Project A

**Diploma of IT**

**Programming 2**



Weight: 20% DUE : Week 10



# TASK DESCRIPTION

This project requires students to create an object oriented design and coded solution for a chosen application, using all the concepts taught in the subject up to and including week 9.

You may choose to complete this assessment with at most one (1) other person. To work as a pair; you and your partner MUST be in the same tutorial class. You will need to spend approx. 10 hours together working on the assessment (outside of tutorial hours). If you do not intend to spend this time working together – choose to complete the assessment on your own. Both partners must put in an equal amount of work and effort, or the group will be dissolved by the tutor.

# OBJECTIVES

1. Evaluate if a software solution is well-designed and fit for purpose.
2. Design a well-constructed object oriented solution from a specification.
3. Demonstrate a working knowledge of list and map data structures.
4. Use inheritance to improve the system design.
5. Construct a GUI interface



This is a take home assessment to evaluate your understanding of Java programming concepts taught so far in IPRO002. The assessment will be composed of three (3) parts: the scenario, the code and explanation (including proof of testing).

# THE SCENARIO

### Welcome Page:

* **Sign Up:**
  + Users can create a new account by providing their username, password, and email address. The system validates the input and displays any errors that need correction before successful account creation.
* **Log In:**
  + Existing users can log in by entering their credentials. If the credentials are correct, they are directed to the Home Page. If not, a notification prompts them to try again.

### Home Page:

The Home Page serves as the main dashboard where users can access various features such as managing their account, handling emails, browsing the market, and utilizing management tools.

#### 1. Account: Edit Profile

* **Edit Profile:**
  + Users can update their personal information by clicking on the "Edit Profile" button. A form appears where they can modify details like their username or email. The system validates the changes and notifies the user of any errors.

#### 2. Mailbox: View Mail, Delete Mail, Send Mail

* **View Mail:**
  + The mailbox displays a list of received emails with their title and sender. Users can select an email and click "View Details" to read the full content in a new window.
* **Delete Mail:**
  + Users can delete an email by selecting it from the list and clicking the "Delete Email" button. The system removes the email from the user's mailbox.
* **Send Mail:**
  + Users can compose and send a new email by clicking "Send Mail." They enter the recipient’s address, subject, and message in a form. The system validates and sends the email, notifying the user of any errors.

#### 3. Market: View Property, Sort Property

* **View Property:**
  + The market page displays properties with brief details such as type, rent price, and size. Users can select a property and click "View Details" to see more comprehensive information in a new window.
* **Sort Property:**
  + Users can sort properties based on criteria like size or rent price. A sorting window allows them to select sorting options (ascending or descending), and the system updates the property list accordingly.

#### 4. Management:

Management features differ for agents and clients, each with specific tools for handling properties and contracts.

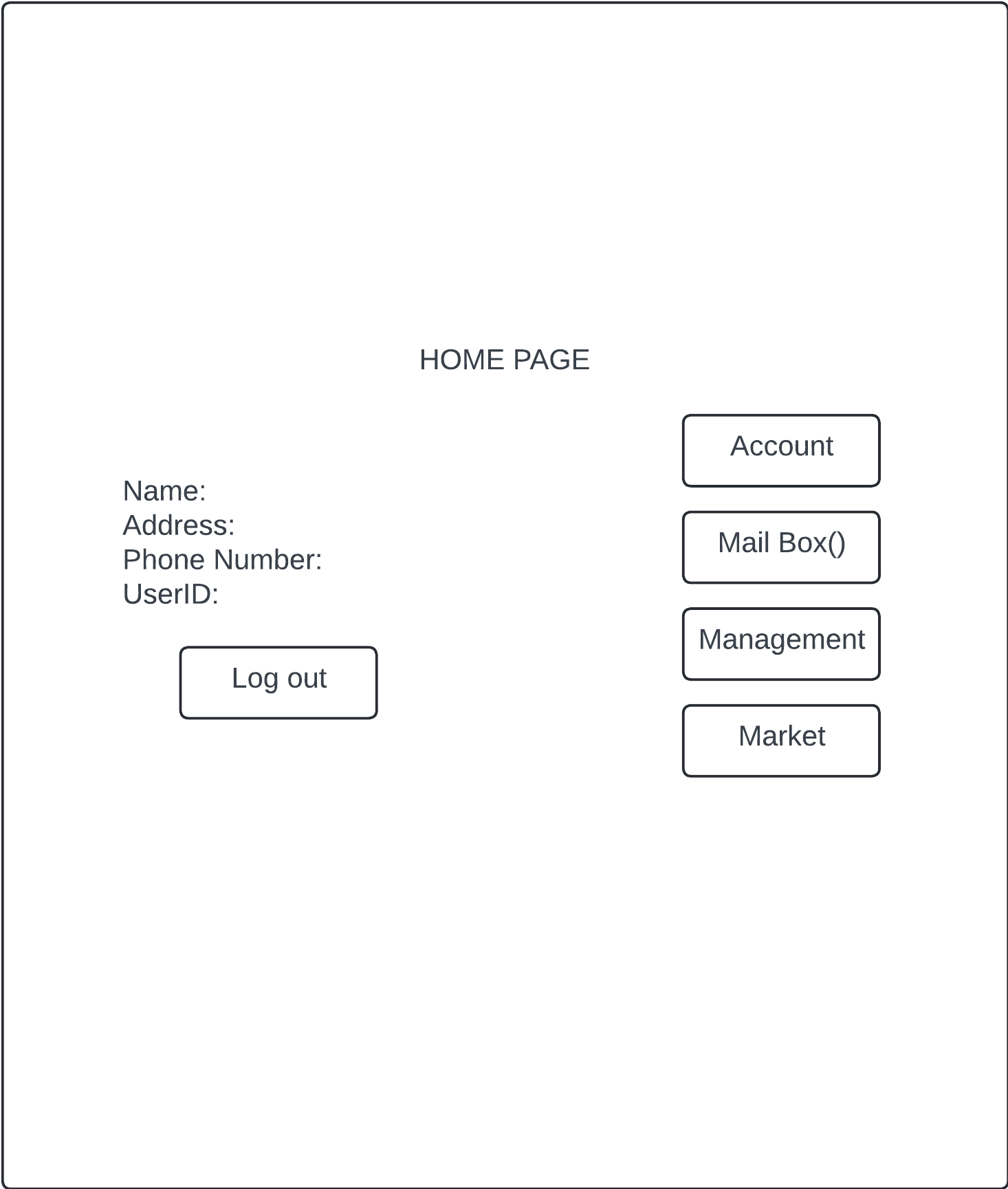
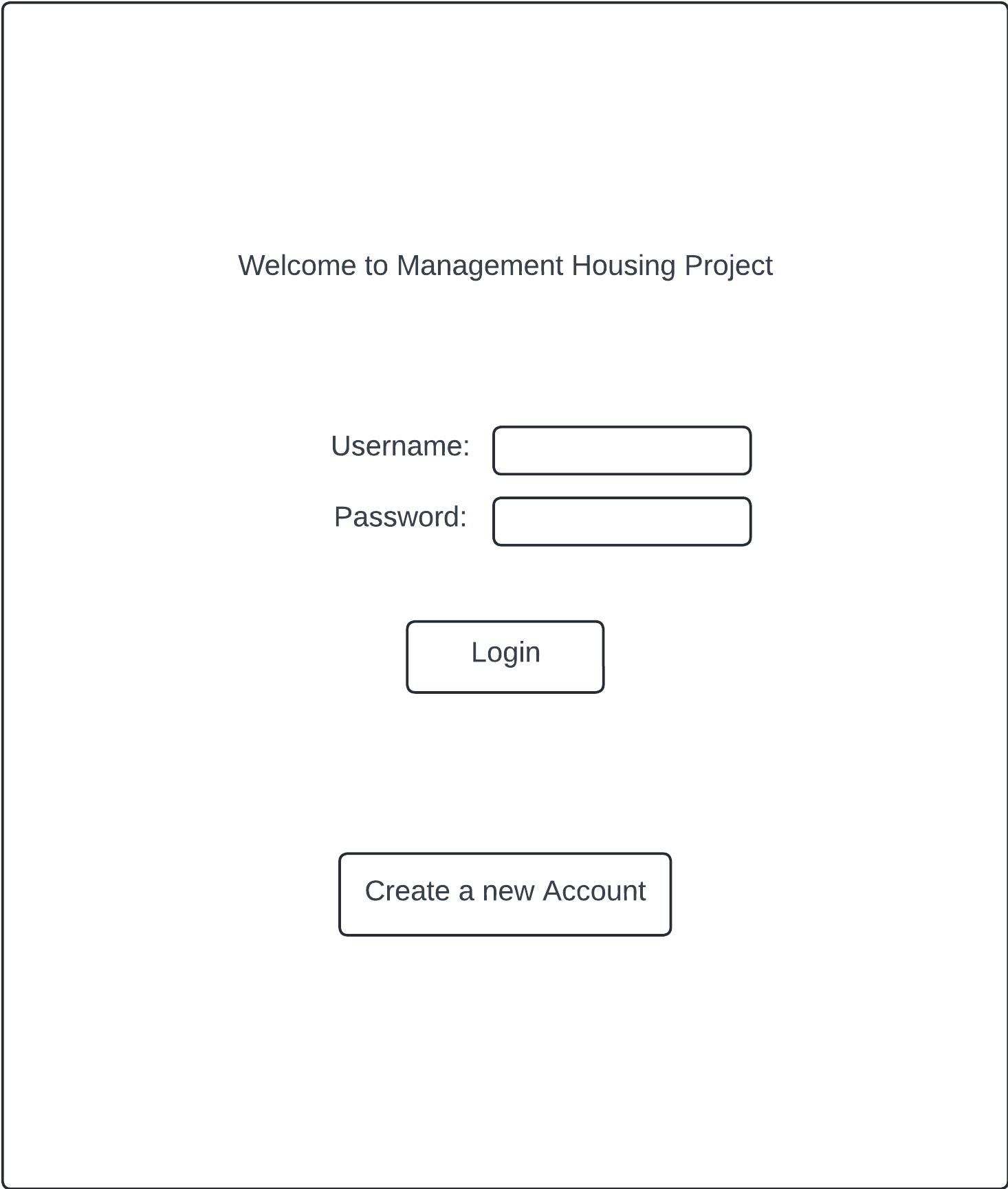
##### 4.1 Management for Agent: Add New Property, Edit Property, Delete Property, View Property Detail

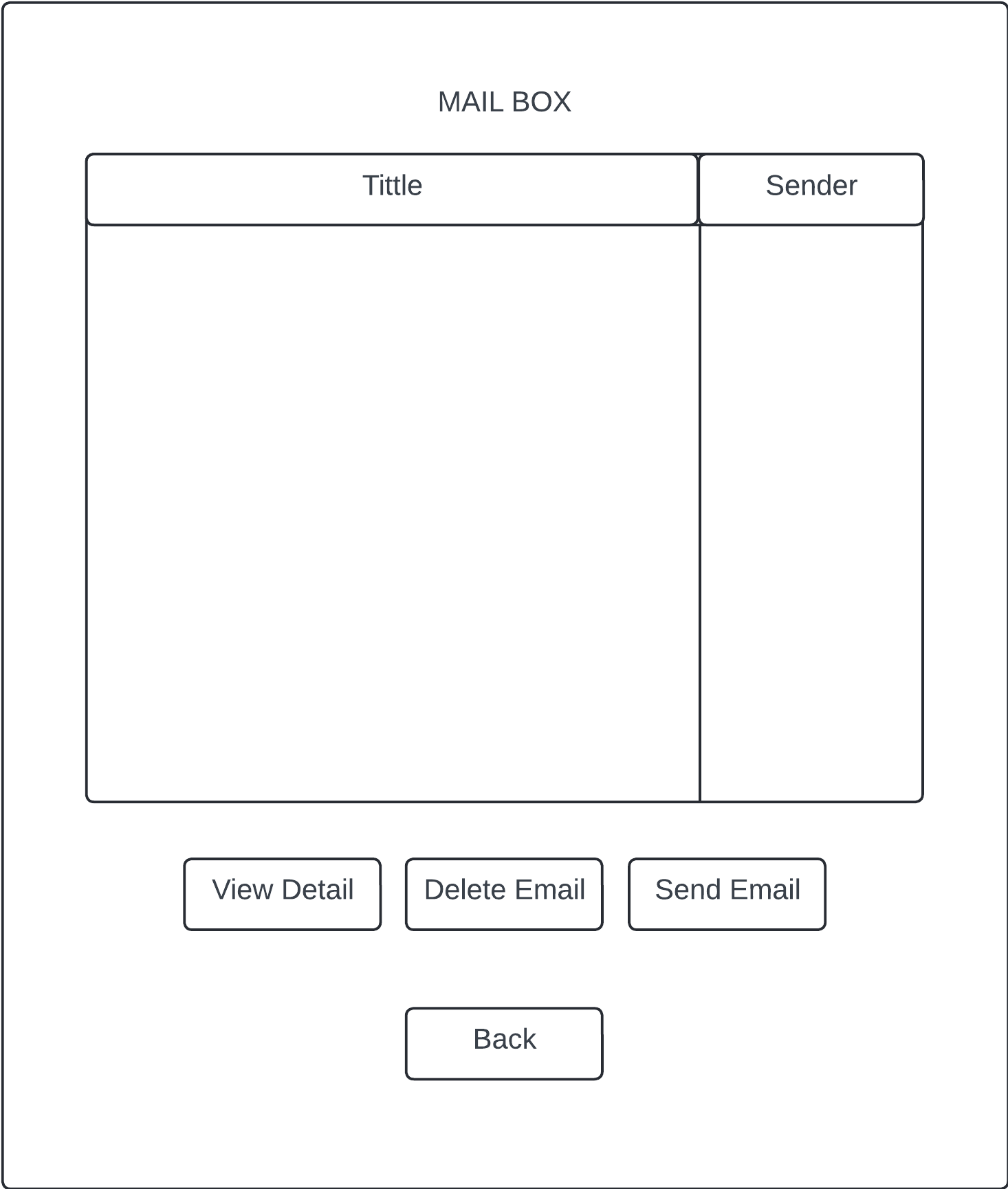
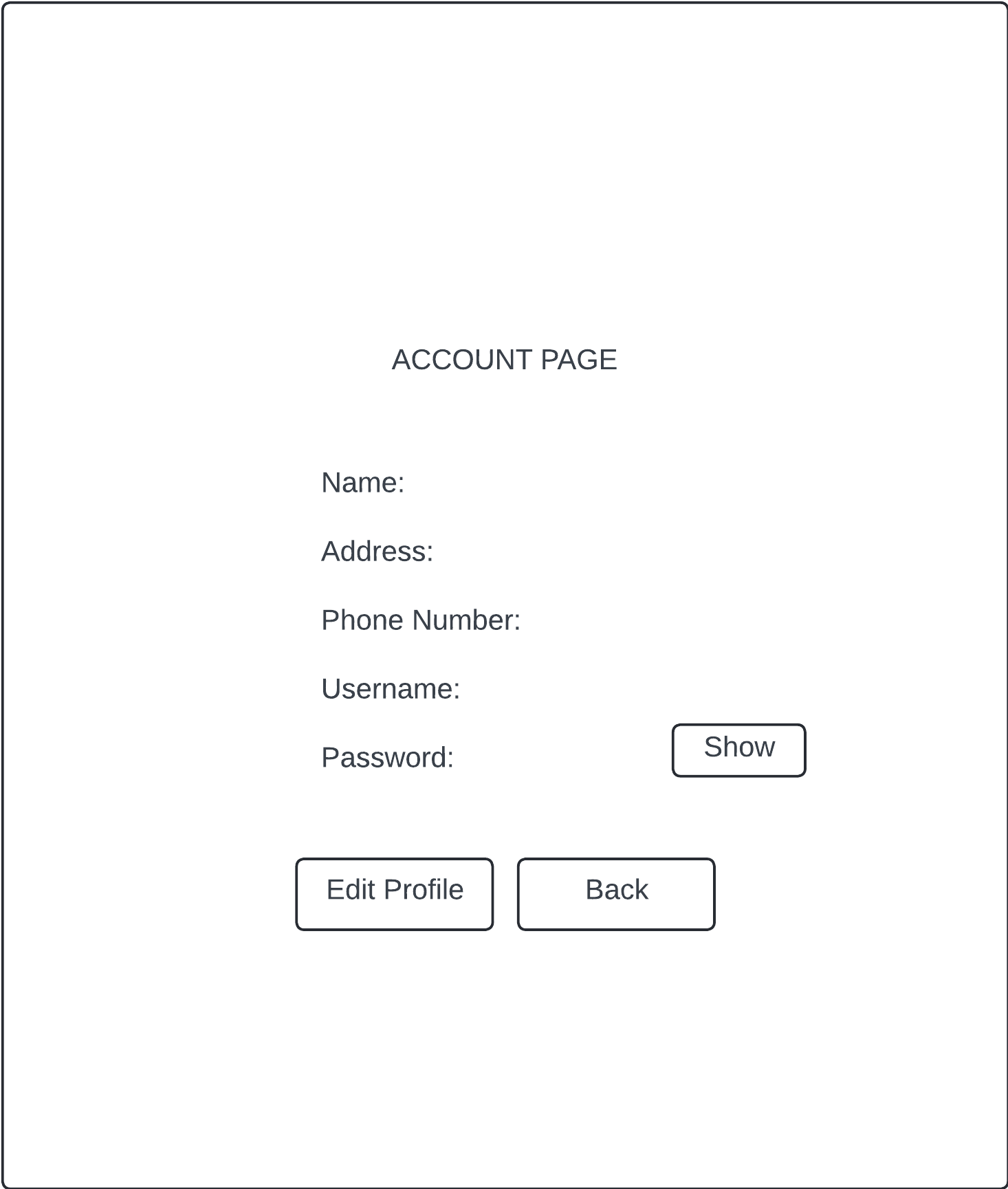
* **Add New Property:**
  + Agents can add a new property by clicking "Add New Property." A form appears for entering property details, which the system validates before adding the property to the agent's inventory.
* **Edit Property:**
  + Agents can update details of an existing property by selecting it and clicking "Edit Property Information." A new window opens for them to enter the updated information, which the system then validates and applies.
* **Delete Property:**
  + Agents can remove a property from their inventory by selecting it and clicking "Remove Property." The system deletes the property and updates all associated tenant leases.
* **View Property Detail:**
  + Agents can view detailed information about a property by selecting it and clicking "View Details." A new window opens to display all the relevant details of the selected property.

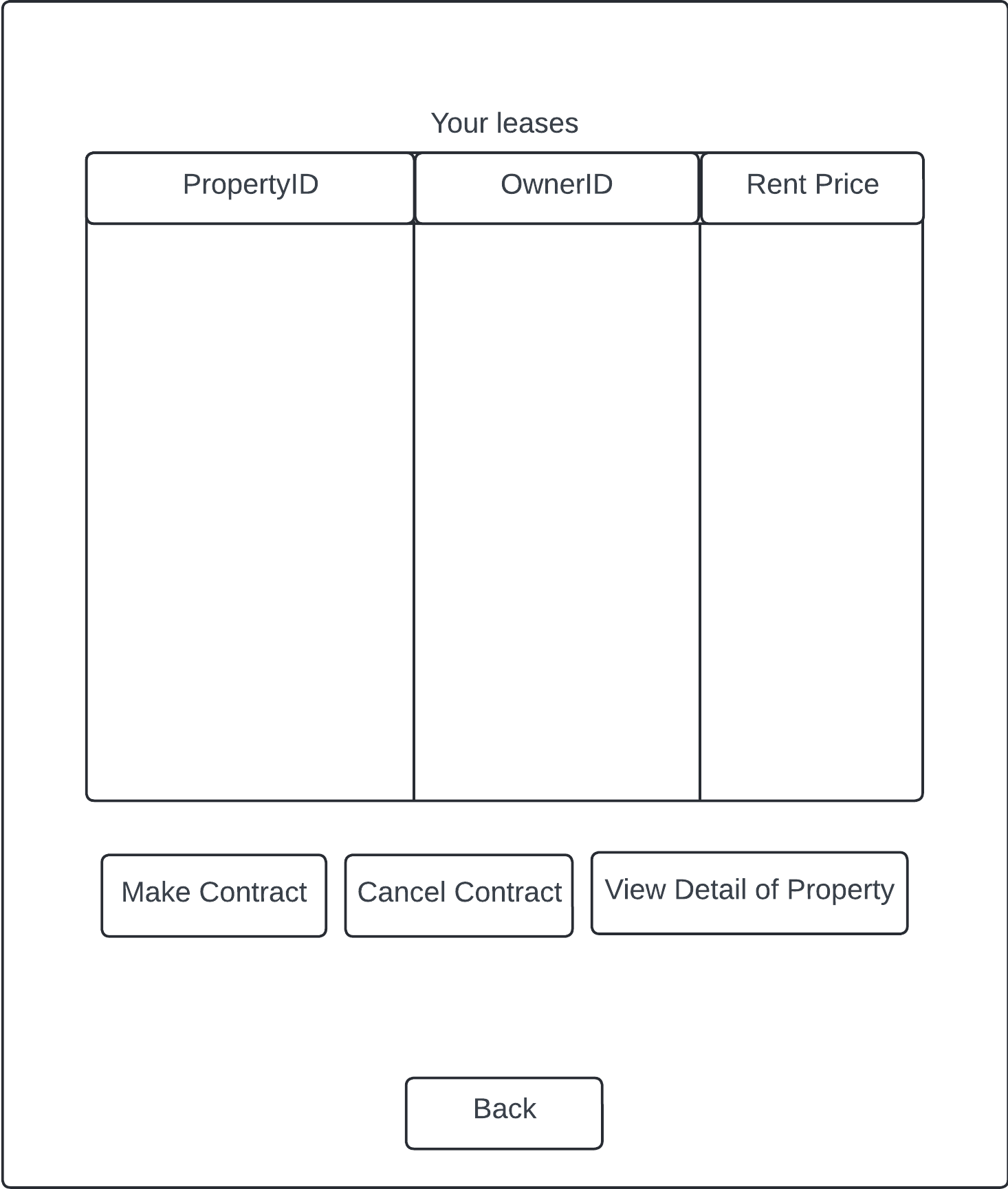
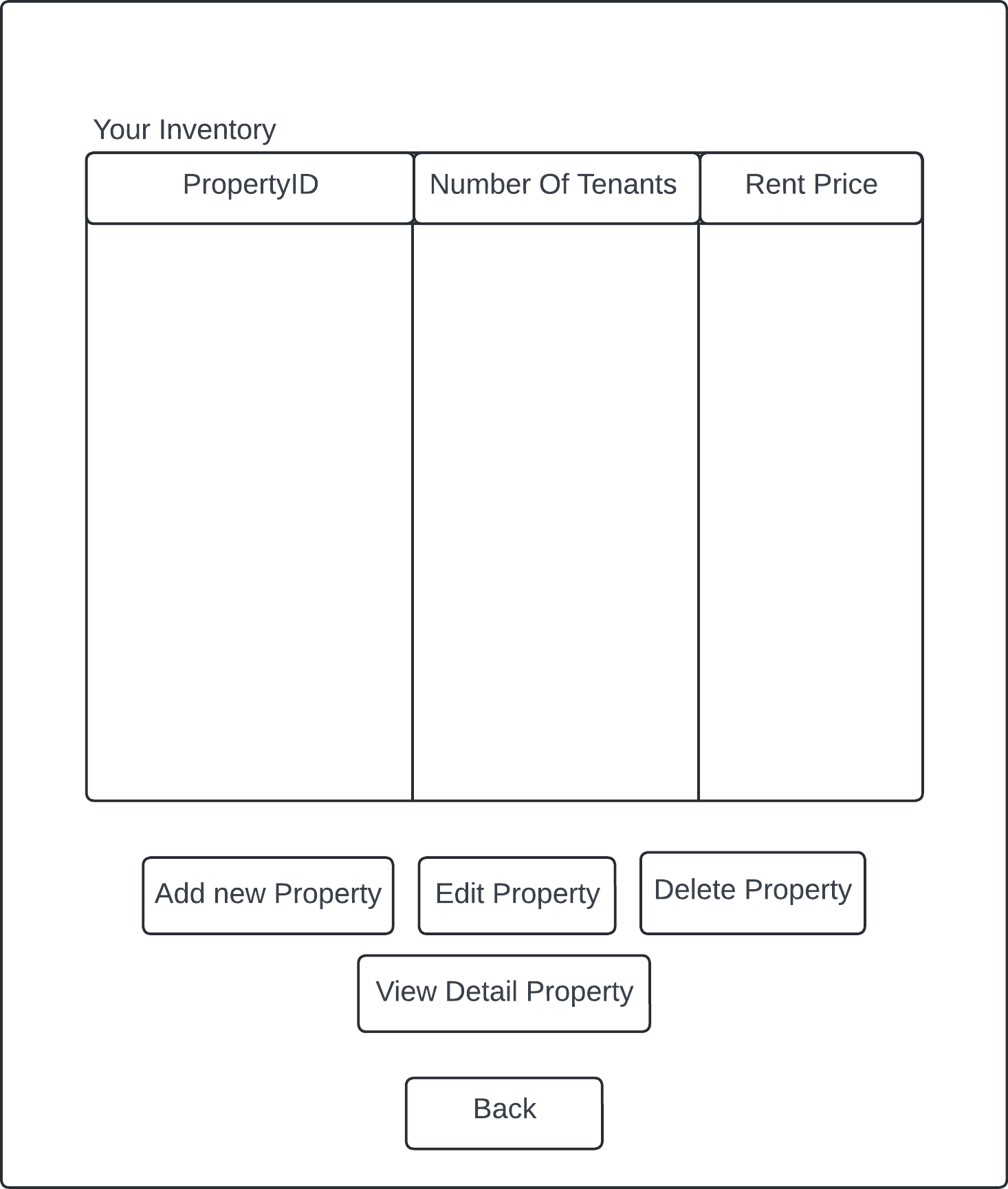
##### 4.2 Management for Client: Make Contract, Cancel Contract, View Property Detail

