TIC2601

Database and Web Applications

AY 2021/2022 Semester 1

Group Project

Team 1

Clifton Kor Way Feng - A0227081L

Denny Wongso - A0227041U

Muhammad Naufal Dusan Urosevic – A0227505J

Toh Jian Feng, Clarence - A0226995L

Yeoh Amy - A0227095B

3rd November 2021

Table of Contents

1. Roofy

Roofy is a property online system that aims to provide a medium for the buying and renting of properties. It is a platform that allows users to search and browse the properties that for sale and for rent.

1.1. Search Features

Roofy has features with search functions that allows users to search by various criterion. Users can search for things such as titles, preferred addresses, residential types, range of prices, number of rooms and more.

In addition, most recent search queries used by users can be stored in the frontend for easier accessibility and convenience. This is done so when users either types or selects a specific search term and enters the search button in which the results are automatically populated, and search query used would be subsequently saved in the search bar.

1.2. Registered Users and Sellers

Both registered users and sellers can access special features that unregistered users are unable to. For instance, only registered users can view seller's contacts for properties of interest. Another feature provided to both registered users and sellers is the ability to view and edit their profile details even after registering for an account.

1.3. Bookmark Feature

Registered users would also be able to bookmark all their favourite listings. All listings that have been bookmarked will be listed on the bookmark page which can be accessed at any point of time. Apart from that, registered users can remove bookmarks saved if they are no longer interested in any listings.

1.4. Listing Feature

For registered sellers, listings can be created, modified, and deleted. These features will be further elaborated below.

1.4.1. Listing Creation

When creating new listings, registered sellers can toggle between creating listings for properties for sale or for rent using an option box. Upon selection of either properties for sale or for rent, specific fields pertaining to them are shown. Fields such as lease term and earliest move-in date can be stated in rental listings and fields such as tenure can be stated for properties for sales. This feature is created in the backend. Additionally, when sellers key in address details of their listings, with the use the Google Maps API, location of the listing would be shown in the map embed into the webpage. Postal code was not used in the API call as properties that are currently being built such as Built-To-Order (BTO) flats cannot be generated on google as it does not exist.

1.4.2. Listing Update

One of the key features in our webpage is that registered sellers can edit and add facilities. If registered sellers find that a certain facility is not being listed, they can simply enter the facility name and add it to our list of facilities. Registered sellers can add up to a maximum of 7 images per listing. Once the property is being sold or rented, registered sellers can indicate that the listings have either been sold or rented. This would be shown on the listing and would be removed from the search results.

1.4.3. Delete

The registered sellers can delete all the images for the listing with just one click. Additionally, the registered sellers can delete the listings if the listing is no longer valid. Once the listing is deleted, all the information will be deleted from database as well.

1.5. Comment Feature

A Comment feature was also incorporated into the site. This allows registered users to communicate with registered sellers show their interest in listings posted by registered sellers. Users are required to login to Roofy to use the comment feature on the listing details page. In addition, registered users can interact with other registered users by replying to comments that have posted. As for the registered sellers, they can only reply to comments on their own listings. In the comment box, the label “Seller” will be indicated next to the seller’s name which can help to differentiate replies from users and sellers.

1.6. Statistics

Every time someone clicks on learn more button for the listings, it logs some of the data into the database. For the Administrators, they have the privilege to view the statistic of the webpage. For example, they can view the number of views, listings, and the number of new sellers & users registered in the past months.

When users search for listings on our site and sees a listing of interest, they may click on “Learn More” to view the listing. This interaction is logged into our database in the views tables and sellers would be able to see the number of interested parties pertaining to their listing.

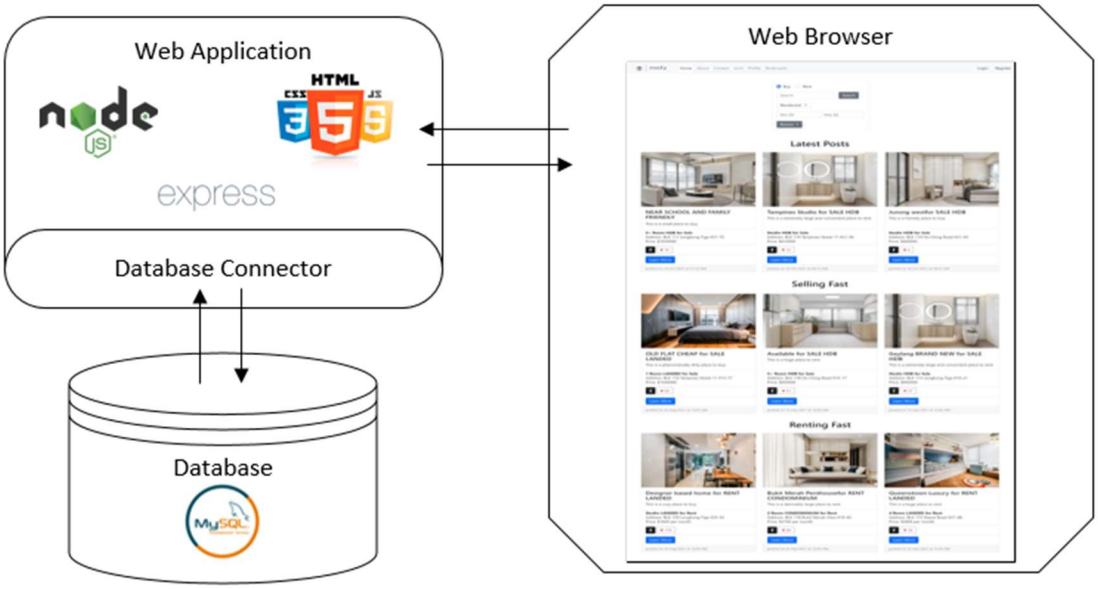
Only Administrators can see how well the site is doing and are able to view visualisations pertaining to metrics such as the total number of views, listings created, and the number of new sellers & users registered in a period.

1.7. Recent Transactions

We have incorporated a HDB API which is used to access recent sales transactions data that is publicly available on the Housing Development Board. Relevant past transactions would be pulled based on which town the listing is located in and number of rooms for each listing. Information such as addresses, resale price, lease term and floor range would be shown on all listing pages for properties meant for sale. This provides users an overview of past selling prices near a particular town and allows users to make comparisons between the asking price of a listing against past resale prices of similar properties.

2. Architecture

2.1. Overview



2.2. Web Server and Database Management System (MySQL)

MySQL was used as the Web Server and Database Management System. MySQL also acts as a Database Connector that connects the Database Management System and the Web Application. The inputs keyed in by both users and sellers in the Web browser will be forwarded to the Web Application which processes the input and send it to MySQL database via Database Connector.

2.3. Server Page Language

The following are the frontend and backend server page language tools that were used in the web page:

2.3.1. JavaScript

JavaScript supports the conditional, functional and application programming interface (API) which makes the Webpage more interactive.

