Connor, Andrew, Matt, Jonathan - Design II

Here are the use cases so you can understand what the code is trying to accomplish

The flow of the system is user builds a 2D floor plan, loads then floorplan as a house in 3D, designs the interior as an interior decorator would, selects a location to build the house. The last epic or stakeholder system is a bunch of classes that would ideally be deployed as APIs to allow 3rd parties to integrate into the system and let respective stakeholders view relevant information about the status of things.

Floor Plan/ Exterior

**Use Case:** user wants to add a wall

**Use Case:** user wants to select a wall material

**Use Case:** user wants to remove a wall

**Use Case:** user wants to transition into 3D Modeling

Interior Design

**Use Case:** user wants to change selected room

**Use Case:** user wants to get updated furnishing suggestions

**Use Case:** selecting an interior style (modern, contemporary, traditional, industrial, minimalist, etc.)

**Use Case:** adding furnishing items to a room

Location Picker

**Use Case:** User searches a city/county in the location picker

**Use Case:** User selects a property to further evaluate property details

**Use Case:** User bookmarks a property

**Use Case:** Software suggesting location near the user

Stakeholder System

**Use Case:** Making a request for additional supplies

**Use Case:** The client receives an update when a portion of their house is completed.

**Use Case:** accessing a client’s financial records

**Use Case:** Checking supply of materials for a particular house

//, # represents comments

**Exterior Design / Floor Plan Pseudo Code**

# Inner Building Objects

# Overarching project, can contain multiple floors,

Class FloorPlan() {

Private planID

Private numFloors

}

# Floor class containing walls that make up the specific floor’s blueprint

Class Floor() {

Private floorName

Private floorNumber

Private wallCount

Private totalDimensions[]

Private walls\_list

}

# Wall class

Class Wall() {

Private wid

Private length, width

Private color

Private material

Private isConnected

Private isInterior

Private isExterior

}

# UI File Editor Class

# Has the ability to create new floorplans, save the project, change views, and transition into 3D

# modeling functionality.

# Can also use “validate” method to cross check compatibility with architecture, materials, and

# location

Class FileController {

Void newProject(); // initialize FloorPlan and one Floor object

Void saveProject();

Void openProject();

Void newFloor();

Void saveFloor();

Void nameFloor();

// check compatibility w/ architecture, materials, location

// under the top left corner files icon, click on the drop down option to validate a floor plan

// to invoke method

Void validate() {

// assert total dimension = walls used/ interior space

// check all walls connected

for x in walls\_list :

If !x.connected { return false printf(Error Not All Walls Connected) }

// if material is defined, check structural architecture/compatibility with..

// dimensions, number of floors with external architecture math API

}

// precondition: in 2D blueprinting mode

// transition into 3D modeling portion of software

Void changeToThirdDim() {

// Change UI to 3D

}

// precondition: already in 3D modeling “mode”

// transition into 2D blueprinting mode

Void changeToSecondDim() {

// Change UI to 2D

}

}

# Represents the toolbar that assists the user in the creation of their “blueprint” like floor plan

# Has the ability to edit wall attributes such as length, width, material as well as remove them.

Class EditController() {

// add wall onto the floor plan

// parameters x and y coordinates generated through mouse position

Void addWall(int x, int y) {

// create a new wall object, set all of its attributes, then add to the wall list

//renderUI

Wall = new Wall

Wall.wid = random.generate(a-0) // generate a wall ID

Wall.length = math.abs(y - x) // length

Wall.width = defaultWidth // 0.5 m change in editor

Wall.color = none // change in editor

Wall.material = none // change in editor

wall\_list(Floor.wallCount) = Wall

Floor.wallCount++

Wall.isConnected = false

Wall.isInterior = false

Wall.isExterior = false

If (x + 1, x - 1 && y + 1, y - 1 ! empty) { // if wall exists past the x and y starting

//and ending coordinate,

Wall.isConnected = true;

}

// User can indicate if exterior or interior on right hand side toolbar after creation

calculateTotalDimension() // update total dimension

renderUI()

}

// remove wall from floor plan

// take wall id parameter, remove from floorplan, render UI

// after clicking on a wall, in the right hand side toolbar, click the remove wall button

Void removeWall(int wid) {

wall\_list.remove(wid)

//remove from UI

calculateTotalDimension() // update total dimension

renderUI()

}

//changeWallLength

// takes in a wall id parameter and desired length, wall id from mouse click on desired

// wall

// after clicking on a wall, in the right hand side toolbar, input a new desired length for the

// wall.

Void changeWallLength(wid int len) {

Wall(wid).length = len

renderUI()

}

//change wallWidth

// takes in a wall id parameter and desired width, wall id from mouse click on desired

// wall

// after clicking on a wall, in the right hand side toolbar, input a new desired width for the

// wall.

Void changeWallWidth(String wid, int len) {

Wall(wid).width = len

renderUI()

}

// change material of a wall

// takes in a wall id parameter and desired material

// after clicking on a wall, in the right hand side toolbar, input a new desired material for

// the wall by clicking

Void changeMaterial(wid, Material mat) {

Wall(wid).material = mat

// prompt user if the whole floor should be the same material interior/exterior

// separated.

If output == yes

For x in wall\_list

#for interior

If x.isInterior = wall\_list(x).material = mat

# for exterior

If x.isExterior = wall\_list(x).material = mat

}

}

**Interior Design Pseudo Code**

-> This section contains a basic neural network Algorithm

Global FurnitureWeights { } #dictionary

Interface Furniture{

changeColor()

changeMaterial()

changeStyle()

editDimensions()

}

# create couch, tv, each will implement the Furniture interface differently as each piece of

# furniture is different … example of TV below

Class TV implements Furniture{

Private static FurnitureParts[ ] # list of parts ex. Screen, bezel, stand, these don’t

change

private Dimensions{ } # dictionary of dimensions of each part, ex. bezels: {height: 10in,

width: 10in, depth: 5in}

Private Colors{ } # ex. { stand: black, bezels: silver}

Private Materials{ } # ex. { stand: wood, bezels: steel}

Public void changeColor(part, newcolor){

Colors[part] = newcolor

}

Public void changeMaterial(part, newMaterial){

Material[part] = newMaterial

}

Public void changeStyle( getStyle(newStyle) ){

For part in FurnitureParts:

changeColor(part, newStyle[color] )

changeMaterial(part, newStyle[materials] )

}

Public dictionary getDimensions(){

Return Dimensions

}

# newDimensions is dictionary of new dimensions ex{height: 7in, width: 7in, depth: 2in}

Public void editDimensions( part: Furniture, newDimensions: dictionary){

Dimensions[part] = newDimensions

}

}

# here style values are hardcoded

Public Class Style{

Private final static Modern = {

AccentColors = [ black, dark brown ]

MainColors = [ white, silver, light brown ]

Materials = [ Glass, Wood, Steel ]

}

Private final static Rustic = {

AccentColors = [ white, light blue]

MainColors = [ brown, light brown, dark brown, black]

Materials = [ wood, leather, fur, stucco ]

}

Private final static Industrial = {

AccentColors = [ teal blue, light beige ]

MainColors = [ brown, red ]

Materials = [ tiles, concert, cement, brick ]

}

Public static getStyle( style ){

If style == modern:

Return Modern

Else If style == rustic:

Return Rustic

Else If style == industrial:

Return Industrial

}

}

# depending on room and style will output suggest furniture you might use

# the furniture you click on will have a positive association with the room and style and

# be weighted more heavily while the furniture that never gets clicked will have a negative

# Association and get weighted less, how recommendations get better over time

# in python if you have a matrix cast as a Dataframe and do an operation like +, -, \* or add (if # x == 1), python will intelligently apply that operation to every row in the matrix / Data Frame # … hence why I wrote the pseudo code as I did

# ALGORITHM

Class RecommendationEngine{

Public List static RunEngine(){

Inputs = convertStringtoInput( getRoom() , getStyle() )

# algo’s neural engine, inputs,outputs, FurnitureWeights are all matrices

Outputs = inputs \* FurnitureWeights # matrix math

If outputs > 0

Outputdisp = 1 # display furniture

Else:

Ouptutdisp = 0 # don’t display furniture

# building Error matrices to update furniture weights

# clicks are stored for each object in each room

For itemOutput in Outputs:

If numclickonObject > 0:

Error.add( itemOutput + constant\*numclickonobejct )

Else:

Error.add( itemOutput - constant\*totalClicks )

FurnitureWeights += Error\*input

OutputdispList.add( furnitureItem if Outputdisp[furnitureItem] == 1)

Return OutputdispList

}

}

# converts the raw data to a nice UI for user

# On x,y,z will show the conditions under which UI will re-render and the Recommendation

# Engine will re-run to update what it displays to the user and train the algo so it gets better as

# user interacts with the program more

Class renderUI () {

public static void main(String [] args)

{

spinRender() # spin means re-run function continously as long as program is

alive

}

Private void spinRender(){

# clicking on arrows in UI will change room

On roomChange:

newFurnitureList = RecommendationEngine.RunEngine()

reRender( recommendationList )

# recommendation engine updates on add and removes (learning what you

# Don’t want associated with a room

On furnitureItemAdd or furnitureItemRemove:

newFurnitureList = RecommendationEngine.RunEngine()

reRender( recommendationList )

reRender( Room )

On itemPropertyChange:

reRender( item )

On FloorPlanChange:

reRender( floorPlan)

}

}

Class Rooms( RoomName ) {

Private RoomName

Private FurnitureList[ ] # list of parts ex. Screen, bezel, stand, these don’t

change

private Dimensions{ } # dictionary of dimensions, ex. {height: 10in,

width: 10in, depth: 5in}

Private Walls[ ] # list of wall objects

Private Windows[ ] # list of wall objects

Private Floor

Public addFurniture( furniture, roomLocation ){

If ( roomLocation.check( furniture.getDimensions ) ):

FurnitureList.add( furniture )

}

Public removeFurniture( furniture ){

FurnitureList.remove( furniture )

}

Public changeRoomStyle( style ){

For furnitureItem in Room:

furnitureItem.changeStyle(style)

For wall in Walls:

wall.changeStyle(style)

}

Public string getRoom(){

Return RoomName

}

}

**Location Picker Pseudo Code**

-> This section contains an insertion sort algo to help with search by specific parameters

#Class is designed to read in a string and use a search algorithm to find available locations

#within the specified area

# search algo uses an insertion sort to order the properties according to relevance of some criteria ex.

#Distance

Public Class LocationPicker {

Private String input

Private String finalLocation

Private Location loc

#Location Picker Constructor

Public LocationPicker(String input) {

This.input = input

finalLocation = “”

inputReader(input)

}

#Read line of input and then call search algorithm

# ex. In could be distance, property size, cost / price

Public void inputReader(String in) {

Scanner scan = new Scanner()

String line = scan.nextLine(in)

finalLocation = search(line)

}

#Search using an insertion sort algorithm

Public List search(String in) {

List list = loc.getPropertyList()

#sort list of properties based on what user searches

List result = insertionSort(list, in)

#Display Locations within a specified radius

Return result

}

Public List insertionSort(List arr, String in)

{

for (int i = 1; i < arr.length; i++)

{

# in python this would get all values of a column where name matched string ‘in’

# this way its sorting based on distance, property size, cost / price, connectivity etc.

int valueToSort = arr.get(in)

int j

for ( j = i; j > 0 && arr[j - 1] > valueToSort; j--) {

arr[j] = arr[j - 1]

}

arr[j] = valueToSort

}

Return arr

#Gets the found final location

Public String getFinalLocation() {

Return finalLocation

}

}

#Location class stores list of locations that are within searched area

Public Class Location {

Private List propertyList

Private int numberOfProperties

#Location Constructor

Public Location() {

propertyList = new List()

numberOfProperties = 0

}

#Insert Location to a list if it is within searched area

Public void insert() {

Property prop = new Property(name)

list.add(prop)

}

#Clear or remove all locations before next search is made

Public void clear() {

propertyList = new List()

numberOfProperties = 0

}

#Returns list of properties

Public List getPropertyList() {

Return propertyList

}

}

#Property class gets details from each available property

Public Class Property {

Private String name

Private String address

Private int dimensionX

Private int dimensionY

Private double cost

# like an enum in C

Private boolean[] connectivity[internet, water, electric]

#Property Constructor, stores information for a property

Public Property(name, address, dimX, dimY, cost) {

This.name = name

This.address = address

This.dimensionX = dimX

This.dimensionY = dimY

This.cost = cost

}

#Gets name of property

Public String getName() {

Return name

}

#Gets address of property

Public String getAddress() {

Return address

}

#Gets dimensions of property

Public int getDimensions() {

Return dimensionX \* dimensionY

}

#Gets cost of property

Public double getCost() {

Return cost

}

#Returns true if property has internet access

Public boolean getConnectivity( type) {

Return connectivity[type]

}

}

**Stakeholder System**

// Deploys/executes the application

Class Application {

// Main method for execution, utilizes SpringApplication

public static void main(String[] args) {

SpringApplication.run(Application.class, AccountService.class, args)

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Inbox class

class Inbox {

# private fields

private String accountID;

private List<Message> messages;

private List<Notifications> notifications;

// constructor

Inbox() {

super();

}

// getter methods

public List<Message> getMessages() {

return messages;

}

public List<Notifications> getNotifications() {

return notification;

}

// Allow the user to modify the inbox, i.e. archive and delete messages

public void editInbox(String condition) {

if (condition.equals(“Archive”) {

// have the user choose a message to save

Message m = mouseClicked()

messages.add(m) // append the message to archive

}

Else if (condition.equals(“Delete”) {

String deleted= mouseSelected ()

// have the user select the one to delete

supplies.remove(newSudeletedpply)

// remove the existing message from the list

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Message class

class Message {

# private fields

private String accountID;

private Account sender;

private List<String> recipients;

private List<String> attachments;

// constructor

Message() {

super();

}

// two getters

public List<Message> getMessage() {

return message;

}

public List<Notifications> getAttachments() {

return attachments;

}

// method to send a message.

