Zoinkers!

The purpose of this project was to create a scheduling system for the new akron zoo. This scheduling system had to meet multiple requirements and added a new spin on a scheduling system.

Abdulkareem Alali

Matt Skeins

5/2/2018

I believe one success from this project was that I got extremely comfortable using vectors. Before this project I had never really used them. Also, this project made me a better programmer overall as I learned how to design my code.

An unexpected event was my code completely broke inside wasp during the last phase of the project. For some reason my text file was inputting incorrectly and was completely corrupted. This caused hours of debugging as I tried to track down the issue.

What I learned from that issue is that text is converted differently when converting from windows to Unix. This caused my text file to become corrupt. To solve this error, I had to make a new text file with the exact same inputs. I learned so much from this project and overall became a better programmer.

This project performs extremely well. I believe the UI was designed to be very user friendly and performs extremely well with firs time users. Also, the process runs quickly as it does not use much memory.

Table of Tables

Contents

Figures

REQUIREMENTS

/*---

System Name: Zoinkers!

Artifact Name: vision.txt

Create Date: march 3, 2018

Author: Matt Skeins, mskeins@kent.edu

Version: 1.0

---*/

introduction:

The goal of this project is to make an effiecient system to track times animals are being

taken care of and then relaying proper viewing times to the customer. The new system will

be efficient, customizable, accessible, and centralized.

The system will be able to suggest the next target for customers to view based on

favourabillity of the animal and amount of funds remaining. We will be utilizing customers smart

phones to push notifications to them.

Customers can also buy special passes to give a discount to all zoo exhibits.

positioning:

Business Opportunity

This system would be extremely adaptable, changes to the system such as getting a new exihibit

wouldn't be an issue to update.As the zoo gets new exhibits it wouldnt take long to add to the system also an influx of customers

would be handled within the system so scaling will not be an issue. With the use of smart phones

integration would be extremely easy as smart phone use is common. this system will be designed with ease of use oin mind for not only customers but employees as well. The only way the system would get

an issue with new technology is if something is created to replace smartphones.

Problem Statement

Issues that might occur within this system is overcrowding of the zoo. The system may have a hard time scheduling if everyone

wants to see a brand new exhibit or if there is too many people not everyone will be happy with the times given.

Product Position Statement

A key feature of this system is to schedule customers based on favourabillity and help them organize their

time during their visit. Also, managers will be able to change prices for exhibits on the fly without resetting the system.

The system also supports infinitely scaling amount of exhibits.

alternatives

There are no direct alternatives to the system being developed

but direct competition is with other zoos such as columbus/cleveland zoo.

Stakeholder Descriptions:

The demographics for this system is users of all ages. this includes guests to the zoo and also

employees.

guest: pre built plan that gives them best viewing times according to favorbillity

manager: wants to be able to quickily and effienctly change prices and track other things around the zoo

keeper/vet/curator: maintain a consistant schedule for the animals and update schedules fast and easily

sales anylst: wants to be able to analyze different sales report and also wants a fast log in process sys admin: wants the system to be easily maintainable to be able to handle errors fast and efficiently

Product Overview:

The zoinkers system will ultimately be an app on users phones or could be on a computer as an app for employees

of the zoo. The system that employees use will be held in general offices where employees frequent or points of interest

identified before installation.

Summary of System Features:

- Change exhibit price (for managers only)
- customer schedule creation, based on favourabilillty and ranked on a hapiness scale
- system logs, all interactions and errors
- animal search (by name)
- transaction handeling, third party software payment authorization services
- account management

/*--

System Name: Zoinkers!

Artifact Name: orderplan.txt

Create Date: March 9, 2018

Author: Matt Skeins, mskeins@kent.edu

Version: 1.0

--*/

Use Case Name:

Order Plan

Scope:

Zoinkers!? system

Level:

user goal

Primary Actor:

Customer

Stakeholders and interests:

-customer: the customer wants to be able to set favourabillity of what animal they would like to see

to see most that day. Also the customer wants a schedule created in a timely manor.

preconditions:

Customer has application downloaded

Postconditions:

customer has a schedule generated with the highest happiness value based on their budget and favourabillity

Main success scenario:

1.) customer opens application on mobile device

2.) customer enters credentials

3.) User selects generate day plan

4.) user sets budget

5.) user sets favourabiliity

6.) user is returned a plan

EXTENSIONS:

1a.) customer is unable to open the application

- 1.) customer restarts application
- 2.) customer chooses to submit a bug report
- 3.) customer continues as normal

2a.) credentials are enetered incorrectly

- 1.) user is directed to try again

  this repeats untill user is logged in correctlly
- 2.) user is asked to make an account
  - 2a.) customer is directed to create account screen
- 3.) customer is authenticated

3a.) Page fails to load

- 1.) app must be restarted
- 2.) User is asked to make a bug report

4a.) user sets an invalid value for budget

- 1.) user is prompted to enter a new budget

  this repeats until correct budget entered
- 2.)customer eneters a valid budget

5a.) user sets an invalid value for favourabillity

- 1.) user is asked to set a correct value for favourabillity
- 2.) user continues as normal

Special Requirments:

- user must be using a compatible device
- all interactions must be processed quickily
- Schedule generation should take at most a minute
- Launguage will be in english

Technology and data variatoins list:

4a.) Must support credit input via digital input of credit card

Frequency of occurence:

occurs as often as customers request (can be very often)

open issues:

-what are the credit laws for applications?

-are there other ways of payment?

/*--

System Name: Zoinkers!

Artifact Name:manageUsers.txt

Create Date: March 9, 2018

Author: Matt Skeins, mskeins@kent.edu

Version: 1.0

--*/

Use Case Name:

manage users

Scope:

Zoinkers!? system

Level:

user goal

Primary Actor:

Manager

the role of this use case is to give managers a way

of looking through all of the users of the park.

The outcomes of this usecase include editing users

accounts and also being able to add or remove accounts if needed.

This will also give an organized list of all current users.

Stakeholders and interests:

-manager: the manager wants to be able to view all users inside of the park. the manager would also like to be able to add or remove accounts as neccesary.

preconditions:

manager has logged in

Postconditions:

Manager was successfully able to view all user data from the park and change account statuses

Main success scenario:

1.) Manager selects "Manage users" from the menu

2.) Manager selects to get a detailed list of all users

3.) Manager chooses to select an account to change

4.) Manager chooses to add or remove an account

5.) Manager logs out

EXTENSIONS:

2a.) No users in the system

1.) Manager is notified there is no users yet

2.) continue as normal

3a.) search result returns nothing

1.) manager is asked to search again

this loops until finished

2.) manager continues

4a.) Manager removes wrong account

1.) Manager is prompted to restore account

2.) Manager continues

Special Requirments:

user must be using a compatible device

all interactions must be processed quickily

List of users should be generated quickly

Launguage will be in english

Technology and data variatoins list:

1.) must not be on mobile device (or chooses to be)

Frequency of occurence:

occurs as often as managers request (can be very often)

open issues:

-are there other things the manager should be able to do with accounts?

/*---

System Name: Zoinkers!

Artifact Name: manageProfile.txt

Create Date: March 9, 2018

Author: Matt Skeins, mskeins@kent.edu

Version: 1.0

---*/

Use Case Name:

Order Plan

Scope:

Zoinkers!? system

Level:

user goal

Primary Actor:

Customer, Manager, zookeeper

the role of this is to change the settings in your profile. This could mean changing the favourbillity of each exhibit in your profile or changing the budget in your profile. Managers will be able to access different things from their profile such as managing orders, etc. Zookeepers will be able to change exhibit times and settings from within their profile.

Stakeholders and interests:

-Manager: wants to be able to manager account info

-Customer: wants to be able to change favourbillity and budget

-Zookeeper: wants to be able to change exhibit times and settings

preconditions:

user has logged in

Postconditions:

user has successfully managed or travesered the profile

Main success scenario:

1.) user selects "manage" from menu

2.) user selects apropriate action from the manage menu

3.) zookeeper changes exhibit times

4.) manager chooses to change prices or give discount

5.) customer changes favourbillity or budget

6.) user leaves manage menu

EXTENSIONS:

1a.) Page does not load

1.) app must be restarted (or page reloaded)

2.) user continues as normal

3a.) zookeeper enters an invalid time

1.) zookeeper is prompted to enter another time

this is looped until time is correctly entered

2.) zookeeper continues as normal

4a.) Manager enters an invalid price

1.) manager is prompted to enter a new price

this repeats until valid price entered

2.) manager enters a valid price

4b.) Manager sets a discount out of resonable bounds

1.) Manager is asked to varify discount with another manager

2.) Manager continues as normal

6a.) user enters an invalid budget or favourbillity

1.) user is prompted to enter a valid input

this step is looped until valid input entered

Special Requirments:

user must be using a compatible device

all interactions must be processed quickily

Schedule generation should take at most a minute

Launguage will be in english

Technology and Data Variations List:

Frequency of Occurrence:

Not very often, maybe during promotional days

Open Issues:

-will schedules be updated if exhibit changes mid day?

/*---

System Name: Zoinkers!

Artifact Name: manageExhibitInfo.txt

Create Date: March 9, 2018

Author: Matt Skeins, mskeins@kent.edu

Version: 1.0

---*/

Use Case Name:

Order Plan

Scope:

Zoinkers!? system

Level:

user goal

Primary Actor:

Zoo Kepper/ Vet / Curator

The goal of this use case is to be able to access and change the information pertaining to an exhibit. the outcome of this use case will produce either a new time schedule of the exhibit or other info about the animal/exhibit.