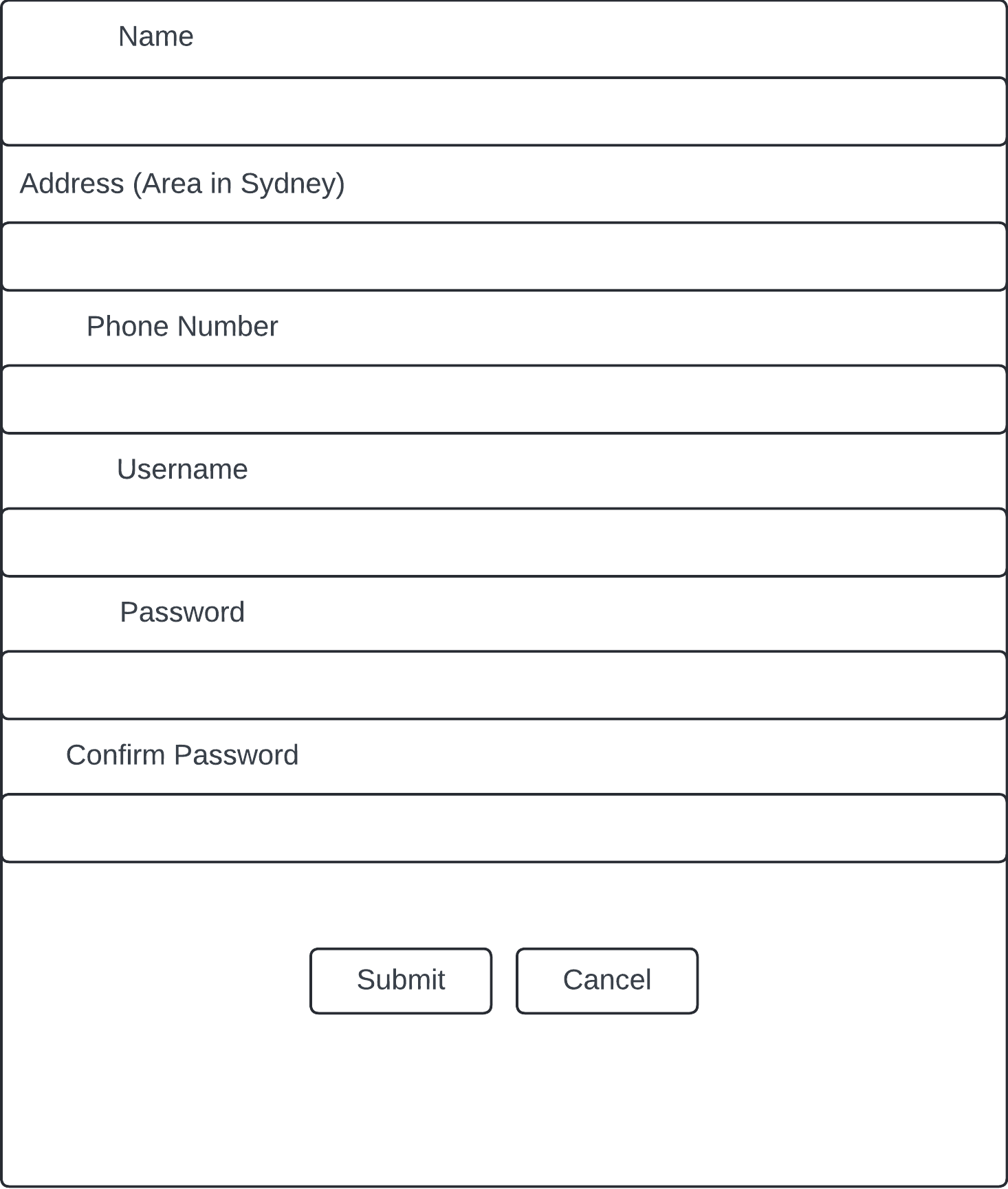
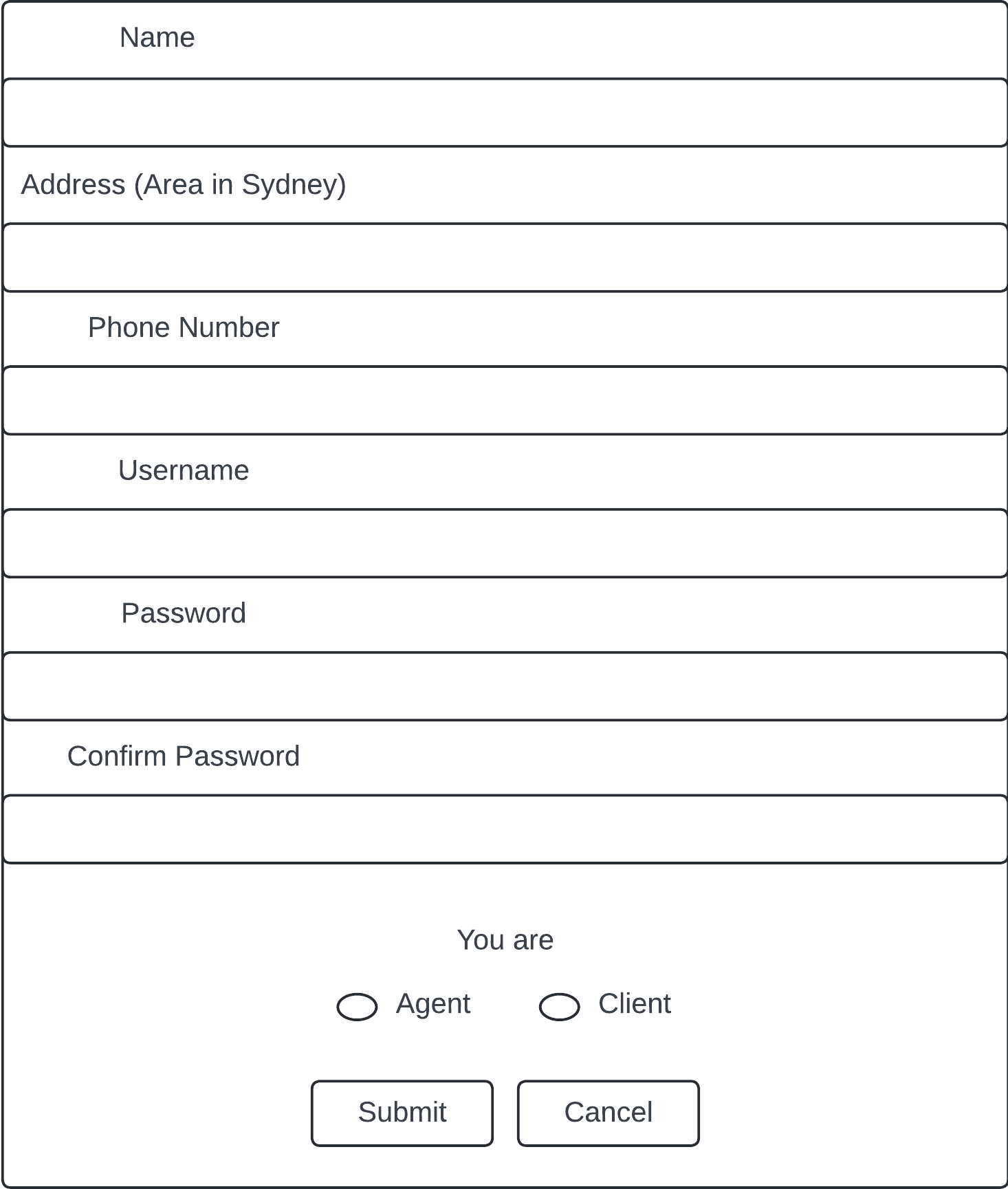
* **Make Contract:**
  + Clients can initiate a rental contract for a property by selecting it from the market and clicking "Make Contract." They confirm the property ID in a new window, and the system finalizes the contract, notifying the owner and adding the property to the client's leases.
* **Cancel Contract:**
  + Clients can cancel an existing rental contract by selecting the property from their list and clicking "Cancel Contract." The system notifies the owner, updates tenant records, and removes the property from the client's leases.
* **View Property Detail:**
  + Clients can view detailed information about a property by selecting it from their list and clicking "View Details." A new window opens, displaying all the property details.

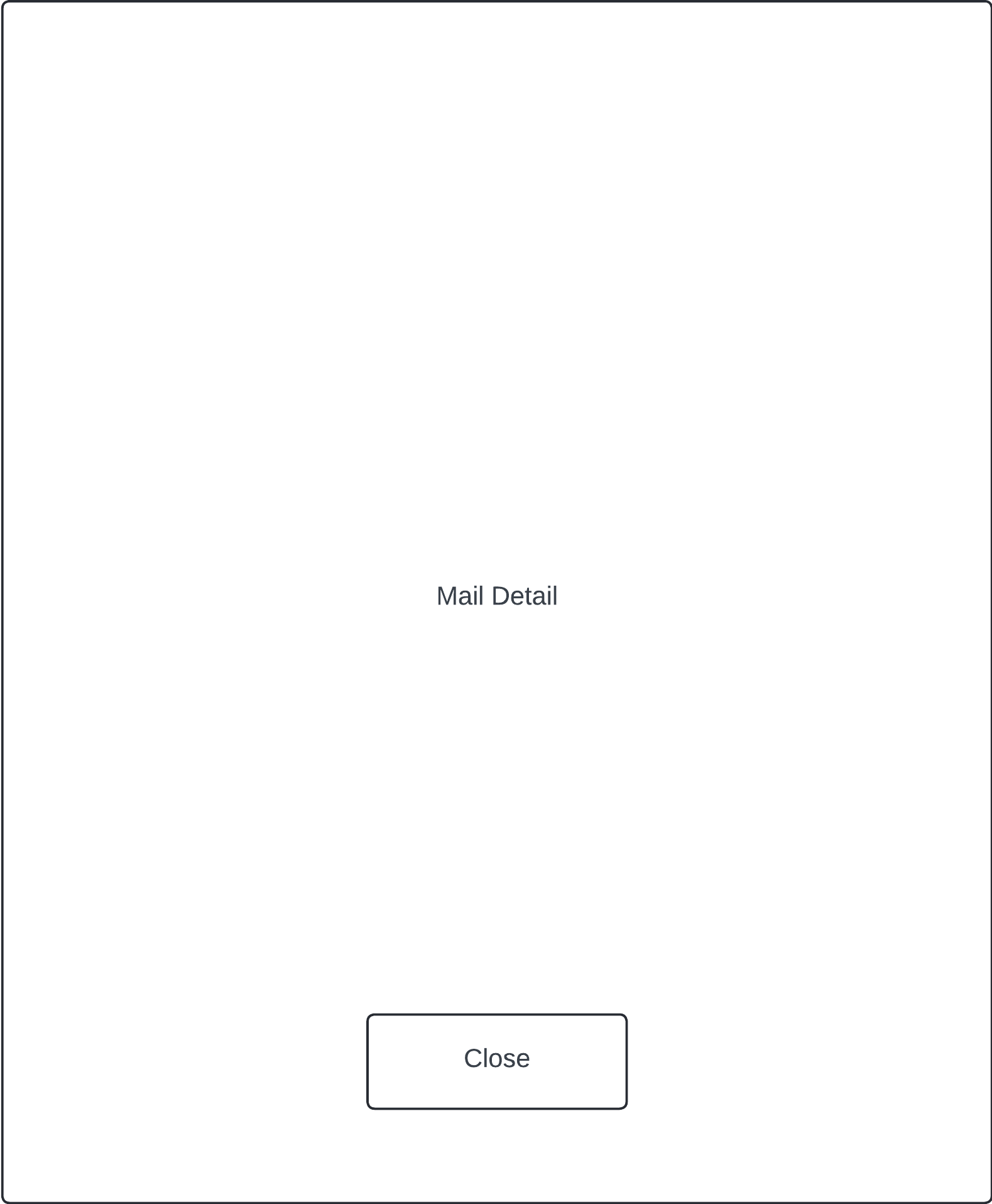
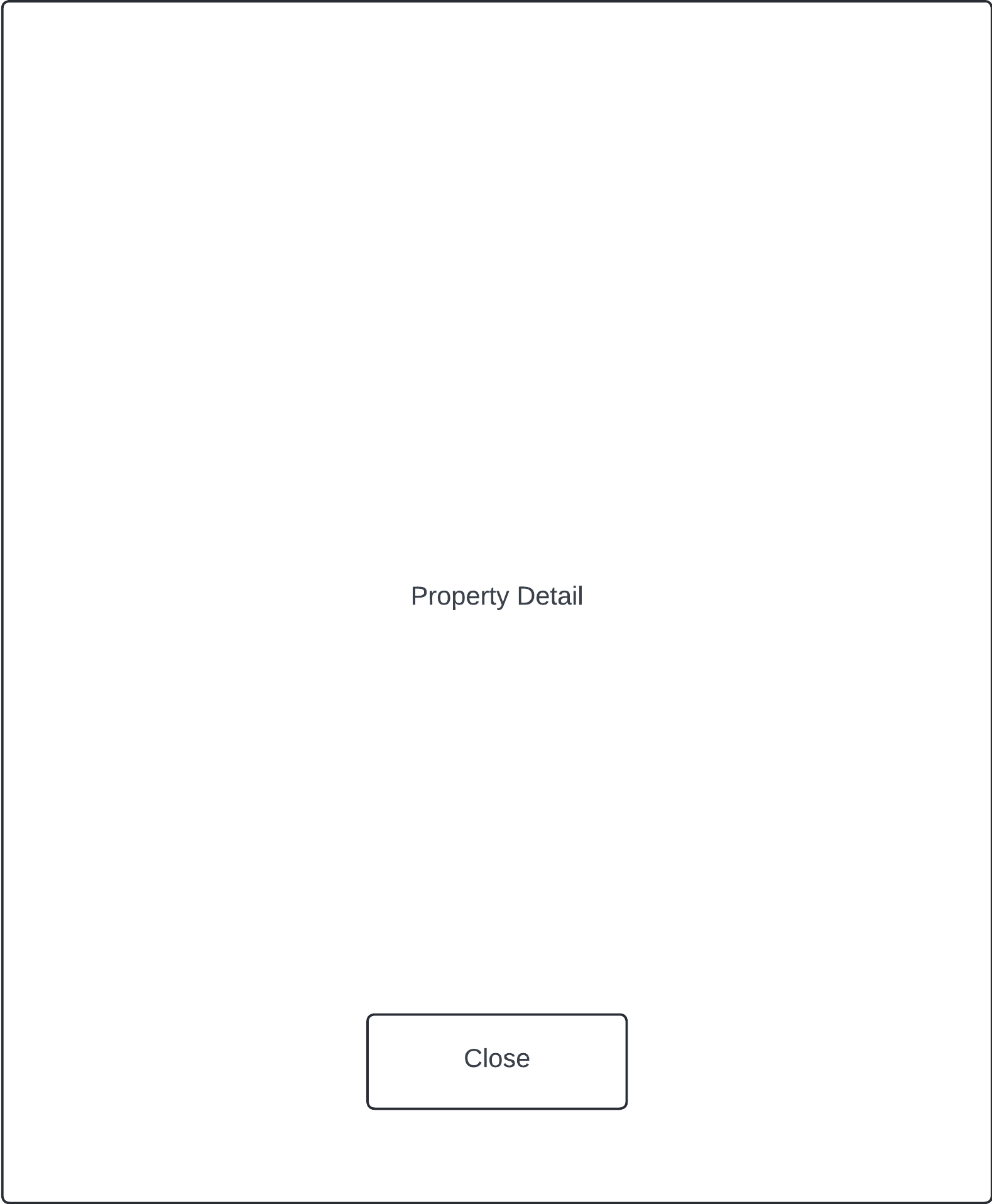
# GUI DESIGN

