Real Estate Website

By: David Benjamin Demers,
Rima Dagher, & Louis Chartier

PropExchange

Background

Realtor websites are revolutionizing the property market. The ability to digitalize listings and improve searchability is beneficial to clients and agents. PropExchange's purpose is to provide a transparent, user-friendly platform that has enhanced property search.

Overview of Features:

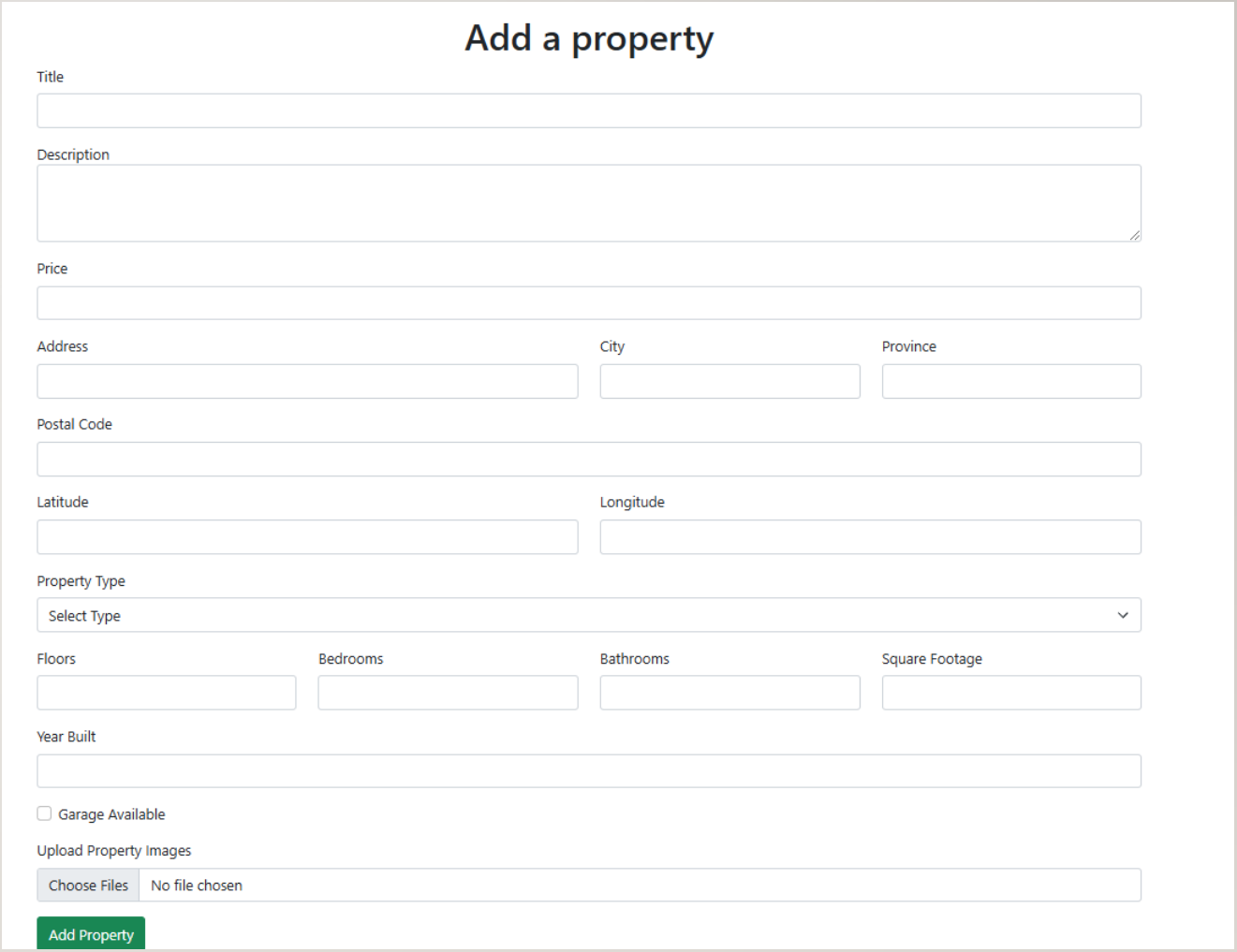
On PropExchange we allow users to search for properties and agents.

