Foreword:

The website for this project is self-hosted on my computer at home and accessible via port forwarding via my public IP. The following DNS entry has been made to facilitate the demonstration and marking of my project, through which my website can be accessed: http://hice.crabdance.com

All relevant code used in this project can be found publicly in my GitHub profile in the following repository: https://github.com/krohnusmelavea/hice

Not relevant to marking, but I wrote a cool python script which rebuilds my database and runs the 55 database schema creation scripts. It performs dependency inversion to determine the run order, so that all scripts are run in such a way that all their dependencies are run first. The file can be found at https://github.com/KrohnusMelavea/hice/blob/main/database/schema/build.py, in case the marker wishes to run this on his/her own computer.

Additionally, I write some utility scripts to startup all the relevant services (PHP, NginX, and MySQL in my case) automatically in a manner that duplicate processes won't be created by keeping track of their PID's in a json file. The number of times my cat has accidentally shut down my computer by walking all over my keyboard necessitated some helper scripts to facilitate a quick startup…

I tried my best to give examples of the formatting and architectural layout of my project via snippets in this document, yet my project contains 175 source files and 6780 lines of code, so I can hardly chuck them all in here.

Project:

For the sake of simplicity (to make the marker's life easier), I've decided to split every screen in my project into one of three categories:

1. List: Displays a collection of elements which get loaded from the webserver
2. Item: Displays a single item which gets loaded from the webserver
3. Ad-Hoc: Contains interactable elements which can make calls to the webserver. These will typically redirect the user to one of the other two types of screens.

If any feature in any particular screen is accessed which requires privileged information (i.e., information linked to a particular individual's personal account), an attempt will be made to refresh the current user's session.
This attempt can fail for one of two reasons:

- The user does not currently have a session (they're not logged in)
- The user's session expired (it's been more than 24 hours since they've last interacted with a privileged feature, configurable by DBA's)

An example of a privileged screen is the /homes page, which loads the list of houses a user owns.

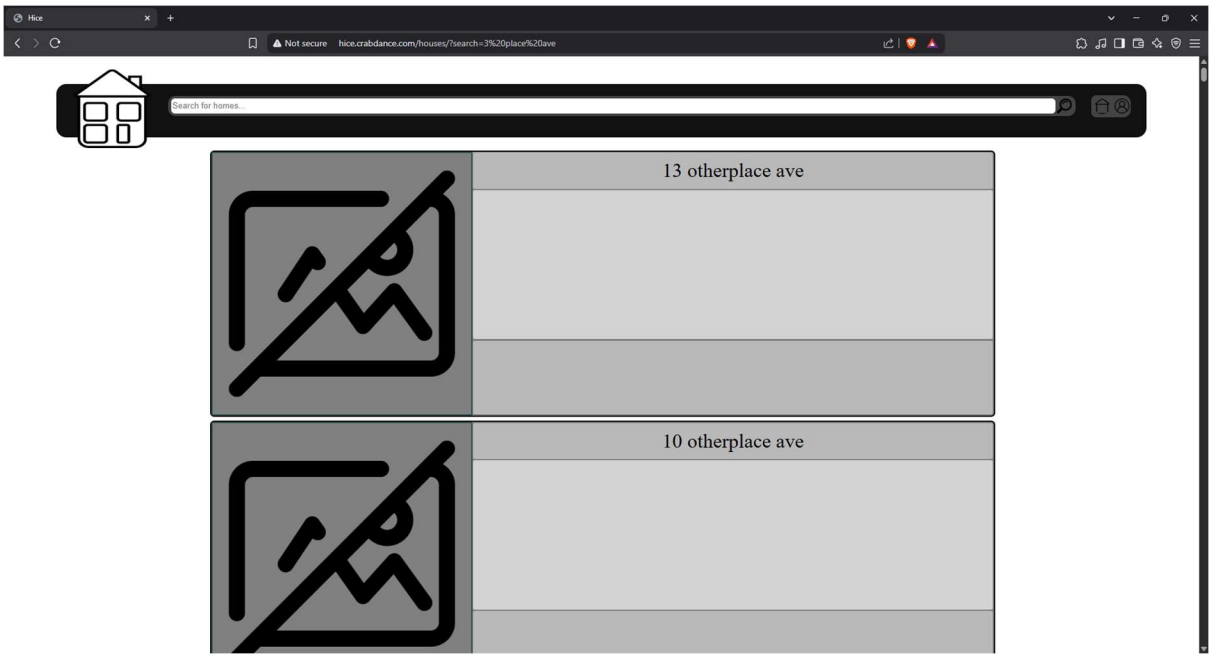
If a user attempts to access a privileged feature without having a valid session, they will be redirected to the login screen (GeeksForGeeks, 2025). Redirects throughout are prepended with a “/” to force domain-relative pathing (Fielding, et al., 1999).