// will use java api POST calls

public void sendMessage() {

// encode the message,

URLencoder.encode(String, UTF-8)

Open.connection(); // open connection to recievers

connect();

setRequestMethod(POST) // api POST call

responseCode = getResponseCode()

if responseCode == HTTPURLConnection { // if connection successful, read message

// read line from keyboard

While (readline()) {

Message.append(character)

}

}

close(); // close connection

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class House {

// Private fields

private String Financier;

private String ContractingCompany;

private String Supplier;

private List<String> supplies

// constructor

House() {

super()

}

// getters

public String getFinancier() {

Return financier

}

public String getContractor() {

Return ContractingCompany

}

public String getSupplier() {

Return financier

}

// Ability to edit existing supply list of a house

public void editSupplies(String condition) {

If (condition.equals(“Edit”) {

String supply = mouseSelected // allow user to select the supply to modify

If (supplies.contains(supply)) {

supplies.edit(realLine()) // edit existing element

}

}

Else if (condition.equals(“Add”) {

String newSupply = readLine() // have the user enter in a new supply

supplies.add(newSupply) // append the new supply to the back of the list

}

Else if (condition.equals(“Delete”) {

String newSupply = mouseSelected ()

// have the user select the one to delete

supplies.remove(newSupply)

// remove the existing supply to the back of the list

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// import statements

Import javax.model.Person;

Import javax.model.Response;

Import javax.ws.rs.GET;

# security wasn’t a functional requirement, but would be built in here

Class Account {

// private fields

private String accountID;

private String password;

private List<House> houses;

private Inbox inbox;

public void createAccount(String ID, String pass) {

Response r = new Response();

Person p = new Person(ID, pass)

// if

If (p.getID() != null || invalid & p.getPassword() != invalid) {

add(Person) // add new account to database

}

}

// Will return account information for a user, uses GET api call

public String getAccountInformation() {

HTTPURLConnection conn = openConnection(URL);

// api GET call

conn.setRequestMethod(“GET”)

// have user enter in proper identification

conn.setRequestProperty(readline(), readline())

Int response = conn.getResponseCode()

If (response == HTTP-OK) {

StringBuilder message;

// get inputstream from the response of the connection

InputStream get = new InputStream(conn.getInputResonse())

BufferedReader input = new BufferedReader(get)

// read and append the message

While (input.readline != null) {

// get the returning message from the input stream

message.append(readLine)

}

// if message properly received by client, return the message

If (builder != null) {

Return builder

}

Else {

print(error.trace())

}

}

}

}

**UI**

Here are the use cases so you can understand what the UI is trying to accomplish. While we are not going to write what every part of each UI does, this will be explained in our final presentation. The UI is designed to meet the following functionality described by the use cases

The flow of the system is user builds a 2D floor plan, loads then floorplan as a house in 3D, designs the interior as an interior decorator would, selects a location to build the house. The last epic or stakeholder system is a bunch of classes that would ideally be deployed as APIs to allow 3rd parties to integrate into the system and let respective stakeholders view relevant information about the status of things.

Floor Plan/ Exterior

**Use Case:** user wants to add a wall

**Use Case:** user wants to select a wall material

**Use Case:** user wants to remove a wall

**Use Case:** user wants to transition into 3D Modeling

Interior Design

**Use Case:** user wants to change selected room

**Use Case:** user wants to get updated furnishing suggestions

**Use Case:** selecting an interior style (modern, contemporary, traditional, industrial, minimalist, etc.)

**Use Case:** adding furnishing items to a room

Location Picker

**Use Case:** User searches a city/county in the location picker

**Use Case:** User selects a property to further evaluate property details

**Use Case:** User bookmarks a property

**Use Case:** Software suggesting location near the user

Stakeholder System

**Use Case:** Making a request for additional supplies

**Use Case:** The client receives an update when a portion of their house is completed.

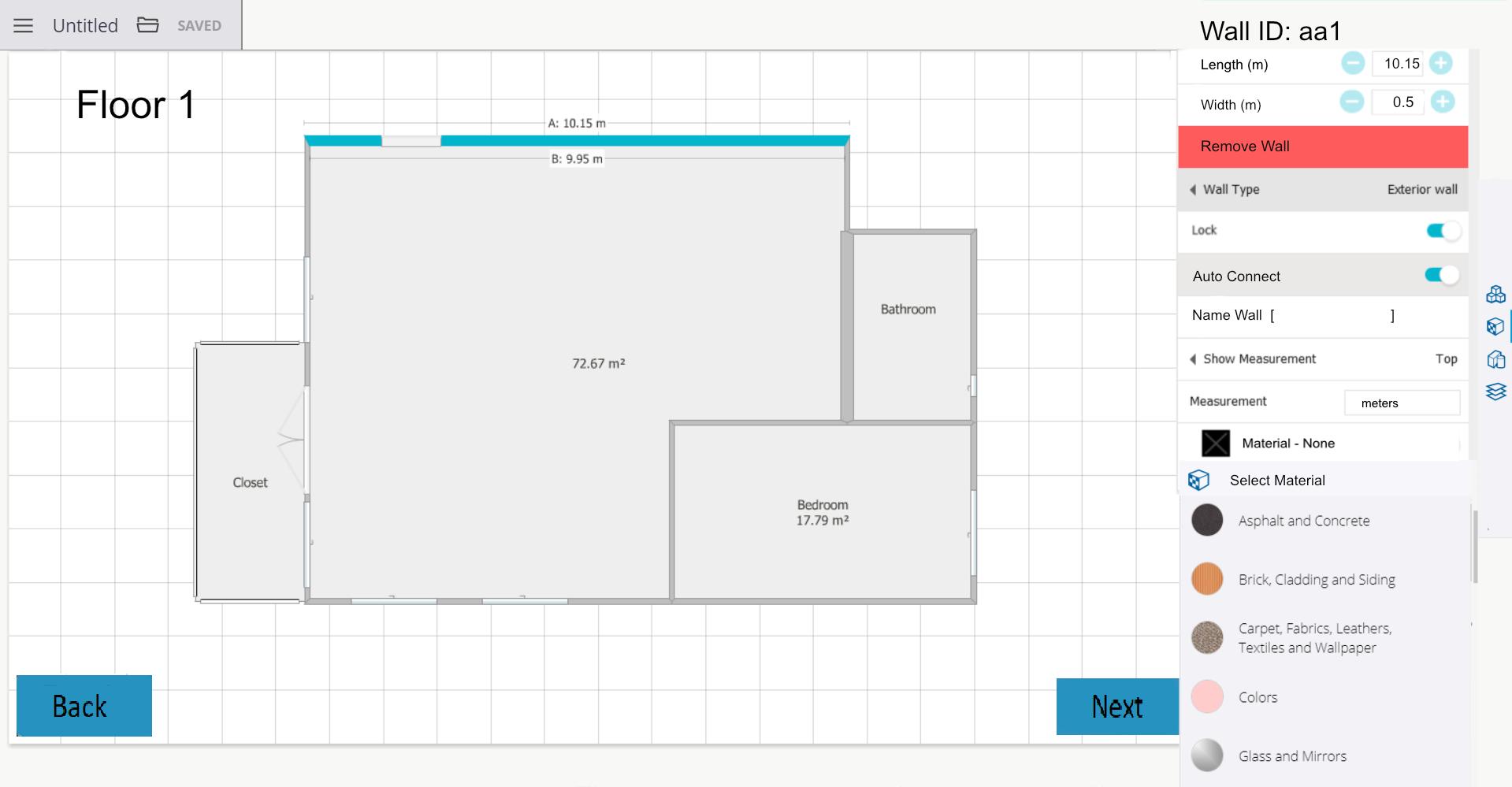
**Use Case:** accessing a client’s financial records