Application Programming Interface (API)

Google Maps API
We have included Google Maps API in the webpage that enables us to display custom Google Maps on every listing page. Google Maps API is able to lookup the exact location on a map based on an address provided. Therefore, we have used Google Maps API to pinpoint the location of each address of a listing and display embed a map on the listing details page. This allows users to check the location of a property from the embed map easily.

HDB API
Besides, we have accessed the Government open data website https://data.gov.sg/ through an API. The library, *fetch-jsonp* is used to fetch data from data.gov.sg API. Recent HDB property transactions are fetched from data.gov.sg using API and displayed on every listing page for properties meant for sale. Relevant HDB property transactions data is pulled based on town and number of rooms that are identical to the listing.

Facebook API
Facebook API is used to connect between the webpage and Facebook system. A Facebook button has been added at each listing, this allows users to share the listing to Facebook with a click.

Bootstrap

Bootstrap is a HTML, CSS and JavaScript library and a front-end framework for developing webpage. Bootstrap is used for button styling. We have also used Carousel which is one of the Bootstrap to create slideshows of images.

Chart.js

Chart.js is a JavaScript library for data visualization. We used Chart.js to create a dashboard with line chart, pie chart, and bar chart to display the statistics of the webpage for administrators. The data was retrieved from MySQL using SQL statements.

Cookie-Parser

Cookie-Parser is a framework which helps to create and manage cookies. It helps to store login information and keep track of login sessions allowing users to continuously be logged in.

Express-Session

Express-Session manages session middleware and works with Cookie-Parser for login and session.

Embedded JavaScript (EJS)

EJS is used to load .ejs file and it allows us to generate HTML pages for front-end.

Express

Express is one of the Web Application frameworks that we have used. Express helps to handle GET and POST requests and acts as a library for Node.js. Express is used for routing and middleware as a callback function.

Multer

Multer is a middleware which used for uploading files, we have used it for image uploading at the create and edit portion of a listing page.

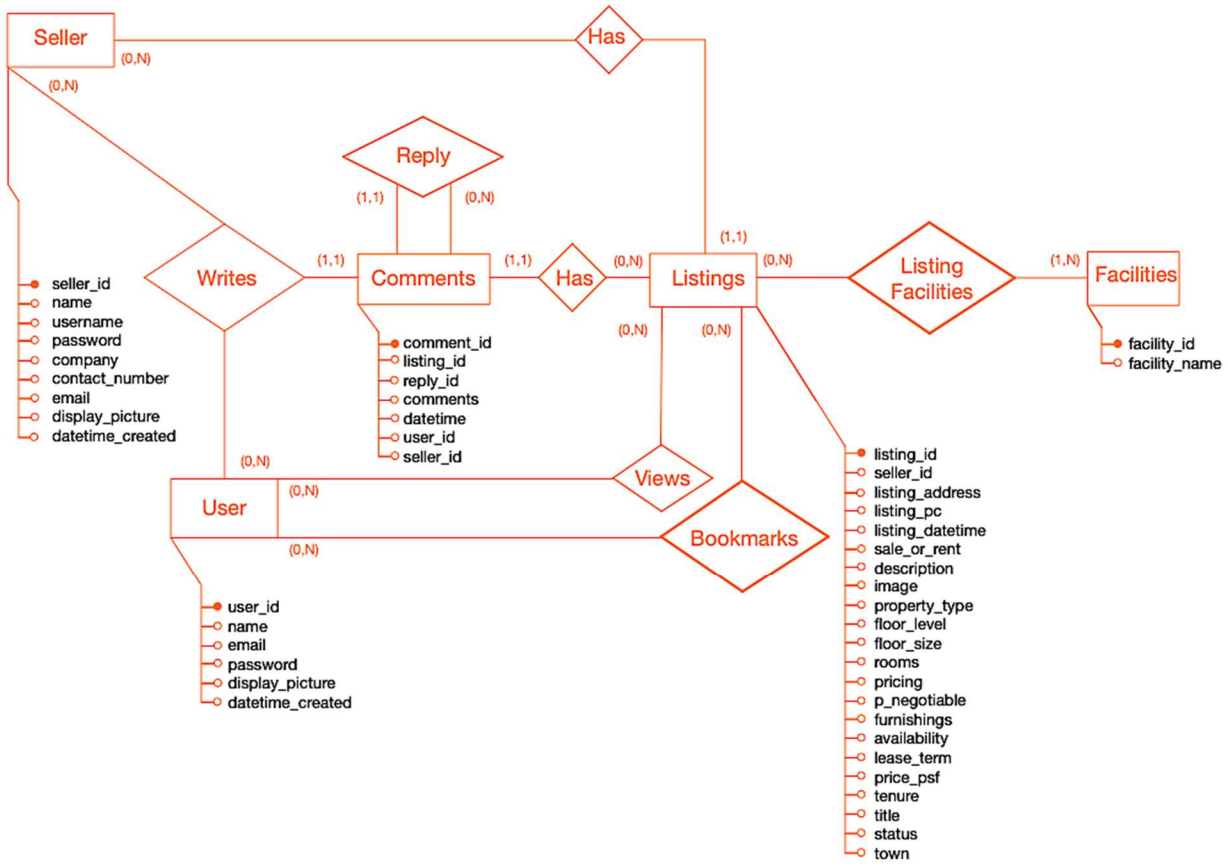
Node.js

Node.js is used as a running environment for JavaScript at the backend. Node.js can generate page content and collect the form data from a Web browser.

2.3.2. Cascading Style Sheets (CSS)

CSS is used to make the front-end of the webpage look better and provide users with a better experience. We used CSS like bootstrap.min.css, styles.css, font-awesome.min.css etc. CSS helps to define the style of the webpage which includes the design, layout, font, and other features. Additionally, the CSS icon is used on listings details page, this provides a unique visual effect that helps the users spot the information they want easily.

3. ERD Diagram



4. Relational Schema

Generated the DDL statements via MySQL with the default engine set to InnoDB and in utf8 format.

