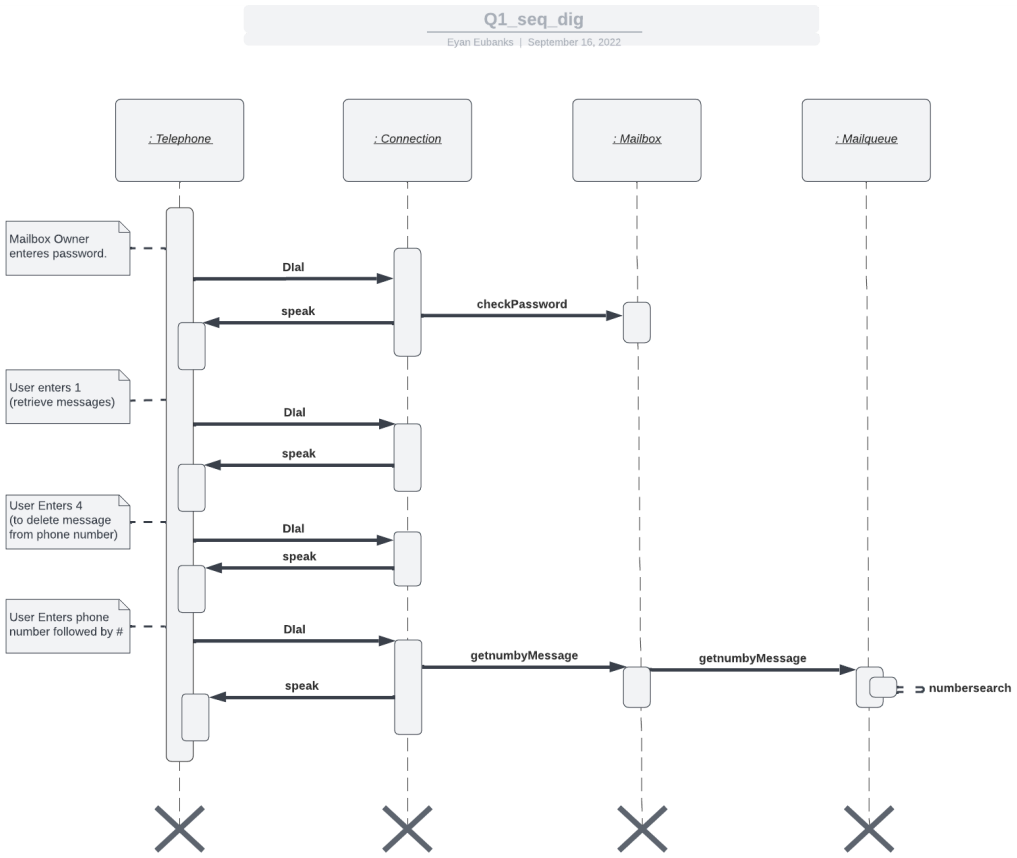
Eyan Eubanks COP4331 003

2.1 - a) Write the use case for this new feature. Add also one variant for an unsuccessful scenario. Make these scenarios integrate nicely with the other voice mail system use cases.

Use Case: erase message left by specific phone number

1) First User uses the “Log in” use case.
2) User Owner mailbox selects “retrieve your messages” menu option.
3) The Voice mail System plays the message menu:
   Enter 1 to listen to the current message.
   Enter 2 to save the current message.
   Enter 3 to delete the current message.
   (new added menu option) Enter 4 to delete messages from a certain phone number.
   Enter 5 to return to the mailbox menu.
4) The user selects “delete messages from a certain phone number”.
5) The Voice mail System asks the user to enter the number that they would like to have messages deleted from.
6) The user enteres the number followed by the “#” key.
7) The Mailbox tells the messageQueue to delete any messages with the caller ID that the User entered before the “#”
8) The Voice mail System replies “messages from (Users entered number) are deleted”
9) The Voice mail System returns to menu options selection



Variant #1)

1.1) In step 5 The user enters the a phone number that does not exist and hits “#”.

1.2) The voice mail system speaks and says “This phone number does not exist in the mailbox.”

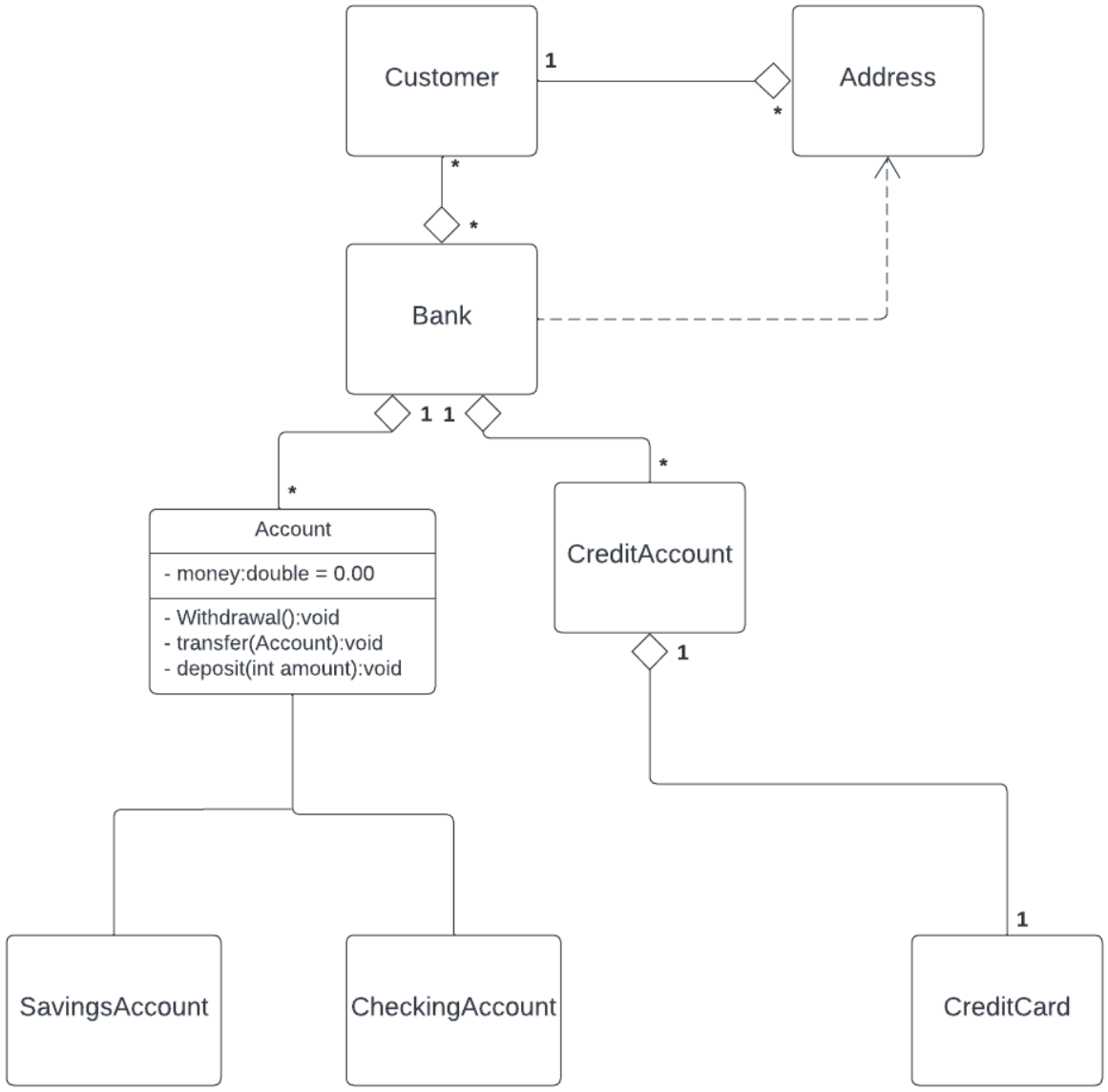
1.3) Continue from step 5.