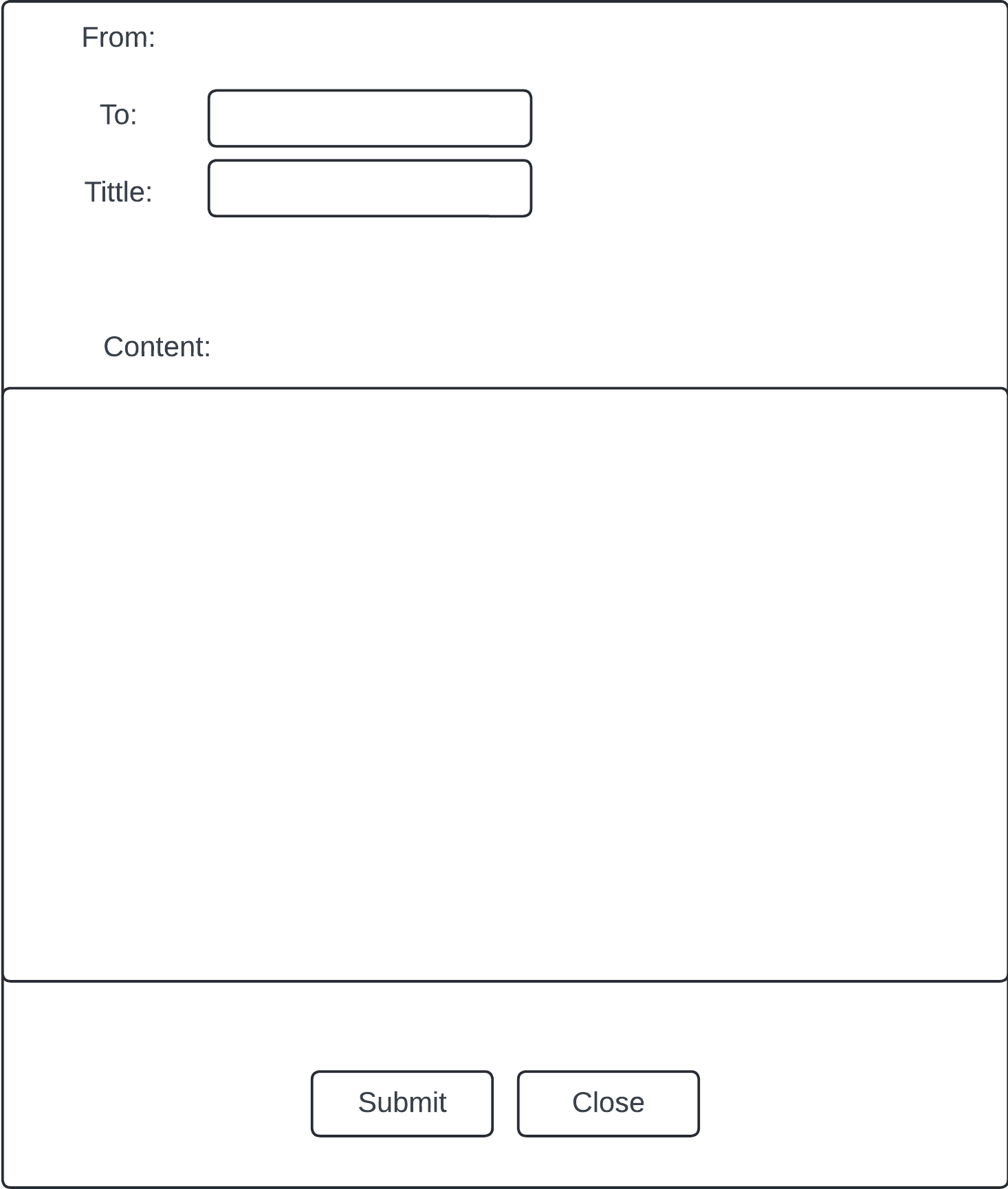
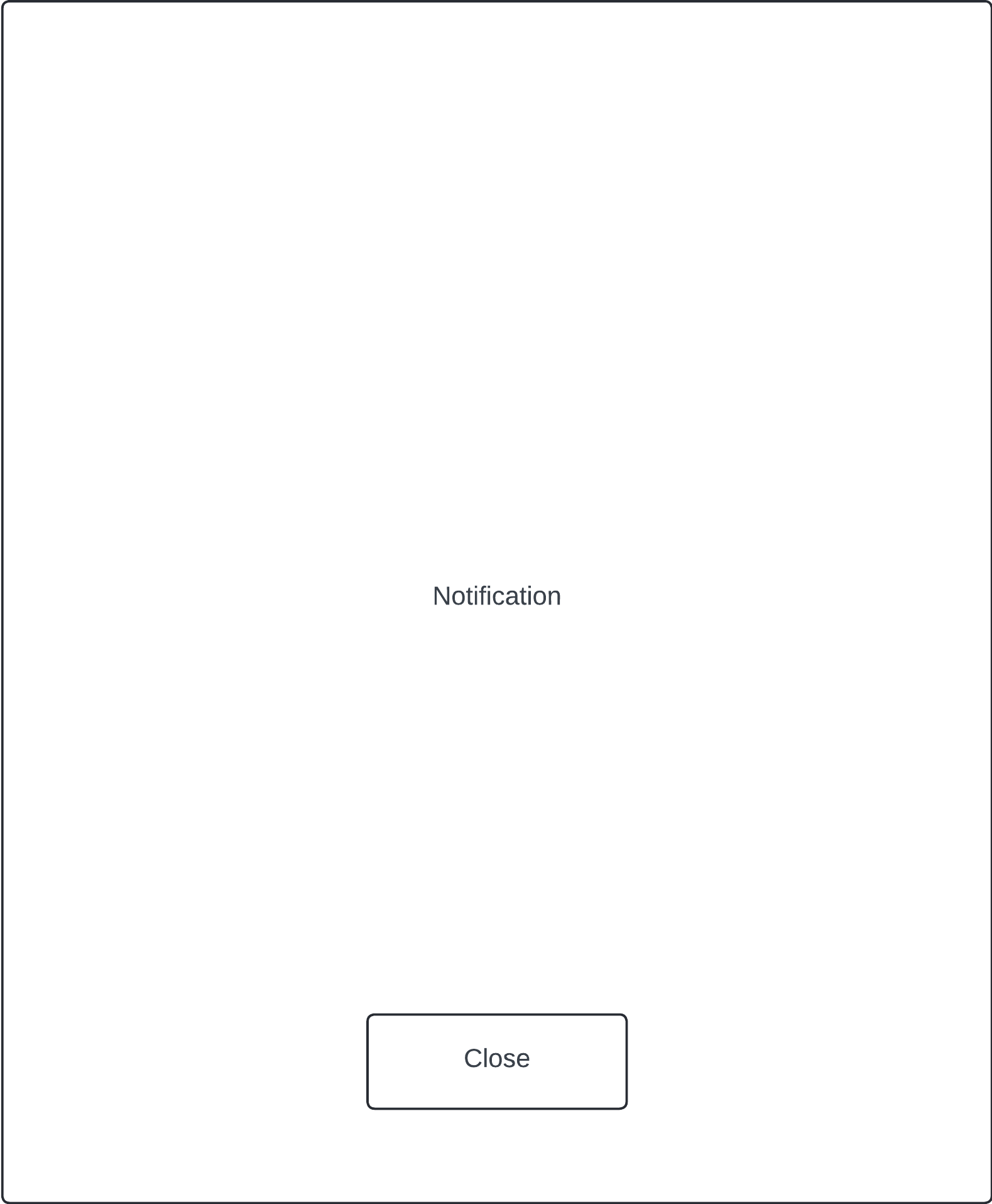


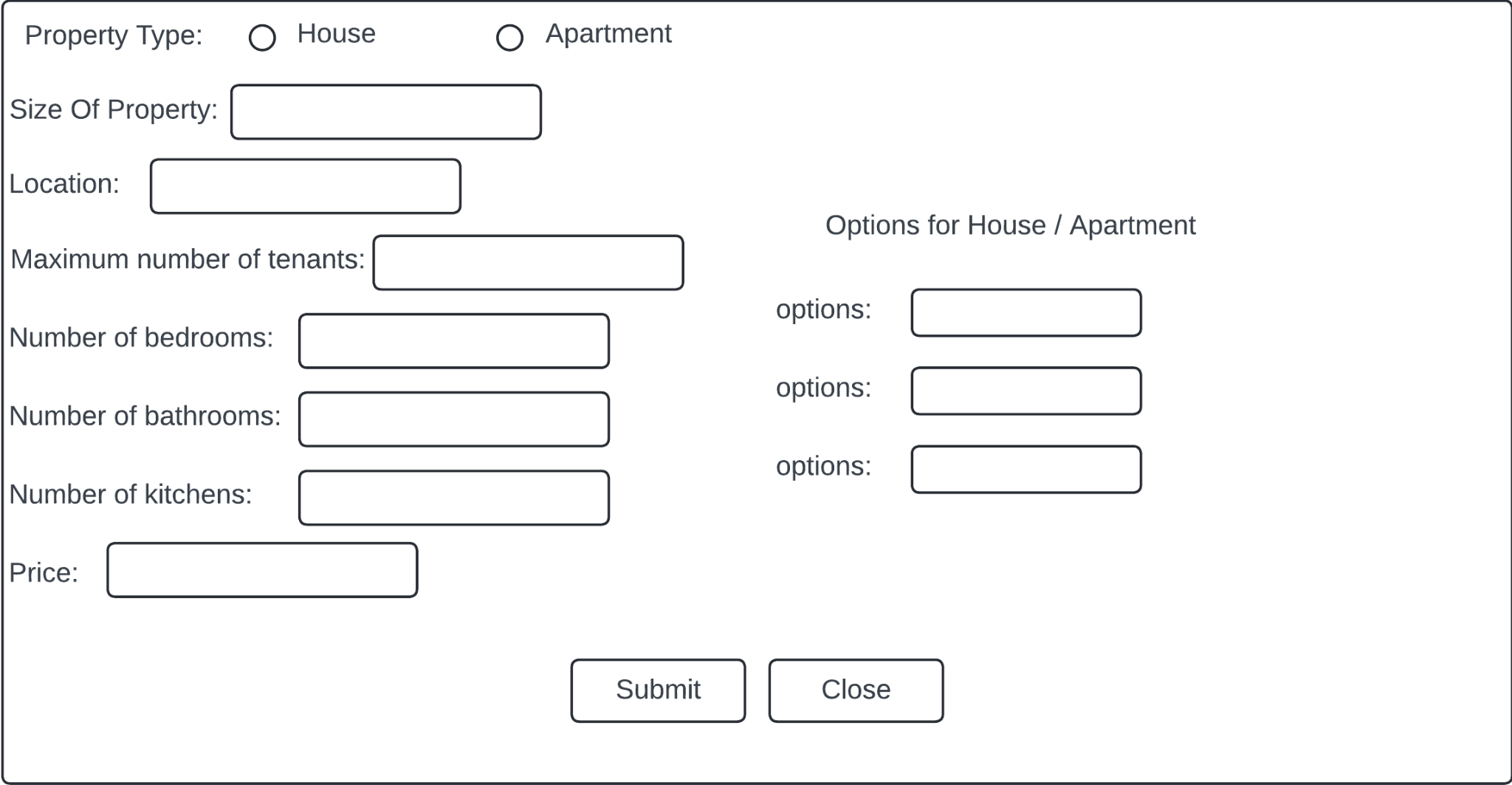
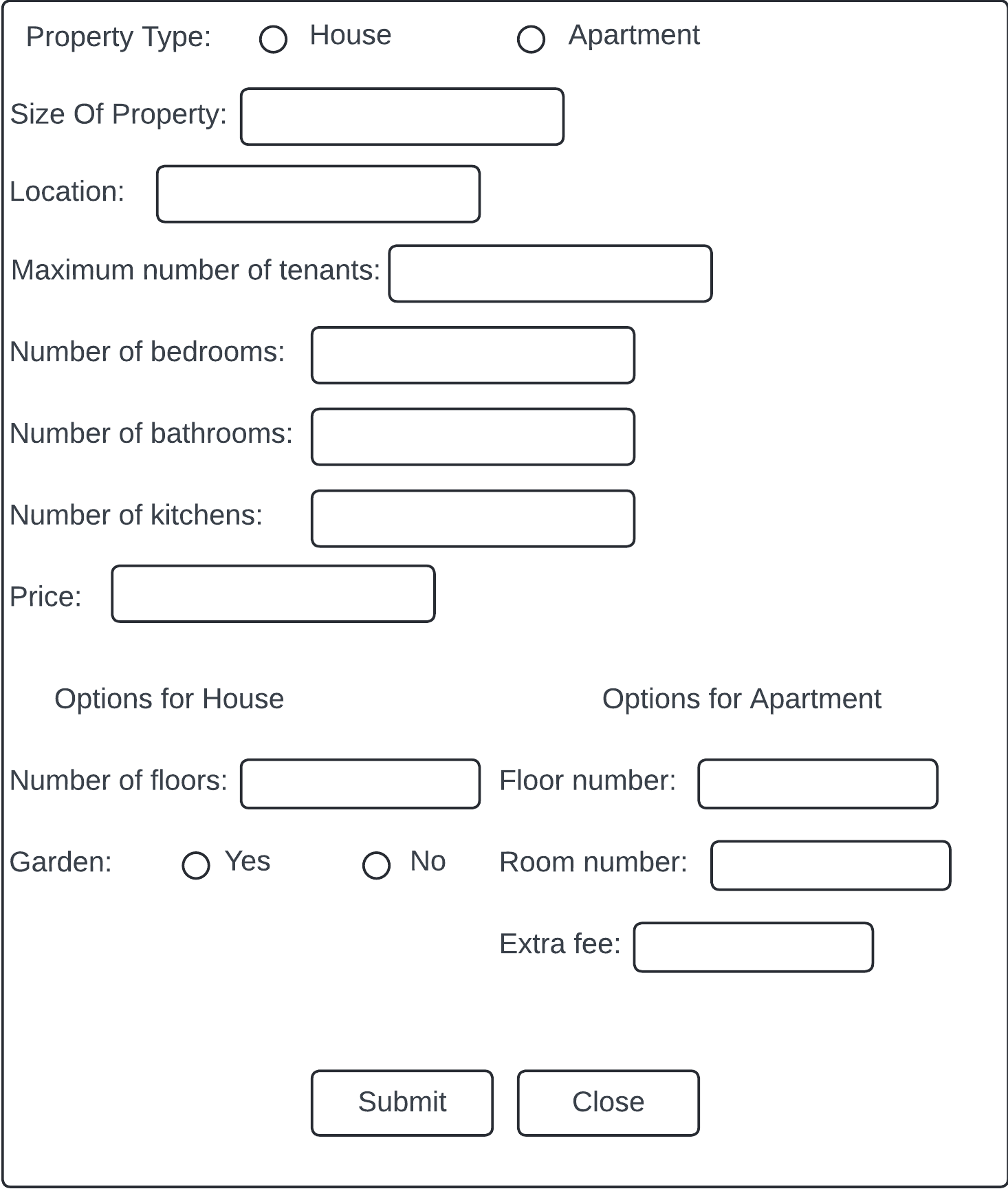


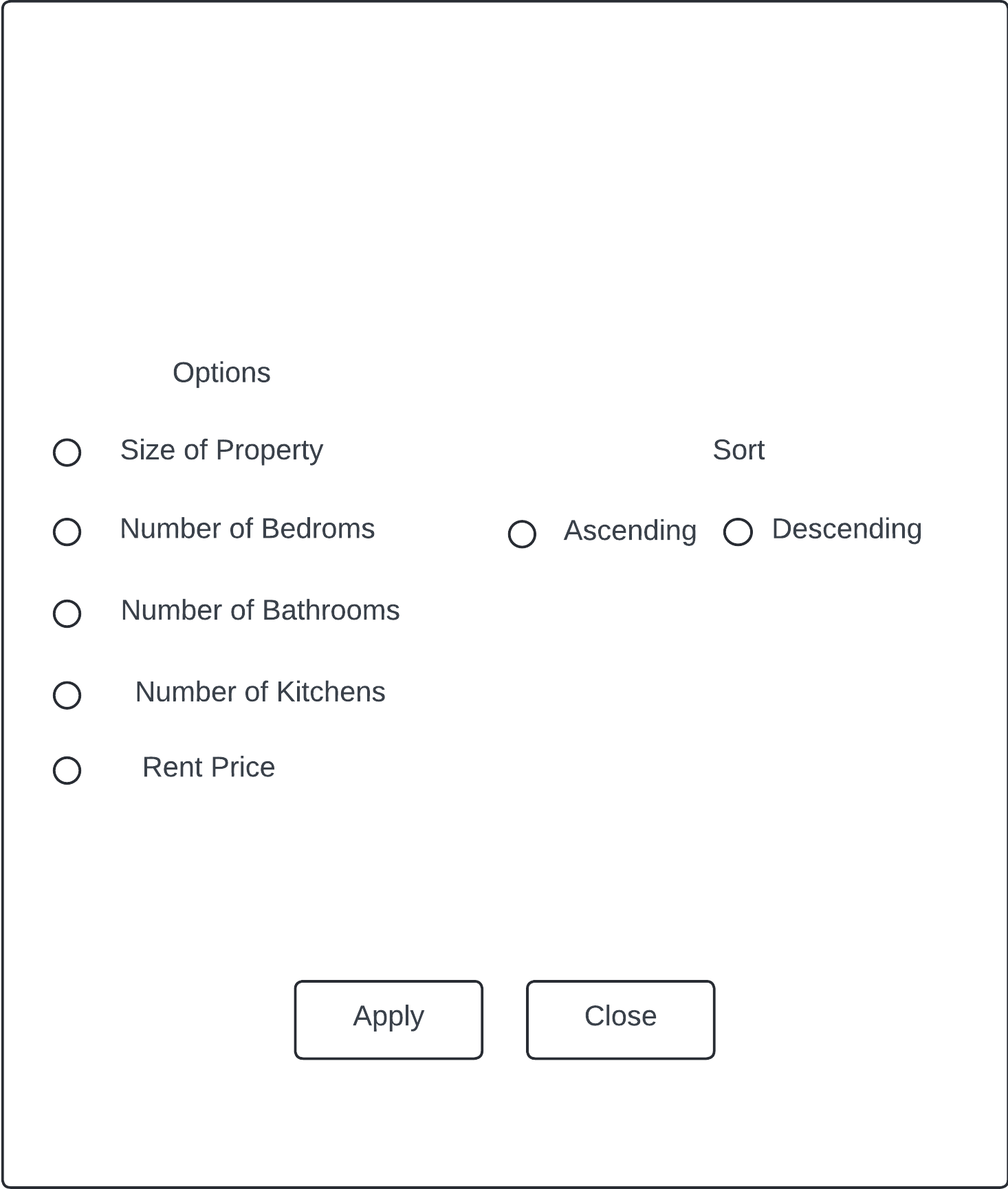












# THE CODE

## AppModel

import java.util.\*;

import javafx.beans.property.*SimpleBooleanProperty*;

import javafx.beans.property.*SimpleDoubleProperty*;

import javafx.beans.property.*SimpleIntegerProperty*;

import javafx.beans.property.*SimpleObjectProperty*;

import javafx.beans.property.*SimpleStringProperty*;

import javafx.collections.*FXCollections*;

import javafx.collections.*ObservableList*;

enum *Address* {

TOWN\_HALL("Town Hall"),

WYNYARD("Wynyard"),

CIRCULAR\_QUAY("Circular Quay"),

ST\_JAMES("St James"),

MUSEUM("Museum"),

CENTRAL("Central"),

REDFERN("Redfern"),

NEWTOWN("Newtown"),

STANMORE("Stanmore"),

PETERSHAM("Petersham"),

LEWISHAM("Lewisham"),

SUMMER\_HILL("Summer Hill"),

ASHFIELD("Ashfield"),

CROYDON("Croydon"),

BURWOOD("Burwood"),

STRATHFIELD("Strathfield"),

NOT\_PREFER("Not Prefer"),

PARRAMATTA("Parramatta"),

WESTMEAD("Westmead"),

AUBURN("Auburn"),

GRANVILLE("Granville"),

MERRYLANDS("Merrylands"),

LIDCOMBE("Lidcombe"),

HOMEBUSH("Homebush"),

RHODES("Rhodes"),

CONCORD("Concord"),

DRUMMOYNE("Drummoyne"),

ROZELLE("Rozelle"),

BALMAIN("Balmain"),

GLEBE("Glebe"),

ULTIMO("Ultimo"),

SURRY\_HILLS("Surry Hills"),

DARLINGHURST("Darlinghurst"),

PADDINGTON("Paddington"),

BONDI("Bondi"),

COOGEE("Coogee"),

MAROUBRA("Maroubra"),

RANDWICK("Randwick"),

KENSINGTON("Kensington"),

MASCOT("Mascot"),

ROSEBERY("Rosebery"),

ZETLAND("Zetland"),

WATERLOO("Waterloo");

private final *String* displayName;

*Address*(*String* *displayName*) {

this.displayName = *displayName*;

}

@Override

public *String* *toString*() {

return displayName;

}

}

public class *AppModel* {

private *User* currentUser;

private *Map*<*String*, *Address*> area;

protected *AppModel*() {

area = new *HashMap*<>();

*List*<*User*> listUsers = new *ArrayList*<>(*Arrays*.*asList*(

*// Agents*

new *Agent*("Cô Chủ Nhà", *Address*.CENTRAL, "0452471326", "a", "a"),

new *Agent*("Adam West", *Address*.TOWN\_HALL, "0412345671", "adamwest", "adampass123"),

new *Agent*("Betty Young", *Address*.CIRCULAR\_QUAY, "0412345672", "bettyyoung", "bettypass789"),

new *Agent*("Carl Zane", *Address*.WYNYARD, "0412345673", "carlzane", "carlpass456"),

new *Agent*("Diana Brown", *Address*.ST\_JAMES, "0412345674", "dianabrown", "dianapass123"),

new *Agent*("Evan Clark", *Address*.MUSEUM, "0412345675", "evanclark", "evanpass456"),

new *Agent*("Fiona Davis", *Address*.CENTRAL, "0412345676", "fionadavis", "fionapass123"),

new *Agent*("George Evans", *Address*.REDFERN, "0412345677", "georgeevans", "georgepass456"),

new *Agent*("Hannah Fox", *Address*.NEWTOWN, "0412345678", "hannahfox", "hannahpass789"),

new *Agent*("Ian Green", *Address*.STANMORE, "0412345679", "iangreen", "iangreen123"),

new *Agent*("Jane Hall", *Address*.PETERSHAM, "0412345680", "janehall", "janepass456"),

new *Agent*("Kyle Iron", *Address*.LEWISHAM, "0412345681", "kyleiron", "kylepass123"),

*// Real Estate Clients*

new *Client*("Anh Thuê Trọ", *Address*.CENTRAL, "0452471325", "b", "b"),

new *Client*("Kathy Lewis", *Address*.LEWISHAM, "0412345691", "kathylewis", "kathypass123"),

new *Client*("Larry Martin", *Address*.SUMMER\_HILL, "0412345692", "larrymartin", "larrypass789"),

new *Client*("Mona Nelson", *Address*.ASHFIELD, "0412345693", "monanelson", "monapass456"),

new *Client*("Nina O'Brien", *Address*.CROYDON, "0412345694", "ninaobrien", "ninapass123"),

new *Client*("Oscar Price", *Address*.BURWOOD, "0412345695", "oscarprice", "oscarpass456"),

new *Client*("Paul Quinn", *Address*.STRATHFIELD, "0412345696", "paulquinn", "paulpass123"),

new *Client*("Quincy Roberts", *Address*.NOT\_PREFER, "0412345697", "quincyroberts", "quincy123"),

new *Client*("Rachel Smith", *Address*.TOWN\_HALL, "0412345698", "rachelsmith", "rachelpass456"),

new *Client*("Steve Thomas", *Address*.CIRCULAR\_QUAY, "0412345699", "stevethomas", "stevepass789"),

new *Client*("Harry Phung", *Address*.CENTRAL, "0412345700", "harry", "123456789"),

new *Client*("Tina Upton", *Address*.WYNYARD, "0412345701", "tinaupton", "tinapass456")

));

*List*<*Property*> listProperties = new *ArrayList*<>(*Arrays*.*asList*(

new *House*(300.5, *Address*.ASHFIELD, 3, 2, 2, 3, 1, true, 950),

new *Apartment*(300.5, *Address*.ASHFIELD, 2, 2, 2, 3, 12, 304, 50, 920),

new *House*(450.0, *Address*.BURWOOD, 4, 3, 3, 4, 2, true, 1800),

new *Apartment*(375.0, *Address*.CROYDON, 3, 2, 1, 2, 1, 23, 30, 1650),

new *House*(320.0, *Address*.STRATHFIELD, 2, 1, 1, 2, 1, false, 1400),

new *Apartment*(500.0, *Address*.NEWTOWN, 5, 4, 3, 4, 3, 123, 60, 1900),

new *House*(285.0, *Address*.LEWISHAM, 2, 1, 1, 1, 0, false, 900),

new *Apartment*(330.0, *Address*.SUMMER\_HILL, 2, 2, 2, 3, 1, 356, 45, 1200),

new *House*(460.0, *Address*.STANMORE, 5, 3, 2, 4, 2, false, 1750),

new *Apartment*(410.0, *Address*.PETERSHAM, 3, 2, 2, 2, 20, 234, 56, 1600),

new *House*(385.0, *Address*.CENTRAL, 5, 3, 1, 3, 1, false, 1950),

new *Apartment*(470.0, *Address*.MUSEUM, 4, 3, 2, 3, 34, 563, 67, 1850)

