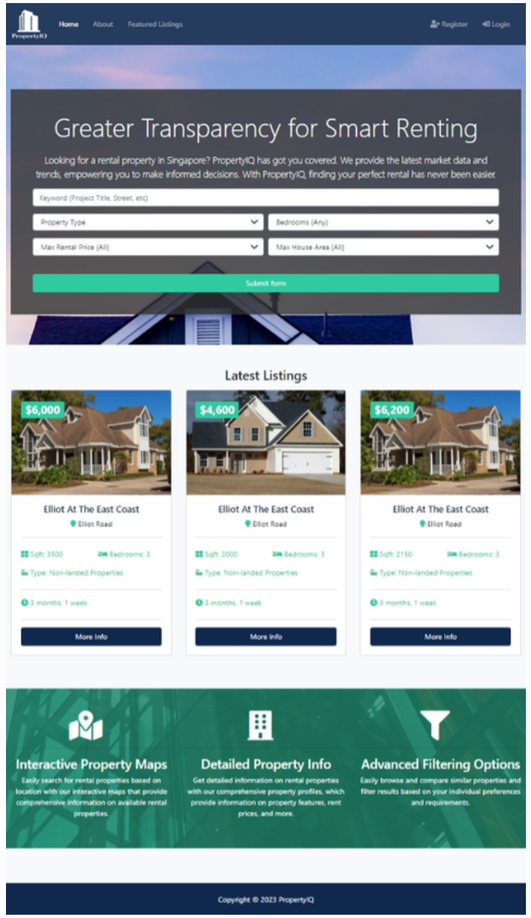
PropertyIQ

TEAM 3.0

TABLE OF CONTENTS



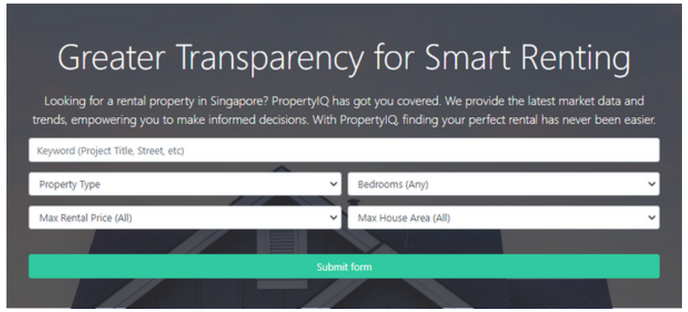
OVERVIEW

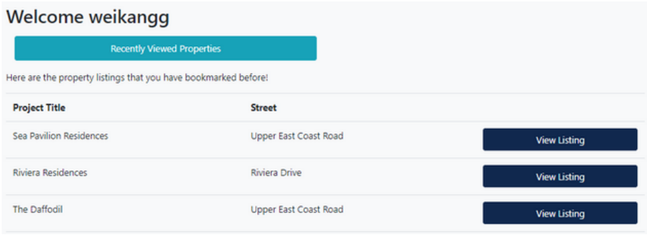
PropertyIQ is a web application designed to be the primary source for the latest information and trends in the Singapore housing rental market

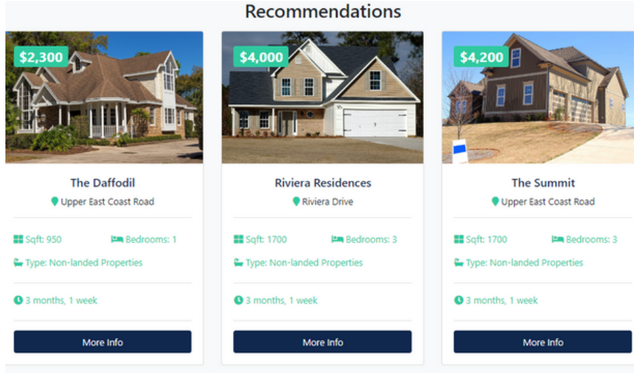
The platform's features include a variety of tools and resources to aid users in their rental search, such as interactive maps that allow users to search for rental properties based on location, and detailed property profiles that provide comprehensive information on available rental properties.

PROPERTYIQ

KEY FEATURES







1. Search Bar

Allow users to search for a property and is equipped with a filter function to further enhance their search query

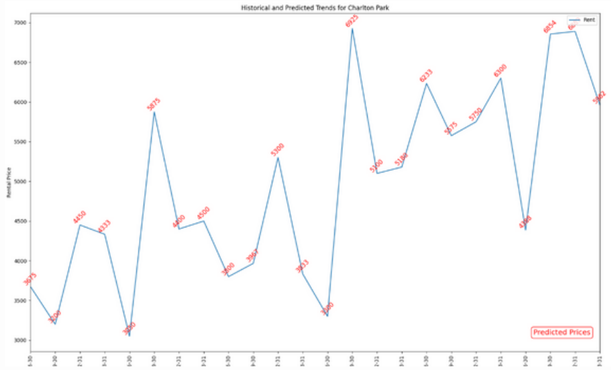
2. Bookmarks

Allow users to view, add and remove their bookmarked properties

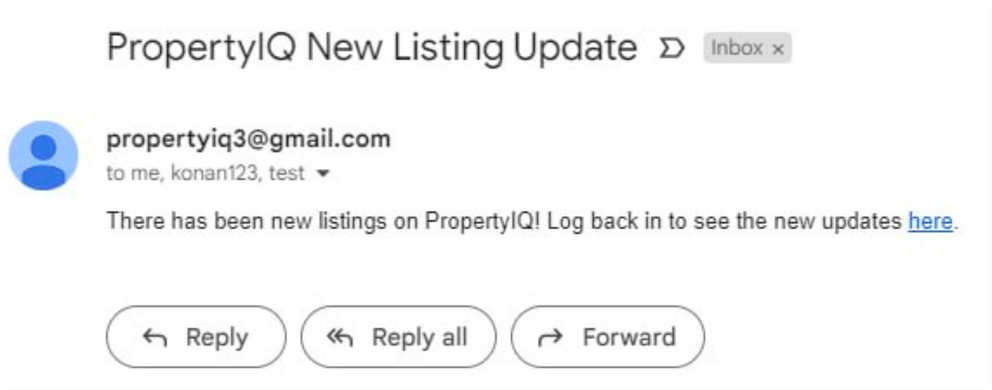
3. Recommendations

While viewing a certain property, several properties will be recommended based on factors like proximity and/or price

