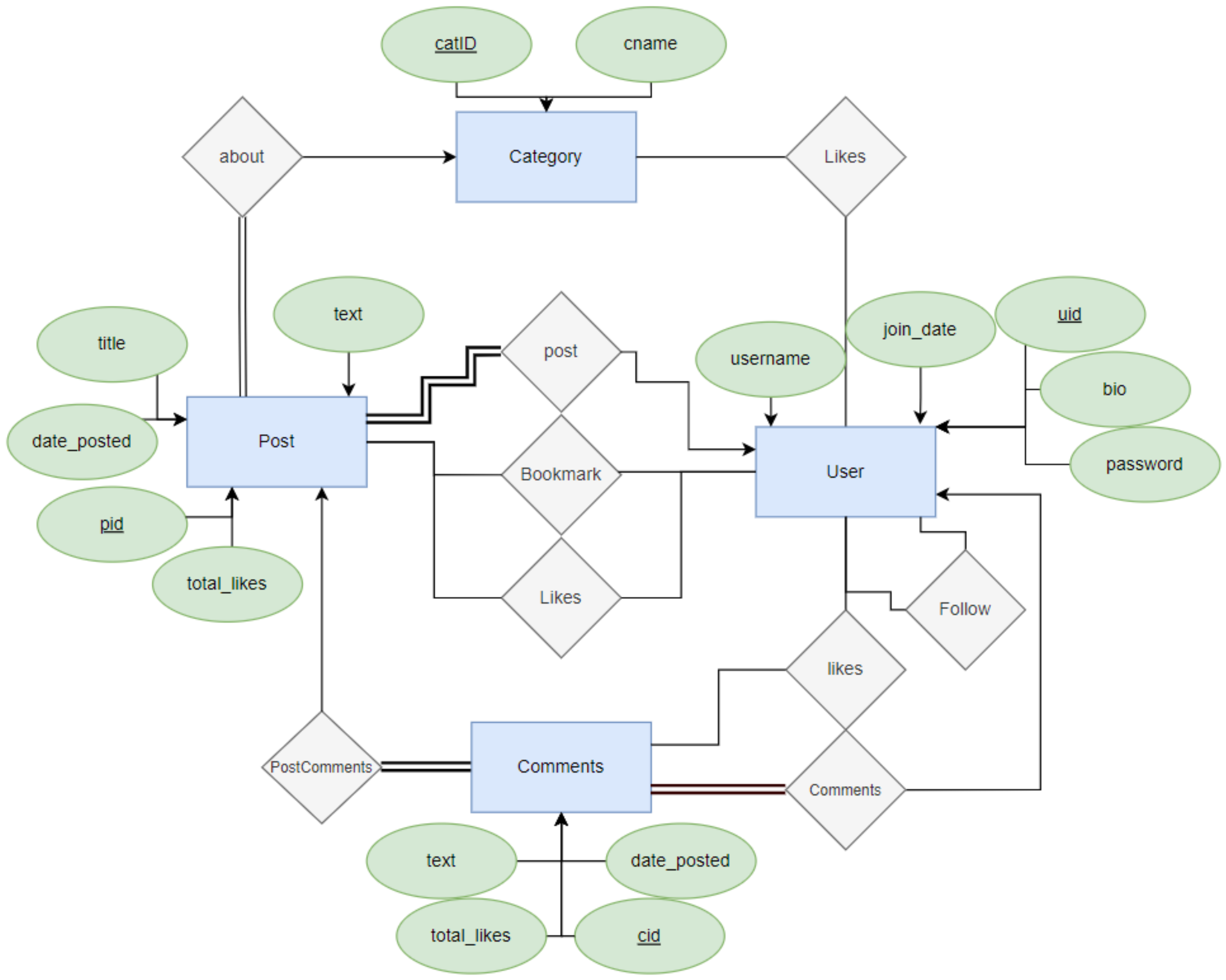
Tech For Dummies - Final Deliverable

By: Dakota Bourne (db2nb) and Matthew Reid (mrr7rn)

Github link: https://github.com/dakotabourne373/techfordummies

1) Database Design



Tables
User(uid, username, password, join_date, bio)
Post(pid, uid, date_posted, catID, title, text, total_likes)
Comments(cid, pid, cdate_posted, text, total_likes)
Category(catID, cname)
UsersComments(uid, cid)
CatLikes(uid, catID)