4.1. Bookmarks

```
CREATE TABLE `bookmarks` (
`user_id` int NOT NULL,
`listing_id` int NOT NULL,
PRIMARY KEY (`user_id`,`listing_id`),
KEY `listing_id` (`listing_id`),
 KEY `user_id` (`user_id`),
 CONSTRAINT `bookmarks_ibfk_1` FOREIGN KEY (`listing_id`) REFERENCES `listings` (`listing_id`) ON DELETE CASCADE,
CONSTRAINT `bookmarks_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `user` (`user_id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_c;
```

4.2. Comments

```
CREATE TABLE `comments` (
  `comment_id` int NOT NULL AUTO_INCREMENT,
  `listing_id` int NOT NULL,
  `reply_id` int DEFAULT NULL,
  `comments` text NOT NULL,
  `datetime` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `user_id` int DEFAULT NULL,
  `seller_id` int DEFAULT NULL,
  PRIMARY KEY (`comment_id`),
  KEY `listing_id` (`listing_id`),
  KEY `seller_id` (`seller_id`),
  KEY `reply_id` (`reply_id`),
  CONSTRAINT `comments_ibfk_2` FOREIGN KEY (`listing_id`) REFERENCES `listings` (`listing_id`) ON DELETE CASCADE,
  CONSTRAINT `comments_ibfk_4` FOREIGN KEY (`seller_id`) REFERENCES `seller` (`seller_id`) ON DELETE CASCADE,
  CONSTRAINT `comments_ibfk_5` FOREIGN KEY (`reply_id`) REFERENCES `comments` (`comment_id`) ON DELETE CASCADE,
  CONSTRAINT `comments_blank_check` CHECK ((`comments` <> _utf8mb4''))
)ENGINE=InnoDB AUTO_INCREMENT=250 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

4.3. Facilities

```
SELECT * FROM roofy.CREATE TABLE `facilities` (
`facility_id` int NOT NULL AUTO_INCREMENT,
`facility_name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT
NULL DEFAULT '',
PRIMARY KEY (`facility_id`),
UNIQUE KEY `facilitites_unique_facility_check` (`facility_name`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

4.4. Listing facilities

```
CREATE TABLE `listing_facilities` (
`listing_id` int NOT NULL,
`facility_id` int NOT NULL,
PRIMARY KEY (`listing_id`,`facility_id`),
KEY `listing_id` (`listing_id`),
KEY `facility_id` (`facility_id`),
CONSTRAINT `listing_facilities_ibfk_1` FOREIGN KEY (`listing_id`) REFERENCES `listings` (`listing_id`) ON
DELETE CASCADE,
CONSTRAINT `listing_facilities_ibfk_2` FOREIGN KEY (`facility_id`) REFERENCES `facilities` (`facility_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

4.5. Listings

```
 CREATE TABLE `listings` (
`listing_id` int NOT NULL AUTO_INCREMENT,
`seller_id` int DEFAULT NULL,
`listing_address` varchar(255) DEFAULT NULL,
`listing_pc` char(6) DEFAULT NULL,
`listing_datetime` datetime DEFAULT CURRENT_TIMESTAMP,
`sale_or_rent` varchar(4) DEFAULT 'SALE',
`description` text,
`image` varchar(128) DEFAULT NULL,
`property_type` varchar(11) DEFAULT NULL,
`floor_level` varchar(20) DEFAULT NULL,
`floor_size` int DEFAULT NULL,
`rooms` char(30) DEFAULT NULL,
`pricing` double DEFAULT NULL,
`p_negotiable` binary(1) DEFAULT NULL,
`furnishings` text,
`availability` date DEFAULT NULL,
`lease_term` char(10) DEFAULT NULL,
`price_psf` double DEFAULT NULL,
```

```
`tenure` varchar(20) DEFAULT NULL,
`title` varchar(45) DEFAULT 'cool house bruv',
`status` binary(1) DEFAULT NULL,
`town` varchar(45) NOT NULL DEFAULT 'central area',
PRIMARY KEY (`listing_id`),
KEY `seller_id` (`seller_id`),
CONSTRAINT `listings_ibfk_1` FOREIGN KEY (`seller_id`) REFERENCES `seller` (`seller_id`) ON DELETE
CASCADE,
CONSTRAINT `listings_floor_level_check` CHECK (((`floor_level` like _utf8mb4'%ground%') or
(`floor_level` like _utf8mb4'%low%') or (`floor_level` like _utf8mb4'%mid%') or (`floor_level` like
_utf8mb4'%high%') or (`floor_level` like _utf8mb4'%penthouse%'))),
CONSTRAINT `listings_furnishings_check` CHECK (((`furnishings` like _utf8mb4'%FULLY FURNISHED%')
or (`furnishings` like _utf8mb4'%PARTIALLY FURNISHED%') or (`furnishings` like
_utf8mb4'%UNFURNISHED%'))),
CONSTRAINT `listings_lease_term_check` CHECK (((`lease_term` = NULL) or (`lease_term` like
_utf8mb4'%Short Term%') or (`lease_term` like _utf8mb4'%1 Year%') or (`lease_term` like _utf8mb4'%2
Years%') or (`lease_term` like _utf8mb4'%Flexible%'))),
CONSTRAINT `listings_listing_pc_check` CHECK (((`listing_pc` >= 10000) and (length(`listing_pc`) = 6))),
CONSTRAINT `listings_property_type_check` CHECK (((`property_type` like _utf8mb4'%CONDO%') or
(`property_type` like _utf8mb4'%HDB%') or (`property_type` like _utf8mb4'%LANDED%'))),
CONSTRAINT `listings_rooms_check` CHECK (((`rooms` like _utf8mb4'%1%') or (`rooms` like
_utf8mb4'%2%') or (`rooms` like _utf8mb4'%3%') or (`rooms` like _utf8mb4'%4%') or (`rooms` like
_utf8mb4'%5%') or (`rooms` like _utf8mb4'%Studio%') or (`rooms` like _utf8mb4'%Room Rental%'))),
CONSTRAINT `listings_sale_or_rent_check` CHECK (((`sale_or_rent` like _utf8mb4'%SALE%') or
(`sale_or_rent` like _utf8mb4'%RENT%'))),
CONSTRAINT `listings_tenure_check` CHECK (((`tenure` = NULL) or (`tenure` like _utf8mb4'%Freehold%')
or (`tenure` like _utf8mb4'%99-year Leasehold%') or (`tenure` like _utf8mb4'%999-year Leasehold%') or
(`tenure` like _utf8mb4'%Unknown Tenure%'))),
CONSTRAINT `sale_no_rent_check` CHECK ((((`sale_or_rent` like _utf8mb4'%SALE%') and (not((`rooms`
like _utf8mb4'%Room Rental%')))) or (`sale_or_rent` like _utf8mb4'%RENT%')))
) ENGINE=InnoDB AUTO_INCREMENT=39886111 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

4.6. Seller

```
CREATE TABLE `seller` (
`seller_id` int NOT NULL AUTO_INCREMENT,
`name` varchar(128) DEFAULT NULL,
`username` varchar(255) NOT NULL,
`password` char(255) NOT NULL,
`company` varchar(255) NOT NULL,
`contact_number` varchar(8) DEFAULT NULL,
`email` varchar(255) NOT NULL,
`display_picture` varchar(128) DEFAULT NULL,
`datetime_created` datetime DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (`seller_id`),
UNIQUE KEY `username` (`username`),
UNIQUE KEY `email` (`email`),
UNIQUE KEY `contact_number` (`contact_number`),
CONSTRAINT `seller_contact_number_check` CHECK ((((length(`contact_number`) = 8) and
(`contact_number` like _utf8mb4'9%')) or (`contact_number` like _utf8mb4'8%'))),
CONSTRAINT `seller_email_check` CHECK ((`email` like _utf8mb4'%@%'))
) ENGINE=InnoDB AUTO_INCREMENT=29766462 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

4.7. User

