IAT 352 - Internet Computing Technologies
Real Estate Listings
PA3: Personalization and Asynchronous Updates (AJAX)

Justin Lau **301291689**

Lucy Huang **301284331**

Eric Joseph Lee **301291689**

Database Design

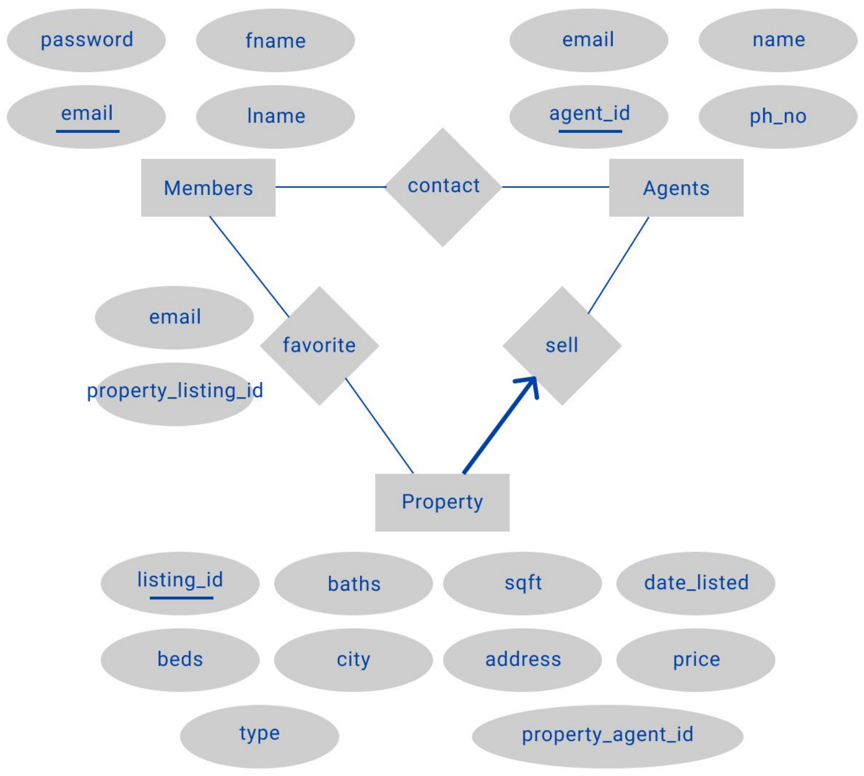


Figure 1. ER Diagram for our justin_lau.sql database design

Our database consists of 3 main entities: Members, Agents, and Properties. All of these have a relationship with each other due to our website's functionality. Each property, which is our main content unit, is sold by only one agent (arrow for key constraint; must have precisely one agent).

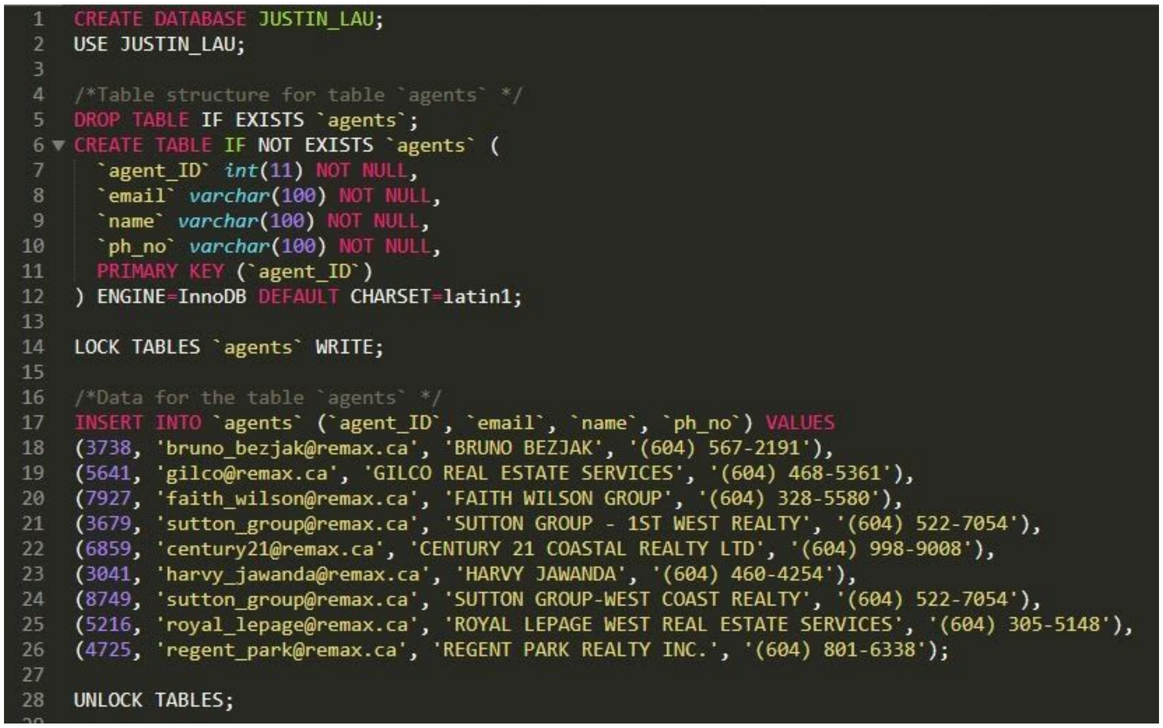
- Everything else is many to many.
- All members can contact all agents, and vice versa.
- All members can favorite all properties and vice versa.

The primary key for Members is the email attribute, listing_id for Properties, and agent_id for Agents. We also have a foreign key in Property called property_agent_id, that connects to the primary key in the Agent entity set.

With the completion of PA3, we have also implemented an additional table, in order to keep track of the members' favorited property listings. This table has two foreign keys, one called email, which references the Member's unique email address, and property_listing_id, which references each Properties listing_id.

Database Connectivity Code

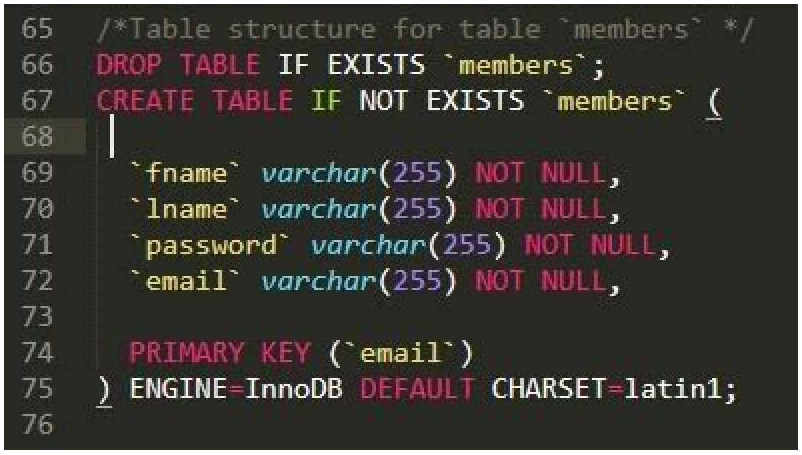
Our SQL database for Agents is created like this:



```sql
CREATE DATABASE JUSTIN_LAU;
USE JUSTIN_LAU;

/*Table structure for table `agents` */
DROP TABLE IF EXISTS `agents`;
CREATE TABLE IF NOT EXISTS `agents` (
  `agent_ID` int(11) NOT NULL,
  `email` varchar(100) NOT NULL,
  `name` varchar(100) NOT NULL,
  `ph_no` varchar(100) NOT NULL,
  PRIMARY KEY (`agent_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

LOCK TABLES `agents` WRITE;

/*Data for the table `agents` */
INSERT INTO `agents` (`agent_ID`, `email`, `name`, `ph_no`) VALUES
(3738, 'bruno_bezjak@remax.ca', 'BRUNO BEZJAK', '(604) 567-2191'),
(5641, 'gilco@remax.ca', 'GILCO REAL ESTATE SERVICES', '(604) 468-5361'),
(7927, 'faith_wilson@remax.ca', 'FAITH WILSON GROUP', '(604) 328-5580'),
(3679, 'sutton_group@remax.ca', 'SUTTON GROUP - 1ST WEST REALTY', '(604) 522-7054'),
(6859, 'century21@remax.ca', 'CENTURY 21 COASTAL REALTY LTD', '(604) 998-9008'),
(3041, 'harvy_jawanda@remax.ca', 'HARVY JAWANDA', '(604) 460-4254'),
(8749, 'sutton_group@remax.ca', 'SUTTON GROUP-WEST COAST REALTY', '(604) 522-7054'),
(5216, 'royal_lepage@remax.ca', 'ROYAL LEPAGE WEST REAL ESTATE SERVICES', '(604) 305-5148'),
(4725, 'regent_park@remax.ca', 'REGENT PARK REALTY INC.', '(604) 801-6338');

UNLOCK TABLES;
```

Figure 2. Create database and Agent table creation and data insertion (justin_lau.sql)

As per the assignment description, we've named our database as *firstname_lastname*. Following the examples provided in class, we created our tables such that the values could not be null. Our Property and Member tables are designed similarly.

