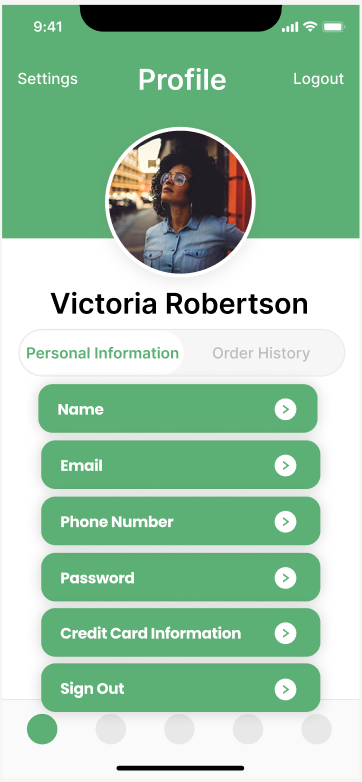
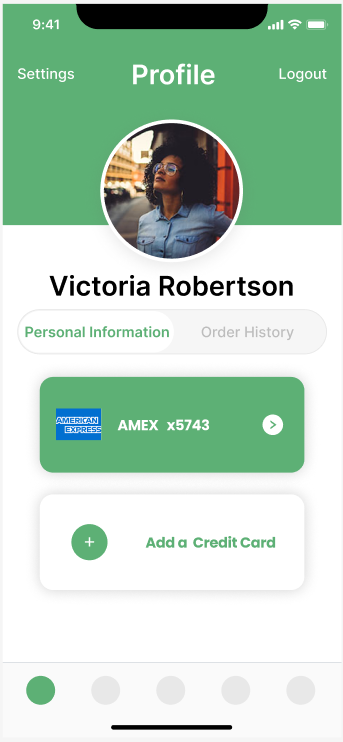
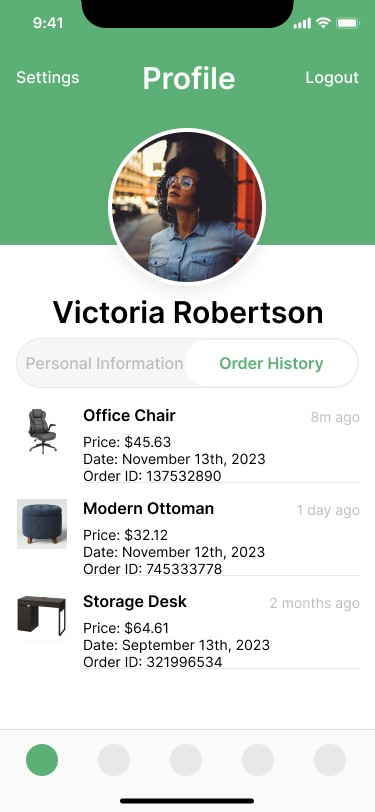
**Design II**

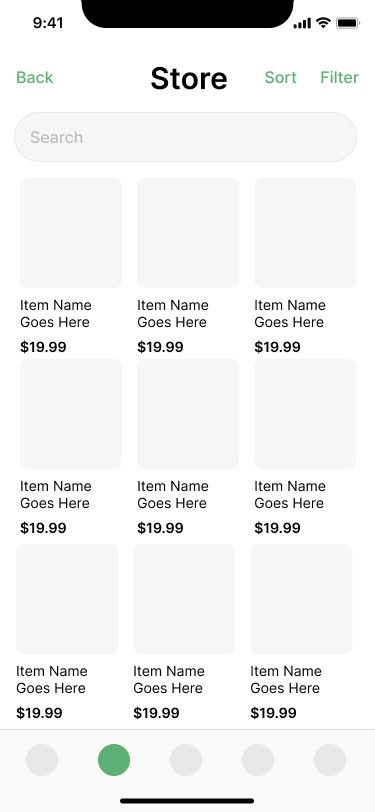
**UI Design:**

* Mock UI:

