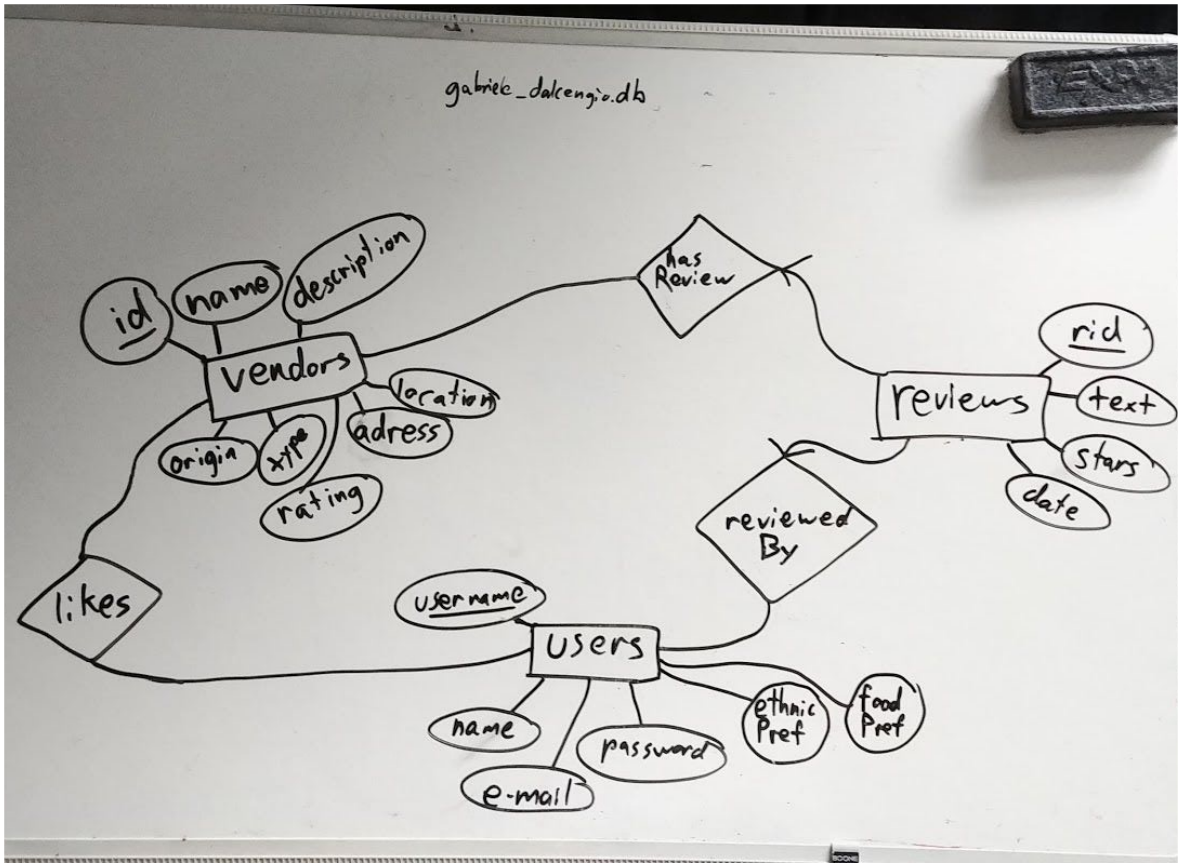
Database Design



My database is quite simple using three main tables. One of which contains all the data relating to the street vendors, another containing user information and the last containing all review information. In the vendors table, we have the primary key ‘id’. This auto increments with each added and is a way to keep this table organized easily. The rest of the attributes are general strings except for rating. Rating ties into the review table and will always only be 0 - 5. In the reviews table, we have a similar primary key because a review can only be written by one user, but users can write many reviews. There is then the text field with the actual content of the review and a stars attribute to influence the average rating of the vendor. This will work via the hasReview relationship, finding the average of all reviews attached to the id of a vendor. The last attribute is the date attribute in reviews which will be automatically generated at the time of submitting. The final table is the users table which contains all information about a registered user. Its primary key is username as none are allowed to be the same. There’s a hashed password, full name, email to soft prevent filling the database with spam. Finally, the ethnicPref and foodPref attributes in order for further customization in the next release. This table is connected to the other two via relationships “likes” and “reviewedBy”. The like relationship is for homepage

customization and the reviewedBy table is for displaying the reviews on the webpage. (initial names are slightly different than the ones in the real database)