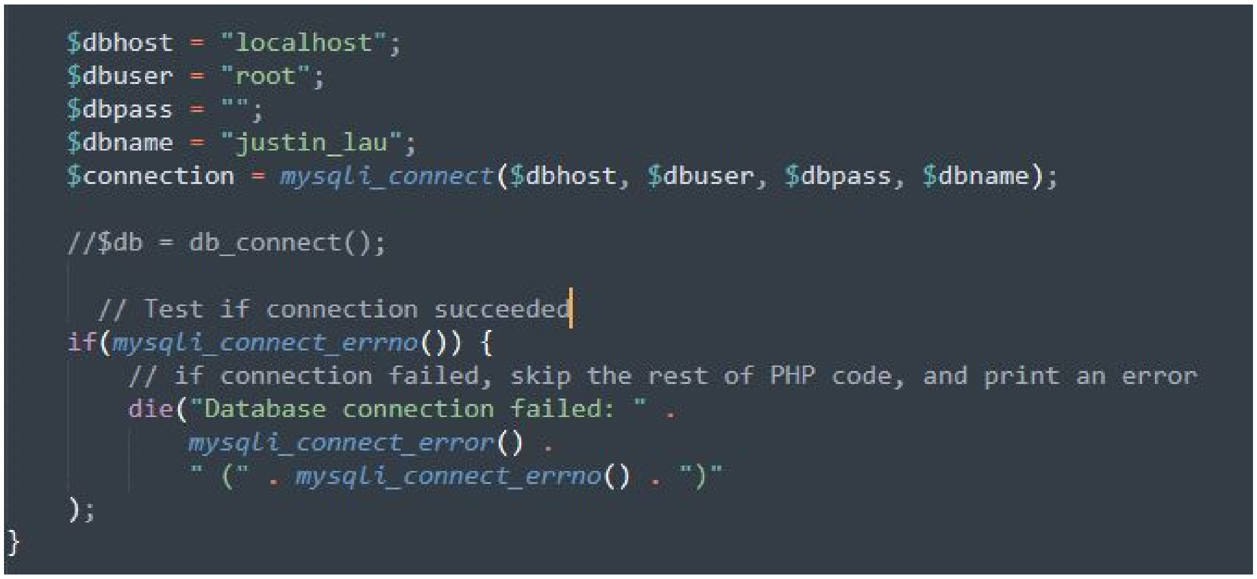


```sql
/*Table structure for table `members` */
DROP TABLE IF EXISTS `members`;
CREATE TABLE IF NOT EXISTS `members` (

  `fname` varchar(255) NOT NULL,
  `lname` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,

  PRIMARY KEY (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

```

Figure 3. Member table code in SQL (justin_lau.sql)

For our Member's table, since the table should dynamically allow for new members' insertion, we did not manually insert the data into the table. This is true for our newly implemented, favorite_properties table as well.

Also, following the assignment's description, we kept the default SQL database connection variables the same as we've been doing in class. Within the PHP code, when we need to connect to the database, we open the connection to the database specified above.

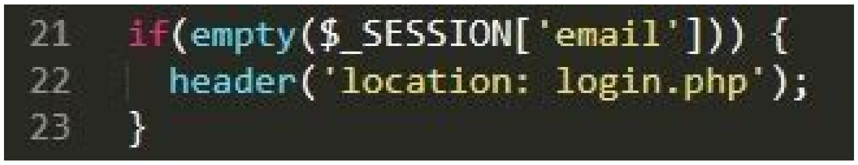


```php
    $dbhost = "localhost";
    $dbuser = "root";
    $dbpass = "";
    $dbname = "justin_lau";
    $connection = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname);

    //$db = db_connect();

      // 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() . ")"
    );
}
```

Figure 4. Database connectivity using PHP code. The connection will safely end if there is an error (server.php)

Secure Authentication Handling

We implemented secure authentication handling of our profile.php page. Here, only logged-in members can view and access this page. To check whether a user was a visitor or a logged-in member, we used **Sessions**. Sessions are held to store and share information between pages during the time a user interacts with the website.



```php
21  if(empty($_SESSION['email'])) {
22    header('location: login.php');
23  }
```

Figure 5. Using Sessions to ensure secure authentication handling (profile.php)

As shown in Figure 5, secure authentication handling can be done quite simply. Line 21 can be read as, if the user **has not** entered in an email during this session (if the email is empty), load the login.php page. This also prevents visitors from manually typing in the profile.php URL, as they will be redirected to the login.php page.

To further ensure that visitors cannot access the member's unique profile page, we've hidden it from the navigation bar. This way, visitors will not even know that there is a page to edit member information, making it easier to prevent them from accessing a secure page. As shown in Figure 6 below, Visitors do not have the same top-navigation bar as logged in Members. Additionally, when a member is signed in, they are greeted with a personalized welcome message, drawing from their first name when they created the account.

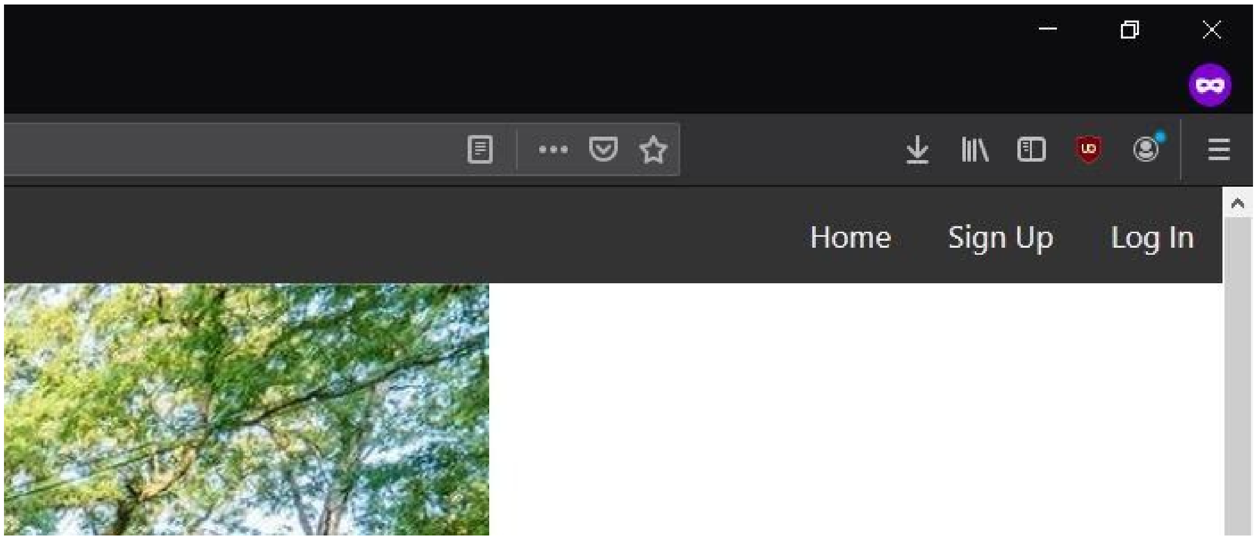
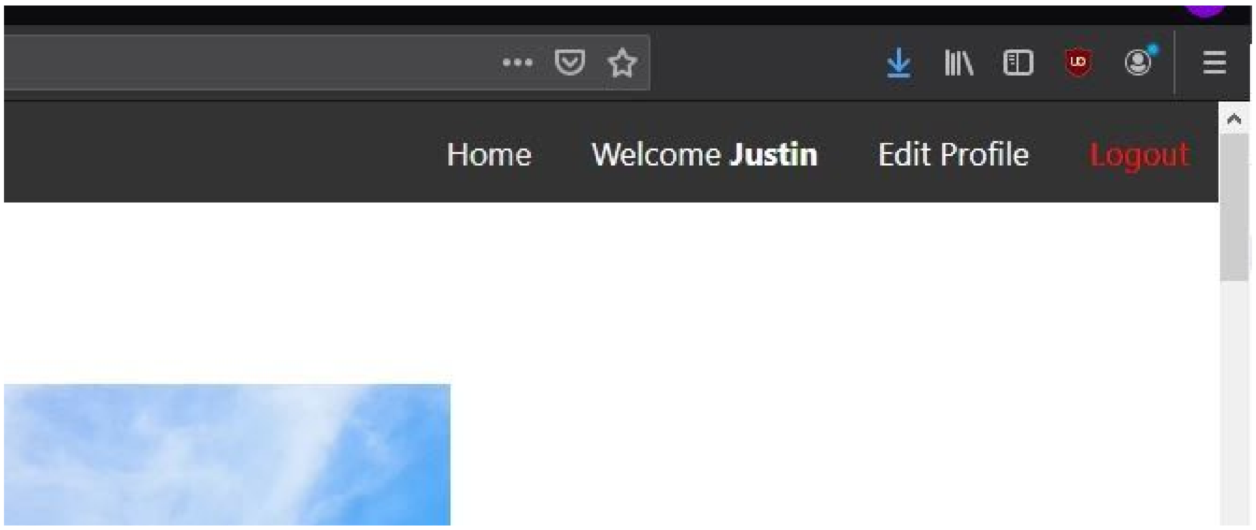
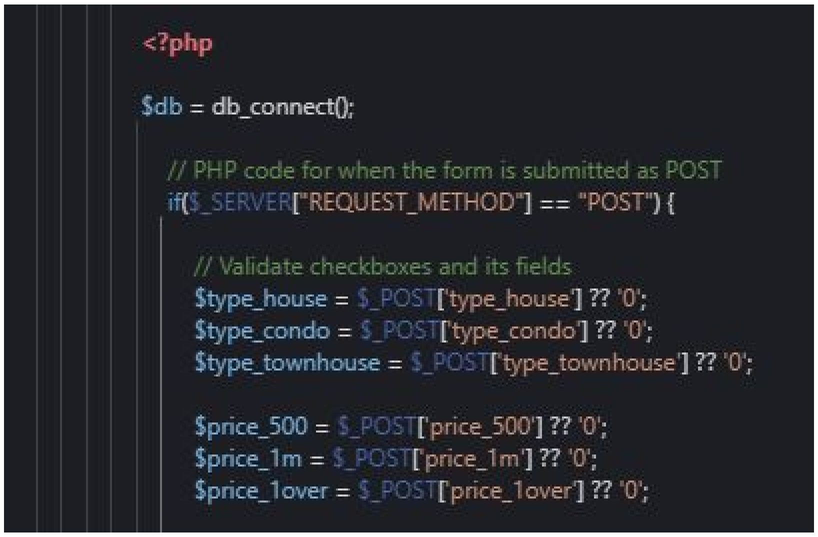