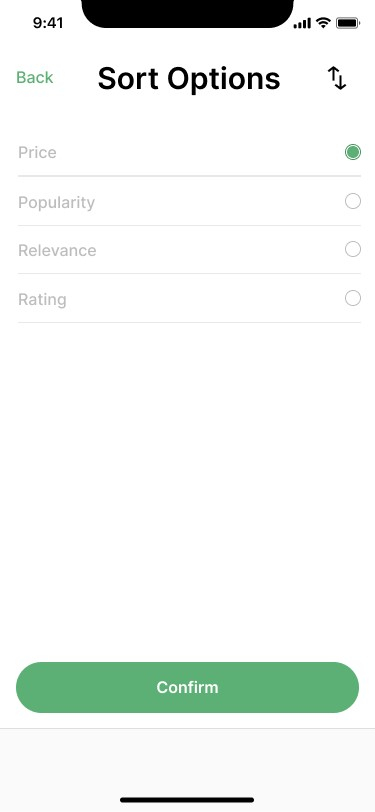
→ User Profile

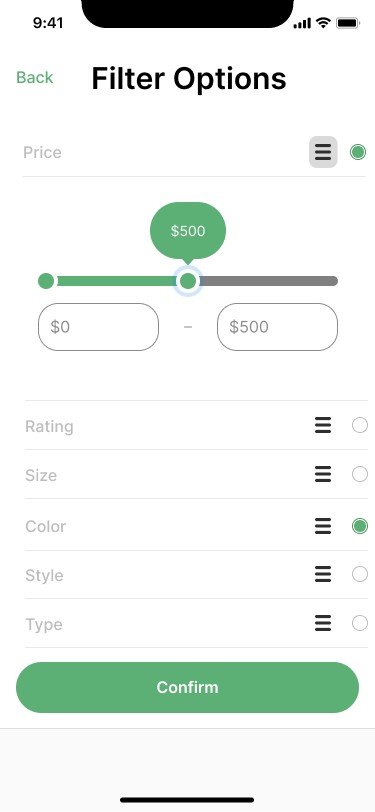
→ Product Search:



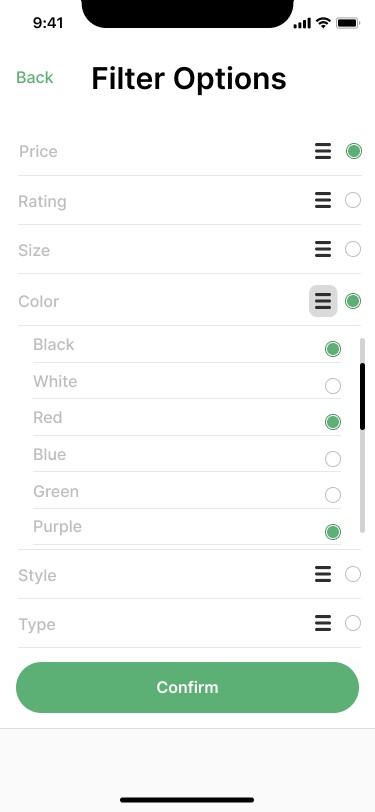
→ Sort Selection:



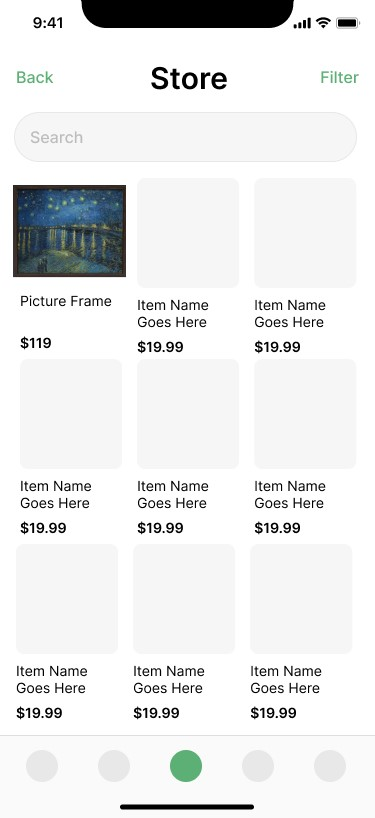
→ Price Selection:



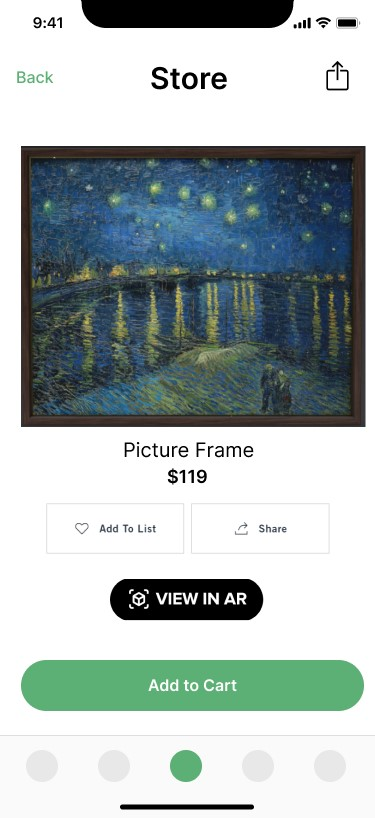
→ Filter Selection:



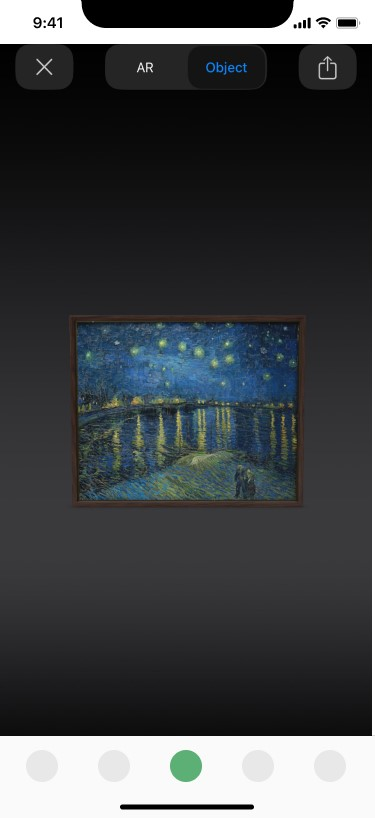
→ Catalog View:



→ Item View:



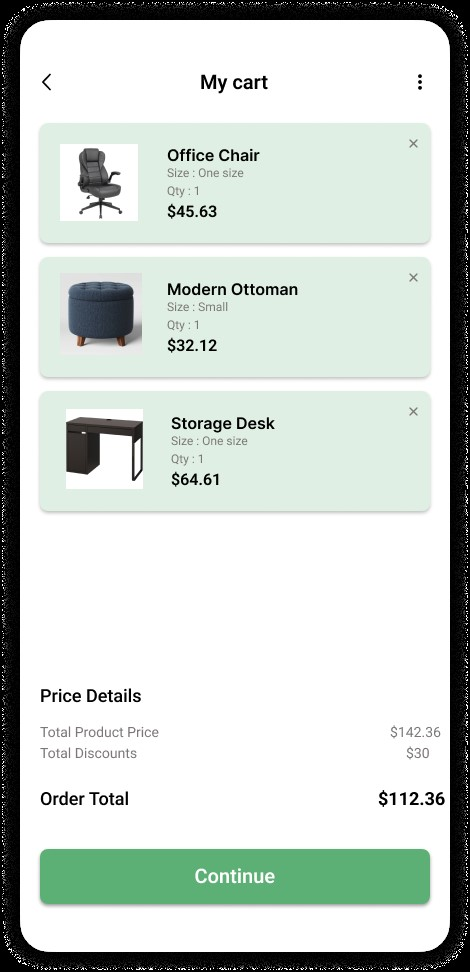
→ Object View:



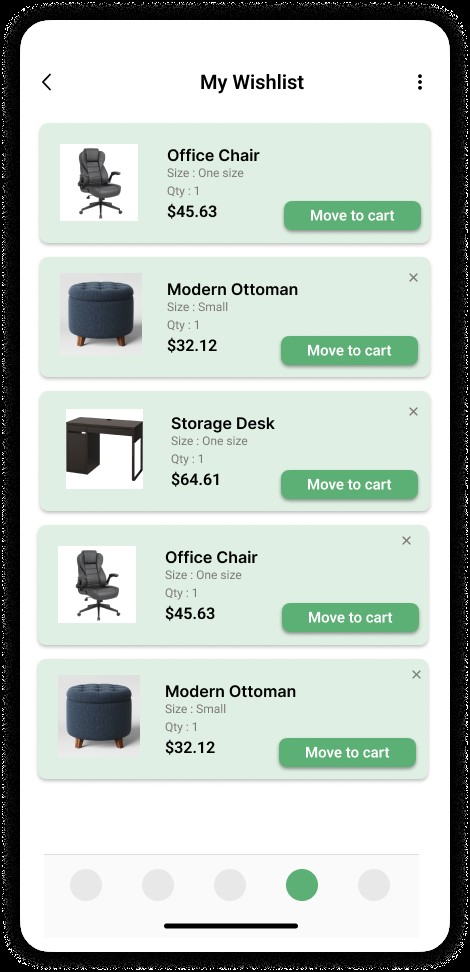
→ AR View:



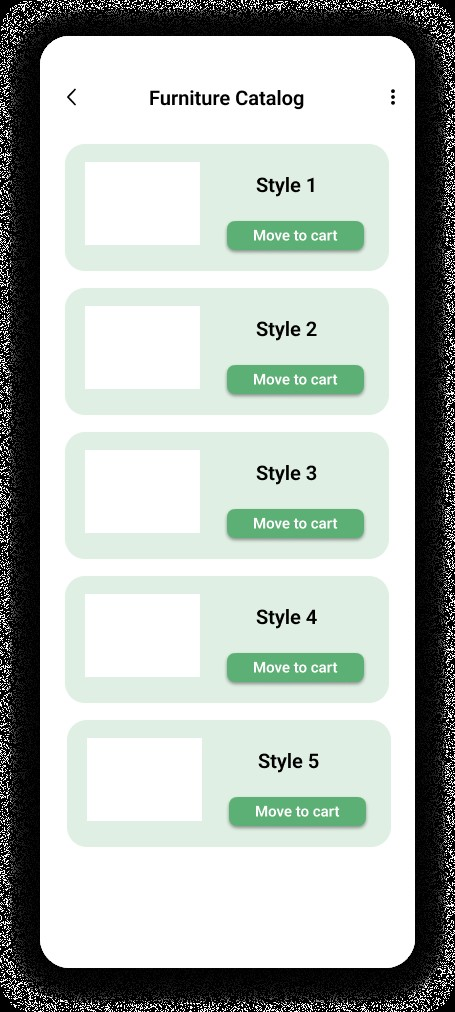
→ Cart:



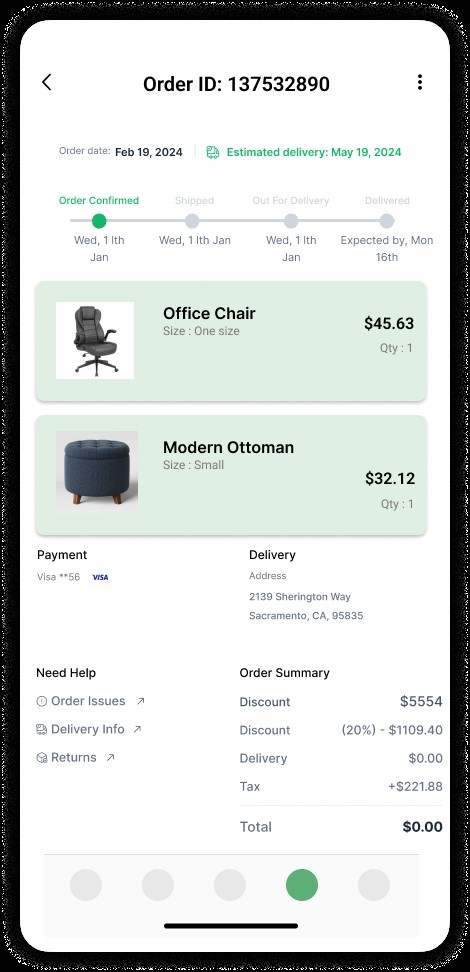
→ Wishlist:



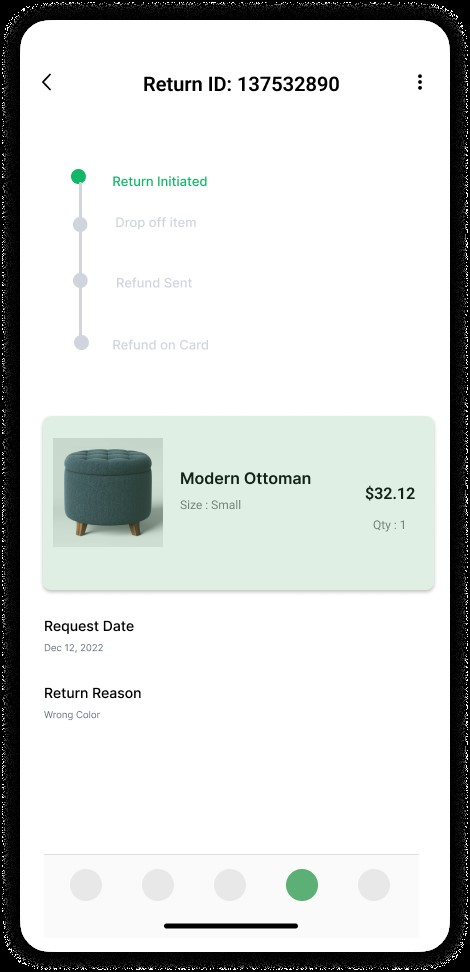
→ Furniture Catalog:



→ Tracking Order:



→ Return Order:



**Algorithm Design:**

* Pseudo-code:

User Class:

class User:

// Constructor to initialize the User object with default values or provided values

function \_\_init\_\_(self, personalInfo, paymentInfo, preferences, orderHistory, userID):

self.personalInfo = personalInfo

self.paymentInfo = paymentInfo

self.preferences = preferences

self.orderHistory = orderHistory

self.userID = userID

// Method to get the userID

function getUserID(self):

return self.userID

// Method to get the order history

function getOrderHistory(self):

return self.orderHistory

// Method to set a new userID

function setUserID(self, newID):

self.userID = newID

// Method to add a new preference

function addPreference(self, newPref):

self.preferences.append(newPref)

// Method to remove a preference

function removePreference(self, pref):

if pref in self.preferences:

self.preferences.remove(pref)

// Method to view payment information

function viewPaymentInfo(self):

return self.paymentInfo

// Method to view preferences

function viewPreferences(self):

return self.preferences

// Method to change payment information

function changePaymentInfo(self, newNum):

self.paymentInfo = newNum

Credit Card Class:

class CreditCard:

// Constructor to initialize the CreditCard object with provided values

function \_\_init\_\_(self, number, name, bank):

self.number = number

self.name = name

self.bank = bank

// Method to change the credit card number

function changeCreditCardNumber(self, newNum):

self.number = newNum

// Method to change the credit card name

function changeCreditCardName(self, newName):

self.name = newName

// Method to change the credit card bank

function changeCreditCardBank(self, newBank):

self.bank = newBank

FurnitureItem and ProductList Enumerations:

// Ways FurnitureItem objects may be sorted

public enum SortType {

PRICE,

POPULARITY,

RELEVANCE,

RATING,

DEFAULT

}

// Ways FurnitureItem objects may be filtered

public enum FilterType {

PRICE,

RATING,

SIZE,

COLOR,

STYLE,

TYPE,

DEFAULT

}

// Recognized FurnitureItem colors

public enum Color {

RED,

ORANGE,

YELLOW,

..., // additional colors here

DEFAULT

}

// Recognized FurnitureItem styles

public enum Style {

MODERN,

TRADITIONAL,

CONTEMPORARY,

..., // additional styles here

DEFAULT

}

// Recognized FurnitureItem types

public enum Type {

COUCH,

DRESSER,

TABLE,

..., // additional types here

DEFAULT

}

FurnitureItem Comparators for sorting:

// Compare two FurnitureItem objects based on price

class PriceComparator implements Comparator<FurnitureItem> {

public int compare(FurnitureItem a, FurnitureItem b) {

return a.getPrice() - b.getPrice();

}

}

// Compare two FurnitureItem objects based on popularity

class PopularityComparator implements Comparator<FurnitureItem> {

public int compare(FurnitureItem a, FurnitureItem b) {

return a.getPopularity() - b.getPopularity();

}

}

// Compare two FurnitureItem objects based on rating

class RatingComparator implements Comparator<FurnitureItem> {

public int compare(FurnitureItem a, FurnitureItem b) {

return a.getRating() - b.getRating();

}

}

// Compare two FurnitureItem objects based on relevance

class PopularityComparator implements Comparator<FurnitureItem> {

public int compare(FurnitureItem a, FurnitureItem b) {

int countA = 0, countB = 0;

// get # of matching tags for a

for (String tag : a.getTags()) {

if (tags.contains(tag)) countA++;

}

// get # of matching tags for b

for (String tag : b.getTags()) {

if (tags.contains(tag)) countB++;

}

return countA - countB;

}

}

ProductList Class (getters, setters, and update methods not shown):