```
CREATE TABLE `user` (
`user_id` int NOT NULL AUTO_INCREMENT,
`name` varchar(128) DEFAULT NULL,
`email` varchar(255) NOT NULL,
`password` char(255) NOT NULL,
`display_picture` varchar(128) DEFAULT NULL,
`datetime_created` datetime DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (`user_id`),
UNIQUE KEY `email` (`email`),
CONSTRAINT `user_email_check` CHECK ((`email` like _utf8mb4'%@%'))
) ENGINE=InnoDB AUTO_INCREMENT=19993407 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

4.8. Views

```
CREATE TABLE `views` (
`view_id` int NOT NULL AUTO_INCREMENT,
`user_id` int DEFAULT NULL,
`listing_id` int DEFAULT NULL,
`seller_id` int DEFAULT NULL,
`datetime_viewed` datetime DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (`view_id`),
KEY `listing_id` (`listing_id`),
KEY `user_id` (`user_id`),
KEY `views_ibfk_3` (`seller_id`),
CONSTRAINT `views_ibfk_1` FOREIGN KEY (`listing_id`) REFERENCES `listings` (`listing_id`) ON DELETE
CASCADE,
CONSTRAINT `views_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `user` (`user_id`) ON DELETE
CASCADE,
CONSTRAINT `views_ibfk_3` FOREIGN KEY (`seller_id`) REFERENCES `seller` (`seller_id`) ON DELETE
CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=2026 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

5. Sample SQL Statements

These statements make up the bulk of the JSON files that is used for transmitting data between the web application and the database.

5.1. Bookmarks

To enable users to create bookmarks, delete bookmarks and to get the bookmarks that they have stored on their accounts. Date is formatted for user readability.

```
const createBookmark = (user_id, listing_id, res) => {
    let sql = `
        INSERT INTO bookmarks VALUES (?, ?)
    `

    db.query(sql, [user_id, listing_id], (err,result,fields) => {
        if (err) {
            console.log("err", err)
            res({status: false, data: [], msg: err.message})
        } else {
            res({status: true, data: [], msg: 'Bookmark Added'})
        }
    })

}
const deleteBookmark = (user_id, listing_id, res) => {
    let sql = `
        DELETE FROM bookmarks
        WHERE user_id = ?
        AND listing_id = ?;
    `

    db.query(sql, [user_id, listing_id], (err,result,fields) => {
        if (err) {
            console.log("err", err)
            res({status: false, data: [], msg: err.message})
        } else {
            res({status: true, data: [], msg: 'Bookmark Added'})
        }
    })
```

```
}

const getBookmarks = (user_id, res) => {
    let sql = `
        SELECT l.*, b.user_id, COUNT(b1.listing_id) AS countbookmarks, DATE_FORMAT(l.listing_datetime,
'%d %b %Y at %h:%i %p') AS niceD8
        FROM listings l
        INNER JOIN bookmarks b ON b.listing_id = l.listing_id
        LEFT JOIN bookmarks b1 ON b1.listing_id = l.listing_id
        WHERE b.user_id = ?
        GROUP BY l.listing_id;
    `
    // run the sql query on db
    db.query(sql, user_id, (err, row) => {
        if (err) {
            res({success:false, msg:err});
        }
        else {
            res({success:true, data:row, msg:'list of listings'});
        }
    })
}
```

5.2. Comments

Users can add comments to a listing and reply to comments made by other users.

```
const getAllComments = (listing_id, res) => {
    //cu standfor combined-user between seller and user
    let sql = `
    SELECT c.listing_id, c.comment_id, c.reply_id, c.comments, c.datetime, c.seller_id, cu.name,
cu.display_picture
    FROM comments c
    INNER JOIN
    (
        SELECT NULL AS "user_id", seller_id, name, display_picture
        FROM seller
    UNION
        SELECT user_id, NULL AS "seller_id", name, display_picture
        FROM user
    ) AS cu
    ON (cu.user_id = c.user_id OR cu.seller_id = c.seller_id)
    WHERE c.listing_id = ?
    ORDER BY reply_id, datetime DESC
    ;`;
```

```
    db.query(sql, listing_id, (err, row) => {
        if(err) {
            res({status: false, data: [], msg: 'not found'})
        } else {
            let comments = [];
            row.forEach(comment => {
                if(comment.reply_id === null){
                    comments.push(comment)
                } else {
                    let index = comments.findIndex((obj => obj.comment_id == comment.reply_id))
                    if(index != -1) {
                        if(!("replies" in comments[index])) {
                            comments[index]["replies"] = [comment]
                        } else {
                            comments[index]["replies"].push(comment)
                        }
                    }
                }
            });
            res({status: true, data: comments, msg: 'list of listings'})
        }
    })
}
const addComment = (listing_id, comments, user_id, res) => {
    let sql = `
        INSERT INTO comments (listing_id, comments, datetime, user_id) VALUES (?, ?, ?, ?)
    `
    let date = new Date();
    db.query(sql, [listing_id, comments, date, user_id], (err) => {
        if (err) {
            console.log("err", err)
            res({status: false, data: [], msg: err.message})
        } else {
            res({status: true, data: [], msg: 'Comment Added'})
        }
    })
}

const replyComment = (listing_id, reply_id, comments, user_id, res) => {
    let sql = `
        INSERT INTO comments (listing_id, reply_id, comments, datetime, user_id) VALUES (?, ?, ?, ?, ?)
    `
    let date = new Date()
    db.query(sql, [listing_id, reply_id, comments, date, user_id], (err) => {
        if (err) {
```

```
            console.log("err", err)
            res({status: false, data: [], msg: err.message})
        } else {
            res({status: true, data: [], msg: 'Reply Added'})
        }
    })
}
```

5.3. Facilities

Sellers can indicate the facilities that they would like to include in their listing. If the facility does not exist (e.g. Golf Course), they are able to add a new facility into the “facilities” table.