Stakeholders and interests:

- zookeeper wants to be able to change information about exhibits

preconditions:

zookeeper has logged in

Postconditions:

zookeeper has made changes to the exhibit

Main success scenario:

1.) user selects "manage" from menu

2.) user selects "change exhibit" from menu

3.) user changes details of exhibit

4.) user changes times of exhibits

5.) user exits exhibit managment

EXTENSIONS:

1a.) Page does not load

1.) app must be restarted (or page reloaded)

2.) user continues as normal

3a.) zookeeper enters an invalid time

1.) zookeeper is prompted to enter another time

this is looped until time is correctly entered

2.) zookeeper continues as normal

4a.) Manager enters an invalid price

1.) manager is prompted to enter a new price

this repeats until valid price entered

2.) manager enters a valid price

4b.) Manager sets a discount out of resonable bounds

1.) Manager is asked to varify discount with another manager

2.) Manager continues as normal

6a.) user enters an invalid budget or favourbillity

1.) user is prompted to enter a valid input

this step is looped until valid input entered

Special Requirments:

user must be using a compatible device

all interactions must be processed quickily

Schedule generation should take at most a minute

Launguage will be in english

Technology and Data Variations List:

Frequency of Occurrence:

could be frequent depening on needs of the animal

Open Issues:

should notification be sent if exhibit changes mid day

/*--

System Name: Zoinkers!

Artifact Name: logAnimalTimes.txt

Create Date: March 9, 2018

Author: Matt Skeins, mskeins@kent.edu

Version: 1.0

--*/

Use Case Name:

Order Plan

Scope:

Zoinkers!? system

Level:

user goal

Primary Actor:

zoo Keeper / vet / curator

The role of this use case is for zoo keepers to update care times for different animals and exhibits. The zoo keeper will need to log into their profile then access the correct exhibit. Once accessed they can edit the times the animal is being cared for or at the vet.

Stakeholders and interests:

-Zookeeper/ vet / curator: wants to be able to update care times for animals

preconditions:

user has logged in

Postconditions:

user has updated times relating to the exhibit

Main success scenario:

1.) user searches for an exhibit

2.) user selects "edit schedule" from menu

3.) user adds times that exhibit will be unavailabe for viewing

4.) user removes previous time schedules that no longer apply

5.) user adds notes for specific scheduling events

6.) user confirms changes

EXTENSIONS:

1a.) user result from search returns nothing

- 1.) user is prompted to enter another search

  this loops until user is finished searching

- 2.) user continues as normal

3a.) user adds a time out of bounds (24 hour format)

- 1.) user is prompted to enter another time

  this is looped until time is correctly entered

- 2.) user enters a time that is already alotted

  - 2a.) user is asked to eneter another time

    this loops until user enters valid time

6a.) user decides to not confirm changes

- 1.) user is taken back to editing

- 2.) user confirms and changes are saved

Special Requirments:

user must be using a compatible device

all interactions must be processed quickily

Schedule generation should take at most a minute

Launguage will be in english

Technology and Data Variations List:

Frequency of Occurrence:

could happen often throughout the day depends on animal

Open Issues:

need protection from user error

```
/*-----------------------------------------------------------------

System Name: Zoinkers!

Artifact Name: orderplan.txt

Create Date: March 9, 2018

Author: Matt Skeins, mskeins@kent.edu

Version: 1.0

-----------------------------------------------------------------*/
```

Use Case Name:

Manage Orders

Scope:

Zoinkers!? system

Level:

user goal

Primary Actor:

manager

the role of this use case is to manage orders generated by customers.

also this will allow managers to look at sale statisics

of the zoo as a whole. This use case also allows the managers

to distribute discounts where neccesary

Stakeholders and interests:

-Zookeeper/ vet / curator: wants to be able to update care times for animals preconditions:

user has logged in

Postconditions:

user has updated times relating to the exhibit

Main success scenario:

1.) user searches for an exhibit

2.) user selects "edit schedule" from menu

3.) user adds times that exhibit will be unavailabe for viewing

4.) user removes previous time schedules that no longer apply

5.) user adds notes for specific scheduling events

6.) user confirms changes

EXTENSIONS:

1a.) user result from search returns nothing

- 1.) user is prompted to enter another search

  this loops until user is finished searching

- 2.) user continues as normal

3a.) user adds a time out of bounds (24 hour format)

- 1.) user is prompted to enter another time

  this is looped until time is correctly entered

- 2.) user enters a time that is already alotted

  - 2a.) user is asked to eneter another time

    this loops until user enters valid time

6a.) user decides to not confirm changes

1.) user is taken back to editing

2.) user confirms and changes are saved

Special Requirments:

user must be using a compatible device

all interactions must be processed quickily

Schedule generation should take at most a minute

Launguage will be in english

Technology and Data Variations List:

Frequency of Occurrence:

as often as needed (could be frequent)

Open Issues:

should implement two person auth on discounts

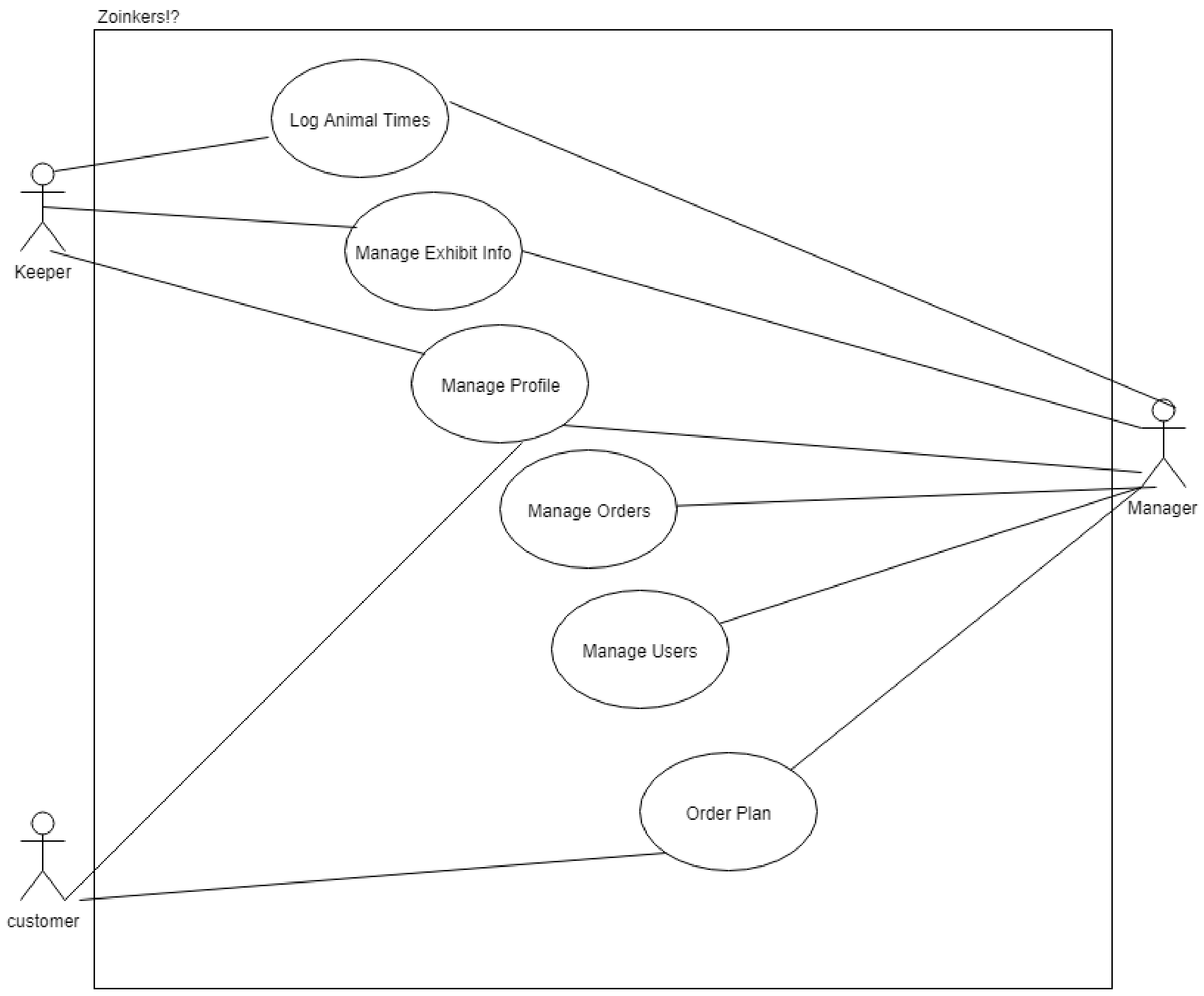

Figure 1 usecase model

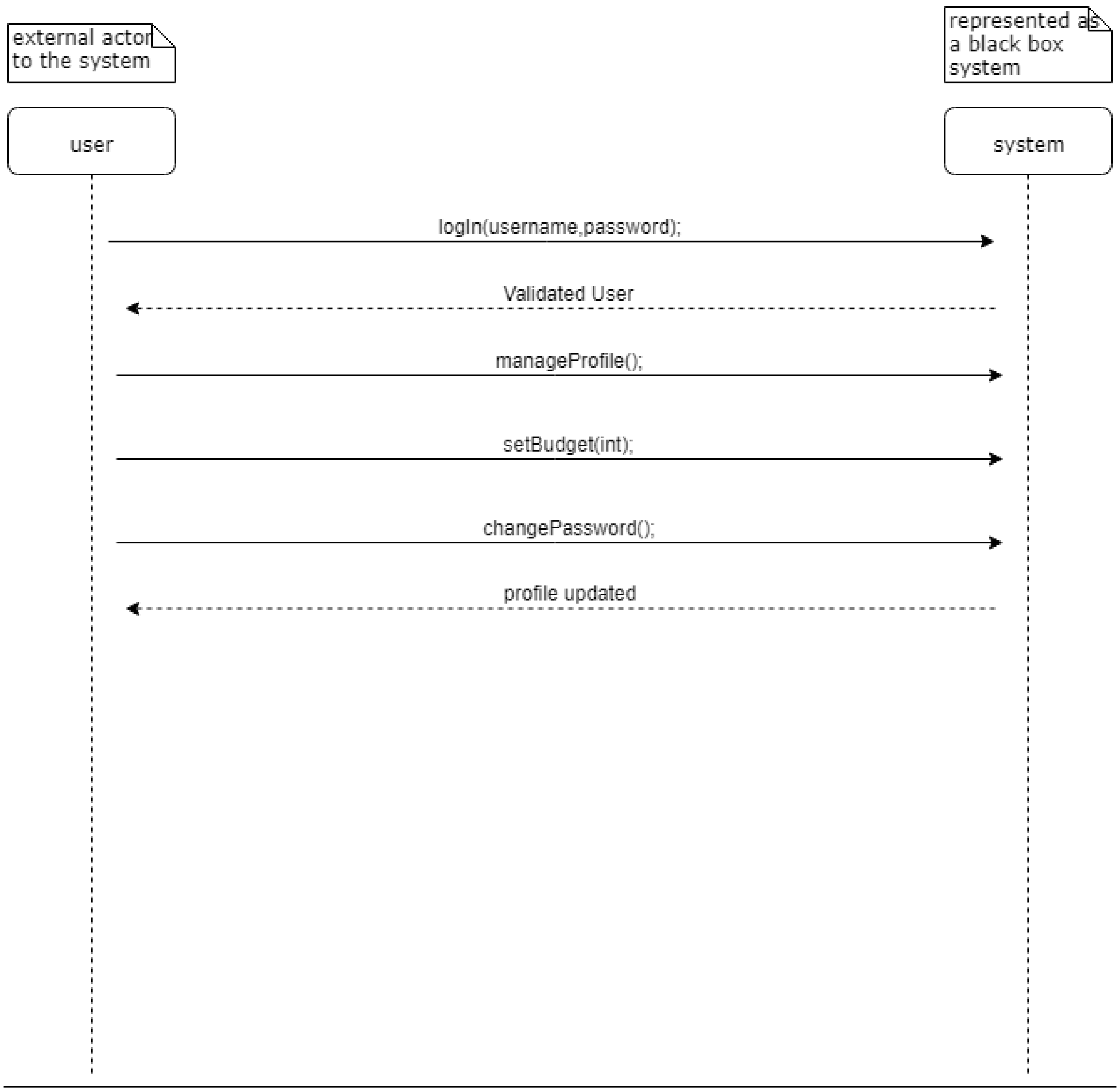
external actor to the system
represented as a black box system
user
system
logIn(username,password);
Validated User
manageProfile();
setBudget(int);
changePassword();
profile updated

Figure 2 sd

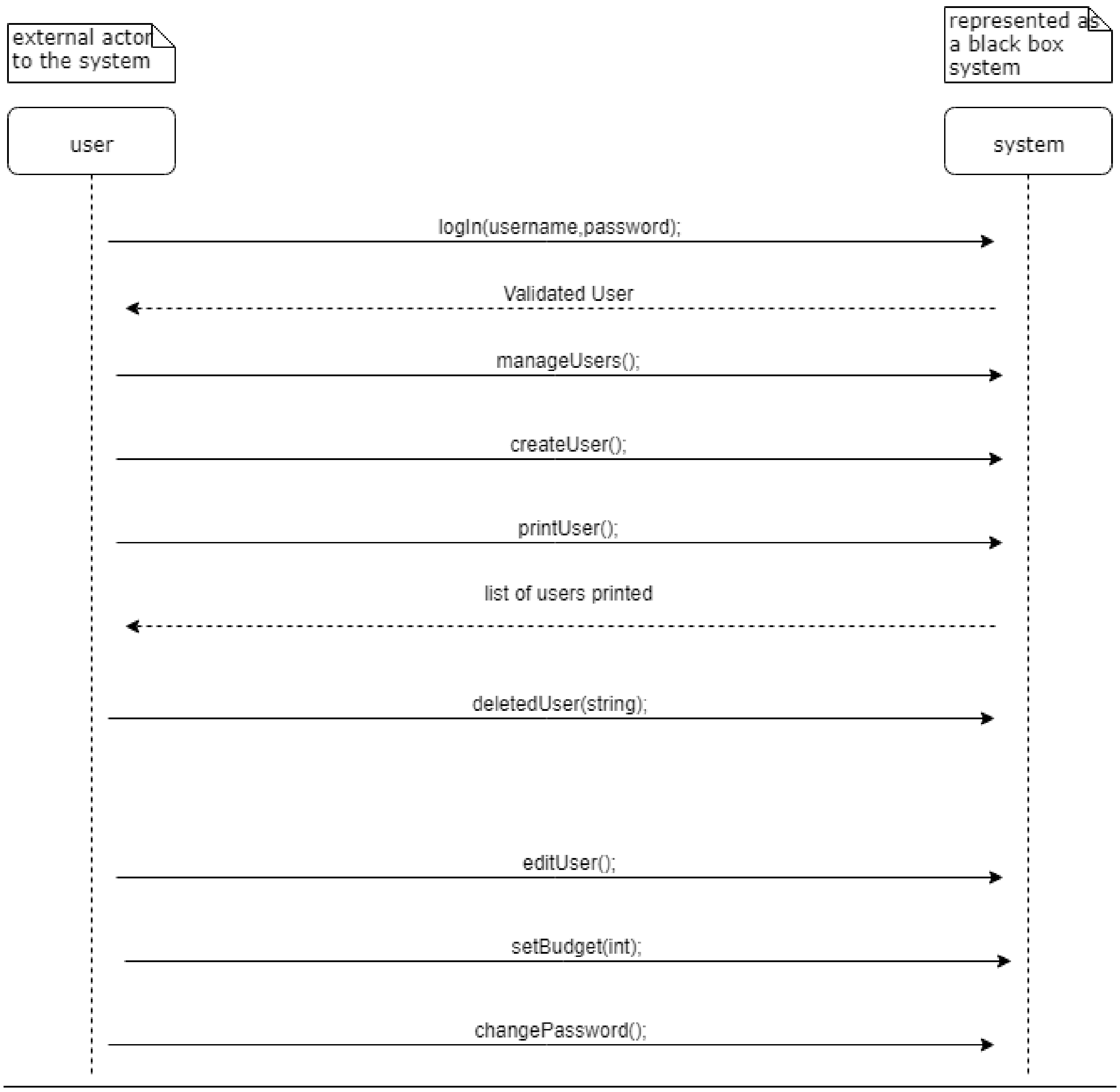
external actor to the system
represented as a black box system
user
system
logIn(username,password);
Validated User
manageUsers();
createUser();
printUser();
list of users printed
deletedUser(string);
editUser();
setBudget(int);
changePassword();

Figure 3 sd

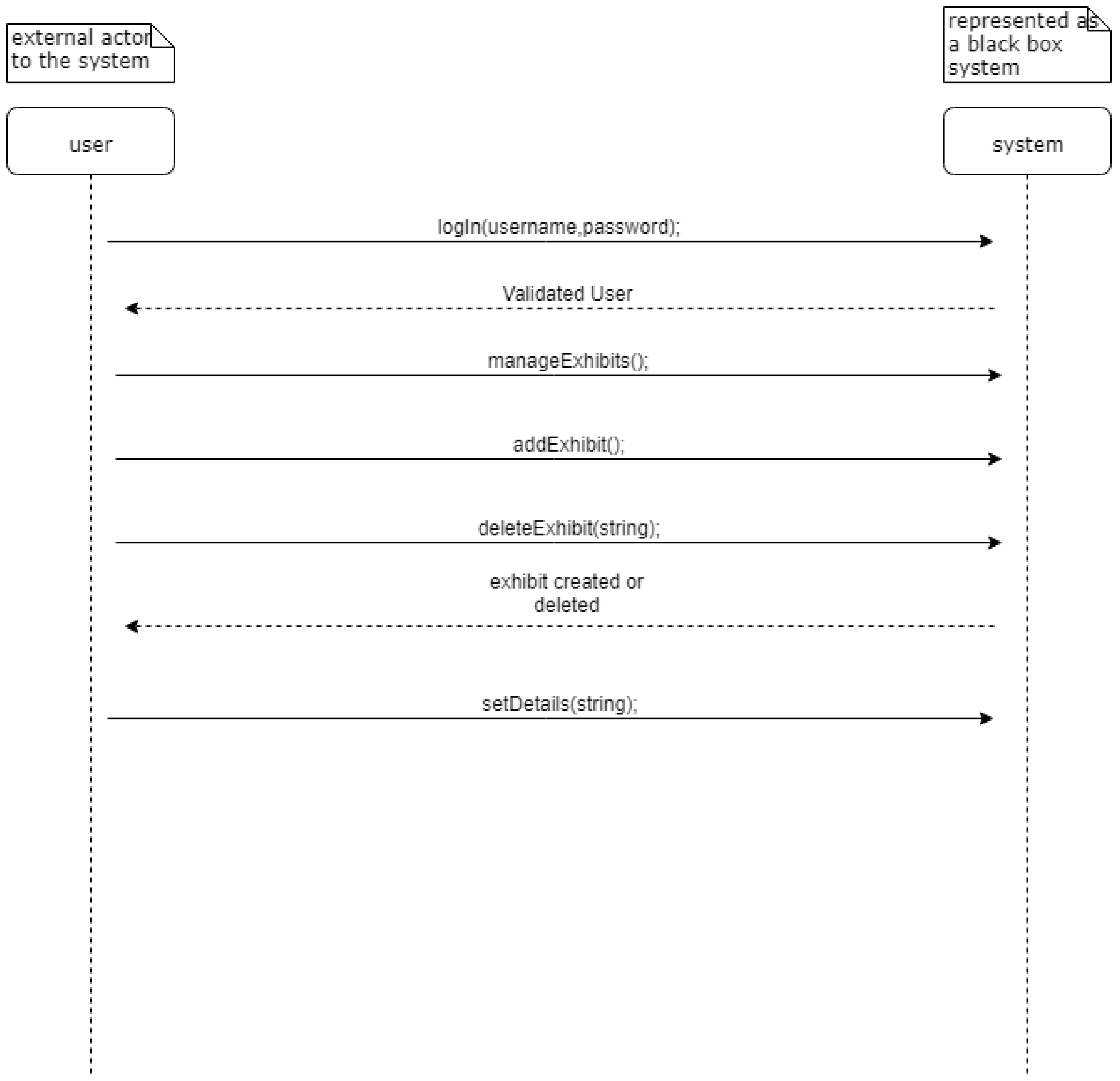
external actor to the system
represented as a black box system
user
system
logIn(username,password);
Validated User
manageExhibits();
addExhibit();
deleteExhibit(string);
exhibit created or deleted
setDetails(string);

