212SHXYUO
fdsa	\$2b\$10\$INPN3lwtQe3HSJbe2/ezd./.WqfXF61gixwO0WEE6HFQTcp1Owryy
abc	\$2b\$10\$8V0MAsvd8LfDal52bd04t0Q7vE4kch/lxHezliT8NcvdhPcwlmIWa

```

47   const sql = `SELECT * FROM account WHERE username=$1;`;
48   const data = await db.queryDbValues(sql, values);

```

### (REQUIRED AGGREGATION WITH HAVING QUERY)

```

63   const sql = `SELECT createdBy
64     FROM Post
65   GROUP BY createdBy
66   HAVING COUNT(postID) >= $1;`;
67   const data = await db.queryDbValues(sql, values);

```

BEFORE

Aggregation with HAVING, users having number of posts SELECT createdBy FROM Post GROUP BY createdBy HAVING COUNT(postID) > 5;  
Input One enter natural number:4

AFTER

Aggregation with HAVING, users having number of posts SELECT createdBy FROM Post GROUP BY createdBy HAVING COUNT(postID) > 5;  
Input One enter natural number:4

Caption.js

```

8   const text = "INSERT INTO caption (caption, postID, advertisement) VALUES($1, $2, $3)"
19   const data = await db.queryDbValues("SELECT caption, postID FROM Caption;")

```

**University of British Columbia, Vancouver**  
Department of Computer Science

Chat.js

```
7   const createChat = `INSERT INTO chat DEFAULT VALUES RETURNING chatId;`;
8   const data = await db.queryDb(createChat);
9   const chatId = data[0].chatid;
```

For ABOVE 1 screenshot:

Before:

Table: chat	
	chatid
	1
	3
	4
	5
	6

DURING:

Chats	Count By
⊕ Create a new chat!	
⌘ Join a new chat!	

AFTER:

Table: chat	
	chatid
	1
	3
	4
	5
	6
	7

# University of British Columbia, Vancouver

## Department of Computer Science

```

10
11     const joinChat = `INSERT INTO participates (chatId, acc) VALUES ($1, $2);`;
12     await db.queryDbValues(joinChat, [chatId, username]);

```

```

24 // no need to sanitize this, already checked in auth
25 const getChats = `SELECT * FROM chat WHERE chatId IN (SELECT chatId FROM participates WHERE acc='${username}')`;
26 const chats = await db.queryDb(getChats);
27
28 chats.map(async (chat) => {
29     const getParticipants = `SELECT acc FROM participates p, chat c WHERE c.chatId=$1 AND p.chatId=$2`;
30     const participants = await db.queryDbValues(getParticipants, [
31         chat.chatid,
32         chat.chatid,
33     ]);
34 });
35
36 if (!doesParticipate, !)
37     const deleteChat = `DELETE FROM chat WHERE chatId = $1`;
38     await db.queryDbValues(deleteChat, [chatId]);
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55

```

ABOVE 1 Screenshot (REQUIRED DELETE STATEMENT)  
BEFORE:

Table: chat	
chatid	
1	
3	
4	
5	
6	
7	

Table: message				
chatid	account	timeanddate	contents	directedto
1	user1	2023-10-10T07:00:00.000Z		
3	user3	2023-10-10T07:00:02.000Z		
4	user4	2023-10-10T07:00:03.000Z		
5	user5	2023-10-10T07:00:04.000Z		
6	test2	2023-12-02T04:58:05.963Z	asdfdsas	user1
7	test	2023-12-02T05:40:48.612Z	adfdas	

DURING (we delete the chat, so it should cascade and delete the messages)

# University of British Columbia, Vancouver

## Department of Computer Science

← Chat: 7 Leave Delete

**test** 2023-12-02T05:48.612Z To: adfdas

Edit

Send

AFTER:

message

Selected columns: chatid, account, timeanddate, contents, directedto

	chatid	account	timeanddate	contents	directedto
1	user1	2023-10-10T07:00:00.000Z			
3	user3	2023-10-10T07:00:02.000Z			
4	user4	2023-10-10T07:00:03.000Z			
5	user5	2023-10-10T07:00:04.000Z			
6	test2	2023-12-02T04:58:05.963Z	asdfdsa	user1	