```
const getFacilities = (res) => {
    let sql = `
        SELECT *
        FROM facilities ORDER BY facility_id;
    `;
    db.query(sql, (err, data) => {
        if(err) {
            res({status: false, data: [], msg: 'not found'})
        } else {
            res({status: true, data: data, msg: 'list of facilities'})
        }
    })
}

const addFacility = (new_facility, res) => {
    // console.log("new facility name: " + new_facility);
    let sql = `
        INSERT INTO facilities(facility_name)
        VALUES (?);
    `;
    db.query(sql, new_facility, (err, data) => {
        if(err) {
            res({status: false, data: [], msg: 'not found'})
        } else {
            res({status: true, data: data, msg: 'list of facilities'})
            console.log("facility added: " + new_facility);
        }
    })
}

const deleteFacilitiesFromListing = (listing_id, res) => {
```

```
    let sql = `
        DELETE FROM listing_facilities
        WHERE listing_id = ?;
    `
    db.query(sql, listing_id, function(err, data){

        if (err) {
            res({status: false, data: [], msg: err.message})
        }
        else {
            res({status: true, data: data, msg: 'list of facilities'})
        }
    })
}


const addFacilities = (listing_id, facilities, res) => {
    let sql = `
        INSERT INTO listing_facilities (listing_id, facility_id)
        VALUES ?;
    `
    var values = []
    for(const facility of facilities) {
        values.push([listing_id, facility])
    }
    db.query(sql, [values], function(err, data){

        if (err) {
            res({status: false, data: [], msg: err.message})
        }
        else {
            res({status: true, data: data, msg: 'added facilities'})
        }
    })
}
```

5.4. Search

Searches can be made with multiple conditions such as renting or buying, price, number of rooms, types of housing etc. Since users can search for properties with various room sizes, we dynamically added SQL constraints for each room type based on user inputs. In addition, if a user is logged in, their bookmarks are retrieved as “countbookmarks”.

```
const searchListings = ([search, sale_or_rent, property_type, price_lower_bound, price_upper_bound,
room_rental, studio, _1room, _2room, _3room, _4room, _5room], userid, res) => {
    let sql;
    let question_mark;
    let and_inputted = false;


    if (property_type != "hdb" && property_type != "condo" && property_type != "landed") {
        property_type = "";
    }
    if (price_lower_bound < 0) {
        price_lower_bound = 0;
    }
    if (price_upper_bound < 0 || price_upper_bound <= price_lower_bound) {
        price_upper_bound = price_lower_bound + 100000000;
    }

    if (userid) { // get bookmarks too
        sql = `
            SELECT l.*, b.user_id, COUNT(b1.listing_id) AS countbookmarks,
DATE_FORMAT(l.listing_datetime, '%d %b %Y at %h:%i %p') AS niceD8
            FROM listings l
            LEFT JOIN bookmarks b ON l.listing_id = b.listing_id AND b.user_id = ?
            LEFT JOIN bookmarks b1 ON b1.listing_id = l.listing_id
            WHERE (l.listing_address LIKE "%"?"%" OR l.title LIKE "%"?"%")
            AND l.sale_or_rent = ?
            AND l.property_type LIKE "%"?"%"
            AND l.pricing >= ? AND pricing <= ?
        `
        question_mark = [userid, search, search, sale_or_rent, property_type, price_lower_bound,
price_upper_bound];
    }
    else {
        sql = `
            SELECT l.*, COUNT(b1.listing_id) AS countbookmarks, DATE_FORMAT(l.listing_datetime,
'%d %b %Y at %h:%i %p') AS niceD8
            FROM listings l
            LEFT JOIN bookmarks b1 ON b1.listing_id = l.listing_id
```

```
            WHERE (l.listing_address LIKE "%"?"%" OR l.title LIKE "%"?"%")
            AND l.sale_or_rent = ?
            AND l.property_type LIKE "%"?"%"
            AND l.pricing >= ? AND l.pricing <= ?
        `
        question_mark = [search, search, sale_or_rent, property_type, price_lower_bound,
price_upper_bound];
    }
    if (room_rental) {
        if (and_inputted) {
            sql += `
                OR l.rooms LIKE 'Room Rental'
            `
        }
        else {
            sql += `
                AND (l.rooms LIKE 'Room Rental'
            `
            and_inputted = true;
        }
    }

    if (studio) {
        if (and_inputted) {
            sql += `
                OR l.rooms LIKE 'studio'
            `
        }
        else {
            sql += `
                AND (l.rooms LIKE 'studio'
            `
            and_inputted = true;
        }
    }

    if (_1room) {
        if (and_inputted) {
            sql += `
                OR l.rooms LIKE '1%'
            `
        }
        else {
            sql += `
                AND (l.rooms LIKE '1%'
```

```
        `
        and_inputted = true;
    }
}

if (_2room) {
    if (and_inputted) {
        sql += `
            OR l.rooms LIKE '2%'
        `
    }
    else {
       sql += `
            AND (l.rooms LIKE '2%'
        `
        and_inputted = true;
    }
}

if (_3room) {
    if (and_inputted) {
        sql += `
            OR l.rooms LIKE '3%'
        `
    }
    else {
        sql += `
            AND (l.rooms LIKE '3%'
        `
        and_inputted = true;
    }
}

if (_4room) {
    if (and_inputted) {
        sql += `
            OR l.rooms LIKE '4%'
        `
    }
    else {
        sql += `
            AND (l.rooms LIKE '4%'
        `
        and_inputted = true;
    }
```

```
}

if (_5room) {
    if (and_inputted) {
        sql += `
            OR l.rooms LIKE '5%'
        `
    }
    else {
        sql += `
            AND (l.rooms LIKE '5%'
        `
        and_inputted = true;
    }
}

if (and_inputted)
    sql += `)`;

sql += `
    GROUP BY l.listing_id
    ORDER BY l.listing_datetime DESC;
`
```

5.5. Listings

Allows sellers to list their property in detail and add multiple images. As a seller, user is allowed to edit their listing and delete images of their listing.

```
const getListings = (id, res) => {
    let sql = `
        SELECT *
        FROM listings
        WHERE seller_id = ? ORDER BY listing_datetime DESC;
    `;
    db.query(sql, id, (err, row) => {
        if(err) {
            res({status: false, data: [], msg: 'not found'})
        } else {
            res({status: true, data: row, msg: 'list of listings'})
        }
    })
}

const addListing = (id, image, req, res) => {
    let user_id = id;
    let title = req.body.title;
    let listing_address = req.body.listing_address;
    let listing_pc = req.body.listing_pc;

    let sale_or_rent = req.body.sale_or_rent;
    let description = req.body.description;
    let property_type = req.body.property_type;
    let floor_level = req.body.floor_level;
    let floor_size = req.body.floor_size;
    let rooms = req.body.rooms;
    let pricing = req.body.pricing;
    let p_negotiable = req.body.p_negotiable;
    let furnishings = req.body.furnishings;
    let availability = req.body.availability == "" ? null : req.body.availability;
    let lease_term = req.body.lease_term == "" ? null : req.body.lease_term;
    let price_psf = pricing/floor_size;
    let tenure = req.body.tenure;
    let facilities = req.body.facilities;
    let town = req.body.town;

    let listingsql = `
        INSERT INTO listings(seller_id, listing_address, listing_pc, sale_or_rent, description, image,
property_type, floor_level, floor_size, rooms, pricing, p_negotiable, furnishings, availability, lease_term,
price_psf, tenure, title, town)
```