2.2) An online bank application provides customers with three types of accounts: checking, savings, and credit. The credit account is tied to the credit card issued by the bank. The bank supports the following transactions involving accounts belonging to the same customer: withdrawal, transfer, deposit. A transfer transaction involves 2 accounts. A customer has an address.

Draw a UML class diagram that shows the relations between these classes:

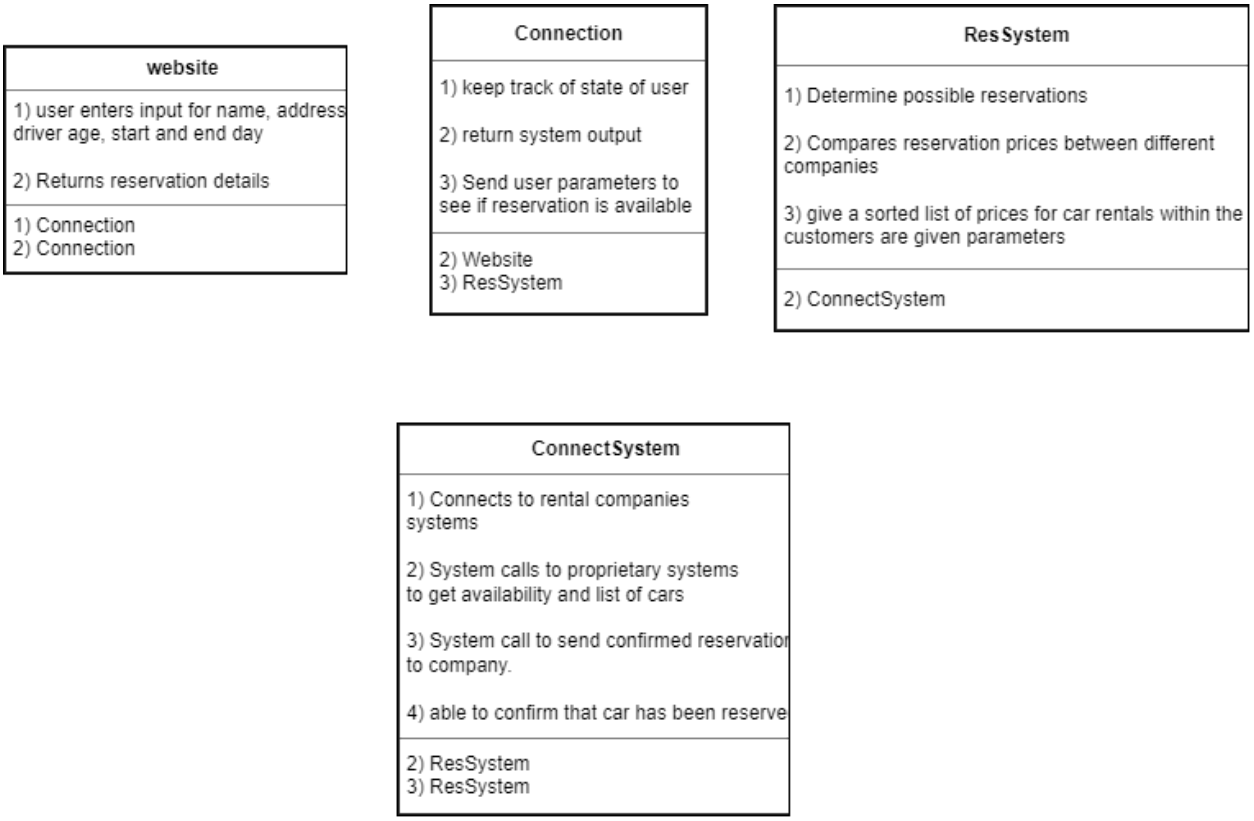
Account
CreditAccount
SavingsAccount
CheckingAccount
Bank
Address

