

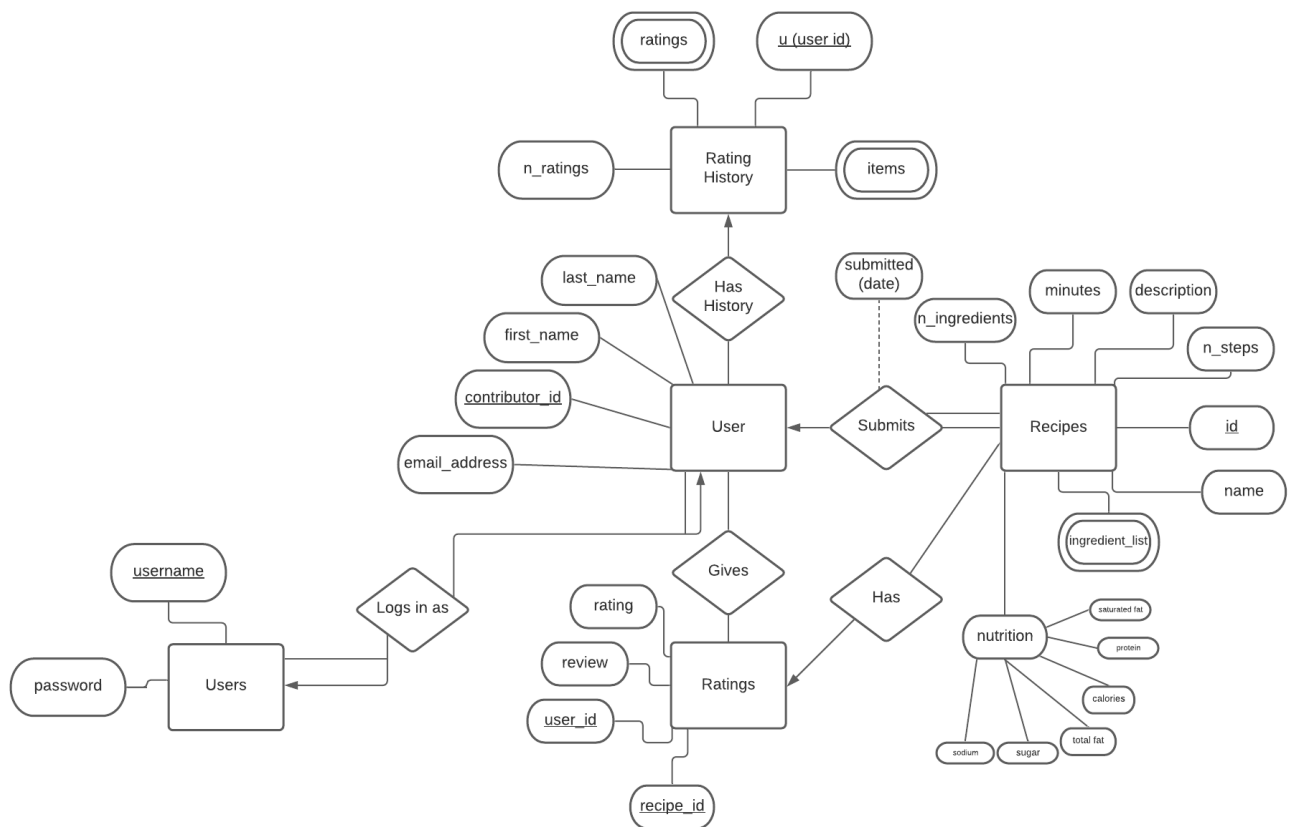
Team Members: Ayushi Ambhore (ara7ke), Hannah Douglas (hcd6tc),  
Tijana Djokic (td4jj), Grace Kisly (gck4mwf)

Final Deliverable

<https://recipe-finder-2.herokuapp.com/registration.php>

## Part 1 - Database Design

E-R diagram:



Tables written in schema statements:

Entity sets:

**RatingHistory(u, n\_ratings)**

**RatingHistory\_ratings(u, ratings)**

**RatingHistory\_items(u, items)**

**User(contributor\_id, first\_name, last\_name, email\_address)**

**Ratings(recipe\_id, user\_id, rating, review)**

**Recipes(id, contributor\_id, submitted, name, n\_ingredients, minutes, description, n\_steps, calories, total\_fat, sugar, sodium, protein, saturated fat)**