Figure 4 sd

/*---

System Name: Zoinkers!

Artifact Name: supplementary_specification.txt

Create Date: March 3, 2018

Author: Matt Skeins, mskeins@kent.edu

Version: 2.0

---*/

Usabillity:

the user is not expected to be extremely familiar with the software. The only issue is

whether they can use a smart phone or not.

Reliablillty:

the system will be using some sort of log in feature to make sure that the customers don't

get access to what managers / keepers can do witht he system. The system will also be logging errors

this will help define issues that come up. During failures we may be able to use a website as a back up for the system instead

of an app on the smartphone.

Performance:

This system will be able to support as many users as the zoo has visitors. we

want this system to be as fast as possible with the user in mind. logging in should

take less than one minute always. also transactions should take no longer than 30 seconds. Schedules

should be created and displayed quickily without hindering users time.

Supportablillty:

This system may need a system admin to maintain and handle errors for the system.

the system admin may also extend the system by adding exhibits.

Constraints:

Implementation:

-customers will have different phones varying in power

-different OS for phones

Interface:

-this system will only be in english upon release

Operations:

-Must be able to handle inputs and give responses quickly

Packaging:

-No special Packaging

Legal

-Third party licensing

```
/*-----------------------------------------------------------------
System Name: Zoinkers!
Artifact Name: glossary.txt
Create Date: March 3, 2018
Author: Matt Skeins, mskeins@kent.edu
Version: 1.0
-----------------------------------------------------------------*/
```

Glossary

-Term-	-Definition-
budget	This is a value set by the user for the amount of money willing to spend at the zoo

User Authorization	a way to validate different types of users who log into the applications
Favourbillity	orbitrary value used to indicate how badly a customers wants to see a certain animal
Logging	Generating organized reports based on the situation "writing down things"

/*---

System Name: Zoinkers!

Artifact Name: business_rules.txt

Create Date: March 3, 2018

Author: Matt Skeins, mskeins@kent.edu

Version: 1.0

---*/

Busnniess rules:

Revision History

Version	Date	Description	Author
inception draft elaboration	March 9th, 2018	First draft will change during	Matt Skeins

ID	Rule	Changabillity	Source
RULE1	Signiture required for credit payments	Must change with current trends must support digital pay and signitures	POlicy of most credit companies
RULE2	Sales tax, see government statutes	High, taxes change anually	Law
RULE3	Credit refunds must be repaid to user's account as credit	Low	Credit Auth Policies