We show our listed properties on a map to see what properties are near you.

Users can list properties with a realtor agent and add images to the property.

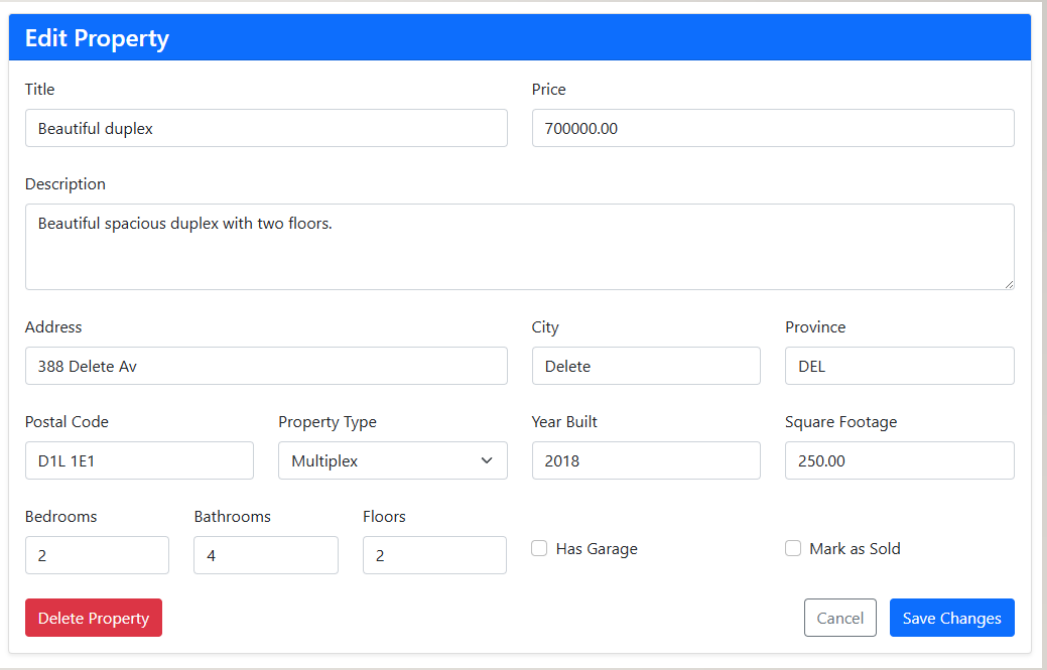
Client Features

- Clients can list a property
- Clients can add images to properties they already have listed
- Clients can change what realtor agent they want to manage their property.