CreditCard
Customer

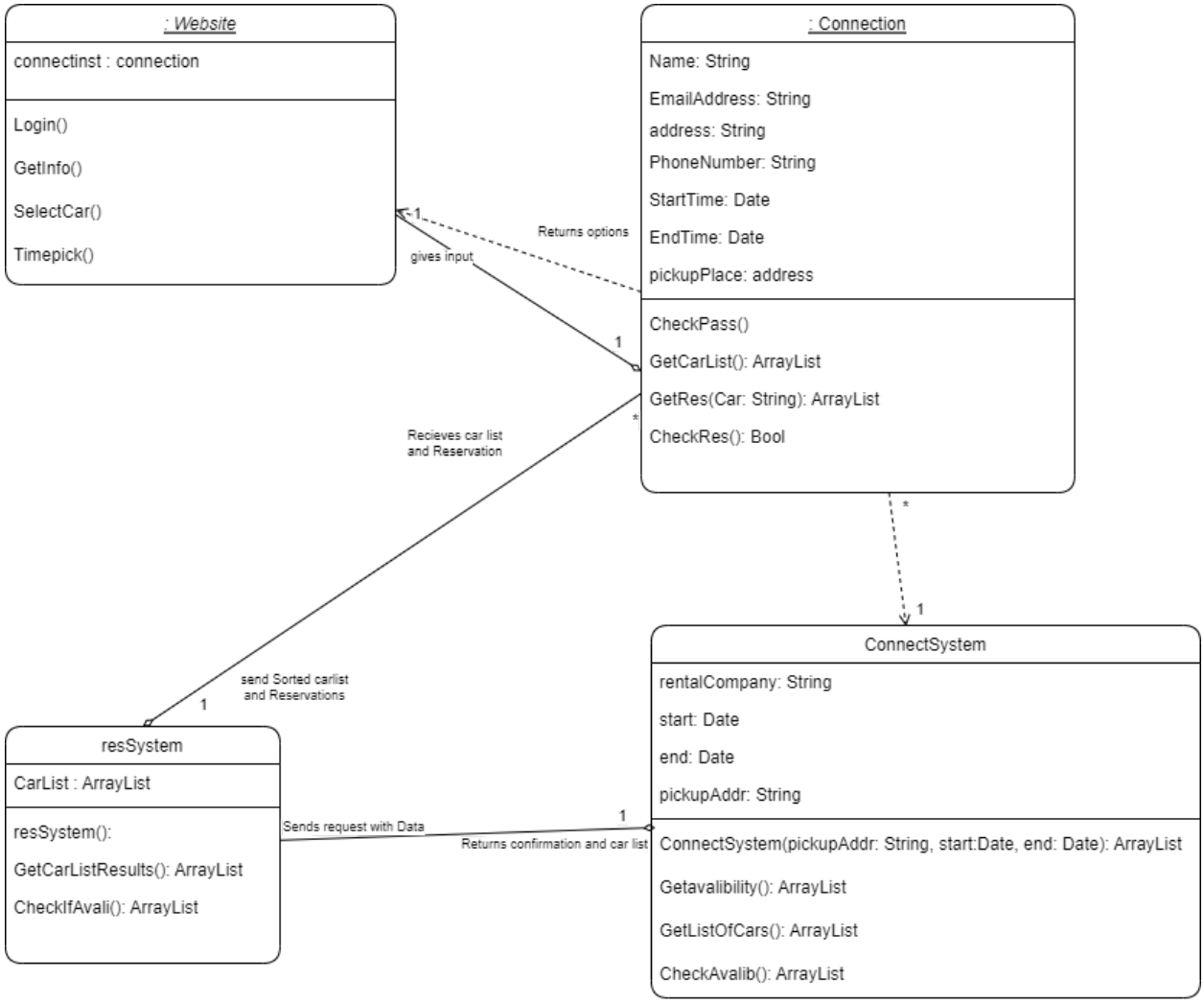


2.3)

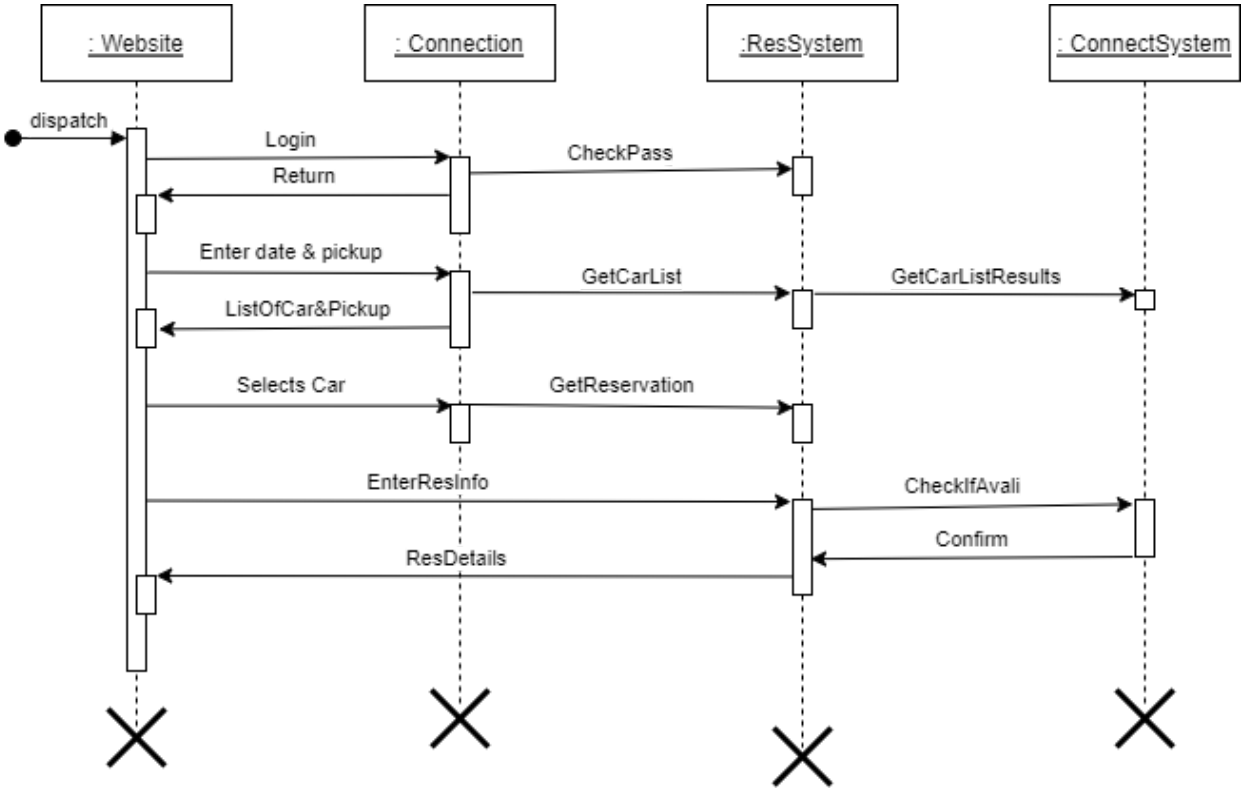
a)


website
1) user enters input for name, address driver age, start and end day
2) Returns reservation details
1) Connection
2) Connection
Connection
1) keep track of state of user
2) return system output
3) Send user parameters to see if reservation is available
2) Website
3) ResSystem
ResSystem
1) Determine possible reservations
2) Compares reservation prices between different companies
3) give a sorted list of prices for car rentals within the customers are given parameters
2) ConnectSystem
ConnectSystem
1) Connects to rental companies systems
2) System calls to proprietary systems to get availability and list of cars
3) System call to send confirmed reservatio to company.
4) able to confirm that car has been reserve
2) ResSystem
3) ResSystem