Figure 6. Visitor nav bar (top) and Member nav bar (bottom)

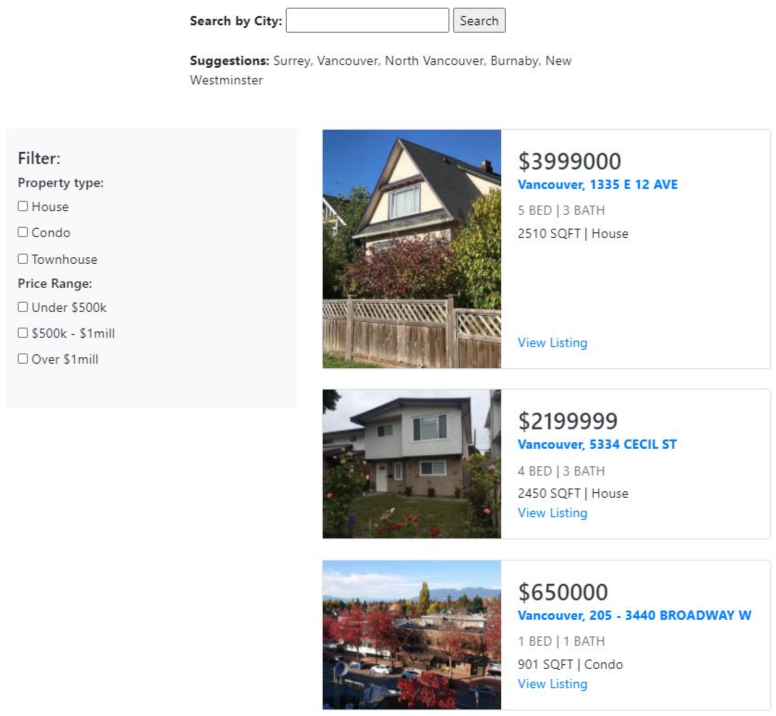


Visitor Functionality

Visitors can view all house listings (content units) on the Home Page. All visitors can filter the listings by the filter checkboxes on the left-hand side of the website and by the search bar above the list of house listings. They can filter by the different "Property Types" (houses, condos, and townhouses) and Price Range (less than $500k, between $500k to $1million, and over $1million) using checkboxes, and by cities by typing into the Search Bar.



```
<?php

$db = db_connect();

  // PHP code for when the form is submitted as POST
  if($_SERVER["REQUEST_METHOD"] == "POST") {

    // Validate checkboxes and its fields
    $type_house = $_POST['type_house'] ?? '0';
    $type_condo = $_POST['type_condo'] ?? '0';
    $type_townhouse = $_POST['type_townhouse'] ?? '0';

    $price_500 = $_POST['price_500'] ?? '0';
    $price_1m = $_POST['price_1m'] ?? '0';
    $price_1over = $_POST['price_1over'] ?? '0';
```



Search by City: [ ] Search

Suggestions: Surrey, Vancouver, North Vancouver, Burnaby, New Westminster

Filter:
Property type:
☐ House
☐ Condo
☐ Townhouse
Price Range:
☐ Under $500k
☐ $500k - $1mill
☐ Over $1mill

$3999000
Vancouver, 1335 E 12 AVE
5 BED | 3 BATH
2510 SQFT | House
View Listing

$2199999
Vancouver, 5334 CECIL ST
4 BED | 3 BATH
2450 SQFT | House
View Listing

$650000
Vancouver, 205 - 3440 BROADWAY W
1 BED | 1 BATH
901 SQFT | Condo
View Listing

Figure 7. Filtering menu for visitors and members (index.php)

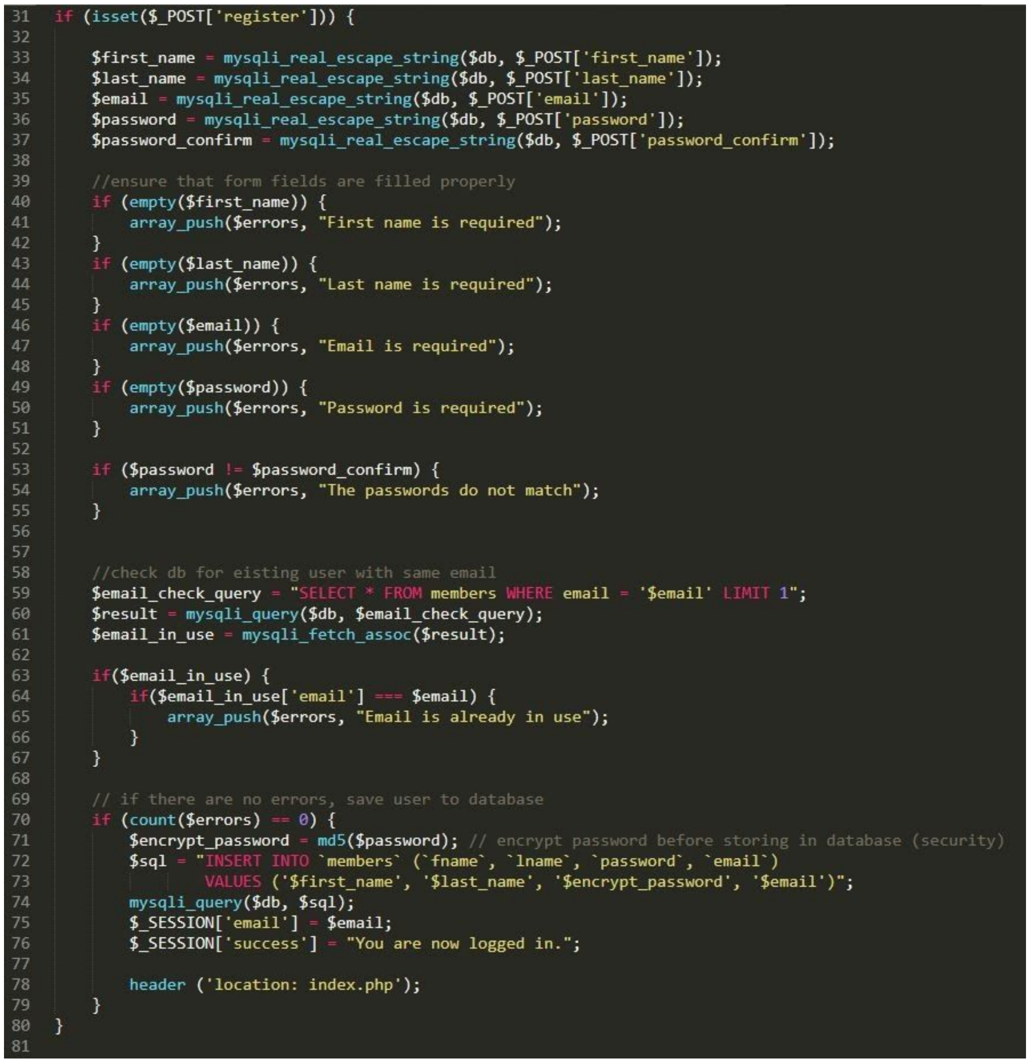
Visitors can also view the individual content pages of each of the listings in the database, providing them with more content and details of each property listed.

