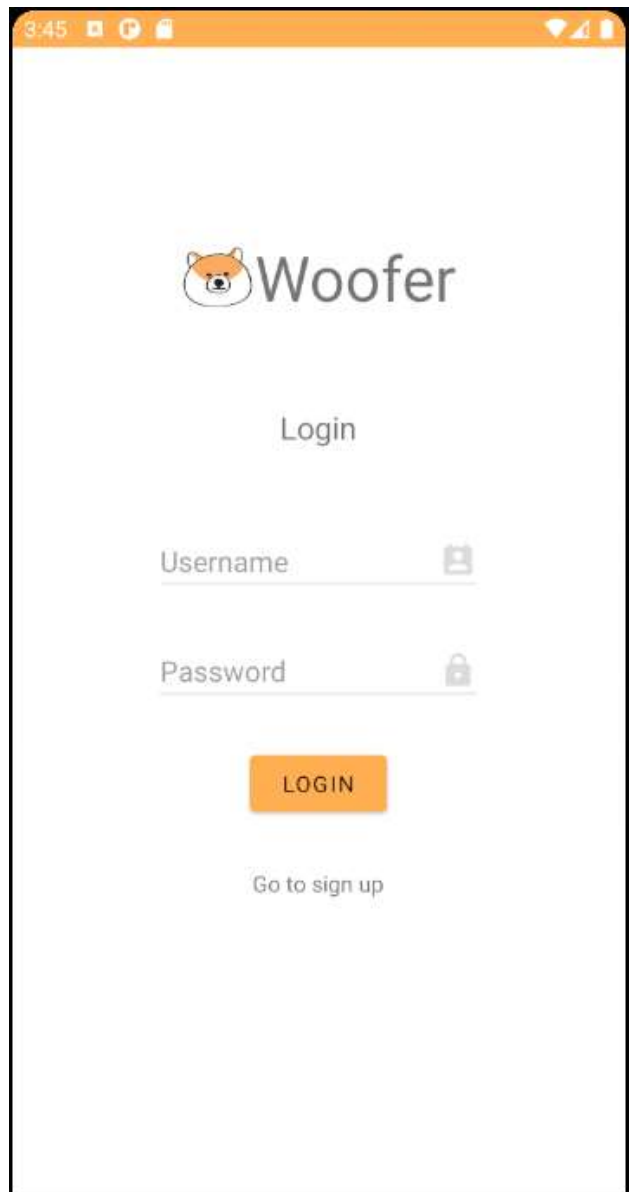


Project Document

Woofer

Jia-Xin Gao 1601812

Matthew Loo 1843138



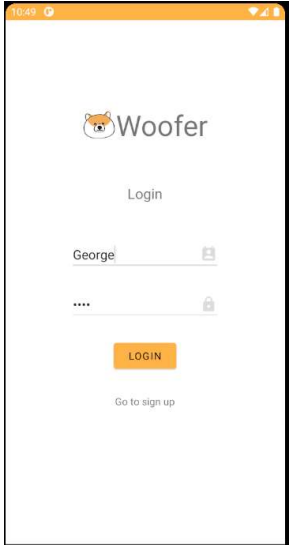
Contents

Woofers	0
Contents	1
App Description	2
DBF Details	5
Procedures/Processes involved	5
Business rules	10
Initial ERD	11
Issues	12
Final ERD	13
Implementation	14
Queries to create tables:	14
Login queries	14
Sign up queries	14
Status queries	15
Add Status queries	15
Friendlist queries	15
Add friend queries	15
Remove friend queries	15
Friend of friend queries	15

App Description

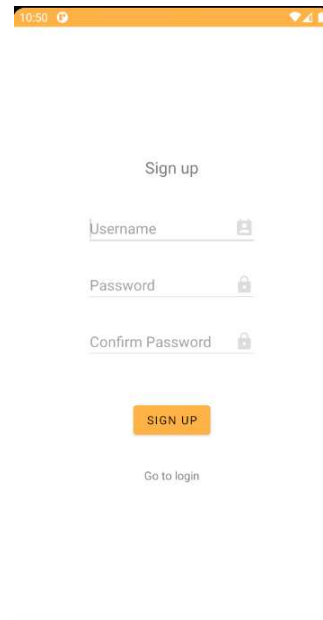
The project topic is called 'Woofers' which is a simple text based social app developed in Android Studio for Android devices. Woofers has the core functionality of being a platform for users to post 'status updates' which are text based messages. The users on the app will be able to view all status updates posted by their friends and themselves as well as add their own status updates to the network for viewing.