When a logged-in user attempts to access the login or registration screen, they will be redirected to the account screen. Users can switch between the login and registration screen (UXWorld, 2024).

The pipeline for page generation per screen type works as follows:

1. List:
    a. Index
    b. Index Generator
    c. Database Call
    d. List Generator
    e. List Item Generator
2. Item:
    a. Index
    b. Index Generator
    c. Database Call
    d. Item Generator

An example of the aforementioned list page would be the /homes screen, of which a screenshot is attached below (pardon grey-scale, I’m colour blind, so I use brightness-contrast to better see components):



Example code snippets detailing the pipeline (for the /houses screen, in this case):

Session extension upon page load, if the user is logged in.

```
session_start();

if (isset($_SESSION["id"])) {
 $connection = connect_to_localhost();
 if (!$connection) {
  header("HTTP/1.1 500 Internal Server Error");
 }
 $result = validate_and_extend_session($connection, $_SESSION["id"]);
 $connection->close();
 if ($result != 0) {
  unset($_SESSION["id"]);
 }
}
```

Code for connecting to the database and returning the connection object:

```
require_once("models/database_credentials.php");

function connect_to_localhost() {
 $credentials = database_credentials::load_localhost();
 return mysqli_connect(
  $credentials->host,
  $credentials->user,
  $credentials->password,
  $credentials->database,
  $credentials->port
 );
}
```

I exclusively make use of “require_once” as supposed to “require” as my call stacks run pretty deep and the odds that I’ll end up “requiring” the same file multiple times along the way is very high. Additionally “require_once” will kill the evaluation of the script if the file could not be found (GeeksForGeeks, 2024).

PHP code which makes DB function call to extend the session:

```
function validate_and_extend_session(
 mysqli $connection,
 string $session_id
) {
 $statement = $connection->prepare("SELECT
`DB_Hice`.`FN_ValidateAndExtendSession`(?)");
 $statement->bind_param("s", $session_id);
 $statement->execute();
```

```
 $result = $statement->get_result();
 $data = $result->fetch_row();
 return $data[0];
}
```

MySQL function which goes and extends the session:

```
CREATE FUNCTION
 `DB_Hice`.`FN_ValidateAndExtendSession`
 (
  vSessionId BINARY(16)
 )
 RETURNS INT
 READS SQL DATA
BEGIN
 DECLARE dtInteraction DATETIME DEFAULT NULL;

 SET dtInteraction =
`DB_Hice`.`FN_GetMostRecentSessionInteractionDateTime`(vSessionId);

 IF (dtInteraction IS NULL)
 THEN
  RETURN 1; -- NO SESSION
 ELSE
  IF (DATE_ADD(dtInteraction, INTERVAL 1 DAY) < NOW())
  THEN
   DELETE FROM
    `DB_Hice`.`TB_Session`
   WHERE
    `DB_Hice`.`TB_Session`.`vId` = vSessionId;
   RETURN 2; -- SESSION EXPIRED
  ELSEc
   UPDATE
    `DB_Hice`.`TB_Session`
   SET
    `DB_Hice`.`TB_Session`.`dtInteraction` = NOW()
   WHERE
    `DB_Hice`.`TB_Session`.`vId` = vSessionId;
   RETURN 0; -- SESSION VALID
  END IF;
 END IF;
END;;
```

MySQL function which retrieves the most recent interaction time for a given session:

```
CREATE FUNCTION
 `DB_Hice`.`FN_GetMostRecentSessionInteractionDateTime`
 (
```

```
  vSessionId BINARY(16)
 )
 RETURNS DATETIME
 READS SQL DATA
BEGIN
 DECLARE dtInteraction DATETIME DEFAULT NULL;

 SELECT
  `DB_Hice`.`TB_Session`.`dtInteraction`
 INTO
  dtInteraction
 FROM
  `DB_Hice`.`TB_Session`
 WHERE
  `DB_Hice`.`TB_Session`.`vId` = vSessionId;

 RETURN dtInteraction;
END;
```

MySQL table creation script for the session table:

```
CREATE TABLE
 `DB_Hice`.`TB_Session`
(
 `vId`           BINARY(16)       NOT NULL DEFAULT(UUID_TO_BIN(UUID())),

 `vUserId`       BINARY(16)       NOT NULL,

 `uIP`           INTEGER UNSIGNED NOT NULL,

 `dtInteraction` DATETIME         NOT NULL DEFAULT(NOW()),

 CONSTRAINT
  `Session_PK`
 PRIMARY KEY
 (
  `vId`
 ),
 CONSTRAINT
  `Session_User_UK`
 UNIQUE
 (
  `vUserId`
 ),
 CONSTRAINT
  `Session_User_FK`
 FOREIGN KEY
 (
```

```
  `vUserId`
 )
 REFERENCES
  `DB_Hice`.`TB_User`
  (
   `vId`
  )
)
```