BUSINESS MODELING

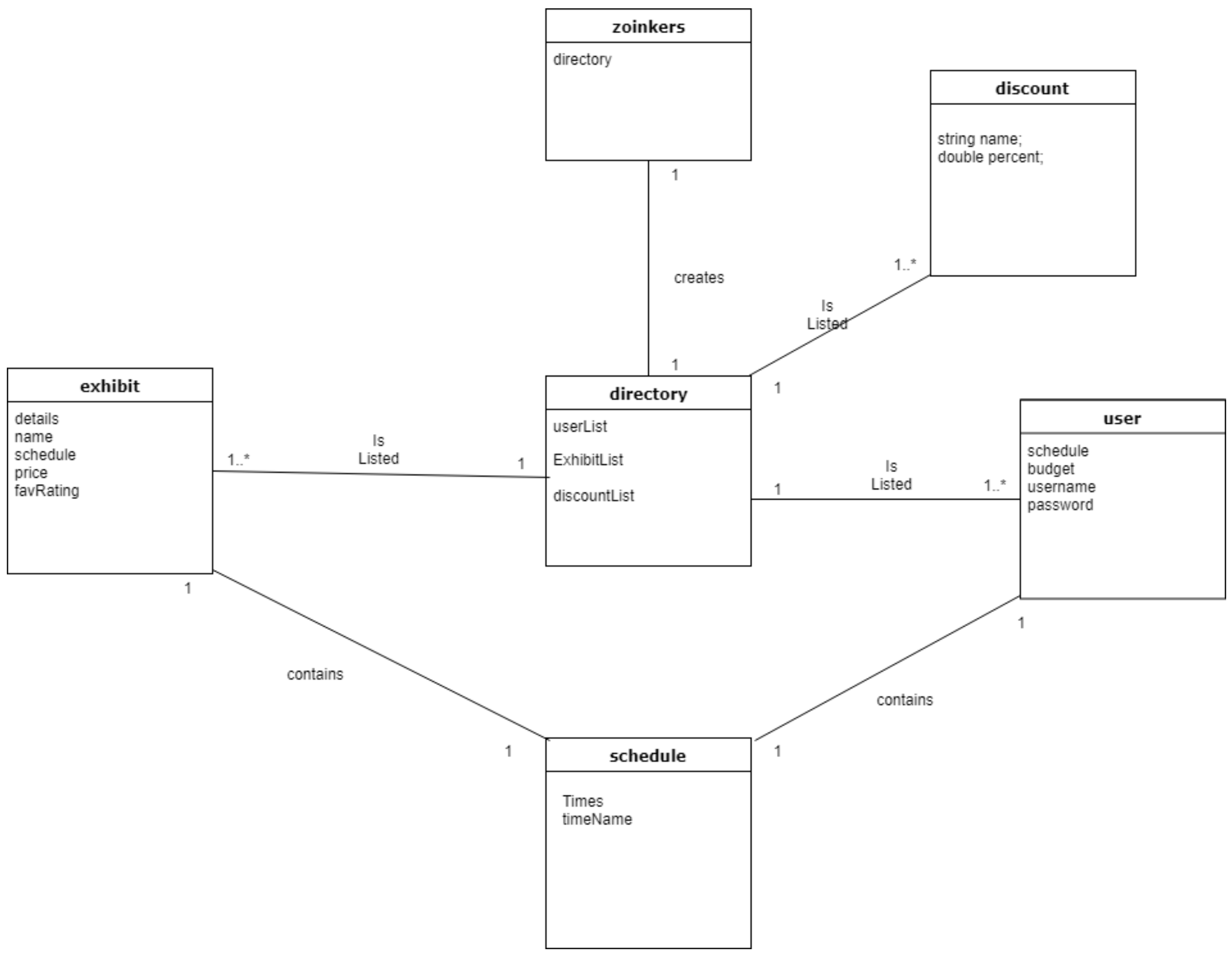

Figure 5 domain model

DESIGN

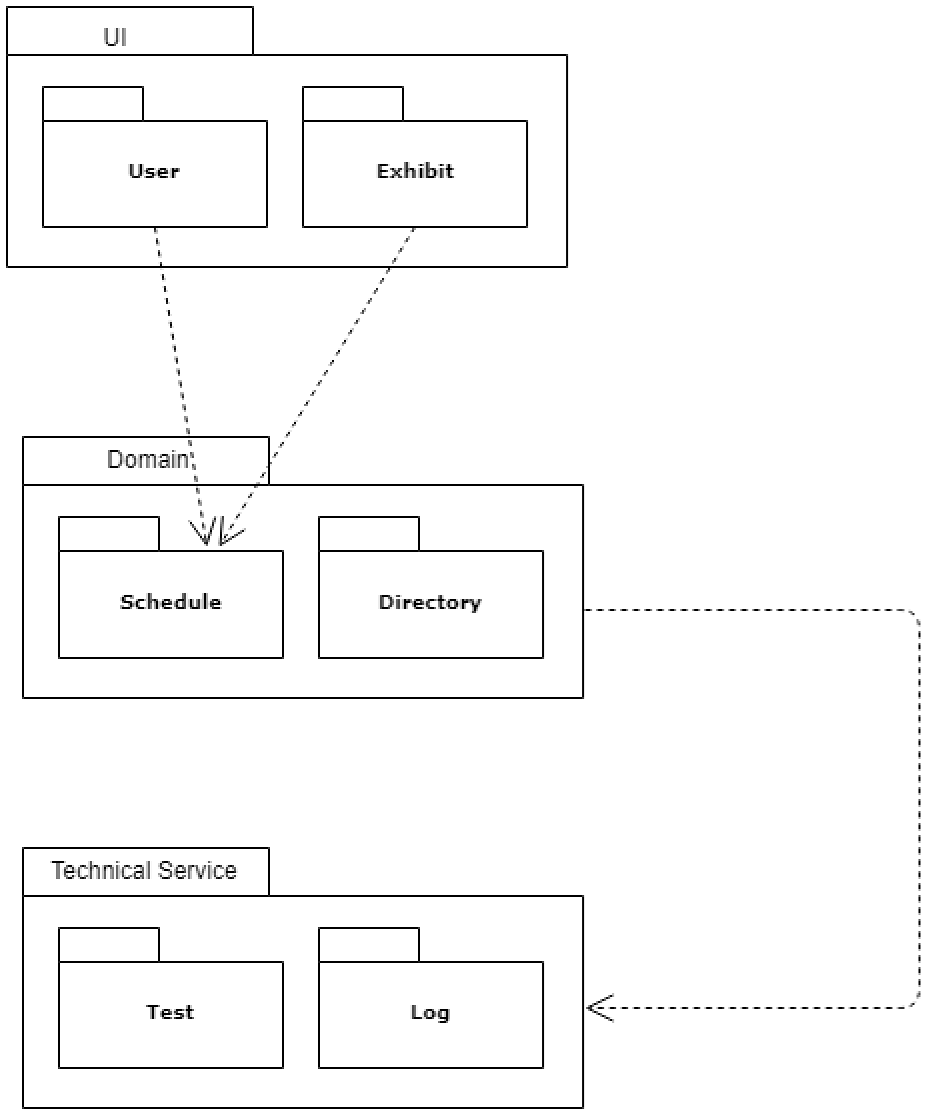

Figure 6 package diagram

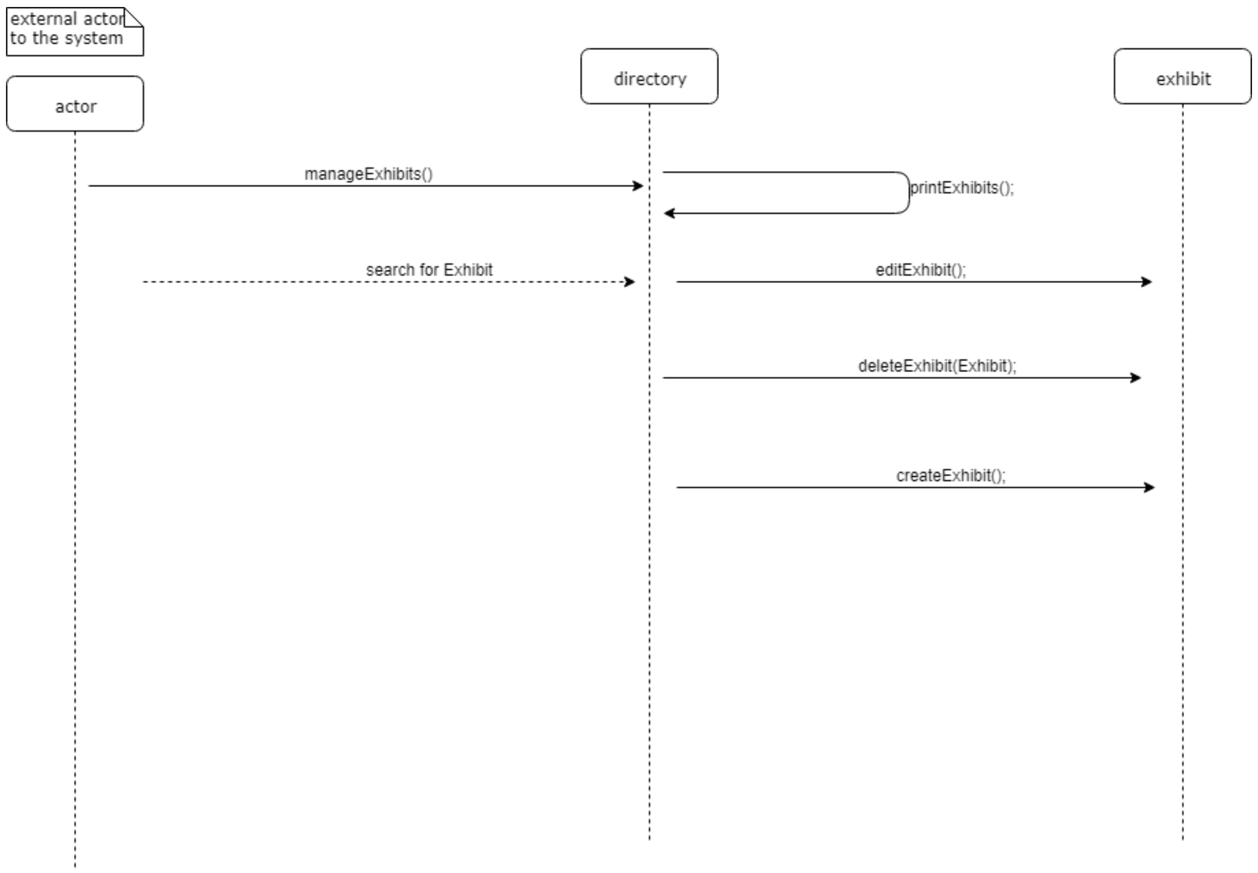
external actor to the system
actor
directory
exhibit
manageExhibits()
printExhibits();
search for Exhibit
editExhibit();
deleteExhibit(Exhibit);
createExhibit();

Figure 7 sd

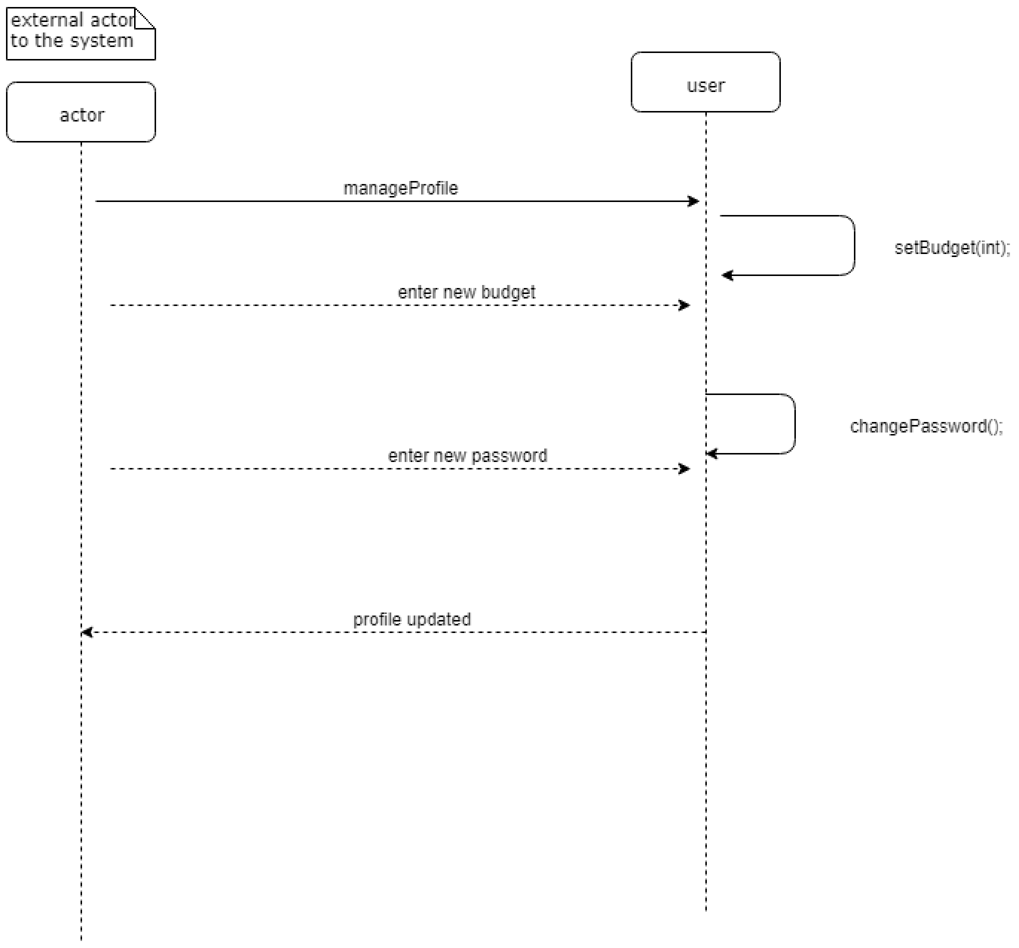
external actor to the system
actor
user
manageProfile
setBudget(int);
enter new budget
changePassword();
enter new password
profile updated

