Damanhour University

Faculty of Computers and Information

SUPERVISED BY

Dr. Nora Shoaip

Outlines

- Introduction & Related Work
- Problem Definition
- System Designs
- Project Aspects
- Implementation
- Results and discussion
- Testing & Security
- Future Work
- References
- Conclusions

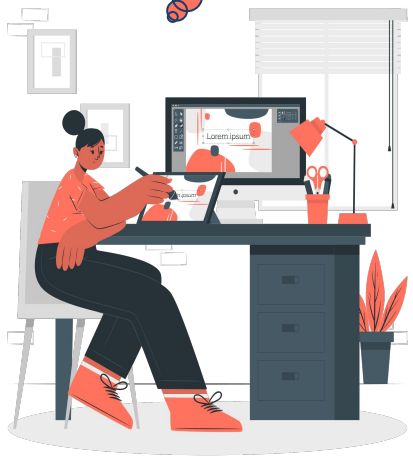
Introduction - Title

MASKANK
مَسكَنك

An application for renting apartments using AI

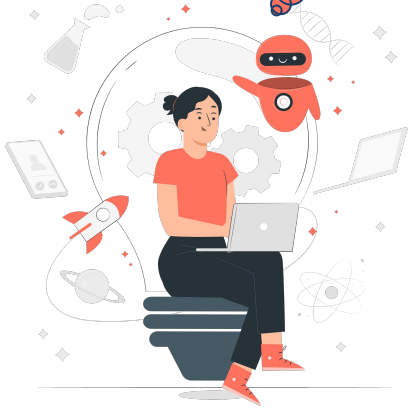
Introduction - Our Team

Ui /UX



Kholoud Osama

Ai



Donia Gaber

Amira Ashraf

Mobile

Ali Sharaf

Ahmed Said

Mariam Edw

Kholoud Osama

ar

Back End

Eman Khalil

Mohamed Salah

Mostafa Abdallah

Mahmoud Mohamed

Introduction

How did we find an apartment to rent in the past?

Introduction

Hello, I am Ali, a student at the Faculty of Computers, Damanhour. I am from Alexandria, and the distance between me and the college is 2 hours. I have a problem in finding an apartment for rent at a fair price.

Introduction

I found an application for renting apartments at a fair price because it uses artificial intelligence to determine the rental price and enables me to communicate with the owner directly without a broker.

Related Works

Bayut egypt

- Rich people
- Buy Only
- Use seller

Related Works

- Property finder
  - So expensive
  - Don’t use Ai
  - Use seller

Related Works

Maskank App

- All people - Students
- Use Ai
- Don’t use seller

MASKANK
مَسكَنك

Problem Definition

First Issue

Unfair Pricing

Second Issue

Middleman Costs

Third Issue

Annoying

Problem Definition - Unfair Pricing

ISSUE

Unfair Pricing

1. Fair rental price becomes a point of contention
2. Difficulty in paying rent.

RESOLUTIONS

Our AI-driven application employs sophisticated algorithms

1. We use AI determine a rental price

OUTCOMES

Fair Pricing

1. Satisfaction
2. Owner's assistant

Problem Definition - Middleman

ISSUE

Middleman Costs

1. The traditional system requires a broker to rent an apartment

RESOLUTIONS

Our app acts as an Middleman

1. We use the app to work in this role

OUTCOMES

Save money

1. Satisfaction
2. Owner protection

Problem Definition - Annoying

ISSUE

Annoying

1. Both landlords and tenants typically face privacy concerns

RESOLUTIONS

Maskanak App save rights of both

1. prioritizing data security for both landlords and tenants

OUTCOMES

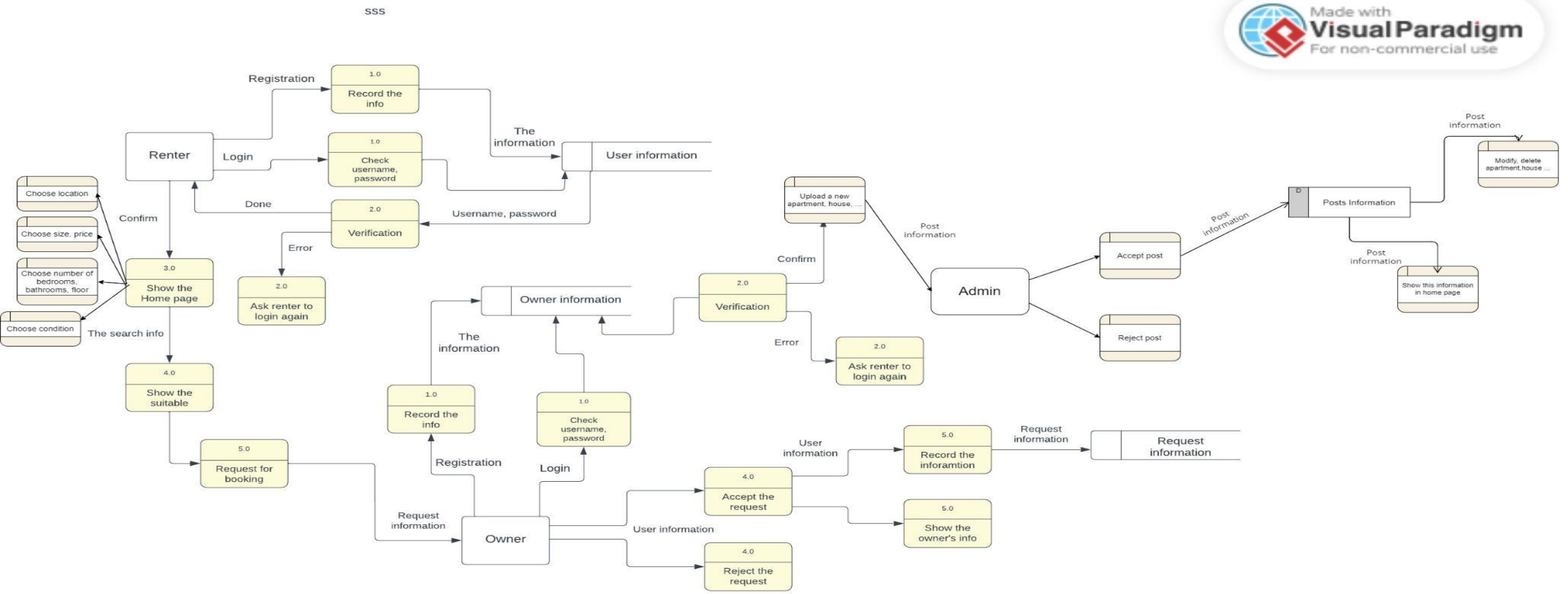
Comfort

1. Save data
2. Protection from crazy u r

System Designs

Diagrams

System Designs - DFD



System Designs

Data Flow Diagram

This DFD details the process flow for user registration, login, and property management in a system involving renters, owners, and admins.

Renter Registration and Login

Registration

- The renter records their information.
- The system verifies the information.
- If successful, the renter's information is stored.
- If there's an error, the renter is prompted to try again.

Login

- The renter provides their username and password.
- The system checks the credentials.
- If verified, the renter is granted access to the home page.
- If there's an error, the renter is prompted to try again.

Owner Registration and Login

Registration

- The owner records their information.
- The system verifies the information.
- If successful, the owner's information is stored.
- If there's an error, the owner is prompted to try again.

Login

- The owner provides their username and password.
- The system checks the credentials.
- If verified, the owner is granted access to their dashboard.
- If there's an error, the owner is prompted to try again.

System Designs

Data Flow Diagram

Admin Management

Post Management

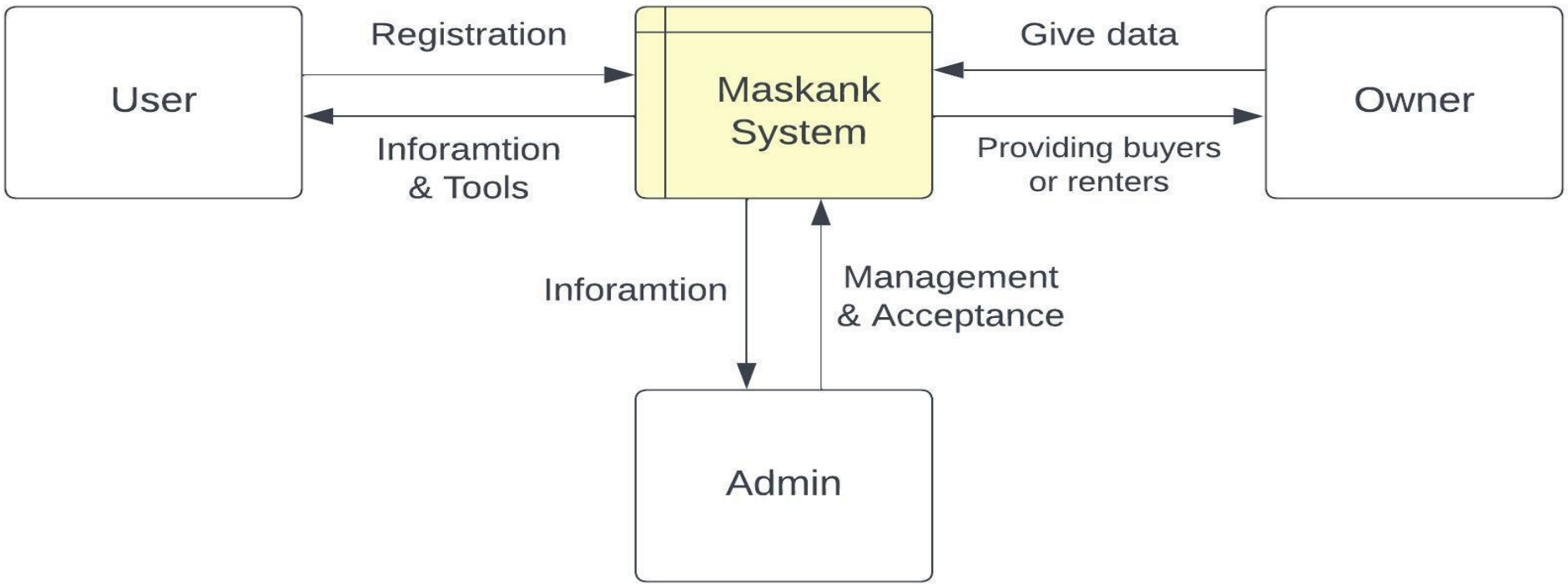
- Owners can upload new apartment or house information.
- Admin verifies the post information.
- Admin can either accept or reject the post.
- Accepted posts are shown in the home page.
- Rejected posts are not displayed and the owner is notified.

Property Search and Booking

- Renters can search for properties based on location, size, number of bedrooms, bathrooms, and condition.
- The system shows suitable properties.
- Renters can request booking for a property.
- The owner's information is displayed once the request is accepted.
- If the request is rejected, the renter is notified.

System Designs

Context Diagram



System Designs

Context Diagram

The context diagram represents the high-level view of the system and its interactions with external entities. In this case, we have three main entities:Renter, Owner, and Admin, interacting with the App

Renter

After registering with the App, the User gains access to various features and tools provided by the App. These features and tools can be used by the User for various purposes, such as searching for apartments, contacting owners, and managing their own information within the App.

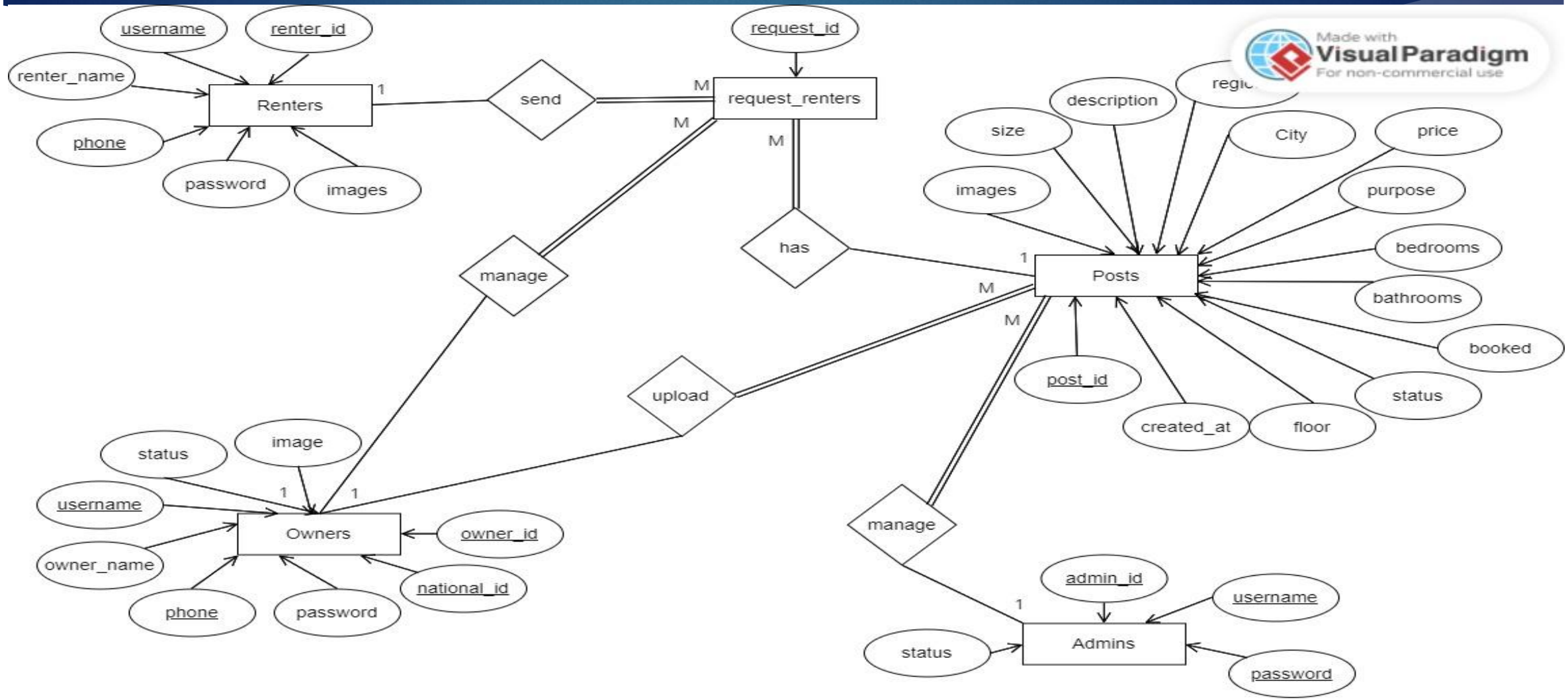
Owner

The Owner provides data to the App, such as apartment listings and relevant information. This data is used by the App to connect potential renters with available apartments. The Owner may also receive booking requests and other communication from Users through the App.

Admin

The Admin is responsible for managing the information within the App. The App provides relevant information to the Admin, which may include apartment listings, and other data. The Admin can perform various administrative tasks, such as approving new apartment listings or resolving disputes between Users and Owners.

System Designs - ERD



System Designs

ER Diagram

This Entity-Relationship Diagram (ERD) illustrates the primary structure of a system involving **renters**, **owners**, **admins**, and **posts**.

Renters

This entity represents the users who are renting properties. Key attributes include renter_name, username, phone, password, and images. Renters can send requests (request_renters) and manage posts.

Owners

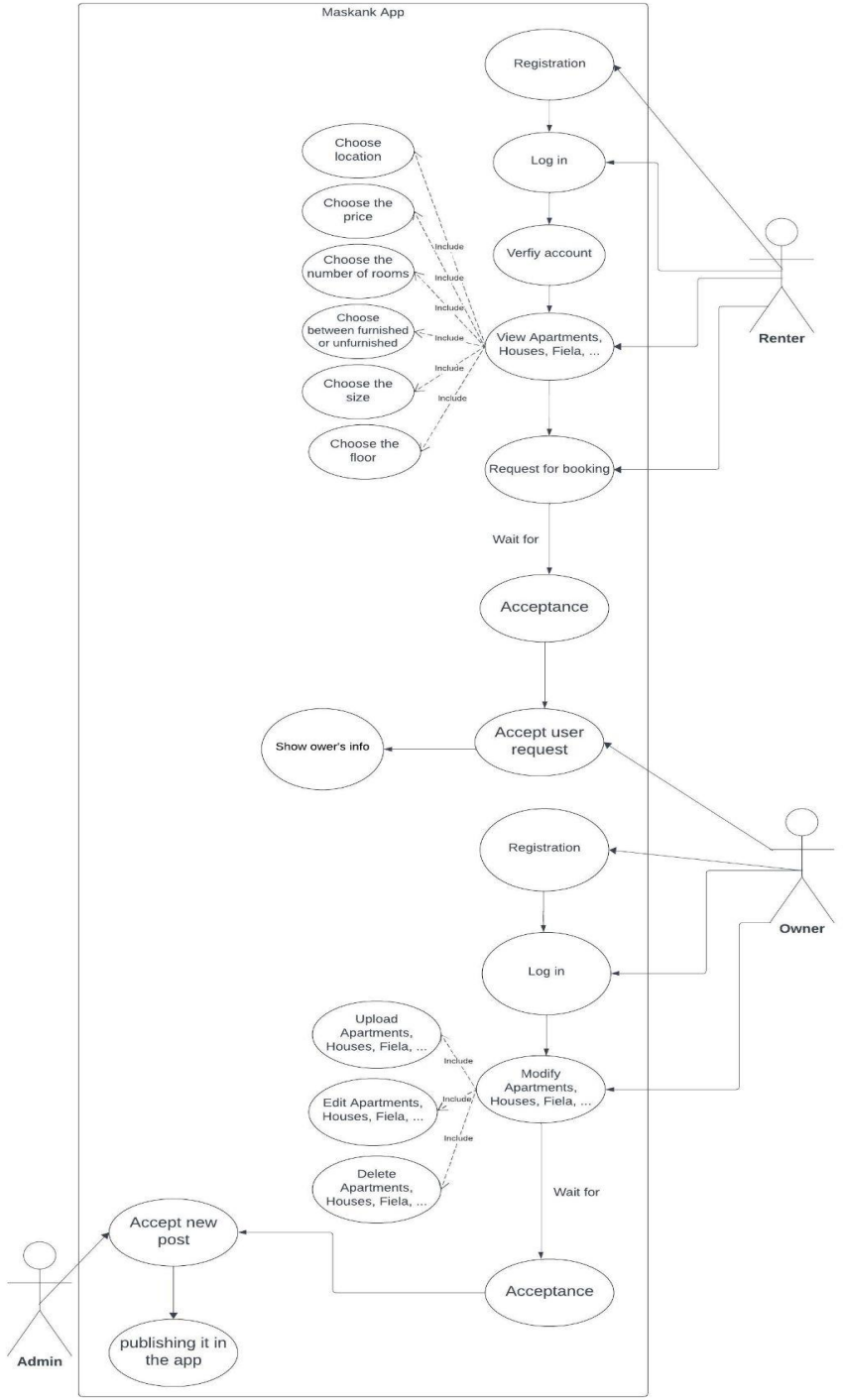
This entity includes property owners with attributes such as owner_name, username, phone, password, national_id, image, and status. Owners can upload and manage multiple posts (Posts).

Admins

This entity details the property listings with attributes like description, size, images, price, purpose, bedrooms, bathrooms, booked, status, created_at, floor, and city. Posts are managed by owners and can be requested by renters. Additionally, admins can manage these posts.