MySQL creation script for the user table:

```
CREATE TABLE
 `DB_Hice`.`TB_User`
(
 `vId`         BINARY(16)              NOT NULL DEFAULT(UUID_TO_BIN(UUID())),

 `sFirstName` NVARCHAR(64)            NOT NULL,
 `sLastName`  NVARCHAR(64)            NOT NULL,
 `sEmail`     VARCHAR(255)            NOT NULL,
 `sPassword`  VARCHAR(255)            NOT NULL,

 `cType`      ENUM('u', 'r', 's', 'a') NOT NULL DEFAULT('u'),

 `bIsDeleted` BOOLEAN                 NOT NULL DEFAULT(0),

 CONSTRAINT
  `User_PK`
 PRIMARY KEY
 (
  `vId`
 ),
 CONSTRAINT
  `User_Email_UK`
 UNIQUE
 (
  `sEmail`
 )
)
```

After session is extended, the index is generated in PHP via:

```
echo generate_page(
 generate_header(
  "Hice",
  [
   "/styles/navigation.css",
   "/styles/house_listings_item.css"
  ],
```

```
  [
   "/scripts/navigation.js",
   "/scripts/house_listings_item.js"
  ]
 ),
 generate_body(
  generate_navigation(),
  generate_index()
 )
);
```

The “generate_page” function takes in header and body html code. The header is generated via “generate_header”:

```
function generate_header(string $title, $styles = [], $scripts = []) {
 $template =
file_get_contents("$_SERVER[DOCUMENT_ROOT]/templates/header.html");
 return sprintf(
  $template,
  $title,
  generate_styles($styles),
  generate_scripts($scripts)
 );
}
```

The template “header.html” looks like so:

```
<!DOCTYPE html>

<html>
 %s

 %s
</html>
```

Where the first “%s” will be populated with the styles, and the second with the scripts.

In this function, the styles (css) and scripts (js) are linked in based on the two arrays based into this function in the index.
PHP code for generating the styles linking code:

```
function generate_styles(array $styles) {
 if (count($styles) == 0) {
  return "";
 }
 $template =
file_get_contents("$_SERVER[DOCUMENT_ROOT]/templates/style.html");
 $generated_styles = sprintf($template, $styles[0]);
 for ($i = 1; $i < count($styles); ++$i) {
  $generated_styles = $generated_styles . "\n" . sprintf($template,
$styles[$i]);
```

```
 }
 return $generated_styles;
}
```
Which is used in tandem with the following template:

```
<link rel="stylesheet" href="%s" />
```

PHP code for generating the scripts linking code:

```
function generate_scripts(array $scripts) {
 if (count($scripts) == 0) {
  return "";
 }
 $template =
file_get_contents("$_SERVER[DOCUMENT_ROOT]/templates/script.html");
 $generated_scripts = sprintf($template, $scripts[0]);
 for ($i = 1; $i < count($scripts); ++$i) {
  $generated_scripts = $generated_scripts . "\n" . sprintf($template,
$scripts[$i]);
 }
 return $generated_scripts;
}
```
Which is used in tandem with the following template:

```
<script src="%s"></script>
```

After the header is done, the body is generated via the “generate_navigation” and “generate_index” functions. The “generate_navigation” function, as well as the functions introduced previously for generating the header, are called on every screen.

The PHP script for generating the navigation page:

```
function generate_navigation() {
 $template =
file_get_contents("$_SERVER[DOCUMENT_ROOT]/templates/navigation.html");
 return sprintf($template, "/images/profile.png");
}
```
In tandem with the following html template:

```
<div id="navigation-bar-container">
 <a href="/">
  <img id="logo" src="/images/icon.png">
 </a>

 <div id="search-container">
  <input id="search-input" placeholder="Search for homes...">
  <div id="search-button" onclick="search_button_onclick()">
   <img id="search-button-magnifier" src="/images/magnifier.png">
```

```
  </div>
 </div>

 <div id="navigation-bar">
  <a href="/homes">
   <div id="homes">
    <img id="home-icon" src="/images/house-icon.png">
   </div>
  </a>

  <a href="/account">
   <div id="account">
    <img id="pfp" src="%s" />
   </div>
  </a>
 </div>
</div>
```