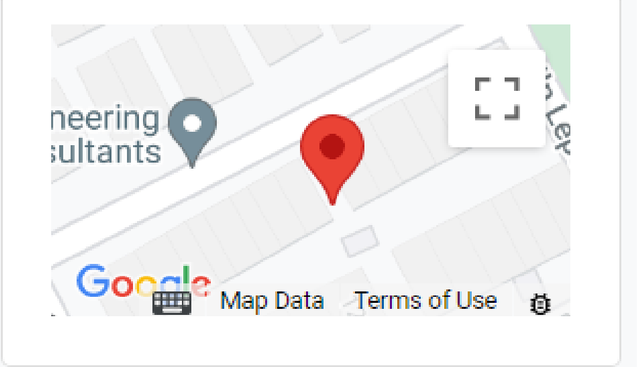
MORE TECH FEATURES



Historical Price Trends of Units

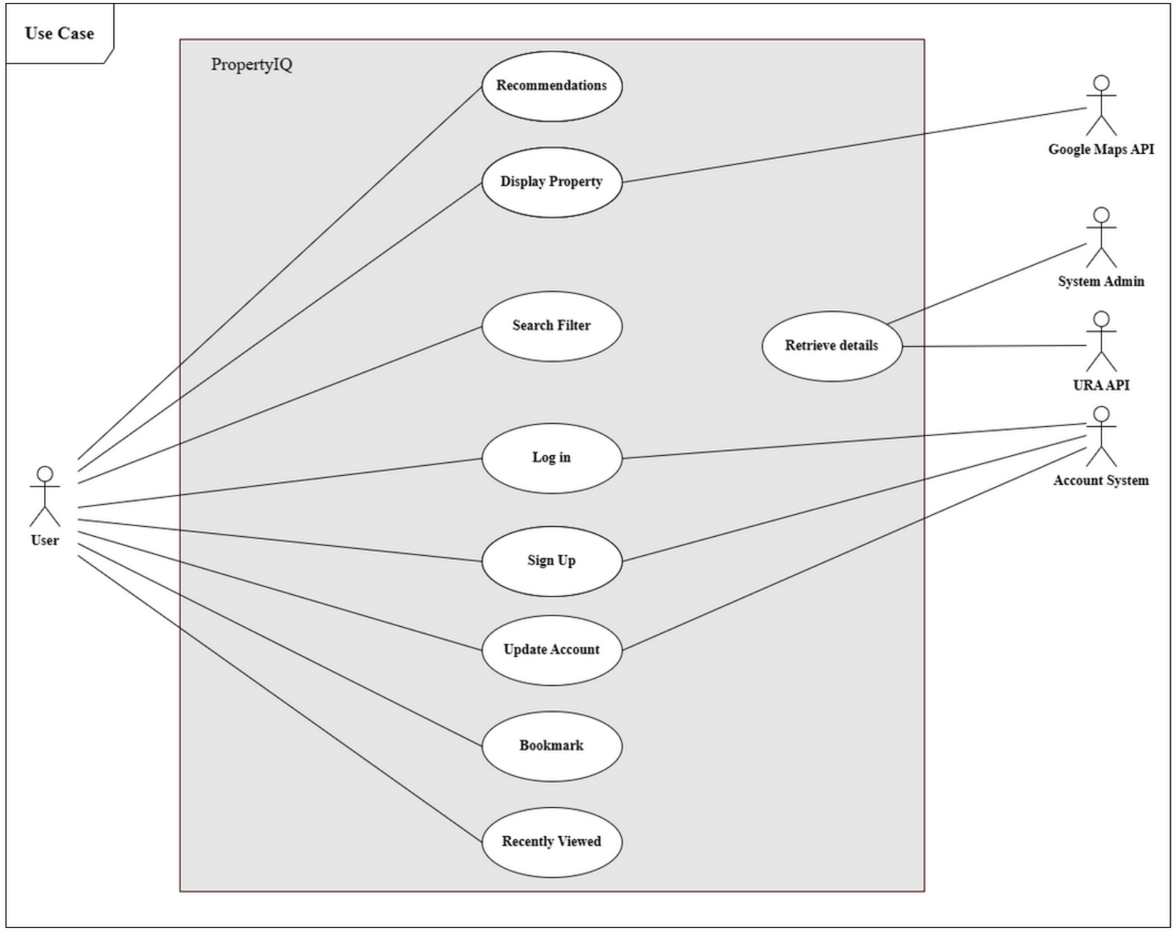


New Property updates sent to email



Location of Rented Property and recommendations shown on Google Maps

PROPERTYIQ



USE CASE DIAGRAM

TECH SOLUTION

APIs

URBAN REDEVELOPMENT AUTHORITY

To make Singapore a great city to live, work and play

Urban Redevelopment Authority (URA) API

Google Maps Platform

Google Maps API

PROPERTYIQ

TECH SOLUTION

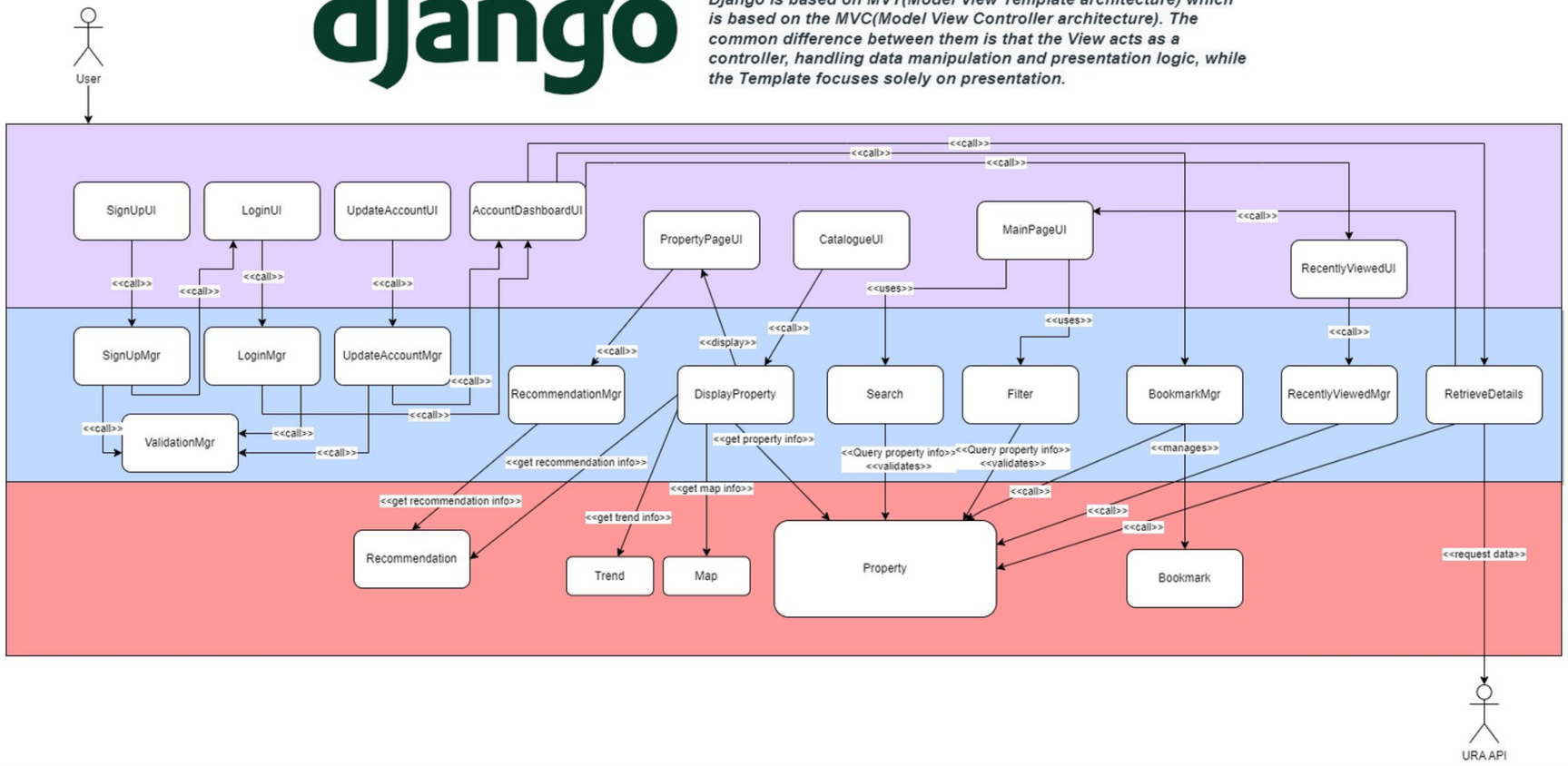
System Architecture

Django is based on MVT(Model View Template architecture) which is based on the MVC(Model View Controller architecture). The common difference between them is that the View acts as a controller, handling data manipulation and presentation logic, while the Template focuses solely on presentation.