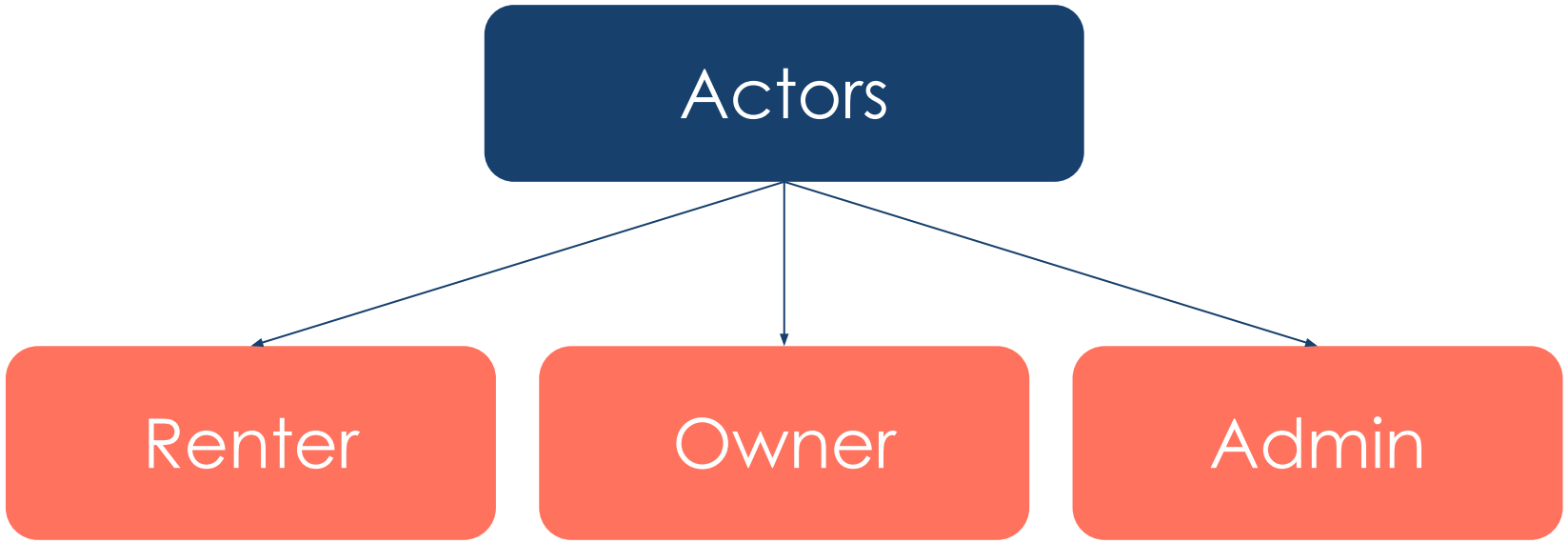
System Designs

Use Case Diagram



System Designs

Use Case Diagram



System Designs

Use Case Diagram

Renter Login/Registration

- Renter visits the app.
- If the renter does not have an account, they can register
- If the renter already has an account, they can log in using their credentials.

Renter Apartment Search

- After logging in, the renter can view a list of available apartments.
- The renter can filter the apartments by location, price range, and other criteria.
- The renter can select an apartment they are interested in.

Renter Apartment Booking

- The renter can send a booking request for the selected apartment to the owner.
- The renter waits for the owner to accept or reject the booking request.

System Designs

Use Case Diagram

Owner Login/Registration

- The owner visits the app.
- If the owner does not have an account, they can
- If the owner already has an account, they can log in using their credentials.

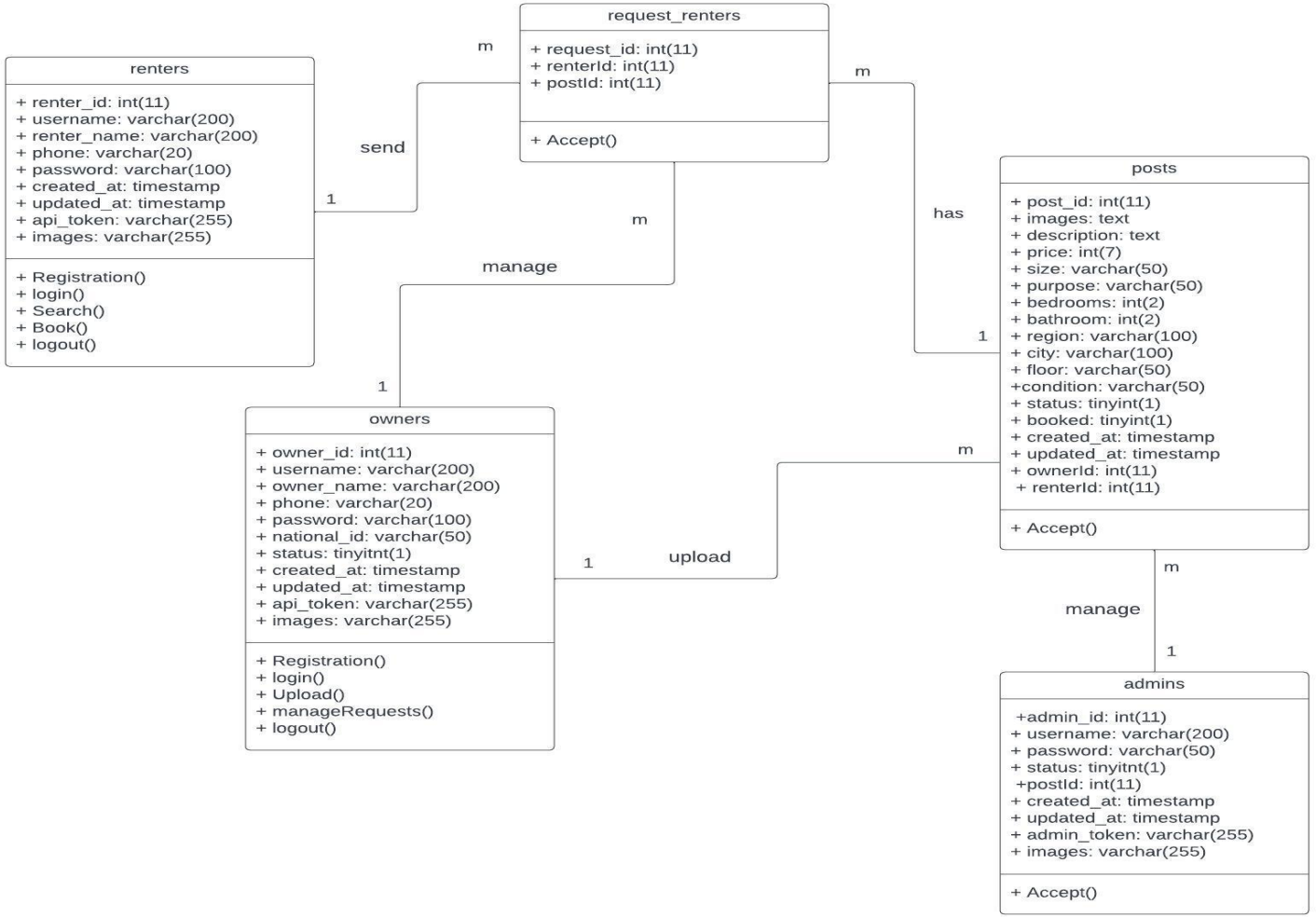
Owner Apartment Posting

- After logging in, the owner can create a new apartment listing.
- The listing includes details about the apartment, such as location, price, amenities, and availability.
- The owner submits the apartment listing for admin approval.

Admin Apartment Approval

- The admin reviews the apartment listing submitted by the owner.
- The admin can either approve or reject the listing.
- If the listing is approved, it becomes visible to renters in the app.
- If the listing is rejected, the owner is notified and can make the necessary changes.

System Designs - Class Diagram



System Designs

Class Diagram

This Class Diagram illustrates the detailed structure and relationships within a system involving renters, owners, admins, and posts, highlighting their interactions and key attributes.

Renters

- Attributes: renter_id, username, renter_name, phone, password, created_at, updated_at, api_token, images.
- Operations: Registration(), login(), Search(), Book(), logout().
- Relationships: Renters can send requests (many-to-many relationship with request_renters), and manage posts (one-to-many relationship with posts).

Owners

- Attributes: owner_id, username, owner_name, phone, password, national_id, status, created_at, updated_at, api_token, images.
- Operations: Registration(), login(), Upload(), manageRequests(), logout().
- Relationships: Owners can upload and manage posts (one-to-many relationship with posts).

System Designs

Admins

- Attributes: admin_id, username, password, status, postId, created_at, updated_at, admin_token, images.
- Operations: Accept().
- Relationships: Admins can manage posts (one-to-many relationship with posts).

Requset_Renters

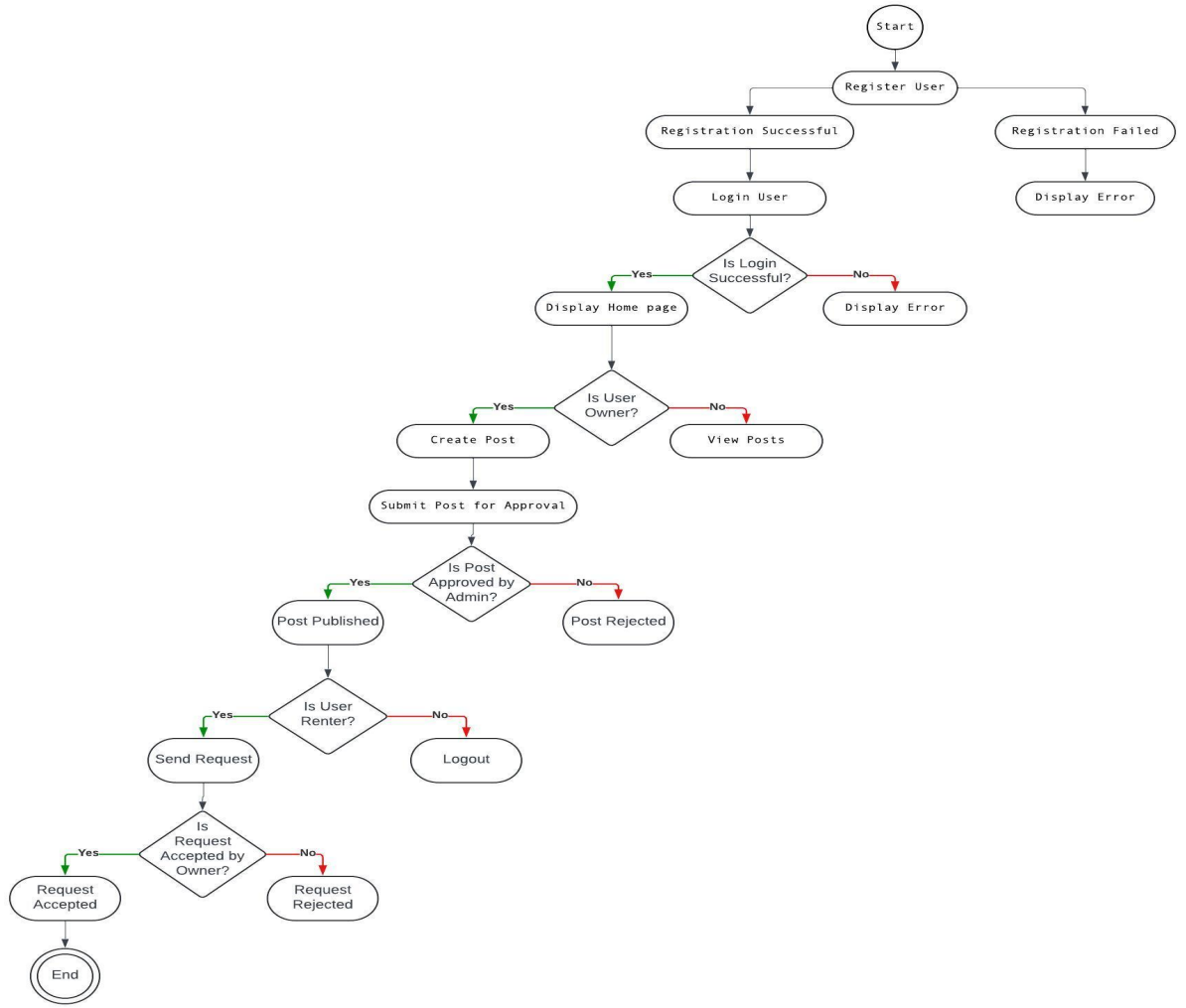
- Attributes: request_id, renterId, postId.
- Operations: Accept().
- Relationships: Handles the many-to-many relationship between renters and posts.

Posts

- Attributes: post_id, images, description, price, size, purpose, bedrooms, bathrooms, region, city, floor, condition, status, booked, created_at, updated_at, ownerId, renterId.
- Operations: Accept().
- Relationships: Posts are linked to owners and renters, can be managed by admins, and requested by renters (many-to-many relationships with request_renters).

System Designs

Activity Diagram



System Designs

Activity Diagram

The activity diagram illustrates the process flow encompassing user registration, login, post creation, and request handling. It begins with the user registration phase, followed by the login procedure. Once logged in, users can create posts, which then undergo an approval process managed by an admin.

User Registration and Login

- The process starts with users registering their information.
- After successful registration, users proceed to login using their credentials.
- Decision points determine whether the login is successful or if users need to retry.

Post Creation and Approval

- Once logged in, users (either owners or renters) can create posts.
- Created posts are submitted for admin approval.
- The admin verifies the posts, leading to either acceptance or rejection.
- Decision points at this stage dictate the flow based on the admin's approval status.

System Designs

Activity Diagram

The activity diagram illustrates the process flow encompassing user registration, login, post creation, and request handling. It begins with the user registration phase, followed by the login procedure. Once logged in, users can create posts, which then undergo an approval process managed by an admin.

User Status and Request Handling

- The flowchart considers different user statuses: owners and renters.
- Owners can manage and upload new posts, while renters can search and request bookings.
- Requests made by renters are handled by owners, who can either accept or reject them.
- Decision points guide the process flow based on the request outcomes

Paths and Outcomes

- Green arrows in the diagram represent positive paths, indicating successful actions and approvals.
- Red arrows represent negative paths, denoting errors, rejections, or the need for retries.
- The flowchart concludes with various outcomes depending on the user actions and statuses, such as successful bookings, rejected posts, or prompted retries

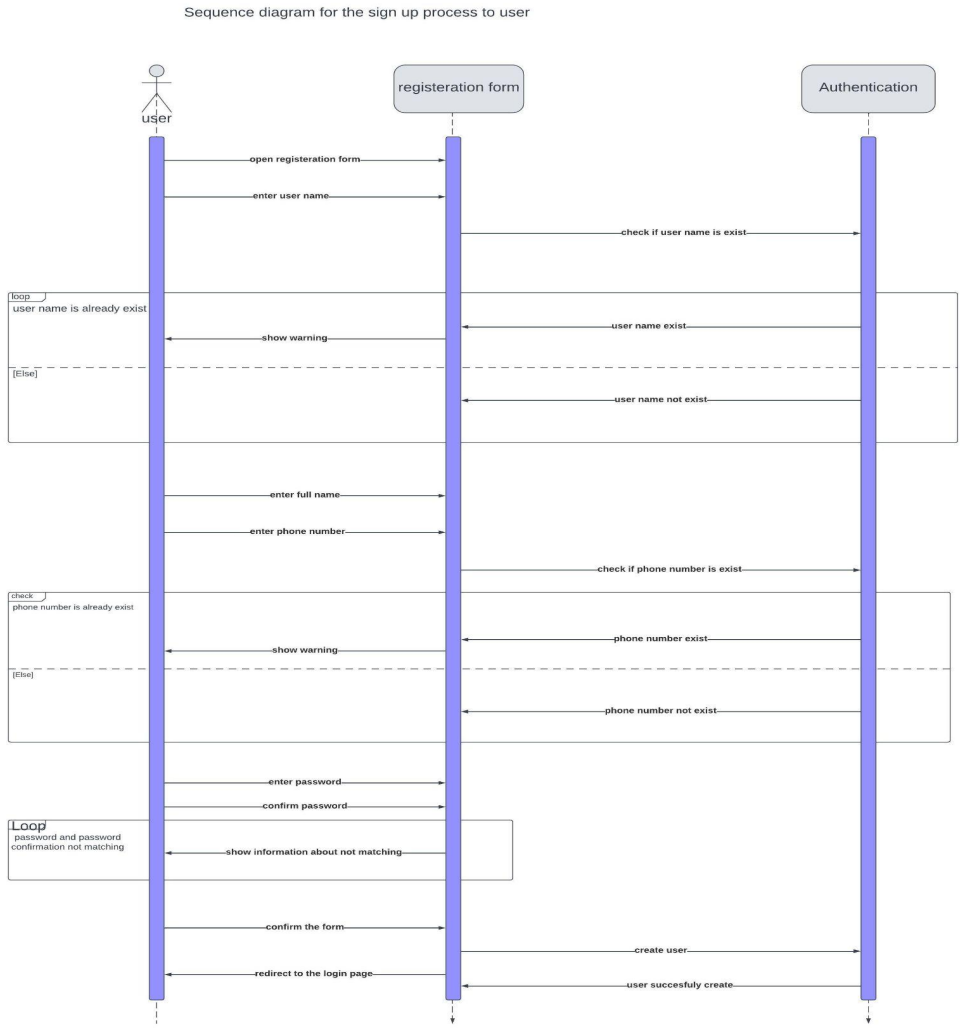
System Designs

Sequence diagram for sign up process to user

1. The user opens the registration form in the app.
2. The user enters a username, and the app checks if the username already exists in the database.
3. If the username does not exist, the user can proceed to enter their full name and phone number.
4. The app checks if the phone number already exists in the database.
5. If the phone number does not exist, the user can proceed to enter a password and confirm the password.
6. The app checks if the confirm password matches the password entered.
7. If the confirm password matches the password, the app creates a new user account in the database and confirms the registration as successful.
8. If the confirm password does not match the password, the app shows a warning and asks the user to re-enter the confirm password.
9. If the username or phone number already exists, the app shows a warning and asks the user to enter a different username or phone number.