// The ProductList class handles sorting and filtering the list of products

// passed by the catalog

class ProductList {

private List<FurnitureItem> catalogList;

private List<FurnitureItem> filteredResults;

private HashSet<FilterType> filters;

private SortType sortedBy;

private int productListID;

private HashSet<String> tags;

// Init new ProductList with the given list of furniture items and ID

ProductList(List<FurnitureItem> catalogList, int productListID) {

this.catalogList = catalogList;

this.productListID = productListID;

filteredResults = catalogList;

filters = new HashSet<FilterType>();

sortedBy = SortType.DEFAULT;

tags = new HashSet<String>();

}

// Sort the list using the currently set sorting method

public void sort(boolean ascending) {

if (sortedBy == SortType.DEFAULT) return;

// init Comparator for sorting

Comparator c = null;

// set Comparator based on sortBy

switch(sortBy) {

case SortType.PRICE:

c = new PriceComparator();

break;

case SortType.POPULARITY:

c = new PopularityComparator();

break;

case SortType.RATING:

c = new RatingComparator();

break;

// Relevance is default sort, intentionally falls through

case SortType.RELEVANCE:

default:

c = new RelevanceComparator();

break;

}

// flip comparator order if descending sort selected

if (!ascending) c = c.reverseOrder();

// sort filteredResults with the given Comparator

Collections.sort(filteredResults, c);

}

// Filter the list using the currently set filters

// Fields are only necessary if the corresponding filter is in the filters set

public void filter(

int minPrice,

int maxPrice,

double minRating,

double minLength,

double minWidth,

double minHeight,

double maxLength,

double maxWidth,

double maxHeight,

HashSet<Color> colors,

HashSet<Style> styles,

HashSet<Type> types

) {

// Empty filteredResults list or reset it to catalogList if filters set

// is empty

filteredResults = (filter.isEmpty()) ? catalogList : new List<FurnitureItem>();

// Check each item in catalogList against each filter in the filters set

// and add valid items to filteredResults

for (FurnitureItem item : filteredResults) {

for (FilterType filter : filters) {

// Check filter value

switch(filter) {

// Check price in range

case FilterType.PRICE:

double price = item.getPrice();

if (item.getPrice() != -1 // no price found

&& item.getPrice() <= maxPrice

&& item.getPrice() >= minPrice) {

filteredResults.add(item);

}

break;

// Check rating in range

case FilterType.RATING:

double rating = item.getRating();

if (item.getRating() != -1 && item.getRating() >= minRating) {

filteredResults.add(item);

}

break;

// Check size in range

case FilterType.SIZE:

if (validSize(item, minLength, minWidth, minHeight,

maxLength, maxWidth, maxHeight)) {

filteredResults.add(item);

}

break;

// Check valid color

case FilterType.COLOR:

// assignment for clarity

Color color = item.getColor();

if (colors.contains(color)) filteredResults.add(item);

break;

// Check valid style

case FilterType.STYLE:

// assignment for clarity

Style style = item.getStyle();

if (styles.contains(style)) filteredResults.add(item);

break;

// Check valid type

case FilterType.TYPE:

// assignment for clarity

Type type = item.getType();

if (type.contains(type)) filteredResults.add(type);

break;

default:

break;

}

}

}

}

// Helper function, check if a FurnitureItem is in the size range

private boolean validSize(

FurnitureItem item,

double minLength,

double minWidth,

double minHeight,

double maxLength,

double maxWidth,

double maxHeight

) {

// get item dimensions

double length = item.getLength();

double width = item.getWidth();

double height = item.getHeight();

// comparisons split for clarity

if (length < minLength || length > maxLength) return false;

if (height < minHeight || height > maxHeight) return false;

if (width < minWidth || width > maxWidth) return false;

// dimensions in range

return true;

}

}

Order Class

Class Order:

String date

double totalAmount

List<FurnitureItem> shoppingCart

List<FurnitureItem> wishlist

String orderDetails

Reciept receipt

// initialize Order

function \_\_init\_\_(date, totalAmount, shoppingCart, wishlist, orderDetails, reciept)

{

self.date=date

self.totalAmount = totalAmount

self.shoppingCart = shoppingCart

Self.wishlist = wishlist

self.orderDetails = orderDetails

Self.receipt = receipt

}

// adds the given item to either the shopping cart or wishlist

function addItem(String type, furnitureItem): void

{

If (type == “shopping cart”)

{

self.shoppingCart.append(furnitureItem)