Figure 8 sd

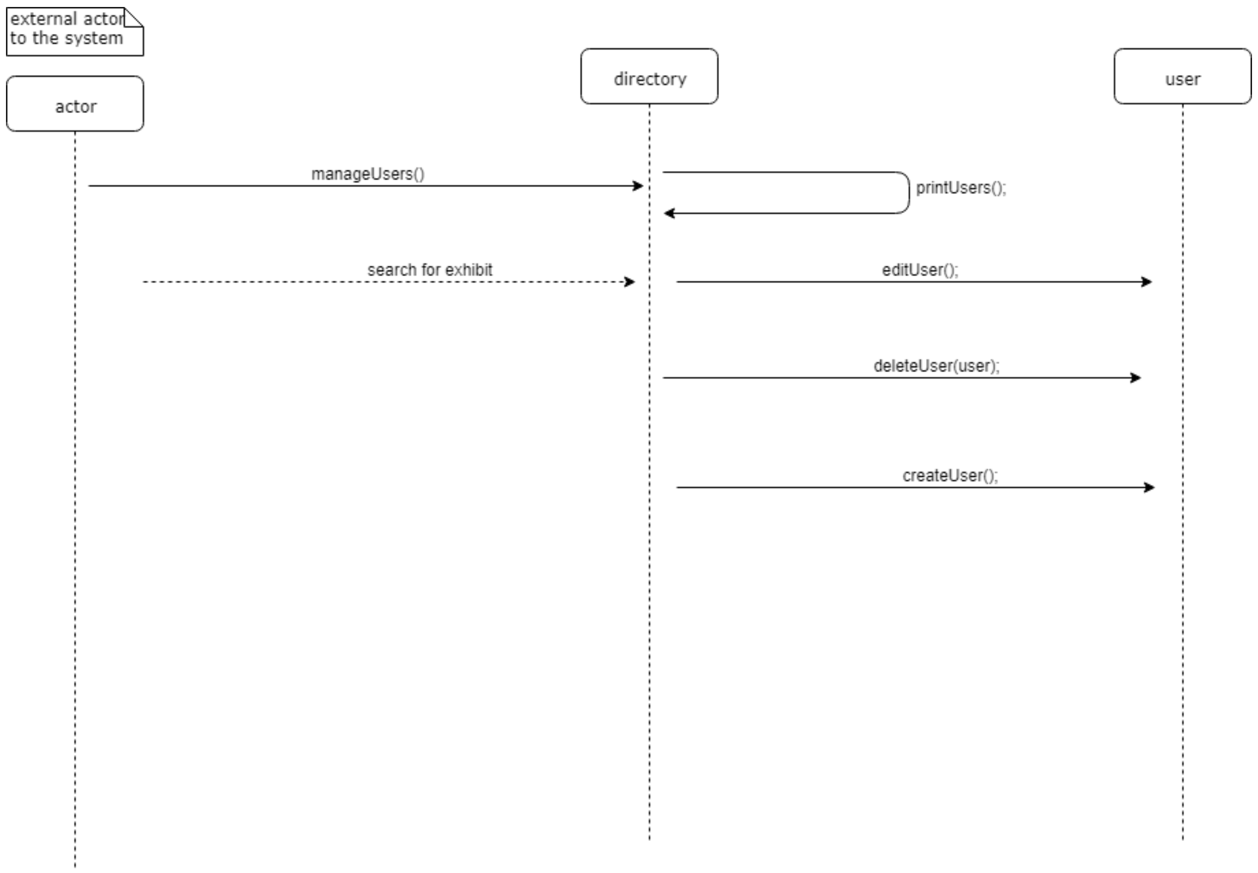
external actor to the system
actor
directory
user
manageUsers()
printUsers();
search for exhibit
editUser();
deleteUser(user);
createUser();

Figure 9 sd

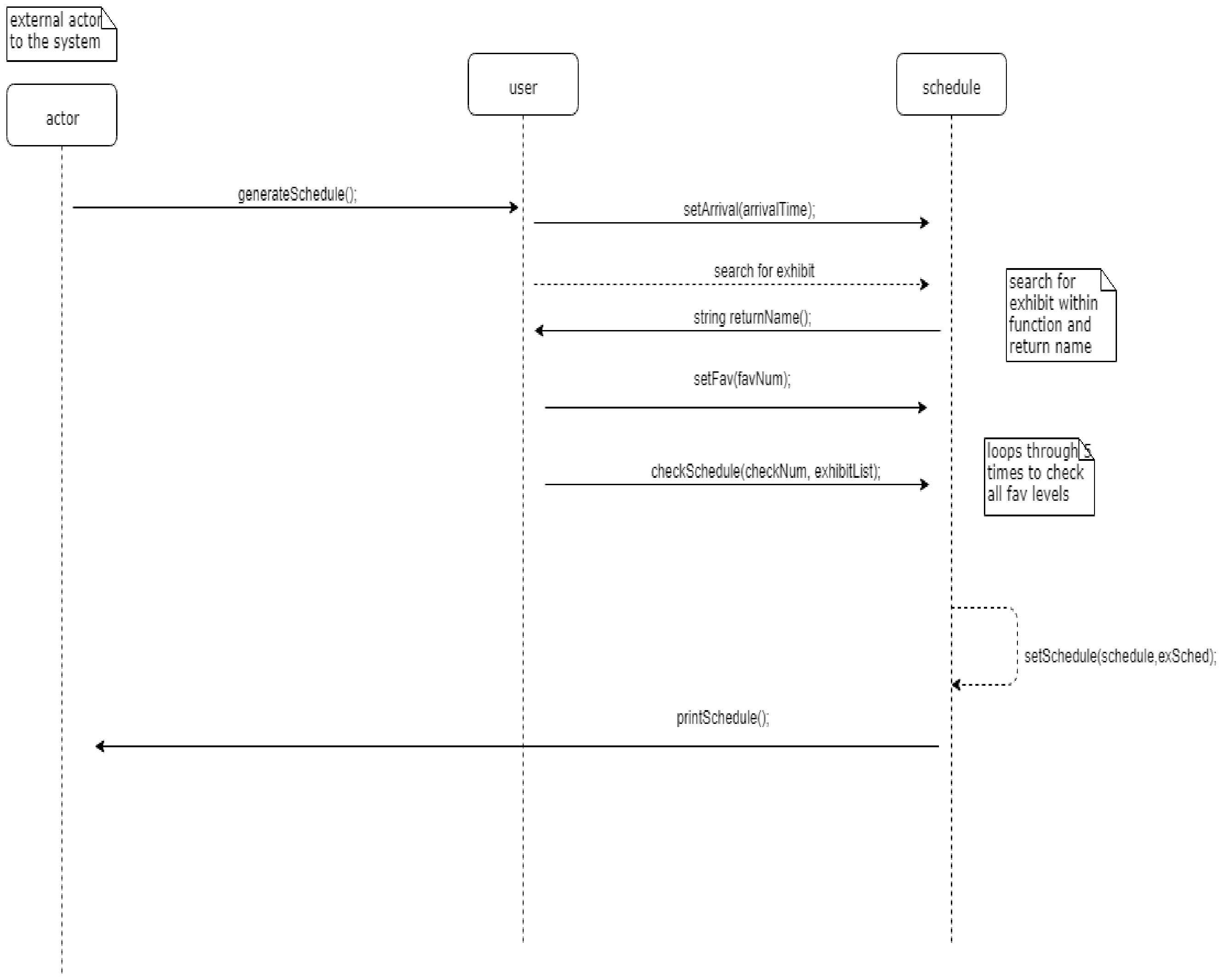
external actor to the system
actor
user
schedule
generateSchedule();
setArrival(arrivalTime);
search for exhibit
search for exhibit within function and return name
string returnName();
setFav(favNum);
checkSchedule(checkNum, exhibitList);
loops through 5 times to check all fav levels
setSchedule(schedule,exSched);
printSchedule();