Template

View

Model



Template: The Template is the user interface — what you see in your browser when you render a website and it refers to the presentation layer that manages the presentation logic in the framework.

Frontend: HTML, CSS, JS, Bootstrap, Jinja Templating Language

View: A view essentially controls the content to display and how to display it for the user as well as some special syntax describing how dynamic content will be inserted.

Backend: Python, Django

Model: The model is going to act as the interface of our data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database

Backend: PostgreSQL

PROPERTYIQ

TECH SOLUTION

Benefits of MVT Architecture

SEPARATION OF CONCERNS

MVT architecture separates data representation (model), user interface (template), and application logic (view).

Allows developers to work independently on different components without affecting each other.

FASTER DEVELOPMENT

MAINTAINABILITY & SCALABILITY

EXTENSIBILITY & REUSABILITY

TECH SOLUTION

Benefits of MVT Architecture

SEPARATION OF CONCERNS

MVT architecture separates data representation (model), user interface (template), and application logic (view).

Allows developers to work independently on different components without affecting each other.

FASTER DEVELOPMENT

Django's MVT architecture leverages the "Don't Repeat Yourself" principle.

Encourages code reusability and modularity.

MAINTAINABILITY & SCALABILITY

EXTENSIBILITY & REUSABILITY

TECH SOLUTION

Benefits of MVT Architecture

SEPARATION OF CONCERNS

MVT architecture separates data representation (model), user interface (template), and application logic (view).

Allows developers to work independently on different components without affecting each other.

FASTER DEVELOPMENT

Django's MVT architecture leverages the "Don't Repeat Yourself" principle.

Encourages code reusability and modularity.

MAINTAINABILITY & SCALABILITY

With a clear separation of concerns, it becomes easier to maintain the codebase as it grows.

The MVT architecture promotes a modular design that can be easily scaled as the application grows.

EXTENSIBILITY & REUSABILITY

PROPERTYIQ

TECH SOLUTION

Benefits of MVT Architecture

SEPARATION OF CONCERNS

MVT architecture separates data representation (model), user interface (template), and application logic (view).

Allows developers to work independently on different components without affecting each other.

FASTER DEVELOPMENT

Django's MVT architecture leverages the "Don't Repeat Yourself" principle.

Encourages code reusability and modularity.

MAINTAINABILITY & SCALABILITY

With a clear separation of concerns, it becomes easier to maintain the codebase as it grows.

The MVT architecture promotes a modular design that can be easily scaled as the application grows.

EXTENSIBILITY & REUSABILITY

The modular structure of MVT allows for the easy reuse of components like models, views, and templates across different parts of the application.

DEMO

STRATEGY PATTERN

```python
def search(request):
    queryset_list = Property.objects.order_by('-leaseDate')

    # Keywords
    if 'keywords' in request.GET:
        keywords = request.GET.get('keywords')
        if keywords:
            # Check that the title contains the keywords
            queryset_list = queryset_list.filter(Q(project_Title__icontains=keywords) | Q(street__icontains = keywords))


    # Property Type
    if 'property_type' in request.GET:
        property_type = request.GET.get('property_type')
        if property_type != '' and property_type != 'All':
            queryset_list = queryset_list.filter(propertyType__iexact=property_type) # Check that the property_type matches the city inputted

    # Bedrooms
    if 'bedrooms' in request.GET:
        bedrooms = request.GET.get('bedrooms')
        if bedrooms != '' and bedrooms != 'All':
            queryset_list = queryset_list.filter(bedrooms__lte=bedrooms) # Check that the no of bedrooms is less than or equal to the no of bedrooms

    # Price
    if 'price' in request.GET:
        price = request.GET.get('price')
        if price != '' and price != 'All':
            if price != '10001':
                queryset_list = queryset_list.filter(rent__lte=price) # Check that the rent is less than or equal to the no of price
            else:
                queryset_list = queryset_list.filter(rent__gt=10000) # Check that the rent is greater than 10,000

    # Area
    if 'area' in request.GET:
        area = request.GET.get('area')
        if area != 'All' and area != '':
            if area != '5001':
                queryset_list = queryset_list.filter(sqft__lte=int(area)) # Check that the area is less than or equal to the area inserted
            else:
                print('reached here')
                queryset_list = queryset_list.filter(sqft__gt=5000) # Check that the area is greater than 5000 sqft
```

BEFORE

OLD IMPLEMENTATION PROBLEMS

- Violates **Open-Closed Principle**
- If we introduce new Search criteria, current implementation has to be **modified**

STRATEGY PATTERN

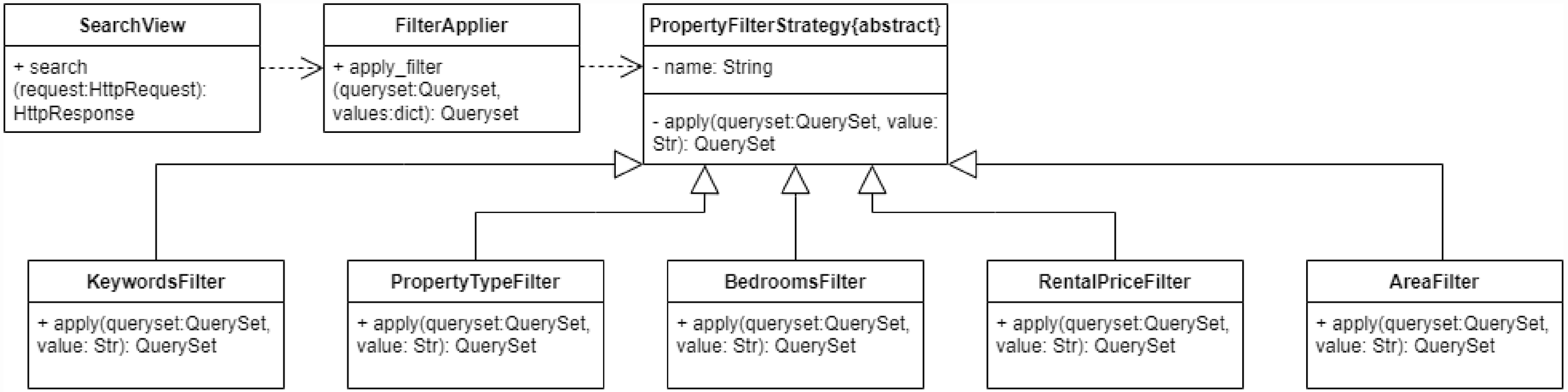
```python
class PropertyFilterStrategy:
    def __init__(self, name):
        self.name = name

    def apply(self, queryset, value):
        pass

class KeywordsFilter(PropertyFilterStrategy):
    def apply(self, queryset, value):
        if value:
            queryset = queryset.filter(Q(project_Title__icontains=value) | Q(street__icontains=value))
        return queryset

class PropertyTypeFilter(PropertyFilterStrategy):
    def apply(self, queryset, value):
        if value != '' and value != 'All':
            queryset = queryset.filter(propertyType__iexact=value)
        return queryset

class BedroomsFilter(PropertyFilterStrategy):
    def apply(self, queryset, value):
        if value != '' and value != 'All':
            queryset = queryset.filter(bedrooms__lte=value)
        return queryset

class PriceFilter(PropertyFilterStrategy):
    def apply(self, queryset, value):
        if value != '' and value != 'All':
            if value != '10001':
                queryset = queryset.filter(rent__lte=value)
            else:
                queryset = queryset.filter(rent__gt=10000)
        return queryset

class AreaFilter(PropertyFilterStrategy):
    def apply(self, queryset, value):
        if value != 'All' and value != '':
            if value != '5001':
                queryset = queryset.filter(sqft__lte=int(value))
            else:
                queryset = queryset.filter(sqft__gt=5000)
        return queryset
```