));

area.*put*("town hall", *Address*.TOWN\_HALL);

area.*put*("wynyard", *Address*.WYNYARD);

area.*put*("circular quay", *Address*.CIRCULAR\_QUAY);

area.*put*("st james", *Address*.ST\_JAMES);

area.*put*("museum", *Address*.MUSEUM);

area.*put*("central", *Address*.CENTRAL);

area.*put*("redfern", *Address*.REDFERN);

area.*put*("newtown", *Address*.NEWTOWN);

area.*put*("stanmore", *Address*.STANMORE);

area.*put*("petersham", *Address*.PETERSHAM);

area.*put*("lewisham", *Address*.LEWISHAM);

area.*put*("summer hill", *Address*.SUMMER\_HILL);

area.*put*("ashfield", *Address*.ASHFIELD);

area.*put*("croydon", *Address*.CROYDON);

area.*put*("burwood", *Address*.BURWOOD);

area.*put*("strathfield", *Address*.STRATHFIELD);

area.*put*("parramatta", *Address*.PARRAMATTA);

area.*put*("westmead", *Address*.WESTMEAD);

area.*put*("auburn", *Address*.AUBURN);

area.*put*("granville", *Address*.GRANVILLE);

area.*put*("merrylands", *Address*.MERRYLANDS);

area.*put*("lidcombe", *Address*.LIDCOMBE);

area.*put*("homebush", *Address*.HOMEBUSH);

area.*put*("rhodes", *Address*.RHODES);

area.*put*("concord", *Address*.CONCORD);

area.*put*("drummoyne", *Address*.DRUMMOYNE);

area.*put*("rozelle", *Address*.ROZELLE);

area.*put*("balmain", *Address*.BALMAIN);

area.*put*("glebe", *Address*.GLEBE);

area.*put*("ultimo", *Address*.ULTIMO);

area.*put*("surry hills", *Address*.SURRY\_HILLS);

area.*put*("darlinghurst", *Address*.DARLINGHURST);

area.*put*("paddington", *Address*.PADDINGTON);

area.*put*("bondi", *Address*.BONDI);

area.*put*("coogee", *Address*.COOGEE);

area.*put*("maroubra", *Address*.MAROUBRA);

area.*put*("randwick", *Address*.RANDWICK);

area.*put*("kensington", *Address*.KENSINGTON);

area.*put*("mascot", *Address*.MASCOT);

area.*put*("rosebery", *Address*.ROSEBERY);

area.*put*("zetland", *Address*.ZETLAND);

area.*put*("waterloo", *Address*.WATERLOO);

area.*put*("not prefer", *Address*.NOT\_PREFER);

for (int i = 0; i < listProperties.*size*() && i < listUsers.*size*(); i++) {

if (listUsers.*get*(i) instanceof *Agent*) {

*Agent* user = (*Agent*) listUsers.*get*(i);

listProperties.*get*(i).*setOwner*(user);

user.*getInventory*().*add*(listProperties.*get*(i));

}

}

}

protected *Map*<*String*, *Address*> *getArea*() {

return area;

}

protected *User* *getCurrentUser*() {

return currentUser;

}

*// welcomePage*

*// check input from controller with existed database and send the result to the controller*

protected boolean *login*(*String* *username*, *String* *password*) {

for (*User* user : *User*.*getUsersList*().*values*()) {

if (*username*.*equals*(user.*getUsername*())) {

if (*password*.*equals*(user.*getPassword*())) {

currentUser = user;

return true;

}

}

}

return false;

}

*//check input from controller and send the result is list of string which is errors when model try to sign up this information*

protected *List*<*String*> *signUp*(*String* *name*, *String* *address*, int *phoneNumber*, *String* *username*, *String* *password*, *String* *confirmPassword*, *String* *role*) {

*List*<*String*> errors = new *ArrayList*<>();

if (*checkPhoneNumber*(*phoneNumber*).*equals*("Valid") && *checkAddress*(*address*) && *checkUsername*(*username*) && *checkPassword*(*password*, *confirmPassword*)) {

if (*role*.*equals*("Agent")) {

new *Agent*(*name*, area.*get*(*address*), "0" + *phoneNumber*, *username*, *password*);

} else {

new *Agent*(*name*, area.*get*(*address*), "0" + *phoneNumber*, *username*, *password*);

}

return errors;

}

if (!*checkPhoneNumber*(*phoneNumber*).*equals*("Valid")) {

errors.*add*("Phone Number is " + *checkPhoneNumber*(*phoneNumber*) + "!!!");

}

if (!*checkAddress*(*address*)) {

errors.*add*("INVALID Address!!!");

}

if (!*checkUsername*(*username*)) {

errors.*add*("This username has been EXISTED!!!");

}

if (!*checkPassword*(*password*, *confirmPassword*)) {

errors.*add*("Password does not match");

}

return errors;

}

*//set currentUser attribute is null when controller requires log out*

protected void *logOut*() {

currentUser = null;

}

*//process information from controller and send list of String if there are some errors happens to controller in order to print out on view*

protected *List*<*String*> *editCurrentUserInfor*(*String* *name*, *String* *address*, int *phoneNumber*, *String* *username*, *String* *password*, *String* *confirmPassword*) {

*List*<*String*> errors = new *ArrayList*<>();

if (*checkPhoneNumber*(*phoneNumber*).*equals*("Valid") && *checkAddress*(*address*) && *checkUsername*(*username*) && *checkPassword*(*password*, *confirmPassword*)) {

if (!*name*.*equals*(currentUser.*getName*())) {

currentUser.*setName*(*name*);

}

if (area.*get*(*address*) != currentUser.*getAddress*()) {

currentUser.*setAddress*(area.*get*(*address*));

}

if (!("0" + *phoneNumber*).*equals*(currentUser.*getPhoneNumber*())) {

currentUser.*setPhoneNumber*("0" + *phoneNumber*);

}

if (!*username*.*equals*(currentUser.*getUsername*())) {

currentUser.*setUsername*(*username*);

}

if (!*password*.*equals*(currentUser.*getPassword*())) {

currentUser.*setPassword*(*password*);

}

}

if (!*checkPhoneNumber*(*phoneNumber*).*equals*("Valid")) {

errors.*add*("Phone Number is " + *checkPhoneNumber*(*phoneNumber*) + "!!!");

}

if (!*checkAddress*(*address*)) {

errors.*add*("INVALID Address!!!");

}

if (!*checkUsername*(*username*)) {

errors.*add*("This username has been EXISTED!!!");

}

if (!*checkPassword*(*password*, *confirmPassword*)) {

errors.*add*("Password does not match");

}

return errors;

}

*//process information from controller and send list of String if there are some errors happens to controller in order to print out on view*

protected *List*<*String*> *sendMail*(*String* *recipient*, *String* *tittle*, *String* *content*) {

*List*<*String*> errors = new *ArrayList*<>();

if (*checkUserID*(*recipient*) && *tittle*.*length*() < 30) {

*User*.*getUsersList*().*get*(*recipient*).*getMails*().*add*(new *Mail*(*tittle*, currentUser.*getUserID*(), *recipient*, *content*));

}

if (!*checkUserID*(*recipient*)) {

errors.*add*("The ID of recipient is not found!!!");

}

if (*tittle*.*length*() >= 30) {

errors.*add*("The tittle is over than 30 characters!!!");

}

return errors;

}

*//sort properties followed by user options*

protected void *sortProperties*(*String* *options*, boolean *ascending*){

if (*options*.*equals*("Size")){

*Property*.*sortPropertiesBySize*(*ascending*);

} else if (*options*.*equals*("Bedrooms")){

*Property*.*sortPropertiesByBedrooms*(*ascending*);

} else if (*options*.*equals*("Bathrooms")){

*Property*.*sortPropertiesByBathrooms*(*ascending*);

} else if (*options*.*equals*("Kitchens")){

*Property*.*sortPropertiesByKitchens*(*ascending*);

} else if (*options*.*equals*("Price")){

*Property*.*sortPropertiesByPrice*(*ascending*);

}

}

*//process information from controller and send list of String if there are some errors happens to controller in order to print out on view*

protected *List*<*String*> *addProperty*(*String* *propertyType*,double *size*, *String* *location*,int *bedrooms*,int *bathrooms*, int *kitchens*, int *maximumNumberOfTenants*, double *rentPrice*, int *floorNumber*,int *roomNumber*, double *extraFee*, int *numberOfFloors*, boolean *garden* ){

*List*<*String*> errors = new *ArrayList*<>();

if (*size* <= 0){

errors.*add*("Size can not be negative or zero");

}

boolean isFound = false;

for (*String* area : area.*keySet*()){

if (*location*.*equals*(area)){

isFound = true;

break;

}

}

if (!isFound) {

errors.*add*("Does not found this location!");

}

if (*bedrooms* < 0 || *bathrooms* < 0 || *kitchens* < 0){

errors.*add*("Rooms can not be negative!");

} else if (*bathrooms* + *bedrooms* + *kitchens* == 0){

errors.*add*("This Property does not have any rooms!");

}

if (*maximumNumberOfTenants* <= 0){

errors.*add*("Maximum number of tenants can not be negative or zero");

}

if (*rentPrice* < 0){

errors.*add*("Rent price can not be negative");

}

if (*extraFee* < 0){

errors.*add*("Extra fee can not be negative");

}

if (*numberOfFloors* < 0){

errors.*add*("Number of floors can not be negative");

}

if (*floorNumber* < 0 || *roomNumber* < 0){

errors.*add*("Floor and Room number can not be negative");

}

if (errors.*isEmpty*()){

((*Agent*) currentUser).*addPropertyToInventory*(*propertyType*, *size*, area.*get*(*location*), *bedrooms*, *bathrooms*, *kitchens*, *maximumNumberOfTenants*, *rentPrice*, *floorNumber*, *roomNumber*, *extraFee*, *numberOfFloors*, *garden* );

}

return errors;

}

*//process information from controller and send list of String if there are some errors happens to controller in order to print out on view*

protected *List*<*String*> *editProperty*(*Property* *property*,double *size*, *String* *location*,int *bedrooms*,int *bathrooms*, int *kitchens*, double *rentPrice*, int *floorNumber*,int *roomNumber*, double *extraFee*, int *numberOfFloors*, boolean *garden*){

*List*<*String*> errors = new *ArrayList*<>();

if (*size* <= 0){

errors.*add*("Size can not be negative or zero");

}

boolean isFound = false;

for (*String* area : area.*keySet*()){

if (*location*.*equals*(area)){

isFound = true;

break;

}

}

if (!isFound) {

errors.*add*("Does not found this location!");

}

if (*bedrooms* < 0 || *bathrooms* < 0 || *kitchens* < 0){

errors.*add*("Rooms can not be negative!");

} else if (*bathrooms* + *bedrooms* + *kitchens* == 0){

errors.*add*("This Property does not have any rooms!");

}

if (*rentPrice* < 0){

errors.*add*("Rent price can not be negative");

}

if (*extraFee* < 0){

errors.*add*("Extra fee can not be negative");

}

if (*numberOfFloors* < 0){

errors.*add*("Number of floors can not be negative");

}

if (*floorNumber* < 0 || *roomNumber* < 0){

errors.*add*("Floor and Room number can not be negative");

}

if (errors.*isEmpty*()){

*property*.*setSizeOfProperty*(*size*);

*property*.*setAddress*(area.*get*(*location*));

*property*.*setPropertyNumberOfBedRooms*(*bedrooms*);

*property*.*setPropertyNumberOfBathRooms*(*bathrooms*);

*property*.*setPropertyNumberOfKitchens*(*kitchens*);

*property*.*setPropertyPrice*(*rentPrice*);

if (*property* instanceof *Apartment*){

((*Apartment*) *property*).*setFloorNumber*(*floorNumber*);

((*Apartment*) *property*).*setRoomNumber*(*roomNumber*);

((*Apartment*) *property*).*setExtraFee*(*extraFee*);

} else if (*property* instanceof *House*){

((*House*) *property*).*setNumberOfFloor*(*numberOfFloors*);

((*House*) *property*).*setGarden*(*garden*);

}

}

return errors;

}

*//process information from controller and send list of String if there are some errors happens to controller in order to print out on view*

protected *String* *makeContract*(*String* *id*){

boolean isFound = false;

*Property* target = null;

for (*Property* property : *Property*.*getPropertyObservableList*()){

if (*id*.*equals*(property.*getPropertyID*())){

target = property;

isFound = true;

break;

}

}

if (isFound){

boolean hasProperty = false;

for (*Property* lease : ((*Client*) currentUser).*getLeases*()){

if (lease.*getPropertyID*().*equals*(target.*getPropertyID*())){

hasProperty = true;

break;

}

}

if (!hasProperty){

if (target.*getCountTenants*() >= target.*getNumberOfTenants*()){

return "This Property is full!";

} else {

((*Client*) currentUser).*makeContract*(target);

target.*getOwner*().*getMails*().*add*(new *Mail*("The property " + target.*getPropertyID*() + " has a new tenant!","System",target.*getOwner*().*getUserID*(),"We are pleased to inform you that a new tenant has been successfully\n registered for your property "+ target.*getPropertyID*() +"\n. Below are the details of the new tenant:"+ currentUser.*toString*()+ "\nnKind regards,\nThe Customer Service Team"));

target.*setCountTenants*(target.*getCountTenants*() + 1);

target.*getTenants*().*add*((*Client*) currentUser);

currentUser.*getMails*().*add*(new *Mail*("Welcome to Your New Home!","System",target.*getOwner*().*getUserID*(),"Congratulations! We are excited to inform you that your application to\n rent the property "+ target.*getPropertyID*() +" has been successfully processed.\nKind regards,\nThe Customer Service Team"));

return "";

}

} else {

return "You have already rented this property!";

}

}

return "Does not found this PropertyID";

}

*//process information from controller and send list of String if there are some errors happens to controller in order to print out on view*

protected void *cancelContract*(*Property* *property*){

*Client* client = (*Client*) currentUser;

client.*getLeases*().*remove*(*property*);

*property*.*getTenants*().*remove*(client);

*property*.*setCountTenants*(*property*.*getCountTenants*() - 1);

currentUser.*getMails*().*add*(new *Mail*("Cancellation of Your Rental Agreement","System",client.*getUserID*(),"We have received your request to cancel the rental agreement for the property with ID: "+ *property*.*getPropertyID*() +"\nYour cancellation has been successfully processed.\n" + "We are sorry to see you go and hope to assist you in the future.\n" + "Kind regards,\n" + "The Customer Service Team"));

*property*.*getOwner*().*getMails*().*add*(new *Mail*("Tenant Cancellation of Rental Agreement","System",client.*getUserID*(),"We would like to inform you that the tenant "+client.*getUserID*() +" has requested to cancel the rental agreement.\n" + "We understand this may be an inconvenience, and we are here to help you find a new tenant as quickly as possible.\n" + "If you have any questions or need further assistance, please feel free to contact our customer service team.\n" + "Kind regards,\n" + "The Customer Service Team"));

}

*//are used in main method to check information of users, current user and properties*

protected boolean *checkAddress*(*String* *address*) {

for (*String* s : area.*keySet*()) {

if (*address*.*equals*(s)) {

return true;

}

}

return false;

}

protected *String* *checkPhoneNumber*(int *phoneNumber*) {

if (*phoneNumber* > 400000000 && *phoneNumber* < 500000000) {

*String* phoneNumberString = "0" + *phoneNumber*;

for (*User* user : *User*.*getUsersList*().*values*()) {

if ( currentUser != null) {

if (phoneNumberString.*equals*(currentUser.*getPhoneNumber*())) {

return "Valid";

}

}

if (phoneNumberString.*equals*(user.*getPhoneNumber*())) {

return "Existed";

}

}

return "Valid";

}

return "Invalid";

}

protected boolean *checkUsername*(*String* *username*) {

for (*User* user : *User*.*getUsersList*().*values*()) {

if (currentUser != null) {

if (*username*.*equals*(currentUser.*getUsername*())) {

return true;

}

}

if (*username*.*equals*(user.*getUsername*())){

return false;

}

}

return true;

}

protected boolean *checkPassword*(*String* *password*, *String* *confirmPassword*) {

return *password*.*equals*(*confirmPassword*);

}

protected boolean *checkUserID*(*String* *id*) {

for (*String* userID : *User*.*getUsersList*().*keySet*()) {

if (*id*.*equals*(userID)) {

return true;

}

}

return false;

}

protected *Property* *checkPropertyID*(*String* *id*){

for (*Property* property : *Property*.*getPropertyObservableList*()){

if (*id*.*equals*(property.*getPropertyID*())){

return property;

}

}

return null;

}

}

abstract class *Property* {

private *User* owner;

private *List*<*Client*> tenants;

private *SimpleIntegerProperty* numberOfTenants;

private *SimpleIntegerProperty* countTenants;

private *SimpleDoubleProperty* price;

private final *SimpleStringProperty* propertyID;

private *SimpleDoubleProperty* sizeOfProperty;

private *SimpleObjectProperty*<*Address*> address;

private *SimpleIntegerProperty* numberOfBedRooms;

private *SimpleIntegerProperty* numberOfKitchens;

private *SimpleIntegerProperty* numberOfBathRooms;

private static *ObservableList*<*Property*> propertyObservableList = *FXCollections*.*observableArrayList*();

protected static final *Comparator*<*Property*> comparatorSize = *Comparator*.*comparing*(*Property*::*getSizeOfProperty*);

protected static final *Comparator*<*Property*> comparatorBedrooms = *Comparator*.*comparing*(*Property*::*getNumberOfBedRooms*);

protected static final *Comparator*<*Property*> comparatorBathrooms = *Comparator*.*comparing*(*Property*::*getNumberOfBathRooms*);

protected static final *Comparator*<*Property*> comparatorKitchens = *Comparator*.*comparing*(*Property*::*getNumberOfKitchens*);

protected static final *Comparator*<*Property*> comparatorPrice = *Comparator*.*comparing*(*Property*::*getPrice*);

protected *Property*(*User* *owner*, double *sizeOfProperty*, *Address* *address*, int *numberOfTenants*, int *numberOfBedRooms*, int *numberOfKitchens*, int *numberOfBathRooms*, double *price*) {

this.owner = *owner*;

this.sizeOfProperty = new *SimpleDoubleProperty*(*sizeOfProperty*);

this.address = new *SimpleObjectProperty*<>(*address*);

this.tenants = new *ArrayList*<>();

this.countTenants = new *SimpleIntegerProperty*(0);

this.numberOfTenants = new *SimpleIntegerProperty*(*numberOfTenants*);

this.numberOfBedRooms = new *SimpleIntegerProperty*(*numberOfBedRooms*);

this.numberOfKitchens = new *SimpleIntegerProperty*(*numberOfKitchens*);

this.numberOfBathRooms = new *SimpleIntegerProperty*(*numberOfBathRooms*);

this.price = new *SimpleDoubleProperty*(*price*);

this.propertyID = new *SimpleStringProperty*(*setPropertyID*());

*Property*.propertyObservableList.*add*(this);

}

protected *Property*(double *sizeOfProperty*, *Address* *address*, int *numberOfTenants*, int *numberOfBedRooms*, int *numberOfKitchens*, int *numberOfBathRooms*, double *price*) {

this.sizeOfProperty = new *SimpleDoubleProperty*(*sizeOfProperty*);

this.address = new *SimpleObjectProperty*<>(*address*);

this.countTenants = new *SimpleIntegerProperty*(0);

this.tenants = new *ArrayList*<>();

this.numberOfTenants = new *SimpleIntegerProperty*(*numberOfTenants*);

this.numberOfBedRooms = new *SimpleIntegerProperty*(*numberOfBedRooms*);

this.numberOfKitchens = new *SimpleIntegerProperty*(*numberOfKitchens*);

this.numberOfBathRooms = new *SimpleIntegerProperty*(*numberOfBathRooms*);

this.price = new *SimpleDoubleProperty*(*price*);

this.propertyID = new *SimpleStringProperty*(*setPropertyID*());

*Property*.propertyObservableList.*add*(this);

}

protected *SimpleDoubleProperty* *getPropertySizeOfProperty*() {

return sizeOfProperty;

}

protected *SimpleObjectProperty*<*Address*> *getPropertyAddress*() {

return address;

}

protected *User* *getOwner*() {

return owner;

}

protected *SimpleIntegerProperty* *getPropertyNumberOfBathRooms*() {

return numberOfBathRooms;

}

protected *SimpleIntegerProperty* *getPropertyNumberOfBedRooms*() {

return numberOfBedRooms;

}

protected *SimpleIntegerProperty* *getPropertyNumberOfKitchens*() {

return numberOfKitchens;

}

protected *SimpleIntegerProperty* *getPropertyNumberOfTenants*() {

return numberOfTenants;

}

protected *SimpleDoubleProperty* *getPropertyPrice*() {

return price;

}

protected *SimpleStringProperty* *getPropertyPropertyID*() {

return propertyID;

}

protected *SimpleIntegerProperty* *getPropertyCountTenants*() {

return countTenants;

}

protected void *setSizeOfProperty*(double *size*) {

this.sizeOfProperty.*set*(*size*);

}

protected void *setAddress*(*Address* *address*) {

this.address = new *SimpleObjectProperty*<>(*address*);

}

protected void *setOwner*(*User* *owner*) {

this.owner = *owner*;

}

protected void *setPropertyNumberOfBathRooms*(int *numberOfBathRooms*) {

this.numberOfBathRooms.*set*(*numberOfBathRooms*);

}

protected void *setPropertyNumberOfBedRooms*(int *numberOfBedRooms*) {

this.numberOfBedRooms.*set*(*numberOfBedRooms*);

}

protected void *setPropertyNumberOfKitchens*(int *numberOfKitchens*) {

this.numberOfKitchens.*set*(*numberOfKitchens*);

}

protected void *setPropertyPrice*(double *price*) {

this.price.*set*(*price*);

}

protected void *setCountTenants*(int *countTenants*) {

this.countTenants.*set*(*countTenants*);

}

protected *String* *setPropertyID*() {

*String* idsetup;

while (true) {

*Random* rand = new *Random*();

int num1 = rand.*nextInt*(10);

int num2 = rand.*nextInt*(10);

int num3 = rand.*nextInt*(10);

int num4 = rand.*nextInt*(10);

idsetup = "PR" + num1 + num2 + num3 + num4;

boolean isFound = false;

for (*Property* property : *Property*.propertyObservableList){

if (idsetup.*equals*(property.*getPropertyID*())){

isFound = true;

break;

}

}

if (isFound){

continue;

}

break;

}

return idsetup;

}

protected double *getSizeOfProperty*() {

return *getPropertySizeOfProperty*().*get*();

}

protected *String* *getPropertyID*() {

return propertyID.*get*();

}

protected int *getNumberOfBathRooms*() {

return *getPropertyNumberOfBathRooms*().*get*();

}

protected int *getNumberOfBedRooms*() {

return *getPropertyNumberOfBedRooms*().*get*();

}

protected int *getNumberOfKitchens*() {

return *getPropertyNumberOfKitchens*().*get*();

}

protected int *getNumberOfTenants*() {

return *getPropertyNumberOfTenants*().*get*();