Figure 10 sd

IMPLEMENTATION

```
#ifndef ZOINKERS_H_

#define ZOINKERS_H_
```

```
#include <cstdlib>
#include <ctime>
#include<iostream>
#include <vector>
#include <fstream>
#include <string>
#include <iomanip>
#include <chrono>
#include <thread>
using namespace std;
/*-----------------------------------------------------------------
System Name: Zoinkers!
Artifact Name: Zoinkers.h
Create Date: May 1, 2018
Author: Matt Skeins
Version: 3.0
-----------------------------------------------------------------*/



/*      TODO:
Done    1.) MAKE LOGIN() FUNCTION
Done    2.) HARD CODE SOME EXHIBITS
Done    3.) CREATE SCHEDULE FUNCTIONS
Done    4.) GENERATE PLAN FUNCTION
Done    5.) SEARCH EXHIBIT FUNCTION
Done    6.) BUILD TESTING CLIENT
Done    7.) UPDATE LOGGING TO BE REGULAR FUNCTRIONS NOT ATTACHED TO A CLASS
Done    8.) DO HAPINESS CALCULATIONS AND (done)ARRIVAL TIME SCHEDULING
Done    9.) ALL LOG EVENTS
Done    10.) SPECIAL PASS BUYING
Done    11.) USE CASES
Done    12.) API STUFF (CREDIT CARD MUST FAIL RANDOMLY SOMETIMES (5%))
Done    13.) AUTO CREATE USER
Done    14.) AUTO CREATE EXHIBIT
Done    15.) AUTO CREATE DISCOUNT FOR PASS BUYING
14.) NEED TO UPDATE TEST CLIENT
Done    16.) DO LOG IN/ OUT FUNCTIONALITY NEED TO UPDATE VECTOR AFTER LOG OUT
Done    17.) ADD DISCOUNT TO GENERATE PLAN
Done    18.) PRINT USERS,EXHIBITS
Done    19.) FIX ERASE FUNCTION
Done    20.) KEEPER FUNCTIONS
21.) REFINE USER ERRORS
*/





class schedule {


public:
        schedule();
        void addTime(int);
        void addTimeName(string, int);
        //bool checkTime(int);
        //void removeTime(int);
        void printSchedule();
        bool checkTime(int x);
        void setArrival(int time);

private:


        bool times[24];
        string timeName[24];

};

class API {

public:
        void creditValidator();

};

class discount {

public:
        string returnName();
        int returnPercent();
        void setName(string tmp);
        void setPercent(int tmp);
private:
        string name;
        double percent;
```

```
};

class exhibit {

public:
        exhibit();
        void setDetails(string);
        void setPrice(double);
        void setName(string);
        void setFav(int);
        string returnName();
        double returnPrice();
        int returnFav();
        schedule returnSchedule();
        string returnDetails();

private:
        //details should include: (name, parents, type, birthdate, picture, history, dietary info, sleeping
info,and bathing info
        string details;
        string name;
        schedule exhibitSchedule;
        double price;
        int favorabillity;

};
```

```
class user {

public:

        void generatePlan(vector<exhibit>, vector<discount>);
        vector<exhibit> setFav(vector<exhibit>); // should this take in an exhibit
        int checkSchedule(int favNum, vector<exhibit> ExList);
        void setSchedules(schedule userSchedule, schedule ExhibitSchedule, string name);

        string returnUsername();
        string returnPassword();
        string returnRole();

        void setUsername(string);
        void setPassword(string);
        void setBudget(double);
        void setArrival(int time);
        void setRole(string newRole);



        void manageProfile();

        //needs updated
        void INPUTgeneratePlan(vector<exhibit>);
        void INPUTmanageProfile();
```

```
private:

        schedule mySchedule;

        double budget;

        string username;

        string password;

        string role;

};


class directory {


public:


        user Login();

        void createUser();

        void createExhibit();


        //needs input

        void createDiscount();


        void autoCreateUser(string, string, string);

        void autoCreateExhibit(string, double);

        void autoCreateDiscount(string, int);


        user searchUser(string username);

        int searchUserReturnIndex(string username);

        discount searchDiscount(string code);

        exhibit searchExhibit(string exhibitName);
```

```
//needs input

void manageUsers();

void manageExhibits();

//


void deleteExhibit(string);

void deleteUser(string);


void printUsers();

void printExhibits();


void updateUser(user);

void updateExhibit(exhibit);


//prints and executes menu stuff

void menuAction(user&, bool&);




bool userExists(string);

vector<exhibit> returnExhibitDir();


//input funcitons for test client

void INPUTmanageExhibits();

void INPUTmanageUsers();

void INPUTcreateDiscount();

void INPUTmenuAction(user&, bool&);

void INPUTcreateUser();
```

```
        void INPUTcreateExhibit();
        user INPUTLogin();
private:
        vector <exhibit> exhibitDir;
        vector <user> userDir;
        vector <discount> allDiscounts;
};

class zoinkers {

public:
        void systemStart();
private:
        directory mainDirectory;

};
class testing {



public:
        bool returnAuth();
        void authenticate();
        string returnUsername();
        string returnPassword();
        bool logIn();
        void start();

        void systemStart();
```

```
private:
        string username = "WARPIGS";
        string password = "SABBATH";
        bool authenticated = false;
        zoinkers zoinkersSystem;
};

class logging {

public:
        void logEvent(string str);
        void getDate(ofstream& myfile);
};

//api functions
void API::creditValidator()
{
        string creditNumber;
        string csv;

        cout << "enter credit card number: ";
        cin >> creditNumber;
        cout << "eneter csv: ";
        cin >> csv;

        int fail;
        srand(time(NULL));
        fail = rand() % 10;
```

```
		if (fail < 2) {
			cout << "card declined" << endl;
		}
		else {
			cout << "card verified" << endl;
		}



}

//discount functions
inline string discount::returnName()
{
		return name;
}

int discount::returnPercent()
{
		return percent;
}

inline void discount::setName(string tmp)
{
		name = tmp;
}

inline void discount::setPercent(int tmp)
{
        percent = tmp;
}

//testing functions

inline bool testing::returnAuth()
{
        return authenticated;
}

inline void testing::authenticate()
{
        authenticated = true;
}



string testing::returnUsername()
{
        return username;
}

inline string testing::returnPassword()
{
        return password;
}

inline bool testing::logIn()
{
	string username;

	string password;

	cout << "eneter the username for testing: ";

	cin >> username;



	if (username == returnUsername()) {

		cout << "Please enter the password: ";

		cin >> password;

		if (password == returnPassword()) {

			authenticate();

		}


	}


	if (returnAuth() == true) {

		cout << "Login successfull" << endl;

		return returnAuth();

	}

	else {

		cout << "INVALID LOGIN, you will now be directed to the main system" << endl;

	}

	return returnAuth();

}




inline void testing::systemStart()
```

```
{
	zoinkersSystem.systemStart();
}

//system functions
void zoinkers::systemStart()
{
	//NEED TO ADD LOOP FOR USERS LOGGING OUT OR LOGGING IN
	//logging event
	logging test;



	test.logEvent("System Started ");
	mainDirectory.autoCreateUser("FLOYD", "SHEEP", "manager");
	mainDirectory.autoCreateUser("ZEPPELIN", "BLACKDOG", "customer");
	mainDirectory.autoCreateUser("APE", "NIRVANA", "keeper");

	mainDirectory.autoCreateExhibit("lion", 5);
	mainDirectory.autoCreateExhibit("sheep", 3);
	mainDirectory.autoCreateExhibit("zebra", 7);

	mainDirectory.autoCreateDiscount("pass", 30);

	char testChoice;
	cout << "what would you like to do?: " << endl;
	cout << "USER SCENARIOS (u)" << endl;
	cout << "INPUT SCENARIOS (i)" << endl;
	cin >> testChoice;
```

```
        if (testChoice == 'u') {

                bool exit = false;
                //loops system until user wants to close
                while (exit != true) {

                        user myUser;
                        bool loggedIn;

                        //ask if they want to log in or create user
                        string choice;
                        cout << "would you like to log in or create a new user?\n1.) new user\n2.) log
in\n3.) exit system\n";
                        cin >> choice;

                        if (choice == "1") {
                                mainDirectory.createUser();
                        }
                        else if (choice == "2") {
                                myUser = mainDirectory.Login();
                                loggedIn = true;
                                while (loggedIn == true) {
                                        mainDirectory.menuAction(myUser, loggedIn);
                                        //update user after action occured
                                        mainDirectory.updateUser(myUser);


                                }
                        }
                        else {
                                exit = true;
                        }
                }



        }
        else if (testChoice == 'i') {
                bool exit = false;
                //loops system until user wants to close
                user myUser;
                bool loggedIn;
                myUser = mainDirectory.INPUTLogin();
                loggedIn = true;

                while (loggedIn == true) {
                        mainDirectory.INPUTmenuAction(myUser, loggedIn);
                        //update user after action occured
                        mainDirectory.updateUser(myUser);
                        std::this_thread::sleep_for(std::chrono::milliseconds(1500));
                }

                myUser = mainDirectory.INPUTLogin();
                loggedIn = true;

                while (loggedIn == true) {
                        mainDirectory.INPUTmenuAction(myUser, loggedIn);
                        //update user after action occured
                mainDirectory.updateUser(myUser);

                system("pause");

            }


            myUser = mainDirectory.INPUTLogin();

            loggedIn = true;


            while (loggedIn == true) {

                mainDirectory.INPUTmenuAction(myUser, loggedIn);

                //update user after action occured

                mainDirectory.updateUser(myUser);

                system("pause");

            }


        }




        //logEvent("System Shut Down");

        test.logEvent("system shutdown ");

        system("PAUSE");

}


//directory functions


inline user directory::Login()

{

        user currentUser;

        user empty;
```

```
string user;

string password;



cout << "Please enter your username: ";

cin >> user;

while (userExists(user) != true) {

        cout << "Please enter your username: ";

        cin >> user;

}

currentUser = searchUser(user);


cout << "please enter your password: ";

cin >> password;


if (password != currentUser.returnPassword()) {

        cout << "Log in failed, incorrect password" << endl;

        //log event

        //logEvent("User Log In (failed)");

        //logging event

        logging test;

        test.logEvent("User Log In (failed) ");

        return empty;

}

else {

        cout << "LOG IN SUCCESSFUL" << endl;

        //log event

        //logEvent("User Log In (success)");
```

```
                logging test;
                test.logEvent("User Log In (success) ");
                return currentUser;
        }


}

void directory::createUser()
{
        //NEED TO ADD ROLE FUNCTIONALITY

        user newUser;