AFTER

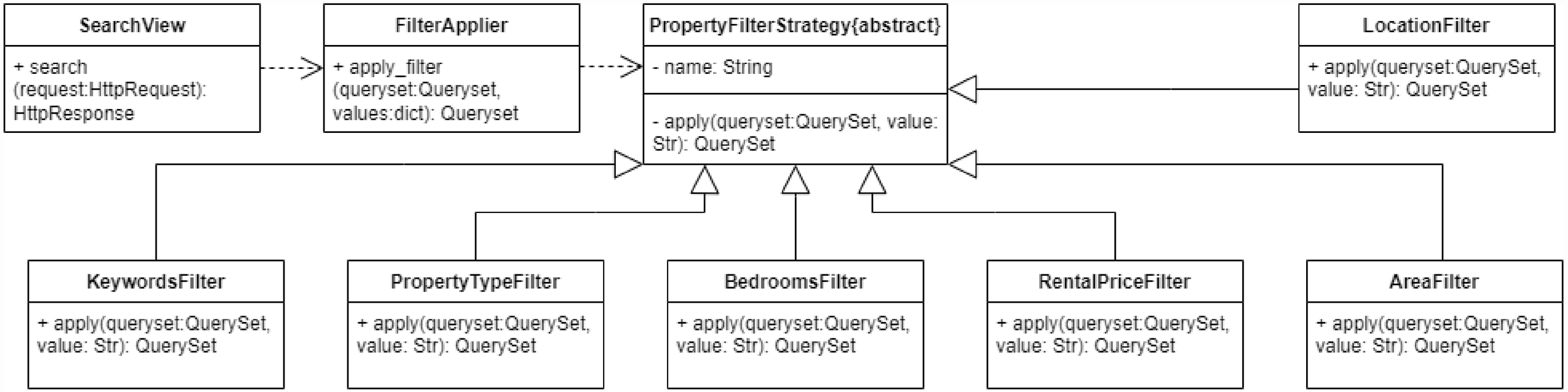
PURPOSE OF STRATEGY PATTERN

- A set of algorithms or objects that should be interchangeable.
- **Perfect** for our Problem!

STRATEGY PATTERN
CLASS DIAGRAM



STRATEGY PATTERN
CLASS DIAGRAM



OBSERVER PATTERN

PROBLEM & SOLUTION

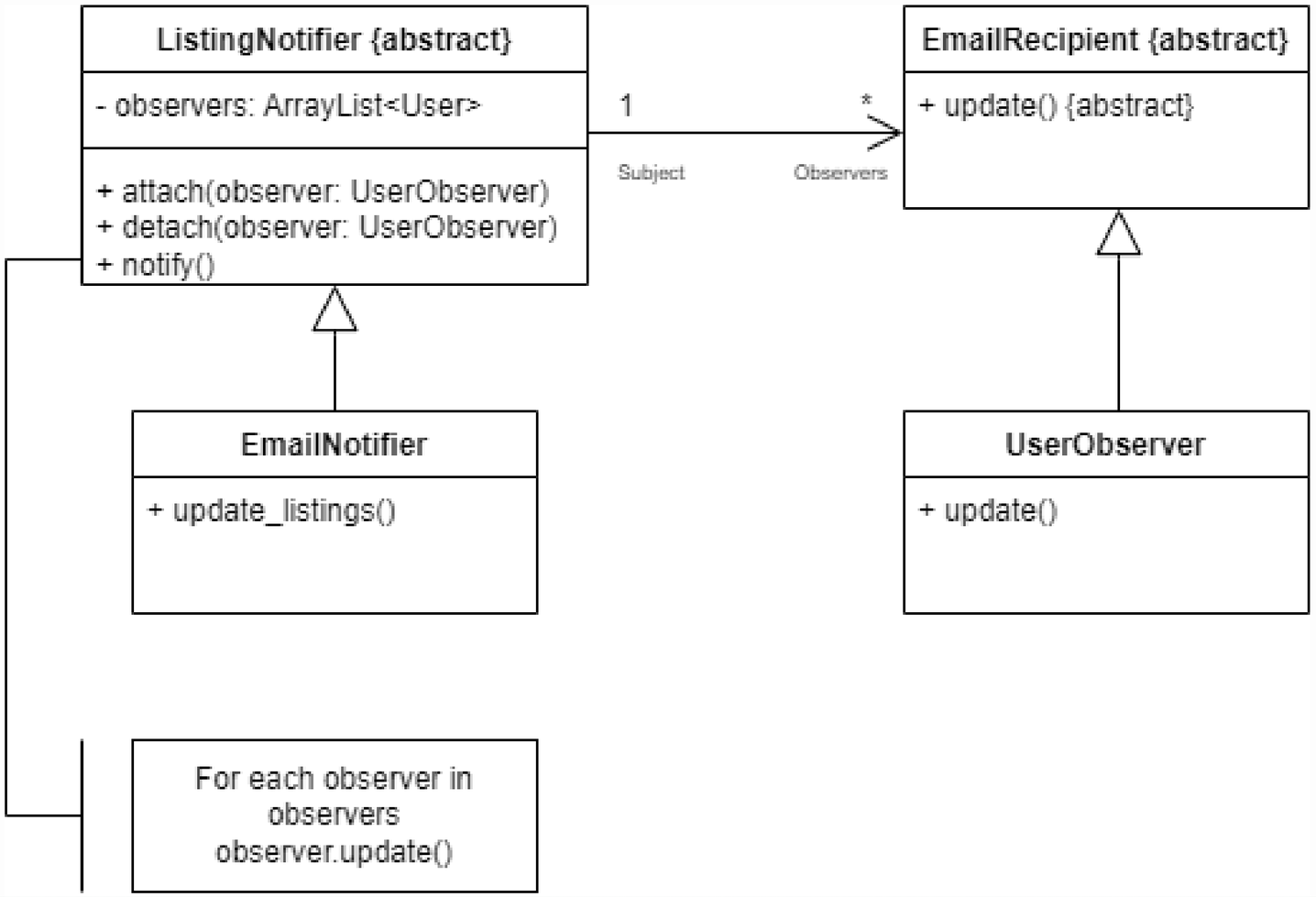
- We realised that users had to be made aware of any recent updates to the property listings.
- Therefore, reminders had to be sent to users whenever the property list was updated.