}

protected double *getPrice*() {

return *getPropertyPrice*().*get*();

}

protected *Address* *getAddress*() {

return *getPropertyAddress*().*get*();

}

protected int *getCountTenants*() {

return *getPropertyCountTenants*().*get*();

}

protected *List*<*Client*> *getTenants*() {

return tenants;

}

protected static void *sortPropertiesBySize*(boolean *ascending*){

*Collections*.*sort*(*getPropertyObservableList*(),*Property*.comparatorSize);

if (!*ascending*) {

*Collections*.*reverse*(*getPropertyObservableList*());

}

}

protected static void *sortPropertiesByBedrooms*(boolean *ascending*){

*Collections*.*sort*(*getPropertyObservableList*(),*Property*.comparatorBedrooms);

if (!*ascending*) {

*Collections*.*reverse*(*getPropertyObservableList*());

}

}

protected static void *sortPropertiesByBathrooms*(boolean *ascending*){

*Collections*.*sort*(*getPropertyObservableList*(),*Property*.comparatorBathrooms);

if (!*ascending*) {

*Collections*.*reverse*(*getPropertyObservableList*());

}

}

protected static void *sortPropertiesByKitchens*(boolean *ascending*){

*Collections*.*sort*(*getPropertyObservableList*(),*Property*.comparatorKitchens);

if (!*ascending*) {

*Collections*.*reverse*(*getPropertyObservableList*());

}

}

protected static void *sortPropertiesByPrice*(boolean *ascending*){

*Collections*.*sort*(*getPropertyObservableList*(),*Property*.comparatorPrice);

if (!*ascending*) {

*Collections*.*reverse*(*getPropertyObservableList*());

}

}

protected static *ObservableList*<*Property*> *getPropertyObservableList*() {

return propertyObservableList;

}

abstract *String* *detailInfo*();

@Override

public *String* *toString*() {

return "" + *getSizeOfProperty*() + "m^2 " + "$" + *getPrice*() + "/week";

}

}

class *Apartment* extends *Property* {

private *SimpleIntegerProperty* floorNumber;

private *SimpleIntegerProperty* roomNumber;

private *SimpleDoubleProperty* extraFee; *// Strata levies*

protected *Apartment*(*User* *owner*, double *sizeOfProperty*, *Address* *address*,int *numberOfTenants*, int *numberOfBedRooms*, int *numberOfKitchens*, int *numberOfBathRooms*,int *floorNumber*, int *roomNumber*, double *extraFee*, double *price*){

super(*owner*, *sizeOfProperty*, *address*,*numberOfTenants* , *numberOfBedRooms*, *numberOfKitchens*, *numberOfBathRooms*,*price*);

this.floorNumber = new *SimpleIntegerProperty*(*floorNumber*);

this.roomNumber = new *SimpleIntegerProperty*(*roomNumber*);

this.extraFee = new *SimpleDoubleProperty*(*extraFee*);

}

protected *Apartment*(double *sizeOfProperty*, *Address* *address*,int *numberOfTenants*, int *numberOfBedRooms*, int *numberOfKitchens*, int *numberOfBathRooms*, int *floorNumber*, int *roomNumber*, double *extraFee*, double *price*){

super(*sizeOfProperty*, *address*,*numberOfTenants* , *numberOfBedRooms*, *numberOfKitchens*, *numberOfBathRooms*, *price*);

this.floorNumber = new *SimpleIntegerProperty*(*floorNumber*);

this.roomNumber = new *SimpleIntegerProperty*(*roomNumber*);

this.extraFee = new *SimpleDoubleProperty*(*extraFee*);

}

*SimpleIntegerProperty* *getPropertyFloorNumber*(){

return floorNumber;

}

*SimpleIntegerProperty* *getPropertyRoomNumber*(){

return roomNumber;

}

*SimpleDoubleProperty* *getPropertyExtraFee*(){

return extraFee;

}

protected int *getFloorNumber*() {

return *getPropertyFloorNumber*().*get*();

}

protected int *getRoomNumber*(){

return *getPropertyRoomNumber*().*get*();

}

protected double *getExtraFee*(){

return *getPropertyExtraFee*().*get*();

}

protected void *setFloorNumber*(int *floorNumber*){

this.floorNumber = new *SimpleIntegerProperty*(*floorNumber*);

}

protected void *setRoomNumber*(int *roomNumber*){

this.roomNumber = new *SimpleIntegerProperty*(*roomNumber*);

}

protected void *setExtraFee*(double *extraFee*){

this.extraFee = new *SimpleDoubleProperty*(*extraFee*);

}

@Override

*String* *detailInfo*() {

return "Property Type: Apartment (" + *getPropertyID*() + ")\nOwner: " + *getOwner*().*getName*() + " (" + *getOwner*().*getUserID*() + ")" + "\nSize: " + *getSizeOfProperty*() + "m^2" + "\nArea: " + *getAddress*() + "\nTenants: " + *getCountTenants*() + "/" + *getNumberOfTenants*() + " ppl\nNumber Of Bedrooms: " + *getNumberOfBedRooms*() + " rooms\nNumber Of Bathrooms: " + *getNumberOfBathRooms*() + " rooms\nNumber Of Kitchens: " + *getNumberOfKitchens*() + " rooms\nFloor Number: " + *getFloorNumber*() + "\nRoom Number: " + *getRoomNumber*() + "\nExtra Fee: $" + *getExtraFee*() + "\nPrice: $" + *getPrice*() + "/week";

}

@Override

public *String* *toString*() {

return "Apartment " + super.*toString*();

}

}

class *House* extends *Property* {

private *SimpleIntegerProperty* numberOfFloor;

private *SimpleBooleanProperty* garden;

protected *House*(*User* *owner*, double *sizeOfProperty*, *Address* *address*, int *numberOfTenants*, int *numberOfBedRooms*, int *numberOfKitchens*, int *numberOfBathRooms*, int *numberOfFloor*, boolean *garden*, double *price*) {

super(*owner*, *sizeOfProperty*, *address*, *numberOfTenants*, *numberOfBedRooms*, *numberOfKitchens*, *numberOfBathRooms*, *price*);

this.numberOfFloor = new *SimpleIntegerProperty*(*numberOfFloor*);

this.garden = new *SimpleBooleanProperty*(*garden*);

}

protected *House*(double *sizeOfProperty*, *Address* *address*, int *numberOfTenants*, int *numberOfBedRooms*, int *numberOfKitchens*, int *numberOfBathRooms*, int *numberOfFloor*, boolean *garden*, double *price*) {

super(*sizeOfProperty*, *address*, *numberOfTenants*, *numberOfBedRooms*, *numberOfKitchens*, *numberOfBathRooms*, *price*);

this.numberOfFloor = new *SimpleIntegerProperty*(*numberOfFloor*);

this.garden = new *SimpleBooleanProperty*(*garden*);

}

protected *SimpleIntegerProperty* *getPropertyNumberOfFloor*(){

return numberOfFloor;

}

protected *SimpleBooleanProperty* *getPropertyGarden*(){

return garden;

}

protected int *getNumberOfFloor*(){

return *getPropertyNumberOfFloor*().*get*();

}

protected boolean *getGarden*(){

return *getPropertyGarden*().*get*();

}

protected void *setNumberOfFloor*(int *numberOfFloor*){

this.numberOfFloor = new *SimpleIntegerProperty*(*numberOfFloor*);

}

protected void *setGarden*(boolean *garden*){

this.garden = new *SimpleBooleanProperty*(*garden*);

}

@Override

*String* *detailInfo*() {

return "Property Type: House (" + *getPropertyID*() + ")" + "\nOwner: " + *getOwner*().*getName*() + " (" + *getOwner*().*getUserID*() + ")" + "\nSize: " + *getSizeOfProperty*() + "m^2" + "\nArea: " + *getAddress*() + "\nTenants: " + *getCountTenants*() + "/" + *getNumberOfTenants*() + " ppl\nNumber Of Bedrooms: " + *getNumberOfBedRooms*() + " rooms\nNumber Of Bathrooms: " + *getNumberOfBathRooms*() + " rooms\nNumber Of Kitchens: " + *getNumberOfKitchens*() + " rooms\nNumber Of Floor: " + *getNumberOfFloor*() + " floors\nGarden: " + *getGarden*()+ "\nPrice: $" + *getPrice*() + "/week";

}

@Override

public *String* *toString*() {

return "House " + super.*toString*();

}

}

abstract class *User* {

private *SimpleStringProperty* name;

private *SimpleObjectProperty*<*Address*> address;

private *SimpleStringProperty* phoneNumber;

private final *SimpleStringProperty* userID;

private *ObservableList*<*Mail*> mails;

private *SimpleStringProperty* username;

private *SimpleStringProperty* password;

private static *HashMap*<*String*, *User*> usersList = new *HashMap*<>();

*User*(*String* *name*, *Address* *address*, *String* *phoneNumber*, *String* *username*, *String* *password*){

this.name = new *SimpleStringProperty*(*name*);

this.address = new *SimpleObjectProperty*<>(*address*);

this.phoneNumber = new *SimpleStringProperty*(*phoneNumber*);

this.username = new *SimpleStringProperty*(*username*);

this.password = new *SimpleStringProperty*(*password*);

this.userID = new *SimpleStringProperty*(*setUserID*());

this.mails = *FXCollections*.*observableArrayList*();

*User*.usersList.*put*(*getUserID*(), this);

mails.*add*(new *Mail*("Welcome to our project!", "System", this.*getUserID*(), "Hi " + this.name.*get*() + ",\n\nThank you for using our application. We sincerely hope it enhances your \nexperience and provides valuable support in your endeavors.\n\nKind Regards,\nProject Development Team"));

}

protected *SimpleStringProperty* *getPropertyName*() {

return name;

}

protected *SimpleObjectProperty*<*Address*> *getPropertyAddress*() {

return address;

}

protected *SimpleStringProperty* *getPropertyPhoneNumber*() {

return phoneNumber;

}

protected *SimpleStringProperty* *getPropertyUserID*() {

return userID;

}

protected *SimpleStringProperty* *getPropertyUsername*() {

return username;

}

protected *SimpleStringProperty* *getPropertyPassword*() {

return password;

}

protected void *setName*(*String* *name*) {

this.name.*set*(*name*);

}

protected void *setAddress*(*Address* *address*) {

this.address.*set*(*address*);

}

protected void *setPhoneNumber*(*String* *phoneNumber*) {

this.phoneNumber.*set*(*phoneNumber*);

}

protected void *setUsername*(*String* *username*) {

this.username.*set*(*username*);

}

protected void *setPassword*(*String* *password*) {

this.password.*set*(*password*);

}

protected *String* *getName*() {

return *getPropertyName*().*get*();

}

protected *Address* *getAddress*() {

return *getPropertyAddress*().*get*();

}

protected *String* *getPhoneNumber*() {

return *getPropertyPhoneNumber*().*get*();

}

protected *String* *getUserID*() {

return *getPropertyUserID*().*get*();

}

protected *ObservableList*<*Mail*> *getMails*() {

return mails;

}

protected *String* *getUsername*() {

return *getPropertyUsername*().*get*();

}

protected *String* *getPassword*() {

return *getPropertyPassword*().*get*();

}

protected static *HashMap*<*String*, *User*> *getUsersList*() {

return usersList;

}

protected void *removeMail*(int *index*){

mails.*remove*(*index*);

}

abstract *String* *setUserID*();

public *String* *toString*(){

return "\nName: " + *getName*() + "\nAddress: " + *getAddress*() + "\nPhone Number: " + *getPhoneNumber*() + "\nUserID: " + *getUserID*();

}

}

class *Agent* extends *User* {

private *ObservableList*<*Property*> inventory;

protected *Agent*(*String* *name*, *Address* *address*, *String* *phoneNumber*, *String* *username*, *String* *password*) {

super(*name*, *address*, *phoneNumber*, *username*, *password*);

inventory = *FXCollections*.*observableArrayList*();

}

protected *ObservableList*<*Property*> *getInventory*() {

return inventory;

}

protected void *removeProperty*(int *index*) {

*Property*.*getPropertyObservableList*().*remove*(inventory.*get*(*index*));

for (*Client* client : inventory.*get*(*index*).*getTenants*()){

client.*getLeases*().*remove*(inventory.*get*(*index*));

client.*getMails*().*add*(new *Mail*("Cancellation of Your Rental Agreement","System",client.*getUserID*(),"We have received owner request to cancel the\n rental agreement for the property with ID: "+ inventory.*get*(*index*).*getPropertyID*() +"\nYour cancellation has been successfully processed.\n" + "We are sorry to see you go and hope to assist you in the future.\n" + "Kind regards,\n" + "The Customer Service Team"));

}

this.*getMails*().*add*(new *Mail*("Remove your property out of market","System",this.*getUserID*(),"We have received your request to remove the property with ID: "+ inventory.*get*(*index*).*getPropertyID*() +"\nThe request has been successfully processed.\n" + "Kind regards,\n" + "The Customer Service Team"));

inventory.*remove*(*index*);

}

protected void *addPropertyToInventory*(*String* *propertyType*, double *size*, *Address* *location*, int *bedrooms*, int *bathrooms*, int *kitchens*, int *maximumNumberOfTenants*, double *rentPrice*, int *floorNumber*, int *roomNumber*, double *extraFee*, int *numberOfFloors*, boolean *garden*) {

if (*propertyType*.*equals*("House")) {

inventory.*add*(new *House*(this, *size*, *location*, *maximumNumberOfTenants*, *bedrooms*, *kitchens*, *bathrooms*, *numberOfFloors*, *garden*, *rentPrice*));

} else if (*propertyType*.*equals*("Apartment")) {

inventory.*add*(new *Apartment*(this, *size*, *location*, *maximumNumberOfTenants*, *bedrooms*, *kitchens*, *bathrooms*, *floorNumber*, *roomNumber*, *extraFee*, *rentPrice*));

}

}

@Override

protected *String* *setUserID*() {

*String* idsetup;

while (true) {

*Random* rand = new *Random*();

int num1 = rand.*nextInt*(10);

int num2 = rand.*nextInt*(10);

int num3 = rand.*nextInt*(10);

int num4 = rand.*nextInt*(10);

idsetup = "AG" + num1 + num2 + num3 + num4;

boolean isFound = false;

for (*String* id : *User*.*getUsersList*().*keySet*()) {

if (idsetup.*equals*(id)) {

isFound = true;

break;

}

}

if (isFound) {

continue;

}

break;

}

return idsetup;

}

}

class *Client* extends *User* {

private *ObservableList*<*Property*> leases;

protected *Client*(*String* *name*, *Address* *address*, *String* *phoneNumber*, *String* *username*, *String* *password*) {

super(*name*, *address*, *phoneNumber*, *username*, *password*);

leases = *FXCollections*.*observableArrayList*();

}

protected *ObservableList*<*Property*> *getLeases*() {

return leases;

}

protected void *makeContract*(*Property* *property*){

leases.*add*(*property*);

}

@Override

protected *String* *setUserID*() {

*String* idsetup;

while (true) {

*Random* rand = new *Random*();

int num1 = rand.*nextInt*(10);

int num2 = rand.*nextInt*(10);

int num3 = rand.*nextInt*(10);

int num4 = rand.*nextInt*(10);

idsetup = "EC" + num1 + num2 + num3 + num4;

boolean isFound = false;

for (*String* id : *User*.*getUsersList*().*keySet*()){

if (idsetup.*equals*(id)){

isFound = true;

break;

}

}

if (isFound){

continue;

}

break;

}

return idsetup;

}

}

class *Mail* {

private *SimpleStringProperty* tittle;

private *SimpleStringProperty* sender;

private *SimpleStringProperty* recipient;

private *SimpleStringProperty* content;

*Mail*(*String* *tittle*, *String* *sender*, *String* *recieve*, *String* *content*){

this.tittle = new *SimpleStringProperty*(*tittle*);

this.sender = new *SimpleStringProperty*(*sender*);

this.recipient = new *SimpleStringProperty*(*recieve*);

this.content = new *SimpleStringProperty*(*content*);

}

protected *SimpleStringProperty* *getSender*() {

return sender;

}

protected *SimpleStringProperty* *getRecipient*(){

return recipient;

}

protected *SimpleStringProperty* *getContent*(){

return content;

}

protected *SimpleStringProperty* *getTittle*(){

return tittle;

}

}

## AppController

import java.util.*ArrayList*;

import java.util.*Arrays*;

import java.util.*List*;

public class *AppController* {

*AppModel* model;

*AppController*(*AppModel* *model*){

this.model = *model*;

}

*//welcomePage*

*//loginRequest will send the username and password to model to check and return to view if the user input is valid or not*

protected boolean *loginRequest*(*String* *username*, *String* *password*){

return model.*login*(*username*,*password*);

}

*//send user input from view to the model and return some errors that can happens from model*

protected *List*<*String*> *signUpRequest*(*String* *name*, *String* *address*,*String* *phoneNumber*,*String* *username*, *String* *password*, *String* *confirmPassword*, *String* *role*){

if (*phoneNumber*.*length*() > 10){

return new *ArrayList*<>(*Arrays*.*asList*("Invalid Phone Number!!!"));

}

return model.*signUp*(*name*,*address*,*convertStringToInt*(*phoneNumber*),*username*,*password*,*confirmPassword*,*role*);

}

*// require and inform model that user require to log out in order to process steps for log out in model*

protected void *logOutRequest*(){

model.*logOut*();

}

*//accountPage*

*//send user information to model to change and return to view list of errors if something is wrong in user input*

protected *List*<*String*> *editInforRequest*(*String* *name*, *String* *address*,*String* *phoneNumber*,*String* *username*, *String* *password*, *String* *confirmPassword*){

return model.*editCurrentUserInfor*(*name*,*address*,*convertStringToInt*(*phoneNumber*),*username*,*password*,*confirmPassword*);

}

*//mailBoxPage*

*//supply the index of mail the user want to remove to model to execute removing mail*

protected void *removeMail*(int *index*){

model.*getCurrentUser*().*removeMail*(*index*);

}

*//send some information to model to send email and require model to return list of errors if this information is wrong or invalid*

protected *List*<*String*> *sendMailRequest*(*String* *recipient*, *String* *tittle*, *String* *content*){

return model.*sendMail*(*recipient*,*tittle*,*content*);

}

*//marketPage*

*// send options from user to model in order to process sorting properties*

protected void *sortPropertiesRequest*(*String* *options*, boolean *ascending*){

model.*sortProperties*(*options*,*ascending*);

}

*//managementAgent*

*//give index of property and require model to remove this property in user' inventory*

protected void *removeProperty*(int *index*) {((*Agent*) model.*getCurrentUser*()).*removeProperty*(*index*);}

*//send some information to model to add new property and require model to return list of errors if this information is wrong or invalid*

protected *List*<*String*> *addPropertyRequest*(*String* *propertyType*,*String* *size*, *String* *location*,*String* *bedrooms*,*String* *bathrooms*, *String* *kitchens*, *String* *maximumNumberOfTenants*, *String* *rentPrice*, *String* *floorNumber*,*String* *roomNumber*, *String* *extraFee*, *String* *numberOfFloors*,boolean *garden* ){

try {

return model.*addProperty*(*propertyType*,*convertStringToDouble*(*size*), *location*, *convertStringToInt*(*maximumNumberOfTenants*), *convertStringToInt*(*bedrooms*), *convertStringToInt*(*bathrooms*), *convertStringToInt*(*kitchens*), *convertStringToDouble*(*rentPrice*), *convertStringToInt*(*floorNumber*), *convertStringToInt*(*roomNumber*), *convertStringToDouble*(*extraFee*), *convertStringToInt*(*numberOfFloors*),*garden*);

} catch (*Exception* *e*){

return new *ArrayList*<>(*Arrays*.*asList*("Please make sure that you only enter numbers!\n The System does not accept input with operations"));

}

}

*//send some information to model to edit property and require model to return list of errors if this information is wrong or invalid*

protected *List*<*String*> *editPropertyRequest*(*Property* *property*,*String* *size*, *String* *location*,*String* *bedrooms*,*String* *bathrooms*, *String* *kitchens*, *String* *rentPrice*, *String* *floorNumber*,*String* *roomNumber*, *String* *extraFee*, *String* *numberOfFloors*,boolean *garden* ){

try {

return model.*editProperty*(*property*,*convertStringToDouble*(*size*), *location*, *convertStringToInt*(*bedrooms*), *convertStringToInt*(*bathrooms*), *convertStringToInt*(*kitchens*), *convertStringToDouble*(*rentPrice*), *convertStringToInt*(*floorNumber*), *convertStringToInt*(*roomNumber*), *convertStringToDouble*(*extraFee*), *convertStringToInt*(*numberOfFloors*),*garden*);

} catch (*Exception* *e*){

return new *ArrayList*<>(*Arrays*.*asList*("Please make sure that you only enter numbers!\n The System does not accept input with operations"));

}

}

*//managementClient*

*//require model to check user input and return a property to show on view*

protected *Property* *checkPropertyID*(*String* *id*){

return model.*checkPropertyID*(*id*);

}

*//require model to make contract between user to this propertyID if something happens the method will return string to show on view*

protected *String* *makeContractRequest*(*String* *id*){

return model.*makeContract*(*id*);

}

*//require model to cancel contract between user to this property*

protected void *cancelContractRequest*(*Property* *property*){

model.*cancelContract*(*property*);

}

private int *convertStringToInt*(*String* *s*) {

if (*s* == null || *s*.*isEmpty*()) {

return 0;

}

if ("-".*equals*(*s*)) {

return 0;

}

return *Integer*.*parseInt*(*s*); *// Convert string into integer*

}

private double *convertStringToDouble*(*String* *s*) {

if (*s* == null || *s*.*isEmpty*()) {

return 0;

}

if ("-".*equals*(*s*)) {

return 0;

}

return *Double*.*parseDouble*(*s*); *// Convert string into double*

}

}

## AppView

import javafx.scene.control.*TextFormatter*.*Change*;

import javafx.scene.*Scene*;

import javafx.scene.control.\*;

import javafx.stage.*Stage*;

import javafx.geometry.*Pos*;

import javafx.scene.layout.*HBox*;

import javafx.scene.layout.*VBox*;

import javafx.stage.*Modality*;

import java.util.*ArrayList*;

import java.util.*List*;

public class *AppView* {

*AppController* controller;

*AppModel* model;

*Stage* primaryStage;

*AppView*(*AppController* *controller*, *AppModel* *model*, *Stage* *primaryStage*){

this.controller = *controller*;

this.model = *model*;

this.primaryStage = *primaryStage*;

}

*Scene* *welcomePage*(){

*Label* welcomeLabel = new *Label*("Welcome to Management Housing Project");

*Label* usernameLabel = new *Label*("Username");