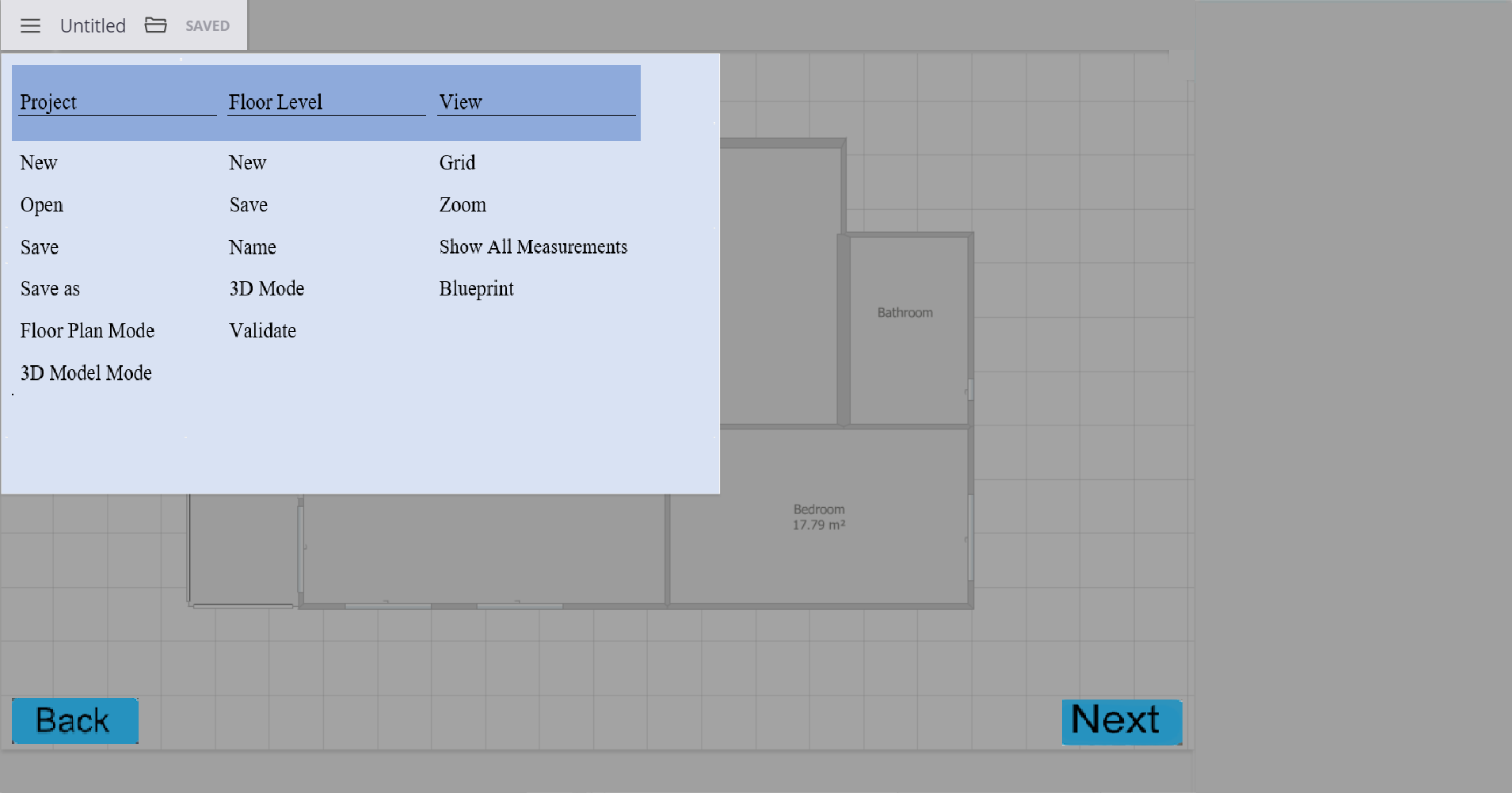
**Use Case:** Checking supply of materials for a particular house