```
        VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?);`;

    let listingfacilitiessql = `
        INSERT INTO listing_facilities (listing_id, facility_id) VALUES ?
        ;`;

    db.query(listingsql, [user_id, listing_address, listing_pc, sale_or_rent, description, image, property_type,
floor_level, floor_size, rooms, pricing, p_negotiable, furnishings, availability, lease_term, price_psf, tenure,
title, town],(err, result, fields)=>{
        if (err) {
            console.log("err", err)
            res({status: false, data: [], msg: err.message})
        } else {
            if (result.insertId && facilities) {
                var values = []
                for(const facility of facilities) {
                    values.push([result.insertId, facility])
                }
                db.query(listingfacilitiessql, [values], function(err){
                    if (err) {
                        res({status: false, data: [], msg: err.message})
                    } else {
                        res({status: true, data: [], msg: 'Added new listing'})
                    }
                })
            } else {
                res({status: true, data: [], msg: 'Added new listing'})
            }
        }
    })
}

const addListingImage = (id, image, req, res) => {
    let sql = `
        SELECT image
        FROM listings
        WHERE listing_id = ?;
    `

    db.query(sql, id, (err,row) => {
        if (err) {
            console.log("err", err)
            res({status: false, data: [], msg: err.message})
        } else {
            // res({status: true, data: [], msg: 'Updated Image'})
```

```
            console.log("row: " + row[0].image);
            // res({row});


            let newImageString;
            if (row[0].image == '')
                newImageString = image;
            else
                newImageString = row[0].image + ',' + image;
            sql = `
                UPDATE listings
                SET image = ?
                WHERE listing_id = ?;
            `
            db.query(sql, [newImageString, id], (err,result,fields) => {
                if (err) {
                    console.log("err", err)
                    res({status: false, data: [], msg: err.message})
                } else {
                    res({status: true, data: [], msg: 'Added Image'})
                }
            })
        }
    })
}


const deleteAllImages = (id, res) => {
    let sql = `
        UPDATE listings
        SET image = ''
        WHERE listing_id = ?;
    `

    db.query(sql, id, (err,row) => {
        if (err) {
            console.log("err", err)
            res({status: false, data: [], msg: err.message})
        } else {
            res({status: true, data: [], msg: 'Deleted All Images'})
        }
    })
}

const editListing = (req, res) => {
```

```
    let listing_id = req.params.listingid;
    let sale_or_rent = req.body.sale_or_rent;
    let title = req.body.title;
    let listing_address = req.body.address;
    let listing_pc = req.body.listing_pc;
    let description = req.body.description;
    let property_type = req.body.property_type;
    let floor_level = req.body.floor_level;
    let rooms = req.body.rooms;
    let furnishings = req.body.furnishings;
    let floor_size = req.body.floor_size;
    let tenure = req.body.tenure;
    let pricing = req.body.pricing;
    let availability = req.body.availability;
    let lease_term = req.body.lease_term;
    let town = req.body.town;

    if (availability == '') { availability = null; }

    let listingsql = `
        UPDATE listings
        SET sale_or_rent = ?, title = ?, listing_address = ?, listing_pc = ?, description = ?, property_type = ?,
        floor_level = ?, rooms = ?, furnishings = ?, floor_size = ?, tenure = ?, pricing = ?,
        availability = ?, lease_term = ?, town = ?
        WHERE listing_id = ?;
    `;
    let questionMark = [sale_or_rent, title, listing_address, listing_pc, description, property_type, floor_level,
rooms, furnishings, floor_size, tenure, pricing, availability, lease_term, town, listing_id];

    db.query(listingsql, questionMark,(err, result)=>{
        if (err) {
            console.log("err", err)
            res({status: false, data: [], msg: err.message})
        } else {
            res({status: true, data: [], msg: 'Added new listing'})
        }
    })
}
```

5.6. Views

Each time a user sees the details of a listing, this function creates a row in the "views" table along with the datetime. The admin will then be able to see statistics on the admin page.

```
const addUserView = (user_id, listing_id, res) => {
    let sql = 'INSERT INTO views (user_id, listing_id) VALUES (?, ?)';

    db.query(sql, [user_id, listing_id], (err, row) => {
        if(err) {
            res({status: false, data: [], msg: 'insert fail'})
        } else {
            res({status: true, data: row, msg: 'insert successfully'})
        }
    })
}

const addSellerView = (seller_id, listing_id, res) => {
    //cu standfor combined-user between seller and user
    let sql = 'INSERT INTO views (seller_id, listing_id) VALUES (?, ?)';