HOW?

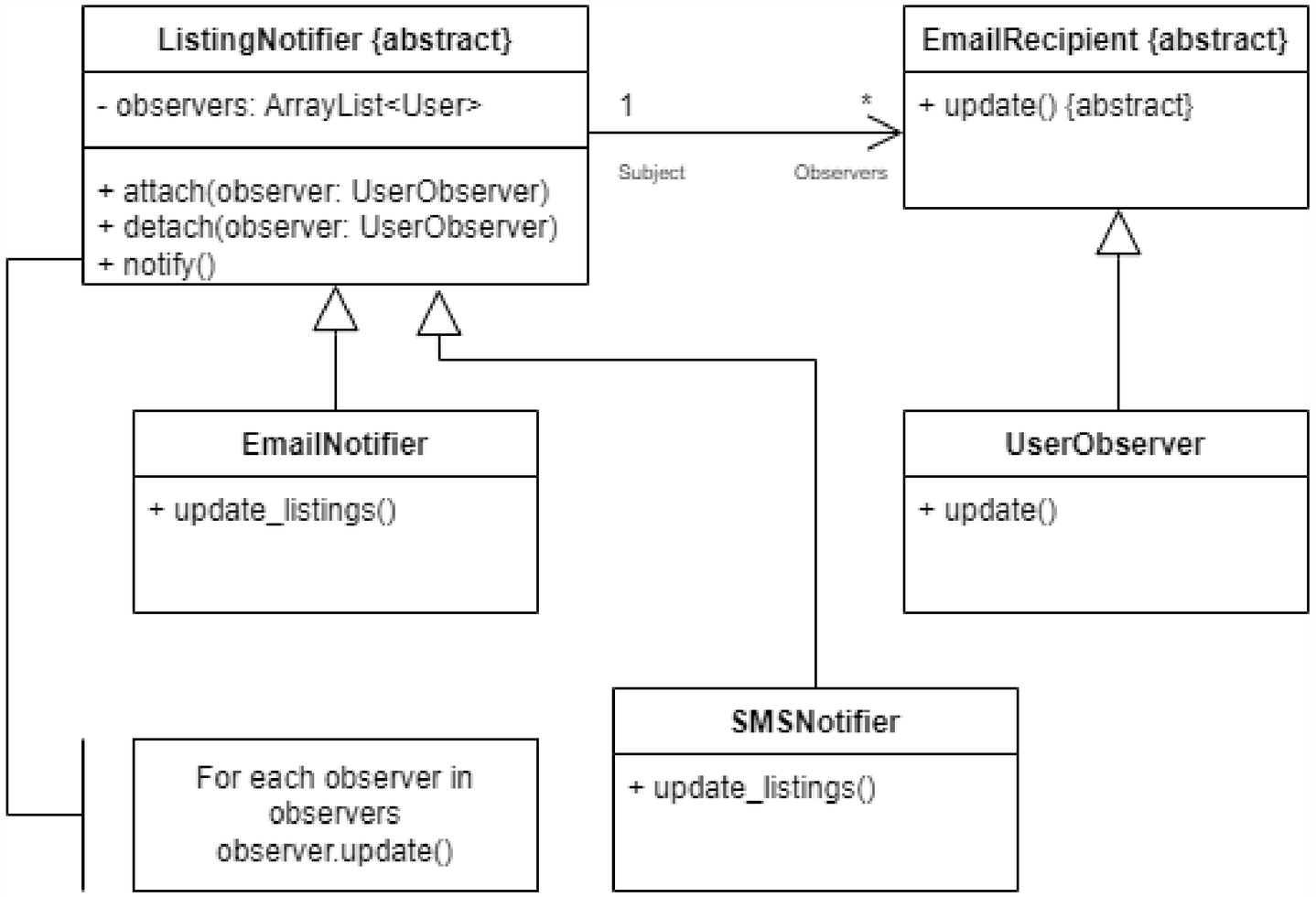
OBSERVER PATTERN

- Reduced Coupling in our code
- Increased extensibility
- Ease of integration of additional notification methods