*Label* passwordLabel = new *Label*("Password");

*TextField* usernameTextField = new *TextField*();

usernameTextField.*setPromptText*("Enter your username");

*TextField* passwordTextField = new *TextField*();

passwordTextField.*setPromptText*("Enter your password");

*Button* signUpBtn = new *Button*("Create a new account");

signUpBtn.*setOnAction*(*event* -> *createSignUpWindow*());

*Button* submitBtn = new *Button*("Login");

submitBtn.*setOnAction*(*event* -> {

*// catch user input*

*String* username = usernameTextField.*getText*().*trim*();

*String* password = passwordTextField.*getText*().*trim*();

if (!username.*isEmpty*() && !password.*isEmpty*()){

*// call controller to process the login request*

if (controller.*loginRequest*(username,password)){

primaryStage.*setScene*(*homePageUser*());

} else {

*createNotificationWindow*("The username or password is incorrect!!!");

}

} else {

*createNotificationWindow*("the username or password can not be empty!!!");

}

});

*HBox* welcomeRow = new *HBox*(welcomeLabel);

welcomeRow.*setAlignment*(*Pos*.TOP\_CENTER);

*HBox* usernameLoginRow = new *HBox*(6, usernameLabel, usernameTextField);

usernameLoginRow.*setAlignment*(*Pos*.CENTER);

*HBox* passwordLoginRow = new *HBox*(6, passwordLabel, passwordTextField);

passwordLoginRow.*setAlignment*(*Pos*.CENTER);

*VBox* buttonsRow = new *VBox*(10,signUpBtn);

buttonsRow.*setAlignment*(*Pos*.CENTER);

*VBox* userLoginRow = new *VBox*(5, usernameLoginRow,passwordLoginRow,submitBtn);

userLoginRow.*setAlignment*(*Pos*.CENTER);

*VBox* root = new *VBox*(50,welcomeRow,userLoginRow,buttonsRow);

root.*setAlignment*(*Pos*.CENTER);

return new *Scene*(root,600,600);

}

*Scene* *homePageUser*() {

*Label* userProfile = new *Label*(model.*getCurrentUser*().*toString*());

*Label* homePageLabel = new *Label*("HOME PAGE");

*Button* accountBtn = new *Button*("Account");

accountBtn.*setOnAction*(*event* -> primaryStage.*setScene*(*accountUserPage*()));

*Button* mailBoxBtn = new *Button*("MailBox" + "(" + model.*getCurrentUser*().*getMails*().*size*() + ")");

mailBoxBtn.*setOnAction*(*event* -> primaryStage.*setScene*(*mailBoxPage*()));

*Button* marketBtn = new *Button*("Market");

marketBtn.*setOnAction*(*event* -> primaryStage.*setScene*(*marketPage*()));

*Button* managementBtn = new *Button*("Management");

managementBtn.*setOnAction*(*event* -> {

if (model.*getCurrentUser*() instanceof *Agent*) {

primaryStage.*setScene*(*managementAgentPage*());

} else if (model.*getCurrentUser*() instanceof *Client*) {

primaryStage.*setScene*(*managementClientPage*());

}

});

*Button* logOutBtn = new *Button*("Log out");

*// process log out request and call controller to inform model that current user is logging out*

logOutBtn.*setOnAction*(*event* -> {

primaryStage.*setScene*(*welcomePage*());

controller.*logOutRequest*();

});

*VBox* buttonsCol = new *VBox*(5, accountBtn,mailBoxBtn,managementBtn,marketBtn);

*VBox* userCol = new *VBox*(5,userProfile,logOutBtn);

*HBox* mainRow = new *HBox*(20, userCol,buttonsCol);

mainRow.*setAlignment*(*Pos*.CENTER);

*VBox* root = new *VBox*(5, homePageLabel, mainRow);

root.*setAlignment*(*Pos*.CENTER);

return new *Scene*(root,600,600);

}

*Scene* *accountUserPage*() {

*Label* accountPageLabel = new *Label*("ACCOUNT PAGE");

*Label* name = new *Label*();

name.*textProperty*().*bind*(model.*getCurrentUser*().*getPropertyName*());

*Label* nameLabel = new *Label*("Name: ");

*Label* address = new *Label*();

address.*textProperty*().*bind*(model.*getCurrentUser*().*getPropertyAddress*().*asString*());

*Label* addressslabel = new *Label*("Address: ");

*Label* phoneNumber = new *Label*();

phoneNumber.*textProperty*().*bind*(model.*getCurrentUser*().*getPropertyPhoneNumber*());

*Label* phoneNumberLabel = new *Label*("Phone Number: ");

*Label* username = new *Label*();

username.*textProperty*().*bind*(model.*getCurrentUser*().*getPropertyUsername*());

*Label* usernameLabel = new *Label*("Username: ");

*Label* password = new *Label*();

password.*setText*("\*\*\*\*\*\*\*\*");

*Label* passwordLabel = new *Label*("Password: ");

*Button* visiblePassBtn = new *Button*("Show");

visiblePassBtn.*setOnAction*(*event* -> {

if (visiblePassBtn.*getText*().*equals*("Show")){

visiblePassBtn.*setText*("Hide");

password.*setText*(model.*getCurrentUser*().*getPassword*());

} else {

visiblePassBtn.*setText*("Show");

password.*setText*("\*\*\*\*\*\*\*\*");

}

});

*Button* editBtn = new *Button*("Edit Profile");

editBtn.*setOnAction*(*event* -> *createEditUserWindow*());

*Button* backBtn = new *Button*("Back");

backBtn.*setOnAction*(*event* -> primaryStage.*setScene*(*homePageUser*()));

*HBox* nameRow = new *HBox*(20, nameLabel,name);

nameRow.*setAlignment*(*Pos*.CENTER);

*HBox* addressRow = new *HBox*(20, addressslabel,address);

addressRow.*setAlignment*(*Pos*.CENTER);

*HBox* phoneNumberRow = new *HBox*(20, phoneNumberLabel,phoneNumber);

phoneNumberRow.*setAlignment*(*Pos*.CENTER);

*HBox* usernameRow = new *HBox*(20, usernameLabel,username);

usernameRow.*setAlignment*(*Pos*.CENTER);

*HBox* passwordRow = new *HBox*(20, passwordLabel,password, visiblePassBtn);

passwordRow.*setAlignment*(*Pos*.CENTER);

*HBox* buttonsRow = new *HBox*(40, editBtn,backBtn);

buttonsRow.*setAlignment*(*Pos*.CENTER);

*VBox* root = new *VBox*(10,accountPageLabel,nameRow,addressRow,phoneNumberRow,usernameRow,passwordRow,buttonsRow);

root.*setAlignment*(*Pos*.CENTER);

return new *Scene*(root,600,600);

}

*Scene* *mailBoxPage*(){

*Label* mailboxLabel = new *Label*("MAIL BOX");

*TableView*<*Mail*> mailTable = new *TableView*<>();

*TableColumn*<*Mail*, *String*> senderCol = new *TableColumn*<>("Sender");

senderCol.*setCellValueFactory*(*cellData* -> *cellData*.*getValue*().*getSender*());

senderCol.*setMinWidth*(200);

*TableColumn*<*Mail*,*String*> tittleCol = new *TableColumn*<>("Tittle");

tittleCol.*setCellValueFactory*(*cellData* -> *cellData*.*getValue*().*getTittle*());

tittleCol.*setMinWidth*(400);

mailTable.*setItems*(model.*getCurrentUser*().*getMails*());

mailTable.*getColumns*().*addAll*(tittleCol,senderCol);

*Button* viewBtn = new *Button*("View Detail");

viewBtn.*setOnAction*(*event* -> {

int index = mailTable.*getSelectionModel*().*getSelectedIndex*();

if (index != -1 ){

*createViewMailWindow*(model.*getCurrentUser*().*getMails*().*get*(index));

} else {

*createNotificationWindow*("Please choose the mail that you want to view detail!");

}

});

*Button* deleteBtn = new *Button*("Delete mail");

deleteBtn.*setOnAction*(*event* -> {

int index = mailTable.*getSelectionModel*().*getSelectedIndex*();

if (index != - 1 ){

*// call controller to process removing this mail*

controller.*removeMail*(index);

} else {

*createNotificationWindow*("Please choose the mail that you want to remove!");

}

});

*Button* sendMailBtn = new *Button*("Send mail");

sendMailBtn.*setOnAction*(*event* -> *createSendEmailWindow*());

*Button* backBtn = new *Button*("Back");

backBtn.*setOnAction*(*event* -> primaryStage.*setScene*(*homePageUser*()));

*HBox* buttonsRow = new *HBox*(20, viewBtn,deleteBtn,sendMailBtn);

buttonsRow.*setAlignment*(*Pos*.CENTER);

*VBox* mainRow = new *VBox*(5, mailTable, buttonsRow);

mainRow.*setAlignment*(*Pos*.CENTER);

*VBox* root = new *VBox*(30, mailboxLabel,mainRow,backBtn);

root.*setAlignment*(*Pos*.CENTER);

return new *Scene*(root,600, 600);

}

*Scene* *marketPage*(){

*Label* marketLabel = new *Label*("MARKET");

*ListView*<*Property*> propertyListView = new *ListView*<>();

propertyListView.*setItems*(*Property*.*getPropertyObservableList*());

*Button* viewBtn = new *Button*("View Detail");

viewBtn.*setOnAction*(*event* -> {

int index = propertyListView.*getSelectionModel*().*getSelectedIndex*();

if ( index != -1 ){

*createViewPropertyWindow*(*Property*.*getPropertyObservableList*().*get*(index));

} else {

*createNotificationWindow*("Please choose the property that you want to view all of details");

}

});

*Button* sortBtn = new *Button*("Sort properties");

sortBtn.*setOnAction*(*event* -> *createSortOptionsWindow*());

*Button* backBtn = new *Button*("Back");

backBtn.*setOnAction*(*event* -> primaryStage.*setScene*(*homePageUser*()));

*VBox* marketCol = new *VBox*(5,marketLabel,propertyListView);

marketCol.*setAlignment*(*Pos*.CENTER);

*HBox* functionRow = new *HBox*(30, viewBtn, sortBtn);

functionRow.*setAlignment*(*Pos*.CENTER);

*VBox* root = new *VBox*(25,marketCol, functionRow, backBtn);

root.*setAlignment*(*Pos*.CENTER);

return new *Scene*(root,500,300);

}

*Scene* *managementAgentPage*(){

*Agent* agent = (*Agent*) model.*getCurrentUser*();

*Label* inventoryLabel = new *Label*("Your Inventory");

*TableView*<*Property*> inventoryTable = new *TableView*<>();

inventoryTable.*setItems*(agent.*getInventory*());

*TableColumn*<*Property*, *String*> propertyIDCol = new *TableColumn*<>("Property ID");

propertyIDCol.*setCellValueFactory*(*cellData* -> *cellData*.*getValue*().*getPropertyPropertyID*());

propertyIDCol.*setMinWidth*(250);

*TableColumn*<*Property*, *String*> numberOfTenantsCol = new *TableColumn*<>("Number Of Tenants");

numberOfTenantsCol.*setCellValueFactory*(*cellData* -> *cellData*.*getValue*().*getPropertyCountTenants*().*asString*());

numberOfTenantsCol.*setMinWidth*(150);

*TableColumn*<*Property*,*String*> rentPriceCol = new *TableColumn*<>("Rent Price (Per Week)");

rentPriceCol.*setCellValueFactory*(*cellData* -> *cellData*.*getValue*().*getPropertyPrice*().*asString*());

rentPriceCol.*setMinWidth*(200);

inventoryTable.*getColumns*().*addAll*(propertyIDCol,numberOfTenantsCol,rentPriceCol);

*Button* viewDetailBtn = new *Button*("View Detail Property");

viewDetailBtn.*setOnAction*(*event* -> {

int index = inventoryTable.*getSelectionModel*().*getSelectedIndex*();

if (index != -1){

*createViewPropertyWindow*(agent.*getInventory*().*get*(index));

} else {

*createNotificationWindow*("Please choose one of property to view detail");

}

});

*Button* deletePropertyBtn = new *Button*("Delete Property");

deletePropertyBtn.*setOnAction*(*event* -> {

int index = inventoryTable.*getSelectionModel*().*getSelectedIndex*();

if (index != -1){

*// call controller to process removing this property*

controller.*removeProperty*(index);

} else {

*createNotificationWindow*("Please choose one of property to delete");

}

});

*Button* editPropertyInfoBtn = new *Button*("Edit Property");

editPropertyInfoBtn.*setOnAction*(*event* -> {

int index = inventoryTable.*getSelectionModel*().*getSelectedIndex*();

if (index != -1){

*createEditPropertyWindow*(agent.*getInventory*().*get*(inventoryTable.*getSelectionModel*().*getSelectedIndex*()));

} else {

*createNotificationWindow*("Please choose one of property to edit");

}

});

*Button* addPropertyBtn = new *Button*("Add new Property");

addPropertyBtn.*setOnAction*(*event* -> *createAddNewPropertyWindow*());

*Button* backBtn = new *Button*("Back");

backBtn.*setOnAction*(*event* -> primaryStage.*setScene*(*homePageUser*()));

*VBox* inventoryArea = new *VBox*(inventoryLabel,inventoryTable);

inventoryArea.*setAlignment*(*Pos*.CENTER);

*HBox* functionRow = new *HBox*(10,addPropertyBtn,editPropertyInfoBtn,deletePropertyBtn);

functionRow.*setAlignment*(*Pos*.CENTER);

*VBox* functionArea = new *VBox*(5,functionRow,viewDetailBtn);

functionArea.*setAlignment*(*Pos*.CENTER);

*VBox* root = new *VBox*(20,inventoryArea,functionArea,backBtn);

root.*setAlignment*(*Pos*.CENTER);

return new *Scene*(root,600,400);

}

*Scene* *managementClientPage*(){

*Client* client = (*Client*) model.*getCurrentUser*();

*Label* leaseLabel = new *Label*("Your leaases");

*TableView*<*Property*> leasesTable = new *TableView*<>();

leasesTable.*setItems*(client.*getLeases*());

*TableColumn*<*Property*, *String*> propertyIDCol = new *TableColumn*<>("Property ID");

propertyIDCol.*setCellValueFactory*(*cellData* -> *cellData*.*getValue*().*getPropertyPropertyID*());

propertyIDCol.*setMinWidth*(250);

*TableColumn*<*Property*, *String*> ownerIDCol = new *TableColumn*<>("OwnerID");

ownerIDCol.*setCellValueFactory*(*cellData* -> *cellData*.*getValue*().*getOwner*().*getPropertyUserID*());

ownerIDCol.*setMinWidth*(150);

*TableColumn*<*Property*,*String*> rentPriceCol = new *TableColumn*<>("Rent Price (Per Week)");

rentPriceCol.*setCellValueFactory*(*cellData* -> *cellData*.*getValue*().*getPropertyPrice*().*asString*());

rentPriceCol.*setMinWidth*(200);

leasesTable.*getColumns*().*addAll*(propertyIDCol,ownerIDCol,rentPriceCol);

*Button* makeContractBtn = new *Button*("Make Contract");

makeContractBtn.*setOnAction*(*event* -> *createMakeContractWindow*());

*Button* cancelContractBtn = new *Button*("Cancel Contract");

cancelContractBtn.*setOnAction*(*event* -> {

int index = leasesTable.*getSelectionModel*().*getSelectedIndex*();

if (index != -1){

*// call controller for cancel request*

controller.*cancelContractRequest*(client.*getLeases*().*get*(index));

*createNotificationWindow*("Cancelled Rental Agreement of this property!");

} else {

*createNotificationWindow*("Please choose one of property to cancel ");

}

});

*Button* viewDetailBtn = new *Button*("View Detail of Property");

viewDetailBtn.*setOnAction*(*event* -> {

int index = leasesTable.*getSelectionModel*().*getSelectedIndex*();

if (index != -1){

*createViewPropertyWindow*(client.*getLeases*().*get*(index));

} else {

*createNotificationWindow*("Please choose one of property to view detail");

}

});

*Button* backBtn = new *Button*("Back");

backBtn.*setOnAction*(*event* -> primaryStage.*setScene*(*homePageUser*()));

*VBox* tableArea = new *VBox*(5,leaseLabel,leasesTable);

tableArea.*setAlignment*(*Pos*.CENTER);

*HBox* functionRow = new *HBox*(20,makeContractBtn,cancelContractBtn,viewDetailBtn);

functionRow.*setAlignment*(*Pos*.CENTER);

*VBox* root = new *VBox*(20,tableArea,functionRow,backBtn);

root.*setAlignment*(*Pos*.CENTER);

return new *Scene*(root,600,400);

}

protected void *createMakeContractWindow*(){

*Stage* stage = new *Stage*();

stage.*setTitle*("Rent Contract");

stage.*initOwner*(primaryStage);

stage.*initModality*(*Modality*.APPLICATION\_MODAL);

*Label* propertyIDLabel = new *Label*("PropertyID: ");

*Label* propertyInforLabel = new *Label*("Property' Detail");

*Label* propertyInFor = new *Label*();

*TextField* propertyIDTextField = new *TextField*();

propertyIDTextField.*textProperty*().*addListener*(*newText* -> {

*Property* property = controller.*checkPropertyID*(propertyIDTextField.*getText*().*trim*());

if (property != null){

propertyInFor.*setText*(property.*detailInfo*());

} else {

propertyInFor.*setText*("Please Enter a PropertyID to view details");

}

});

*Button* signBtn = new *Button*("Sign");

signBtn.*setOnAction*(*event* -> {

*String* propertyID = propertyIDTextField.*getText*().*trim*();

*String* error = controller.*makeContractRequest*(propertyID);

if (error.*isEmpty*()){

*createNotificationWindow*("Congratulations!!! You have just signed a new rental contract!");

stage.*close*();

} else {

*createNotificationWindow*(error);

}

});

*Button* closeBtn = new *Button*("Close");

closeBtn.*setOnAction*(*event* -> stage.*close*());

*HBox* propertyIDRow = new *HBox*(15, propertyIDLabel,propertyIDTextField);

propertyIDRow.*setAlignment*(*Pos*.CENTER);

*VBox* propertyInforArea = new *VBox*(5,propertyInforLabel,propertyInFor);

propertyInforArea.*setAlignment*(*Pos*.CENTER);

*HBox* buttonsRow = new *HBox*(15,signBtn,closeBtn);

buttonsRow.*setAlignment*(*Pos*.CENTER);

*VBox* root = new *VBox*(30,propertyIDRow,propertyInforArea,buttonsRow);

root.*setAlignment*(*Pos*.CENTER);

*Scene* scene = new *Scene*(root,500,450);

stage.*setScene*(scene);

stage.*show*();

}

protected void *createEditPropertyWindow*(*Property* *property*){

*Stage* stage = new *Stage*();

stage.*setTitle*("Edit Property Information");

stage.*initOwner*(primaryStage);

stage.*initModality*(*Modality*.APPLICATION\_MODAL);

*Label* sizeOfPropertyLabel= new *Label*("Size of Property: ");

*Label* locationLabel = new *Label*("Location: ");

*Label* numberOfBedroomsLabel = new *Label*("Number Of Bedrooms: ");

*Label* numberOfBathroomsLabel = new *Label*("Number Of Bathrooms: ");

*Label* numberOfKitchensLabel = new *Label*("Number Of Kitchens: ");

*Label* priceLabel = new *Label*("Price: ");

*Label* optionsForApartmentLabel = new *Label*("Options for Apartment");

*Label* floorNumberLabel = new *Label*("Floor Number: ");

*Label* roomNumebrLabel = new *Label*("Room Number: ");

*Label* estraFeeLabel = new *Label*("Satra Fee: ");

*Label* optionsForHouseLabel = new *Label*("Options For House");

*Label* numberOfFloorLabel = new *Label*("Number Of Floors: ");

*Label* gardenLabel = new *Label*("Garden: ");

*TextField* sizeOfPropertyTextField = new *TextField*();

*configTextFieldForDoubles*(sizeOfPropertyTextField);

sizeOfPropertyTextField.*setPromptText*("Enter size of property");

*TextField* locationTextField = new *TextField*();

locationTextField.*setPromptText*("Enter with lowercase (EX: central,...)");

locationTextField.*setMinWidth*(250);

*TextField* numberOfBedroomsTextField = new *TextField*();

*configTextFieldForInts*(numberOfBedroomsTextField);

numberOfBedroomsTextField.*setPromptText*("Enter number of bedrooms");

numberOfBedroomsTextField.*setMinWidth*(250);

*TextField* numberOfBathroomsTextField = new *TextField*();

*configTextFieldForInts*(numberOfBathroomsTextField);

numberOfBathroomsTextField.*setMinWidth*(250);

numberOfBathroomsTextField.*setPromptText*("Enter number of bathrooms");

*TextField* numberOfKitchensTextField = new *TextField*();

numberOfKitchensTextField.*setMinWidth*(250);

*configTextFieldForInts*(numberOfKitchensTextField);

numberOfKitchensTextField.*setPromptText*("Enter number of kitchens");

*TextField* priceTextField = new *TextField*();

*configTextFieldForDoubles*(priceTextField);

priceTextField.*setPromptText*("Enter rent price");

*TextField* floorNumberTextField = new *TextField*();

floorNumberTextField.*setMinWidth*(250);

*configTextFieldForInts*(floorNumberTextField);

floorNumberTextField.*setPromptText*("Enter floor number");

*TextField* roomNumberTextField = new *TextField*();

*configTextFieldForInts*(roomNumberTextField);

roomNumberTextField.*setPromptText*("Enter room number");

*TextField* extraFeeTextField = new *TextField*();

*configTextFieldForDoubles*(extraFeeTextField);

extraFeeTextField.*setPromptText*("Enter extra fee");

*TextField* numberOfFloorTextField = new *TextField*();

*configTextFieldForInts*(numberOfFloorTextField);

numberOfFloorTextField.*setPromptText*("Enter number of floors");

*ToggleGroup* gardenToggleGroup = new *ToggleGroup*();

*RadioButton* yesGardenBtn = new *RadioButton*("Yes");

yesGardenBtn.*setToggleGroup*(gardenToggleGroup);

*RadioButton* noGardenBtn = new *RadioButton*("No");

noGardenBtn.*setToggleGroup*(gardenToggleGroup);

*Button* submitBtn = new *Button*("Submit");

submitBtn.*setOnAction*(*event* -> {

*String* size = sizeOfPropertyTextField.*getText*().*trim*();

*String* location = locationTextField.*getText*().*trim*();

*String* numberOfBedrooms = numberOfBedroomsTextField.*getText*().*trim*();

*String* numberOfBathrooms = numberOfBathroomsTextField.*getText*().*trim*();