System Designs

Sequence diagram for sign up process to user



System Designs

Sequence diagram for sign up process to owner

1. The owner opens the registration form.

2. The owner enters a username.

3. The app checks if the username already exists in the database.

4. If the username doesn't exist, the user is allowed to proceed.

5. The owner enters their full name and phone number.

6. The app checks if the phone number already exists in the database.

7. If the phone number doesn't exist, the user is allowed to proceed.

8. The owner enters a password and confirms the password.

9. The app checks if the confirm password matches the password.

10. If the passwords match, the owner is prompted to upload a photo.

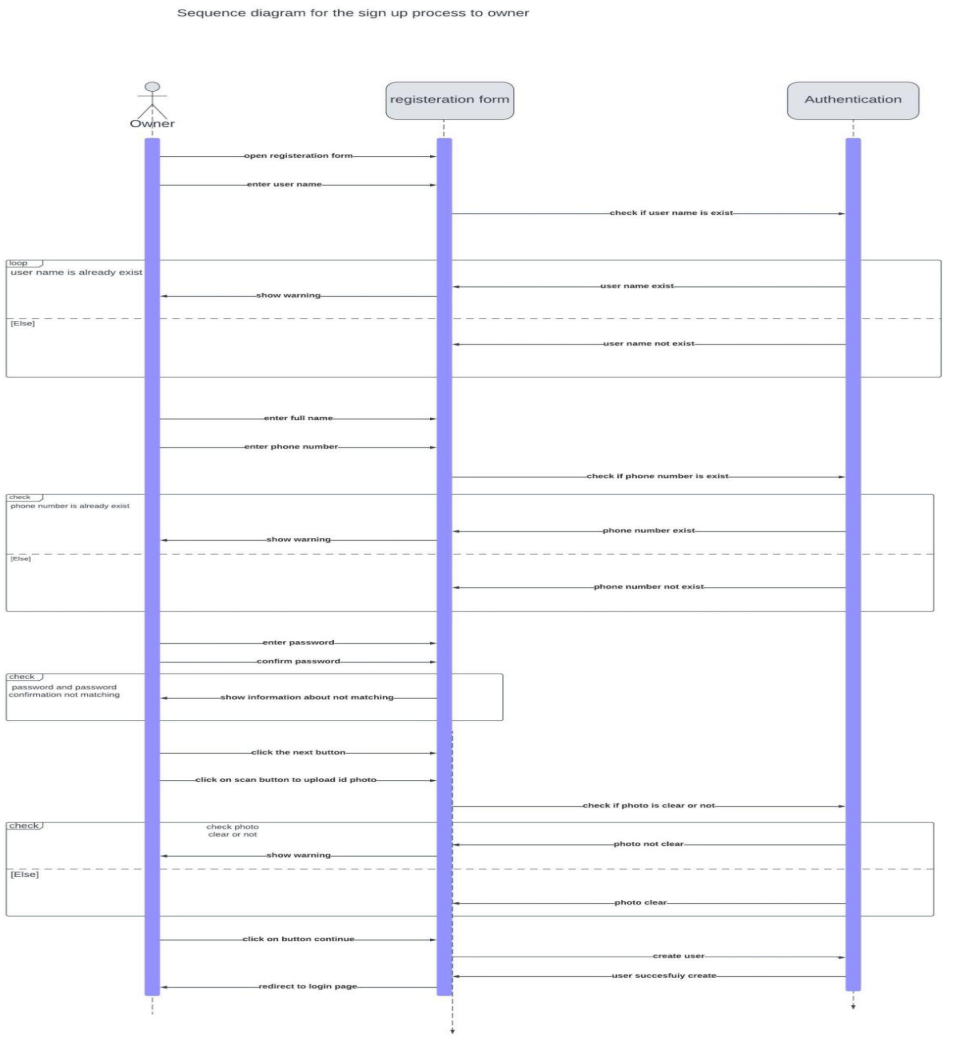
11. The app analyzes the quality of the uploaded photo.

12. If the photo is not clear, the app asks the owner to upload a clearer photo

13. If the photo is clear, the app creates a new owner account in the database with the user's information and the profile photo.

System Designs

Sequence diagram for sign up process to owner



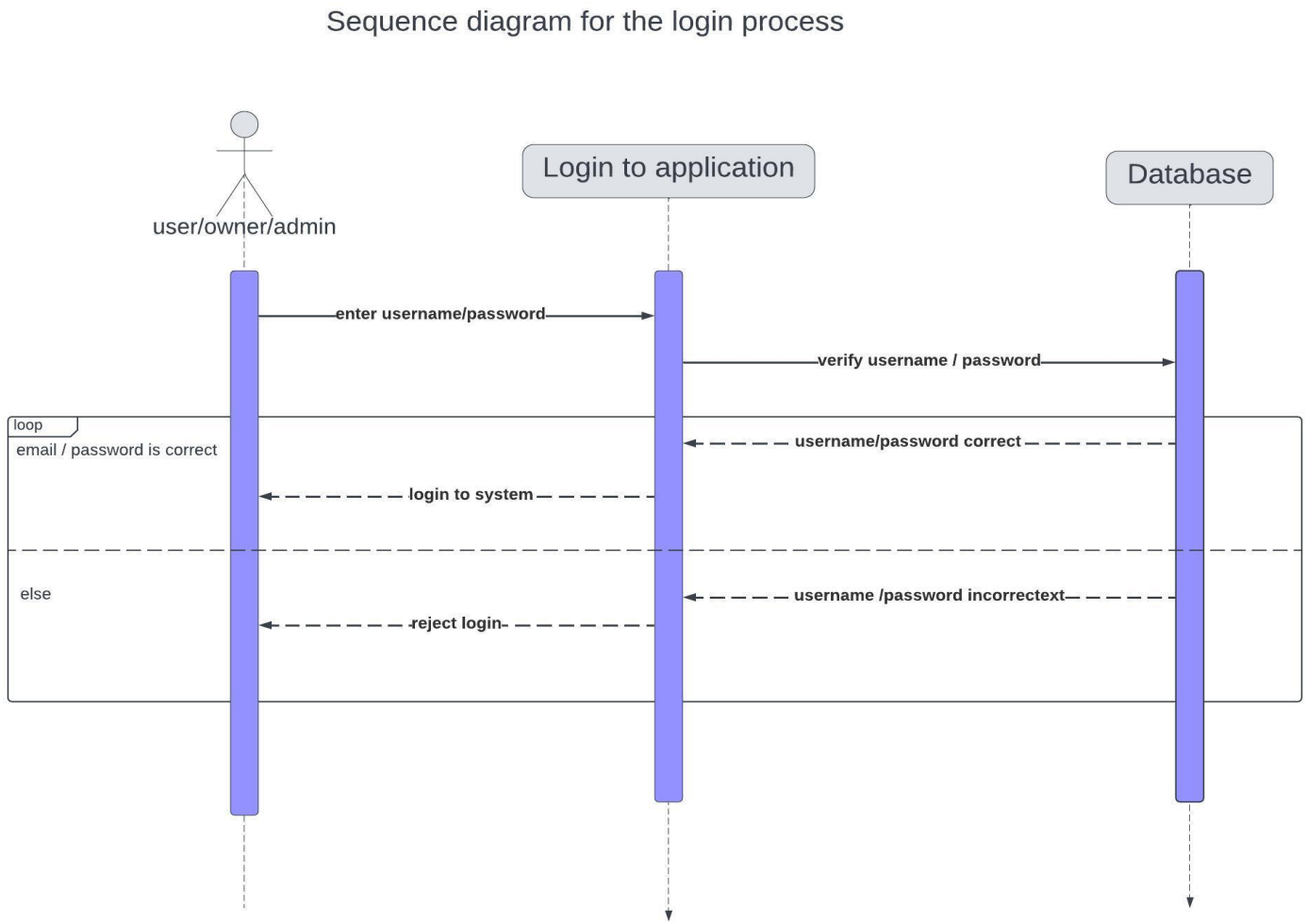
System Designs

Sequence diagram for login process

1. The user opens the login page.
2. The user enters their username and password.
3. The app checks the username and password against the information stored in the database.
4. If the username and password match, the user is logged in and granted access to the app.
5. If the username and password do not match, the app shows an error message and asks the user to try again.

System Designs

Sequence diagram for login process



System Designs

Sequence diagram between admin and owner

1. The owner opens the application and signs in with their credentials.

2. If the authentication is successful, the app shows the home page to the owner.

3. The owner creates a new post in the app.

4. The app prompts the owner to provide post details and upload media (e.g., images, videos).

5. The app notifies the admin about the new post submission.

6. The admin checks the details and media of the new post.

7. The app displays the post to the admin for review.

8. The admin has two options:

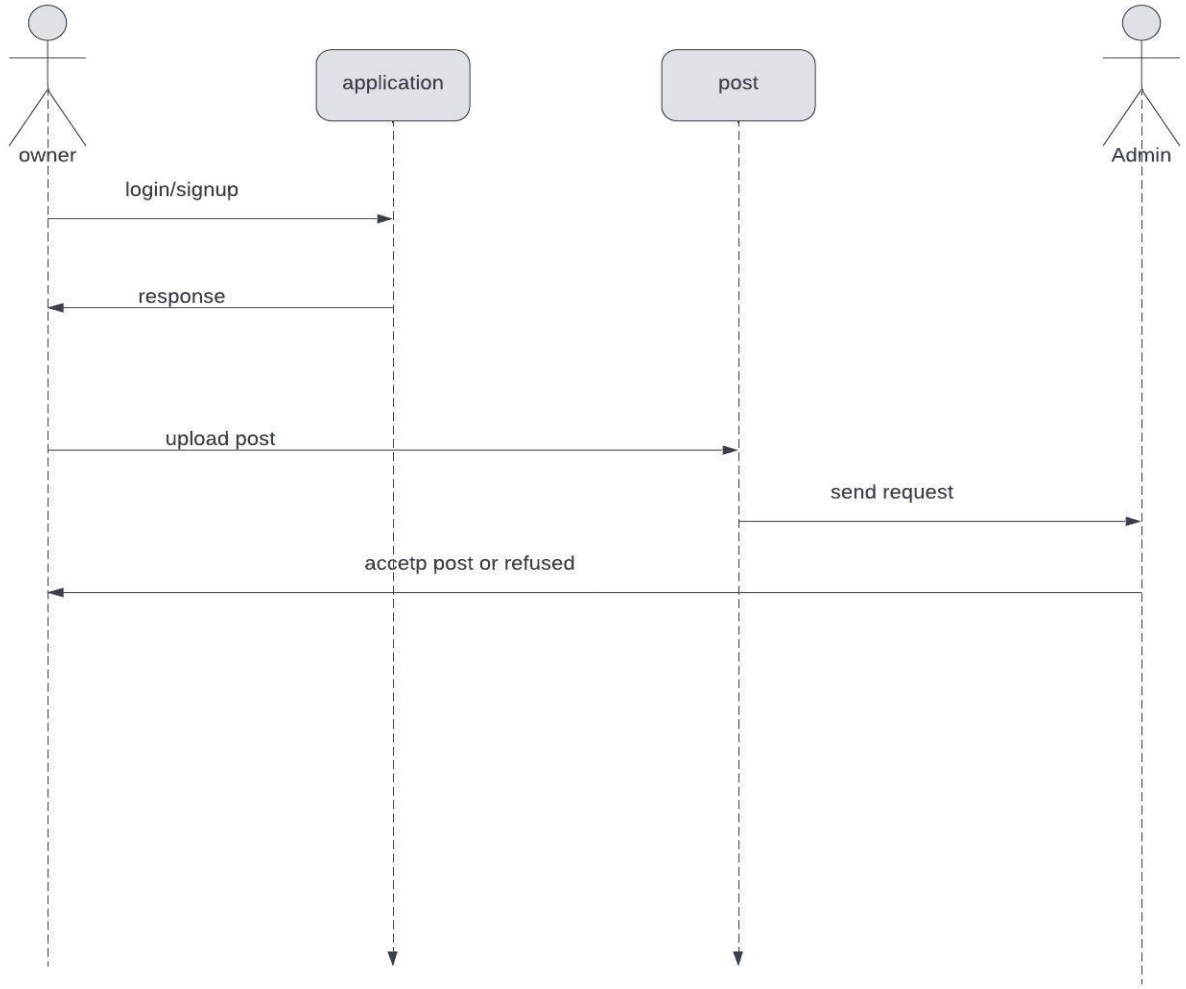
9. If the admin approves the post, they accept it.

10. If the admin rejects the post, they reject it.

11. Depending on the admin's decision, the app notifies the owner of the post's acceptance or rejection.

System Designs

Sequence diagram between admin and owner



System Designs

Sequence diagram between user and owner

1. The user opens the application and signs in with their credentials.

2. If the authentication is successful, the app shows the home page to the user.

3. The user searches for an apartment within the app.

4. The app retrieves apartment listings based on the user's search criteria from the database.

5. The app displays the search results to the user.

6. The user selects a specific apartment from the search results.

7. The app retrieves the details of the selected apartment from the database.

8. The app displays the apartment details to the user.

9. The user contacts the owner of the apartment to book it.

10. The app notifies the owner of the booking request.

11. The owner checks the booking request and the availability of the apartment.

12. The app displays the booking request details to the owner.

13. The owner has two options:

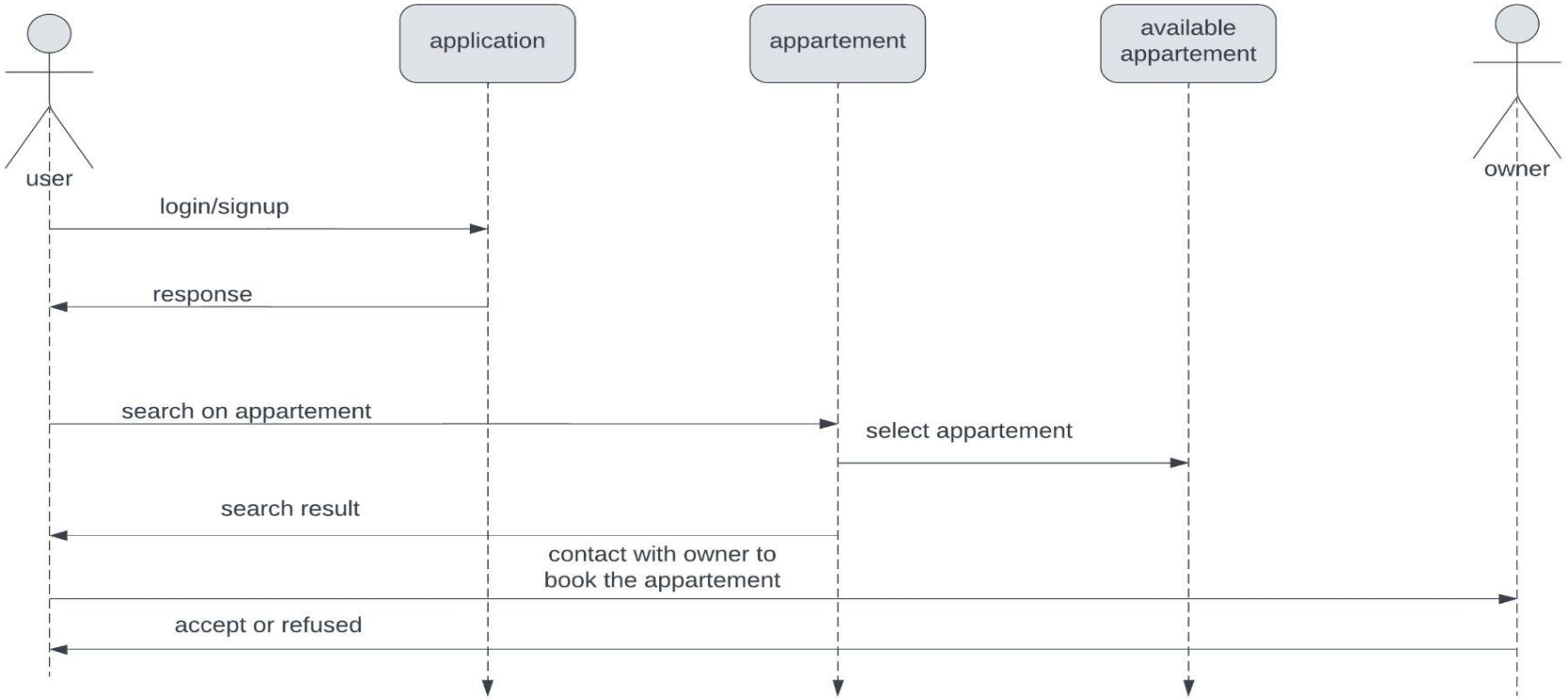
14. If the owner accepts the booking request, they accept it.

15. If the owner rejects the booking request, they reject it.

16. Depending on the owner's decision, the app notifies the user of the booking request's acceptance or rejection.

System Designs

Sequence diagram between user and owner



Project Aspects

Project Aspects

Developing Methodology

Agile Methodology

Organizes project phases in an iterative and incremental cycle.

Enables feedback-centric, adaptable software development.

Promotes collaboration.

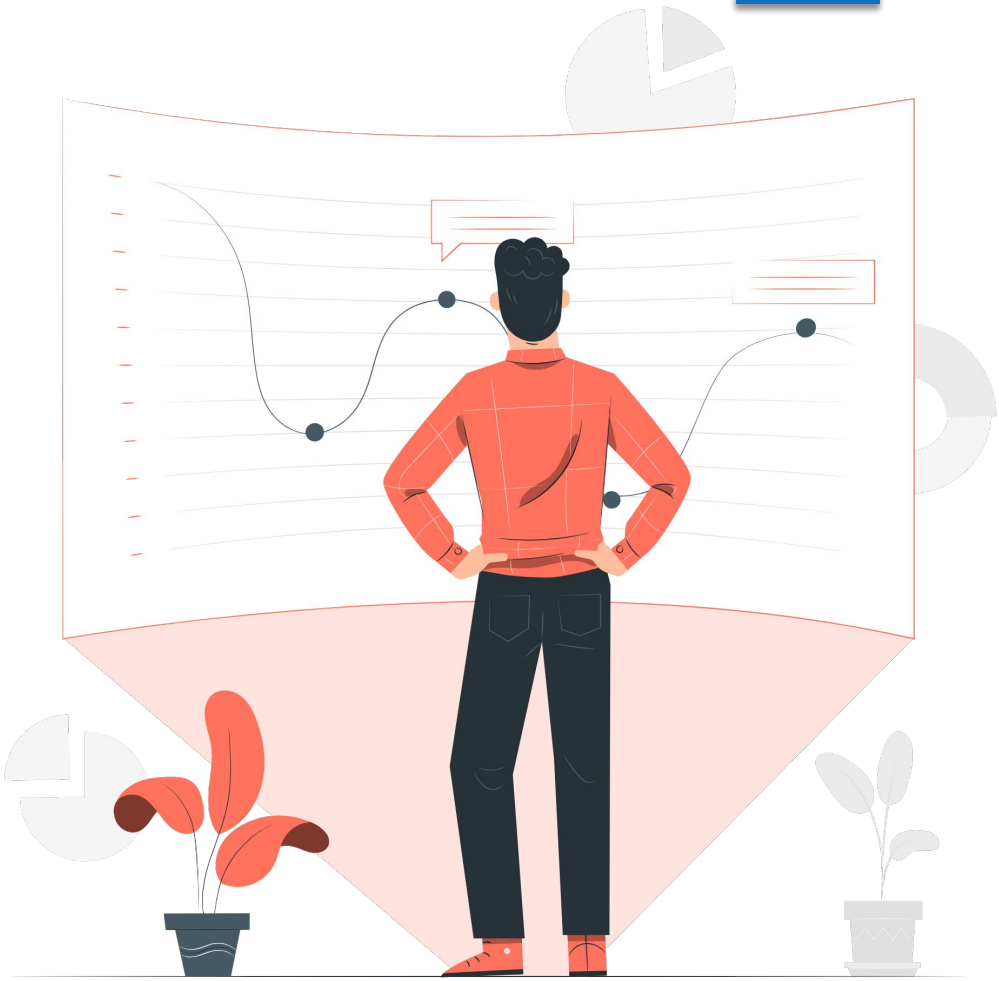
Continuous learning, and frequent value delivery.

Developing Methodology

Agile Methodology

Adopting Agile for our MASKANK application project will make our development plan flexible and open to changes. This approach allows us to quickly identify and correct mistakes or bugs, ensuring a higher quality product and faster delivery of features.