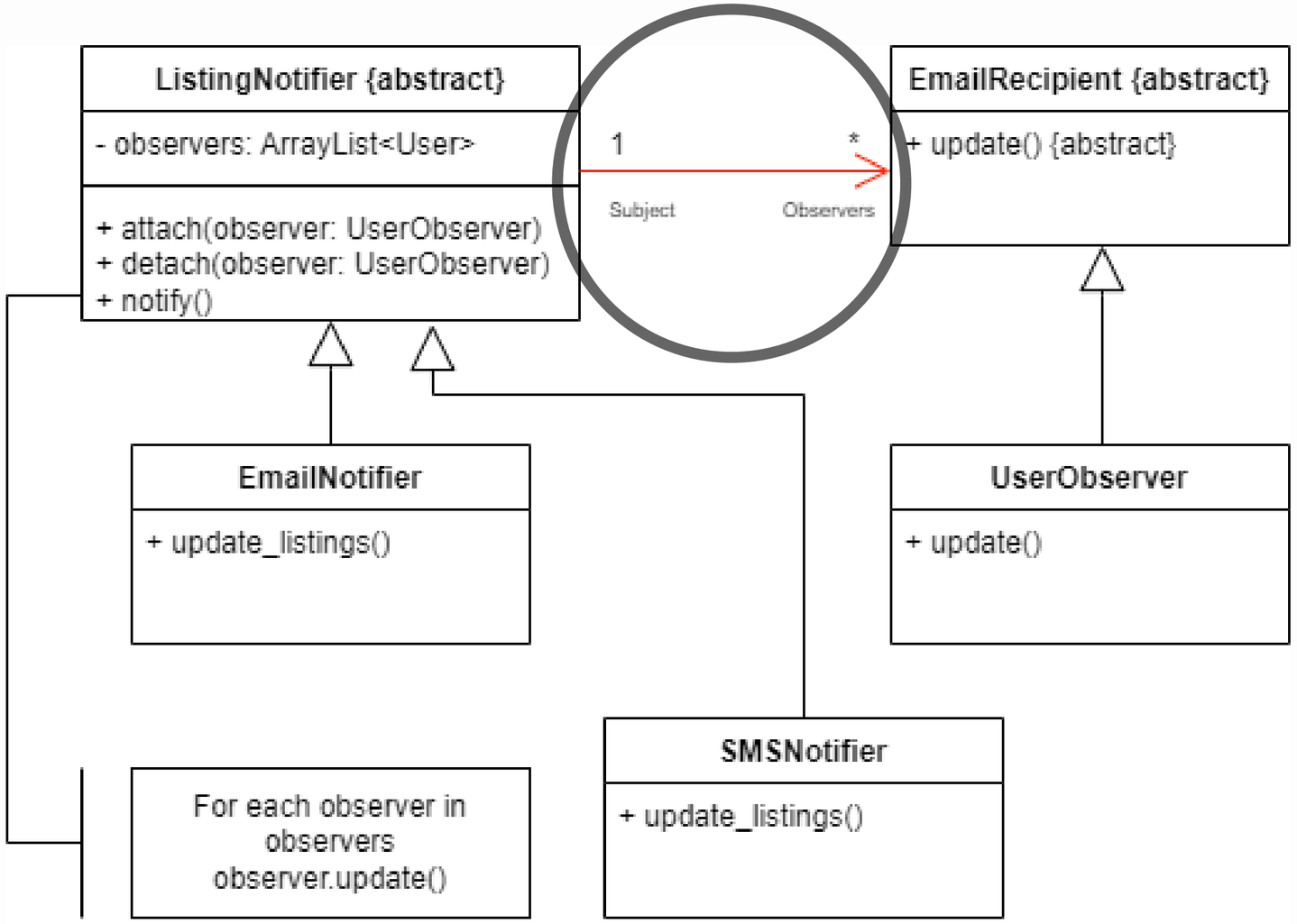
OBSERVER PATTERN



OBSERVER PATTERN



OBSERVER PATTERN

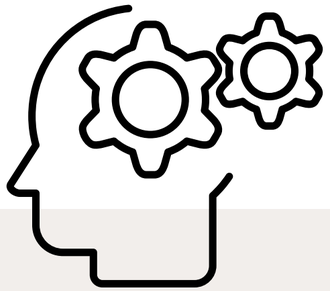


PROPERTYIQ

GOOD SOFTWARE PRACTICES
DESIGN PRINCIPLES

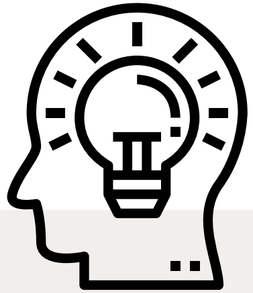
OPEN CLOSED PRINCIPLE

- Objects or entities should be open for extension but closed for modification
- Can be seen in the implementation of Search Function



SINGLE RESPONSIBILITY PRINCIPLE

- A class should only have one job
- This principle led us to design features such as, login, registration, and update functions where each function controlled only one part of our application's logic



PRINCIPLE OF LEAST KNOWLEDGE

- Functions should have minimal knowledge of other functions
- This allowed us to easily modify individual components and led to lower coupling

PROPERTYIQ

GOOD SOFTWARE PRACTICES
GOOD DESIGN

MAINTAINABILITY & SCALABILITY

Software system or component can be easily modified to correct faults or bugs

Abstraction and separation of classes facilitate this

EXTENSIBILITY & REUSABILITY

Addition of new capabilities or functionalities can be done easily by inheriting from the abstract base class instead of modifying the code

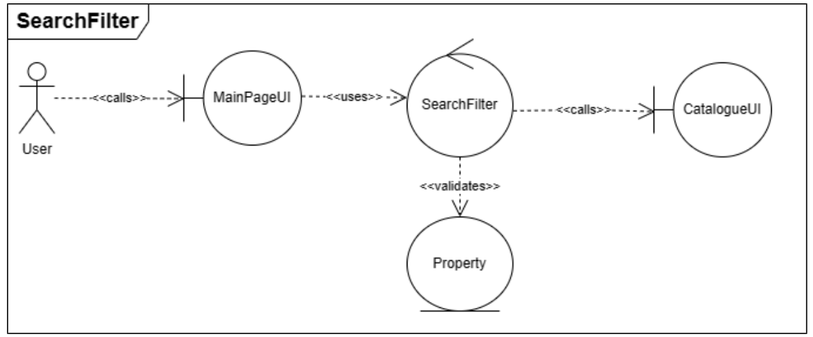
SEARCH FILTER FUNCTION

TRACEABILITY

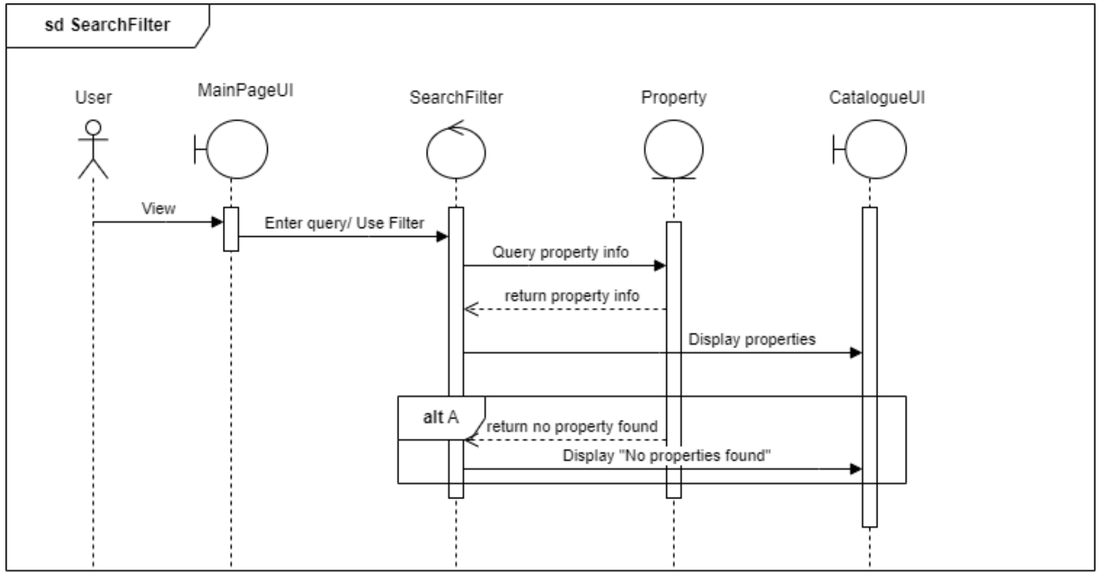
REQUIREMENTS

1. System must be able to accept search queries from user.
2. User could use filter functions to refine their queries if required.
3. System will display relevant information from user queries and filters used.

BOUNDARY CONTROL



SEQUENCE DIAGRAM



PROPERTYIQ

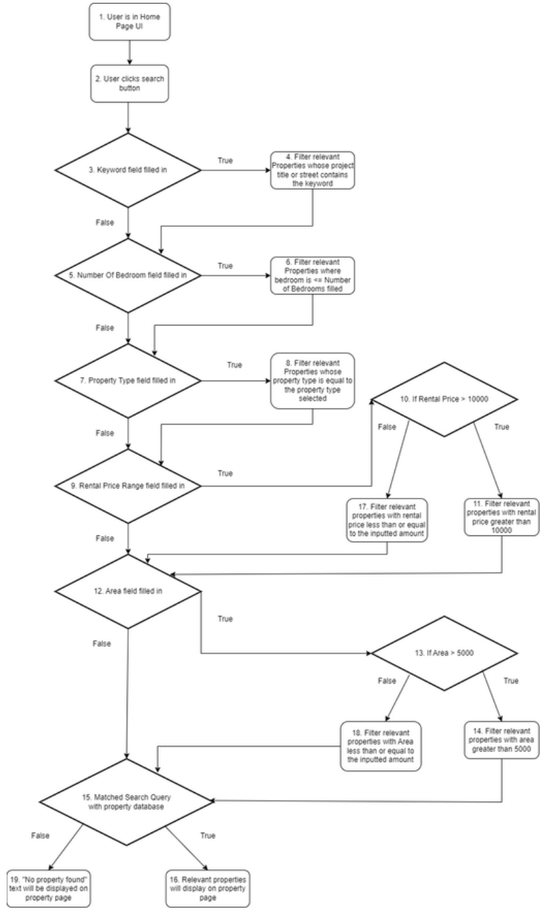
SEARCH FUNCTION

TRACEABILITY

IMPLEMENTATION

```python
def search(request):
    queryset_list = Property.objects.all()
    queryset_list = apply_filters(queryset_list, request.GET)
    queryset_list = queryset_list.order_by('-leaseDate')

    paginator = Paginator(queryset_list,6) # 6 property on each page
    page = request.GET.get('page')
    paged_listings = paginator.get_page(page)
    print(request.GET.get('keywords'))
    context = {
        'propertyType_choices': propertyType_Choices,
        'bedroom_choices': bedroom_choices,
        'price_choices': price_choices,
        'area_choices': area_choices,
        'listings': paged_listings,
        'values': request.GET
    }
    return render(request,'listings/search.html', context)
```