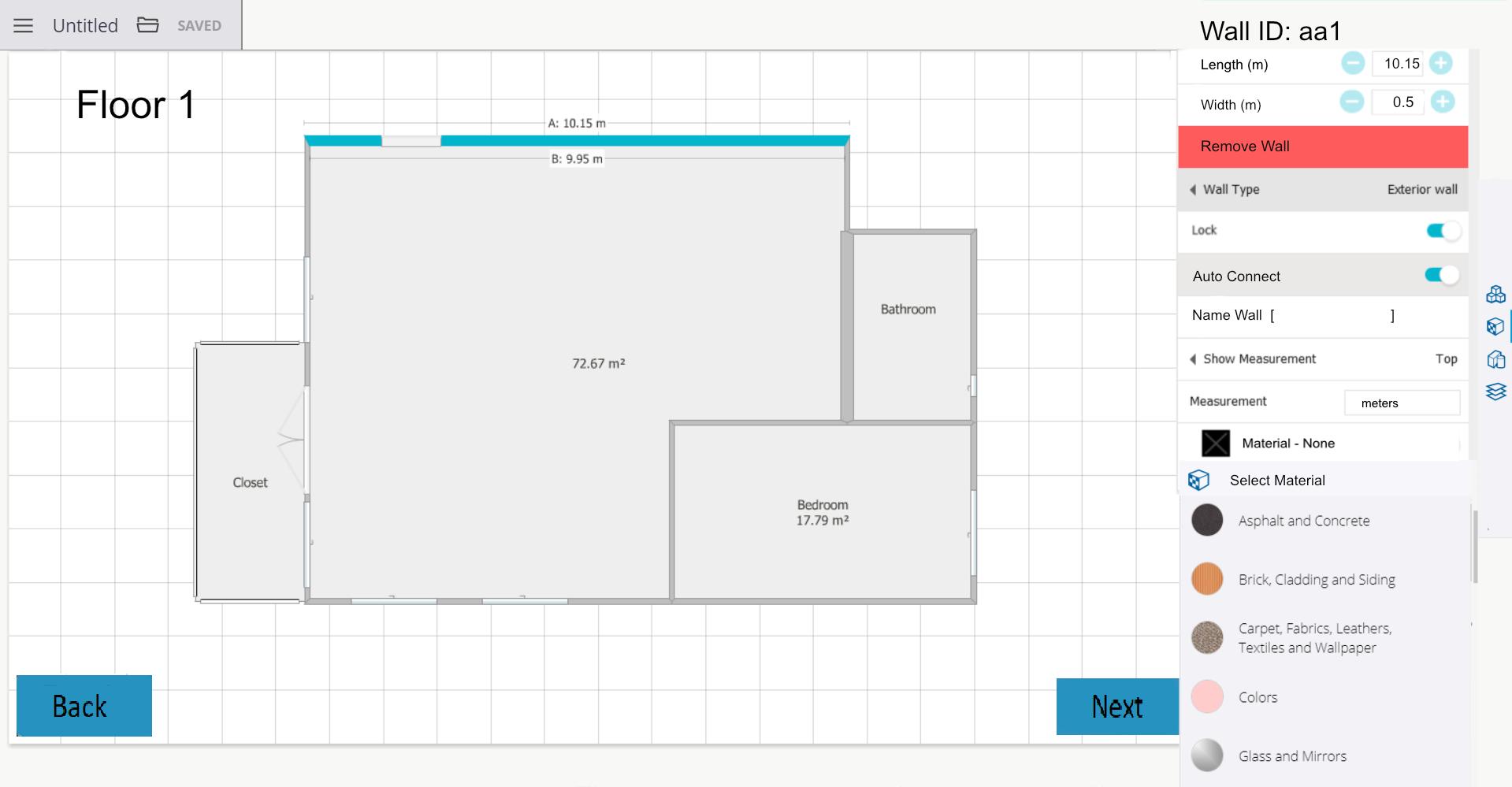
**Floor Plan/ Exterior**

Pic 1) Lets User transition in 3D model, save , open , see their floor plans

Pic 2) lets users add, select, modify, remove a wall(s) and windows



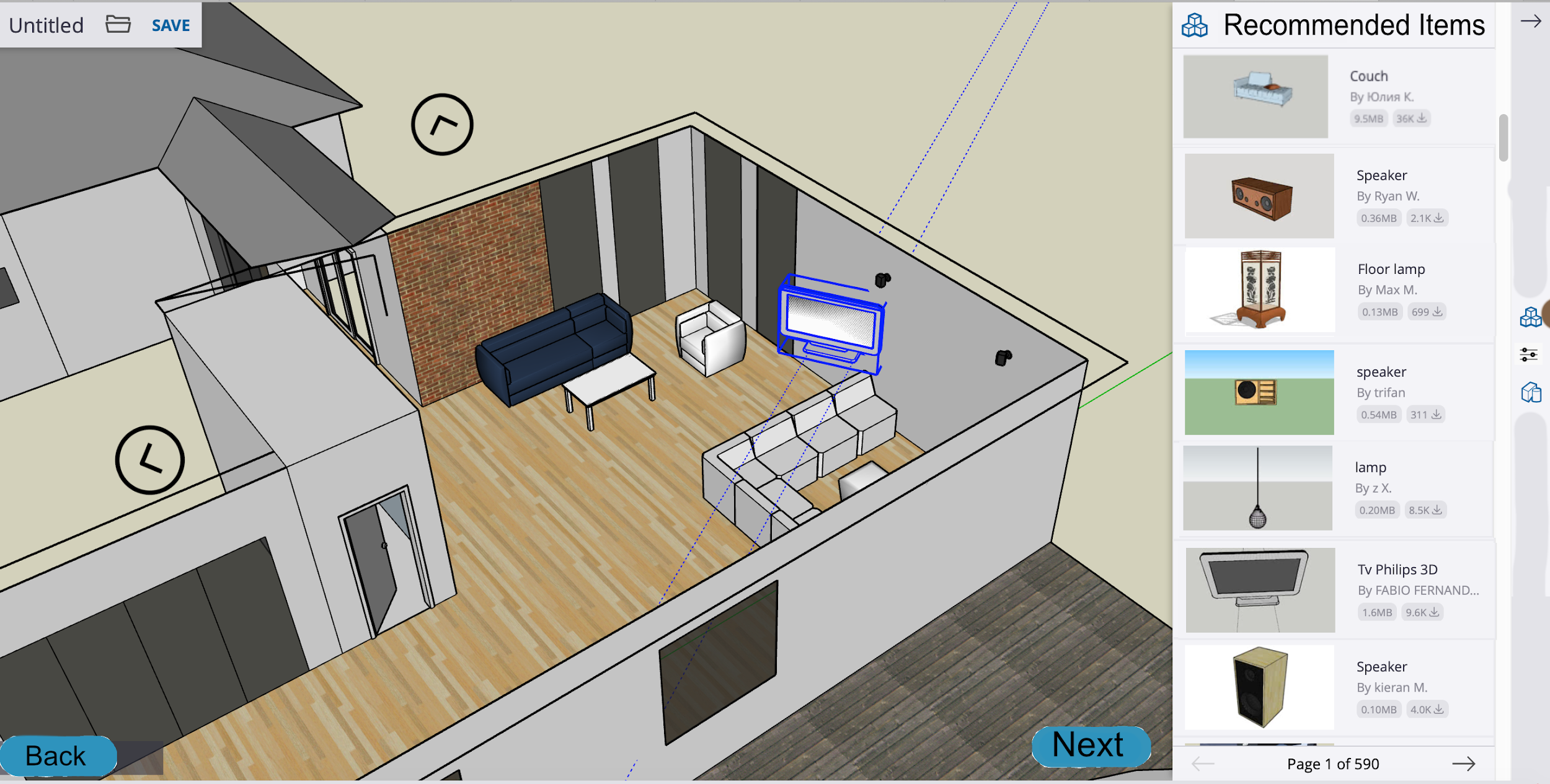


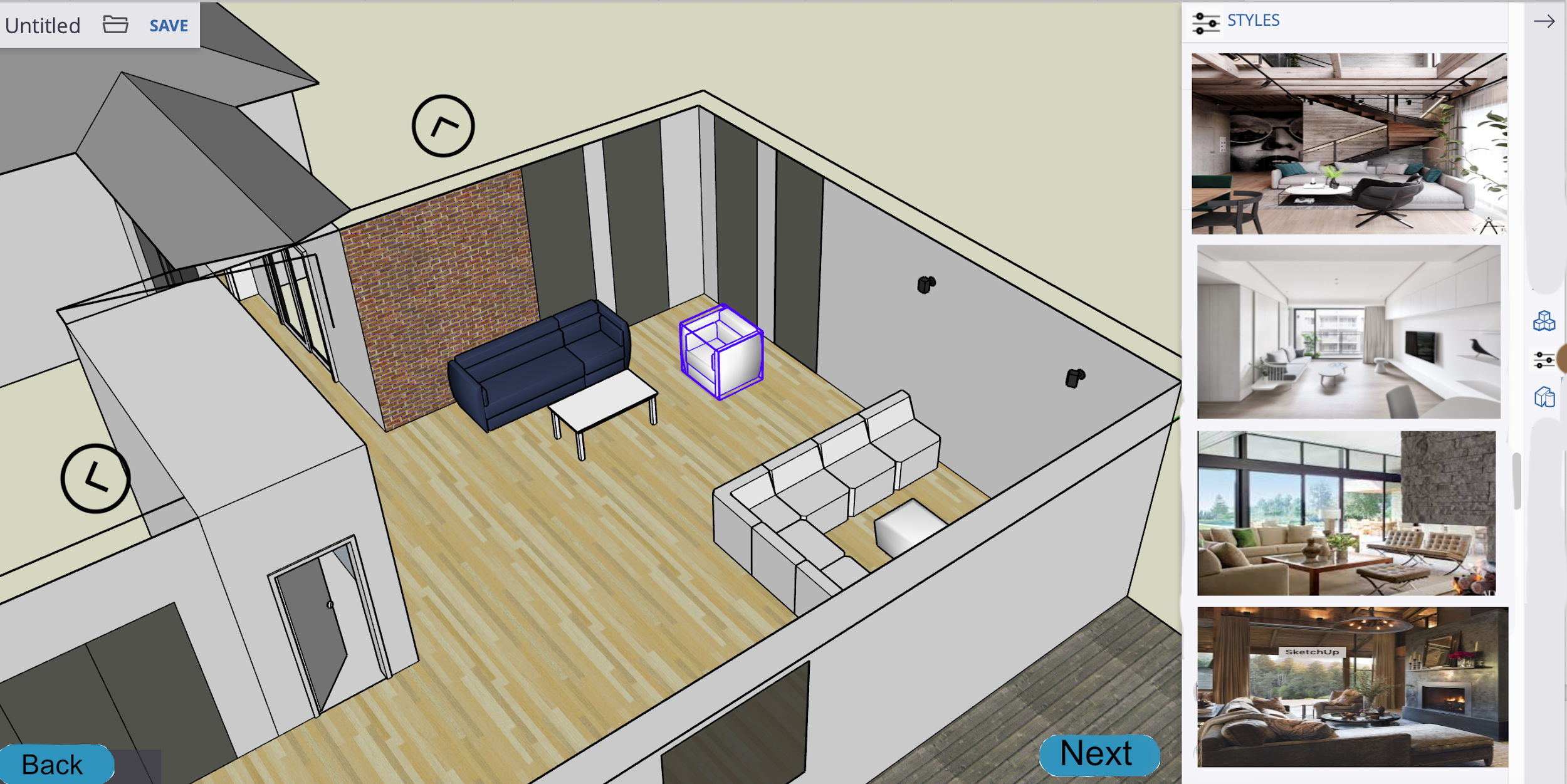


**Interior Design**

Pic 1) allows users to insert items recommended by the recommendation engine algo

Pic 2) lets user change rooms, select style for room and all objects in it

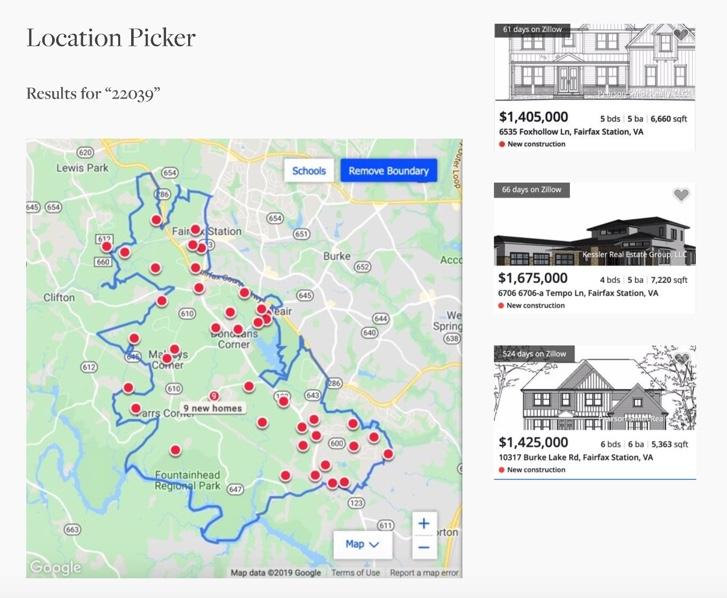
Pic 3) lets user select style, color, materials for specific item 





**Location Picker**

Pic 1)Main Location picker home screen: The main function that this page displays is the