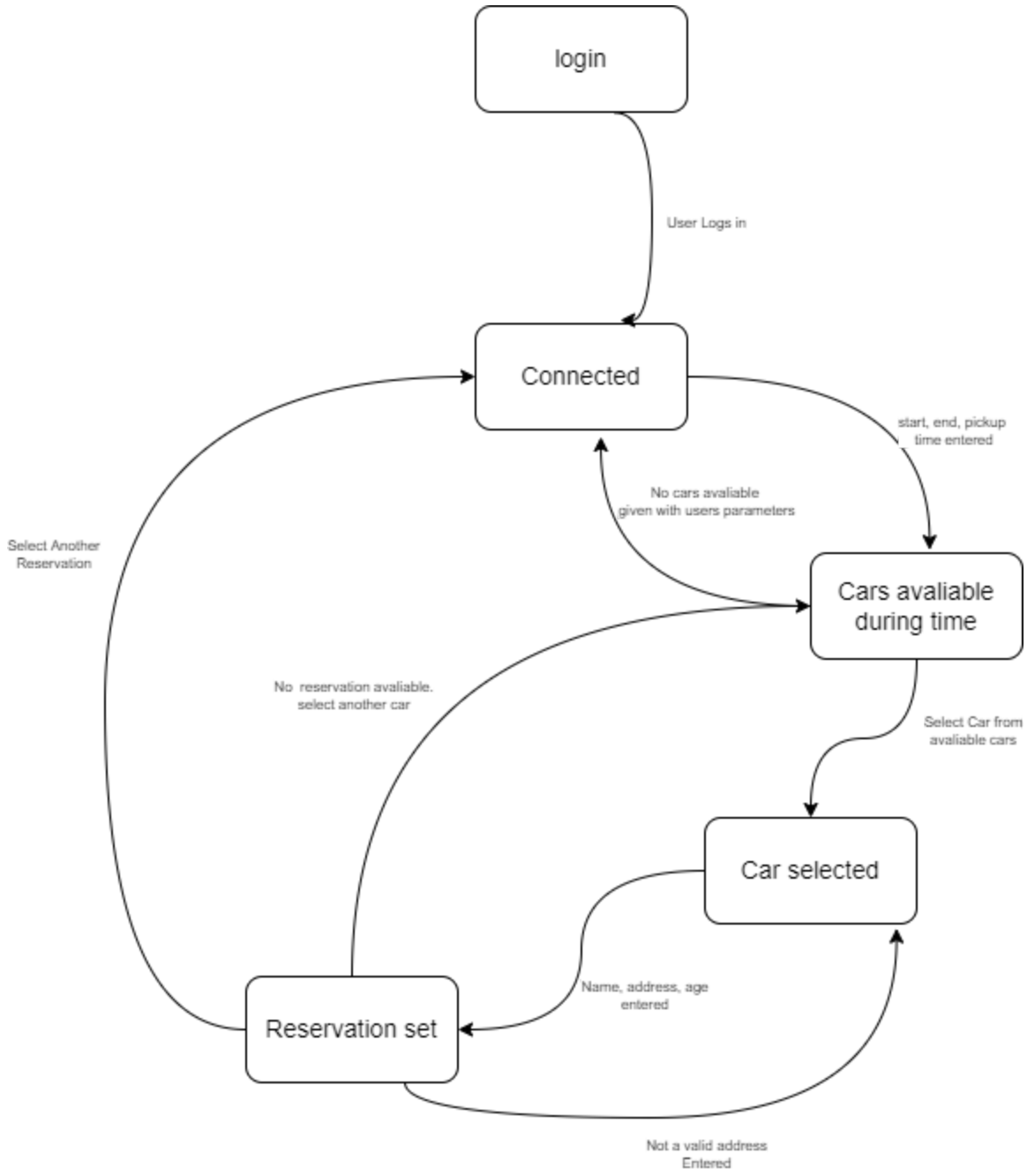
b)


: Website
connectinst : connection
Login()
GetInfo()
SelectCar()
Timepick()
: Connection
Name: String
EmailAddress: String
address: String
PhoneNumber: String
StartTime: Date
EndTime: Date
pickupPlace: address
CheckPass()
GetCarList(): ArrayList
GetRes(Car: String): ArrayList
CheckRes(): Bool
1
Returns options
gives input
1
1
*
Recieves car list
and Reservation
*
1
ConnectSystem
rentalCompany: String
start: Date
end: Date
pickupAddr: String
ConnectSystem(pickupAddr: String, start:Date, end: Date): ArrayList
Getavalibility(): ArrayList
GetListOfCars(): ArrayList
CheckAvalib(): ArrayList
send Sorted carlist
and Reservations
1
resSystem
CarList : ArrayList
resSystem():
GetCarListResults(): ArrayList
CheckIfAvali(): ArrayList
Sends request with Data
1
Returns confirmation and car list

c)


: Website
: Connection
:ResSystem
: ConnectSystem
dispatch
Login
Return
CheckPass
Enter date & pickup
ListOfCar&Pickup
GetCarList
GetCarListResults
Selects Car
GetReservation
EnterResInfo
ResDetails
CheckIfAvali
Confirm

d)



2.4)

CRC Diagram

UPCScanner:

- Scan's Customers Items — Register

Register:

- Handles user input — UPCScanner
- Handles User money — UPCScanner
- Handles User confirmation of last item — UPCScanner