WHITE BOX TESTING



PROPERTYIQ

SEARCH FUNCTION

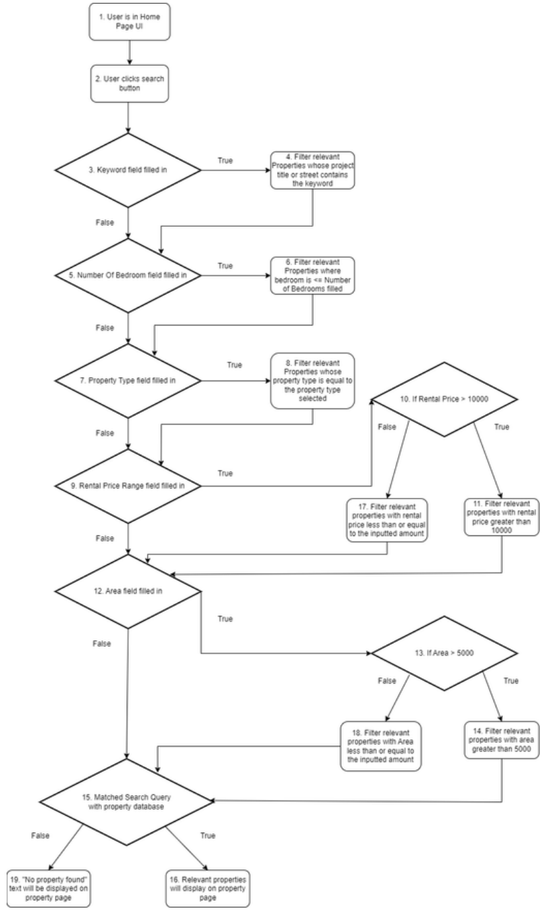
Test Case 1: All Search Fields Filled In
Test Path 1: 1-2-3-4-5-6-7-8-9-10-17-12-13-18-15-16 (Matched)

No.	Action	Input	Expected Output	Actual Output
1	- Enter keywords, select no of bedrooms, property type, rent, area - Click Submit Form	- Keyword input: 'paya lebar' - No. of Bedroom input: <= 10 - Property type input: Non-Landed - Rental Price Range input: <= $10,000 - Area Input: <= 5000 Sqft	List of properties matching search query will be displayed on the property page.	List of properties matching search query will be displayed on the property page.

Test Case 9: All Search Field Filled and no matches
Test Path 9: 1-2-3-4-5-6-7-8-9-10-17-12-13-18-15-19 (Not Matched)

No.	Action	Input	Expected Output	Actual Output
9	- Enter Keyword, select no of bedrooms, property type, rent, area - Click Submit Form	- Keyword input: 'qweqwe' - No. of Bedroom input: <= 10 - Property type input: Non-Landed - Rental Price Range input: <= $10,000 - Area Input: <= 5000 Sqft	Prompt showing "No property found" displayed on property page.	Prompt showing "No property found" displayed on property page.

WHITE BOX TESTING



PROPERTYIQ

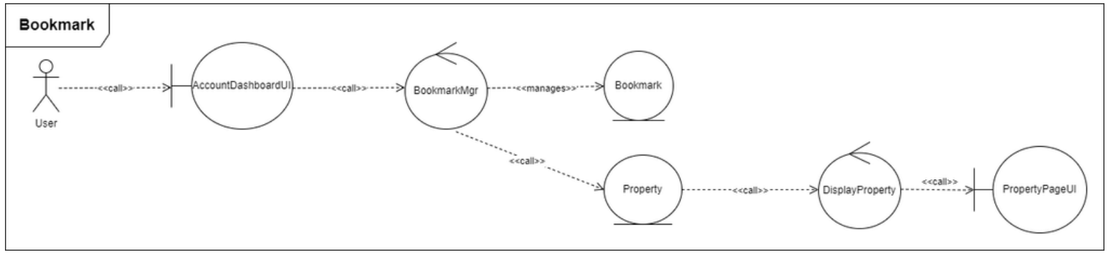
BOOKMARK FUNCTION

TRACEABILITY

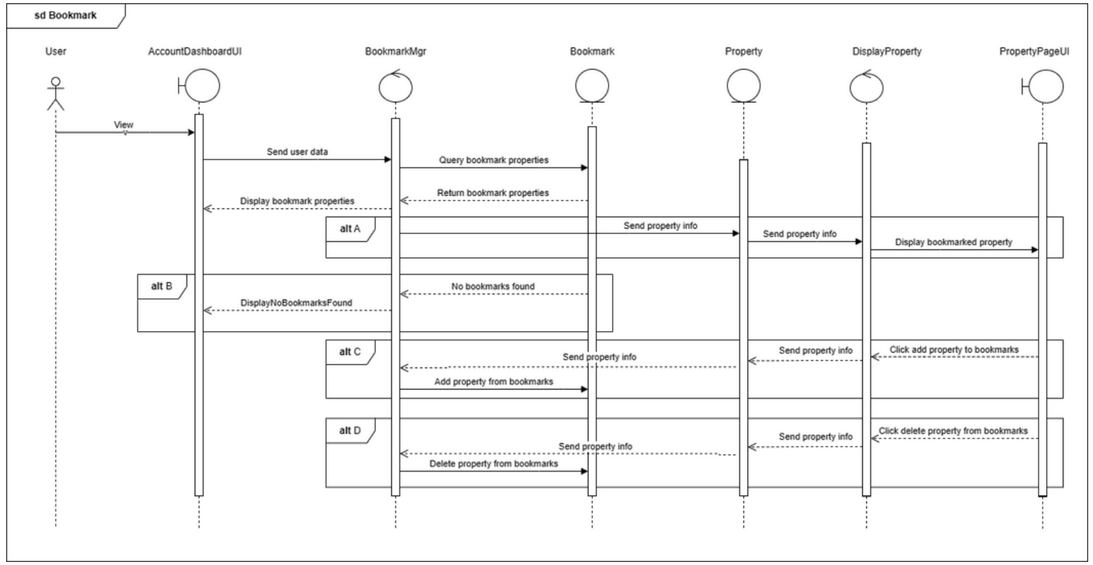
REQUIREMENTS

1. System must allow the user to add or remove bookmarks.
2. System will store the users bookmarks in the database.
3. System must be able to display the users' bookmarked properties.