The app has 5 user interfaces, each which react with the database to perform their respective tasks, these interfaces are:

<p>1. Login</p> <ul style="list-style-type: none">○ Receive user login credentials and authenticate the user credentials against the user credentials stored on the database	
--	--

2. Sign-up

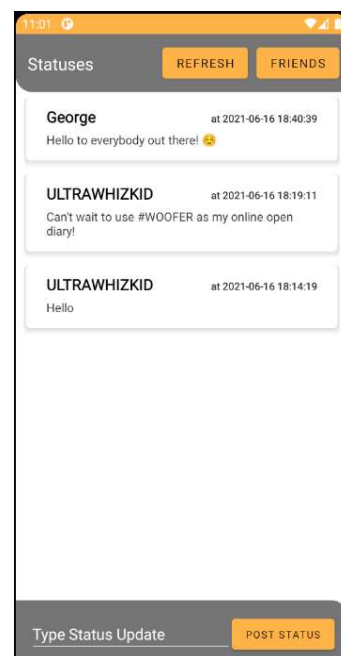
- Receive signup credentials from the user and check against the database for redundancy, if not redundant, will allow the user to add their login details to the database.



A mobile app interface for signing up. At the top, there's a status bar with the time 10:30 and various icons. Below it, the title "Sign up" is centered. There are three input fields: "Username" with a person icon, "Password" with a lock icon, and "Confirm Password" with a lock icon. Below these fields is an orange button labeled "SIGN UP". At the bottom, there's a link that says "Go to login".

3. Statuses

- Displays all statuses from the user and their friends ordered by the latest status posts, retrieves the status posts from the database using the User's added friends



A mobile app interface for viewing statuses. At the top, there's a status bar with the time 13:01 and various icons. Below it, the title "Statuses" is centered, with two orange buttons labeled "REFRESH" and "FRIENDS" to its right. The main content area displays three status posts. Each post shows the user's name, the time it was posted, and the text of the status. The first post is from "George" at 2021-06-16 18:40:39 with the text "Hello to everybody out there! 🐶". The second post is from "ULTRAWHIZKID" at 2021-06-16 18:19:11 with the text "Can't wait to use #WOOFER as my online open diary!". The third post is from "ULTRAWHIZKID" at 2021-06-16 18:14:19 with the text "Hello". At the bottom, there's a text input field labeled "Type Status Update" and an orange button labeled "POST STATUS".

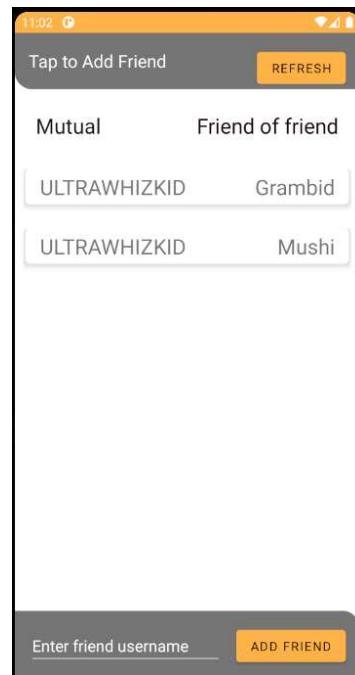
4. Friendlist

- Displays the friends of the user by retrieving the list from the database and allows them to delete existing friends.



5. Add Friend

- Displays a list of friends of friends in table form showing mutual friends. Also allows the user to add the friend of their mutual friends as well as add them by typing in a username.



Details about database processes explained in below sections.

DBF Details

Procedures/Processes involved

The first step is to create a database with the required tables implemented. Below are the tables that are required for the app to work with basic functionality.

Tables:

USERS
FRIENDS
STATUS

Once the database and the tables are created, we can then create a PHP API that handles all GET and POST requests from the app.

Woofer in its current state did not require the use of SQL Procedures, however the data retrieved from the table is processed in PHP, occasionally for use in further SQL queries.

Database queries and processes are each tied to their respective PHP files and UI screens.

1. Login

- The database receives a query from the PHP file in the form:

```
SELECT * from USERS where USERNAME=?
```

Where the '?' is the username entered by the user in the app.

The database is queried and returns the Username and hashed password of the user in the database, if no user is found it returns an empty set. The response is then received and authenticated by the client machine.

2. Sign Up

- The database receives a query from the PHP file which is exactly the same as the login SELECT query, for the purpose of checking whether

the desired new username exists already in the database, if not, returns an empty set.