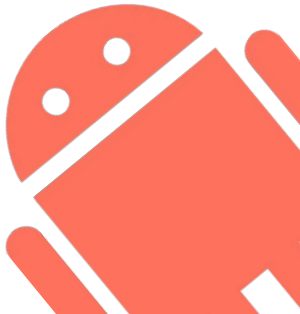
PROJECT COMPONENTS



project component

Flutter Application

The FLutter application is the main access point to our real estate project, where users can access all the features we provide. These features include receiving instant listing notifications, getting bespoke property recommendations, searching for properties, scheduling viewings.



PERFORMANCE AND ANALYTICS DEVELOPMENT

PERFORMANCE AND ANALYTICS DEVELOPMENT

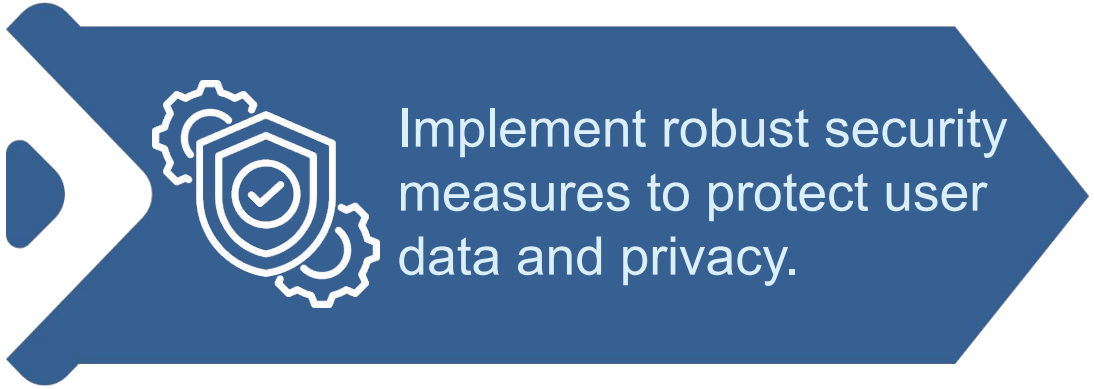
- Advanced-Data Analytics:

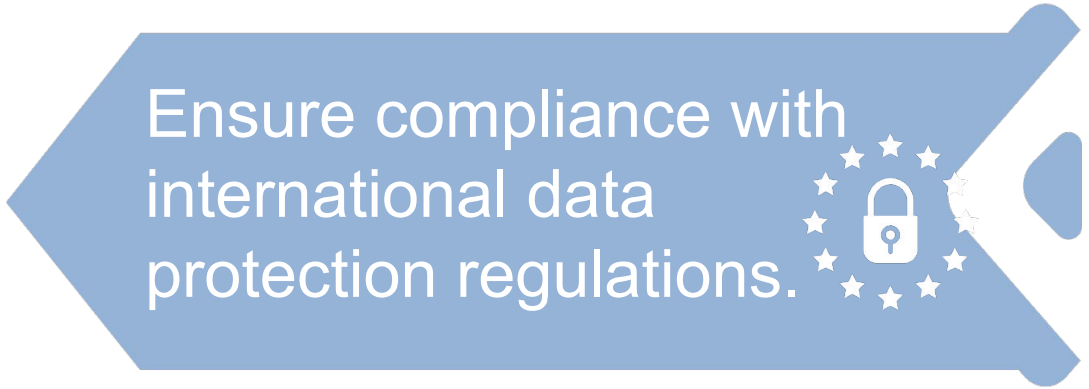
- Integrate data analytics tools to assess property performance and user preferences.
- Utilize predictive analytics to forecast market trends and inform strategic decision-making.

Security and Privacy Enhancements

Security and Privacy Enhancements

Enhanced Data Protection:





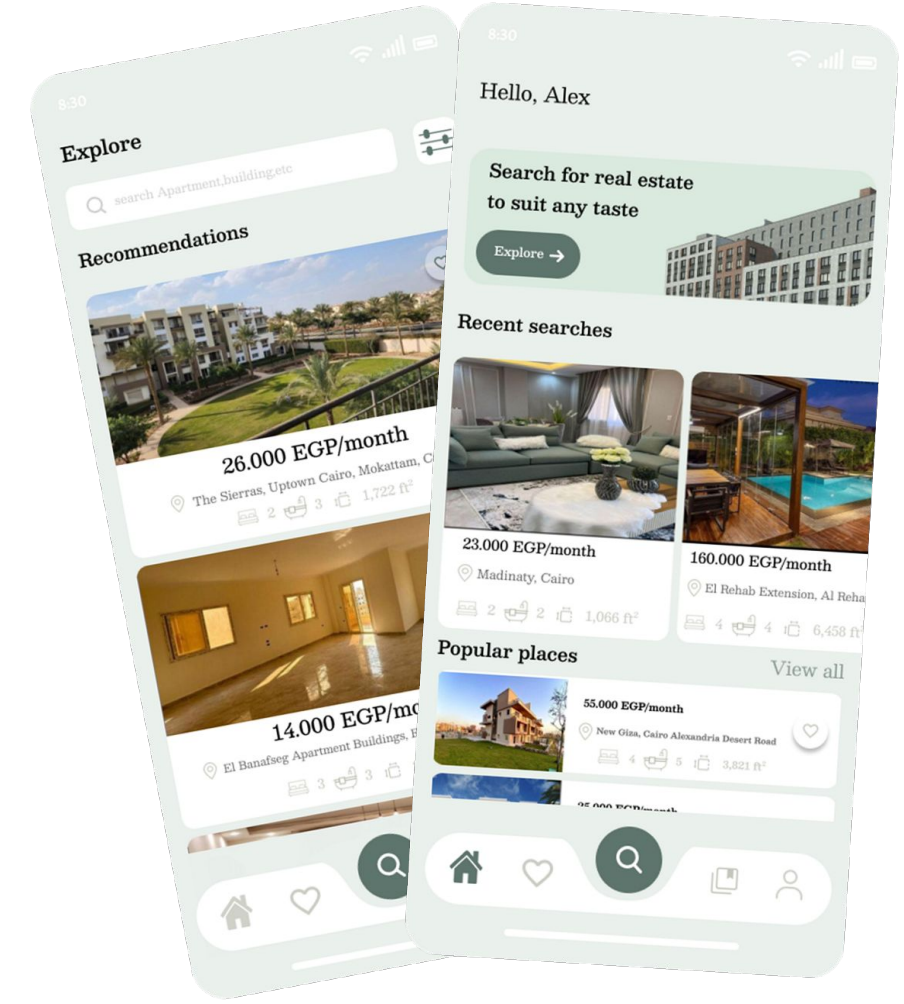
User Experience and Interface

User Experience and Interface

Customizable Interface:

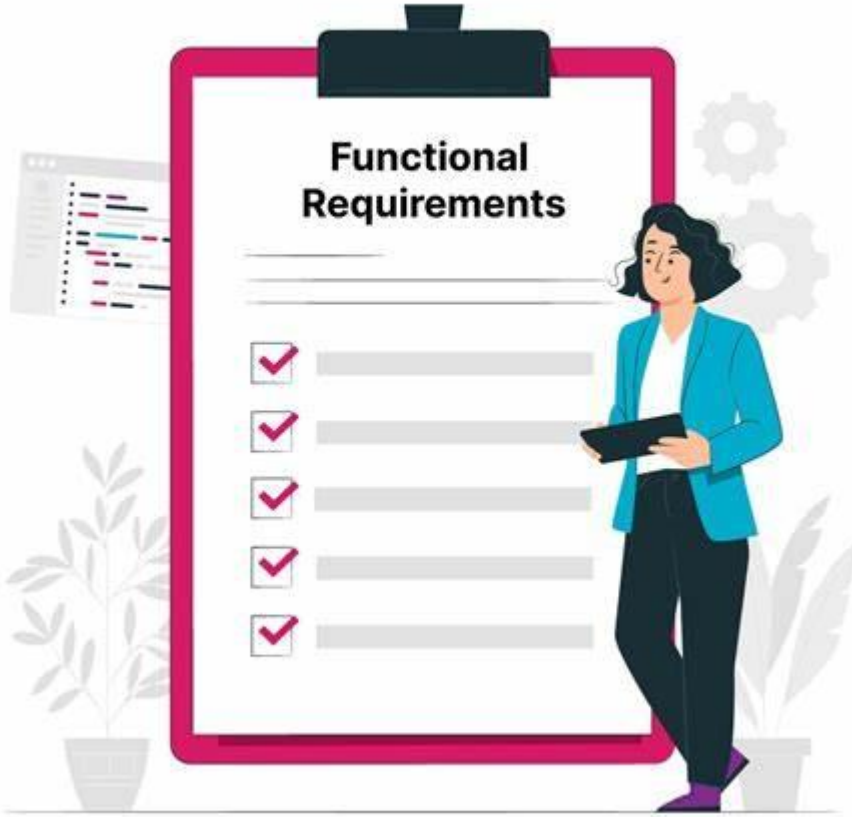
- Offer a customizable user interface that allows users to adjust the layout and settings according to their preferences.

- Improve user engagement by providing a personalized app experience.



Requirement Analysis

1-Functional Requirements



Owner

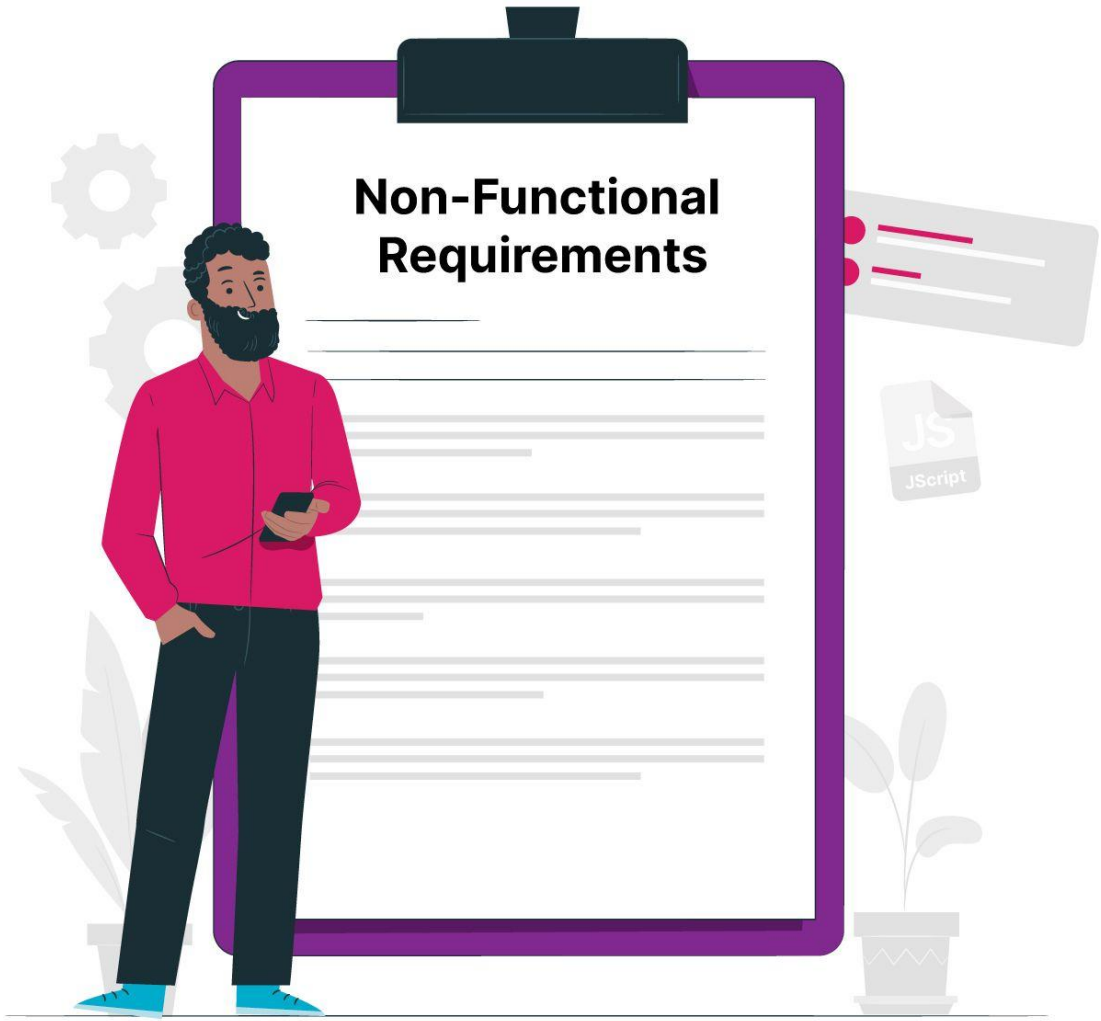
- logging in/out the system
- can add apartment
- All requests can be viewed
- View profile

Renter

- logging in/out the system
- can add request
- All apartments can be viewed
- View profile
- Can add the apartments to favourites
- Search for apartments
- Filter for apartments
- Display the apartment page

Admin

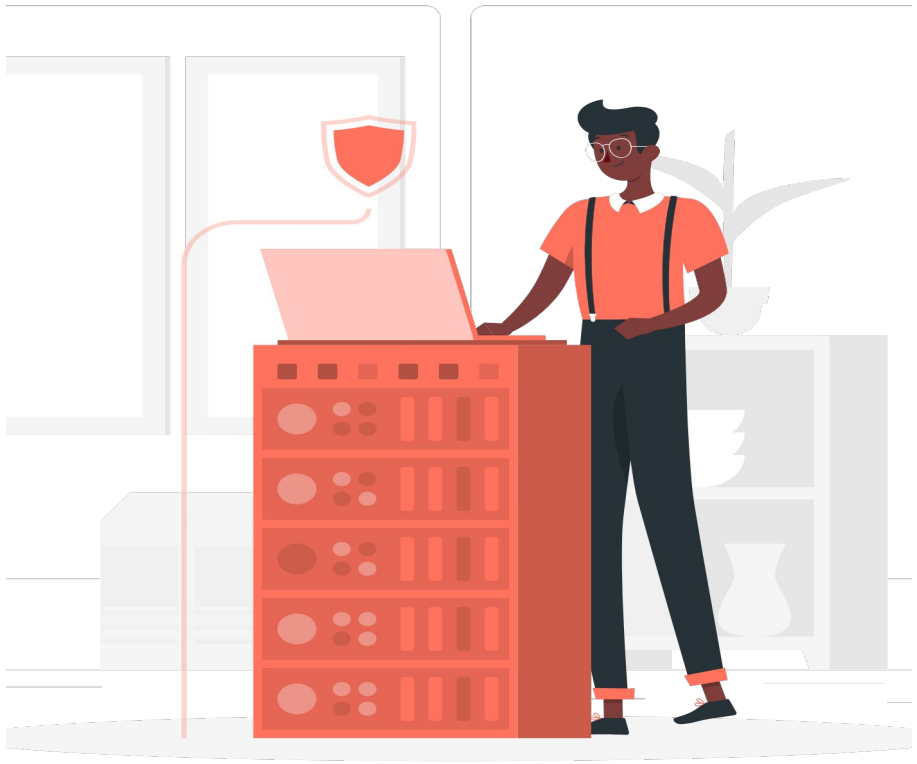
- logging in/out the system
- control the whole system
- can delete and insert apartments
- maintain the system
- manage renter and owner inside the system
- accept posts



2- Non-Functional Requirements

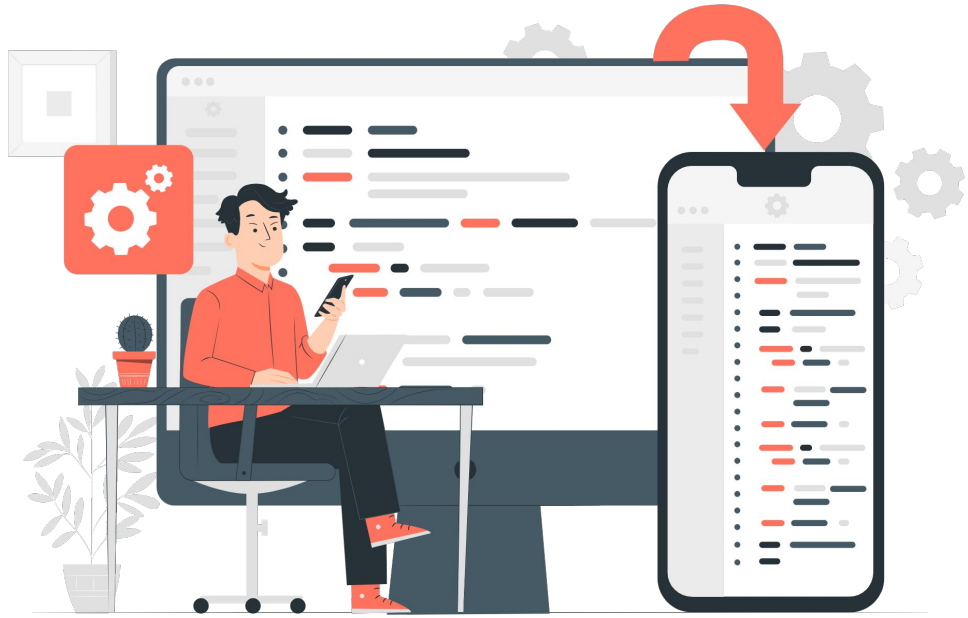
2- Non-Functional Requirements

- Security
- Performance
- Reliability
- Usability



System Implementation

1- Mobile Application (Flutter)



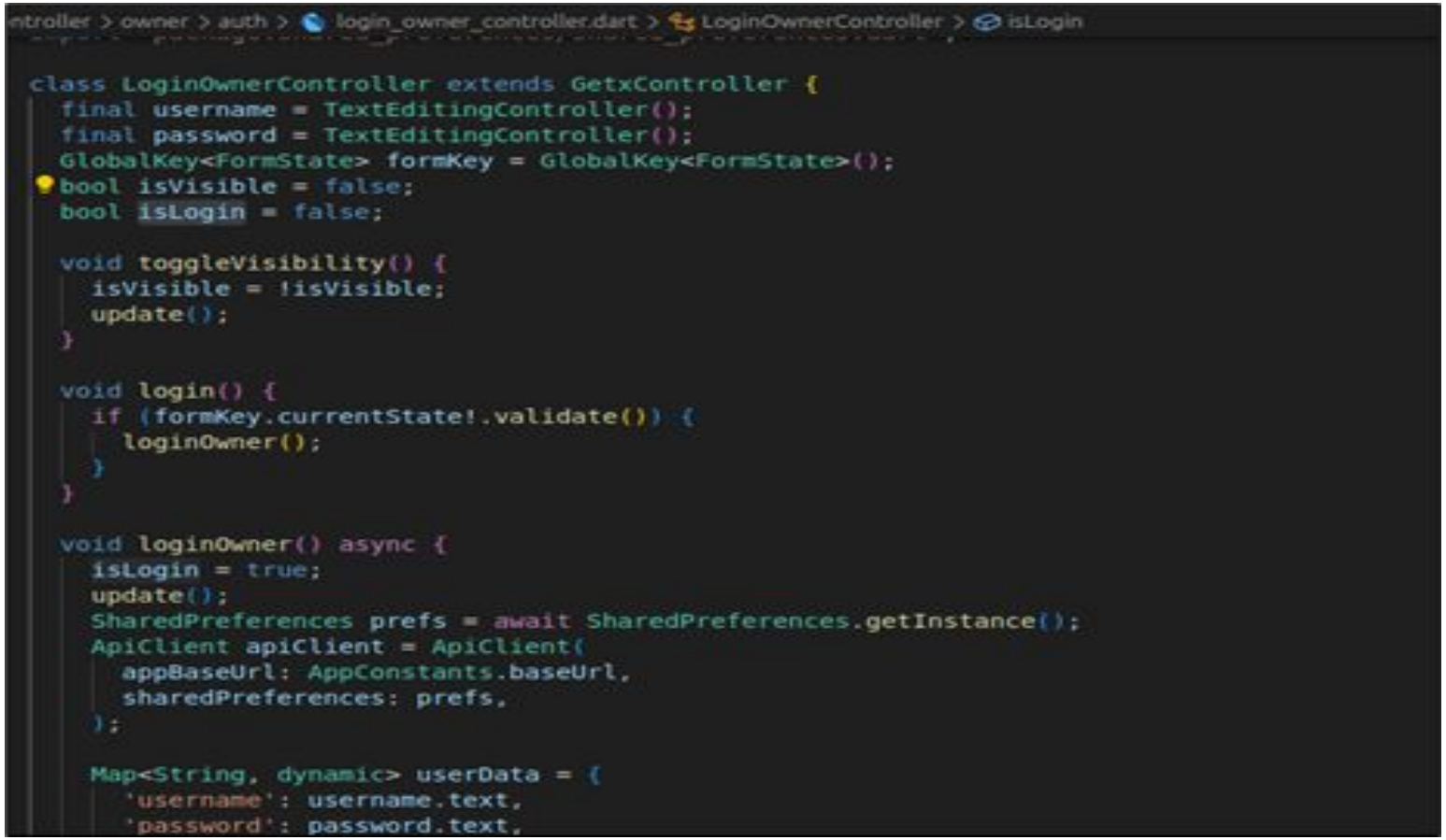
1- Mobile Application (Flutter)