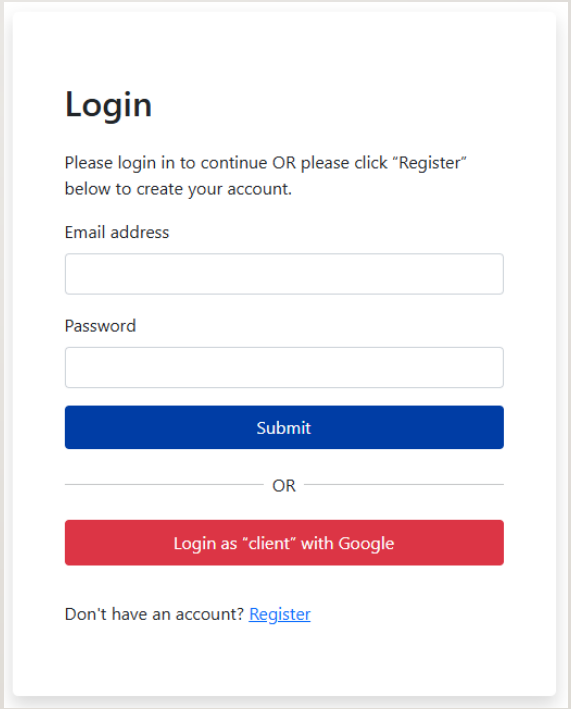
Agent Features

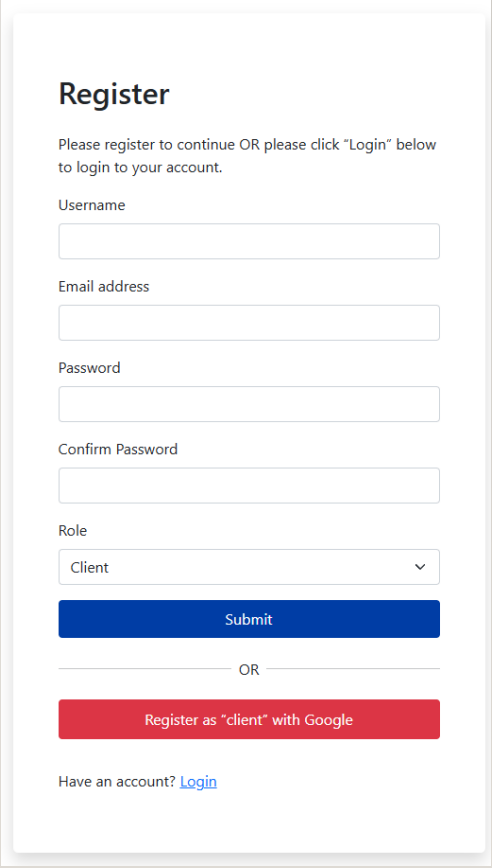
- Agents can edit a property's details.
- Agents can mark a property as sold.
- Agents can delete a property from our website.



Additional features

- Users can register accounts and log into them
- Users can search for properties by city name in the landing page or get a more detailed search in the "Find a Property" page
- Google login for client accounts only