Visitors can also sign up to become a member or log in if they are already a member. This will be elaborated on below.

Member Registration and Login

The Registration works as follows:

Any visitor can register as a member using the form on the page signup.php.

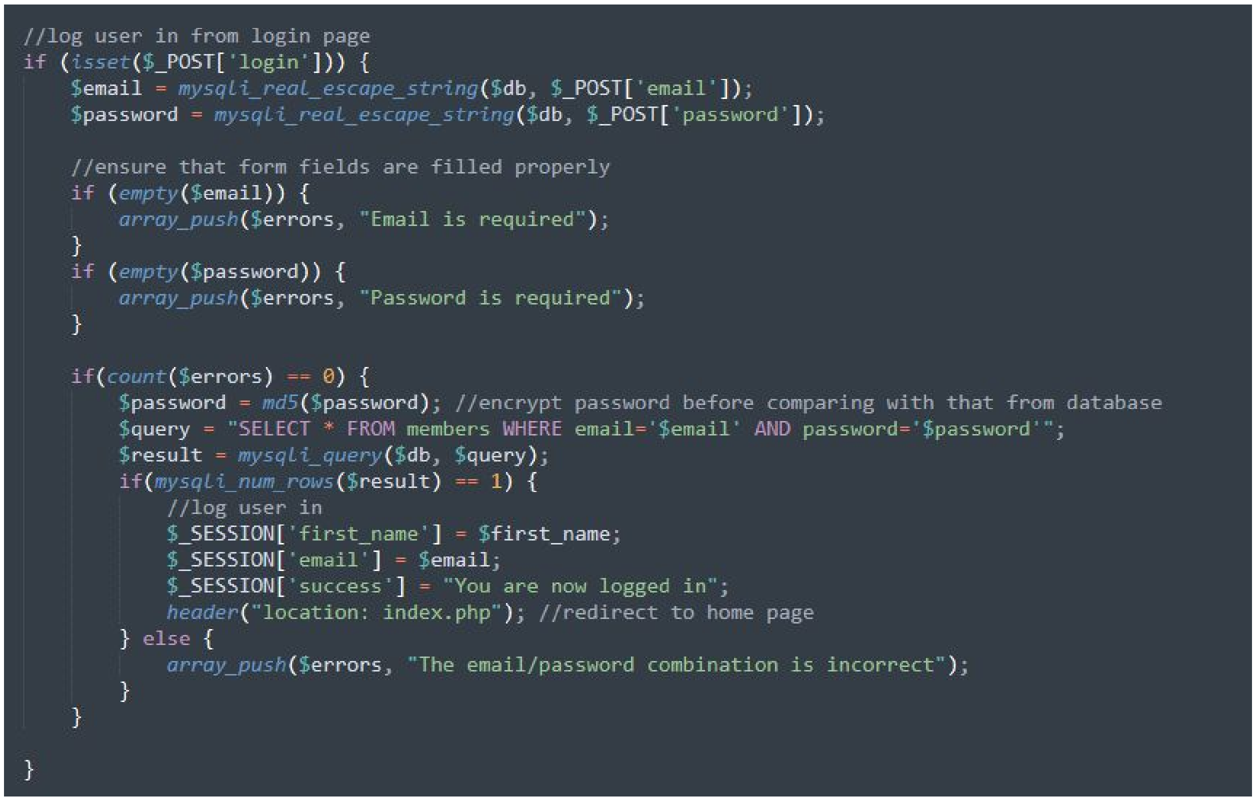


```php
if (isset($_POST['register'])) {

    $first_name = mysqli_real_escape_string($db, $_POST['first_name']);
    $last_name = mysqli_real_escape_string($db, $_POST['last_name']);
    $email = mysqli_real_escape_string($db, $_POST['email']);
    $password = mysqli_real_escape_string($db, $_POST['password']);
    $password_confirm = mysqli_real_escape_string($db, $_POST['password_confirm']);

    //ensure that form fields are filled properly
    if (empty($first_name)) {
        array_push($errors, "First name is required");
    }
    if (empty($last_name)) {
        array_push($errors, "Last name is required");
    }
    if (empty($email)) {
        array_push($errors, "Email is required");
    }
    if (empty($password)) {
        array_push($errors, "Password is required");
    }

    if ($password != $password_confirm) {
        array_push($errors, "The passwords do not match");
    }


    //check db for eisting user with same email
    $email_check_query = "SELECT * FROM members WHERE email = '$email' LIMIT 1";
    $result = mysqli_query($db, $email_check_query);
    $email_in_use = mysqli_fetch_assoc($result);

    if($email_in_use) {
        if($email_in_use['email'] === $email) {
            array_push($errors, "Email is already in use");
        }
    }

    // if there are no errors, save user to database
    if (count($errors) == 0) {
        $encrypt_password = md5($password); // encrypt password before storing in database (security)
        $sql = "INSERT INTO `members` (`fname`, `lname`, `password`, `email`)
                VALUES ('$first_name', '$last_name', '$encrypt_password', '$email')";
        mysqli_query($db, $sql);
        $_SESSION['email'] = $email;
        $_SESSION['success'] = "You are now logged in.";

        header ('location: index.php');
    }
}

```

Figure 8. PHP code that stores the information inputted by the user (server.php)

As shown in Figure 7, it takes the input of the visitor's First Name, Last Name, Email, Password, and Password Confirmation. After the register button is pressed, it calls upon server.php, which opens up a new session and the database justin_lau.sql. Once "Register" is prompted, it collects each of the fields users have inputted. It checks if the user has successfully input into each field - if any are empty, an error message is pushed to show specifically which field is missing. Since a Member's key attribute is their email, we also write a check to see if the email is already stored in the database. Once there are no errors, the password is first encrypted, then all of the inputs are now saved in the 'members' table in the database. After successfully registering for an account or logging in with an existing account, the user will be redirected to the home page (Figure 7, line 78). They will have a unique header and features unique to registered members. As discussed in the Secure Authentication Handling section, this includes editing their profile information, such as their first or last name.

If a member is logged out, they must log back in through the Login page (login.php). It works as such: The form takes in their email (the unique identifier), as well as their password. Once the "Login" button is clicked, it calls upon server.php, just like in registration. It makes sure that both email and password are typed in, and if they both are, it encrypts the password first then finds the member with the same email and password. If it is successful, it redirects them to the Home Page as that member; otherwise, it will display an error message that tells the user that the email/password combination is incorrect.