- Confirm item is scanned StoreSystem

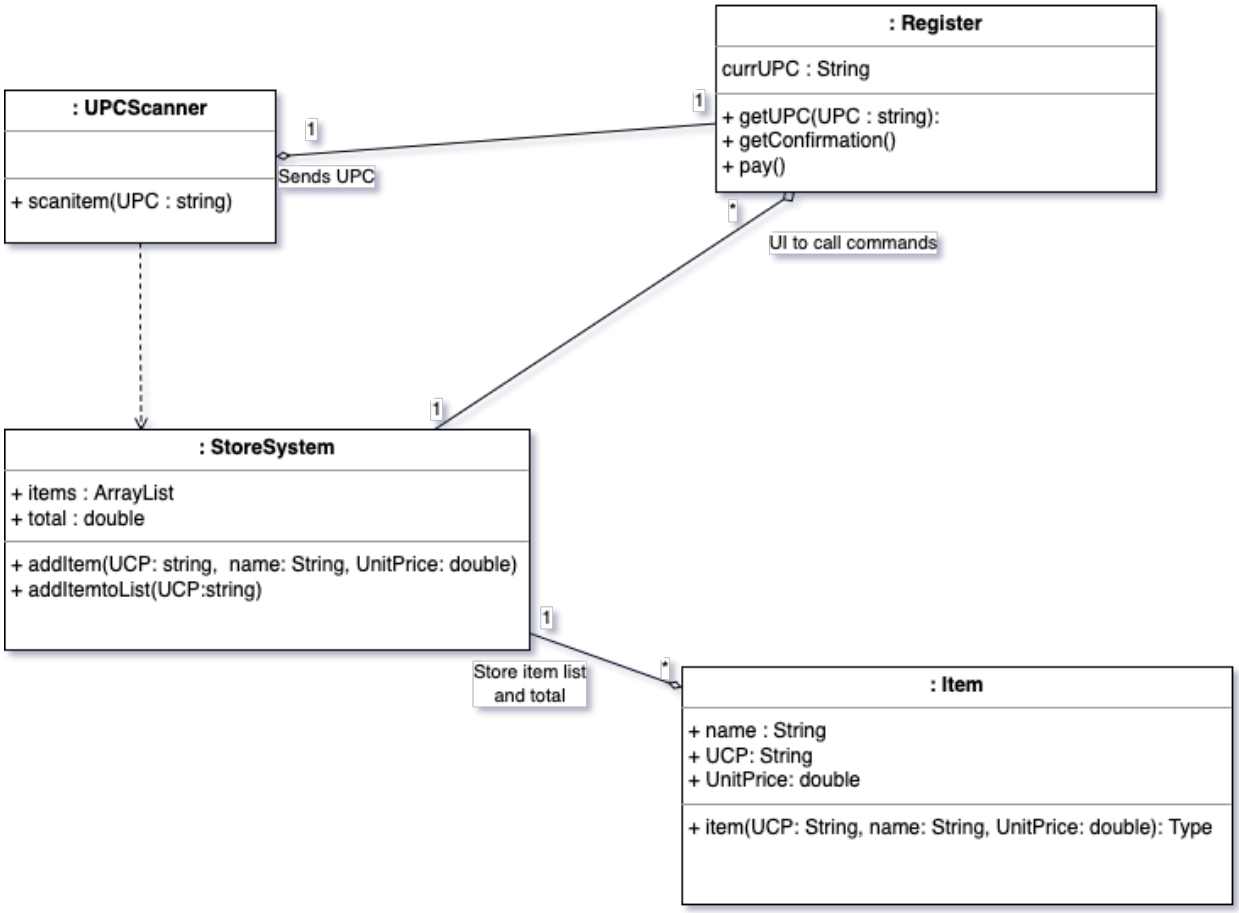
StoreSystem:

- Gets item Unit Price, item name, UPC items
- Stores list of items scanned
- Return list of items when user finished scanning Register