}

If (type ==”wishlist”)

{

self.wishlist.append(furnitureItem)

}

}

// removes the given item from either the shopping cart or wishlist

function removeItem(furnitureItem): void

{

If (type == “shopping cart”)

{

self.shoppingCart.remove(furnitureItem)

}

If (type ==”wishlist”)

{

self.wishlist.remove(furnitureItem)

}

}

// updates the given item from the shopping cart or wishlist with the given update (color, size or amount) and what update needs to be made

function updateItem(furnitureItem item, String update, String updateSpec, String type) : void

{

if(type == “shopping cart”){

If (update == “color”){

item.setColor(updateSpec)

}

If (update == “size”){

item.setSize(updateSpec)

}

If (update == “amount”){

for (int i = 0; i <updateSpec.asInt(); i++){

self.addItem(furntureItem, “shopping cart”)

}

}

}

if(type == “shopping cart”){

If (update == “color”){

item.setColor(updateSpec)

}

If (update == “size”){

item.setSize(updateSpec)

}

If (update == “amount”){

for (int i = 0; i <updateSpec.asInt(); i++){

self.addItem(furntureItem, “wishlist”)

}

}

}

}

// updates order history with details and the receipt

function updateOrderHistory(orderDetails): void

{

orderDetails += self.getReciept(self.date, self.totalAmount, self.orderDetails)

}

// generates a receipt

function getReciept(date, totalAmount, orderDetails): Receipt

{

self.reciept.generateReciept(date, totalAmount. orderDetails)

}

Package Courier Class

Class PackageCourier:

String deliveryDate

String trackingNumber

Integer packageCourierID

String shipmentStatus

//initializes PackageCourier

Function \_\_init\_\_(deliveryDate, trackingNumber, packageCourierID, shipmentStatus)

{

self.deliveryDate = deliveryDate

self.trackingNumber = trackingNumber

self.packageCourierID = packageCourierID

self.shipmentStatus = shipmentStatus

}

Function getShipmentStatus():String

{

Return self.shipmentStatus

}

Function getDeliveryDate():String

{

Return self.deliveryDate

}

Function getTrackingNumber():String

{

Return self.trackingNumber

}

Receipt Class

Class Reciept:

String date

Double totalAmount

String orderDetails

Integer recieptID

// initializes receipt

Function \_\_init\_\_(date, totalAmont, orderDetails, recieptID)

{

self.date = date

self.totalAmount = totalAmount

self.orderDetails = orderDetails

self.recieptID = recieptID

}

// generates the receipt with date, amount and details

Function generateReciept(date, totalAmount, orderDetails):String

{

String receipt = date + “ “ + totalAmount + “ “ + orderDetails + “ “

Return receipt

}

//sends the receipt to the User

Function sendReciept(User): void

{

String reciept = generateReciept(self.date, self.totalAmount, self.orderDetails)

sendEmail(User, reciept)

}

FurnitureItem Class:  
class FurnitureItem:

// Attributes with default values or placeholders

color: String = ""

style: String = ""

size: String = ""

rating: Double = 0.0

popularity: Integer = 0

type: String = ""

keywords: List<String> = []

furnitureID: Integer = 0

arFurniture: ARObject = null

// Constructor to initialize a FurnitureItem

function \_\_init\_\_(color, style, size, rating, popularity, type, keywords, furnitureID, arFurniture):

self.color = color

self.style = style

self.size = size

self.rating = rating

self.popularity = popularity

self.type = type

self.keywords = keywords

self.furnitureID = furnitureID

self.arFurniture = arFurniture

// Method to return the color of the furniture

function getColor():

return self.color

// Method to set the color of the furniture

function setColor(newColor):

self.color = newColor

// Method to return the style of the furniture

function getStyle():

return self.style

// Method to set the style of the furniture

function setStyle(newStyle):

self.style = newStyle

// Method to return the size of the furniture

function getSize():

return self.size

// Method to set the size of the furniture

function setSize(newSize):

self.size = newSize

// Method to return the rating of the furniture

function getRating():

return self.rating

// Method to set the rating of the furniture

function setRating(newRating):

if newRating >= 0 and newRating <= 5:

self.rating = newRating

else:

// Handle invalid rating

// Method to return the popularity of the furniture

function getPopularity():

return self.popularity

// Method to set the popularity of the furniture

function setPopularity(newPopularity):

self.popularity = newPopularity

// Method to return the type of the furniture

function getType():

return self.type

// Method to set the type of the furniture

function setType(newType):

self.type = newType

// Method to return the keywords associated with the furniture

function getKeywords():

return self.keywords

// Method to set the keywords associated with the furniture

function setKeywords(newKeywords):

self.keywords = newKeywords

// Method to display the furniture in AR

function displayFurniture():

// Pseudo code to display the furniture using AR technology

// This could involve setting up an AR session, placing the AR object in the environment, etc.