*String* numberOfKitchens = numberOfKitchensTextField.*getText*().*trim*();

*String* rentPrice = priceTextField.*getText*().*trim*();

*String* floorNumber = floorNumberTextField.*getText*().*trim*();

*String* roomNumber = roomNumberTextField.*getText*().*trim*();

*String* extraFee = extraFeeTextField.*getText*().*trim*();

*String* numberOfFloors = numberOfFloorTextField.*getText*().*trim*();

*List*<*String*> errors = new *ArrayList*<>();

if (size.*isEmpty*() || location.*isEmpty*() || numberOfBedrooms.*isEmpty*() || numberOfBathrooms.*isEmpty*() ||

numberOfKitchens.*isEmpty*() || rentPrice.*isEmpty*()) {

*createNotificationWindow*("Please fill in all required fields!");

} else {

if (*property* instanceof *Apartment*) {

if (floorNumber.*isEmpty*() || roomNumber.*isEmpty*() || extraFee.*isEmpty*()) {

*createNotificationWindow*("Please fill in all apartment-specific fields!");

return;

}

errors = controller.*editPropertyRequest*( *property*, size, location, numberOfBedrooms, numberOfBathrooms, numberOfKitchens, rentPrice, floorNumber, roomNumber, extraFee, numberOfFloors, true);

} else if (*property* instanceof *House*) {

if (numberOfFloors.*isEmpty*() || !yesGardenBtn.*isSelected*() && !noGardenBtn.*isSelected*()) {

*createNotificationWindow*("Please fill in all house-specific fields!");

return;

}

errors = controller.*editPropertyRequest*(*property*, size, location, numberOfBedrooms, numberOfBathrooms, numberOfKitchens, rentPrice, floorNumber, roomNumber, extraFee, numberOfFloors, yesGardenBtn.*isSelected*());

}

if (errors.*isEmpty*()){

*createNotificationWindow*("Edited Property");

stage.*close*();

} else {

*String* errorString = "";

for (*String* error : errors){

errorString += "\n" + error;

}

*createNotificationWindow*(errorString);

}

}

});

*Button* closeBtn = new *Button*("Close");

closeBtn.*setOnAction*(*event* -> stage.*close*());

*HBox* sizeOfProperty = new *HBox*(5,sizeOfPropertyLabel,sizeOfPropertyTextField);

*HBox* location = new *HBox*(5,locationLabel,locationTextField);

*HBox* numberOfBedrooms = new *HBox*(5,numberOfBedroomsLabel,numberOfBedroomsTextField);

*HBox* numberOfBathrooms = new *HBox*(5,numberOfBathroomsLabel,numberOfBathroomsTextField);

*HBox* numberOfKitchens = new *HBox*(5,numberOfKitchensLabel,numberOfKitchensTextField);

*HBox* price = new *HBox*(5,priceLabel,priceTextField);

*HBox* floorNumber = new *HBox*(5,floorNumberLabel,floorNumberTextField);

*HBox* roomNumber = new *HBox*(5,roomNumebrLabel,roomNumberTextField);

*HBox* extraFee = new *HBox*(5,estraFeeLabel,extraFeeTextField);

*HBox* numberOfFloors = new *HBox*(5,numberOfFloorLabel,numberOfFloorTextField);

*HBox* garden = new *HBox*(10, gardenLabel,yesGardenBtn,noGardenBtn);

*HBox* buttonsRow = new *HBox*(20, submitBtn,closeBtn);

buttonsRow.*setAlignment*(*Pos*.CENTER);

*VBox* houseOptionsArea = new *VBox*(10, optionsForHouseLabel,numberOfFloors,garden);

houseOptionsArea.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* apartmentOptionsArea = new *VBox*(10,optionsForApartmentLabel,floorNumber,roomNumber,extraFee);

apartmentOptionsArea.*setAlignment*(*Pos*.TOP\_LEFT);

*VBox* generalArea = new *VBox*(10,sizeOfProperty,location,numberOfBedrooms,numberOfBathrooms,numberOfKitchens,price);

generalArea.*setAlignment*(*Pos*.CENTER\_RIGHT);

*HBox* optionsArea = new *HBox*();

if (*property* instanceof *House*){

optionsArea.*getChildren*().*add*(houseOptionsArea);

} else {

optionsArea.*getChildren*().*add*(apartmentOptionsArea);

}

optionsArea.*setAlignment*(*Pos*.CENTER);

*HBox* inforArea = new *HBox*(10,generalArea,optionsArea);

*VBox* root = new *VBox*(30, inforArea,buttonsRow);

root.*setAlignment*(*Pos*.CENTER);

*Scene* scene = new *Scene*(root,700,500);

stage.*setScene*(scene);

stage.*show*();

}

protected void *createAddNewPropertyWindow*(){

*Stage* stage = new *Stage*();

stage.*setTitle*("Add New Property");

stage.*initOwner*(primaryStage);

stage.*initModality*(*Modality*.APPLICATION\_MODAL);

*Label* propertyTypeLabel = new *Label*("Property Type: ");

*Label* sizeOfPropertyLabel= new *Label*("Size of Property: ");

*Label* locationLabel = new *Label*("Location: ");

*Label* numberOfBedroomsLabel = new *Label*("Number Of Bedrooms: ");

*Label* numberOfBathroomsLabel = new *Label*("Number Of Bathrooms: ");

*Label* numberOfKitchensLabel = new *Label*("Number Of Kitchens: ");

*Label* maximumOfTenantsLabel = new *Label*("Maximum Number Of Tenants: ");

*Label* priceLabel = new *Label*("Price: ");

*Label* optionsForApartmentLabel = new *Label*("Options for Apartment");

*Label* floorNumberLabel = new *Label*("Floor Number: ");

*Label* roomNumebrLabel = new *Label*("Room Number: ");

*Label* estraFeeLabel = new *Label*("Satra Fee: ");

*Label* optionsForHouseLabel = new *Label*("Options For House");

*Label* numberOfFloorLabel = new *Label*("Number Of Floors: ");

*Label* gardenLabel = new *Label*("Garden: ");

*TextField* sizeOfPropertyTextField = new *TextField*();

*configTextFieldForDoubles*(sizeOfPropertyTextField);

sizeOfPropertyTextField.*setPromptText*("Enter size of property");

*TextField* locationTextField = new *TextField*();

locationTextField.*setPromptText*("Enter with lowercase (EX: central,...)");

locationTextField.*setMinWidth*(250);

*TextField* numberOfBedroomsTextField = new *TextField*();

*configTextFieldForInts*(numberOfBedroomsTextField);

numberOfBedroomsTextField.*setPromptText*("Enter number of bedrooms");

numberOfBedroomsTextField.*setMinWidth*(250);

*TextField* numberOfBathroomsTextField = new *TextField*();

*configTextFieldForInts*(numberOfBathroomsTextField);

numberOfBathroomsTextField.*setMinWidth*(250);

numberOfBathroomsTextField.*setPromptText*("Enter number of bathrooms");

*TextField* numberOfKitchensTextField = new *TextField*();

numberOfKitchensTextField.*setMinWidth*(250);

*configTextFieldForInts*(numberOfKitchensTextField);

numberOfKitchensTextField.*setPromptText*("Enter number of kitchens");

*TextField* maximumOfTenantsTextField = new *TextField*();

maximumOfTenantsTextField.*setMinWidth*(250);

*configTextFieldForInts*(maximumOfTenantsTextField);

maximumOfTenantsTextField.*setPromptText*("Enter maximum number of tenants");

*TextField* priceTextField = new *TextField*();

*configTextFieldForDoubles*(priceTextField);

priceTextField.*setPromptText*("Enter rent price");

*TextField* floorNumberTextField = new *TextField*();

floorNumberTextField.*setMinWidth*(250);

*configTextFieldForInts*(floorNumberTextField);

floorNumberTextField.*setPromptText*("Enter floor number");

*TextField* roomNumberTextField = new *TextField*();

*configTextFieldForInts*(roomNumberTextField);

roomNumberTextField.*setPromptText*("Enter room number");

*TextField* extraFeeTextField = new *TextField*();

*configTextFieldForDoubles*(extraFeeTextField);

extraFeeTextField.*setPromptText*("Enter extra fee");

*TextField* numberOfFloorTextField = new *TextField*();

*configTextFieldForInts*(numberOfFloorTextField);

numberOfFloorTextField.*setPromptText*("Enter number of floors");

*ToggleGroup* propertyTypeToggleGroup = new *ToggleGroup*();

*ToggleGroup* gardenToggleGroup = new *ToggleGroup*();

*RadioButton* houseTypeBtn = new *RadioButton*("House");

houseTypeBtn.*setToggleGroup*(propertyTypeToggleGroup);

*RadioButton* apartmentTypeBtn = new *RadioButton*("Apartment");

apartmentTypeBtn.*setToggleGroup*(propertyTypeToggleGroup);

*RadioButton* yesGardenBtn = new *RadioButton*("Yes");

yesGardenBtn.*setToggleGroup*(gardenToggleGroup);

*RadioButton* noGardenBtn = new *RadioButton*("No");

noGardenBtn.*setToggleGroup*(gardenToggleGroup);

*Button* submitBtn = new *Button*("Submit");

submitBtn.*setOnAction*(*event* -> {

if (!houseTypeBtn.*isSelected*() && !apartmentTypeBtn.*isSelected*()) {

*createNotificationWindow*("Please select type of property!");

return;

}

*String* size = sizeOfPropertyTextField.*getText*().*trim*();

*String* location = locationTextField.*getText*().*trim*();

*String* numberOfBedrooms = numberOfBedroomsTextField.*getText*().*trim*();

*String* numberOfBathrooms = numberOfBathroomsTextField.*getText*().*trim*();

*String* numberOfKitchens = numberOfKitchensTextField.*getText*().*trim*();

*String* maximumTenants = maximumOfTenantsTextField.*getText*().*trim*();

*String* rentPrice = priceTextField.*getText*().*trim*();

*String* floorNumber = floorNumberTextField.*getText*().*trim*();

*String* roomNumber = roomNumberTextField.*getText*().*trim*();

*String* extraFee = extraFeeTextField.*getText*().*trim*();

*String* numberOfFloors = numberOfFloorTextField.*getText*().*trim*();

*List*<*String*> errors = new *ArrayList*<>();

if (size.*isEmpty*() || location.*isEmpty*() || numberOfBedrooms.*isEmpty*() || numberOfBathrooms.*isEmpty*() ||

numberOfKitchens.*isEmpty*() || maximumTenants.*isEmpty*() || rentPrice.*isEmpty*()) {

*createNotificationWindow*("Please fill in all required fields!");

} else {

if (apartmentTypeBtn.*isSelected*()) {

*String* propertyType = "Apartment";

if (floorNumber.*isEmpty*() || roomNumber.*isEmpty*() || extraFee.*isEmpty*()) {

*createNotificationWindow*("Please fill in all apartment-specific fields!");

return;

}

errors = controller.*addPropertyRequest*(propertyType, size, location, numberOfBedrooms, numberOfBathrooms, numberOfKitchens, maximumTenants, rentPrice, floorNumber, roomNumber, extraFee, numberOfFloors, true);

} else if (houseTypeBtn.*isSelected*()) {

*String* propertyType = "House";

if (numberOfFloors.*isEmpty*() || !yesGardenBtn.*isSelected*() && !noGardenBtn.*isSelected*()) {

*createNotificationWindow*("Please fill in all house-specific fields!");

return;

}

errors = controller.*addPropertyRequest*(propertyType, size, location, numberOfBedrooms, numberOfBathrooms, numberOfKitchens, maximumTenants, rentPrice, floorNumber, roomNumber, extraFee, numberOfFloors, yesGardenBtn.*isSelected*());

}

if (errors.*isEmpty*()){

*createNotificationWindow*("Added Property");

stage.*close*();

} else {

*String* errorString = "";

for (*String* error : errors){

errorString += "\n" + error;

}

*createNotificationWindow*(errorString);

}

}

});

*Button* closeBtn = new *Button*("Close");

closeBtn.*setOnAction*(*event* -> stage.*close*());

*HBox* propertyType = new *HBox*(2,propertyTypeLabel,houseTypeBtn,apartmentTypeBtn);

*HBox* sizeOfProperty = new *HBox*(5,sizeOfPropertyLabel,sizeOfPropertyTextField);

*HBox* location = new *HBox*(5,locationLabel,locationTextField);

*HBox* numberOfBedrooms = new *HBox*(5,numberOfBedroomsLabel,numberOfBedroomsTextField);

*HBox* numberOfBathrooms = new *HBox*(5,numberOfBathroomsLabel,numberOfBathroomsTextField);

*HBox* numberOfKitchens = new *HBox*(5,numberOfKitchensLabel,numberOfKitchensTextField);

*HBox* maximumNumberOfTenants = new *HBox*(5,maximumOfTenantsLabel,maximumOfTenantsTextField);

*HBox* price = new *HBox*(5,priceLabel,priceTextField);

*HBox* floorNumber = new *HBox*(5,floorNumberLabel,floorNumberTextField);

*HBox* roomNumber = new *HBox*(5,roomNumebrLabel,roomNumberTextField);

*HBox* extraFee = new *HBox*(5,estraFeeLabel,extraFeeTextField);

*HBox* numberOfFloors = new *HBox*(5,numberOfFloorLabel,numberOfFloorTextField);

*HBox* garden = new *HBox*(10, gardenLabel,yesGardenBtn,noGardenBtn);

*HBox* buttonsRow = new *HBox*(20, submitBtn,closeBtn);

buttonsRow.*setAlignment*(*Pos*.CENTER);

*VBox* houseOptionsArea = new *VBox*(10, optionsForHouseLabel,numberOfFloors,garden);

houseOptionsArea.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* apartmentOptionsArea = new *VBox*(10,optionsForApartmentLabel,floorNumber,roomNumber,extraFee);

apartmentOptionsArea.*setAlignment*(*Pos*.TOP\_LEFT);

*VBox* generalArea = new *VBox*(10,propertyType,sizeOfProperty,location,maximumNumberOfTenants,numberOfBedrooms,numberOfBathrooms,numberOfKitchens,price);

generalArea.*setAlignment*(*Pos*.TOP\_RIGHT);

*HBox* optionsArea = new *HBox*(30,houseOptionsArea,apartmentOptionsArea);

optionsArea.*setAlignment*(*Pos*.TOP\_CENTER);

*VBox* inforArea = new *VBox*(10,generalArea,optionsArea);

*VBox* root = new *VBox*(30, inforArea,buttonsRow);

root.*setAlignment*(*Pos*.CENTER);

*Scene* scene = new *Scene*(root,700,500);

stage.*setScene*(scene);

stage.*show*();

}

protected void *createSortOptionsWindow*(){

*Stage* stage = new *Stage*();

stage.*setTitle*("Sort Options");

stage.*initOwner*(primaryStage);

stage.*initModality*(*Modality*.APPLICATION\_MODAL);

*Label* optionsLabel = new *Label*("Options");

*Label* sortlabel = new *Label*("Sort");

*ToggleGroup* sortOptionsToggleGroup = new *ToggleGroup*();

*ToggleGroup* ascenOrDescenTogglegroup = new *ToggleGroup*();

*RadioButton* sizeBtn = new *RadioButton*("Size of Properties");

sizeBtn.*setToggleGroup*(sortOptionsToggleGroup);

*RadioButton* bedRoomsBtn = new *RadioButton*("Number Of Bedrooms");

bedRoomsBtn.*setToggleGroup*(sortOptionsToggleGroup);

*RadioButton* bathRoomsBtn = new *RadioButton*("Number Of Bathrooms");

bathRoomsBtn.*setToggleGroup*(sortOptionsToggleGroup);

*RadioButton* kitchensBtn = new *RadioButton*("Number Of Kitchens");

kitchensBtn.*setToggleGroup*(sortOptionsToggleGroup);

*RadioButton* priceBtn = new *RadioButton*("Rent Price");

priceBtn.*setToggleGroup*(sortOptionsToggleGroup);

*RadioButton* ascendingBtn = new *RadioButton*("Ascending");

ascendingBtn.*setToggleGroup*(ascenOrDescenTogglegroup);

*RadioButton* descendingBtn = new *RadioButton*("descending");

descendingBtn.*setToggleGroup*(ascenOrDescenTogglegroup);

*Button* applyBtn = new *Button*("Apply");

applyBtn.*setOnAction*(*event* -> {

if (ascendingBtn.*isSelected*() || descendingBtn.*isSelected*()){

if (sizeBtn.*isSelected*()) {

controller.*sortPropertiesRequest*("Size", ascendingBtn.*isSelected*());

stage.*close*();

} else if (bedRoomsBtn.*isSelected*()){

controller.*sortPropertiesRequest*("Bedrooms", ascendingBtn.*isSelected*());

stage.*close*();

} else if (bathRoomsBtn.*isSelected*()){

controller.*sortPropertiesRequest*("Bathrooms", ascendingBtn.*isSelected*());

stage.*close*();

} else if (kitchensBtn.*isSelected*()){

controller.*sortPropertiesRequest*("Kitchens", ascendingBtn.*isSelected*());

stage.*close*();

} else if (priceBtn.*isSelected*()){

controller.*sortPropertiesRequest*("Price", ascendingBtn.*isSelected*());

stage.*close*();

} else {

*createNotificationWindow*("Please choose one of these options to sort properties");

}

} else {

*createNotificationWindow*("Please choose options to show how properties are sorted (Ascending or Descending)");

}

});

*Button* closeBtn = new *Button*("Close");

closeBtn.*setOnAction*(*event* -> stage.*close*());

*HBox* sortRow = new *HBox*(10, ascendingBtn,descendingBtn);

sortRow.*setAlignment*(*Pos*.CENTER);

*VBox* options = new *VBox*(5,sizeBtn,bedRoomsBtn,bathRoomsBtn,kitchensBtn,priceBtn);

options.*setAlignment*(*Pos*.CENTER);

*VBox* optionsArea = new *VBox*(5, optionsLabel,options);

optionsArea.*setAlignment*(*Pos*.CENTER);

*VBox* sortArea = new *VBox*(5,sortlabel,sortRow);

sortArea.*setAlignment*(*Pos*.CENTER);

*HBox* mainArea = new *HBox*(10, optionsArea,sortArea);

mainArea.*setAlignment*(*Pos*.CENTER);

*HBox* buttonsRow = new *HBox*(30,applyBtn,closeBtn);

buttonsRow.*setAlignment*(*Pos*.CENTER);

*VBox* root = new *VBox*(30,mainArea,buttonsRow);

root.*setAlignment*(*Pos*.CENTER);

*Scene* scene = new *Scene*(root, 350,300);

stage.*setScene*(scene);

stage.*show*();

}

protected void *createEditUserWindow*(){

*Stage* stage = new *Stage*();

stage.*setTitle*("Edit User Information");

stage.*initOwner*(primaryStage);

stage.*initModality*(*Modality*.APPLICATION\_MODAL);

*Label* nameLabel = new *Label*("Name");

*Label* addressLabel = new *Label*("Address (Areas in Sydney)");

*Label* phoneNumberLabel = new *Label*("Phone Number");

*Label* usernameLabel = new *Label*("Username");

*Label* passwordLabel = new *Label*("Password");

*Label* confirmPasswordLabel = new *Label*("Confirm password");

*Label* roleLabel = new *Label*("Your are:");

*TextField* nameTextField = new *TextField*();

nameTextField.*setPromptText*("Type in your name (Ex: John, Emily,..)");

*TextField* addressTextField = new *TextField*();

addressTextField.*setPromptText*("Please type in with lowercase (Ex: central, st peters,...");

*TextField* phoneNumberTextField = new *TextField*();

phoneNumberTextField.*setPromptText*("Please type in a string of numbers (Ex: 04xxxxxxxx)");

*configTextFieldForInts*(phoneNumberTextField);

*TextField* usernameTextField = new *TextField*();

usernameTextField.*setPromptText*("Ex: jt253, john123,...");

*TextField* passwordtextField = new *TextField*();

*TextField* confirmPasswordTextField = new *TextField*();

*Button* submitBtn = new *Button*("Submit");

submitBtn.*setOnAction*(*event* -> {

*String* name = nameTextField.*getText*().*trim*();

*String* address = addressTextField.*getText*().*trim*();

*String* phoneNumber = phoneNumberTextField.*getText*().*trim*();

*String* username = usernameTextField.*getText*().*trim*();

*String* password = passwordtextField.*getText*().*trim*();

*String* confirmPassword = confirmPasswordTextField.*getText*().*trim*();

*List*<*String*> errors = controller.*editInforRequest*(name,address,phoneNumber,username,password,confirmPassword);

if (errors.*isEmpty*()){

*createNotificationWindow*("Completed edit!");

stage.*close*();

} else {

*String* errorString = "";

for (*String* error : errors){

errorString += "\n" + error;

}

*createNotificationWindow*(errorString);

}

});

*Button* cancelBtn = new *Button*("Cancel");

cancelBtn.*setOnAction*(*event* -> stage.*close*());

*VBox* setName = new *VBox*(5, nameLabel, nameTextField);

setName.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* setAddress = new *VBox*(5, addressLabel,addressTextField);

setAddress.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* setPhoneNumber = new *VBox*(5,phoneNumberLabel,phoneNumberTextField);

setPhoneNumber.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* setUsername = new *VBox*(5, usernameLabel, usernameTextField);

setUsername.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* setPassword = new *VBox*(5, passwordLabel, passwordtextField);

setPassword.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* setConfirmPassword = new *VBox*(5, confirmPasswordLabel, confirmPasswordTextField);

setConfirmPassword.*setAlignment*(*Pos*.CENTER\_LEFT);

*HBox* buttonsRow = new *HBox*(10, submitBtn,cancelBtn);

buttonsRow.*setAlignment*(*Pos*.CENTER);

*VBox* root = new *VBox*(25, setName, setAddress, setPhoneNumber, setUsername, setPassword, setConfirmPassword, buttonsRow);

*Scene* scene = new *Scene*(root,500,600);

stage.*setScene*(scene);

stage.*show*();

}

protected void *createNotificationWindow*(*String* *announcement*){

*Stage* stage = new *Stage*();

stage.*setTitle*("Notification");

stage.*initOwner*(primaryStage);

*Label* annoucementLabel = new *Label*(*announcement*);

*Button* closeBtn = new *Button*("Close");

closeBtn.*setOnAction*(*e* -> stage.*close*());

*VBox* root = new *VBox*(20, annoucementLabel, closeBtn);

root.*setAlignment*(*Pos*.CENTER);

*Scene* scene = new *Scene*(root,400,400);

stage.*setScene*(scene);

stage.*show*();