**Recipes\_ingredient\_list(recipe\_id, ingredient\_list)**

**Recipes\_ingredient\_list(recipe\_id, ingredient\_list)**

**Users(username, password, contributor\_id)**

Relationships:

**HasHistory(u, contributor\_id)**

**Gives(contributor\_id, user\_id, recipe\_id)**

**Has(id, user\_id, recipe\_id)**-- recipe that is rated, user\_id who rated recipe and their u

Our unique attributes are: u, username, user\_id, recipe\_id, contributor\_id, id. Our not null attributes are: u, username, password contributor\_id, recipe\_id, user\_id, id, n\_ratings, n\_ingredients, ingredients\_list, name, rating. Our primary keys are underlined above in the E-R diagram.

## Part 2 - Database Programming

We host our database on PHPMyAdmin, and the app we created to run it is hosted on Heroku. We began by creating a git repository where we would store all of our files needed to run the website. We then created a project on Heroku and pushed our app to Heroku using the terminal commands `heroku git:remote -a recipe-finder-2` and `git push heroku main`. After we deployed our app, we needed to connect it to a MySQL database. We added the ClearDB MySQL add-on to our Heroku project and configured PHPMyAdmin to connect to our Heroku database by editing our config.inc.php file to include these lines:

```
/* Heroku remote server */
$i++;
$config["Servers"][$i]["host"] = "us-cdbr-east-04.cleardb.com"; //provide hostname
$config["Servers"][$i]["user"] = "bf4e3ebdca9a1e"; //user name for your remote server
$config["Servers"][$i]["password"] = "cd386ac5"; //password
$config["Servers"][$i]["auth_type"] = "config"; // keep it as config
```

and then on PHPMyAdmin we were able to connect to this server and database. Lastly, we imported our .sql file to PHPMyAdmin, which included all of our code to create tables, set

constraints/privileges, etc., and connected our database to our app in our index.php file like so:

```
Recipe-finder > index.php
1  <?php
2  //Get Heroku ClearDB connection information
3  $cleardb_url = parse_url(getenv("CLEARDB_DATABASE_URL"));
4  $SERVER = $cleardb_url["host"];
5  $USERNAME = $cleardb_url["user"];
6  $PASSWORD = $cleardb_url["pass"];
7  $DATABASE = substr($cleardb_url["path"],1);
8  $active_group = 'default';
9  $query_builder = TRUE;
10 // Connect to DB
11 $con = mysqli_connect($SERVER, $USERNAME, $PASSWORD, $DATABASE);
12 ?>
```























After this step our database was connected to our app! You can use this link to view the website: <https://recipe-finder-2.herokuapp.com/registration.php>. The steps we followed to produce this website can be found at <https://www.doabledanny.com/Deploy-PHP-And-MySQL-to-Heroku>.

The advanced sql commands were incorporated into the sql file to ensure that only the expected values were produced and to simplify code that is reused. In our application, users are allowed to add ratings to recipes as well as edit previous ratings. To secure our data and verify that ratings were within the desired range, we added a constraint. This constraint checks that the rating value entered is only within 1 to 5, and otherwise the rating will not be added or updated. In addition, our application utilized select statements to retrieve recipes of a certain degree of difficulty. Since we had multiple checks to see if a recipe was easy, medium or hard, stored procedures were created to condense our code and prevent repetition. Instead of having to check whether a recipe had a certain number of steps each time the difficulty was needed, the procedure for that level of difficulty was instead called. This helped us when we created the tab where users could find and filter recipes by difficulty.

## Part 3 - Database Security at the Database Level

When we granted privileges we ensured that only us, the developers, had access to deleting, updating, selecting, and adding to all tables. On the other hand, end users that created accounts and will be using our application only have access to a few tables and can only make changes to those. Users create a username and password and thus are added to the Users table. Then, they are able to add, update and delete their own uploaded recipes via their username. They are unable to delete or edit any tables other than from things they have uploaded, whether that be ratings or recipes. Users can add recipes to the tab1\_2 table which contains all of our recipes. They can also delete recipes that they have uploaded to this table via

our “Delete Table” tab. They can edit ratings that they have done in the “Rate A Recipe” tab. In this tab of the website, they are able to add to the ratings table, which updates the ratinghistory\_ratings and ratinghistory\_items table. They are also able to see their rating history which accesses data in the ratings table. Lastly, they are able to edit ratings they have given in the past. These are the only tables in which the user directly has access to manipulating, and even then they can only edit, add, and delete based on their own username. On the other hand, the developers, Hannah, Ayushi, Tiki, and Grace have the ability to update, edit, delete and view data from any of the tables. This can be seen by our global privileges. In the picture below, hcdouglas, grace, ayushi, and td3 all have global privileges whereas user1 which is one we created for demoing only has the ability to select, insert, update, delete, execute on the specific tables specified above.