Submit

```

70 | const getMessages = `SELECT * FROM message WHERE chatId=$1 ORDER BY timeAndDate ASC;`;
71 | const messages = await db.queryDbValues(getMessages, [chatId]);
93 | const insertParticipates = `INSERT INTO participates (chatId, acc) VALUES ($1, $2);`;
94 | await db.queryDbValues(insertParticipates, [chatId, username]);
104 | const doesChatExist = `SELECT * FROM chat WHERE chatId=$1;`;
105 | const rows = await db.queryDbValues(doesChatExist, [chatId]);
120 | const deleteParticipates = `DELETE FROM participates WHERE chatId = $1 AND acc = $2;`;
121 | await db.queryDbValues(deleteParticipates, [chatId, username]);
136 | const createMessage = `INSERT INTO message (chatId, account, timeAndDate, contents) VALUES ($1, $2, TO_TIMESTAMP($3 / 1000.0), $4) RETURNING timeAndDate;`;
137 | const data = await db.queryDbValues(createMessage, [
138 |   chatId,
139 |   username,
140 |   timeNow,
141 |   message,
142 | ]);
158 | const editMessage = `UPDATE message SET contents=$1, directedto=$2 WHERE chatId=$3 AND account=$4 and timeAndDate=$5::timestamptz;`;
159 | const data = await db.queryDbValues(editMessage, [
160 |   contents,
161 |   directedToStr,
162 |   chatId,
163 |   account,
164 |   timeanddate,
165 | ]);

```

ABOVE 1 SCREENSHOT (REQUIRED UPDATE STATEMENT)  
BEFORE:

# University of British Columbia, Vancouver

## Department of Computer Science

message

Selected columns: chatid, account, timeanddate, contents, directedto

chatid	account	timeanddate	contents	directedto
1	user1	2023-10-10T07:00:00.000Z		
3	user3	2023-10-10T07:00:02.000Z		
4	user4	2023-10-10T07:00:03.000Z		
5	user5	2023-10-10T07:00:04.000Z		
6	test2	2023-12-02T04:58:05.963Z	asdfds	user1
8	test	2023-12-02T05:42:25.438Z	hello	
9	test	2023-12-02T05:42:29.323Z	two	

Submit

DURING:

The screenshot shows a dark-themed web interface. At the top, there's a navigation bar with links for Home, Chat, Dev, Logout, and Profile. Below the navigation, a modal dialog is open, prompting the user to "Enter new message:" with a text input field containing "What is this edited thingy?". There are "Cancel" and "OK" buttons at the bottom of the modal. In the background, the main chat area shows a message from "test" at 2023-12-02T05:42:29.323Z addressed to "two". A "Send" button is visible at the bottom right of the message input field.

AFTER:

message

Selected columns: chatid, account, timeanddate, contents, directedto

chatid	account	timeanddate	contents	directedto
1	user1	2023-10-10T07:00:00.000Z		
3	user3	2023-10-10T07:00:02.000Z		
4	user4	2023-10-10T07:00:03.000Z		
5	user5	2023-10-10T07:00:04.000Z		
6	test2	2023-12-02T04:58:05.963Z	asdfds	user1
8	test	2023-12-02T05:42:25.438Z	hello	
9	test	2023-12-02T05:42:29.323Z	What is this edited thingy?	

Submit

```

● 235   const participatesInChat = `SELECT * FROM participates WHERE acc=$1 AND chatId=$2;`;
236   const rows = await db.queryDbValues(participatesInChat, [username, chatId]);
237
238   if (rows.length > 0) {
239     const getChats = `
179       SELECT COUNT(*) AS count
180       FROM chat c, participates p
181       WHERE c.chatId = p.chatId
182       AND c.chatId IN (
183         |   SELECT chatId FROM participates WHERE acc=$1
184       )
185       GROUP BY p.acc
186       HAVING p.acc=$2;`;
187
188   const chats = await db.queryDbValues(getChats, [countText, username]);
189

```

**University of British Columbia, Vancouver**  
Department of Computer Science

FOR ABOVE 1 screenshot (REQUIRED AGGREGATION WITH GROUP BY STATEMENT)

BEFORE:

The screenshot shows a database interface. On the left, there is a selection dialog for the 'participates' table, with 'chatid' and 'acc' selected. On the right, the 'participates' table is displayed with the following data:

chatid	acc
1	user1
3	user3
4	user4
5	user5
6	test2
8	test
9	test

DURING:

The screenshot shows a 'Chats' list with two entries: '#8 Participants: test' and '#9 Participants: test'. Below the list are buttons for 'Create a new chat!' and 'Join a new chat!'. A modal dialog titled 'Count By' is open, showing a dropdown menu set to 'Username' and a text input field containing 'test'. There are 'Submit' and 'Close' buttons at the bottom of the dialog.

AFTER: Same as before as this doesn't change anything.