Which uses the following styling (css):

```
#navigation-bar-container {
 display: flex;
 position: relative;
 align-items: center;
 left: 4%;
 width: 92%;
}
#navigation-bar-container::before {
 content: '';
 position: absolute;
 width: 100%;
 height: 60%;
 left: 0%;
 top: 25%;
 z-index: -1;
 border-radius: 16px;
 background-color: #111111;
}
#logo {
 flex-grow: 0;
 width: 128px;
 height: 128px;
 margin-left: 25px;
 margin-top: 10px;
 margin-right: 25px;
}
#search-container {
 display: flex;
 position: relative;
 align-items: center;
```

```
 flex-grow: 1;
 height: 24pt;
 background-color: #444444;
 border-radius: 12px;
}
#search-input {
 flex-grow: 1;
 height: 16pt;
 border-radius: 6px;
 border: none;
 margin-left: 4pt;
 margin-right: 1pt;
}
#search-button {
 display: flex;
 flex-grow: 0;
 position: relative;
 justify-content: center;
 align-items: center;
 border-radius: 8px;
 margin-right: 5px;
 background-color: #444444;
 cursor: pointer;
}
#search-button-magnifier {
 width: 24px;
 height: 24px;
 align-items: center;
 margin: 1px;
}
#navigation-bar{
 display: flex;
 flex-grow: 0;
 position: relative;
 align-items: center;
 justify-content: space-between;
 margin-left: 25px;
 margin-right: 25px;
 background-color: #444444;
 border-radius: 12px;
}
#homes {
 display: flex;
 flex-grow: 0;
 position: relative;
 align-items: center;
 justify-content: center;
```

```
 border-radius: 8px;
 background-color: #444444;
 margin-right: 1px;
}
#home-icon {
 width: 25px;
 height: 25px;
 align-items: center;
 margin: 5px;
 border-radius: 50%;
}
#account {
 display: flex;
 flex-grow: 0;
 position: relative;
 align-items: center;
 justify-content: center;
 border-radius: 8px;
 background-color: #444444;
 margin-right: 1px;
}
#pfp {
 width: 24px;
 height: 24px;
 align-items: center;
 margin: 1px;
 border-radius: 50%;
}
```

And the following script (js):

```
function generate_house_listings_relative_url(search_text) {
 return "/houses?search=" + search_text;
}

function search_button_onclick() {
 const search_text = document
  .getElementById("search-input")
  .value;
 window.location.href = generate_house_listings_relative_url(search_text);
}
```

Regarding the actual list screen contents, the generate_index function (which differs for every page) is called. The houses' page's looks like so:

```
function generate_index() {

 $connection = connect_to_localhost();
```

```
 if (!$connection) {
  header("HTTP/1.1 500 Internal Server Error");
 }
 $house_listings = get_property_listings_by_filter($connection, new
house_listings_filter($_GET["search"]));
 $connection->close();

 $template = file_get_contents("$_SERVER[DOCUMENT_ROOT]/houses/index.html");
 return sprintf(
  $template,
  generate_house_listings($house_listings)
 );
}
```

PHP function for getting a list of property listings from the database:

```
require_once("models/house_listings_item.php");
require_once("filters/house_listings_filter.php");

function get_property_listings_by_filter(
 mysqli $connection,
 house_listings_filter $filter
) {
 $statement = $connection->prepare("CALL
`DB_Hice`.`SP_GetPropertyListingsByFilter`(?)");
 $statement->bind_param("s", $filter->search_phrase);
 $statement->execute();
 $result = $statement->get_result();
 $data = $result->fetch_all(MYSQLI_ASSOC);
 return house_listings_item::from_associative_array($data);
}
```

This calls the following MySQL procedure, which uses keyword matching to sort the listings in the homes screen:

```
CREATE PROCEDURE
 `DB_Hice`.`SP_GetPropertyListingsByFilter`
 (
  sSearchPhrase NVARCHAR(255)
 )
BEGIN
 SELECT
  `DB_Hice`.`TB_PropertyListing`.`vId`,
  `DB_Hice`.`TB_Property`.`sAddress`,
  `DB_Hice`.`TB_PropertyListing`.`uPriceCents`,
  `DB_Hice`.`TB_PropertyListing`.`sDescription`
 FROM
  `DB_Hice`.`TB_PropertyListing`
```

```
 INNER JOIN
  `DB_Hice`.`TB_Property` ON
   `DB_Hice`.`TB_Property`.`vId` =
`DB_Hice`.`TB_PropertyListing`.`vPropertyId`
 WHERE
  NOT `DB_Hice`.`TB_PropertyListing`.`bIsDeleted`
 ORDER BY
  `DB_Hice`.`FN_PercentageOfWordsInString`
  (
   sSearchPhrase,
   `DB_Hice`.`TB_Property`.`sAddress`
  )
  DESC;
END;;
```

This introduces the property listing and property tables, which looks as follows:

TB_Property:

```
CREATE TABLE
 `DB_Hice`.`TB_Property`
(
 `vId`             BINARY(16)       NOT NULL DEFAULT(UUID_TO_BIN(UUID())),

 `sAddress`        NVARCHAR(255)    NOT NULL,
 `uSurfaceAreaM2` INTEGER UNSIGNED     NULL,
 `uFloorCount`     INTEGER UNSIGNED NOT NULL DEFAULT(1),
 `uBedroomCount`   INTEGER UNSIGNED NOT NULL DEFAULT(1),
 `uBathroomCount` INTEGER UNSIGNED NOT NULL DEFAULT(1),
 `uRoomCount`      INTEGER UNSIGNED NOT NULL DEFAULT(1),
 `cPropertyType`   ENUM('a', 'h')   NOT NULL,

 `bIsDeleted`      BOOLEAN          NOT NULL DEFAULT(0),

 CONSTRAINT
  `Property_PK`
 PRIMARY KEY
 (
  `vId`
 )
)
```