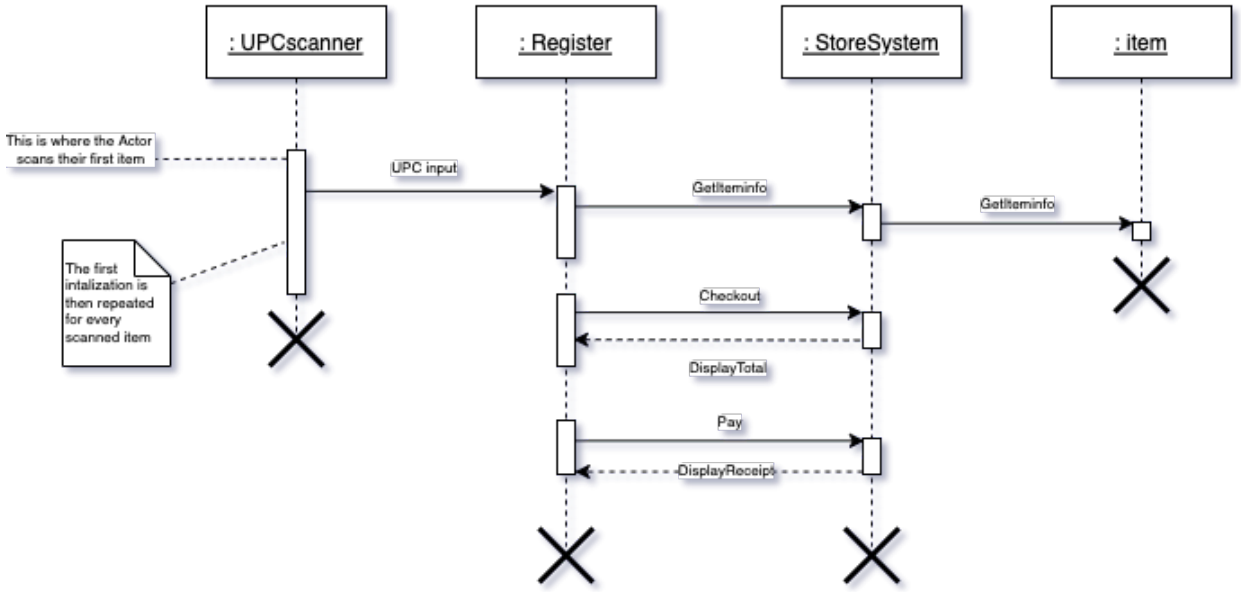
item

- Stores item attributes, item name, UPC, unit price

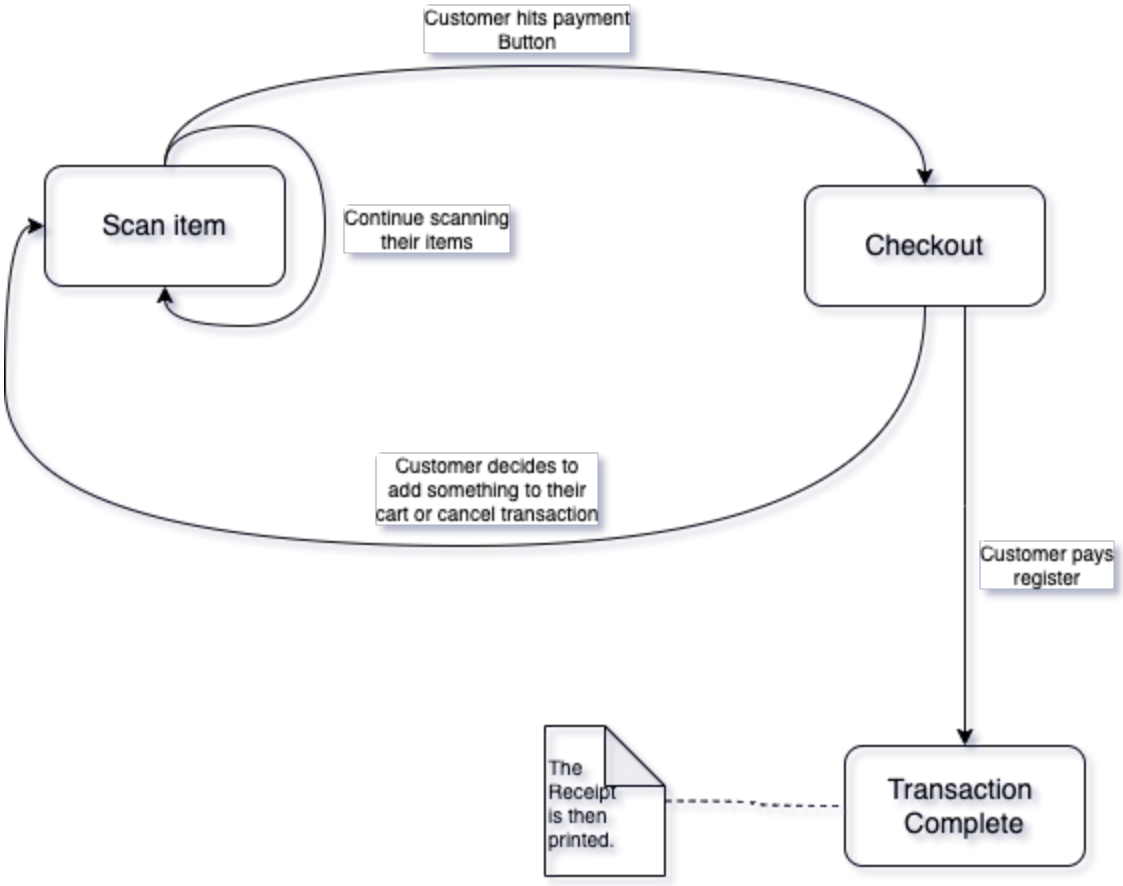
2.4) Class_Diagram



2.4) Sequence_Diagram



2.4) State_Diagram



Code implementation
CPCscanner.java

```
package pack;
import java.util.Scanner;


public class UPCscanner{

   public UPCscanner(){
       System.out.println("You can now scan your items, please enter a UPC
number.");
   }

   /**
    * This class takes user input to simulate scanning an item
    * @param upc
    */
   public String scanitem(){
```

```
        Scanner in = new Scanner(System.in);
        String upc = in.nextLine();
        return upc;
    }
}
```

Register.java

```
import pack.StoreSystem;
import pack.UPCscanner;

public class Register{

    /**
     * This initalizes register. Creates a UPCscanner that is running
throughout the life of the register
     */
    public Register(){
        System.out.println("Please Enter q or Q when you want to checkout");
        initscanner = new UPCscanner();  // creates a new scanner object
        initStoreSystem = new StoreSystem();
        checkout = false;
    }

    /**
    * This is the entry point into the program.
    */
    public static void main(String[] args){
        Register initReg = new Register();
       // This is pre-entered data, enter the first 12-digit number for UPC.
        initReg.initStoreSystem.additem("123123122341","Pickles","1.30");
        initReg.initStoreSystem.additem("902349232312","Pizza","8.75");
        initReg.initStoreSystem.additem("809234231231","Chicken
Breast","10.45");
        initReg.initStoreSystem.additem("908221423423","Bread","3.47");
        initReg.initStoreSystem.additem("678121332191","eggs","4.37");
        initReg.initStoreSystem.additem("234812903231","icecream","4.65");
        initReg.initStoreSystem.additem("798123361235","milk","3.97");

        while(!initReg.checkout) {


            String curr = initReg.initscanner.scanitem();

            if (!initReg.initStoreSystem.additemtoList(curr)){    // Adds item
to customers cart
                initReg.pay();
                continue;
            }
            initReg.print_UPC_name_unitPrice(initReg.getUPC(curr)); // prints
item name, upc, and unit price after each scan
```

```
        }
        initReg.Displaytotal();

    }

    /**
     * gets calls StoreSystem to get info about item with UPC number
     */
    public String[] getUPC(String upc){
        return initStoreSystem.getUPC(upc);
    }

    /**
    * Displays all items purchased with upc, name, unitprice, and total
     */
    public void Displaytotal(){

        System.out.println("Thank you for paying, here is your receipt");
        String[] tempitem = new String[3];
        for(int i = 0; i < this.initStoreSystem.items.size(); i++){
            tempitem[1] = this.initStoreSystem.items.get(i).upc;
            tempitem[0] = this.initStoreSystem.items.get(i).name;
            tempitem[2] = this.initStoreSystem.items.get(i).unitPrice;
            print_UPC_name_unitPrice(tempitem);
        }
        System.out.println("The total of your cart is: " +
this.initStoreSystem.total);
    }

    /**
     * Is called when Customer is ready to pay for their total
     */
    public void pay(){
        checkout = true;    // sets checkout equal to true
    }

    /**
     * Prints item info back to console/register responses to customer
     * @param iteminfo
     */
    public void print_UPC_name_unitPrice(String[] iteminfo){

System.out.println("**********************************************************
*******");
        System.out.println("item name: " + iteminfo[0] + "\nitem UPC: " +
iteminfo[1] + "\nitem Unit Price: " + iteminfo[2]);
```

```
System.out.println("**********************************************************
*******");
    }

    public UPCscanner initscanner;  // stores UPC scanner object
    public StoreSystem initStoreSystem; // stores StoreSystem
    boolean checkout;               // changes state of register so customer
can checkout

}
```