Connectivity to Database

Technical report describing the design and implementation decisions

Database connectivity is standard across all pages, so I've decided to create a **functions.php** containing the general functions that all pages are able to use.

```php
//function to open connection to database
function openConnection(&$connection) {
    //open the connection to the database
    $connection = mysqli_connect("localhost", "root", "", "gabriele_dalcengio");

    // Test if connection succeeded
    if(mysqli_connect_errno()) {
        // if connection failed, skip the rest of PHP code, and print an error
        die("Database connection failed: " .
            mysqli_connect_error() .
            " (" . mysqli_connect_errno() . ")"
            );
    }
}


//function to perform the query to the table
function performQuery(&$connection, &$result, $queryToPerform) {
    //Perform database query
    $result = mysqli_query($connection, $queryToPerform);

    //Test if there was a query error
    if (!$result) {
        die("Database query failed.");
    }
}


//function to select the right listing for content.php
function getListing(&$connection, &$result) {
    $query = "SELECT * FROM vendors WHERE id=\"".$_GET["id"]."\";";
    // echo $query;

    performQuery($connection, $result, $query);   //now perform the query
}
```

This allowed for quick connections by setting up the $connection and $result variables, and passing them in the parameters by reference to the functions within

functions.php. I included this through the require_once(“functions.php”); function at the top so no issues arise.

```php
<?php
    require_once("functions.php");
?>

<!DOCTYPE html>
<html lang="en">

<head>

    <meta charset="utf-8">
    <title>Van Street Eats</title>
```

I can then call the functions I set up with the $connection and $result variables.

```php
        <section class="box box-main col-2of3">
    <?php
        $connection;
        $result;

        openConnection($connection);
        getListings($connection, $result);
        printRows($result);
        closeConnection($connection, $result);
    ?>

        </section>

        <!-- filter section -->
        <aside class="box col-1of3">
```

Because I made the parameters pass-by-reference using the ampersand character in the function definitions, the functions are allowed to change the variables outside their local scope. This also led to much much cleaner code in my opinion. I think I could’ve even set up another php page using the functions themselves, but I’ll save figuring out that for another time.

Secure Authentication Handling

In terms of secure authentication handling, I used the basic strategy of inserting hashed passwords into the database and checking via the database.

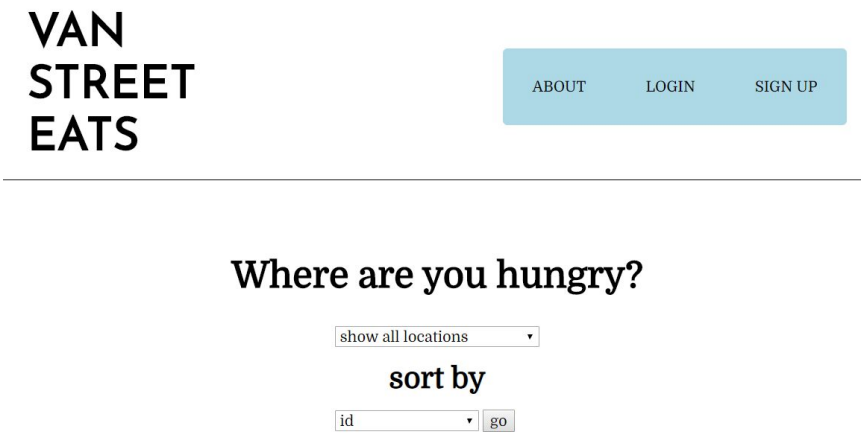
```php
    $connection;
    $result;
    //open up connection to user table
    openConnection($connection);

    //setting up variables and initializing them to receive the data from the form submitted
    $user = $_POST["user"];
    $passHashed = sha1($_POST["pass"]);
    $name = $_POST["name"];
    $email = $_POST["email"];
    $ethnic = $_POST["ePref"];
    $type = $_POST["fPref"];
```

```php
//function to perform the query to the table
function performQueryIsTrue(&$connection, &$result, $queryToPerform) {
    //Perform database query
    $result = mysqli_query($connection, $queryToPerform);

    //Test if there was a query error
    if (!$result) {
        die("Database query failed.");
    }

    $row = mysqli_fetch_row($result);
    if ($row) {     //if it returned any rows at all
        return true;
    } else {
        return false;
    }
}


//function to check registered user calling the function to perform query
function userRegistered(&$connection, &$result, $username, $password) {
    $pass = sha1($password);
    $query = "SELECT * FROM users WHERE username = '".$username."' AND pass = '".$pass."';";
    // echo $query;

    return performQueryIsTrue($connection, $result, $query);   //now perform the query
}
```