Which has the following history table for auding purposes in case a user or realtor makes a mistake and their changes need to be undone by a DBA:

```
CREATE TABLE
 `DB_Hice`.`TB_PropertyHistory`
(
```

```
`vId`             BINARY(16)          NOT NULL DEFAULT(UUID_TO_BIN(UUID())),

`vPropertyId`     BINARY(16)          NOT NULL,

`sAddress`        NVARCHAR(255)       NOT NULL,
`uSurfaceAreaM2`  INTEGER UNSIGNED        NULL,
`uFloorCount`     INTEGER UNSIGNED    NOT NULL,
`uBedroomCount`   INTEGER UNSIGNED    NOT NULL,
`uBathroomCount`  INTEGER UNSIGNED    NOT NULL,
`uRoomCount`      INTEGER UNSIGNED    NOT NULL,
`cPropertyType`   ENUM('a', 'h')      NOT NULL,

`bIsDeleted`      BOOLEAN             NOT NULL DEFAULT(0),

`dtActionOn`      DATETIME            NOT NULL DEFAULT(NOW()),
`vActionBy`       BINARY(16)          NOT NULL,
`cActionType`     ENUM('c', 'u', 'd') NOT NULL DEFAULT('c'),

CONSTRAINT
 `PropertyHistory_PK`
PRIMARY KEY
(
 `vId`
),
CONSTRAINT
 `PropertyHistory_Property_FK`
FOREIGN KEY
(
 `vPropertyId`
)
REFERENCES
 `DB_Hice`.`TB_Property`
 (
  `vId`
 ),
CONSTRAINT
 `PropertyHistory_DatabaseUser_FK`
FOREIGN KEY
(
 `vActionBy`
)
REFERENCES
 `DB_Hice`.`TB_DatabaseUser`
 (
  `vId`
 )
)
```

This history table is populated on INSERT and UPDATE by the following triggers:

TG_PropertyInsert:

```
CREATE TRIGGER
 `DB_Hice`.`TG_PropertyInsert`
AFTER INSERT ON
 `DB_Hice`.`TB_Property`
FOR EACH ROW
 INSERT INTO
  `DB_Hice`.`TB_PropertyHistory`
  (
   `vPropertyId`,
   `sAddress`,
   `uSurfaceAreaM2`,
   `uFloorCount`,
   `uBedroomCount`,
   `uBathroomCount`,
   `uRoomCount`,
   `cPropertyType`,
   `vActionBy`
  )
 VALUE
  (
   NEW.`vId`,
   NEW.`sAddress`,
   NEW.`uSurfaceAreaM2`,
   NEW.`uFloorCount`,
   NEW.`uBedroomCount`,
   NEW.`uBathroomCount`,
   NEW.`uRoomCount`,
   NEW.`cPropertyType`,
   `DB_Hice`.`FN_GetCurrentDatabaseUserId`()
  )
```

TG_PropertyUpdate:

```
CREATE TRIGGER
 `DB_Hice`.`TG_PropertyUpdate`
AFTER UPDATE ON
 `DB_Hice`.`TB_Property`
FOR EACH ROW
 INSERT INTO
  `DB_Hice`.`TB_PropertyHistory`
  (
   `vPropertyId`,
   `sAddress`,
   `uSurfaceAreaM2`,
   `uFloorCount`,
   `uBedroomCount`,
```

```
  `uBathroomCount`,
  `uRoomCount`,
  `cPropertyType`,
  `bIsDeleted`,
  `vActionBy`,
  `vActionType`
 )
VALUE
 (
  NEW.`vId`,
  NEW.`sAddress`,
  NEW.`uSurfaceAreaM2`,
  NEW.`uFloorCount`,
  NEW.`uBedroomCount`,
  NEW.`uBathroomCount`,
  NEW.`uRoomCount`,
  NEW.`cPropertyType`,
  NEW.`bIsDeleted`,
  `DB_Hice`.`FN_GetCurrentDatabaseUserId`(),
  IF(NEW.`bIsDeleted` = 0, 'u', 'd')
 )
```