1- Logic of signup with Getx

```dart
class RegesterOwnerController extends GetxController {
  final usernameController = TextEditingController();
  final fullnameController = TextEditingController();
  final phoneController = TextEditingController();
  final passwordController = TextEditingController();
  final confirmpasswordController = TextEditingController();
  GlobalKey<FormState> formkey = GlobalKey<FormState>();
  bool isRegister = false;
  bool isvisible = false;
  void changeVisible() {
    isvisible = !isvisible;
    update();
  }

  void registerUser() async {
    isRegister = true;

    update();
    SharedPreferences prefs = await SharedPreferences.getInstance();

    ApiClient apiClient = ApiClient(
      appBaseUrl: AppConstants.baseUrl,
      sharedPreferences: prefs,
    );

    Map<String, dynamic> userData = {
      'username': usernameController.text,
      'password': passwordController.text,
      'owner_name': fullnameController.text,
      'phone': phoneController.text,
      'image': 'test.jpg'
```

Mobile Application (Flutter)

2- Logic login of owner with GetX controller :



Mobile Application (Flutter)

maskank.dart > _MaskankAppState > build

```dart
class _MaskankAppState extends State<MaskankApp> {
  late bool _isConnected = false;

  @override
  void initState() {
    super.initState();
    init();
    _checkConnectivity();
    Connectivity().onConnectivityChanged.listen((result) {
      setState(() {
        _isConnected = result != ConnectivityResult.none;
      });
    });
  }

  Future<void> _checkConnectivity() async {
    var connectivityResult = await Connectivity().checkConnectivity();
    setState(() {
      _isConnected = connectivityResult != ConnectivityResult.none;
    });
  }

  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      title: 'Maskank App',
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        body: _isConnected ? LoginOwner() : const NoConnection(),
      ), // Scaffold
    ); // GetMaterialApp
```

Ln 47, Col 40 Spaces: 2 UTF-8 LF {} Dart

3-Here logic code to show if network coneetion or no

Mobile Application (Flutter)

controller > owner > auth > regestter_owner_controller.dart > ...

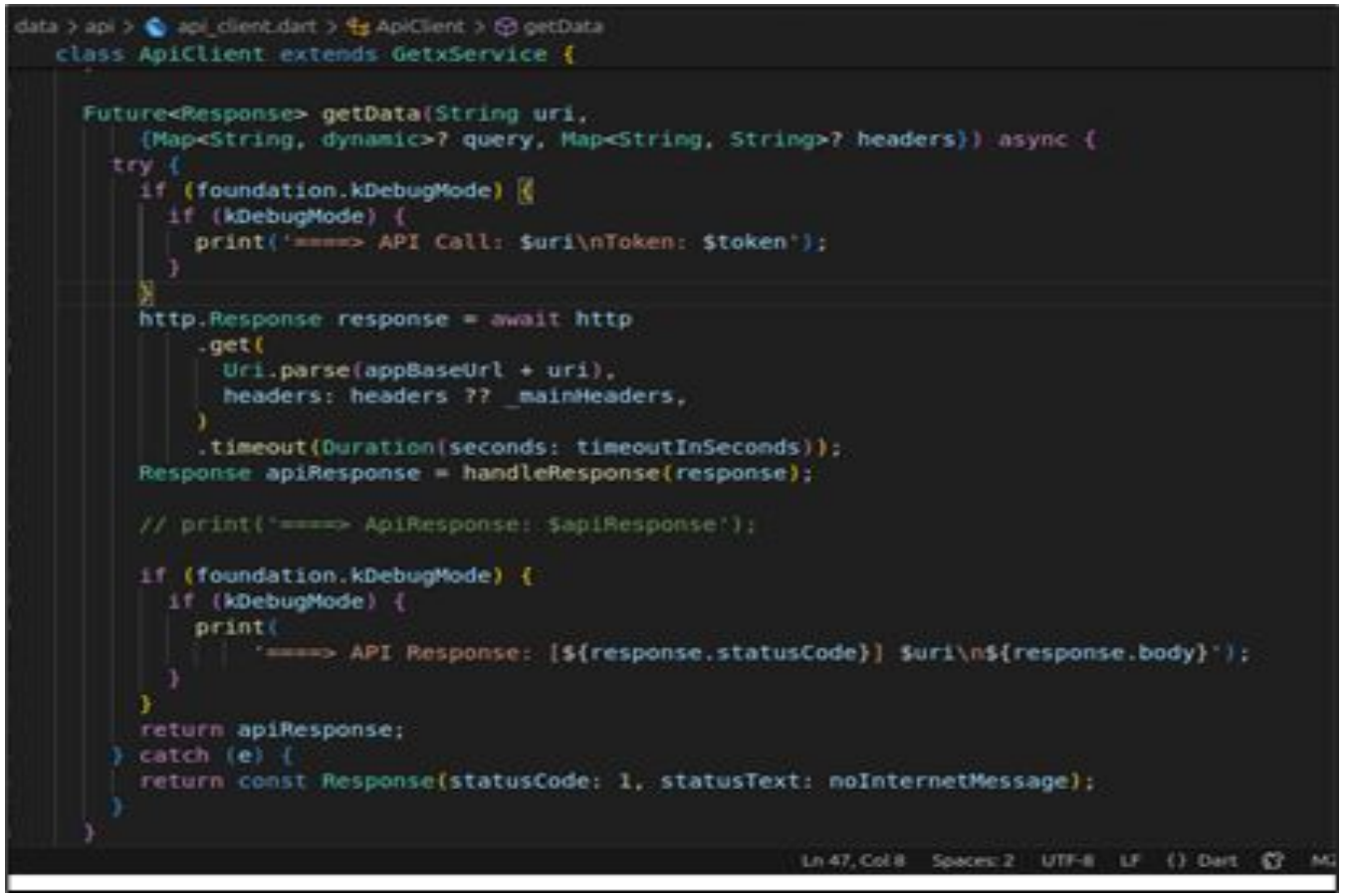
```dart
class RegesterOwnerController extends GetxController {
  void registerUser() async {
    //

    Response response =
        await apiClient.postAuthData(AppConstants.ownerRegisterUrl, userData);
    // Check the response status
    if (response.statusCode == 200) {
      isRegister = false;

      update();
      print('Owner registered successfully');
      isRegister = false;
      update();
      Fluttertoast.showToast(
          msg: "Owner registered successfully",
          toastLength: Toast.LENGTH_SHORT,
          gravity: ToastGravity.BOTTOM,
          timeInSecForIosWeb: 1,
          backgroundColor: Colors.green,
          textColor: Colors.white,
          fontSize: 16.0);
      Get.to(() => const ScanScreen());
    } else {
      isRegister = false;
      update();
      if (response.body["message"] == "Owner already exists") {
        Fluttertoast.showToast(
            msg: "Owner already exists",
            toastLength: Toast.LENGTH_SHORT,
            gravity: ToastGravity.BOTTOM,
            timeInSecForIosWeb: 1,
```

Ln 1, Col 1 Spaces: 2 UTF-8 LF {} Dart

4-Here we used post method of API to send data to backend :

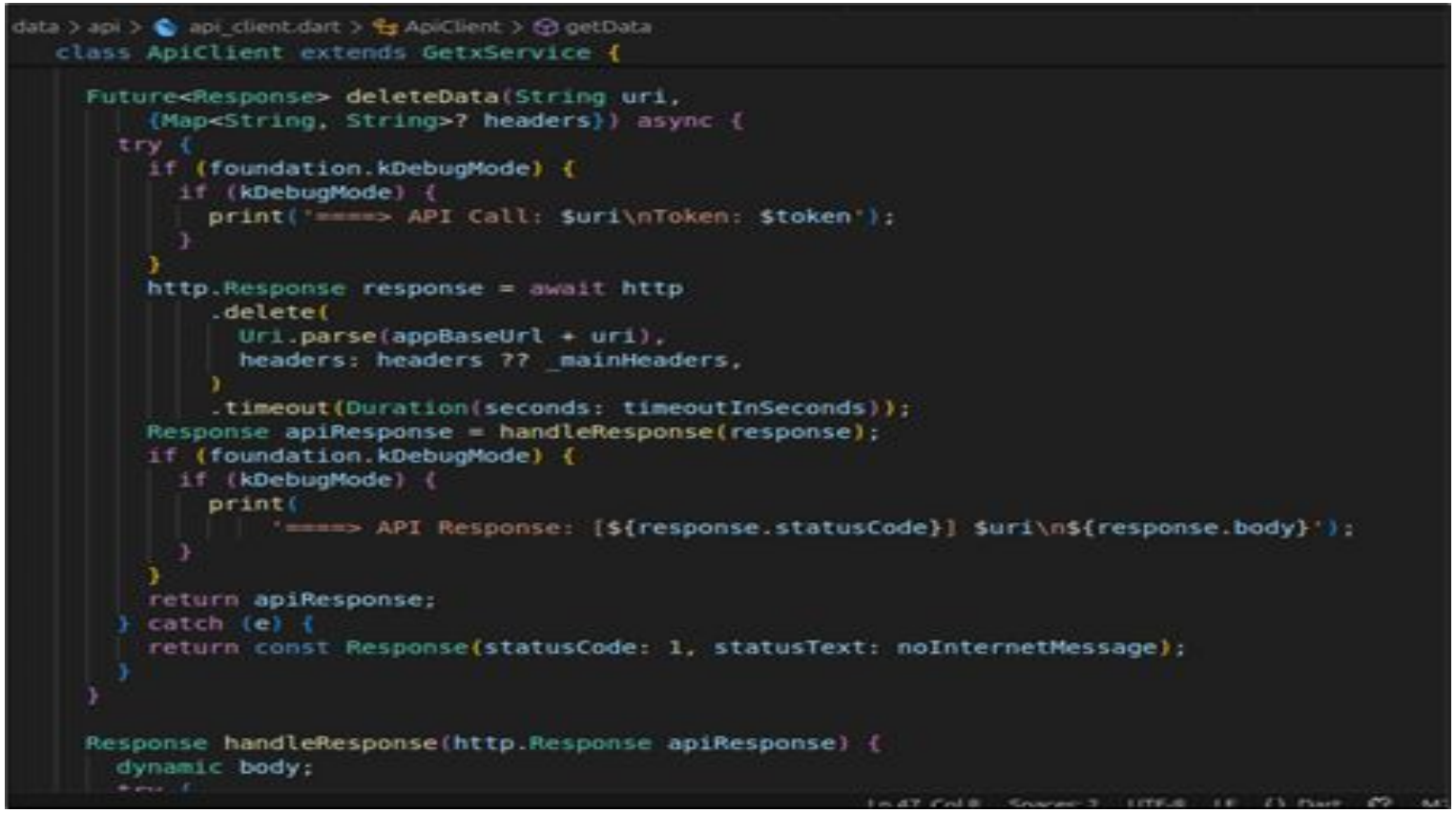
Mobile Application (Flutter)

5-We make API method helper here like post and get && put to share use:



Mobile Application (Flutter)

6-Delete method API:



```
data > api > api_client.dart > ApiClient > getData
class ApiClient extends GetxService {
  Future<Response> deleteData(String uri,
      {Map<String, String>? headers}) async {
    try {
      if (foundation.kDebugMode) {
        if (kDebugMode) {
          print('====> API Call: $uri\nToken: $token');
        }
      }
      http.Response response = await http
          .delete(
            Uri.parse(appBaseUrl + uri),
            headers: headers ?? _mainHeaders,
          )
          .timeout(Duration(seconds: timeoutInSeconds));
      Response apiResponse = handleResponse(response);
      if (foundation.kDebugMode) {
        if (kDebugMode) {
          print(
              '====> API Response: [${response.statusCode}] $uri\n${response.body}');
        }
      }
      return apiResponse;
    } catch (e) {
      return const Response(statusCode: 1, statusText: noInternetMessage);
    }
  }

  Response handleResponse(http.Response apiResponse) {
    dynamic body;
```

Mobile Application (Flutter)

```
data > api > api_client.dart > ApiClient > getData
class ApiClient extends GetxService {
  }

  Future<Response> postData(String uri, dynamic body,
      {Map<String, String>? headers}) async {
    try {
      if (foundation.kDebugMode) {
        if (kDebugMode) {
          print('====> API Call: $uri\nToken: $token');
        }
        if (kDebugMode) {
          print('====> API Body: $body');
        }
      }
      http.Response response = await http
          .post(
            Uri.parse(appBaseUrl + uri),
            body: jsonEncode(body),
            headers: headers ?? _mainHeaders,
          )
          .timeout(Duration(seconds: timeoutInSeconds));
      Response apiResponse = handleResponse(response);
      if (foundation.kDebugMode) {
        if (kDebugMode) {
          print(
              '====> API Response: [${response.statusCode}] $uri\n${response.body}');
        }
      }
      return apiResponse;
    } catch (e) {
      return const Response(statusCode: 1, statusText: noInternetMessage);
    }
```

7-This post method :

2-Backend (PHP Laravel)

Backend (PHP Laravel)

1-Register login logout for owner:

```
4 references | 0 implementations
class OwnerController extends Controller
{
    1 reference | 0 overrides
    public function store(createOwnerRequest $request){

        $data = $request->validated();

        if(!empty($data['photo'])){
            $ext = $data['photo']->getClientOriginalExtension();
            $imageName = time().'_'.rand().'.'.$ext;
            $photoName =  $data['photo']->move(public_path().'/OwnerPhoto', $imageName);
            $newName = explode('\\',$photoName);
            $data['photo'] = end($newName);
          }else{  unset($data['photo']);  }


          $owner = Owner::create($data);

          $owner->notify(new EmailVerificationNotification());
          $success['message'] = "Registration successful";
          $success['data'] = new OwnerResource($owner);

          return response()->json($success);
```

```
class OwnerController extends Controller
{

    1 reference | 0 overrides
    public function login(Request $request){
        $owner =Owner::where("username" , $request->input("username"))->first();
        if(!$owner){
            return response()->json(["message"=> "owner not found"] , 401);
        }
        if(!Hash::check($request->input("password") , $owner->password)){
            return response()->json(["message" => "wrong password"] , 401);
        }
       $token = $owner->createToken("Maskank");
        return response()->json(["token" => $token->plainTextToken ,
        "message" => "owner successfully logged in", "owner" => new OwnerResource($owner)] , 200);
    }

    1 reference | 0 overrides
    public function logout(Request $request)
    {
        try{
            $request->user()->currentAccessToken()->delete();
            return response()->json(["message" => "owner successfully logged out"] , 200);
        }
        catch (ValidationException $e) {
            return response()->json(["error" => $e->errors()], 422);
        }
    }
```

Backend (PHP Laravel)

2- Register login logout for renter :

```php
4 references | 0 implementations
class RenterController extends Controller
{
    1 reference | 0 overrides
    public function store(createRenterRequest $request){

        $data = $request->validated();

        if(!empty($data['photo'])){
            $ext = $data['photo']->getClientOriginalExtension();
            $imageName = time().'_'.rand().'.'.$ext;
            $photoName =  $data['photo']->move(public_path().'/RenterPhoto', $imageName);
            $newName = explode('\\',$photoName);
            $data['photo'] = end($newName);
          }else{  unset($data['photo']);  }

          $renter = Renter::create($data);
          $renter->notify(new EmailVerificationNotification());

          return response()->json(["message"=> "Registration successful","data"=> new RenterResource($renter)]);

    }
```

```php
class RenterController extends Controller

    1 reference | 0 overrides
    public function login(Request $request){
        $renter =Renter::where("username" , $request->input("username"))->first();
        if(!$renter){
            return response()->json(["message"=> "user not found"] , 401);
        }
        if(!Hash::check($request->input("password") , $renter->password)){
            return response()->json(["message" => "wrong password"] , 401);
        }
       $token = $renter->createToken("Maskank");
        return response()->json(["token" => $token->plainTextToken ,
        "message" => "renter successfully logged in" , "data"=> new RenterResource($renter)] , 200);
    }
    1 reference | 0 overrides
    public function logout(Request $request){
        try{
            $request->user()->currentAccessToken()->delete();
            return response()->json(["message" => "renter successfully logged out"] , 200);
        }
        catch (ValidationException $e) {
            return response()->json(["error" => $e->errors()], 422);
        }

    }
```

Backend (PHP Laravel)

3-Verification email :

```php
ss EmailVerificationController extends Controller
3 references
private $otp;
0 references | 0 overrides
public function __construct(){
    $this->otp = new Otp;
}
1 reference | 0 overrides
public function  verifyEmailRenter(EmailVerificationRequest $request){
    $otp2 = $this->otp->validate($request->email,$request->otp);
    if(!$otp2->status){
        return response()->json(['error'=>$otp2],401);
    }

    $renter = Renter::where('email',$request->email)->first();
    $renter->update(['email_verified_at'=>now()]);
    $success['success'] = true;
    $success['data']    = new RenterResource($renter);
    $success['message'] = 'you verification Email';
    return response()->json([$success],200);
}
1 reference | 0 overrides
public function  verifyEmailOwner(EmailVerificationRequest $request){

    $otp2 = $this->otp->validate($request->email,$request->otp);


    if(!$otp2->status){
        return response()->json(['error'=>$otp2],401);
    }

    $owner = Owner::where('email',$request->email)->first();
    $owner->update(['email_verified_at'=>now()]);
    $success['success'] = true;
    $success['data']    = new OwnerResource($owner);
    $success['message'] = 'you verification Email';
    return response()->json([$success],200);
}
```