search bar in which a user can use to search for potential locations. This page of the application also contains a link to “bookmarked properties”, so that the user can easily access any properties that they have saved.



Pic 2) Location Picker after search: After the user searches for a location he or she is prompted with an outline of the searched area with properties highlighted within that location. Next to that outline are some of the properties. The user can choose to scroll through those properties or click directly on the map to view the exact spots that they are looking for.

**Stakeholder System**

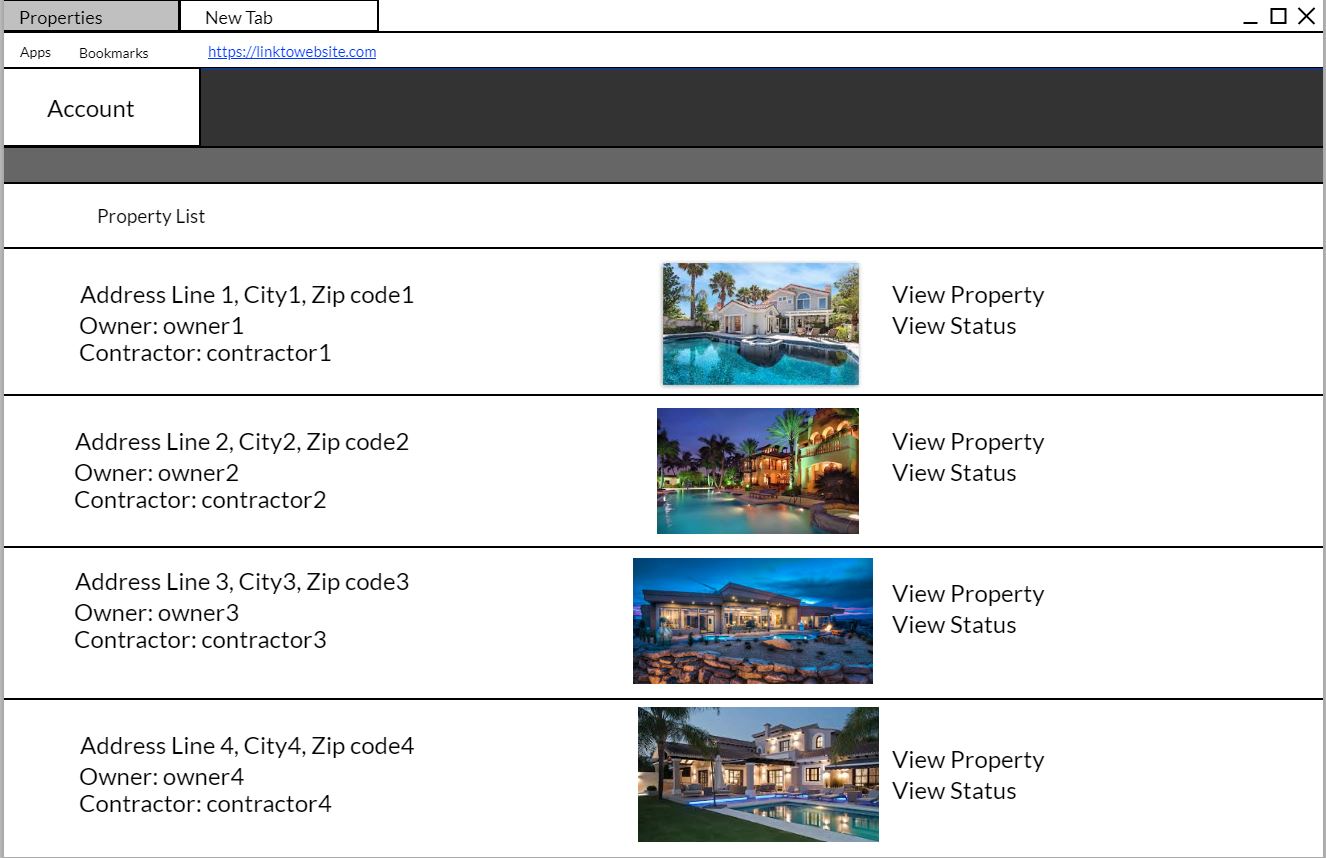
Pic 1) All stakeholders start at the login screen