// Method to capture the space for placing furniture in AR

function captureSpace():

// Pseudo code to capture the room using AR technology

// Typically involves accessing the device's camera and sensors to map out the space

UserDatabase Class:

class UserDatabase:

// A dictionary to store user personal information keyed by user IDs

userPersonalInfoList: Dict<String, Dict<String, String>> = {}

databaseID: Integer = 0

// Constructor to initialize the UserDatabase with an ID

function \_\_init\_\_(databaseID):

self.databaseID = databaseID

// Method to retrieve personal information of a user

function getUserPersonalInformation(userID):

if userID in self.userPersonalInfoList:

return self.userPersonalInfoList[userID]

else:

return {} // User not found

// Method to update personal information of a user

function setUserPersonalInformation(userID, newInfo):

if userID in self.userPersonalInfoList:

self.userPersonalInfoList[userID].update(newInfo)

else:

// Handle the case where the user ID does not exist

// Method to add a new user's information to the database

function addUserInfo(userID, userInfo):

if userID not in self.userPersonalInfoList:

self.userPersonalInfoList[userID] = userInfo

else:

// Handle the case where the user already exists

// Method to remove a user's information from the database

function removeUserInfo(userID):

if userID in self.userPersonalInfoList:

del self.userPersonalInfoList[userID]

else:

// Handle the case where the user does not exist

FurnitureCatalog Class

class FurnitureCatalog:

catalog = List<List<FurnitureItem>>()

shoppingCart = ShoppingCart()

function moveToCart(List<FurnitureItem> collection) {

shoppingCart.moveToCart(collection)

}

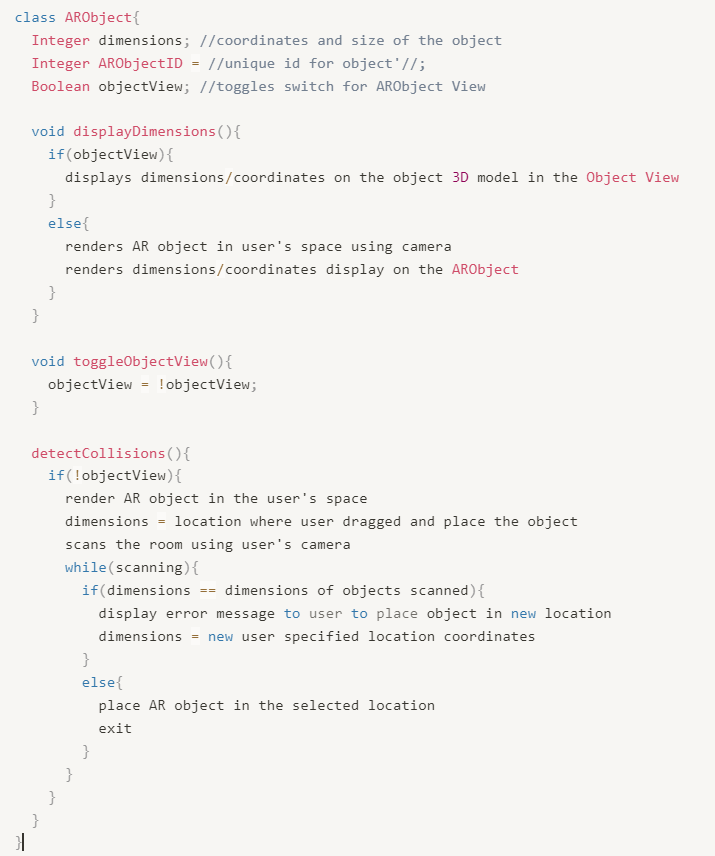
function filter(Predicate filterRequirements) {}

function removeFromCatalog(List<FurnitureItem> collection) {

catalog.remove(collection)

}

ARObject class



FurnitureAPIObject



Furniture Item (AR Methods)