Backend (PHP Laravel)

```php
public function store(CreatePostRequest $request)
{
    $validatedData = $request->validated();

    $images = $request->file('images');
    $imageName='';
    foreach($images as $img){
        $ext = $img->getClientOriginalExtension();
        $newName = time().rand().'.'.$ext;
       $fullImage =  $img->move(public_path().'/imagesPosts', $newName);
        $newName = explode('\\',$fullImage);

        $imageName =$imageName.end($newName).',';
        }

        $validatedData['images']=$imageName;

    try {
        $post = Post::create($validatedData);

        return response()->json([
            'message' => 'Post created successfully',
            'item' =>new PostResource($post),
        ], 201);
    } catch (QueryException $exception) {
        logger()->error('Failed to create item: ' . $exception->getMessage());
        return response()->json(['message' => 'Failed to create item. Error: ' . $exception->getMessage()], 500);
    }
}
1 reference | 0 overrides
public function updateStatus(Request $request, $post_id)
{
    $post = Post::findOrFail($post_id);
    $post->status = 1;
    $post->save();
    return response()->json(['message' => 'Post status updated successfully', 'post' => new PostResource($post)]);
}
```

4-Add post && Approved :

Backend (PHP Laravel)

5-Reset password :

```php
0 references | 0 overrides
public function __construct(){
    $this->otp = new Otp;
}
1 reference | 0 overrides
public function passwordResetOwner(ResetPasswordRequest $request){
    $otp2 = $this->otp->validate($request->email,$request->otp);

    if(! $otp2->status){
        return response()->json(['error'=> $otp2 ],401);
    }

    $user = Owner::where('email',$request->email)->first();
    $user->update(['password'=>Hash::make($request->password)]);
    $user->tokens()->delete();

    $success['success']=true;
    $success['message']='you Reset password';
    return response()->json([$success],200);
}
1 reference | 0 overrides
public function passwordResetRenter(ResetPasswordRequest $request){
    $otp2 = $this->otp->validate($request->email,$request->otp);

    if(! $otp2->status){
        return response()->json(['error'=> $otp2 ],401);
    }

    $user = Renter::where('email',$request->email)->first();
    $user->update(['password'=>Hash::make($request->password)]);
    $user->tokens()->delete();

    $success['success']=true;
    $success['message']='you Reset password';
    return response()->json([$success],200);
```

Backend (PHP Laravel)

```php
class UpdatePostController extends Controller
{
    1 reference | 0 overrides
    public function update(UpdatePostRequest $request ,$post_id){
        try {
            $post = Post::findOrFail($post_id);
            $data = $request->validated();
            if ($request->hasFile('images')) {
                $images = $request->file('images');
                $imageName='';
                foreach($images as $img){
                $ext = $img->getClientOriginalExtension();
                $newName = time().rand().'.'.$ext;
                $fullImage =  $img->move(public_path().'/imagesPosts', $newName);
                $newName = explode('\\',$fullImage);
                $imageName =$imageName.end($newName).',';
                    }
                if (!empty($imageName)) {
                    $imagesPost = explode(',', $post->images);

                    foreach ($imagesPost as $image) {
                        if (!empty($image)) {
                        $imagePath = public_path('imagesPosts/' . $image);
                            if (file_exists($imagePath)) {
                            unlink($imagePath);
                                }
                        }
                    }
                    $data['images'] = $imageName;
                }
            }
            $post->update($data);
            return response()->json(['message' => 'Post updated successfully', 'post' => new PostResource($post)], 200);
        } catch (Exception $e) {
            return response()->json(['message' => 'Error: ' . $e->getMessage()], 500);
        }
```

6-Update post :

3-Machine Learning
(python)

Machine Learning (python)

1- Label Encoding :is a technique for converting categorical data into numerical data that machine learning algorithms can understand

```
types_of_apartments = ['Apartment', 'Duplex', 'Studio', 'Room', 'Penthouse']

label_encoder = LabelEncoder()
df['House_Type'] = label_encoder.fit_transform(df['House_Type'])

def encode_floor(floor):
    if floor == 'Ground':
        floor = 0
    if floor == '10+':
        floor = 11
    if floor == 'Highest':
        floor = 12
    return int(floor)

df['Floor'] = df['Floor'].apply(encode_floor)

# for i, apartment_type in enumerate(types_of_apartments):
#     print(apartment_type, '-->', i)

df['Furnished'] = label_encoder.fit_transform(df['Furnished'])

df['Region'] = label_encoder.fit_transform(df['Region'])

df['City'] = label_encoder.fit_transform(df['City'])

df_filtered = df[df['For_rent'] != 'No']

df_filtered.head()
```

2 - Dropped the houses isn't for rent features.

```
df.drop(df[df['For_rent'] == 'No'].index, inplace=True)
```

Machine Learning (python)

3-Extracted a new feature from the product name using Python functions Label Encoding

```
df['no_of_rooms'] = df['Bedrooms'] + df['Bathrooms']
df.head()
```

	House_Type	Size	Bedrooms	Bathrooms	Floor	Furnished	For_rent	Region	City	Price	no_of_rooms
0	0	200	3	3	1	0	Yes	55	1	12000	6
1	0	130	3	2	2	1	Yes	55	1	8500	5
2	0	130	2	1	3	0	Yes	55	1	5000	3
3	0	200	3	1	0	0	Yes	54	1	8500	4
4	0	160	3	2	0	0	Yes	55	1	7000	5

4- Splitting data into train and test data:

```
X = df.drop(['Price', 'For_rent'], axis=1)
y = df['Price']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Machine Learning (python)

5-Choosing the best model:

```
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression()
```

```
y_pred = model.predict(X_test)
```

```
# Evaluate the model
rmse = mean_squared_error(y_test, y_pred, squared=False)

# Calculate R-squared (R2) score
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2) Score: {r2}")
```

```
Root Mean Squared Error (RMSE): 15617.660354775175
R-squared (R2) Score: 0.12736498525918372
```

```
# Initialize XGBoost Regressor
xgb_regressor = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)

# Train the model
xgb_regressor.fit(X_train, y_train)

# Make predictions
y_pred = xgb_regressor.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)

print(f"XGBoost Regression - Root Mean Squared Error (RMSE): {rmse}")
print(f"XGBoost Regression - R-squared (R2) Score: {r2}")
```

```
XGBoost Regression - Root Mean Squared Error (RMSE): 13999.144862634157
XGBoost Regression - R-squared (R2) Score: 0.29886168126816626
```

```
# Initialize Gradient Boosting Regressor
gb_regressor = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)

# Train the model
gb_regressor.fit(X_train, y_train)

# Make predictions
y_pred = gb_regressor.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2) Score: {r2}")
```

```
Root Mean Squared Error (RMSE): 15012.013796250443
R-squared (R2) Score: 0.1937335335269348
```

```
# Initialize Random Forest Regressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
rf_regressor.fit(X_train, y_train)

# Make predictions
y_pred = rf_regressor.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)

print(f"Random Forest Regression - Root Mean Squared Error (RMSE): {rmse}")
print(f"Random Forest Regression - R-squared (R2) Score: {r2}")
```

```
Random Forest Regression - Root Mean Squared Error (RMSE): 14926.798082532536
Random Forest Regression - R-squared (R2) Score: 0.20286109853563805
```

Machine Learning (python)

6- Create API using Flask

```
        h_type, h_fur, h_region, h_city = preproces(h_type, h_fur, h_region, h_city)
        output = np.array([h_type, h_size, h_bedrooms, h_bathrooms, h_floor, h_fur, h_region, h_city, num_rooms])
        price = model.predict(output.reshape(1, -1))
        return jsonify(price[0])

    except Exception as e:
        # Handle exceptions and return an error message in JSON format
        return jsonify({'error': str(e)})


if __name__ == '__main__':
    # Run the Flask application in debug mode
    app.run(debug=True)
```

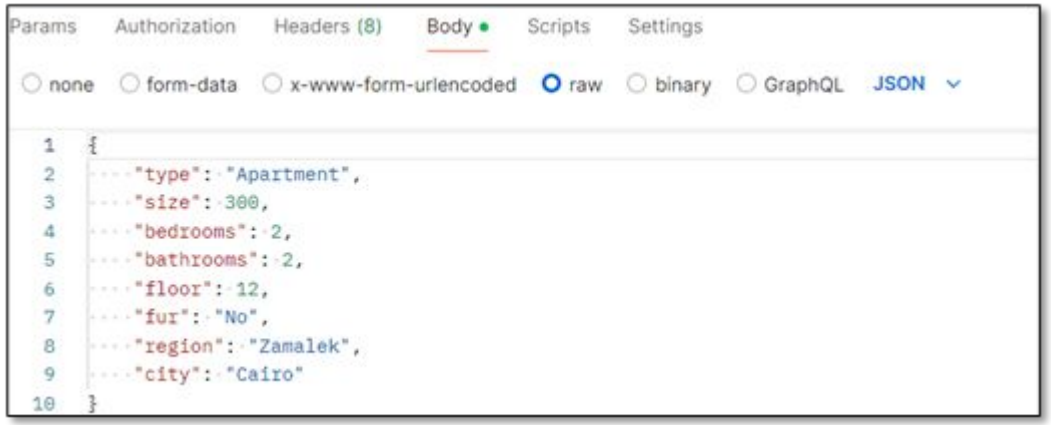
```
from flask import Flask, request, jsonify
import joblib
import numpy as np
from preproces import preproces

# load the model from disk
model = joblib.load('final_model.pkl')

app = Flask(__name__)

@app.route('/house_pred', methods=['POST'])
def prediction():
    try:
        house_details = request.get_json()
        h_type = house_details['type']
        h_size = house_details['size']
        h_bedrooms = house_details['bedrooms']
        h_bathrooms = house_details['bathrooms']
        h_floor = house_details['floor']
        h_fur = house_details['fur']
        h_region = house_details['region']
        h_city = house_details['city']

        num_rooms = h_bathrooms+h_bedrooms
```

Machine Learning (python)

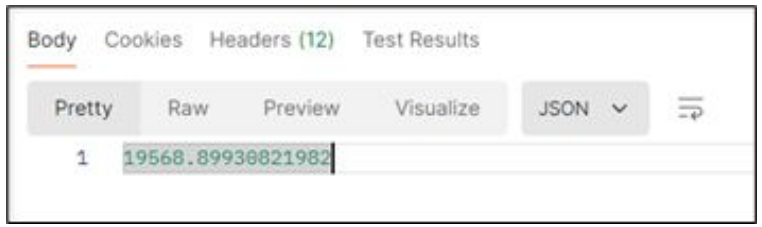
7- The accuracy :