BOUNDARY CONTROL



SEQUENCE DIAGRAM



PROPERTYIQ

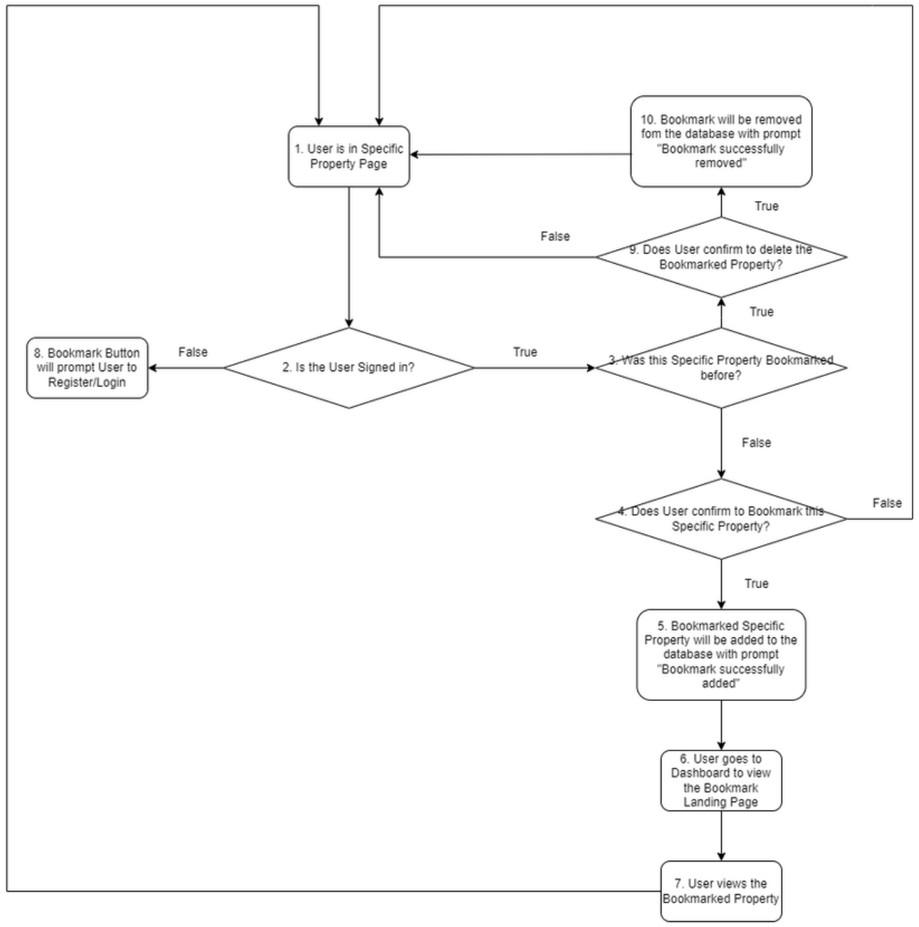
BOOKMARK FUNCTION

TRACEABILITY

IMPLEMENTATION

```python
def bookmarks(request, listing_id):
    property = get_object_or_404(Property, pk=listing_id)
    bookmark, created = Bookmark.objects.get_or_create(user=request.user, property=property)

    if created:
        messages.success(request, "Bookmark successfully added!")
    else:
        bookmark.delete()
        messages.success(request, "Bookmark successfully removed!")

    return redirect('listing', listing_id)
```

WHITE BOX TESTING



PROPERTYIQ

BOOKMARK FUNCTION

Test Case 1: User signed in + Property not bookmarked + User confirms bookmark
Test Path 1: 1-2-3-4-5-6-7-1

No.	Prerequisite	Action	Expected Output	Actual Output
1	User is signed in, Property is not bookmarked	User clicks on confirm bookmark button	Prompt for "Bookmark successfully added" is shown. User is redirected to a specific property landing page.	Prompt for "Bookmark successfully added" is shown. User is redirected to a specific property landing page.

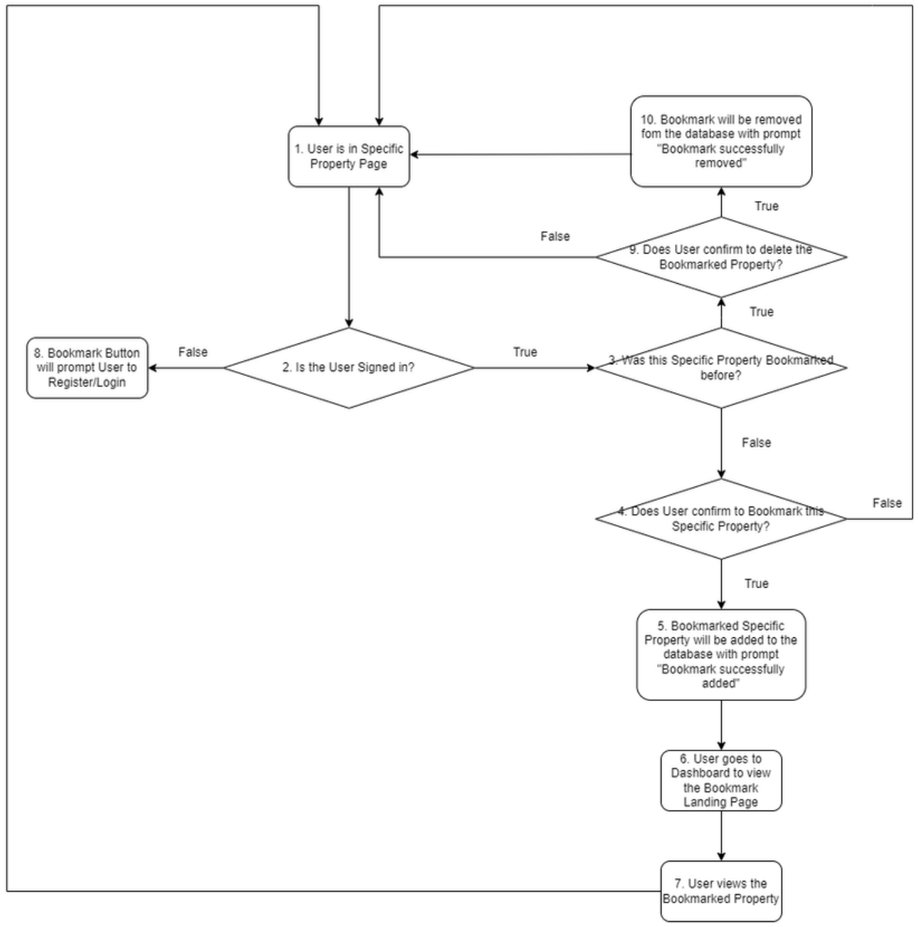
Test Case 3: User signed in + Property not bookmarked + User does not bookmark bookmark
Test Path 3: 1-2-3-4-1

No.	Prerequisite	Action	Expected Output	Actual Output
3	User is signed in, property is not bookmarked before	User clicks on cancel bookmark button	User will be redirected to specific property landing page	User will be redirected to specific property landing page

Test Case 4: User signed in + Property bookmarked + User removes bookmark
Test Path 4: 1-2-3-9-10-1

No.	Prerequisite	Action	Expected Output	Actual Output
4	User is signed in, Property is bookmarked.	User clicks on delete bookmark button and confirms	"Bookmark successfully removed" prompt shown. User is redirected to specific property landing page	"Bookmark successfully removed" prompt shown. User is redirected to specific property landing page

WHITE BOX TESTING



THANK YOU FOR YOUR ATTENTION