    db.query(sql, [seller_id, listing_id], (err, row) => {
        if(err) {
            res({status: false, data: [], msg: 'insert fail'})
        } else {
            res({status: true, data: row, msg: 'insert successfully'})
        }
    })
}
```

5.7. Admin

Allows Admin to view all available statistics of listings, users and views.

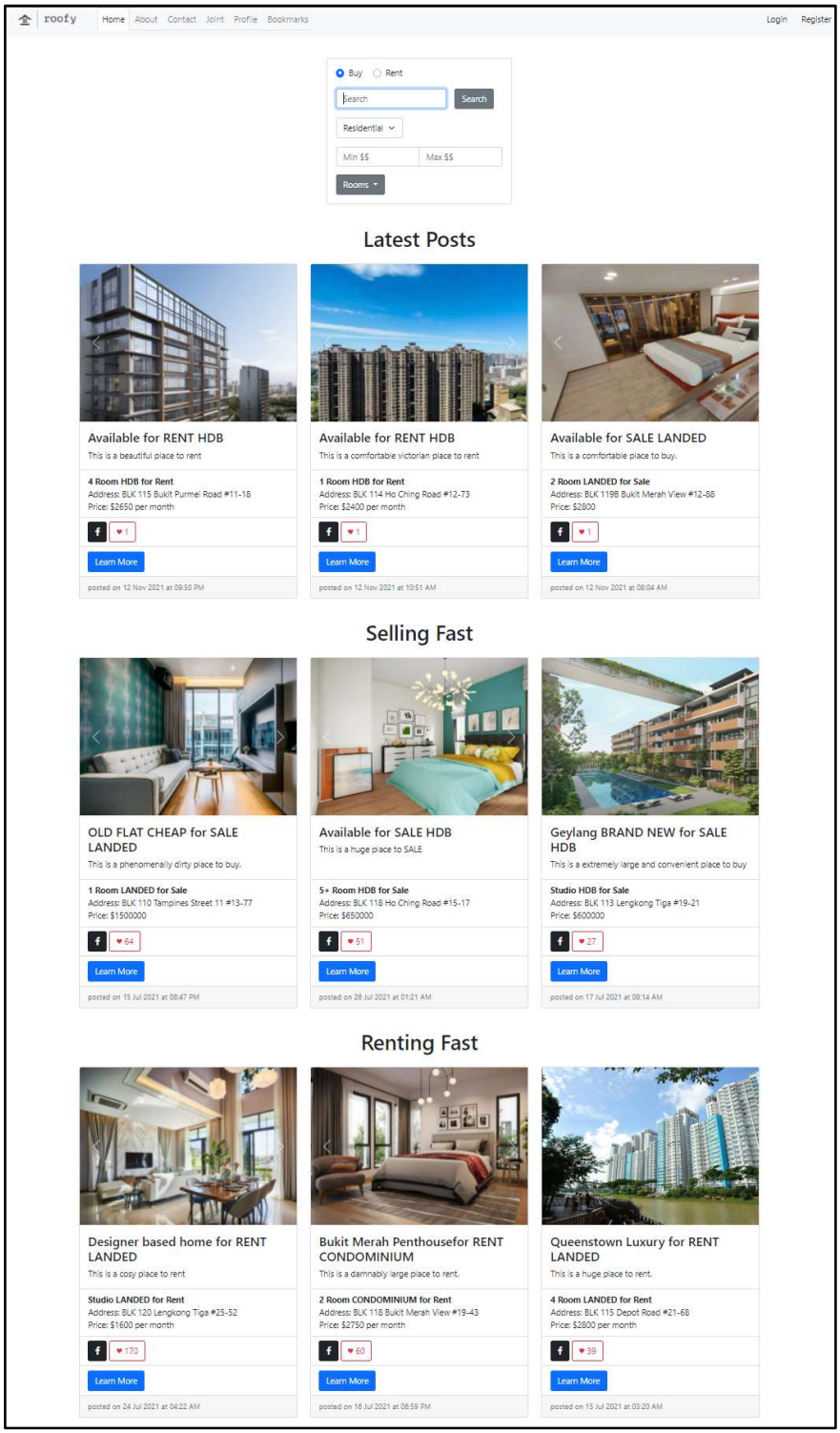
```
const getListingStats = (res) => {
    let sql = `
        (
            SELECT COUNT(*) AS last_week
            FROM listings l
            WHERE l.listing_datetime BETWEEN DATE_SUB(CURRENT_DATE(),
INTERVAL 30 DAY) AND CURRENT_DATE()
        )
        UNION ALL (
            SELECT COUNT(*)
            FROM listings
            WHERE listing_datetime BETWEEN DATE_SUB(CURRENT_DATE(),
INTERVAL 60 DAY) AND DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
        )
        UNION ALL (
            SELECT COUNT(*)
            FROM listings
            WHERE listing_datetime BETWEEN DATE_SUB(CURRENT_DATE(),
INTERVAL 90 DAY) AND DATE_SUB(CURRENT_DATE(), INTERVAL 60 DAY)
        )
        UNION ALL (
            SELECT COUNT(*)
            FROM listings
            WHERE listing_datetime BETWEEN DATE_SUB(CURRENT_DATE(),
INTERVAL 120 DAY) AND DATE_SUB(CURRENT_DATE(), INTERVAL 90 DAY)
        );
    `
    // run the sql query on db
    db.query(sql, (err, row) => {
        if (err) {
            res({success:false, msg:err});
        }
        else {
            res({success:true, data:row, msg:'past month listing
stats'});
        }
    })
}

const getUserStats = (res) => {
    let sql = `
        SELECT (
```

```
            SELECT COUNT(*) FROM user WHERE datetime_created BETWEEN
DATE_SUB(CURRENT_DATE(), INTERVAL 366 DAY) AND CURRENT_DATE()
        ) AS userCount, (
            SELECT COUNT(*) FROM seller WHERE datetime_created BETWEEN
DATE_SUB(CURRENT_DATE(), INTERVAL 366 DAY) AND CURRENT_DATE()
        ) AS sellerCount;
    `;
    // run the sql query on db
    db.query(sql, (err, row) => {
        if (err) {
            res({success:false, msg:err});
        }
        else {
            res({success:true, data:row, msg:'past week user stats'});
        }
    })
}

const getViewStats = (res) => {
    let sql = `
        Select day(datetime_viewed) as day, count(*) as countViews
        FROM views
        WHERE month(datetime_viewed) = month(CURRENT_DATE())
        AND year(datetime_viewed) = year(CURRENT_DATE())
        GROUP BY day
        ORDER BY day ASC
    `;
    // run the sql query on db
    db.query(sql, (err, row) => {
        if (err) {
            res({success:false, msg:err});
        }
        else {
            res({success:true, data:row, msg:'past week user stats'});
        }
    })
}

const getViewStatsMonth = (req, res) => {
    let sql;
    if (!(req.params.month) || !(req.params.year)) { // normal
        sql = `
            Select day(datetime_viewed) as day, count(*) as countViews
            FROM views
            WHERE month(datetime_viewed) = month(CURRENT_DATE())
```

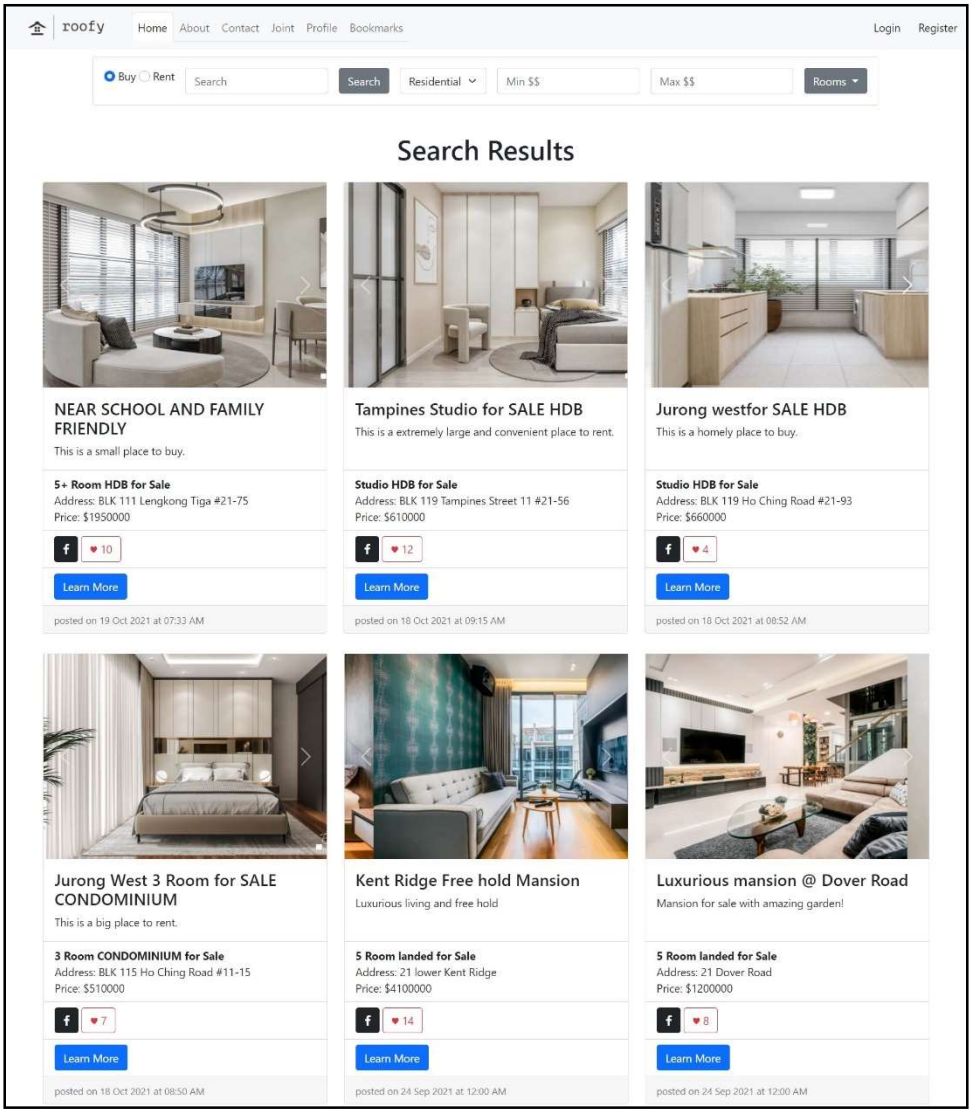
```
            AND year(datetime_viewed) = year(CURRENT_DATE())
            GROUP BY day
            ORDER BY day ASC
        `;
        // run the sql query on db
        db.query(sql, (err, row) => {
            if (err) {
                res({success:false, msg:err});
            }
            else {
                res({success:true, data:row, msg:'past month user
stats'});
            }
        })
    }
    else {
        let month = req.params.month;
        let year = req.params.year;
        sql = `
            Select day(datetime_viewed) as day, count(*) as countViews
            FROM views
            WHERE month(datetime_viewed) = ?
            AND year(datetime_viewed) = ?
            GROUP BY day
            ORDER BY day ASC
        `;
        // run the sql query on db
        db.query(sql, [month, year], (err, row) => {
            if (err) {
                res({success:false, msg:err});
            }
            else {
                res({success:true, data:row, msg:'past month user
stats'});
            }
        })
    }