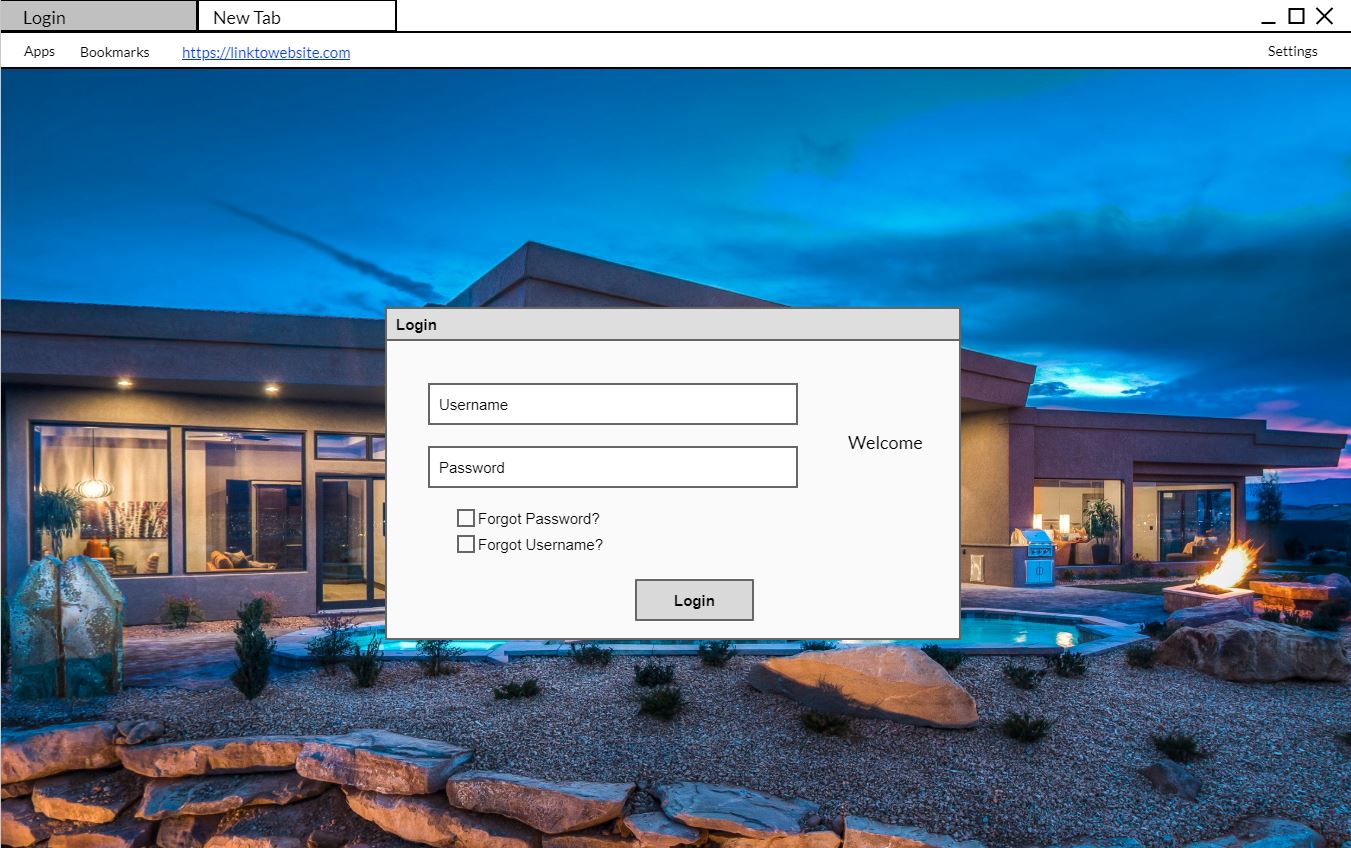
Pic 2) Contractors, Suppliers, Banks (financial institutions) will see all properties under construction