Dev.js

```
7 const getTableColumns = `SELECT column_name FROM information_schema.columns WHERE table_name = $1;`  
8 const columns = await db.queryDbValuesgetTableColumns([tableName]);  
30 const queryString = `SELECT ${columnStr} FROM ${tableName};`;  
31 const rows = await db.queryDb(queryString);
```

# University of British Columbia, Vancouver

## Department of Computer Science

```
47 | const getTables = `SELECT tablename FROM pg_catalog.pg_tables WHERE schemaname='public';`;
48 | const tables = await db.queryDb(getTables);
```

ABOVE 3 screenshots for the projection criteria (REQUIRED PROJECTION STATEMENT)

BEFORE:

The screenshot shows a table titled "Table: account". It has three columns: "username", "email", and "url". The "username" column contains five entries: user1, user2, user3, user4, and user5. The "email" column contains five entries: test1@gmail.com, test2@gmail.com, test3@gmail.com, test4@gmail.com, and test5@gmail.com. The "url" column contains five entries: http://www.example1.com, http://www.example2.com, http://www.example3.com, http://www.example4.com, and http://www.example5.com.

AFTER / DURING:

The screenshot shows the same "Table: account" table as before, but it now includes a new row at the top labeled "test". The "username" column for this row contains the value "a", the "email" column contains "a", and the "url" column contains "a". All other rows (user1 to user5) remain the same as in the "BEFORE" screenshot.

username	email	url
user1	test1@gmail.com	http://www.example1.com
user2	test2@gmail.com	http://www.example2.com
user3	test3@gmail.com	http://www.example3.com
user4	test4@gmail.com	http://www.example4.com
user5	test5@gmail.com	http://www.example5.com
test	a	a
test2	test2	test2
new	newaccount@gmail.com	http://localhost/user/new
asdf	asdf	http://localhost/user/asdf
fdsa	fdsa	http://localhost/user/fdsa
abc	abc	http://localhost/user/abc

Follow.js

```
11 | const data = await db.queryDbValues(
12 |   `INSERT INTO follow (follower, following) VALUES ($1, $2);`, [follower, following]
13 | );
```

before:

# University of British Columbia, Vancouver

## Department of Computer Science

follow	
Selected columns: follower, following	
follower	
following	
	Submit
Table	
follower	following
user1	user2
user2	user3
user3	user4
user4	user5
user5	user1

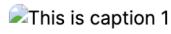


**user1**

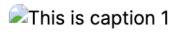
test

**1** followers **1** following **6** posts

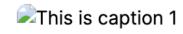
[Follow](#)



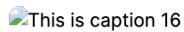
This is caption 1



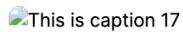
This is caption 1



This is caption 1



This is caption 16



This is caption 17



This is caption 18

After

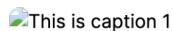


**user1**

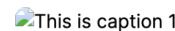
test

**2** followers **1** following **6** posts

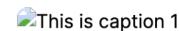
[Following](#)



This is caption 1



This is caption 1



This is caption 1



This is caption 16



This is caption 17



This is caption 18

# University of British Columbia, Vancouver

## Department of Computer Science

The screenshot shows a database interface with a search bar at the top containing the word 'follow'. Below the search bar is a table titled 'Table' with two columns: 'follower' and 'following'. The data in the table is as follows:

follower	following
user1	user2
user2	user3
user3	user4
user4	user5
user5	user1
test	user1

On the left side of the interface, there is a sidebar with a dropdown menu set to 'follow' and a 'Submit' button.