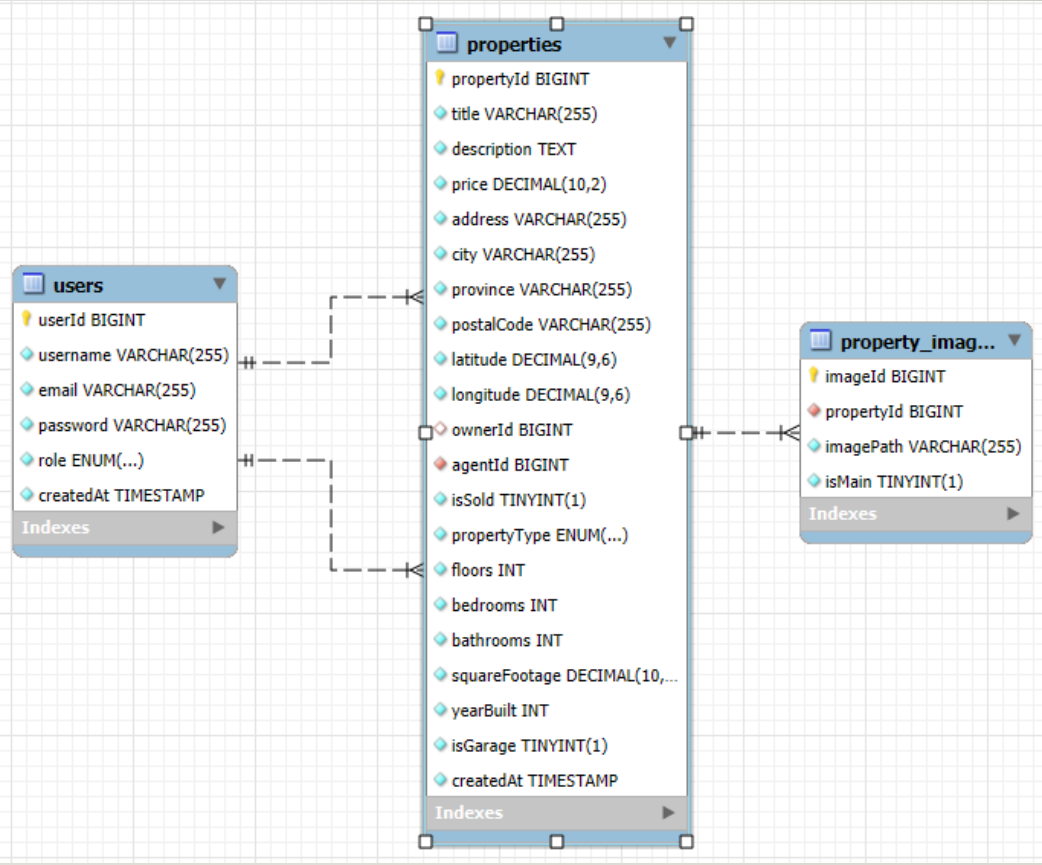
Google Login

Challenge: Create a fast and easy login using Google

Solutions: Used Laravel Socialite, configured OAuth credentials, redirect URIs, and services.php. Created dedicated routes & callback methods.

DATABASE STRUCTURE

- Users table, username, email, password and user role.
- Property table, information relevant to the property and foreign keys of the userId for the owner and the realtor agent.
- Property Images table, foreign key for the propertyId, as well as the image path.



Laravel and blade

Challenge: Create a project with laravel for the first time using blade for views.

Solutions: Watching laravel project toturial's and using laracasts to search for specific solutions.

Authentication & User Management

Challenge: Implement secure, registration and login.

Solution: Used Laravel's built in Auth to hash password, refresh sessions on login, and have role-based access control.

LARAVEL AUTHENTICATION

- In Laravel user authentication is made easy with its default Auth class;
  - Illuminate\Support\Facades\Auth
- Auth creates a session for the user when you call Auth::attempt($credentials) or Auth::login($user) which hard forces a login.
- Auth can also check the user in a session by calling Auth::user()->'colum name'

LARAVEL AUTH EXAMPLES

Login attempt

```php
if (Auth::attempt($credentials, $request->remember)) {
    $request->session()->regenerate();
    return redirect()->intended('/')->with('success', 'Login successful!');
}
```

Registering user and forcibly logging them in

```php
Auth::login($user);
return redirect()->route('welcome')->with('success', 'Registration successful!');
```

Checking with Auth if user is the agent of property

```php
if (Auth::id() !== $property->agentId) {
    return back()->with('error', 'Unauthorized Action');
}
```

Using Auth for middleware

```php
// Protected routes
Route::middleware('auth')->group(function () {
    Route::post('/logout', [UserController::class, 'logout'])->name('logout');
```

LARAVEL AUTH METHODS FOR VIEWS

```
<li class="nav-item">
    <span class="nav-link"> Hello, {{ Auth::user()->username }}</span>
</li>
```

```
@if(Auth::user()->role === 'client')
    <a href="{{ route('properties.create') }}" class="btn btn-primary">
        <i class="bi bi-plus-lg"></i> Add New Property
    </a>
@endif
```

```
@if(Auth::user()->role === 'client')
    <a href="{{ route('properties.agent.create', $property->propertyId) }}"
        class="btn btn-sm btn-outline-info">
        Change Agent
    </a>
@else
    <a href="{{ route('properties.edit', $property->propertyId) }}"
        class="btn btn-sm btn-outline-warning">
        Edit
    </a>
@endif
```

Google Maps

Challenge: Incorporate a dynamic google map that display all property listings.