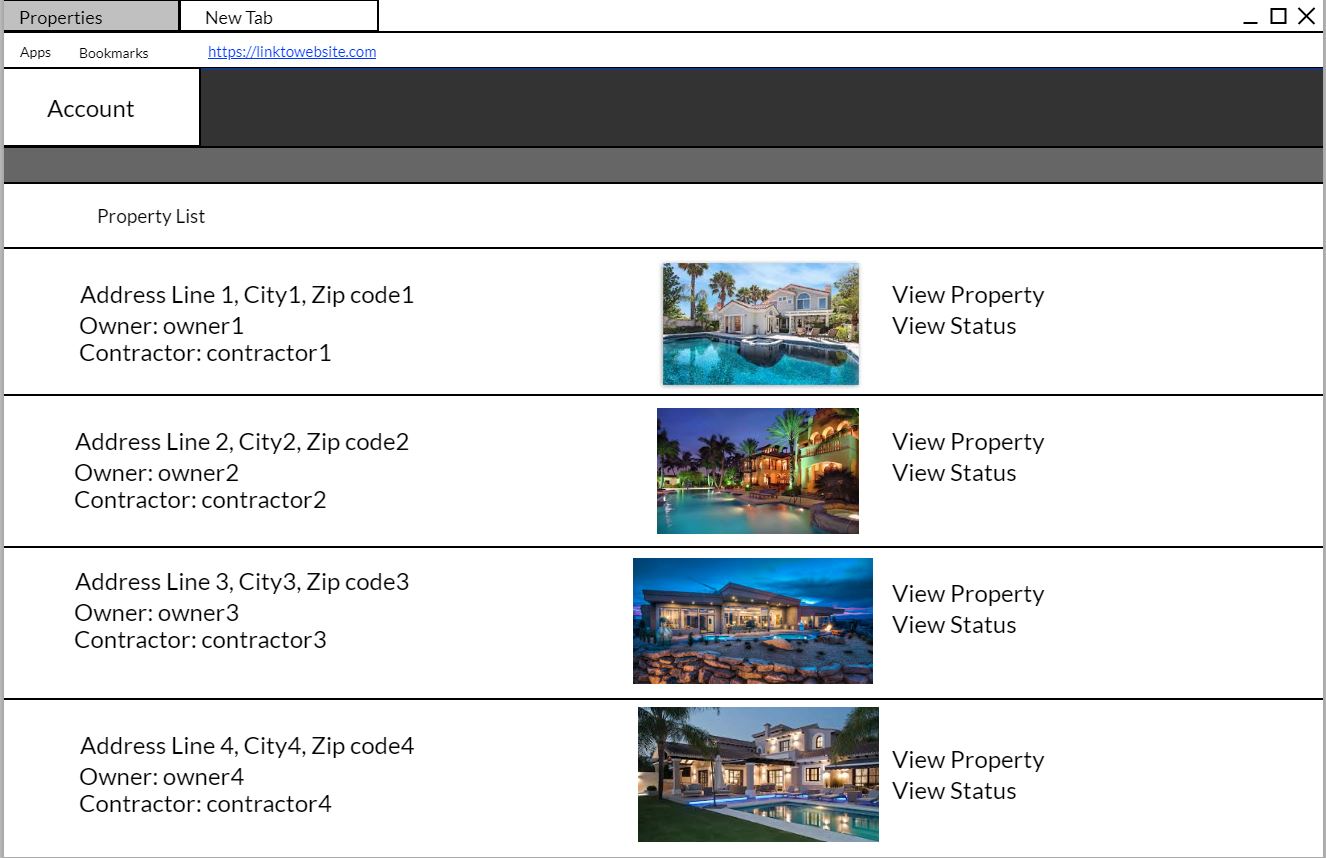
-> a homeowner would see their own

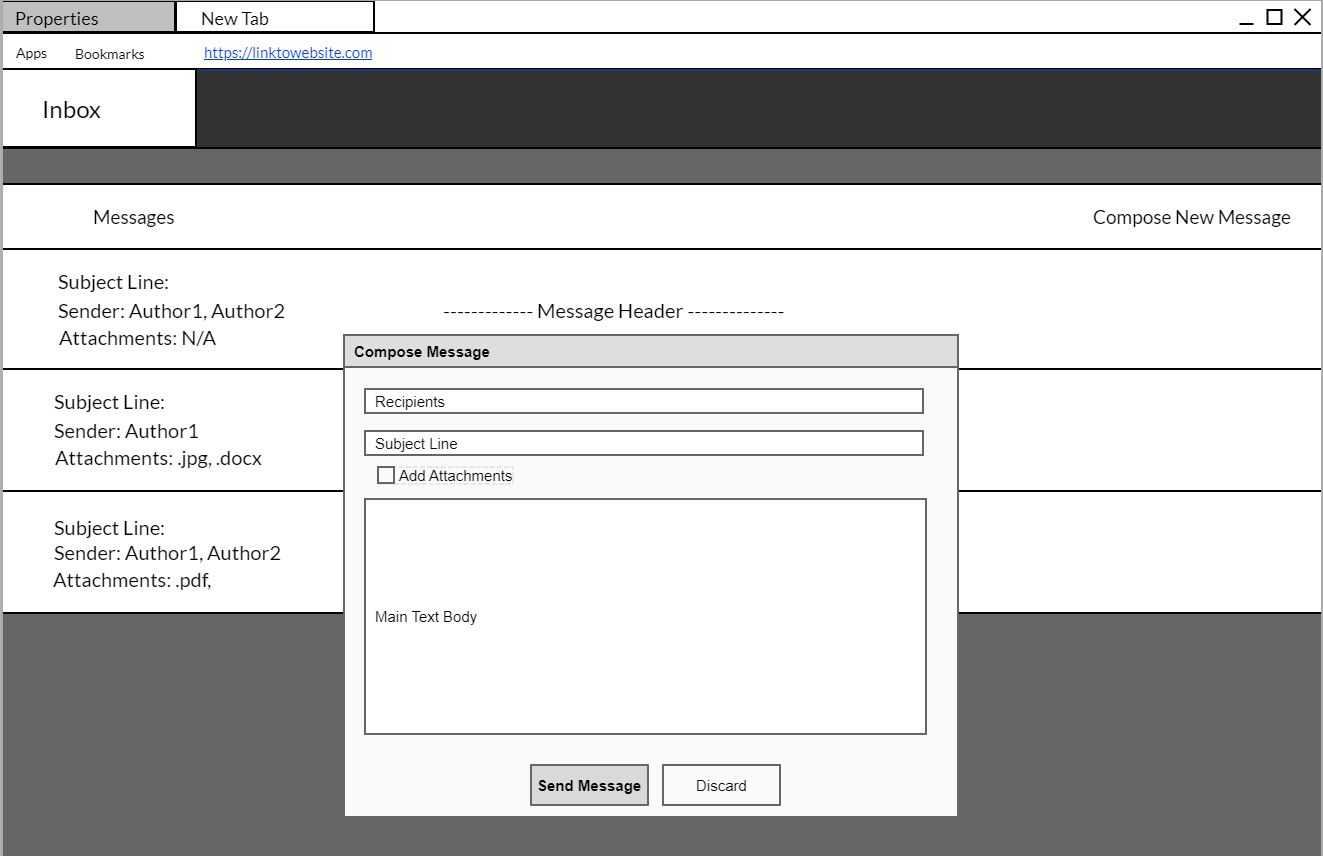
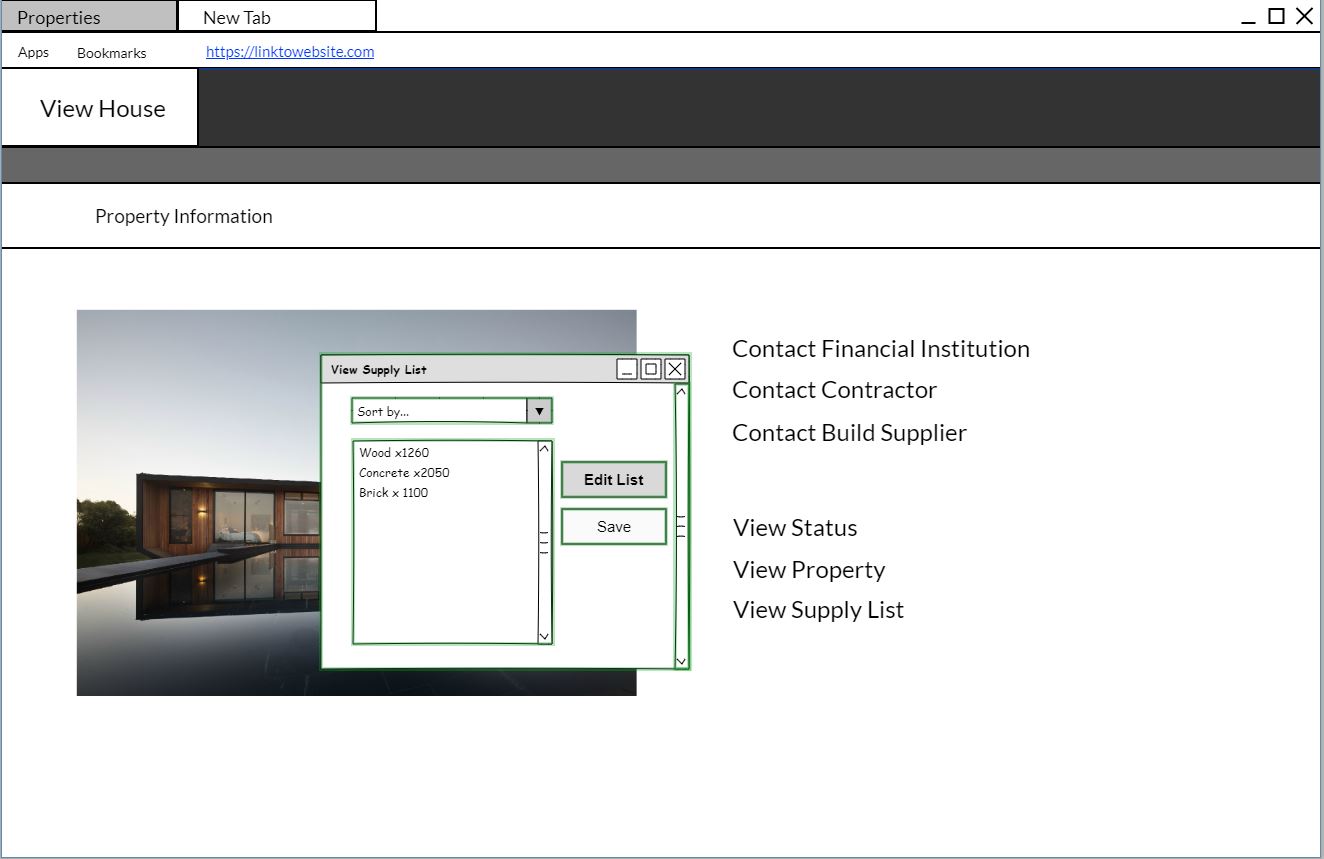
Pic 3) each stakeholder can contact one another via a messaging system in pic 4

Pic 4) the messaging system





****

****