Consequently to the empty set being sent it will then receive a query from the PHP file in the form:

```
INSERT INTO USERS (USERNAME,HASHEDPASS)
VALUES (?,?)
```

Where the first '?' is the username received from the client's machine and the second '?' is the hashedPassword processed from the client's machine.

3. Statuses

- Retrieving statuses
 - i. In order to retrieve a status for display on the app, the PHP file is sent a request to query the database for all the user's and their friends statuses in 2 steps:

1. Query the database to retrieve a list of all the user's friends:

```
SELECT FRIEND FROM FRIENDS WHERE USERNAME = ?;
```

2. Use the results from the previous query to build a new SQL statement:

```
$statusList = "SELECT *
FROM STATUS
WHERE USERNAME IN (";
foreach ($names as $name) {
    $statusList=$statusList."'" . $name . "'".',';
}
$statusList = $statusList."'"$user_id') ORDER BY
DATE DESC;"
```

Which returns all statuses which authors are either the user or the friends, in descending chronological order.

- Posting new status
 - i. Posting the status is a simple insert statement which requires the app to send the PHP file the user's username and the

content of their status post, makes use of the CURRENT_TIMESTAMP function to accurately post the status.

```
INSERT INTO STATUS (USERNAME, DATE, CONTENT)
VALUES (?, CURRENT_TIMESTAMP, ?);
```

4. Friendlist

- Retrieving Friendlist

- i. App uses the PHP file to retrieve and return a list of all the user's friends using the statement:

```
SELECT FRIEND FROM FRIENDS WHERE USERNAME = ?;
```

- Removing a Friend

- i. On the Friendlist page, the user has the option to remove a friend from their friend list, whenever a friend is added, the database will make 2 insertions, one for the current user adding their friend, and another for the friend adding the current user as their friend; Thus a delete statement that deletes both entries is needed:

```
DELETE FROM FRIENDS where (USERNAME = ? AND FRIEND = ?)
OR (FRIEND = ? AND USERNAME=?)
```

5. Add Friend

- The add friend UI screen allows the functionality for the user to add friends manually by typing their username, or by tapping on a list of their friend's friends.

- i. Manual Add

- 1. The Manual Add uses a statement which requires the user's USERNAME which the program already has, and the friend's USERNAME which the user enters into a text field; both are sent to the PHP API. However there are 3 steps to ensure no problems are encountered before data entry
 - a. Check the friend exists in the table , a query is sent by the API to check if there exists a record of the friend in the table:


```
SELECT * FROM USERS WHERE USERNAME ='$friend'
```

- b. If the record is not empty, the API will then check if the friend is not already part of the current User's friendlist using the same SQL query from the friendlist screen.
- c. If the friendlist query returns an empty set, then a query is run to finally add the friend to the table:

```
INSERT INTO FRIENDS VALUES (?, ?), (?, ?);
```

Where the first and last '?' is the current User's USERNAME, and the middle '?' is the friend's USERNAME

- ii. Adding through a Mutual friend. To add a user through a mutual friend the Add Friends UI screen must be able to display a list of the set of Friends of the User's friends, and then be able to add any of the mutual friend's friends using the previous method, thus this can be broken down into 2 steps:

1. Displaying Friends of Friends on the App's screen. Will need 2 things: the User's friends, and the Friend's friends. The PHP API queries the database to find all the user's friends using the previous friendlist query, which returns an array of the User's friends. The array is used to build a new SQL query which returns all friends of friends:

```
$fofsql = "SELECT *  
FROM FRIENDS  
WHERE USERNAME IN ("  
foreach ($names as $name) {  
    $fofsql=$fofsql."'" . $name . "'" . ',' . ' ';  
}
```

But then the SQL query is processed further to ensure that only users who are not already the logged-in User's friend are returned :

```
$fofsql = $fofsql."AND FRIEND NOT  
IN('". $user_id . "',";  
foreach ($names as $name) {
```

Mutual	Friend of friend
ULTRAHIZKID	Grambid
ULTRAHIZKID	Mushi

```
$fofsql=$fofsql."'"'.$name.'"'.',';
}
```

The last step in returning the list for friends of friends, is to group all friends of friends together so that it shows all mutual friends on the left, using :

```
ORDER BY USERNAME ASC;
```

Thus finally returning a set of with Mutual friends, and their friends.

2. A user can then tap on any of the Friend of Friend components to add the friend, which will then invoke the previous Manual-add SQL and feed the 'friend of friend's name as a parameter.

Business rules

There are three entities (tables) in the Woofer database, they are:

1. USERS, which stores all the users information.
2. FRIENDS, which stores all the friends of each user.
3. STATUS, which store all the statuses of each user.