TB_PropertyListing:

```
CREATE TABLE
 `DB_Hice`.`TB_PropertyListing`
(
 `vId`          BINARY(16)          NOT NULL DEFAULT(UUID_TO_BIN(UUID())),

 `vPropertyId`  BINARY(16)          NOT NULL,

 `uPriceCents`  INTEGER UNSIGNED        NULL,
 `cSaleType`    ENUM('s', 'r', 'a') NOT NULL,

 `sDescription` NVARCHAR(4000)          NULL,

 `bIsDeleted`   BOOLEAN             NOT NULL DEFAULT(0),

 CONSTRAINT
  `PropertyListing_PK`
 PRIMARY KEY
 (
  `vId`
 ),
 CONSTRAINT
  `PropertyListing_Property_FK`
 FOREIGN KEY
 (
```

```
  `vPropertyId`
 )
 REFERENCES
  `DB_Hice`.`TB_Property`
  (
   `vId`
  )
)
```

TG_PropertyListingInsert:

```
CREATE TRIGGER
 `DB_Hice`.`TG_PropertyListingInsert`
AFTER INSERT ON
 `DB_Hice`.`TB_PropertyListing`
FOR EACH ROW
 INSERT INTO
  `DB_Hice`.`TB_PropertyListingHistory`
  (
   `vPropertyListingId`,
   `uPriceCents`,
   `cSaleType`,
   `vActionBy`
  )
 VALUE
  (
   NEW.`vId`,
   NEW.`uPriceCents`,
   NEW.`cSaleType`,
   `DB_Hice`.`FN_GetCurrentDatabaseUserId`()
  )
```

TG_PropertyListingUpdate:

```
CREATE TRIGGER
 `DB_Hice`.`TG_PropertyListingUpdate`
AFTER UPDATE ON
 `DB_Hice`.`TB_PropertyListing`
FOR EACH ROW
 INSERT INTO
  `DB_Hice`.`TB_PropertyListingHistory`
  (
   `vPropertyListingId`,
   `uPriceCents`,
   `cSaleType`,
   `bIsDeleted`,
   `vActionBy`,
   `cActionType`
```

```
  )
 VALUE
  (
   NEW.`vId`,
   NEW.`uPriceCents`,
   NEW.`cSaleType`,
   NEW.`bIsDeleted`,
   `DB_Hice`.`FN_GetCurrentDatabaseUserId`(),
   IF(NEW.`bIsDeleted` = 0, 'u', 'd')
  )
```

Back to the keyword searching, the MySQL function which actually rates a property listing based on how closely the search phrase matches is looks like so:

```
DELIMITER ;;

CREATE FUNCTION
 `DB_Hice`.`FN_PercentageOfWordsInString`
 (
  sWords  NVARCHAR(255),
  sString NVARCHAR(255)
 )
 RETURNS FLOAT
 DETERMINISTIC
BEGIN
 DECLARE uWordFoundCount INTEGER UNSIGNED DEFAULT 0;
 DECLARE uIndex          INTEGER UNSIGNED DEFAULT 0;
 DECLARE uTotal          INTEGER UNSIGNED DEFAULT LENGTH(sWords) -
LENGTH(REPLACE(sWords, ' ', '')) + 1;

 WHILE (uIndex < uTotal) DO
  IF LENGTH(sString) - LENGTH(REPLACE(sString,
SUBSTRING_INDEX(SUBSTRING_INDEX(sWords, ' ', uIndex + 1), ' ', -1), '')) != 0
THEN
   SET uWordFoundCount = uWordFoundCount + 1;
  END IF;
  SET uIndex = uIndex + 1;
 END WHILE;

 RETURN CAST(uWordFoundCount AS FLOAT) / CAST(uTotal AS FLOAT);
END;;

DELIMITER ;
```

After all's said and done and the list of property listings arrive back at the webserver, it calls the following function to populate houses template:

```
function generate_house_listings(array $house_listings_items) {
 $template =
file_get_contents("$_SERVER[DOCUMENT_ROOT]/templates/house_listings.html");
 if (count($house_listings_items) == 0) {
  return sprintf(
   $template,
   ""
  );
 }
 $generated_house_listing_items =
generate_house_listings_item($house_listings_items[0]);
 for ($i = 1; $i < count($house_listings_items); ++$i) {
  $generated_house_listing_items =
   $generated_house_listing_items .
   "\n" .
   generate_house_listings_item($house_listings_items[$i]);
 }
 return sprintf(
  $template,
  $generated_house_listing_items
 );
}
```

Which populates its template by iteratively calling the following function:

```
require_once("models/house_listings_item.php");
require_once("util/uuid_bytes_to_hex.php");

function generate_house_listings_item(house_listings_item
$house_listings_item) {
 $template =
file_get_contents("$_SERVER[DOCUMENT_ROOT]/templates/house_listings_item.html"
);
 $uuid_hex = uuid_string_to_hex($house_listings_item->id);
 return sprintf(
  $template,
  $uuid_hex,
  $uuid_hex,
  $house_listings_item->address,
  $house_listings_item->description
 );
}
```