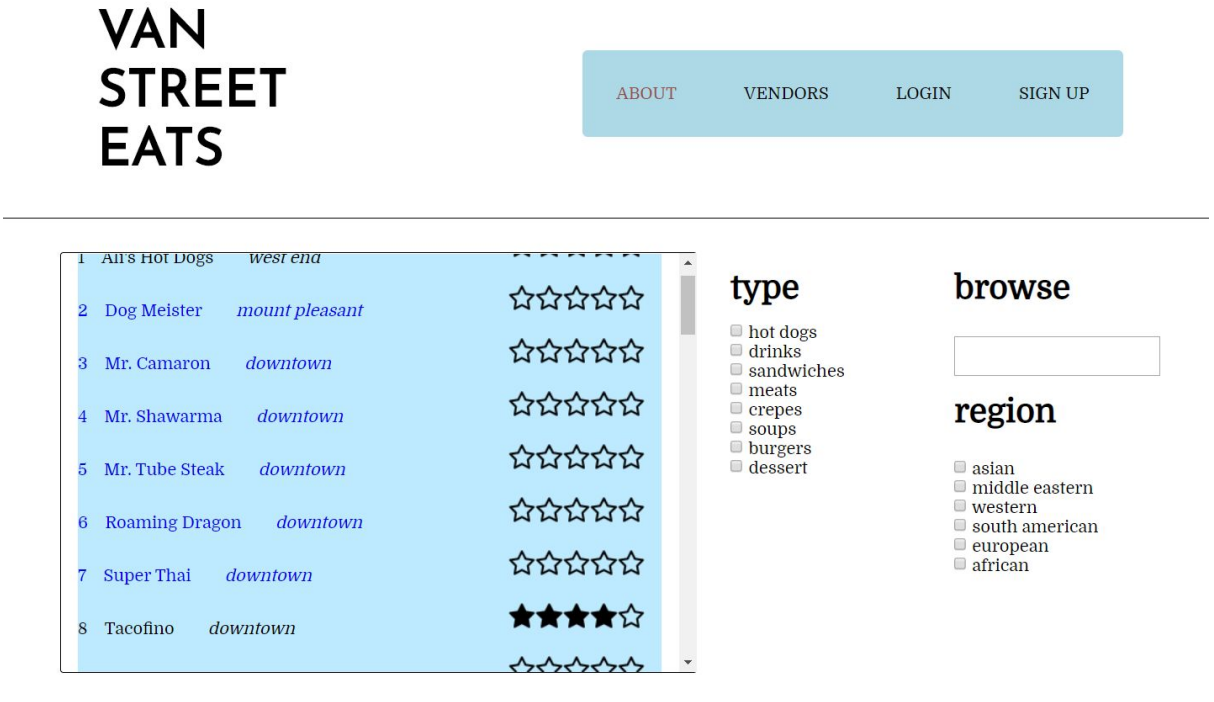
This is safer than hardcoding, but is still transmitted through plain text. With a larger application and more time, I would implement the htpassword strategy, but the current way works fine for my uses.