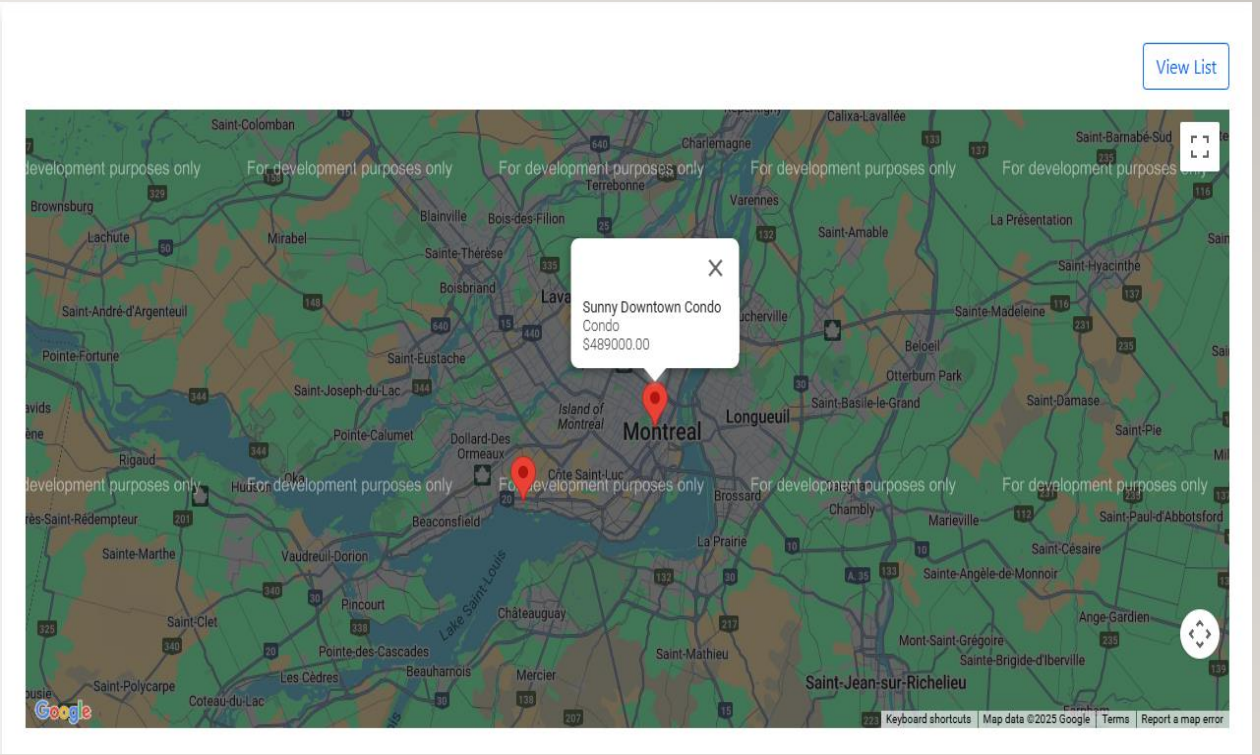
Solutions:

Fetched property data from the backend and passed it to the frontend.

Used Google Maps JavaScript API to render the map.

Placed markers using property latitude and longitude.

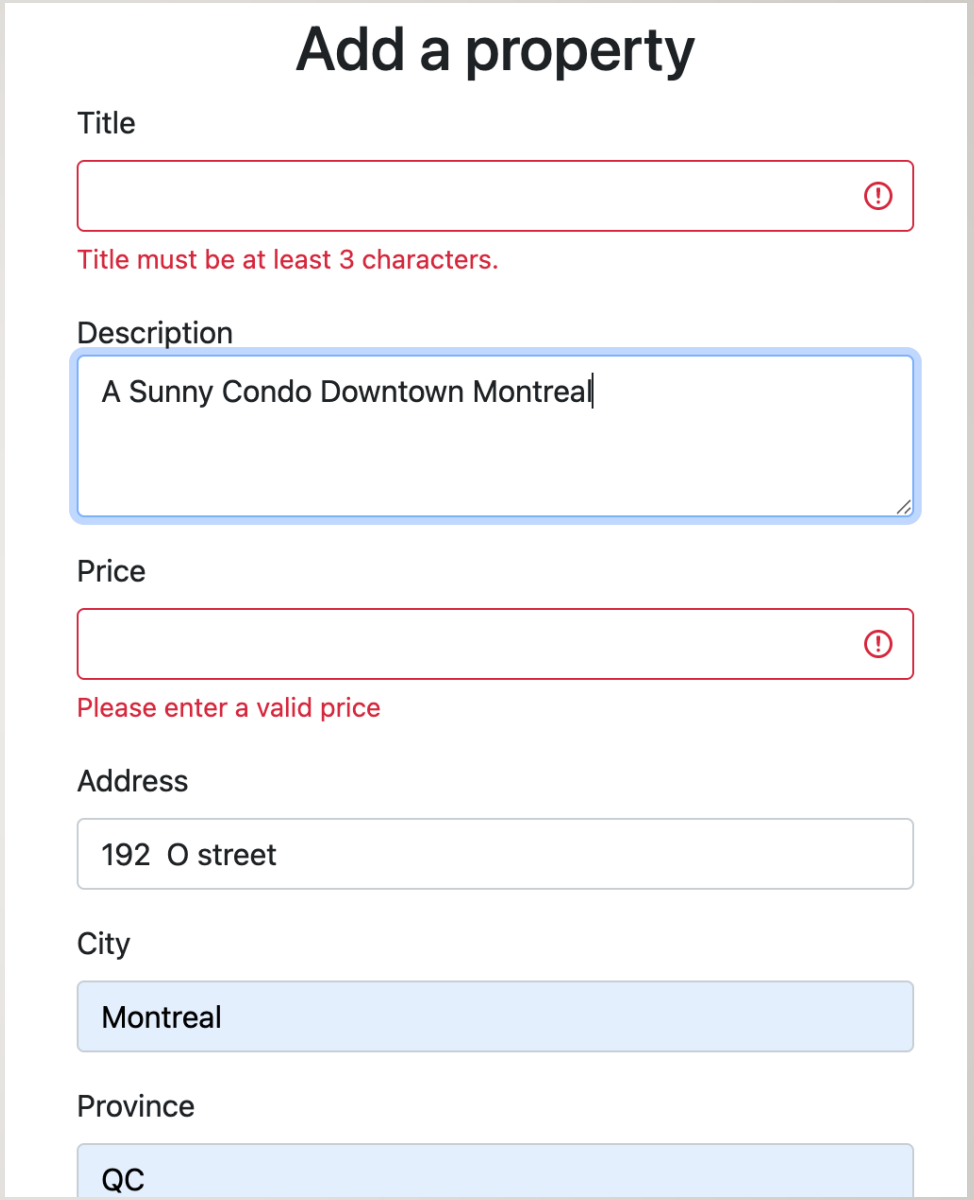
Added clickable infoWindow for extra details.