Which makes use of the following function I wrote:

```
function uuid_string_to_hex(string $bytes) {
 $HEX_CODES = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b',
'c', 'd', 'e', 'f'];
```

```
 return
  $HEX_CODES[ord($bytes[ 0]) >> 4] . $HEX_CODES[ord($bytes[ 0]) & 15] .
  $HEX_CODES[ord($bytes[ 1]) >> 4] . $HEX_CODES[ord($bytes[ 1]) & 15] .
  $HEX_CODES[ord($bytes[ 2]) >> 4] . $HEX_CODES[ord($bytes[ 2]) & 15] .
  $HEX_CODES[ord($bytes[ 3]) >> 4] . $HEX_CODES[ord($bytes[ 3]) & 15] .
  $HEX_CODES[ord($bytes[ 4]) >> 4] . $HEX_CODES[ord($bytes[ 4]) & 15] .
  $HEX_CODES[ord($bytes[ 5]) >> 4] . $HEX_CODES[ord($bytes[ 5]) & 15] .
  $HEX_CODES[ord($bytes[ 6]) >> 4] . $HEX_CODES[ord($bytes[ 6]) & 15] .
  $HEX_CODES[ord($bytes[ 7]) >> 4] . $HEX_CODES[ord($bytes[ 7]) & 15] .
  $HEX_CODES[ord($bytes[ 8]) >> 4] . $HEX_CODES[ord($bytes[ 8]) & 15] .
  $HEX_CODES[ord($bytes[ 9]) >> 4] . $HEX_CODES[ord($bytes[ 9]) & 15] .
  $HEX_CODES[ord($bytes[10]) >> 4] . $HEX_CODES[ord($bytes[10]) & 15] .
  $HEX_CODES[ord($bytes[11]) >> 4] . $HEX_CODES[ord($bytes[11]) & 15] .
  $HEX_CODES[ord($bytes[12]) >> 4] . $HEX_CODES[ord($bytes[12]) & 15] .
  $HEX_CODES[ord($bytes[13]) >> 4] . $HEX_CODES[ord($bytes[13]) & 15] .
  $HEX_CODES[ord($bytes[14]) >> 4] . $HEX_CODES[ord($bytes[14]) & 15] .
  $HEX_CODES[ord($bytes[15]) >> 4] . $HEX_CODES[ord($bytes[15]) & 15];
}
```

To populate the following template:

```
<div class="house-listings-item" id="house-listing-%s"
onclick="house_listing_onclick('%s')">
 <div class="house-listing-image">
  <img class="house-listing-image-actual" src="/images/no_image.png">
 </div>
 <label class="house-listing-address">%s</label>
 <label class="house-listing-description">%s</label>
 <div class="house-listing-details">
 </div>
</div>
```

With the following CSS:

```
.house-listings-item {
 display: grid;

 width: 66%;
 margin-left: 17%;
 margin-bottom: 5pt;

 grid-template-columns: repeat(3, 33.33%);
 grid-template-rows: repeat(7, 14.29%);

 background-color: grey;
 border-radius: 5pt;
 border: 2pt black solid;
```

```
 cursor: pointer;
}

.house-listing-image {
 grid-column: 1 / span 1;
 grid-row: 1 / span 7;

 border-radius: 2pt;
 border: 2pt solid darkslategray;

 cursor: pointer;
}
.house-listing-image-actual {
 width: 100%;
 height: 100%;
 object-fit: contain;

 cursor: pointer;
}
.house-listing-address {
 display: flex;

 justify-content: center;
 align-items: center;

 font-size: 24pt;

 grid-column: 2 / span 2;
 grid-row: 1 / span 1;

 background-color: rgba(211, 211, 211, 0.685);
 margin: 1pt;
 border-radius: 2pt;

 cursor: pointer;
}
.house-listing-description {
 grid-column: 2 / span 2;
 grid-row: 2 / span 4;

 font-size: 16pt;

 background-color: lightgrey;
 margin: 1pt;
 padding: 5pt;
 border-radius: 2pt;
```

```
 cursor: pointer;
}
.house-listing-details {
 grid-column: 2 / span 2;
 grid-row: 6 / span 2;

 background-color: rgba(211, 211, 211, 0.685);
 margin: 1pt;
 border-radius: 2pt;

 cursor: pointer;
}
```

I think I've supplied enough PHP, SQL, and HTML code. In the interest of not bloating this project too much (and given that all this code can be found in the repository as well), I'll be omitting here the code for the login screen. Below however are the JavaScript functions I used on the client side to validate user registration input, the logic for which has been mirrored on the API (Raj, 2024) side as well to maintain UI-API validation parity for security purposes (Bitecode, 2024):

scripts/ORDINAL_CHARACTER.js

```
class ORDINAL_CHARACTER {
 static EXCLAMATION_MARK = 33;
 static FORWARD_SLASH    = 47;
 static NUM0             = 48;
 static NUM9             = 57;
 static COLON            = 58;
 static AT               = 64;
 static UPPER_A          = 65;
 static UPPER_Z          = 90;
 static OPEN_BRACKET     = 91;
 static BACKTICK         = 96;
 static LOWER_A          = 97;
 static LOWER_Z          = 122;
 static OPEN_BRACE       = 123;
 static TILDE            = 126;
};
```