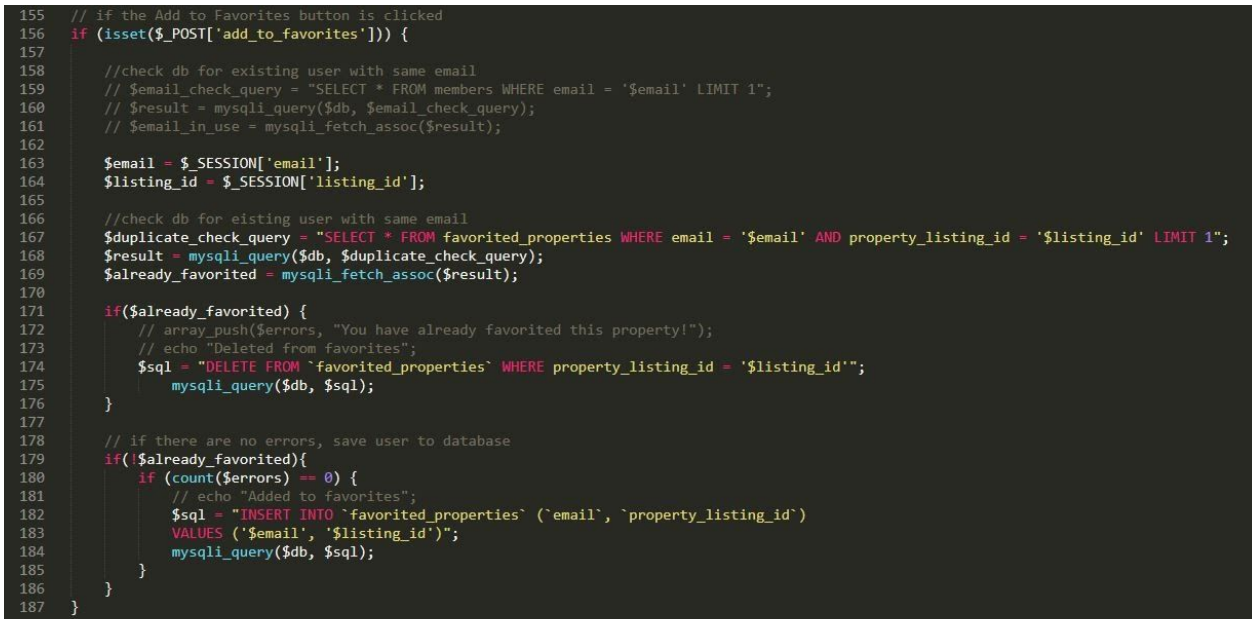


```php
//log user in from login page
if (isset($_POST['login'])) {
    $email = mysqli_real_escape_string($db, $_POST['email']);
    $password = mysqli_real_escape_string($db, $_POST['password']);

    //ensure that form fields are filled properly
    if (empty($email)) {
        array_push($errors, "Email is required");
    }
    if (empty($password)) {
        array_push($errors, "Password is required");
    }

    if(count($errors) == 0) {
        $password = md5($password); //encrypt password before comparing with that from database
        $query = "SELECT * FROM members WHERE email='$email' AND password='$password'";
        $result = mysqli_query($db, $query);
        if(mysqli_num_rows($result) == 1) {
            //log user in
            $_SESSION['first_name'] = $first_name;
            $_SESSION['email'] = $email;
            $_SESSION['success'] = "You are now logged in";
            header("location: index.php"); //redirect to home page
        } else {
            array_push($errors, "The email/password combination is incorrect");
        }
    }

}
```

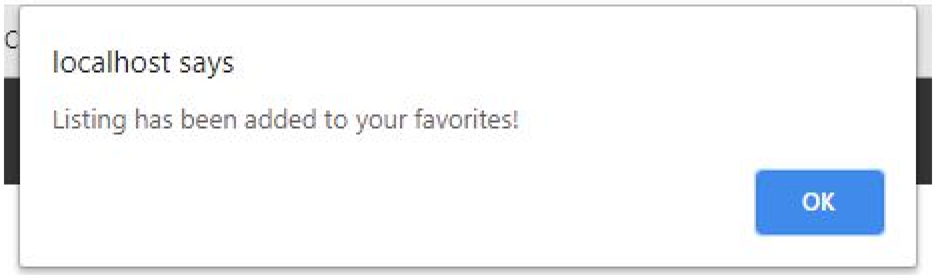
Figure 9. PHP code for verifying a user's login information (server.php)

Personalization for Members

Once a visitor signs up to become a Member and logs in, they are given the **unique personalization functionality** of saving their favorite listings. When a Member sees a property listing that they like, they can click “Add to Favorites,” which stores the listing_id of the property they are viewing and adds it onto the Home page, displaying all of the ones they have liked. Once the button “Add to Favorites” is clicked, the system stores the selection into the database into a new table and alerts the user that they have added the listing into their favorites.



```php
// if the Add to Favorites button is clicked
if (isset($_POST['add_to_favorites'])) {

    //check db for existing user with same email
    // $email_check_query = "SELECT * FROM members WHERE email = '$email' LIMIT 1";
    // $result = mysqli_query($db, $email_check_query);
    // $email_in_use = mysqli_fetch_assoc($result);

    $email = $_SESSION['email'];
    $listing_id = $_SESSION['listing_id'];

    //check db for eisting user with same email
    $duplicate_check_query = "SELECT * FROM favorited_properties WHERE email = '$email' AND property_listing_id = '$listing_id' LIMIT 1";
    $result = mysqli_query($db, $duplicate_check_query);
    $already_favorited = mysqli_fetch_assoc($result);

    if($already_favorited) {
        // array_push($errors, "You have already favorited this property!");
        // echo "Deleted from favorites";
        $sql = "DELETE FROM `favorited_properties` WHERE property_listing_id = '$listing_id'";
            mysqli_query($db, $sql);
    }

    // if there are no errors, save user to database
    if(!$already_favorited){
        if (count($errors) == 0) {
            // echo "Added to favorites";
            $sql = "INSERT INTO `favorited_properties` (`email`, `property_listing_id`)
            VALUES ('$email', '$listing_id')";
            mysqli_query($db, $sql);
        }
    }
}
```

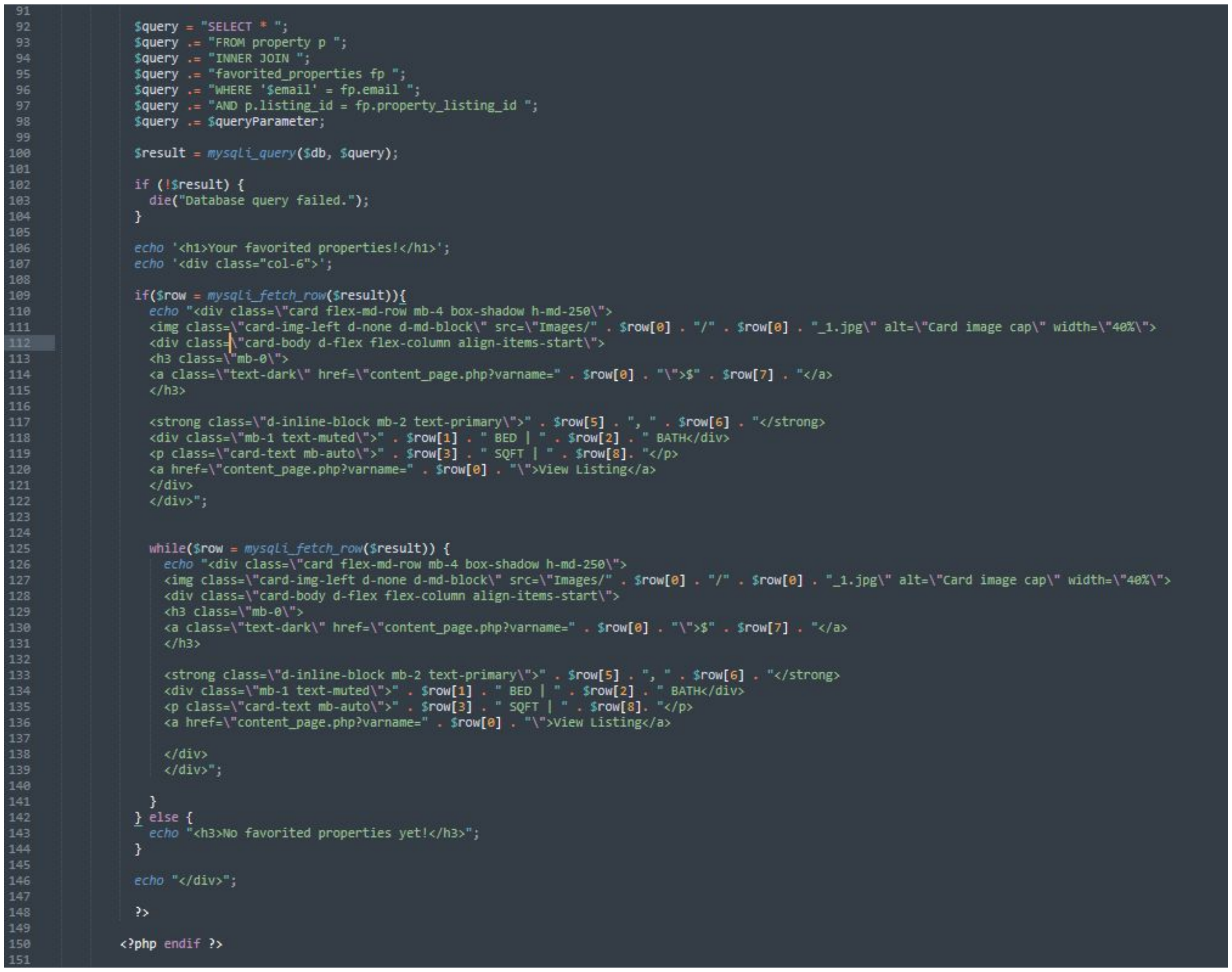


localhost says

Listing has been added to your favorites!