PROPERTY FILTERING

- Dynamic search form with multiple filters (city, title, type, price range, etc.)
- Real-time database querying based on user input
- An example query (WHERE city LIKE '%Montreal%' AND price >= 500000)
- Handles empty result states
- Integrate pagination for smooth UX

ADD PROPERTY PAGE



HOW WE INTEGRATED GOOGLE MAPS IN LARAVEL

We wanted users to visualize where each property is located using an interactive map on the /map-properties page.

How it works:

- Controller: I fetched all properties from the database and passed them to the view.
- Blade View: Used @json($properties) to pass the PHP data to javaScript.
- Google Maps Api: Added a map using a JS API.
- Markers: Placed for each property with an InfoWindow showing title, type and price.

HOW WE INTEGRATED GOOGLE MAPS IN LARAVEL

```
function initMap() {
  const map = new google.maps.Map(document.getElementById("map"), {
    zoom: 6,
    center: {
      lat: 45.4215,
      lng: -75.6993
    },
  });

  properties.forEach(property => {
    const marker = new google.maps.Marker({
      position: {
        lat: parseFloat(property.latitude),
        lng: parseFloat(property.longitude)
      },
      map: map,
      title: property.title
    });
```

(parameter) property: any

```
// get all properties for map
public function mapView()
{
    $properties = Property::all();
    return view('map-properties', compact('properties'));
}
```

```
Route::get('/map-properties', [PropertyController::class, 'mapView'
])->name('map-properties');
```

Future Work

- User ability to exchange messages with Agents
- Add more ways to login(Facebook, Apple, etc.)
- Change property images order and delete image

Summary

- Developed a realtor website with features that are similar to sites Realtor.ca & duPropiu.com.
- Implemented searching, listings, and image uploads.
- The results being a functioning platform where users can search for properties and agents as well as create property listings.

Workload

David

- All controllers
- Models and migrations
- Main routes
- My-properties page
- Edit-properties page
- Add-image page
- Change agent page
- Show-property page

Rima

- Add-property page
- Search-properties page
- Map-properties page
- Frontend- validation
- Integrated search with backend filters
- Page styling using Bootstrap

Louis

- Welcome page
- Login page
- Register page
- Google Login
- Search-agents page
- App Layout