}
```

6. Web Interface

6.1. Home Page



6.2. Search Feature



6.3. Listing Details

Punggol Point Woods

2 Room, HDB

Listing ID
33

Status
Available

Posted by
5

Posted on
18 Jul 2021 at 07:52 AM

amy
+6595674321
+6595674321
Ratings

Description
Punggol Point Woods, located within the Punggol Point district of Punggol Eco-Town, is bounded by Punggol Way and New Punggol Road. This development comprises 6 residential blocks ranging from 5 to 22 storeys, and offers 940 units of 2-room Flexi, 3-, 4-, and 5-room flats.

Sales or Rent
SALE

Property Type
HDB

Floor Level
mid

Floor Size
502

Rooms
2 Room

Furnishings
Partially Furnished

Facilities
- Air conditioning
- Balcony

Pricing
$ 490000

Price per psf
$ 976.096

Price Negotiable?
no

Tenure
Freehold

Town
punggol

Address
432B Northshore Dr

Postal Code
824409

Recent 2 Room transactions in punggol

Address	Address	Address
256C SUMANG WALK	622C PUNGGOL CTRL	209C PUNGGOL PL
Resale Price	**Resale Price**	**Resale Price**
$ 280000	$ 298000	$ 278000
Model A (47sqm)	Model A (47sqm)	Model A (46sqm)
Lease Term	**Lease Term**	**Lease Term**
94 years 03 months	91 years 07 months	92 years 03 months
Storey Range	**Storey Range**	**Storey Range**
01 TO 03	13 TO 15	07 TO 09

New Comment

w0w such listing prof A+ grade pls

Message

Post comment

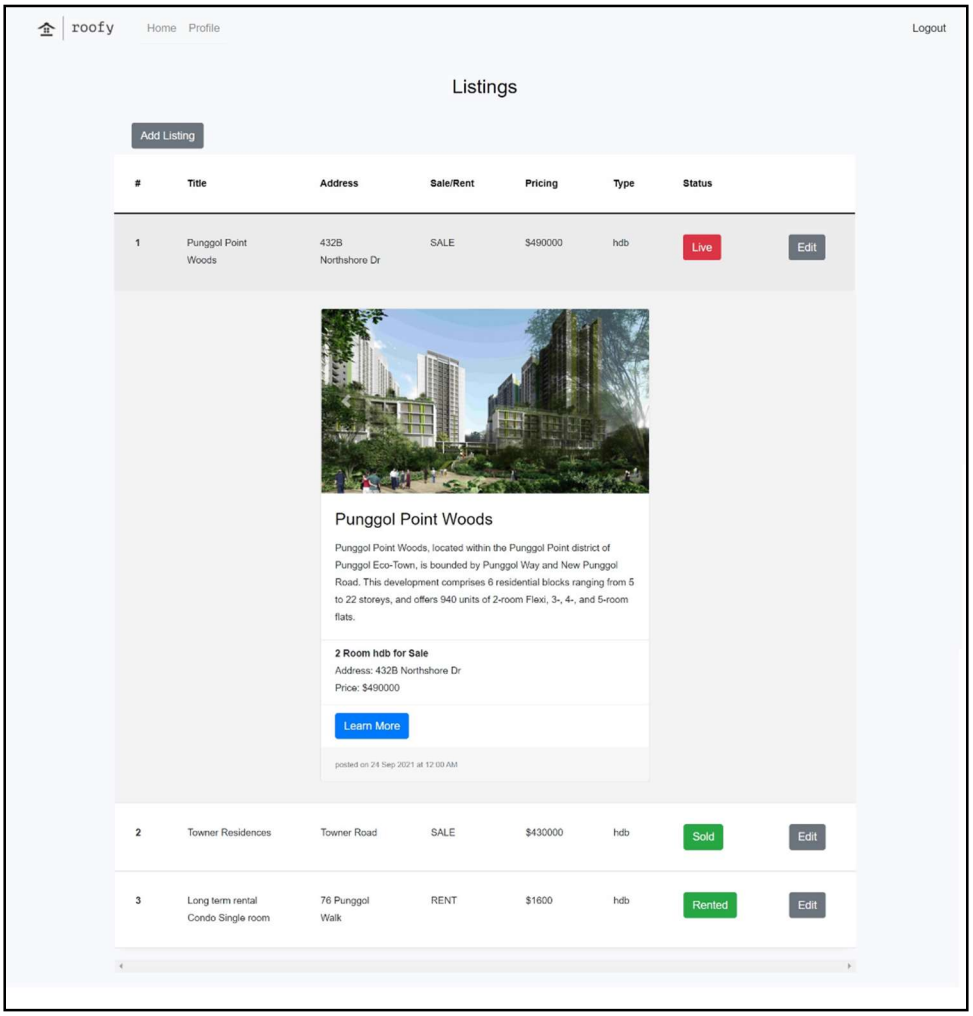
1 Comments

test
a few seconds ago

w0w such listing prof A+ grade pls

Reply

6.4. Seller's Listings



6.5. Admin Page