StoreSystem.java

```
package pack;

import java.util.ArrayList;
import pack.item;

public class StoreSystem {

    /**
     * Is initalized when Register is initalized
     * an empty ArrayList is initalized, and total of cart is set to 0.0
     */
    public StoreSystem(){
        items = new ArrayList<item>();
        itemDB = new ArrayList<item>();
        total = 0.0;
    }

    /**
     * add an item with the UPC into the array list
     * @param upc
     * @param name
     * @param unitPrice
     */
    public void additem(String upc, String name, String unitPrice){
        item tempitem = new item(upc,name,unitPrice);
        itemDB.add(tempitem);
    }

    /**
     * Add an item to the list current customers list. If the item is not in
the itemDB, then @return return false
     * else if the item is in the itemDB, then return true
     * @param upc
     */
    public Boolean additemtoList(String upc){
        if(upc == "q" || upc == "Q"){
            return false;
```

```
        }
        for(int i = 0; i < itemDB.size(); i++){
            if(upc.equals(itemDB.get(i).upc)){  // if given upc is equal to any
items.upc in itemDB then return true
                items.add(new
item(itemDB.get(i).upc,itemDB.get(i).name,itemDB.get(i).unitPrice));
                total += Double.parseDouble(itemDB.get(i).unitPrice);    // then
add the unit price to the total
                return true;
            }
        }
        return false;
    }

    /**
     * extends Register and gets unitprice, name and upc
     */
    public String[] getUPC(String upc){
        String[] iteminfo = new String[3];

        for(int i = 0; i < itemDB.size(); i++){
            if(upc.equals(itemDB.get(i).upc)){
                iteminfo[0] = itemDB.get(i).name;
                iteminfo[1] = itemDB.get(i).upc;
                iteminfo[2] = itemDB.get(i).unitPrice;

                return iteminfo;
            }
        }
        return new String[]{"0","0","0"};   // if item is not found, then return
array [0,0,0]
    }

    public ArrayList<item> items;
    public ArrayList<item> itemDB;
    public double total;
}
```

Item.java

```
package pack;

public class item {
    public item(String upc, String name, String unitPrice){
        this.upc = upc;
        this.unitPrice = unitPrice;
        this.name = name;
    }

    public String name;
    public String upc;
```

```
    public String unitPrice;
}
```

2.5

a)

Use Case: Read Recent Post

1) User “X” enters Username and password onto beginning webpage.
2) Before user X is redirected to page, the social media system does a check through all of the users friends. It then creates a list of the most recent post their friends made.
3) User is then redirected to a page where all the users “friends” most recent post are. This list is sorted with most recent at the top and the user X can scroll down.

User Case: Edit Profile

1) Do steps 1-3 in use case “Read Recent Post”
2) The user can then click their user icon in the top right. This will bring down a menu:
    a) View Profile
    b) Block Users
    c) Edit Profile
    d) Log out
3) The user then picks c) Edit Profile.
4) The user can then edit their name, education, work history, and the list of schools they have been to.
5) The user then presses “save changes” button

User Case: Write Post

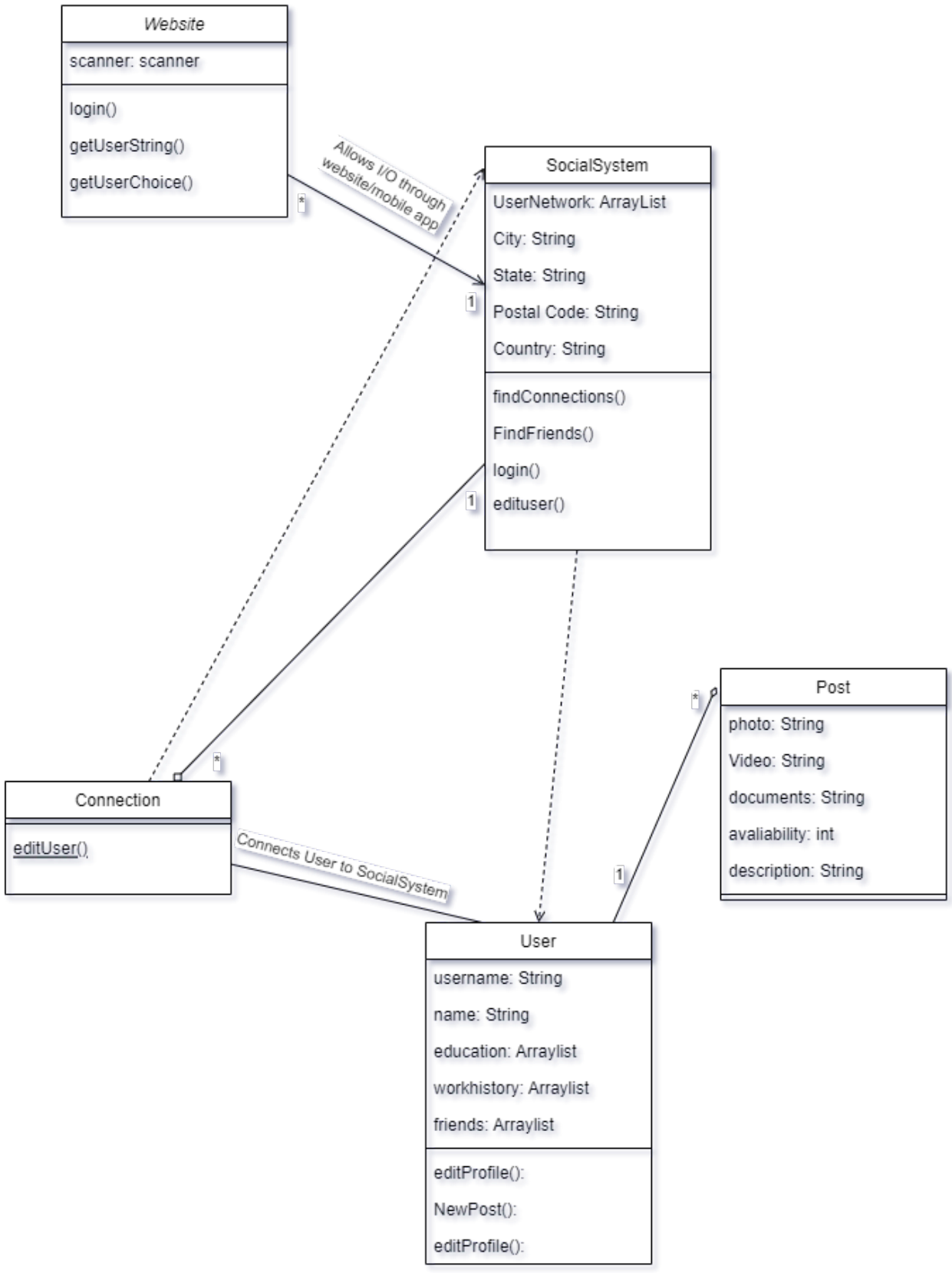
1) Do steps 1-2 in case “Read Recent Post”
2) User then looks at the bottom tab on the website.
    a) My friends
    b) Post
    c) Notifications
    d) Jobs
3) The user then clicks/selects b) “post”.
4) The user then is redirected to another page that allows them to write what they want in a textbox. The user can also add several attributes to the post. These attributes could each be a variant case.
    a) Add a photo
    b) Take a video
    c) Add a document
    d) Create an event
    e) Create a poll
    f) Post availability:
        i) Anyone
        ii) Friends only
5) User types in what they want in their post with the selected attributes. The user then hits the post button in the top right of the screen.