The relationships between these entities are:

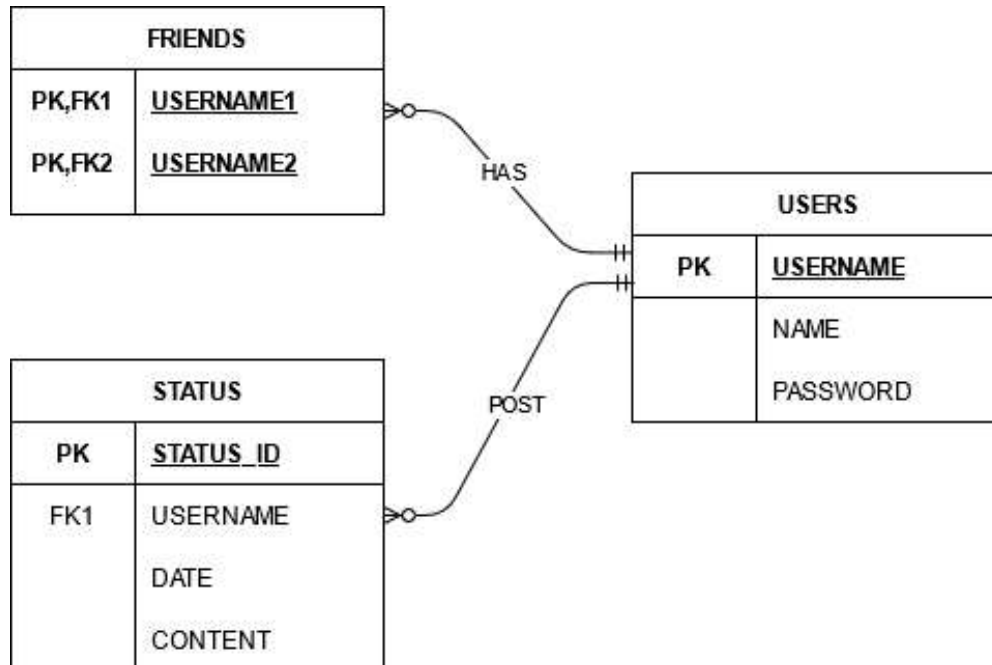
- USERS (1:M): FRIENDS (1:1) \Rightarrow 1:M (One to Many)
A user can have multiple friends, but each friend can only be matched with one user.
A friend must always be matched with one user, but a user can have multiple friends.
- USERS(1:M): STATUS(1:1) \Rightarrow 1:M (One to Many)
A user can have multiple statuses, but each status can only be matched with one user.
A status must always be matched with one user, but a user can have multiple statuses.

The constraints between the entities are:

- USERS (1, 1) \Leftrightarrow (0, N) FRIENDS
A user can have zero or up to N friends (N is a positive integer), but each friend instance must be matched to one user.
- USERS (1, 1) \Leftrightarrow (0, N) STATUS
A user can have zero or up to N statuses (N is a positive integer), but each status instance must be matched to one user.

Initial ERD

Crow foot notation diagram for Woofer database:



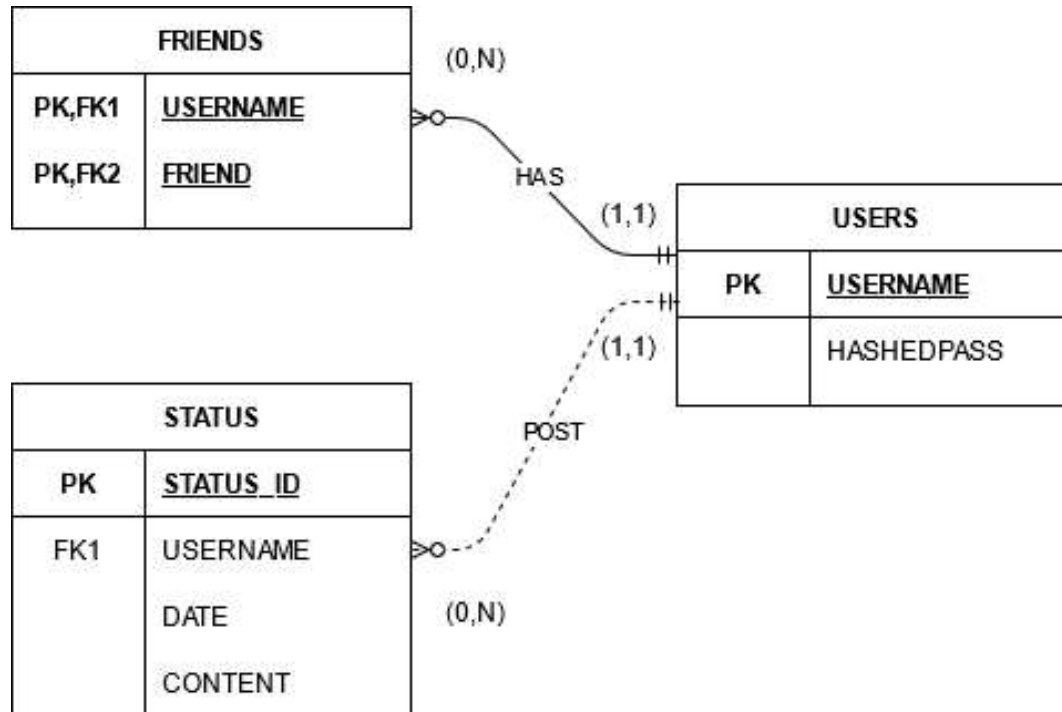
Issues

Some issues encountered when trying to implement the database:

1. Users can add multiple users as friends, those users can also add other users as friends, this is a many to many relationship. To solve this we use a bridge entity called friends that creates a 1:M relationship. Each user can have multiple friends but a friend can only be matched to one user.
2. Null values should be handled correctly on the front-end. But when creating the tables, the columns that cannot be NULL should be created to not accept NULL values. An example is USERNAME and HASHPASSWORD in the USERS table.
3. Ideally users should be able to accept and decline friend requests, however we did not have the time to implement a friend request feature. So instead each time a user adds a new friend, the friend also adds the user to their friends list automatically.
4. Initially names of users were going to be stored in the USERS table as NAME, but the names were never used so the NAME column was dropped from the USERS table.
5. Unfortunately users will not be able to change their usernames due to the use of USERNAME being used as the primary key in some tables, but this is also due to a design choice similar to how a username cannot be changed on many platforms such as 'Steam'

Final ERD

Crow foot notation diagram for Woofer database:



Implementation

Create tables queries:

```
CREATE TABLE USERS (
    USERNAME VARCHAR(40) NOT NULL,
    HASHEDPASS CHAR(64) NOT NULL,
    PRIMARY KEY (USERNAME)
);

CREATE TABLE STATUS (
    STATUS_ID INT(10) NOT NULL AUTO_INCREMENT,
    USERNAME VARCHAR(40) NOT NULL,
    DATE DATETIME NOT NULL,
    CONTENT TEXT NOT NULL,
    PRIMARY KEY (STATUS_ID),
    FOREIGN KEY (USERNAME) REFERENCES USERS (USERNAME)
);

CREATE TABLE FRIENDS (
    USERNAME VARCHAR(40) NOT NULL,
    FRIEND VARCHAR(40) NOT NULL,
    PRIMARY KEY (USERNAME, FRIEND),
    FOREIGN KEY (USERNAME) REFERENCES USERS (USERNAME),
    FOREIGN KEY (FRIEND) REFERENCES USERS (USERNAME)
);
```

Login queries

```
SELECT * from USERS where USERNAME=USER_USERNAME
```

Sign up queries

```
INSERT INTO USERS (USERNAME, HASHEDPASS)
VALUES (USER_USERNAME, HASHED_PASSWORD);
```

Status queries

```
SELECT *
FROM STATUS
WHERE USERNAME IN (FRIEND_USERNAME, ?, ?, ..., ?, ?, USER'S USERNAME) ORDER BY
DATE DESC;"
```

Add Status queries

```
INSERT INTO STATUS (USERNAME, DATE, CONTENT)
VALUES (USER_USERNAME, CURRENT_TIMESTAMP, STATUS_CONTENT) ;
```

Friendlist queries

```
SELECT FRIEND FROM FRIENDS WHERE USERNAME = FRIEND_USERNAME;
```

Add friend queries

```
INSERT INTO FRIENDS VALUES
(USER_USERNAME, FRIEND_USERNAME) , (FRIEND_USERNAME, USER_USERNAME) ;
```

Remove friend queries

```
DELETE FROM FRIENDS where (USERNAME = ? AND FRIEND = ?) OR (FRIEND = ? AND
USERNAME=?)
```

Friend of friend queries

```
SELECT *
FROM FRIENDS
WHERE USERNAME IN (FRIEND_USERNAME, ?, ?, ..., ?, ?)
AND FRIEND NOT IN ( (FRIEND_USERNAME, ?, ?, ..., ?, ?, USER'S USERNAME)
ORDER BY USERNAME ASC;
```