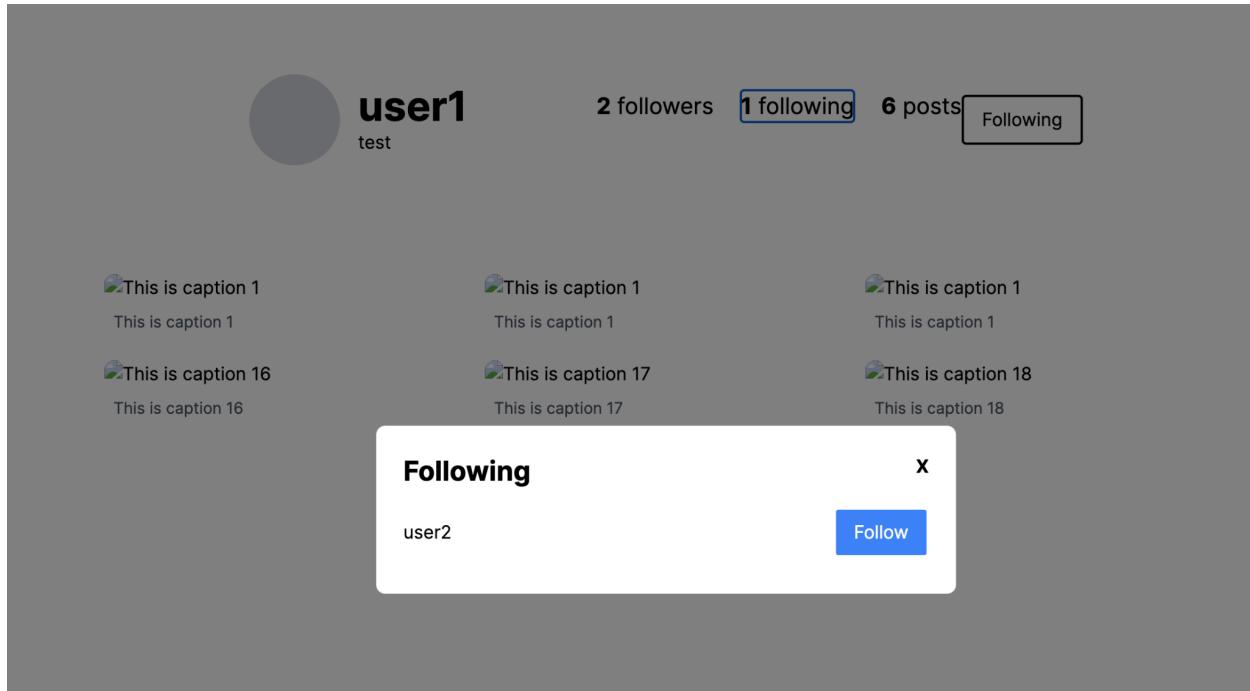
```
27 |     const data = await db.queryDbValues(`  
28 |         `DELETE FROM follow WHERE follower=$1 AND following=$2;`, [follower, following]  
29 |     );
```

reverse of the above 2 images

The screenshot shows a user profile for 'user1'. The profile includes a placeholder profile picture, the username 'user1', and the name 'test'. To the right of the profile information, there are three status indicators: '2 followers', '1 following', and '6 posts'. A 'Following' button is located next to the 'following' indicator. Below the profile information, there are several small thumbnail images with captions, such as 'This is caption 1', 'This is caption 16', etc. At the bottom of the screen, a modal window titled 'Followers' is displayed, listing 'user5' and 'test' as followers. Each follower entry has a 'Follow' button to its right.

```
56 |     const data = await db.queryDbValues(`SELECT following FROM follow WHERE follower=$1;`, [id]);
```

**University of British Columbia, Vancouver**  
Department of Computer Science



Hashtag.js (REQUIRED NESTED AGGREGATION WITH GROUP BY QUERY and division)

**NESTED AGGREGATION WITH GROUP BY STATEMENT SCREENSHOTS**

```
15      const sql = `SELECT AVG(hashtag_count) as avg_hashtags_per_post
16          FROM (
17              SELECT postID, COUNT(hashTag) as hashtag_count
18                  FROM associateHashtag
19                      GROUP BY postID
20              ) as hashtag_counts`;
21      const data = await db.queryDb(sql);
```

Before:

Nested Aggregation with GROUP BY, avg number of hashtags per post Query

After:

# University of British Columbia, Vancouver

Department of Computer Science

Nested Aggregation with GROUP BY, avg number of hashtags per post Query  
1.3333333333333333

## DIVISION QUERY SCREENSHOTS

```
31 |           const sql = `SELECT postID  
32 |             FROM associateHashtag  
33 |               WHERE hashTag IN ($1, $2)  
34 |                 GROUP BY postID  
35 |                   HAVING COUNT(DISTINCT hashTag) = 2;`;  
36 |           const data = await db.queryDbValues(sql, values);
```

Before:

DIVISION, all posts that contain these hashtags  
Input One:Hashtag1  
Input Two:Hashtag2  
Query

After:

DIVISION, all posts that contain these hashtags  
Input One:Hashtag1  
Input Two:Hashtag2  
Query  
1

ImagePost.js

# University of British Columbia, Vancouver

## Department of Computer Science

```
8 |     const text = `INSERT INTO ImagePost(postID, content) VALUES($1, $2);`;
9 |     const values = [pid, file]
10|     const data = await db.queryDbValues(text, values);
```

```
19 |     const data = await db.queryDb("SELECT postID, content FROM ImagePost;")
```

### Likes.js

```
13 |     const data = await db.queryDbValues(`DELETE FROM likePost WHERE postID=$1 AND acc=$2;`, [post, user]);
```

```
25 |     const data = await db.queryDbValues(`INSERT INTO likePost(postID, acc) VALUES ($1, $2);`, [post, user]);
```

```
35 |     const data = await db.queryDbValues(`SELECT * FROM likePost WHERE postID=$1;`, [post]);
```

| - | user1

PostID: 1

This is caption 1

This is caption 1

10/10/2023, 12:00:00 AM

Likes: 1

Bookmarks: 0

Shares:

Comments:

| - | user1

PostID: 1

This is caption 1

This is caption 1

10/10/2023, 12:00:00 AM

Likes: 2

Bookmarks: 0

Shares:

Comments:

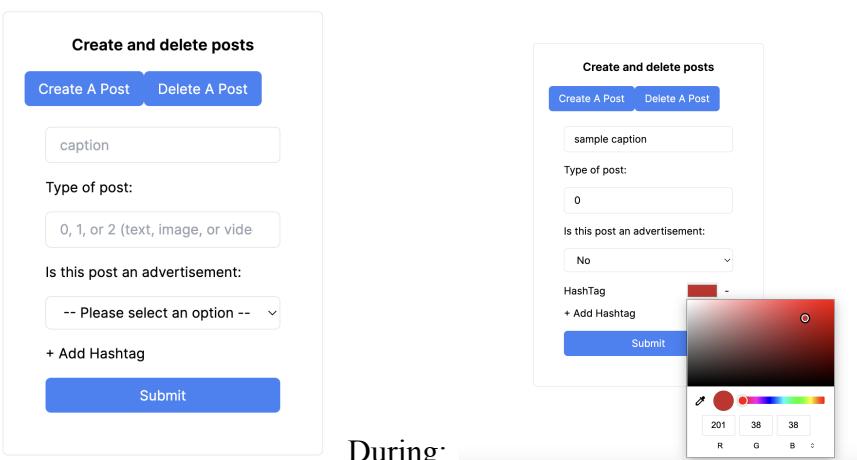
### Post.js (REQUIRED INSERT STATEMENT)

# University of British Columbia, Vancouver

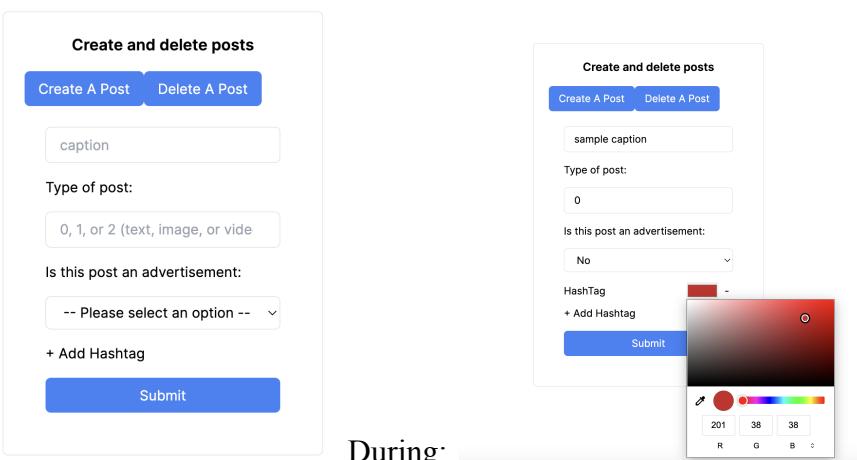
## Department of Computer Science

```
18 |         const pidData = await db.queryDb(`SELECT MAX(postID) AS max FROM Post;`);  
19 |         await hashtags.forEach(hashtag) => {  
30 |             db.queryDbValues(`SELECT * FROM Hashtags WHERE text=$1`, [hashtag.text]).then((res) => {  
  
36 |                 db.queryDbValues(  
37 |                     `INSERT INTO Hashtags (text, color) VALUES ($1, $2)`, [hashtag.text, hashtag.color.substring(1)]  
38 |                 );  
  
50 |             const text = `INSERT INTO post (postID, URL, caption, createdBy, timestamp, type) VALUES  
51 |                 ($1, $2, $3, $4, to_timestamp($5 / 1000.0), $6)`;  
52 |             const values = [pid, `${POST_URL}/${pid}`, caption, username, time, typeInt];  
53 |             const data = await db.queryDbValues(text, values);  
  
81 |             db.queryDbValues(  
82 |                 `INSERT INTO associateHashtag (postID, hashtag) VALUES ($1, $2)`, [pid, hashtag.text]  
83 |             )
```

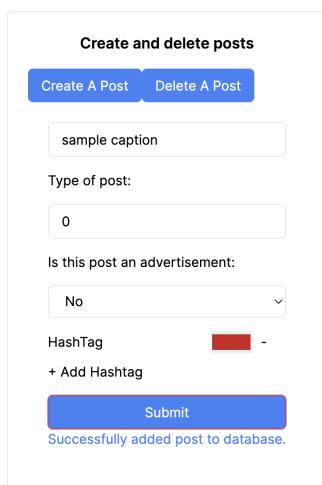
Before:



During:



After:



Delete a post:

# University of British Columbia, Vancouver

## Department of Computer Science

```
99      |         const dataUser = await db.queryDbValues(
100     |             `SELECT username FROM account WHERE username=$1;`, [username]
101    );
102
103
104
105
106      |         const checkCreator = await db.queryDbValues(
107     |             `SELECT createdBy FROM Post WHERE postID = $1;`, [target]
108    );
109
110
111
112
113
114
115
116
117      |         const data = await db.queryDbValues(`DELETE FROM Post WHERE postID = $1`, [target]);
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149      |         if (username) {
150     |             values.push(username)
151     |             query = `
152     |                 SELECT Post.*,
153     |                     CASE
154     |                         WHEN Post.type = 0 THEN TextPost.content
155     |                         WHEN Post.type = 1 THEN encode(ImagePost.content, 'base64')
156     |                         WHEN Post.type = 2 THEN encode(VideoPost.content, 'base64')
157     |                     END AS content
158     |                 FROM Post
159     |                 LEFT JOIN TextPost ON Post.postID = TextPost.postID AND Post.type = 0
160     |                 LEFT JOIN ImagePost ON Post.postID = ImagePost.postID AND Post.type = 1
161     |                 LEFT JOIN VideoPost ON Post.postID = VideoPost.postID AND Post.type = 2
162     |                 WHERE Post.createdBy=$1;
163     |             `;
164     |         } else if (postId) {
165     |             values.push(postId)
166     |             query = `
167     |                 SELECT Post.*,
168     |                     CASE
169     |                         WHEN Post.type = 0 THEN TextPost.content
170     |                         WHEN Post.type = 1 THEN encode(ImagePost.content, 'base64')
171     |                         WHEN Post.type = 2 THEN encode(VideoPost.content, 'base64')
172     |                     END AS content
173     |                 FROM Post
174     |                 LEFT JOIN TextPost ON Post.postID = TextPost.postID AND Post.type = 0
175     |                 LEFT JOIN ImagePost ON Post.postID = ImagePost.postID AND Post.type = 1
176     |                 LEFT JOIN VideoPost ON Post.postID = VideoPost.postID AND Post.type = 2
177     |                 WHERE Post.postID=$1;
178     |             `;
179     |         } else {
180     |             query = "SELECT * FROM Post;" allanx01, 20 hours ago • home page feed d
181     |         }
182
183         data = await db.queryDbValues(query, values);
```

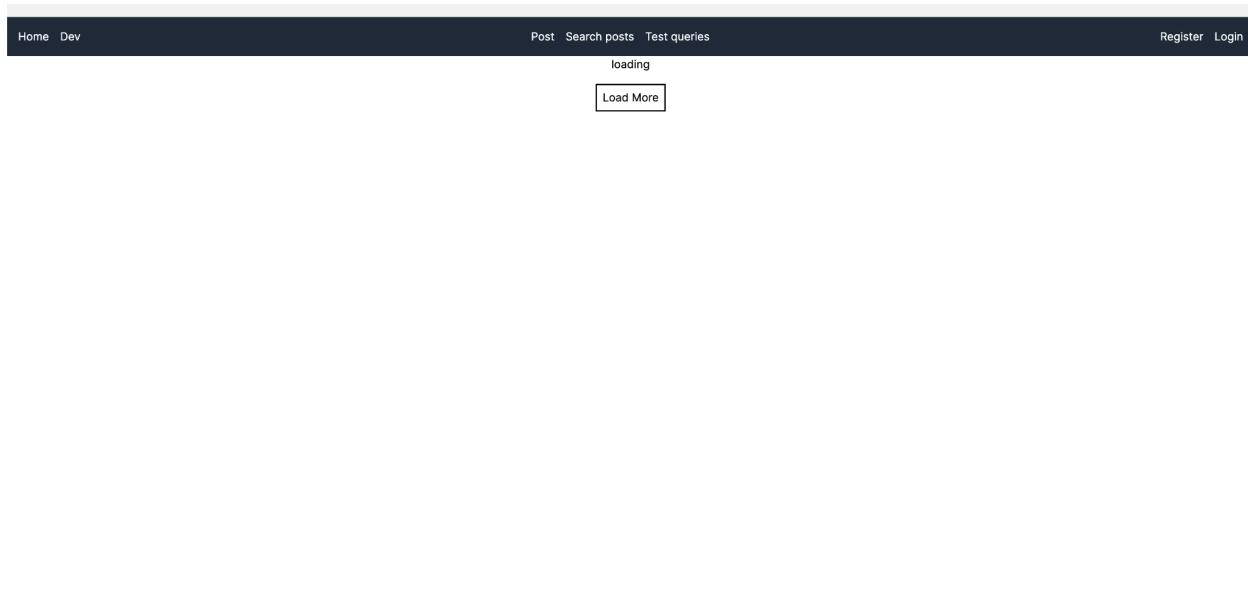
# University of British Columbia, Vancouver