scripts/is_special_character.js:

```
function is_special_character(ordinal_character) {
 return (
  (ordinal_character >= ORDINAL_CHARACTER.EXCLAMATION_MARK &&
ordinal_character <= ORDINAL_CHARACTER.FORWARD_SLASH) ||
  (ordinal_character >= ORDINAL_CHARACTER.COLON            &&
ordinal_character <= ORDINAL_CHARACTER.AT           ) ||
  (ordinal_character >= ORDINAL_CHARACTER.OPEN_BRACKET     &&
ordinal_character <= ORDINAL_CHARACTER.BACKTICK     ) ||
```

```
  (ordinal_character >= ORDINAL_CHARACTER.OPEN_BRACE       &&
ordinal_character <= ORDINAL_CHARACTER.TILDE        )
 );
}
```

scripts/is_lowercase_character.js:

```
function is_lowercase_character(ordinal_character) {
 return ordinal_character >= ORDINAL_CHARACTER.LOWER_A && ordinal_character <=
ORDINAL_CHARACTER.LOWER_Z;
}
```

scripts/is_uppercase_character.js:

```
function is_uppercase_character(ordinal_character) {
 return ordinal_character >= ORDINAL_CHARACTER.UPPER_A && ordinal_character <=
ORDINAL_CHARACTER.UPPER_Z;
}
```

scripts/is_numeric_character.js:

```
function is_numeric_character(ordinal_character) {
 return ordinal_character >= ORDINAL_CHARACTER.NUM0 && ordinal_character <=
ORDINAL_CHARACTER.NUM9;
}
```

scripts/does_password_meed_policy.js:

```
function does_password_meet_policy(password) {
 if (password.length < 16 || password.length > 255) {
  return false;
 }

 let index = 0;
 let contains_special_character = false;
 let contains_lowercase_character = false;
 let contains_uppercase_character = false;
 let contains_numeric_character = false;
 while (true) {
  const ordinal_character = password.charCodeAt(index);

  if (is_special_character(ordinal_character)) {
   contains_special_character = true;
  } else if (is_lowercase_character(ordinal_character)) {
   contains_lowercase_character = true;
  } else if (is_uppercase_character(ordinal_character)) {
   contains_uppercase_character = true;
  } else if (is_numeric_character(ordinal_character)) {
   contains_numeric_character = true;
  }

  ++index;
  if (contains_special_character && contains_lowercase_character &&
contains_uppercase_character && contains_numeric_character) {
```

```
   return true;
  }
  if (index === password.length) {
   return false;
  }
 }
}
```

scripts/validate_registration.js:

```
function validate_registration() {
 console.log("IM HERE BTW");

 const firstname = document.getElementById("registration-form-firstname-
input").value.trim();
 const lastname  = document.getElementById("registration-form-lastname-
input").value.trim();
 const username  = document.getElementById("registration-form-username-
input").value.trim();
 const password  = document.getElementById("registration-form-password-
input").value.trim();

 if (firstname.length === 0) {
  console.warn("Firstname Empty");
  return false;
 } else if (lastname.length === 0) {
  console.warn("Lastname Empty");
  return false;
 } else if (username.length === 0) {
  console.warn("Username Empty");
  return false;
 }

 const password_meets_policy = does_password_meet_policy(password);
 if (!password_meets_policy) {
  console.warn(`Password '${password}' did not meet policy.`);
 }

 return password_meets_policy;
}
```

References

Bitecode. (2024). *Yes, you need to duplicate your frontend business logic on the server*. Retrieved from Bitecode: https://www.bitecode.dev/p/yes-you-need-to-duplicate-your-frontend

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999, June). *Hypertext Transfer Protocol -- HTTP/1.1*. Retrieved from Datatracker: https://datatracker.ietf.org/doc/html/rfc2616#section-3.2

GeeksForGeeks. (2024). *Difference Between Require and Require Once*. Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/php/difference-between-require-and-require_once-in-php/

GeeksForGeeks. (2025). *How to redirect a user to the registration if he has not logged in* . Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/php/how-to-redirect-a-user-to-the-registration-if-he-has-not-logged-in/

Raj, P. (2024). *Data Validation in Your Backend: A Practical Guide*. Retrieved from DEV.TO: https://dev.to/starkprince/data-validation-in-your-backend-a-practical-guide-1kn6

UXWorld. (2024). *12 Best Practices for Sign-Up and Login Page Design*. Retrieved from UX Design World: https://uxdworld.com/12-best-practices-for-sign-up-and-login-page-design/#:~:text=7.%C2%A0Allow%20switching%20between%20sign%2Dup%20and%20login