```
Root Mean Squared Error (RMSE): 15617.660354775175
R-squared (R2) Score: 0.12736498525918372
```

8- Test the API Using Postman:

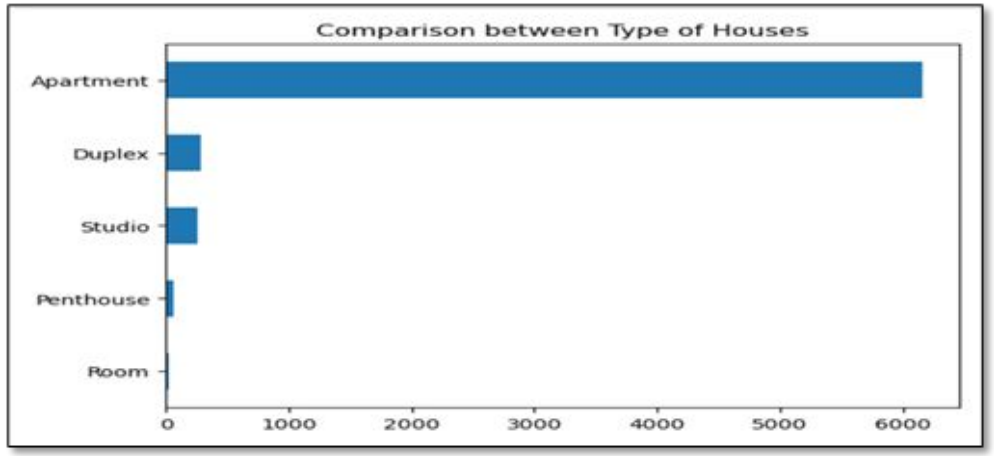


9 -The Result :

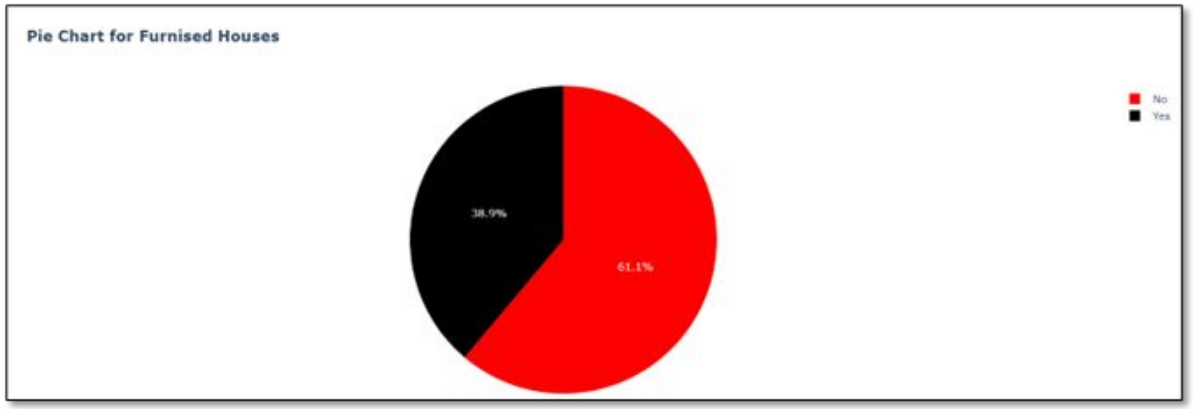


Results and discussion

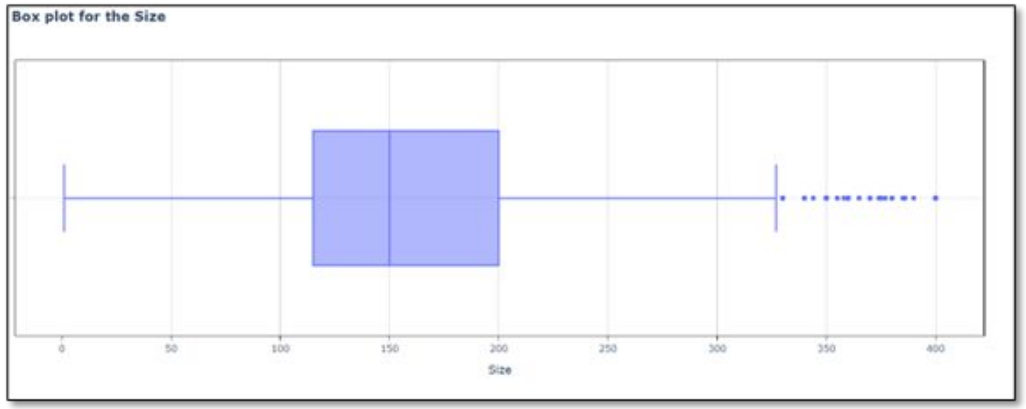
Results and discussion

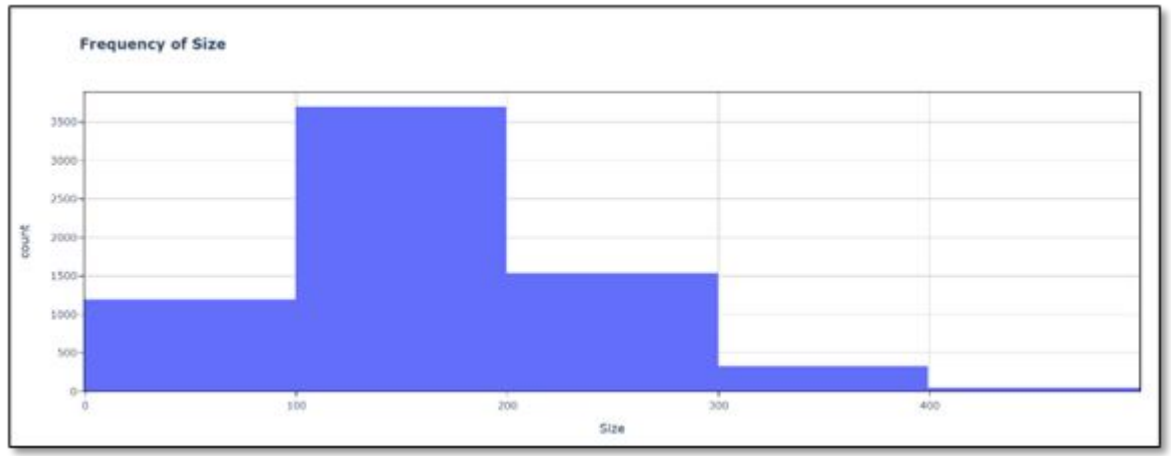


Clearly most common type of houses is Apartment



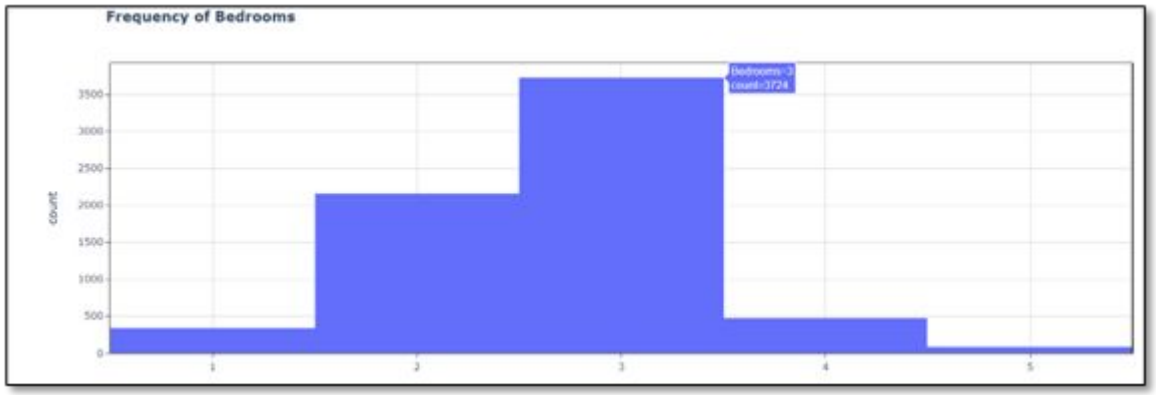
About 61% of the dataset houses are not furnished

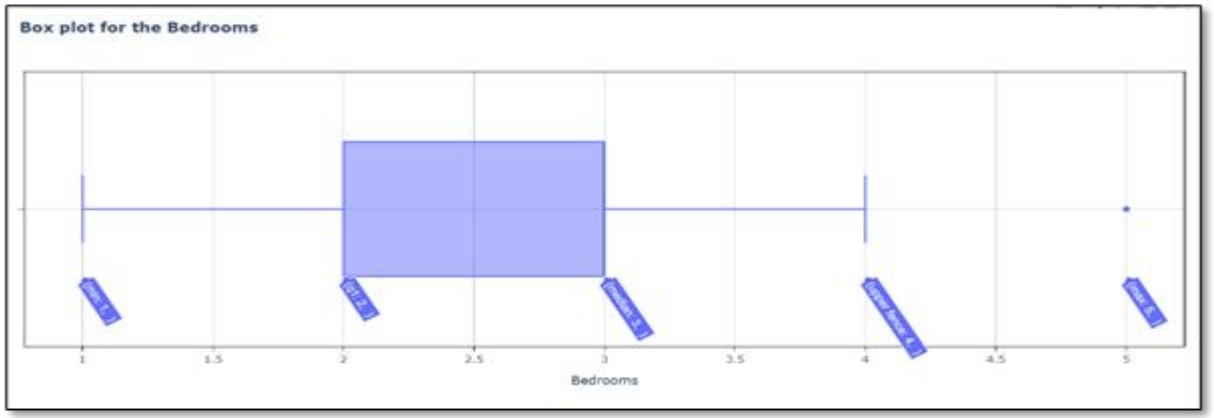




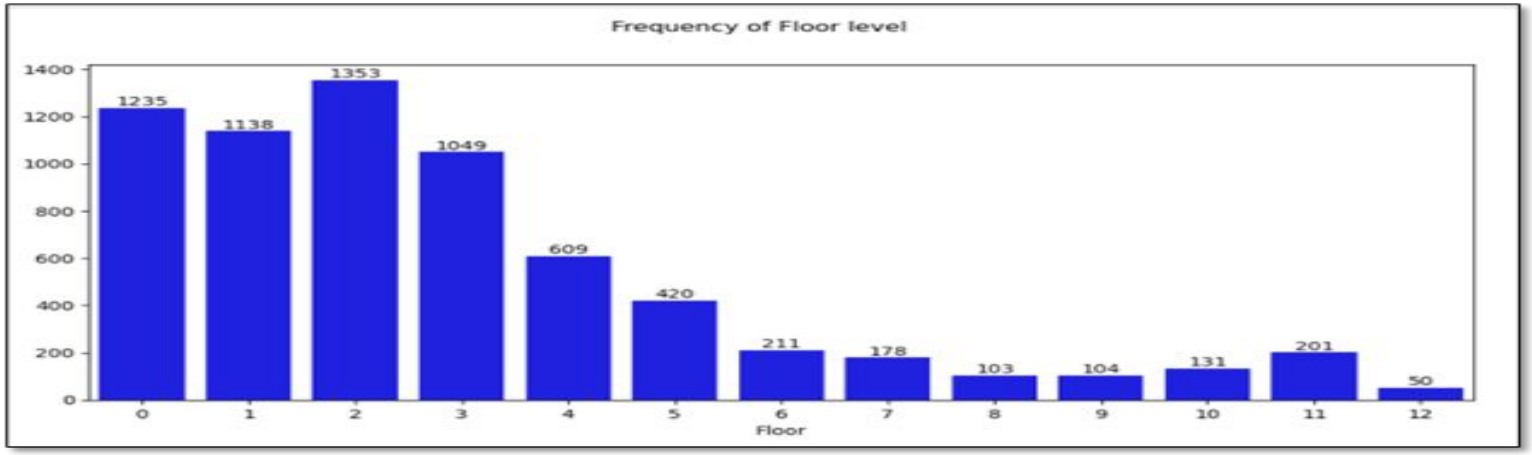
The most common house size in range 100- 200

Results and discussion



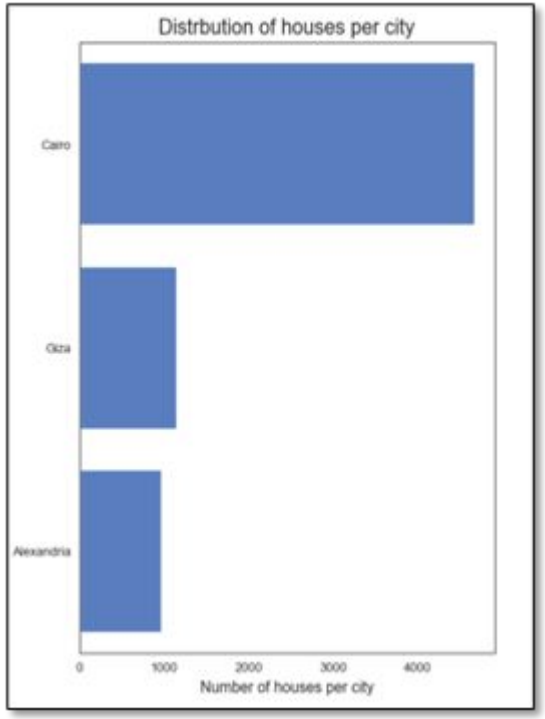


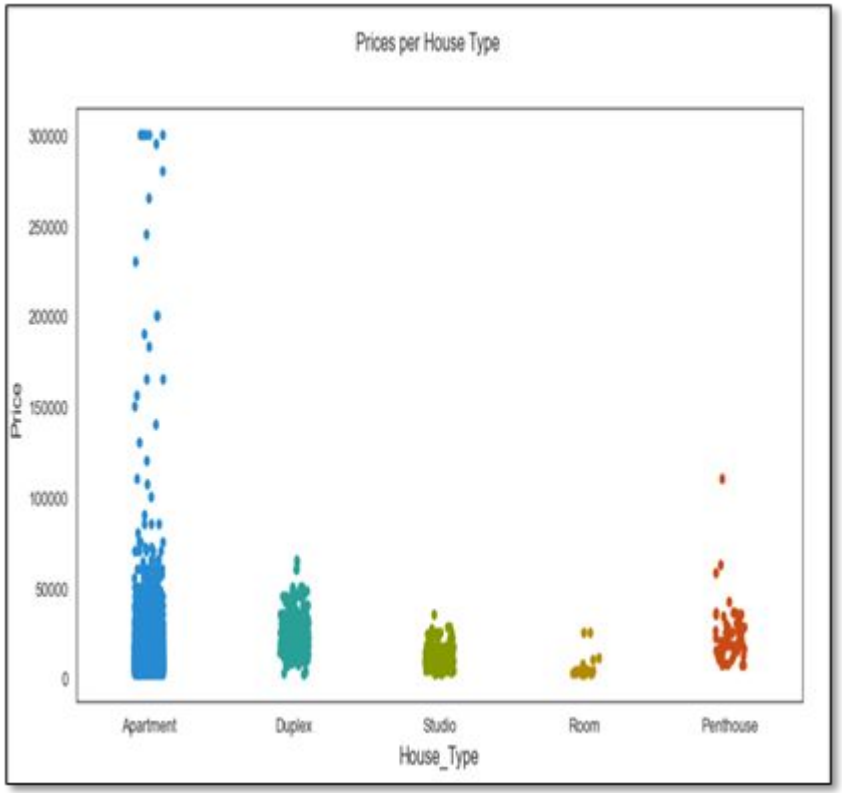
The most common house size in range 100- 200



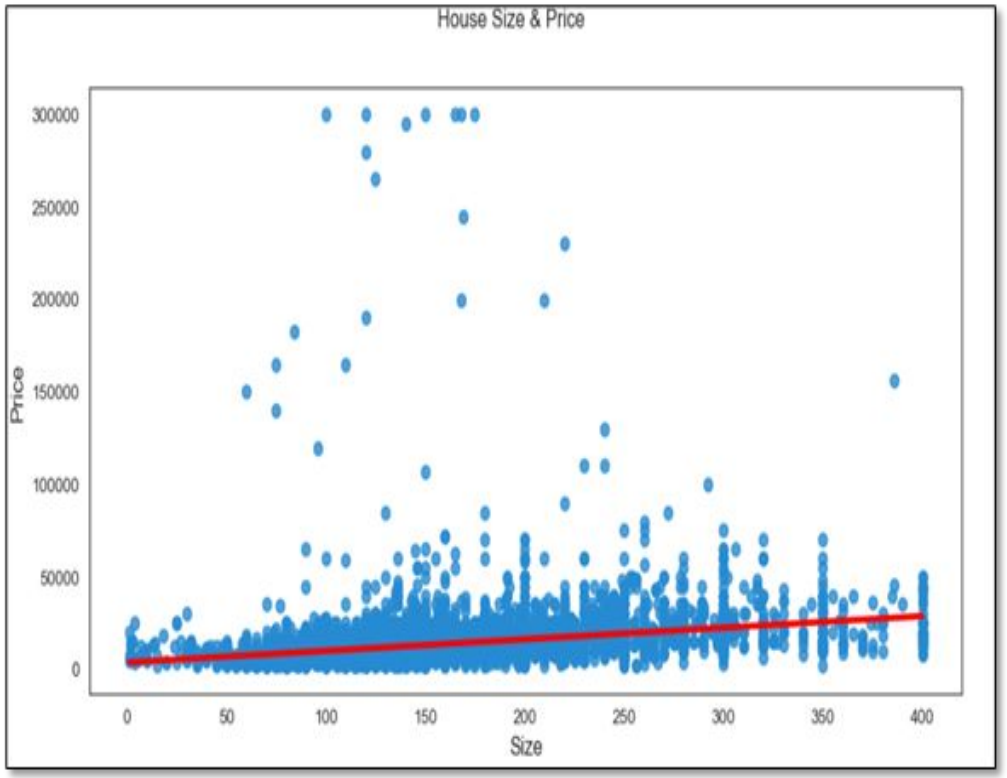
Most common # of bedrooms 3 and 2

Results and discussion



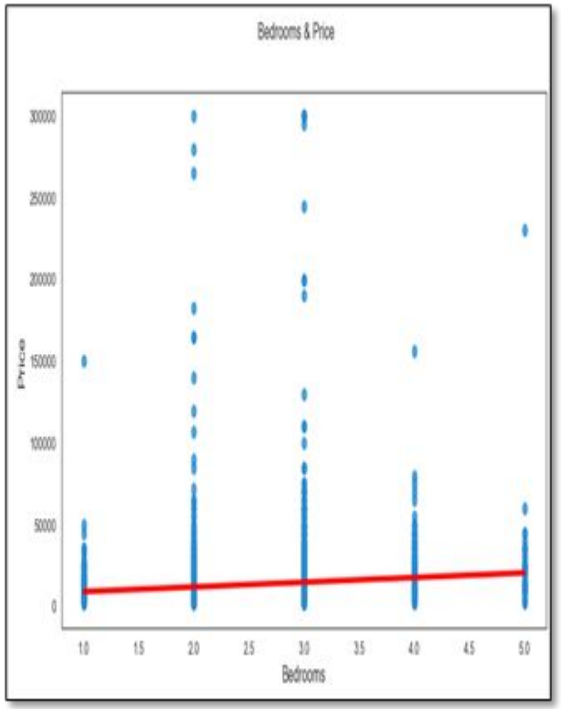


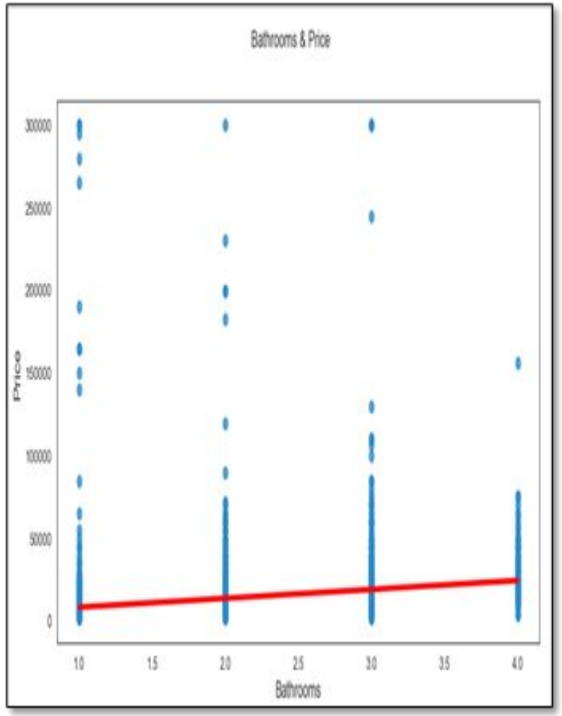
The highest price is the Apartment

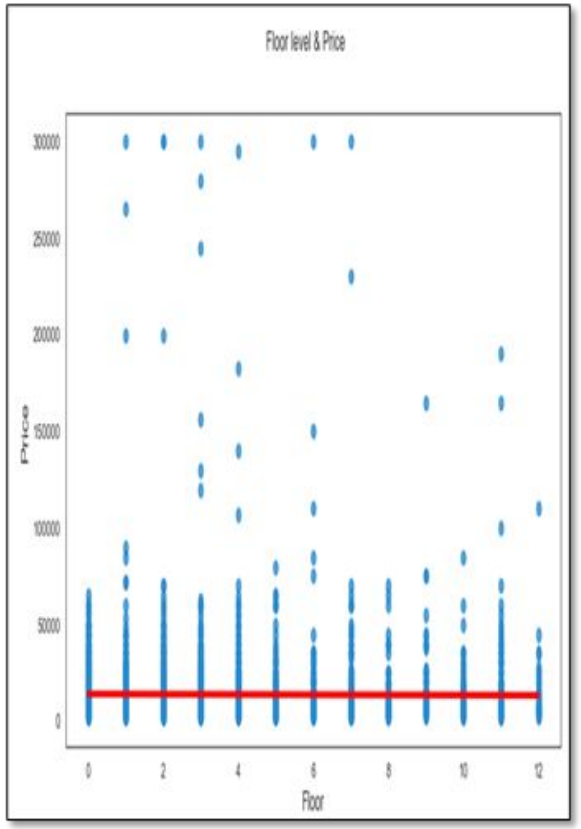


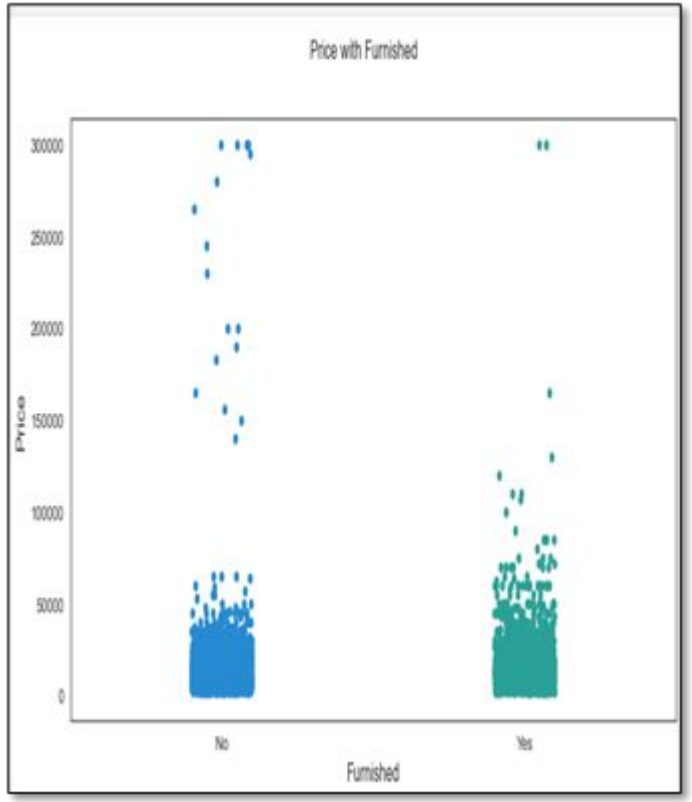
The higher the size, the higher the price (Positive relationship)

Results and discussion

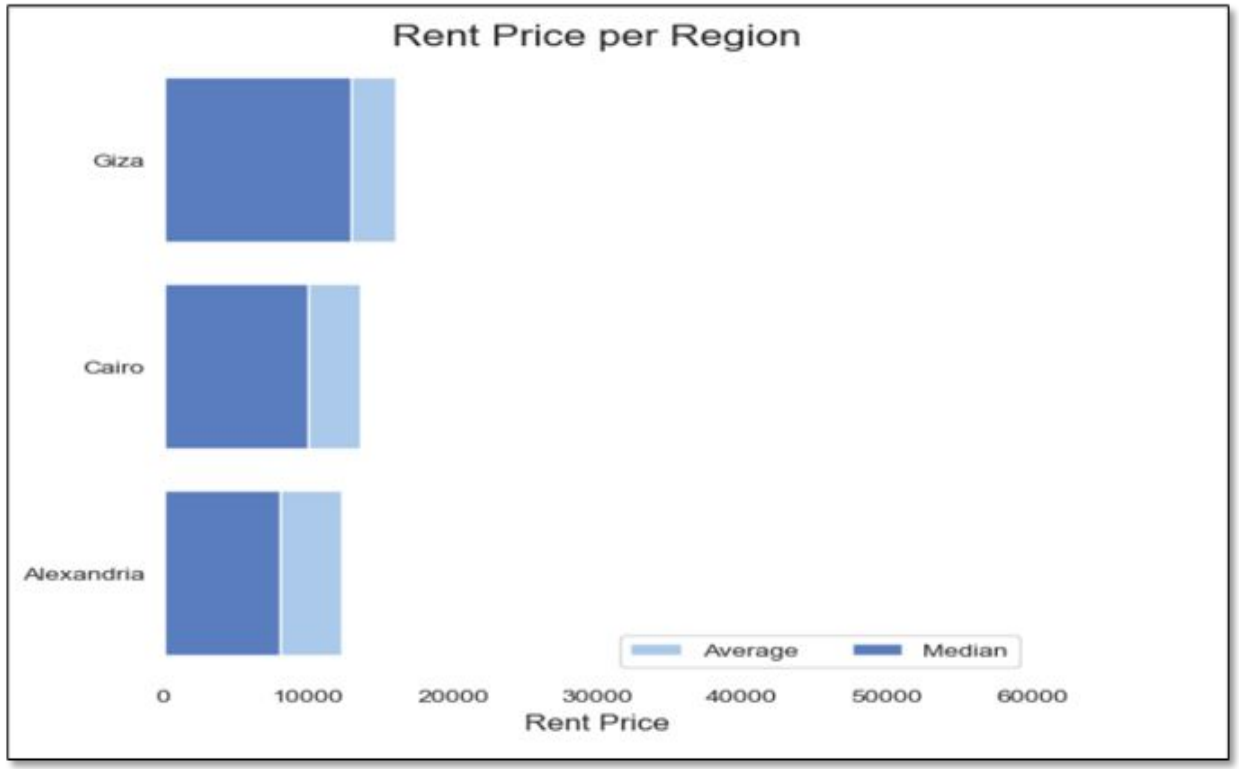


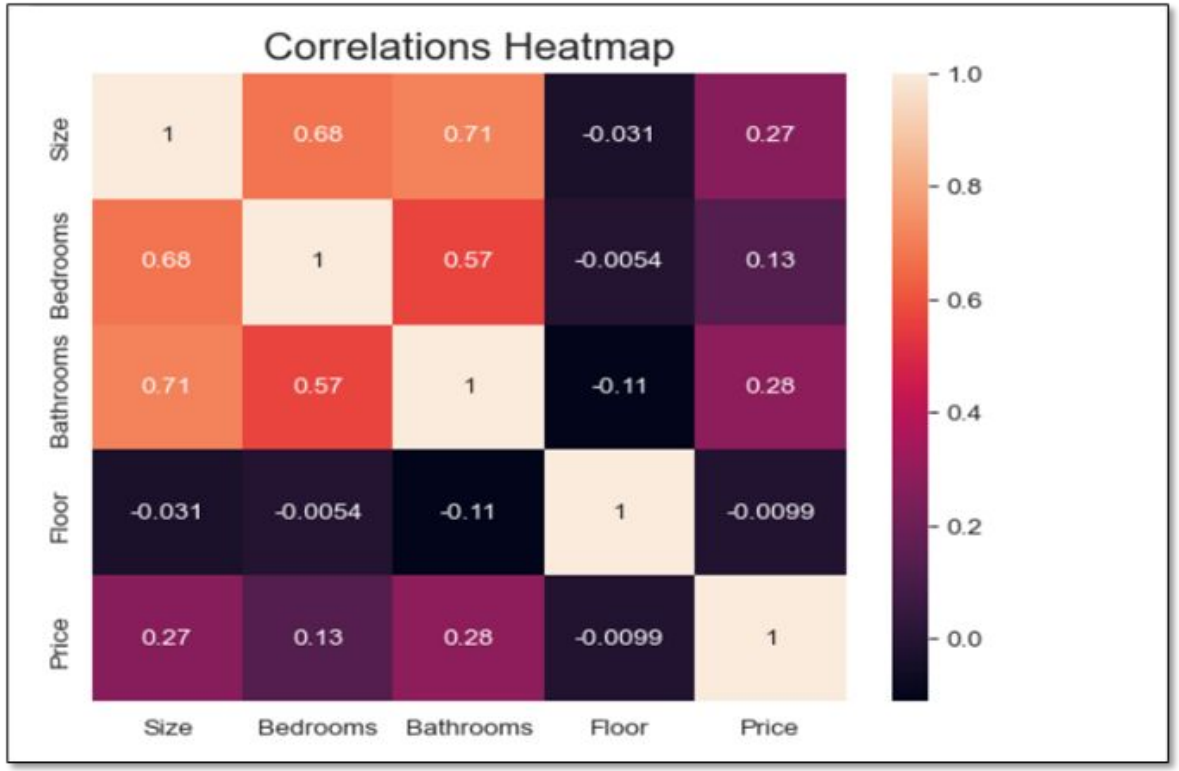




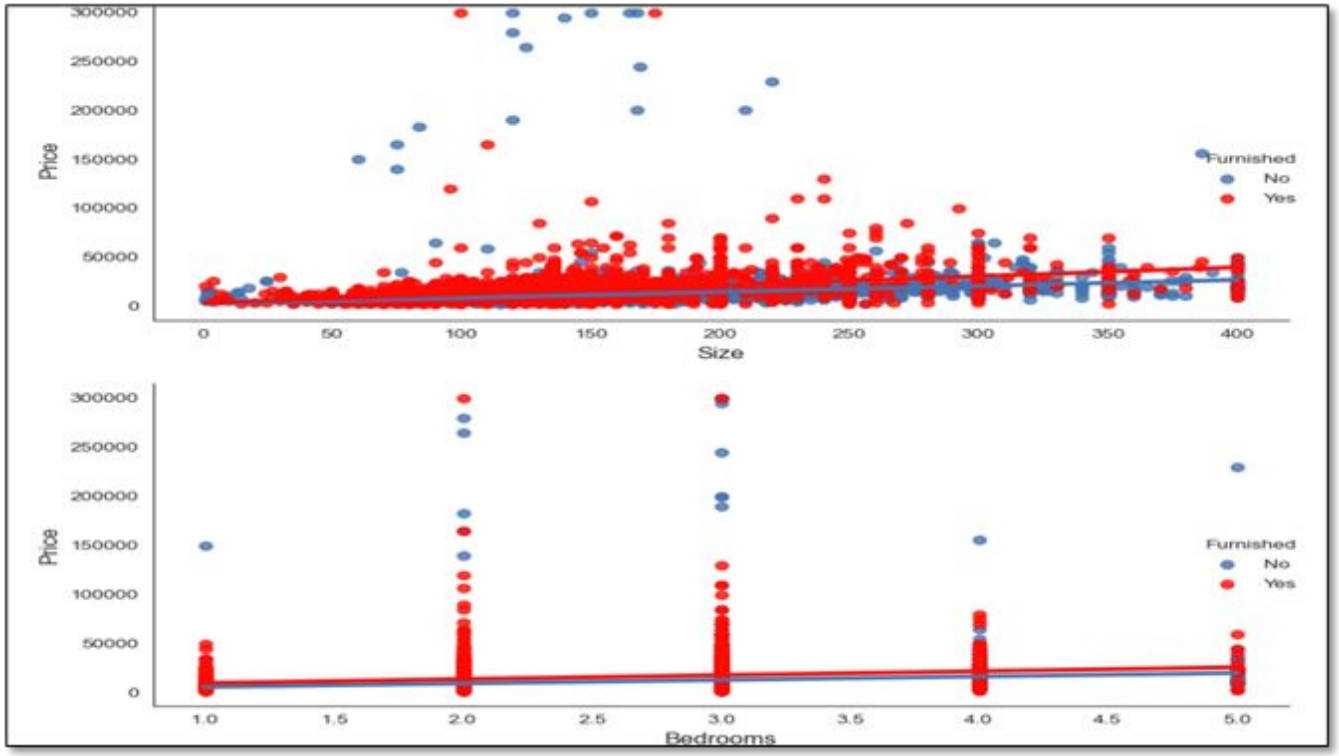


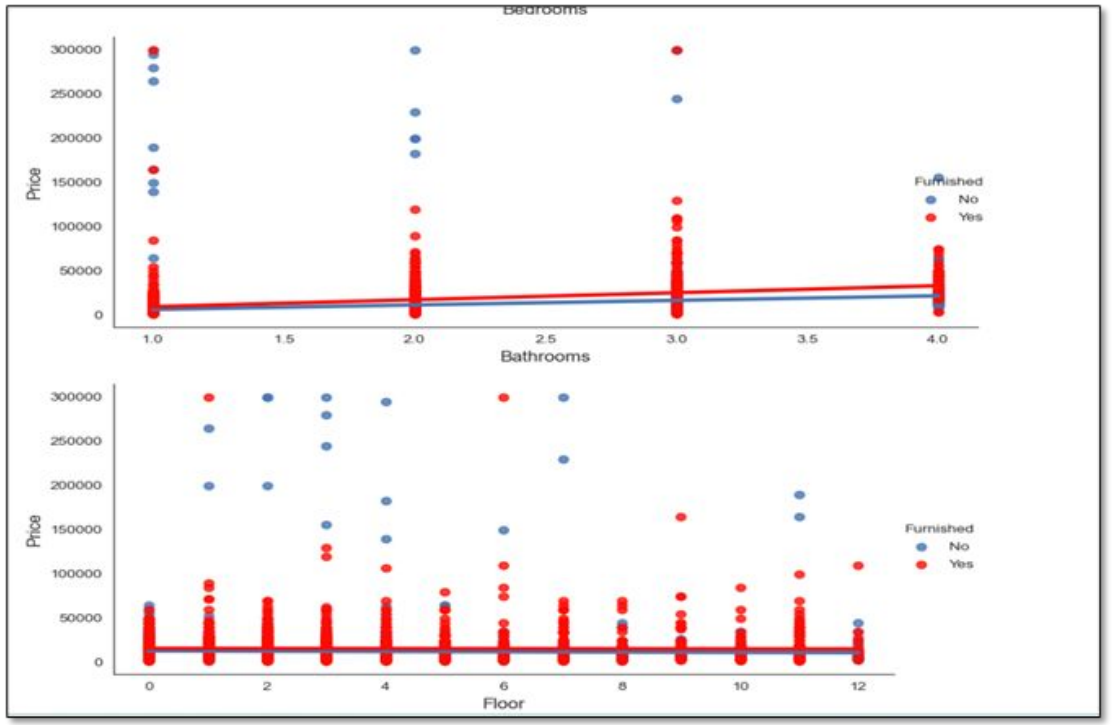
Results and discussion



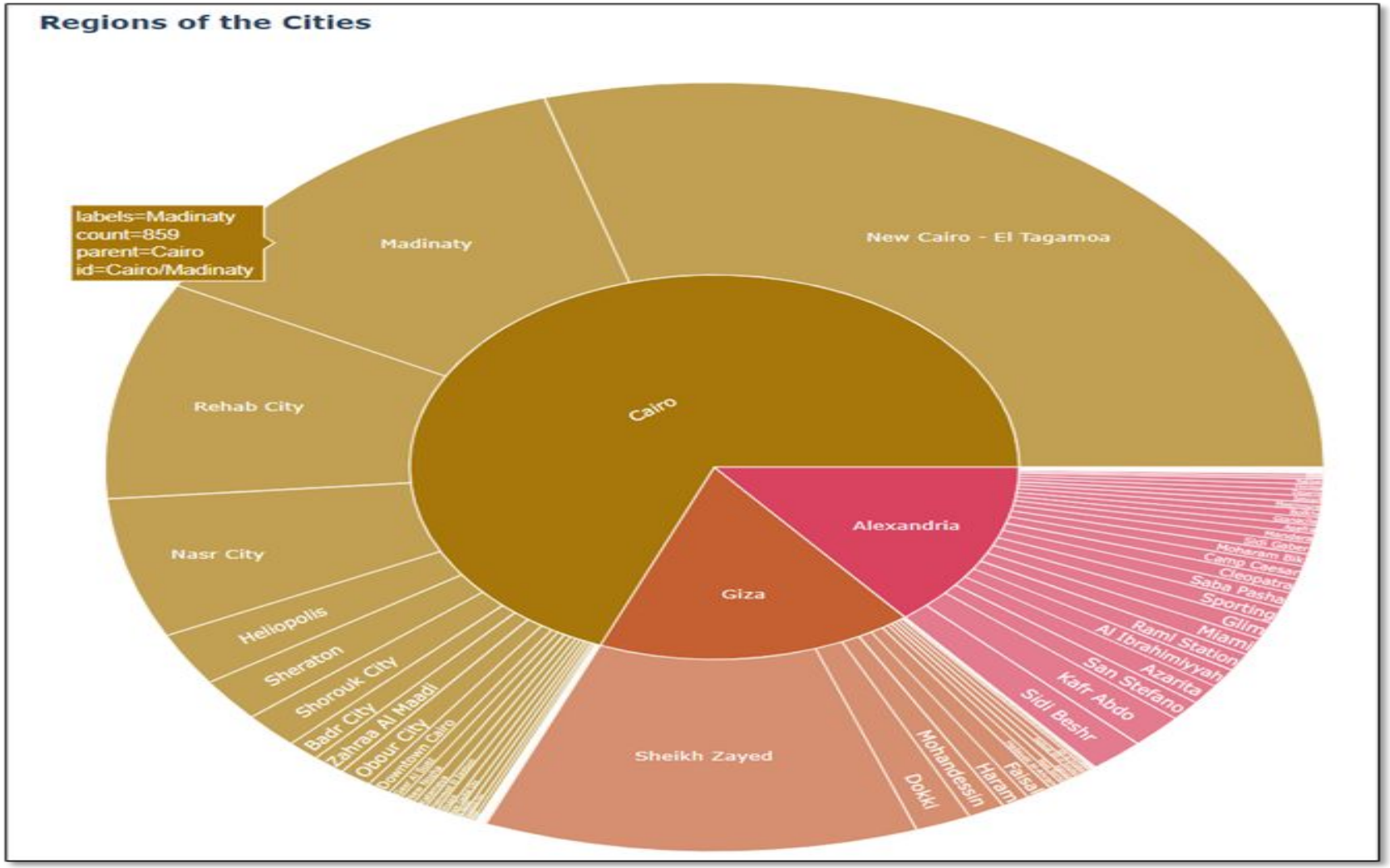


Results and discussion





Results and discussion



Testing

FUNCTIONAL REQUIREMENTS TESTING:

Functional Requirement Number	Test Case No.	Functional Requirement Description
R 1	T 1	Login to the application
R 2	T 2	Users can view all properties. The owner can upload their apartment.
R 3	T 3	Users can buy apartments
R 4	T 4	Users can search for a property

Test Case: 1

Title	•Log in system
Description	•Login to the application
Precondition	•The user and owner should register
Assumption	•The home page will open
Test Steps	•Enter the email address and password. •Click on the button “Login”
Expected Result	•The home page will open
Actual Result	•The home page opens

Test Case: 2

Title	•View all properties
Description	•Users can view all properties. •The owner can upload their apartment.
Precondition	•After the user logs in, he can view all properties on the home page. •After the owner logs in, he can upload his apartments to the home page.
Assumption	•The home page will show all properties.
Test Steps	1.logged-in 2.open home page
Expected Result	•The home page will show all properties.