Department of Computer Science

(REQUIRED JOIN STATEMENT)

```
200      const sql = `SELECT Post.*,
201          CASE
202              WHEN Post.type = 0 THEN TextPost.content
203              WHEN Post.type = 1 THEN encode(ImagePost.content, 'base64')
204              WHEN Post.type = 2 THEN encode(VideoPost.content, 'base64')
205          END AS content
206      FROM
207          Post
208          LEFT JOIN TextPost ON Post.postID = TextPost.postID AND Post.type = 0
209          LEFT JOIN ImagePost ON Post.postID = ImagePost.postID AND Post.type = 1
210          LEFT JOIN VideoPost ON Post.postID = VideoPost.postID AND Post.type = 2
211      INNER JOIN
212          follow ON Post.createdBy = follow.following
213      WHERE
214          follow.follower = $1
215      ORDER BY
216          Post.timestamp DESC
217      LIMIT $2
218      OFFSET $3;`;
219      const data = await db.queryDbValues(sql, [user, limit, offset]);
```

BEFORE



AFTER

# University of British Columbia, Vancouver

## Department of Computer Science

The screenshot shows a web interface with a header bar containing 'Home', 'Chat', 'Dev', 'Post', 'Search posts', 'Test queries', 'Logout', and 'Profile'. Below the header are three search results for 'user1'. Each result card includes the user's name, post ID, caption, creation date, and interaction metrics (Likes, Bookmarks, Shares, Comments). A note 'CURRENTLY NOT SUPPORTED' is present under each caption.

PostID	Caption	Creation Date	Likes	Bookmarks	Shares	Comments
1	This is caption 1	10/10/2023, 12:00:00 AM	2	0	0	0
6	This is caption 1	10/10/2023, 12:00:00 AM	0	0	0	0
11	This is caption 1	10/10/2023, 12:00:00 AM	0	0	0	0

### Search.js (REQUIRED SELECTION STATEMENT)

```
10  const { postId, caption, createdBy, type, postTime } = req.query;
11  var query = `SELECT * FROM Post WHERE `;
12  var values = [];
13  var i = 1;
14  if (postId !== undefined) {
15      const pid = parseInt(postId);
16      query += `postID=$${i} AND `;
17      values.push(pid);
18      i++;
19  }
20  if (caption !== undefined) {
21      query += `caption=$${i} AND `;
22      values.push(caption);
23      i++;
24  }
25  if (type !== undefined) {
26      const typeInt = parseInt(type);
27      query += `type=$${i} AND `;
28      values.push(typeInt);
29      i++;
30  }
31  if (createdBy !== undefined) {
32      query += `createdBy=$${i} AND `;    You, 34 minutes ago * Uncommitted changes
33      values.push(createdBy);
34      i++;
35  }
36  if (postTime !== undefined) {
37      query += `timestamp=$${i} AND `;
38      values.push(postTime);
39      i++;
40  }
41
42  let finalQuery = query.substring(0, query.length - 5) + ";";
const data = await db.queryDbValues(finalQuery, values);
```

# University of British Columbia, Vancouver

## Department of Computer Science

**Search**

Search for posts:

Post ID:

Post caption:

Post creator:

Post type (0 = Text, 1 = Image, or 2 = Video)

Time created (Format: YYYY-MM-DD HH:MM:SS:ms)

**Begin Search**

**Search**

Search for posts:

Post ID:

Post caption:

Post creator:

Post type (0 = Text, 1 = Image, or 2 = Video)

Time created (Format: YYYY-MM-DD HH:MM:SS:ms)

**Begin Search**

Before:

**Search**

This is caption 1

Post creator:

Post type (0 = Text, 1 = Image, or 2 = Video)

Time created (Format: YYYY-MM-DD HH:MM:SS:ms)

**Begin Search**

Success! Please view the data:

**user1**

PostID: 1  
This is caption 1  
10/10/2023, 12:00:00 AM  
 Bookmarks: 0 Shares: Comments:

During:

**Search**

**user1**

PostID: 1  
This is caption 1  
10/10/2023, 12:00:00 AM  
 Bookmarks: 0 Shares: Comments:

**user1**

PostID: 6  
This is caption 1  
CURRENTLY NOT SUPPORTED  
10/10/2023, 12:00:00 AM  
 Bookmarks: 0 Shares: Comments:

**user1**

PostID: 11  
This is caption 1

After:

### SELECT STATEMENT FOR SEARCH: SCREENSHOTS

# University of British Columbia, Vancouver

## Department of Computer Science

```
95      query = `
96          SELECT Post.*,
97              CASE
98                  WHEN Post.type = 0 THEN TextPost.content
99                  WHEN Post.type = 1 THEN encode(ImagePost.content, 'base64')
100                 WHEN Post.type = 2 THEN encode(VideoPost.content, 'base64')
101             END AS content
102         FROM Post
103         LEFT JOIN TextPost ON Post.postID = TextPost.postID AND Post.type = 0
104         LEFT JOIN ImagePost ON Post.postID = ImagePost.postID AND Post.type = 1
105         LEFT JOIN VideoPost ON Post.postID = VideoPost.postID AND Post.type = 2
106     WHERE Post.createdBy=$1;
107`;
```

```
110      query = `
111          SELECT Post.*,
112              CASE
113                  WHEN Post.type = 0 THEN TextPost.content
114                  WHEN Post.type = 1 THEN encode(ImagePost.content, 'base64')
115                  WHEN Post.type = 2 THEN encode(VideoPost.content, 'base64')
116             END AS content
117         FROM Post
118         LEFT JOIN TextPost ON Post.postID = TextPost.postID AND Post.type = 0
119         LEFT JOIN ImagePost ON Post.postID = ImagePost.postID AND Post.type = 1
120         LEFT JOIN VideoPost ON Post.postID = VideoPost.postID AND Post.type = 2
121     WHERE Post.postID=$1;
122`;
```

124

query = "SELECT \* FROM Post;"

```
144      const sql = `SELECT Post.*,
145          CASE
146              WHEN Post.type = 0 THEN TextPost.content
147              WHEN Post.type = 1 THEN encode(ImagePost.content, 'base64')
148              WHEN Post.type = 2 THEN encode(VideoPost.content, 'base64')
149          END AS content
150      FROM
151          Post
152          LEFT JOIN TextPost ON Post.postID = TextPost.postID AND Post.type = 0
153          LEFT JOIN ImagePost ON Post.postID = ImagePost.postID AND Post.type = 1
154          LEFT JOIN VideoPost ON Post.postID = VideoPost.postID AND Post.type = 2
155      INNER JOIN
156          follow ON Post.createdBy = follow.following
157      WHERE
158          follow.follower = $1
159      ORDER BY
160          Post.timestamp DESC
161      LIMIT $2
162      OFFSET $3`;
163      const data = await db.queryDbValues(sql, [user, limit, offset]);
```

TextPost.js

# University of British Columbia, Vancouver

Department of Computer Science

```
 9      const text = `INSERT INTO textpost(postID, content) VALUES($1, $2)`;  
10     const values = [pid, caption]  
11     const data = await db.queryDbValues(text, values);  
20     const data = await db.queryDbValues("SELECT postID, content FROM textpost;", [])
```

VideoPost.js

```
 7      try {  
● 8        const text = `INSERT INTO VideoPost(postID, content) VALUES($1, $2)`;  
 9        const values = [pid, file]  
10       const data = await db.queryDbValues(text, values);  
11  
12      } catch (err) {  
● 31        const data = await db.queryDb("SELECT postID, caption FROM VideoPost;")  
13    }
```

THIS FILE IS /backend/src/middleware/PassportConfig.js

```
 9      const query = `SELECT * FROM login WHERE username=$1`;  
10  
11      try {  
12        db.queryDbValues(query, [username]).then((data) => {  
13          if (data.length > 0) {  
14            const user = data[0];  
15            req.user = user;  
16            return res.status(200).json(user);  
17          } else {  
18            res.status(401).json({ message: "User not found" });  
19          }  
20        }  
21      } catch (err) {  
22        console.error(err);  
23        res.status(500).json({ message: "Internal server error" });  
24      }  
25    }  
26  
27    passport.use(new LocalStrategy((username, password, done) => {  
28      db.queryDbValues(`SELECT * FROM login WHERE username=$1`, [username]).then((data) => {  
29        if (data.length > 0) {  
30          const user = data[0];  
31          bcrypt.compare(password, user.password, (err, isMatch) => {  
32            if (err) {  
33              done(err);  
34            } else if (isMatch) {  
35              done(null, user);  
36            } else {  
37              done(null, false);  
38            }  
39          });  
40        } else {  
41          done(null, false);  
42        }  
43      });  
44    }  
45    ,  
46    const query = `SELECT * FROM login WHERE username=$1`;  
47    const data = await db.queryDbValues(query, [id]);  
48    if (data.length > 0) {  
49      const user = data[0];  
50      req.user = user;  
51      return res.status(200).json(user);  
52    } else {  
53      res.status(401).json({ message: "User not found" });  
54    }  
55  })  
56
```