Figure 10. PHP code for Server and what the user sees for adding to/removing from Favorites (server.php)

This page displays all of the listings as their content units, in the way that listings are displayed on the filtering/search page. The Member simply has to click onto one of the content units to access that listing’s full content page, where they can view the description and information of the house listing and choose to remove it from their favorites using the same method they added it (Figure 10), except instead it will show as a “Remove from Favorites” button. Only Members will have these functionalities, and they must be logged in. If they choose to log out and then log back in, their Favorite listings will be saved under their session so they can always come back to it. In the SQL query in Figure 11 (lines 52-58), it cross references the logged in member’s email with an email in the favorited_properties table, and then cross references each property listing with a listing again from the favorited_properties table.



```
91
92        $query = "SELECT * ";
93        $query .= "FROM property p ";
94        $query .= "INNER JOIN ";
95        $query .= "favorited_properties fp ";
96        $query .= "WHERE '$email' = fp.email ";
97        $query .= "AND p.listing_id = fp.property_listing_id ";
98        $query .= $queryParameter;
99
100       $result = mysqli_query($db, $query);
101
102       if (!$result) {
103         die("Database query failed.");
104       }
105
106       echo '<h1>Your favorited properties!</h1>';
107       echo '<div class="col-6">';
108
109       if($row = mysqli_fetch_row($result)){
110         echo "<div class=\"card flex-md-row mb-4 box-shadow h-md-250\">
111         <img class=\"card-img-left d-none d-md-block\" src=\"Images/" . $row[0] . "/" . $row[0] . "_1.jpg\" alt=\"Card image cap\" width=\"40%\">
112         <div class=\"card-body d-flex flex-column align-items-start\">
113         <h3 class=\"mb-0\">
114         <a class=\"text-dark\" href=\"content_page.php?varname=" . $row[0] . "\">$" . $row[7] . "</a>
115         </h3>
116
117         <strong class=\"d-inline-block mb-2 text-primary\">" . $row[5] . ", " . $row[6] . "</strong>
118         <div class=\"mb-1 text-muted\">" . $row[1] . " BED | " . $row[2] . " BATH</div>
119         <p class=\"card-text mb-auto\">" . $row[3] . " SQFT | " . $row[8]. "</p>
120         <a href=\"content_page.php?varname=" . $row[0] . "\">View Listing</a>
121         </div>
122         </div>";
123
124
125         while($row = mysqli_fetch_row($result)) {
126           echo "<div class=\"card flex-md-row mb-4 box-shadow h-md-250\">
127           <img class=\"card-img-left d-none d-md-block\" src=\"Images/" . $row[0] . "/" . $row[0] . "_1.jpg\" alt=\"Card image cap\" width=\"40%\">
128           <div class=\"card-body d-flex flex-column align-items-start\">
129           <h3 class=\"mb-0\">
130           <a class=\"text-dark\" href=\"content_page.php?varname=" . $row[0] . "\">$" . $row[7] . "</a>
131           </h3>
132
133           <strong class=\"d-inline-block mb-2 text-primary\">" . $row[5] . ", " . $row[6] . "</strong>
134           <div class=\"mb-1 text-muted\">" . $row[1] . " BED | " . $row[2] . " BATH</div>
135           <p class=\"card-text mb-auto\">" . $row[3] . " SQFT | " . $row[8]. "</p>
136           <a href=\"content_page.php?varname=" . $row[0] . "\">View Listing</a>
137
138           </div>
139           </div>";
140
141         }
142       } else {
143         echo "<h3>No favorited properties yet!</h3>";
144       }
145
146       echo "</div>";
147
148       ?>
149
150     <?php endif ?>
151
```

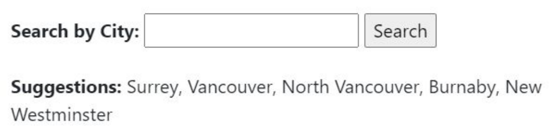
Figure 11. PHP code to view Favourite listings on the Home page for Members (index.php)

AJAX

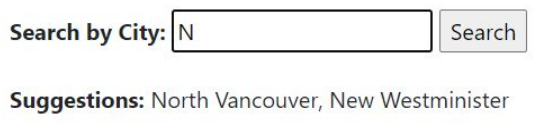
We have two AJAX Components to our project for three different functionalities. They are as follows:

1. **SearchBar Suggestions:**

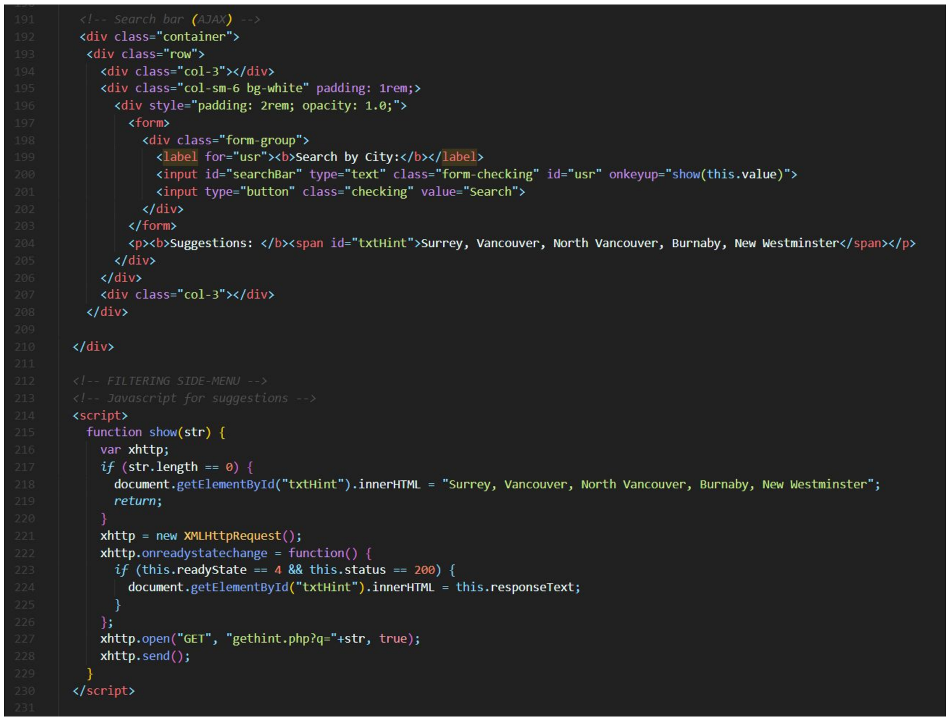
The search bar is used to filter the data by the name of the cities. They suggest the names of the city that's being typed. As the user types common alphabets, the suggestion features all the cities that can be typed which is all done in AJAX shown in Figure 12. The suggestions list changes dynamically depending on what the user has started to type in. This helps to ensure that when they are using the Search bar, they spell each city's name correctly, otherwise they will not be able to view any listings.