UserLikes(uid, pid)
CommentLikes(uid, cid)
Bookmarks(uid, pid)
FollowedUsers(uid, uid)
User_Audit(log_date, who_update, join_date, uid, username, total_posts, old_bio, new_bio)

2) Database Programming

a)

Our database is hosted locally on XAMPP.

b)

The app is also hosted locally on XAMPP.

c)

Github link: https://github.com/dakotabourne373/techfordummies

1. Clone the github repo to a folder within C:/xampp/htdocs/
2. Change the variable within index.php in the root of the project, instructions are in the file, and I will include them below as well.
3. Turn on Apache and MySQL in the XAMPP control panel
    a. import the SQL file in the root of the project in your localhost/phpmyadmin
4. This will automatically create a database and include the base data for the project
5. Navigate to the base url in your browser, and the app should be working!

Instructions for updating url variable:
Change this variable to match how the url would look on your localhost's browser

e.g. on my xampp I would access it as localhost/techfordummies/,
so i set the variable to "/techfordummies/"

if your system has sub folders you need to include them

e.g. if you access this project with localhost/cs4750/project/blue/green/,
you would set the variable equal to "/cs4750/project/blue/green/"

NOTE: At the top of the sql file there is a create database and use database command, so you don't have to create your own. If it's easier in your workflow, just delete those lines or comment them out, I thought I would try to be nicer.

d)

For our project we use procedures and triggers to interact with the likes and dislikes for each post/comment. For example, if a user likes a post, we have a procedure that ticks up the likes for that post, and the trigger on insert of the user liking the post. This goes the same for comment's likes.

3) Database Level Security

We have a security feature for the developers to monitor the users of the app in the form of a user editing audit log. This was done to keep track of user inputs into their profiles and have an after incident record of user interactions.

```
CREATE TABLE `User_Audit` (
    log_date date NOT NULL,
    who_update varchar(30) NOT NULL,
    join_date date DEFAULT NULL,
    uid int(11) DEFAULT NULL,
    username varchar(15) NOT NULL,
    total_posts int NOT NULL default 0,
    old_bio text NOT NULL,
    new_bio text NOT NULL
);
```

```
DELIMITER $$
CREATE TRIGGER user_auditTrail BEFORE UPDATE ON User
FOR EACH ROW
BEGIN
    INSERT INTO user_audit
    VALUES (CURRENT_DATE, CURRENT_USER, old.join_date, old.uid,
    old.username, old.total_posts, old.bio, new.bio);
END$$
DELIMITER ;
```

We could also add a user end database security, by revoking user access to change the categories role, as they are not permitted to, while implementing an admin or moderator role. This role would be able to add, delete, or update categories. This is out of reach currently. This would be a version of Role-Based Access Control.

4) Application Level Security

We have multiple instances of Application level of security. I will list a few to be concise:

1. Password hashing

```
$hash = password_hash($_POST["password"], PASSWORD_DEFAULT);
password_verify($_POST["password"], $data[0]["password"])
```

2. Checking if the user is signed in for authenticated sections of the app

```
if (!isset($_SESSION["username"])) {
            header("Location: {$this->url}index/");
            exit();
        }
if (!isset($_SESSION["userID"])) {
            $data["error"] = true;
            header("Content-Type: application/json");
            echo json_encode($data, JSON_PRETTY_PRINT);
            return;
        }
```

3. Each SQL Query is also done through prepare and bind_param, stopping injection from occurring

```
public function query($query, $bparam = null, ...$params) {
        $stmt = $this->mysqli->prepare($query);

        if ($bparam != null)
            $stmt->bind_param($bparam, ...$params);

        if (!$stmt->execute()) {
            // execute failed
            return false;
        }

        // execute succeeded
        if (($res = $stmt->get_result()) !== false)
            return $res->fetch_all(MYSQLI_ASSOC);

        return true;
```

```
    }
```

4. We also have client side security measures validating form inputs

```
function passwordCheck(num) {
    let pw = $("#password");
    let submit = $("#submit");
    let pwhelp = $("#pwhelp");

    if (pw.val().length < num) {
        pwhelp.text("Password must be " + num + " characters or longer");
        pw.addClass("is-invalid");
        submit.prop("disabled", true);
    } else {
        pwhelp.text("");
        pw.removeClass("is-invalid");
        submit.prop("disabled", false);
    }
}

$("#password").keyup(function () {
    passwordCheck(8);
})
```