Visitor functionality

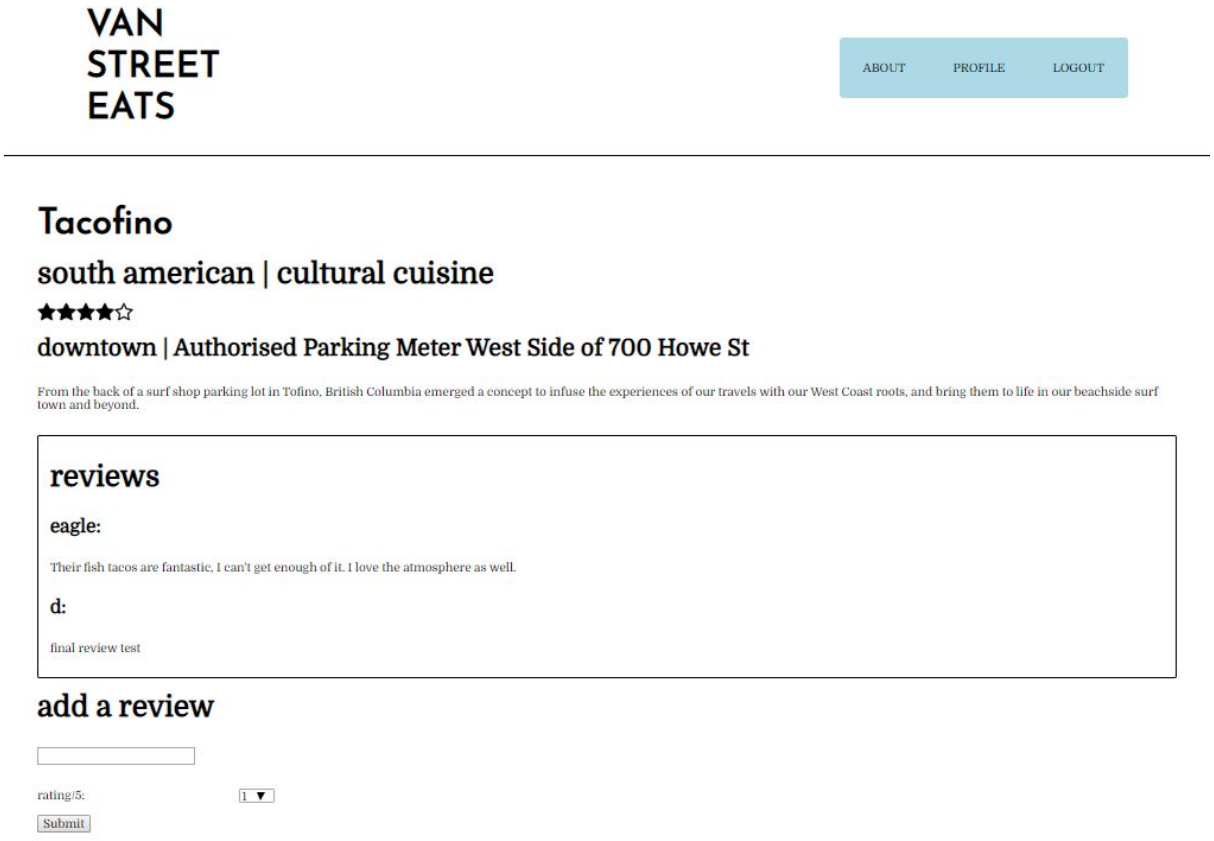
The visitor functionality is quite similar to registered user functionality. The main functionality is the core itself. From **index.php** the user can select which geographical region they would like the search to return, and the way that the information is sorted. The user can then press go leading to the next page.



The next page is **listings.php** which displays the listings in a scrollable box, based on what was submitted in the form of the last page. Each listing shows the id, name, location and stars. The filter on the smaller box to the right is for the ajax functionality which filters the listings in real time. There is a small bug where the click does not register when you click the words instead of checkboxes themselves. A browse input box was also added for the final implementation. Every time it is updated, the web is updated with the closest keyword match.



When the user clicks on any of the listings, they will be taken to **content.php**. In this page, the user will see the rest of the details of the listing.



(window size at zoom 50% to include everything)

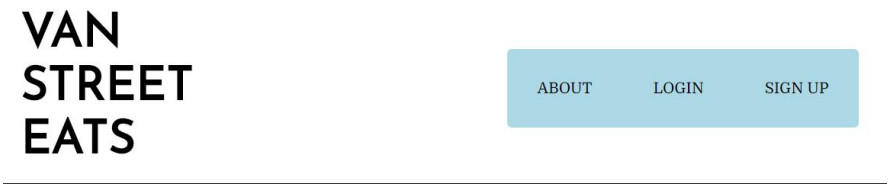
At this point, the structure differs a little bit for the members. Members will be able to see an add a review section as seen above whereas the normal visitors will just be able to see the reviews themselves. This is based on the $_SESSION variable. If (isset($_SESSION[“set-user”])) then it means that the user is a member and will echo the button to lead to the review writing page.