b)

Class	Responsibility	Relyon
website	take input Creates a connection with SocialSystem	SocialSystem
SocialSystem:(contains)----------<Users>	attributes:UsersNetwork:(contains)—————<>Users Contains Users network, finds connections between users, finds friends to show post Determines if users login	Users Connection
connection:	Creates a connection between SocialSystem and user Calls saveuser edits when user edits profile	SocialSystem Users
Users:(contains)----------<>Post	attributes: name, education, work history, friends Is users persona	Post
Post:	Attributes: photo, video, documents, avaliability Creates post associated with users	

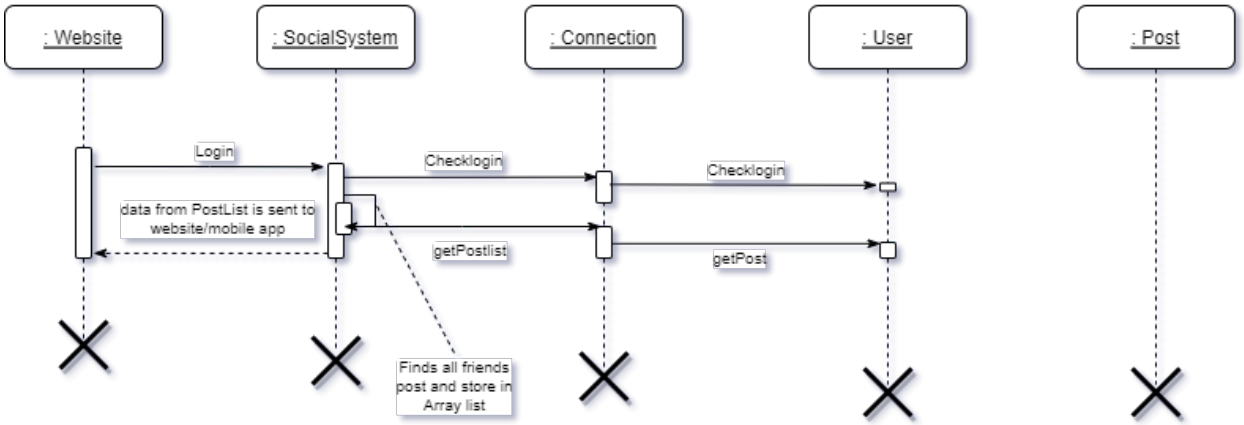
c)

Class_diagram



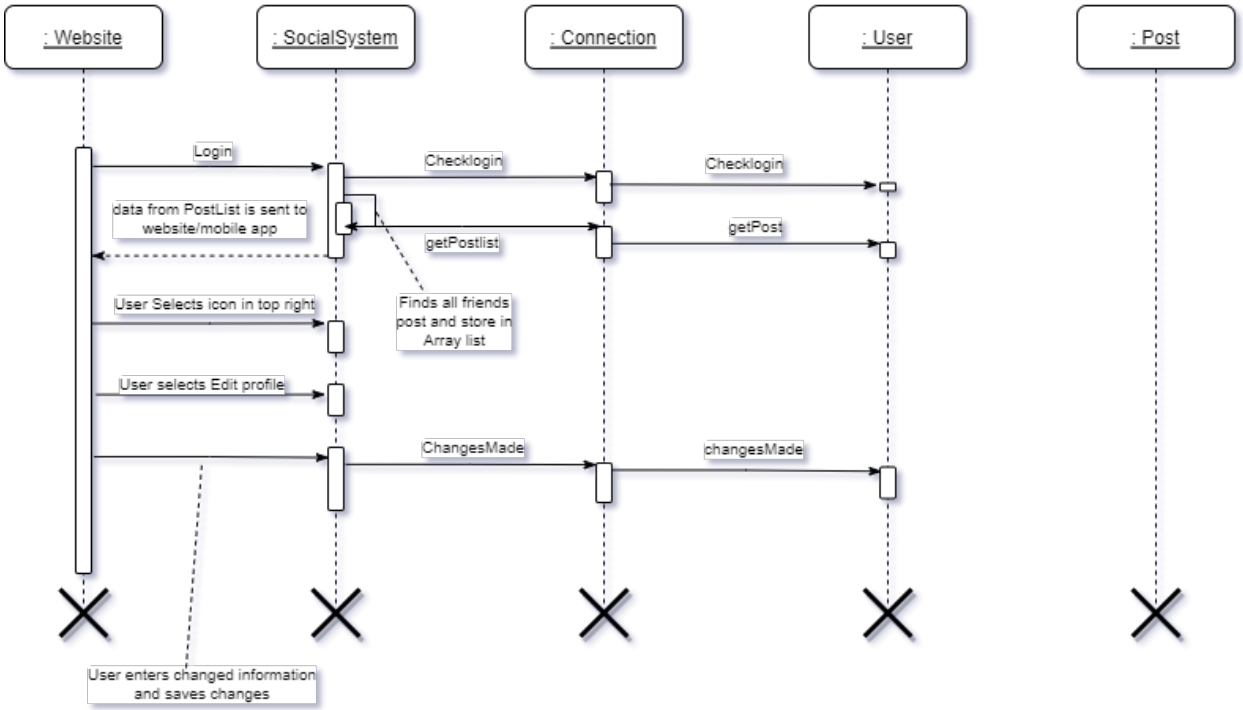
d)

Read Recent Post



EditProfile

Sequence_diagram



Write A post

Sequence_diagram