<input type="checkbox"/>	ayushi	localhost	Yes	ALL PRIVILEGES	Yes	 Edit privileges	 Export
<input type="checkbox"/>	globaluser	localhost	Yes	ALL PRIVILEGES	Yes	 Edit privileges	 Export
<input type="checkbox"/>	grace	localhost	Yes	ALL PRIVILEGES	Yes	 Edit privileges	 Export
<input type="checkbox"/>	hcdouglas	localhost	Yes	ALL PRIVILEGES	Yes	 Edit privileges	 Export
<input type="checkbox"/>	pma	localhost	No	USAGE	No	 Edit privileges	 Export
<input checked="" type="checkbox"/>	root	127.0.0.1	No	ALL PRIVILEGES	Yes	 Edit privileges	 Export
<input type="checkbox"/>	root	::1	No	ALL PRIVILEGES	Yes	 Edit privileges	 Export
<input type="checkbox"/>	root	localhost	No	ALL PRIVILEGES	Yes	 Edit privileges	 Export
<input type="checkbox"/>	td3	localhost	Yes	ALL PRIVILEGES	Yes	 Edit privileges	 Export
<input type="checkbox"/>	testing	%	Yes	ALL PRIVILEGES	Yes	 Edit privileges	 Export
<input type="checkbox"/>	user1	localhost	Yes	SELECT, INSERT, UPDATE, DELETE, EXECUTE	No	 Edit privileges	 Export

The security that we used at the database level was done with the below SQL commands. These same ones were done for ayushi, grace, td3. Since Hannah was the chief administrator she ran the commands so that the other developers could have access.

```
GRANT ALL PRIVILEGES ON Projectmilestone2.* TO 'hcdouglas_a'@'%';
GRANT ALL PRIVILEGES ON Projectmilestone2.* TO 'hcdouglas_b'@'%';
GRANT ALL PRIVILEGES ON Projectmilestone2.* TO 'hcdouglas_c'@'%';
GRANT ALL PRIVILEGES ON Projectmilestone2.* TO 'hcdouglas_d'@'%';
```

For general users:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON Projectmilestone2.tab1_2 TO 'PUBLIC'@'%';
GRANT SELECT, INSERT, UPDATE ON Projectmilestone2.ratings TO 'PUBLIC'@'%';
```

These grant privileges were manually added at the bottom of the SQL file used to import all tables and data.

## Part 4 - Database Security at the Application Level

Database security at the application level is incorporated in our project by using password hashing. PHP has a built in `password_hash()` function that creates a new password hash using a strong one-way hashing algorithm based on the password that the user enters at registration. We incorporated this in the `registration.php` file so that when a user entered their username or password, the `password_hash` function was used on their inputted password, and then used as the value for the password in the users table:

```
// Hash a new password for storing in the database.
// The function automatically generates a cryptographically safe salt.
$hashPassword = password_hash($password,PASSWORD_DEFAULT);

$sql="INSERT INTO users (username, password)
VALUES ('".$username."','".$hashPassword."')";
```

Hashing is beneficial because the password that is stored in the database is not the original password, so if the database is compromised somehow, the attacker would not be able to gain access to the original passwords. However, we are still able to compare the resulting hash password to the original password in the future when authenticating a user when they are logging in by using PHP's built in `password_verify()`. We did this in `login.php` to verify a specific user's original password with the hash password stored in the database. They are only logged in if the two are equivalent:

```
if($numRows == 1){
    $row = mysqli_fetch_assoc($rs);
    if(password_verify($password,$row['password'])){
        session_start();
        $_SESSION["loggedin"] = true;
        $_SESSION["id"] = $id;
        $_SESSION["username"] = $username;
        // Redirect user to welcome page
        header("location: main.php");
        exit;
    } else {
        echo "Wrong password";
    }
}
```