}

protected void *createSignUpWindow*(){

*Stage* stage = new *Stage*();

stage.*setTitle*("Sign Up");

stage.*initOwner*(primaryStage);

stage.*initModality*(*Modality*.APPLICATION\_MODAL);

*Label* nameLabel = new *Label*("Name");

*Label* addressLabel = new *Label*("Address (Areas in Sydney)");

*Label* phoneNumberLabel = new *Label*("Phone Number");

*Label* usernameLabel = new *Label*("Username");

*Label* passwordLabel = new *Label*("Password");

*Label* confirmPasswordLabel = new *Label*("Confirm password");

*Label* roleLabel = new *Label*("Your are:");

*TextField* nameTextField = new *TextField*();

nameTextField.*setPromptText*("Type in your name (Ex: John, Emily,..)");

*TextField* addressTextField = new *TextField*();

addressTextField.*setPromptText*("Please type in with lowercase (Ex: central, st peters,...");

*TextField* phoneNumberTextField = new *TextField*();

phoneNumberTextField.*setPromptText*("Please type in a string of numbers (Ex: 04xxxxxxxx)");

*configTextFieldForInts*(phoneNumberTextField);

*TextField* usernameTextField = new *TextField*();

usernameTextField.*setPromptText*("Ex: jt253, john123,...");

*TextField* passwordtextField = new *TextField*();

*TextField* confirmPasswordTextField = new *TextField*();

*ToggleGroup* roletoggleGroup = new *ToggleGroup*();

*RadioButton* agentToggleGroupBtn = new *RadioButton*("Agent");

agentToggleGroupBtn.*setToggleGroup*(roletoggleGroup);

*RadioButton* clientToggleGroupBtn = new *RadioButton*("Client");

clientToggleGroupBtn.*setToggleGroup*(roletoggleGroup);

*Button* submitBtn = new *Button*("Submit");

submitBtn.*setOnAction*(*event* -> {

*String* name = nameTextField.*getText*().*trim*();

*String* address = addressTextField.*getText*().*trim*();

*String* phoneNumber = phoneNumberTextField.*getText*().*trim*();

*String* username = usernameTextField.*getText*().*trim*();

*String* password = passwordtextField.*getText*().*trim*();

*String* confirmPassword = confirmPasswordTextField.*getText*().*trim*();

*String* role;

if (name.*isEmpty*() || address.*isEmpty*() || phoneNumber.*isEmpty*() || username.*isEmpty*() || password.*isEmpty*() || confirmPassword.*isEmpty*()){

*createNotificationWindow*("Please enter all of the information");

}

if (agentToggleGroupBtn.*isSelected*()){

role = "Agent";

} else if (clientToggleGroupBtn.*isSelected*()) {

role = "Client";

} else {

role = null;

*createNotificationWindow*("Please choose your role");

}

if (role != null) {

*List*<*String*> errors = controller.*signUpRequest*(name,address,phoneNumber,username,password,confirmPassword,role);

if (errors.*isEmpty*()){

*createNotificationWindow*("Your Account has been created!\nThanks for choosing to use our services");

stage.*close*();

} else {

*String* errorString = "";

for (*String* error : errors){

errorString += "\n" + error;

}

*createNotificationWindow*(errorString);

}

}

});

*Button* cancelBtn = new *Button*("Cancel");

cancelBtn.*setOnAction*(*event* -> stage.*close*());

*VBox* setName = new *VBox*(5, nameLabel, nameTextField);

setName.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* setAddress = new *VBox*(5, addressLabel,addressTextField);

setAddress.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* setPhoneNumber = new *VBox*(5,phoneNumberLabel,phoneNumberTextField);

setPhoneNumber.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* setUsername = new *VBox*(5, usernameLabel, usernameTextField);

setUsername.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* setPassword = new *VBox*(5, passwordLabel, passwordtextField);

setPassword.*setAlignment*(*Pos*.CENTER\_LEFT);

*VBox* setConfirmPassword = new *VBox*(5, confirmPasswordLabel, confirmPasswordTextField);

setConfirmPassword.*setAlignment*(*Pos*.CENTER\_LEFT);

*HBox* roleBtns = new *HBox*(40, agentToggleGroupBtn,clientToggleGroupBtn);

roleBtns.*setAlignment*(*Pos*.CENTER);

*VBox* setRole = new *VBox*(5,roleLabel,roleBtns);

setRole.*setAlignment*(*Pos*.CENTER);

*HBox* buttonsRow = new *HBox*(10, submitBtn,cancelBtn);

buttonsRow.*setAlignment*(*Pos*.CENTER);

*VBox* root = new *VBox*(25, setName, setAddress, setPhoneNumber, setUsername, setPassword, setConfirmPassword, setRole, buttonsRow);

*Scene* scene = new *Scene*(root,500,600);

stage.*setScene*(scene);

stage.*show*();

}

protected void *createViewMailWindow*(*Mail* *mail*){

*Stage* stage = new *Stage*();

stage.*setTitle*("View Detail Mail");

stage.*initOwner*(primaryStage);

*Label* fromLabel = new *Label*("From: " + *mail*.*getSender*().*get*());

fromLabel.*setAlignment*(*Pos*.CENTER\_RIGHT);

*Label* toLabel = new *Label*("To: " + *mail*.*getRecipient*().*get*());

toLabel.*setAlignment*(*Pos*.CENTER\_RIGHT);

*Label* tittleLabel = new *Label*("Tittle: " + *mail*.*getTittle*().*get*());

tittleLabel.*setAlignment*(*Pos*.CENTER\_RIGHT);

*Label* contentLabel = new *Label*("Content:\n" + *mail*.*getContent*().*get*());

contentLabel.*setAlignment*(*Pos*.CENTER);

*Button* closeBtn = new *Button*("Close");

closeBtn.*setOnAction*(*event* -> stage.*close*());

*VBox* headRow = new *VBox*(5, fromLabel,toLabel,tittleLabel);

*VBox* root = new *VBox*(30,headRow,contentLabel,closeBtn);

*Scene* scene = new *Scene*(root, 450,300);

stage.*setScene*(scene);

stage.*show*();

}

protected void *createSendEmailWindow*() {

*Stage* stage = new *Stage*();

stage.*setTitle*("New Mail");

stage.*initOwner*(primaryStage);

stage.*initModality*(*Modality*.APPLICATION\_MODAL);

*Label* fromLabel = new *Label*("From: " + model.*getCurrentUser*().*getUserID*() + " (Me)");

*Label* toLabel = new *Label*("To: ");

*Label* tittleLabel = new *Label*("Tittle: ");

*Label* contentLabel = new *Label*("Content: ");

*TextField* toTextField = new *TextField*();

toTextField.*setPromptText*("Type in the ID of recipient");

*TextField* tittleTextField = new *TextField*();

tittleTextField.*setPromptText*("Enter short tittle");

*TextArea* contentTextArea = new *TextArea*();

contentTextArea.*setPromptText*("Enter your content");

*Button* submitBtn = new *Button*("Submit");

submitBtn.*setOnAction*(*event* -> {

*String* to = toTextField.*getText*().*trim*();

*String* tittle = tittleTextField.*getText*().*trim*();

*String* content = contentTextArea.*getText*().*trim*();

*List*<*String*> errors = controller.*sendMailRequest*(to, tittle, content);

if (errors.*isEmpty*()) {

*createNotificationWindow*("Sent Successfully!!!");

stage.*close*();

} else {

*String* errorString = "";

for (*String* error : errors) {

errorString += "\n" + error;

}

*createNotificationWindow*(errorString);

}

});

*Button* closeBtn = new *Button*("Close");

closeBtn.*setOnAction*(*event* -> stage.*close*());

*HBox* toRow = new *HBox*(10, toLabel, toTextField);

*HBox* tittleRow = new *HBox*(10, tittleLabel, tittleTextField);

*HBox* buttonsRow = new *HBox*(10, submitBtn, closeBtn);

buttonsRow.*setAlignment*(*Pos*.CENTER);

*VBox* contentCol = new *VBox*(5, contentLabel, contentTextArea);

*VBox* headRow = new *VBox*(10, fromLabel, toRow, tittleRow);

*VBox* root = new *VBox*(20, headRow, contentCol, buttonsRow);

*Scene* scene = new *Scene*(root, 500, 300);

stage.*setScene*(scene);

stage.*show*();

}

protected void *createViewPropertyWindow*(*Property* *property*){

*Stage* stage = new *Stage*();

stage.*setTitle*("View Detail Property");

stage.*initOwner*(primaryStage);

*Label* detaillabel = new *Label*(*property*.*detailInfo*());

*Button* closeBtn = new *Button*("Close");

closeBtn.*setOnAction*(*event* -> stage.*close*());

*VBox* root = new *VBox*(30,detaillabel,closeBtn);

root.*setAlignment*(*Pos*.CENTER);

*Scene* scene = new *Scene*(root,400,350);

stage.*setScene*(scene);

stage.*show*();

}

protected void *configTextFieldForDoubles*(*TextField* *field*) {

*field*.*setTextFormatter*(new *TextFormatter*<*Integer*>((*Change* *c*) -> {

if (*c*.*getControlNewText*().*matches*("(\\d+.?\\d\*)?")) {

return *c*;

}

return null;

}));

}

protected void *configTextFieldForInts*(*TextField* *field*) {

*field*.*setTextFormatter*(new *TextFormatter*<*Integer*>((*Change* *c*) -> {

if (*c*.*getControlNewText*().*matches*("\\d\*")) {

return *c*;

}

return null;

}));

}

}

# THE EXPLANATION (300 WORDS)

## Explanation:

Firstly, the project begins with a welcome page where users must either sign up or log in to the system. In the AppView, we used text fields to capture user input for the username and password. To clearly indicate which text field corresponds to the username or password, we added labels and set prompt text for each field. When the user clicks the login button, the system trims the input from the text fields and sends the data to the AppController to handle the login request. The loginRequest method in the controller captures the username and password and passes them to the Model to execute the login method. If the user enters an incorrect username or password, the system triggers the createNotificationWindow() to prompt the user to try again.

Additionally, if the user chooses to create a new account instead of logging in, the system will call the createSignUpWindow method in the View, which opens a window for the user to enter their information. This sign-up window is initialized to the primary stage and set to modality, ensuring that the user must interact with it to complete the sign-up process. After the user clicks the submit button, the system collects all the input from the sign-up window and sends it to the Controller's signUpRequest method, which then runs the signUp method in the Model to create a new account. If any errors occur during this process, the View will display a notification window listing all the errors, and the system will inform the user once the account has been successfully created.

After logging in, the View will set the primary stage's scene to the home page, which is designed to offer several functions for the user, including account management, market access, mailbox, and other management tools. Additionally, this home page displays the user's information and includes a logout button. In the account section, users can edit their information by clicking on the "Edit Profile" button, which opens a new window for them to enter the details they wish to update. When the submit button in the edit window is clicked, the View captures the text entered in the fields and triggers an edit request in the Controller. The system then sends this information to the Model. The editProfile method in the Model returns a list of errors, if any, encountered while attempting to update the user's information. The account page also features a back button, allowing users to return to the home page by resetting the primary stage's scene to the home page.

In our project, we developed a mailbox feature that allows users to send and receive emails, enhancing interaction among users. On the mailbox page, we use a TableView to display the title and sender of all emails briefly. If users want to read the full content of an email, they can click the "View Details" button, which requires them to select an email from the table. After making a selection, the system calls the createViewMailWindow method to display all the information of the selected email in a new window. To achieve this, when the user selects and clicks "View Details," we record the index of the email in the list and use it to retrieve the email from the current user's mailbox, displaying it in the new window.

If users wish to delete an email, they can click the "Delete Email" button, which also records the email's index and sends it to the controller to execute the removeMail method in the Model. This method retrieves the current user's mailbox and deletes the email corresponding to the recorded index.

The "Send New Mail" function is one of the standout features of this project, allowing users to send an email to another user. When the "Send Mail" button is clicked, the system, thanks to setOnAction, creates a window where users must enter the recipient's name, email title, and content. After filling in all the fields, the system captures this information when the user clicks the submit button and sends it to the controller. The controller then calls the sendMailRequest method to instruct the Model to add a new email to the recipient's inbox with the provided information. If there are any errors, the system returns a list of error messages and creates a notification window via the View to display them. Otherwise, the user is notified that the email has been successfully sent to the recipient.

In the market page functions, we implemented two key features: "View Details" and "Sort Properties." The market page utilizes a ListView to display brief information about properties, including their type, rent price, and size. If a user wants to view more detailed information about a property, they can select the specific property from the ListView and click on "View Details." The View will then create a new window and invoke methods to display the full property details.

For users interested in properties with specific attributes, such as larger size or lower rent, the "Sort Properties" function allows them to sort the properties according to their preferences. This function opens a new window in the View where users can select sorting criteria by clicking on radio buttons, as well as choose the sorting order (ascending or descending). After clicking the "Apply" button, all properties will be sorted based on the selected criteria using comparators in the Model. The Controller handles the sort request and communicates with the Model to carry out the sorting process.

The final set of functions is called "Management," which includes tools for agents to manage properties and for clients to manage contracts. Depending on the user type, the View sets a different scene for the primary stage when they access the management page. Although both user types will see buttons and a TableView, each has unique functionalities tailored to their needs.

For agents, the TableView lists the property ID, the number of tenants, and the rent price, making it easy for them to monitor their inventory. Below the TableView, there are buttons such as "View Details," "Add New Property," and others.

First, the "Add New Property" function prompts the View to create a new window where the user can enter property information into a form. This information is then sent to the Controller, which calls the addProperty method in the Model. If the user input contains errors, such as a negative number of rooms or a zero tenant capacity, these errors are returned as a list of strings and displayed to the user in a notification window.

Second, if an agent needs to update property information, they can click the "Edit Property Information" button. This function also prompts the View to create a new window where the agent can enter the updated information. Any errors in the input will be returned as a list, and the user will be notified accordingly.

For deleting a property, the agent can select a property from their inventory and click the "Remove Property" button. The View sends a request to the Controller to invoke the removeProperty method in the Model, which removes the property from the user's inventory, any associated tenant leases, and the system as a whole.

Finally, since the TableView only displays key property details, agents can click on the "View Details" button to see all the information about a selected property. The View will then create a new window to display the complete property details.

For clients, the management page also features a TableView that displays the property ID, owner ID, and rent price. This information helps clients easily contact the property owner if they wish to negotiate or communicate. The management page offers three main functions for clients: "View Details," "Make Contract," and "Cancel Contract."

Similar to agents, if a client wants to view detailed information about a property—such as the number of bedrooms, presence of a garden, or number of floors—they need to select the property from the TableView and click on "View Details." This function prompts the View to create a new window displaying all the property details.

If a client wants to sign a rental contract for a property they found on the market, they can use the "Make Contract" function. When this button is clicked, the system creates a new window asking the user to enter the property ID of the property they wish to lease. Based on the user's input, the label displaying property information will update accordingly. If the user enters a valid property ID, the label will show all the relevant details of the property, allowing the user to verify that it is the correct property they wish to lease. Once the user clicks the "Sign" button, the Controller calls the makeContract method, instructing the Model to send a notification to the owner that they have a new tenant. The property is then added to the user's list of leases, and the tenant is added to the property's tenant list.

On the other hand, if a client wishes to cancel a rental contract, the "Cancel Contract" function is available. This button records the index of the selected property from the TableView and calls the Controller to execute the cancelContractmethod. This method notifies the property owner that a tenant has canceled their rental agreement. Additionally, the function removes the user from the list of tenants for that property, updates the countTenants (the number of tenants currently renting the property), and deletes the property from the user's list of leases.

## Issues:

1. ***Switch from HashMap/List to ObservableList for UI Selection Compatibility***

In Project A, we contacted a property or user by their ID. However, in this project, users can click on a table or list view to select an object directly. Therefore, I had to change the type of attributes from HashMap or List to ObservableList to allow users to select an object.

This issue arises due to the difference in how data is interacted with between the two projects. The initial project used IDs to identify and manipulate objects, which worked well with HashMap or List. However, the new project requires the ability to select items directly from the user interface (via a table or list), necessitating the use of ObservableList. ObservableList supports data binding with the UI, enabling automatic updates and allowing users to select objects more intuitively.

1. ***Dynamic Label Update with addListener for Real-Time Property ID Validation in makeContract Method***

In the makeContract method, we want to validate the user's input. When a user types in a valid property ID to sign a contract, we aim to display the details of the property for the user to review. However, we encountered an issue where we didn't know how to update the label dynamically as the user entered new input. To resolve this, we revisited the lesson and utilized the addListener method for the text field where the user enters the property ID. This addListener monitors changes in the text field and, each time the user inputs a new value, it triggers a check of the property ID by calling the controller, which in turn requests the model to verify the ID. If the property ID is valid, the model returns a property object, which is then used to update the label on the window, allowing the user to see the property details in real-time.

1. ***Optimizing Error Handling with Multi-Error Reporting for User Input Validation***

When we were developing the login, sign-up methods, or any methods requiring user input, we encountered a significant problem: we didn't know how to effectively inform the user when their input was incorrect or invalid. Initially, we considered having the methods return a single string that would contain an error message if the model detected any errors. However, this approach was not ideal because if the user's input caused multiple errors, the model would have to perform repetitive work, which could slow down the system's response time and increase the overall processing time.

To address this issue, we decided to return a list of strings instead. This list would contain all the error messages related to the user's input, allowing them to correct all the mistakes at once before submitting again. This approach not only reduces the workload on the model by avoiding redundant checks but also ensures the system processes input more quickly and smoothly, resulting in a better user experience.



# You will submit your assessment in MS Word document format – yourName-studNo.doc.

The document will have 3 sections:

GUI Design: Your designed GUI including buttons, panels, labels and textfields to support your application.

The code: You will copy the code from VS Code into this section The explanation.

# You will also submit a copy of your VS Code project as a zip file



| CRITERIA | WEIGHT | SLOs / LLOs | PLOs |
| --- | --- | --- | --- |
| **Functional specifications** | **35%** | **2, 3** | **A1, B1, E1** |
| **Code layout** | **35%** | **1, 3, 4, 5** | **A1, B1, E1** |
| **Explanation** | **30%** | **4, 5, W4** | **C1** |

SLOs: subject learning outcomes PLOs: program learning outcomes LLOs: language learning outcomes

Please refer to the next page for the rubric definition for each criteria and grade.





|  | | HD | D | C | P | F3 | F2 | F1 | F0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | Functional specifications | The program works correctly | The program runs and | The program produces | The program is producing | The program is producing | The program is producing all incorrect | The program does not run; | Program does no |
|  |  | in every way and | produces mostly | mostly correct | some | mostly | results or mostly | Some code is | required |
|  |  | meets all of the | correct results | results using | incorrect | incorrect | incomplete. | valid | tasks |
|  |  | functional | using MVC | MVC | results or a | results or a |  |  |  |
|  |  | specifications | architecture. It | architecture. | part of the | most of the |  |  |  |
|  |  | using MVC | also meets most | Some minor | program is | program is |  |  |  |
|  |  | architecture. The | of the other | functionality | incomplete. | incomplete. |  |  |  |
|  |  | interface is very | specifications. | may be | Some MVC |  |  |  |  |
|  |  | intuitive and | Some error | missing. The | architecture is |  |  |  |  |
|  |  | easy to use. | checking may | interface is | present but |  |  |  |  |
|  |  |  | be missing. The | somewhat | may not be |  |  |  |  |
|  |  |  | interface is | intuitive. | working |  |  |  |  |
|  |  |  | intuitive and |  | correctly. The |  |  |  |  |
|  |  |  | easy to use. |  | interface isn’t |  |  |  |  |
|  |  |  |  |  | intuitive to |  |  |  |  |
|  |  |  |  |  | use. |  |  |  |  |
| **2** | Code layout | The code is | The code is easy | The code is | The code is a | The code is | The code is poorly | The code is | No valid |
|  |  | exceptionally | to follow and | fairly easy to | little hard to | hard to read | organised and very | poorly | code |
|  |  | well organised, | well organised | follow and | read and | and variable | difficult to follow. | organised and | written |
|  |  | flow of control is | and variable | variable | variable | names chosen | Variable names chosen | very difficult to |  |
|  |  | very easy to | names chosen | names chosen | names chosen | at least once | do not help understand | follow. Variable |  |
|  |  | follow and | are very useful | are useful in | are somewhat | help | what they are used for. | names chosen |  |
|  |  | variable names | in | understanding | useful in | understand | Reader must guess what | use names that |  |
|  |  | chosen clearly | understanding | the code. | understanding | what they are | variables are for. Java | will not compile |  |
|  |  | explain what the | what the code | Some design | the code. The | used for. Java | naming conventions not | e.g. with |  |
|  |  | code is | does and how it | rules or | flow of | naming | used. | spaces. Java |  |
|  |  | accomplishing | is accomplished. | patterns have | control is | conventions | Only 1 Java construct is | naming |  |
|  |  | and how. No design rules or | Some minor design rules or | been broken. Java naming | difficult to follow. Java | not used. | from week 1 to 9. | conventions not used. |  |

|  |  | patterns have been broken. Java naming conventions always used. All Java constructs are from week 1 to 9. | patterns have been broken. Java naming conventions mostly used. Most Java constructs are from week 1 to 9. | conventions sometimes used.  Java constructs are generally from week 1  to 9. | naming conventions used at least once.  Some Java constructs are from week 1  to 9. |  |  |  |  | No Java construct is from week 1 to 9. |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A few Java  constructs are from week 1  to 9. | |  |
| **3** | Explanation |  |  |  |  |  | |  | |  | No Sub- |
|  |  | Demonstrates a | Demonstrates a | Demonstrates | Demonstrates | Demonstrates | | Demonstrates almost no | | Demonstrates | mission |
|  |  | clear and | clear | a moderate | little | very little | | understanding of GUI | | almost no |  |
|  |  | comprehensive | understanding | understanding | understanding | understanding | | concepts. | | understanding |  |
|  |  | understanding of | of GUI concepts. | of GUI | of GUI | of GUI | | The code contains no | | of GUI |  |
|  |  | GUI concepts. | The code | concepts. | concepts. | concepts. | | comments that explain | | concepts. |  |
|  |  | The code | contains many | The code | The code | The code | | steps in the MVC pattern | | The code |  |
|  |  | contains | comments that | contains some | contains a few | contains a | |  | | contains no |  |
|  |  | comprehensive | explain steps in | comments | comments | couple of | |  | | comments that |  |
|  |  | comments that | the MVC | that explain | that explain | comments | |  | | explain steps in |  |
|  |  | explain steps in the MVC pattern | pattern | steps in the MVC pattern | steps in the MVC pattern | that explain steps in the MVC pattern | |  | | the MVC pattern. |  |