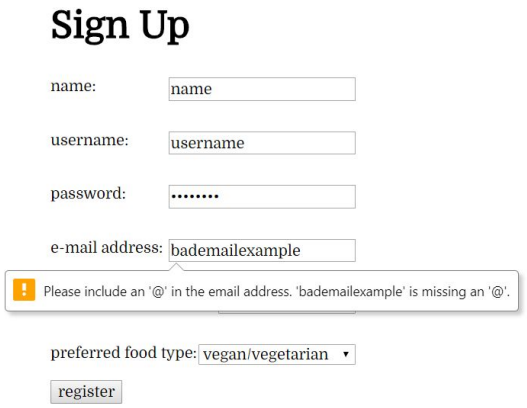
```php
if (isset($_SESSION["set_user"])) {
    echo "<form action=\"addedReview.php?id=".$id."\" method=\"POST\">";
    echo "<h2>add a review</h2>";
    echo "<div class=\"form-input\">";
        // text (reviewtext)
    echo "<input type=\"text\" name=\"reviewText\" required>";
    echo "</div>";
    echo "<div class=\"form-input\">";
        // rating (stars)
    echo "<label for=\"stars\">rating/5:</label>";
    echo "<select name=\"stars\">";
    echo "    <option value=\"1\">1</option>";
    echo "    <option value=\"2\">2</option>";
    echo "    <option value=\"3\">3</option>";
    echo "    <option value=\"4\">4</option>";
    echo "    <option value=\"5\">5</option>";
    echo "</select>";
    echo "</div>";
        // submit button
    echo "<input type=\"submit\">";
    echo "</form>";
}
```

Member registration and login

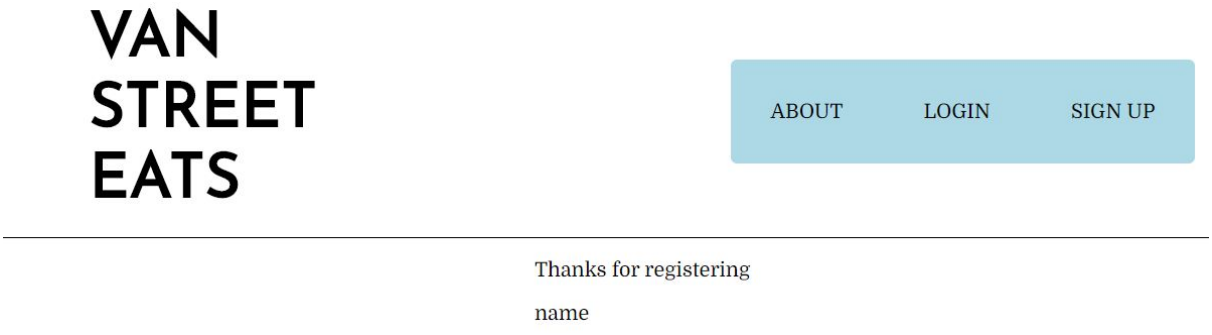
Registration

Members can register on the **signup.php** page on the nav bar at the top, which has a simple form asking for basic information such as name, email, ethnic preference, type of food preference, username, and password. If the user does not provide information, the form will stop the submission and throw an error for the user. This was implemented through the require keyword in the options of the form.





When the user presses submit, the data is then uploaded in the database with the password immediately hashed. This is through a select query which varies based on the form input. If any information is similar to another in the database, the database will throw an error, which will be error checked in the next release.

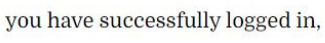


Once the data is submitted successfully, the user is taken to another page displaying the text “welcome, “ and then their name is appended onto the string.