Suggestions: Surrey, Vancouver, North Vancouver, Burnaby, New Westminster



Suggestions: North Vancouver, New Westminister

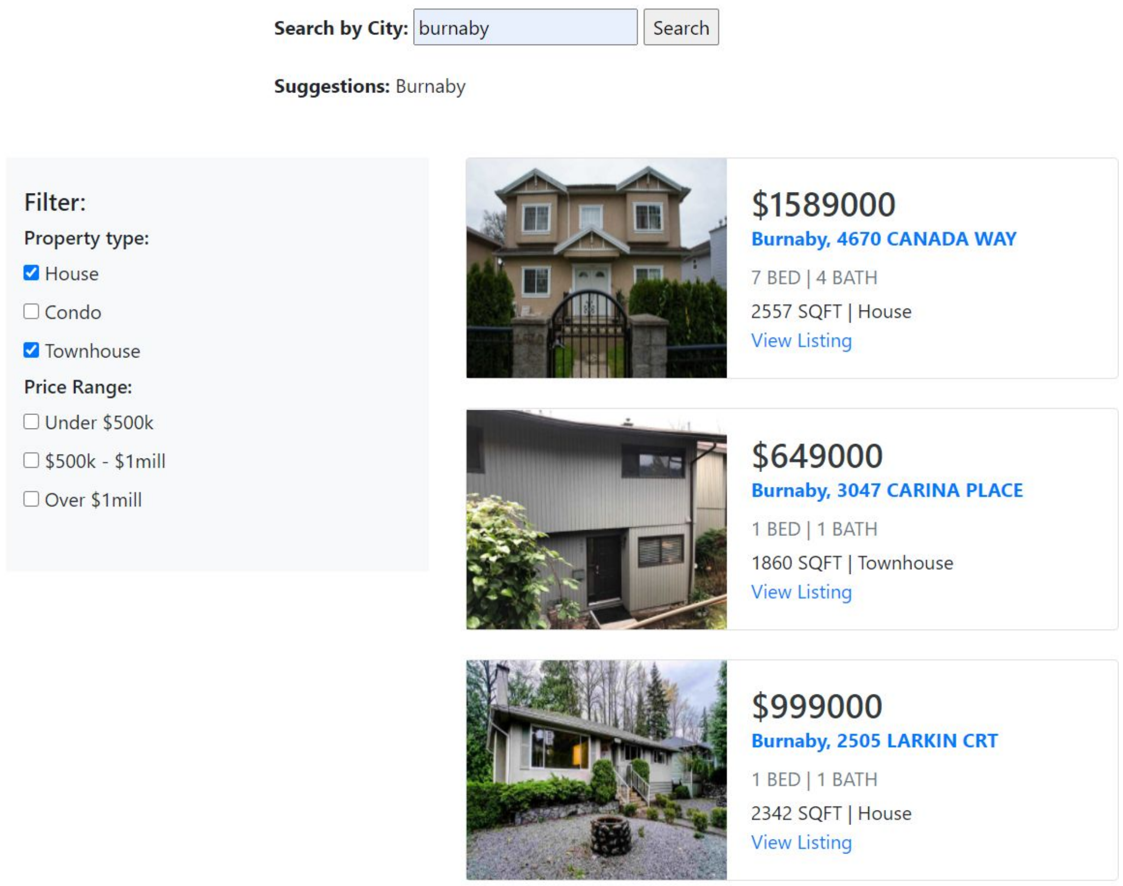


```
 <!-- Search bar (AJAX) -->
 <div class="container">
  <div class="row">
    <div class="col-3"></div>
    <div class="col-sm-6 bg-white" padding: 1rem;>
      <div style="padding: 2rem; opacity: 1.0;">
        <form>
          <div class="form-group">
            <label for="usr"><b>Search by City:</b></label>
            <input id="searchBar" type="text" class="form-checking" id="usr" onkeyup="show(this.value)">
            <input type="button" class="checking" value="Search">
          </div>
        </form>
        <p><b>Suggestions: </b><span id="txtHint">Surrey, Vancouver, North Vancouver, Burnaby, New Westminster</span></p>
      </div>
    </div>
    <div class="col-3"></div>
  </div>

</div>

<!-- FILTERING SIDE-MENU -->
<!-- Javascript for suggestions -->
<script>
  function show(str) {
    var xhttp;
    if (str.length == 0) {
      document.getElementById("txtHint").innerHTML = "Surrey, Vancouver, North Vancouver, Burnaby, New Westminster";
      return;
    }
    xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        document.getElementById("txtHint").innerHTML = this.responseText;
      }
    };
    xhttp.open("GET", "gethint.php?q="+str, true);
    xhttp.send();
  }
</script>
```

Figure 12. AJAX SearchBar Suggestion function on Home Page (index.php)

Checkbox Filtering and Search filter by City:

Filtering of displayed data through checkbox for the Property type and the Price range is done asynchronously the moment they are checked. They do not require a button to be pressed which solves our previous problem of not having the checkboxes automatically removed when they were applied. When a city is typed correctly, and the search button is pressed, then the listings will also filter according to the city and the checkboxes as shown below in Figure 13.



Search by City: burnaby Search

Suggestions: Burnaby

Filter:

Property type:

- [x] House
- [ ] Condo
- [x] Townhouse

Price Range:

- [ ] Under $500k
- [ ] $500k - $1mill
- [ ] Over $1mill

$1589000
Burnaby, 4670 CANADA WAY
7 BED | 4 BATH
2557 SQFT | House
View Listing

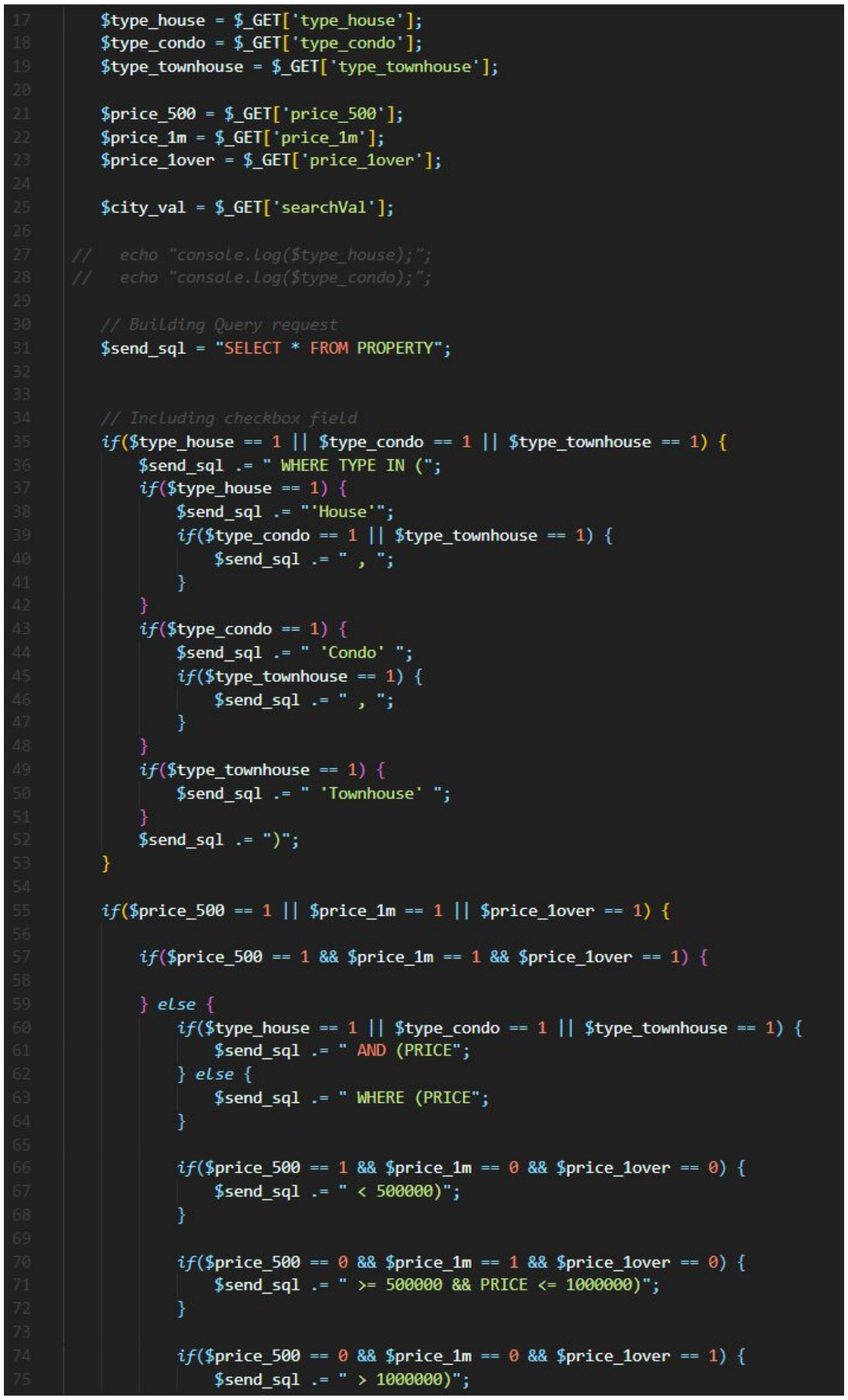
$649000
Burnaby, 3047 CARINA PLACE
1 BED | 1 BATH
1860 SQFT | Townhouse
View Listing

$999000
Burnaby, 2505 LARKIN CRT
1 BED | 1 BATH
2342 SQFT | House
View Listing

Figure 13. Filtering on Home Page (index.php)

All the above is done on one AJAX script using one function. When this function is called, it takes the values of all the checkboxes and the search text and sends it to a PHP file getrecords_ajax.php, where the SQL Query is processed and retrieved from the database.