Actual Result	•The home page will show all properties.

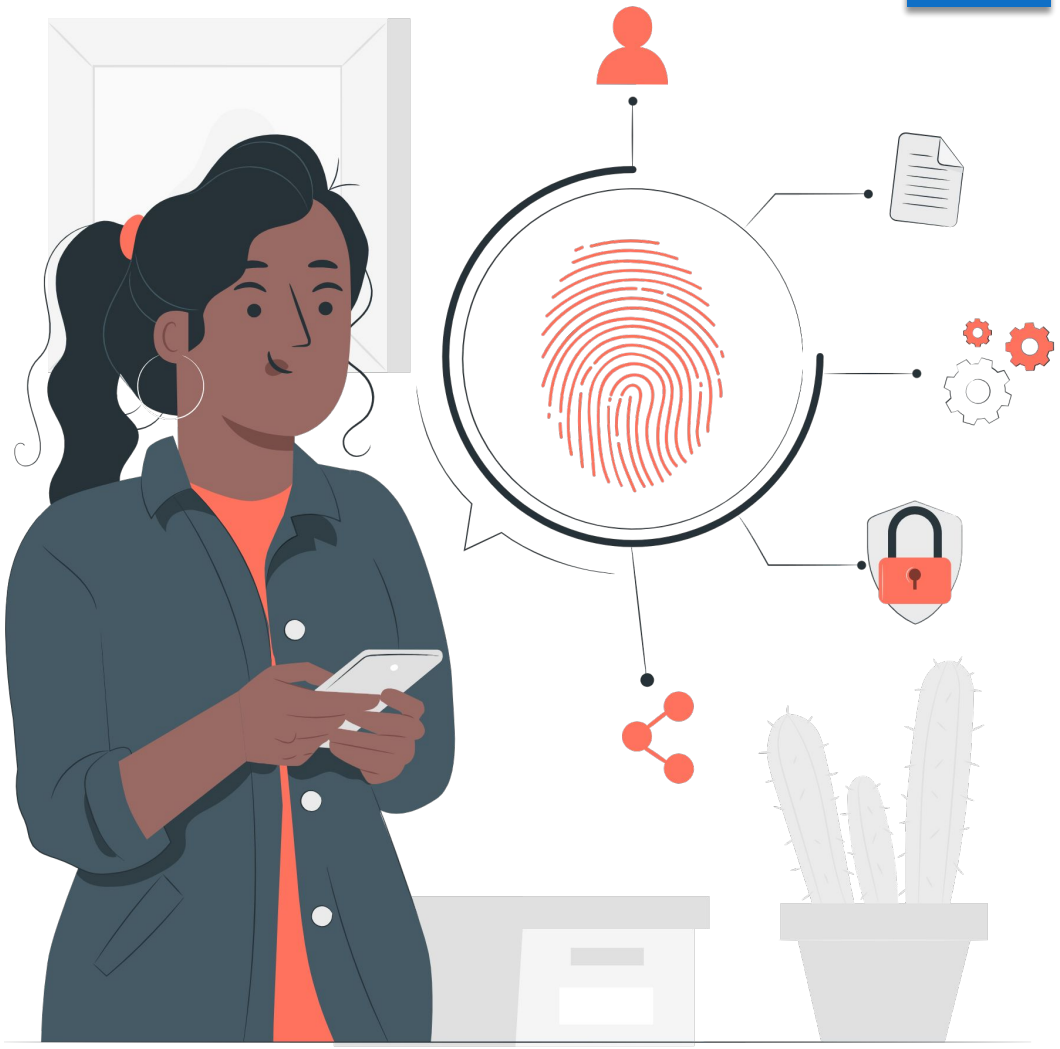
Test Case: 3

Title	•Buy apartments
Description	•Users can buy any property.
Precondition	•After the user clicks on the button “Book” he will redirected to a booked page.
Assumption	•Users can buy any property.
Test Steps	1.open the property page. 2.click on the button “Book”
Expected Result	•Users can buy any property.
Actual Result	•Users can buy any property successfully.

Test Case: 4

Title	•Search for a apartments
Description	•Users can search any property.
Precondition	•After a user clicks on the search icon on the navigation bar, they can type the property they want or the filter feature can facilitate it.
Assumption	•Users can search any property.
Test Steps	1.open the home page. 2.click on the search icon. 3.the user writes of the property. 4.click on the result. 5.open the property page
Expected Result	•Users can search any property.
Actual Result	•Users find the property successfully.

Security



Security

Overview of Application security:

- introduction to application security
- Common threats and vulnerabilities
- protection and testing

Security

Best practices for application security:

- App prioritizes data security
- Secure coding practices
- Authentication and authorization
- Encryption and decryption
- Secure transactions

Security

CIA triad:

- Confidentiality
- Integrity
- Availability

Future work

Future work

This is not the end of our road, we will keep working on our application to release it to the public and increase the quality of the features, it still needs more research and development, and this is what we will continue to do to reach the best possible efficiency.

Future work

Enhance the accuracy of models.

find more datasets for all governorates.

Conclusion

Conclusion

Summarizes our work at every stage of the project. We started with many ideas and features and needed a clear implementation plan, but we kept searching and working to make them suitable for students. With machine learning and continuous learning knowledge, we worked diligently for over six months. Research was crucial, and we explored various modeling approaches. We consistently applied new techniques to overcome challenges.

References

Flutter

https://pub.dev/

https://fluttergems.dev/

Back-End

https://laravel.com

https://www.php.net/manual/en/langref.php

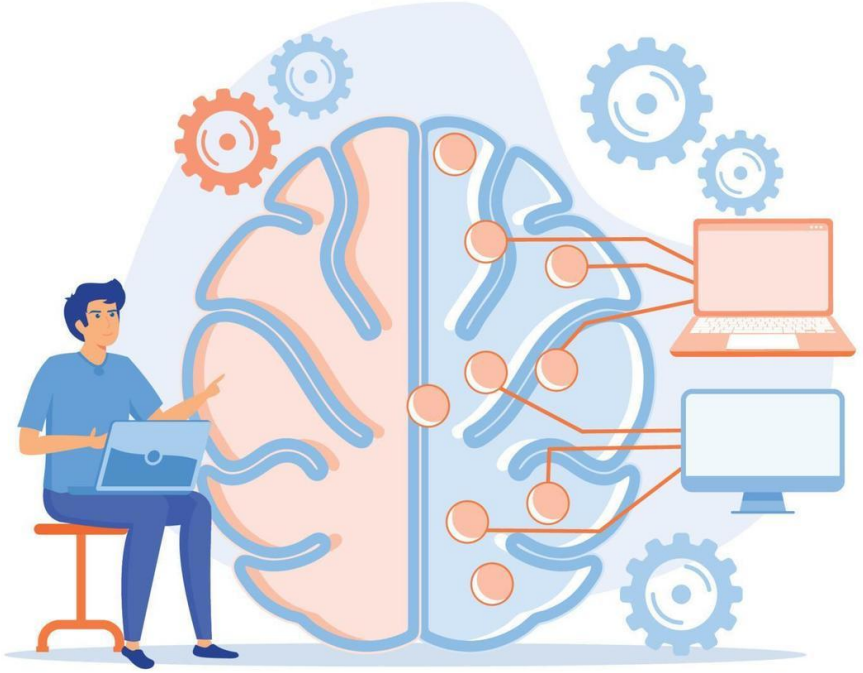
Machine Learning

http://scikit-learn.org

http://kaggle.com

https://docs.python.org/3/library/index.html

https://gemini.google.com/app



Thank you