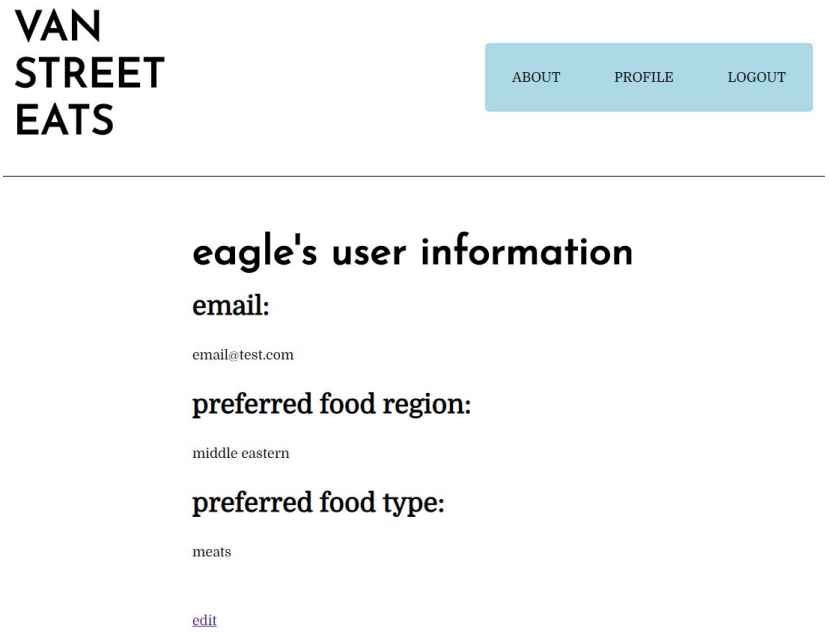
Login

Members can login through the login.php navigation at the top. On this page, there is a simple form asking for the username and password. The code then checks to see if the hashed password matches one on the database and if it does, the user is now logged in as shown in the user authentication section above. As soon as the user is logged in, the navigation bar at the top changes from (about | login | signup) to (about | profile | logout).

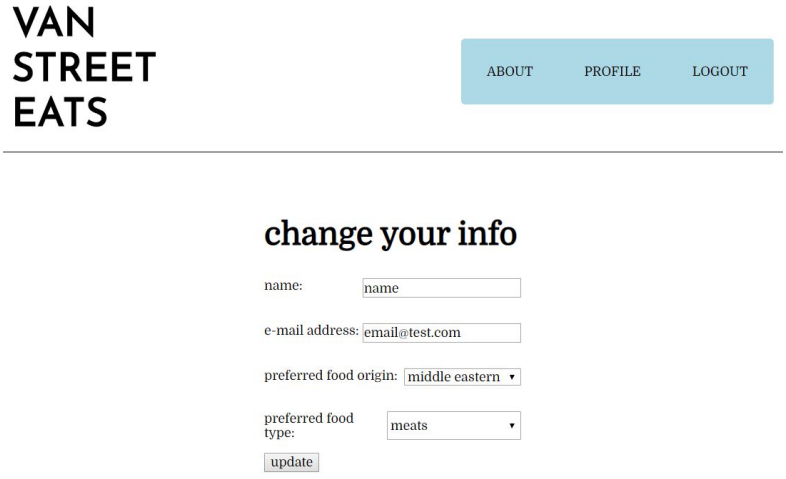
When the user is signed in, they will be led to a confirmation page similar to the registration page. (note the changed header)



At this point, the user can go to the profile.php page and view their information.



If they would like to change their information, they can go to the edit link at the bottom of the page.



At this point, the user can fill out their new information, or leave some at default. On submit, the code will send a query to the database to update and set this new information if the username matches. If the user would like to log out, they simply press the logout button on the nav bar.

you have successfully logged out

Reflection

Learning Experience and challenges

I really enjoyed how much I've learned in this project, having the interaction between front end and databases gave me the practical knowledge I needed to create similar, more refined projects. I had a bit of a struggle making sure that my query strings were consistent. They constantly had bugs and that is where I spent the most time. I enjoyed jquery very much and will be using it much more in the future. Another challenge I had was the overall ux. I need to get better and iterate quicker when designing websites. I'm satisfied as this was a first attempt, but there are many areas I could improve. The most important area I found was modularization of code. Since I did not quite understand php at the start, I wrote spaghetti code. If I were to redo this project, I'd modularize it from the start to keep it plenty organized.