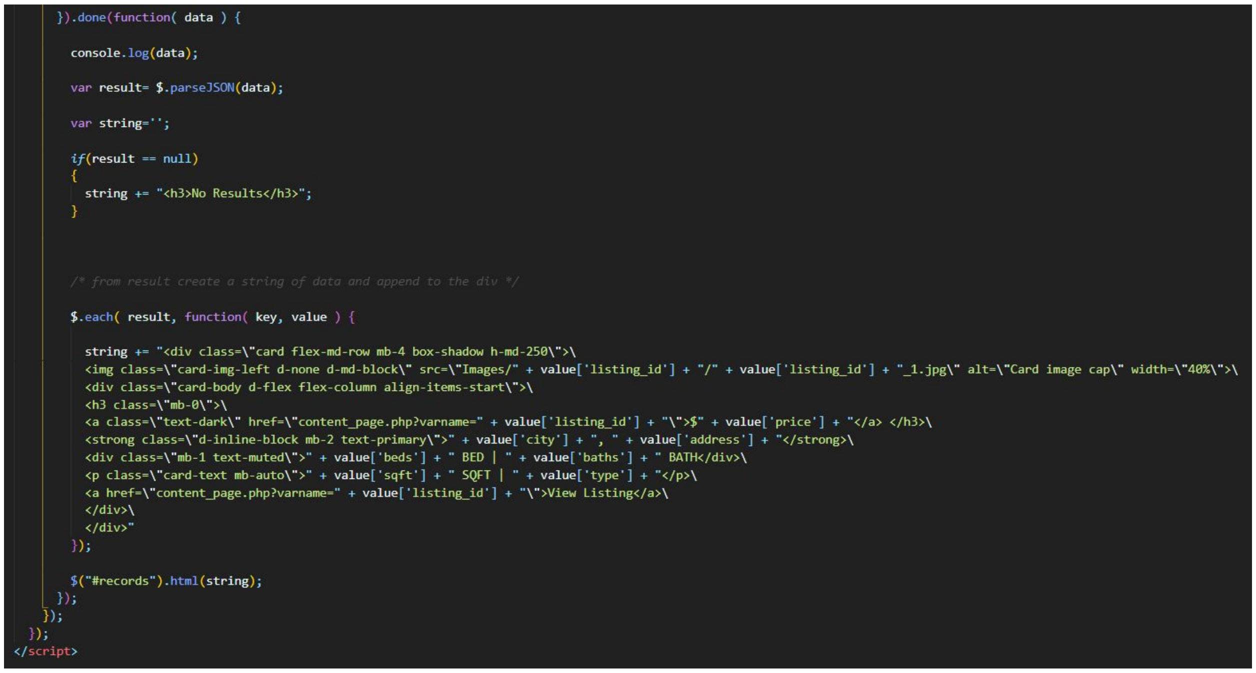
As shown in Figure 14 below, the code for getrecords_ajax.php, the values are taken and processed into an SQL Query and sent to retrieve the data.



```
    $type_house = $_GET['type_house'];
    $type_condo = $_GET['type_condo'];
    $type_townhouse = $_GET['type_townhouse'];

    $price_500 = $_GET['price_500'];
    $price_1m = $_GET['price_1m'];
    $price_1over = $_GET['price_1over'];

    $city_val = $_GET['searchVal'];

//   echo "console.log($type_house);";
//   echo "console.log($type_condo);";

    // Building Query request
    $send_sql = "SELECT * FROM PROPERTY";


    // Including checkbox field
    if($type_house == 1 || $type_condo == 1 || $type_townhouse == 1) {
        $send_sql .= " WHERE TYPE IN (";
        if($type_house == 1) {
            $send_sql .= "'House'";
            if($type_condo == 1 || $type_townhouse == 1) {
                $send_sql .= " , ";
            }
        }
        if($type_condo == 1) {
            $send_sql .= " 'Condo' ";
            if($type_townhouse == 1) {
                $send_sql .= " , ";
            }
        }
        if($type_townhouse == 1) {
            $send_sql .= " 'Townhouse' ";
        }
        $send_sql .= ")";
    }

    if($price_500 == 1 || $price_1m == 1 || $price_1over == 1) {

        if($price_500 == 1 && $price_1m == 1 && $price_1over == 1) {

        } else {
            if($type_house == 1 || $type_condo == 1 || $type_townhouse == 1) {
                $send_sql .= " AND (PRICE";
            } else {
                $send_sql .= " WHERE (PRICE";
            }

            if($price_500 == 1 && $price_1m == 0 && $price_1over == 0) {
                $send_sql .= " < 500000)";
            }

            if($price_500 == 0 && $price_1m == 1 && $price_1over == 0) {
                $send_sql .= " >= 500000 && PRICE <= 1000000)";
            }

            if($price_500 == 0 && $price_1m == 0 && $price_1over == 1) {
                $send_sql .= " > 1000000)";
```

Figure 14. SQL code that adjusts the query depending on what data was retrieved from the checkboxes (getrecords_ajax.php)

After the data is retrieved, it is sent back to index.php, where the data is processed to be displayed.



```
      }).done(function( data ) {

        console.log(data);

        var result= $.parseJSON(data);

        var string='';

        if(result == null)
        {
          string += "<h3>No Results</h3>";
        }



        /* from result create a string of data and append to the div */

        $.each( result, function( key, value ) {

          string += "<div class=\"card flex-md-row mb-4 box-shadow h-md-250\">\
          <img class=\"card-img-left d-none d-md-block\" src=\"Images/" + value['listing_id'] + "/" + value['listing_id'] + "_1.jpg\" alt=\"Card image cap\" width=\"40%\">\
          <div class=\"card-body d-flex flex-column align-items-start\">\
          <h3 class=\"mb-0\">\
          <a class=\"text-dark\" href=\"content_page.php?varname=" + value['listing_id'] + "\">$" + value['price'] + "</a> </h3>\
          <strong class=\"d-inline-block mb-2 text-primary\">" + value['city'] + ", " + value['address'] + "</strong>\
          <div class=\"mb-1 text-muted\">" + value['beds'] + " BED | " + value['baths'] + " BATH</div>\
          <p class=\"card-text mb-auto\">" + value['sqft'] + " SQFT | " + value['type'] + "</p>\
          <a href=\"content_page.php?varname=" + value['listing_id'] + "\">View Listing</a>\
          </div>\
          </div>"
        });

        $("#records").html(string);
      });
    });
  });
</script>
```

Figure15. PHP code to display a filtered list of property content units (index.php)

<u>Learning Experience + Challenges</u>

Overall, this was a great learning experience for PHP, MySQL, and AJAX. The way the project was split up into three major parts was beneficial for us. It helped us learn which parts needed to come first and build upon that base to create a functional website that can handle and link together common functionalities like filtering and registration. Although we were able to troubleshoot most problems we encountered quickly, there were a few challenges we spent some time overcoming in the later stages of building our Real Estate website - especially in the final part of this project where we had to implement personalization and AJAX.

Since we focused on implementing the “favorite” functionality for Members’ personalization aspect, we noticed several ways of doing it. Although the outcome at the end might’ve been the same, we had to try a few different methods to figure not only which was the easiest to build on top of what we already had but also which method was the most efficient out of all the different options. This took a lot of patience, trial, and error.

Another challenge we faced was keeping the filtering checkboxes checked upon submitting the filters on the second part of the project (PA2). Because the page would refresh when the “Submit” button was hit, it would wipe clean what was selected, and we could not figure out how to keep the checkboxes checked to remind the user what they have chosen for the filtering. This, however, was solved when we learned of AJAX for the next and final part of the project because it allowed the search to happen asynchronously.

While implementing AJAX for the filtering, we first tried adding AJAX to each filtering category (Price, Property Type, and City). We quickly found that the checkbox filtering form could not filter along with the search bar for city filtering at the same time when the AJAX function was used on the filter categories separately. We were able to solve this once we figured out that the AJAX function had to hold all of the checkbox form variables at once to work.

Self Reflection:

Lucy:
This project was a great learning experience since I was only familiar with the frontend aspect of design using HTML and CSS but have never touched the backend processes of a working website. Learning PHP and MySQL helped me better understand how databases work and interact with the web pages and how there are many aspects to consider when creating a fully functioning website. Security is one of the most important things I learned while doing this project as you have to consider what the best method to use is and have the best interest of the user in mind. We must encrypt the password not only when storing it but also when comparing it to the one in the database, which I never really thought of before. For this assignment, we used md5 to encrypt the password, but I have learned that this is, in fact, outdated, and there are better and safer methods available. Something like encryption choice could put users at risk, which is essential to keep in mind when creating a website that holds integrity.

Eric:
This was an enjoyable project since it was my first time building a back-end of a website. Building PHP and AJAX gave me precious experience regarding how the back end is made. I can use this experience to create other frameworks like MongoDB and React.JS, which is very similar. I can see myself building on top of this to become a Full-Stack front end developer in the future as my career.

Justin:
Overall, this project was exciting, as this was my first time being exposed to back-end programming. One of the aspects of back-end programming that I enjoyed was database design, especially ER diagrams. Working various jobs, whether part-time or co-op, gives me an appreciation for how complex large company and organization systems need to be designed to facilitate efficiency and ease of understanding. I can't even begin to imagine what the database for a large organization such as SFU might look like. Completing this course will make me a more organized and efficient designer.