        cout << "please enter a username: ";
        string username;
        string password;
        cin >> username;



        //checks if username exists already in the system
        while (userExists(username) == true) {
                cout << "Please enter a different username, chosen username is taken: " << endl;
                cin >> username;
        }

        newUser.setUsername(username);
        cout << "please enter a password: ";
        cin >> password;
```

```
	newUser.setPassword(password);




	//assigning role to user
	char choice;
	cout << "what type of account are you creating? \n 1.)customer \n 2.)manager \n 3.)keeper \n";
	cin >> choice;
	if (choice == '1') {
		newUser.setRole("customer");
	}
	else if (choice == '2') {
		string code;
		cout << "Please eneter verification code for manager creation (for testing use 0000):" << 
endl;
		cin >> code;
		if (code == "0000") {
			cout << "VERIFIED \n";
			newUser.setRole("manager");
		}
		else {
			cout << "verification failed defaulting to customer account" << endl;
			newUser.setRole("customer");
		}
	}
	else {
		string code;
```

```
                cout << "Please eneter verification code for keeper creation (for testing use 1111):" <<
endl;

                cin >> code;

                if (code == "1111") {

                        cout << "VERIFIED \n";

                        newUser.setRole("keeper");

                }

                else {

                        cout << "verification failed defaulting to customer account" << endl;

                        newUser.setRole("customer");

                }

        }


        //pushing new user to directory of users

        userDir.push_back(newUser);

        cout << "username:       " << userDir.at(0).returnUsername() << "\npassword:      " <<
userDir.at(0).returnPassword() << "\nrole: " << userDir.at(0).returnRole() << endl;

        cout << "account creation successful" << endl;


        cout << "current size of user directory:   " << userDir.size() << endl;


        //log event

        //logEvent("User Created");

        logging test;

        test.logEvent("User Created ");

}


inline void directory::createExhibit()

{
        exhibit newExhibit;
        string name;
        double price;

        cout << "enter name for exhibit: ";
        cin >> name;
        cout << "enter price for exhibit: ";
        cin >> price;

        newExhibit.setName(name);
        newExhibit.setPrice(price);

        exhibitDir.push_back(newExhibit);
        cout << "Number of exhibits: " << exhibitDir.size() << endl;

        //log event
        //logEvent("Exhibit Created");
        logging test;
        test.logEvent("Exhibit Created ");
}

inline void directory::createDiscount()
{
        discount newDiscount;
        string newName;
        int disPercent;

        cout << "eneter code for new discount: ";
        cin >> newName;
        cout << "eneter percent of discount: ";
        cin >> disPercent;

        newDiscount.setName(newName);
        newDiscount.setPercent(disPercent);

        allDiscounts.push_back(newDiscount);
        cout << "Number of discounts: " << allDiscounts.size() << endl;

        logging test;
        test.logEvent("Discount Created ");
}

inline void directory::autoCreateUser(string name, string newPassword, string userRole)
{
        //can hard code users for testing
        user newUser;
        newUser.setUsername(name);
        newUser.setPassword(newPassword);
        newUser.setRole(userRole);

        userDir.push_back(newUser);
        logging test;
        test.logEvent("User Created ");
}

inline void directory::autoCreateExhibit(string name, double price)
```

```
{
        //can hard exhibits for testing
        exhibit newExhibit;
        newExhibit.setName(name);
        newExhibit.setPrice(price);

        exhibitDir.push_back(newExhibit);

        logging test;
        test.logEvent("Exhibit Created ");

}

inline void directory::autoCreateDiscount(string name, int percent)
{
        discount tmp;
        tmp.setName(name);
        tmp.setPercent(percent);



        allDiscounts.push_back(tmp);
        logging test;
        test.logEvent("Discount Created ");
}

inline user directory::searchUser(string username)
{
        user tmp;
```

```
	int dirSize = userDir.size();
	for (int x = 0; x < dirSize; x++) {
		string tmpUsername = userDir[x].returnUsername();
		if (tmpUsername == username) {
			tmp = userDir[x];
			return tmp;
		}
	}
}

inline int directory::searchUserReturnIndex(string username)
{
	user tmp;
	int dirSize = userDir.size();
	for (int x = 0; x < dirSize; x++) {
		string tmpUsername = userDir[x].returnUsername();
		if (tmpUsername == username) {
			return x;
		}
	}
}

inline discount directory::searchDiscount(string code)
{
	discount tmp;
	int dirSize = allDiscounts.size();
	for (int x = 0; x < dirSize; x++) {
		string tmpName = allDiscounts[x].returnName();
		if (tmpName == code) {
			tmp = allDiscounts[x];
			return tmp;
		}
	}
}

inline exhibit directory::searchExhibit(string exhibitName)
{
	exhibit tmp;
	int dirSize = exhibitDir.size();
	for (int x = 0; x < dirSize; x++) {
		string tmpUsername = exhibitDir[x].returnName();
		if (tmpUsername == exhibitName) {
			tmp = exhibitDir[x];
			return tmp;
		}
	}
}

inline void directory::manageUsers()
{
	string choice;
	cout << "What would you like to do?" << endl;
	cout << "1.) Print Users\n";
	cout << "2.) Edit User\n";
	cout << "3.) Delete User\n";
	cout << "4.) Create User\n";
```

```
        cin >> choice;
        if (choice == "1") {
                printUsers();
        }
        else if (choice == "2") {
                user tmpUser;
                string userName;
                cout << "what user would you like to edit: ";
                cin >> userName;
                tmpUser = searchUser(userName);
                updateUser(tmpUser);
                cout << "Account successfully updated" << endl;
        }
        else if (choice == "3") {
                string userDelete;
                cout << "enter user to be deleted" << endl;
                cin >> userDelete;
                deleteUser(userDelete);
        }
        else if (choice == "4") {
                createUser();
        }
        else { cout << "invalid input"; }
}

inline void directory::manageExhibits()
{
        string choice;
        cout << "What would you like to do?" << endl;
        cout << "1.) Print Exhibits\n";
        cout << "2.) Edit Exhibit\n";
        cout << "3.) Delete Exhibit\n";
        cout << "4.) Create Exhibit\n";
        cin >> choice;
        if (choice == "1") {
                printExhibits();
        }
        else if (choice == "2") {
                //need edit user function manage profile?
        }
        else if (choice == "3") {
                string exhibitToDelete;
                cout << "enter exhibit to be deleted" << endl;
                cin >> exhibitToDelete;
                deleteExhibit(exhibitToDelete);
        }
        else if (choice == "4") {
                createExhibit();
        }
        else { cout << "invalid input"; }
}

//takes user by reference then performs action based on input
inline void directory::menuAction(user &currentUser, bool& loggedStatus)
{
        string choice;
```

```
if (currentUser.returnRole() == "customer") {
        cout << "What would you like to do?" << endl;
        cout << "1.) Generate Order\n";
        cout << "2.) Manage Account\n";
        cout << "3.) Log Out\n";
        cin >> choice;
        if (choice == "1") {
                currentUser.generatePlan(exhibitDir, allDiscounts);
        }
        else if (choice == "2") {
                currentUser.manageProfile();
        }
        else if (choice == "3") {
                //will cause system start to exit loop
                loggedStatus = false;
                cout << "LOG OUT SUCCESSFUL" << endl;
        }
        else { cout << "invalid input"; }
}
else if (currentUser.returnRole() == "manager") {
        cout << "What would you like to do?" << endl;
        cout << "1.) Manage Users\n";
        cout << "2.) Manage Exhibits\n";
        cout << "3.) Add Discount\n";
        cout << "4.) Log Out\n";
        cin >> choice;
        if (choice == "1") {
                manageUsers();
```

```
		}
		else if (choice == "2") {
			manageExhibits();
		}
		else if (choice == "3") {
			createDiscount();
		}
		else if (choice == "4") {
			//will cause system start to exit loop
			loggedStatus = false;
			cout << "LOG OUT SUCCESSFUL" << endl;
		}
		else { cout << "invalid input"; }
	}
	else if (currentUser.returnRole() == "keeper") {
		cout << "What would you like to do?" << endl;
		cout << "1.) Add Exhibit\n";
		cout << "2.) Remove Exhibit\n";
		cout << "3.) Add Details\n";
		cout << "4.) Log Out\n";
		cin >> choice;
		if (choice == "1") {
			createExhibit();
		}
		else if (choice == "2") {
			string exhibitToDelete;
			cout << "input exhibit to delete";
			cin >> exhibitToDelete;
		deleteExhibit(exhibitToDelete);
	}
	else if (choice == "3") {
		//need to be able to add details
		string exhibitToChange;
		exhibit tmpExhibit;

		cout << "eneter exhibit you wish to add details to:  ";
		cin >> exhibitToChange;
		tmpExhibit = searchExhibit(exhibitToChange);

		string details;
		cout << "enter correct details for the exhibit:	";
		cin.ignore();
		getline(std::cin, details);
		tmpExhibit.setDetails(details);

		updateExhibit(tmpExhibit);
		cout << "exhibit updated" << endl;

		//delete this
		cout << tmpExhibit.returnDetails() << endl;

	}
	else if (choice == "4") {
		//will cause system start to exit loop
		loggedStatus = false;
		cout << "LOG OUT SUCCESSFUL" << endl;
```

```
                }
                else { cout << "invalid input"; }
        }
}
inline void directory::deleteExhibit(string exhibitName)
{
        exhibit tmp;
        int dirSize = userDir.size();
        for (int x = 0; x < dirSize; x++) {
                string tmpName = exhibitDir[x].returnName();
                if (tmpName == exhibitName) {
                        exhibitDir.erase(exhibitDir.begin() + x);
                        cout << "directory size: " << userDir.size() << endl;
                        return;
                }
        }

}

inline void directory::deleteUser(string username)
{
        user tmp;
        int dirSize = userDir.size();
        for (int x = 0; x < dirSize; x++) {
                string tmpUsername = userDir[x].returnUsername();
                if (tmpUsername == username) {
                        userDir.erase(userDir.begin() + x);
                        cout << "directory size: " << userDir.size() << endl;
```

```
                    return;
                }
            }

}



//prints all users in the vector
inline void directory::printUsers()
{
        int dirSize = userDir.size();
        for (int x = 0; x < dirSize; x++) {
                cout << userDir[x].returnUsername() << endl;
                cout << userDir[x].returnRole() << endl;
                cout << endl;
        }
}

inline void directory::printExhibits()
{
        int dirSize = exhibitDir.size();
        for (int x = 0; x < dirSize; x++) {
                cout << exhibitDir[x].returnName() << endl;
                cout << exhibitDir[x].returnPrice() << endl;
                cout << endl;
        }
}
```

```
inline void directory::updateUser(user tmp)
{
        //iterates through directoiry of users and finds the user in vecor and updates
        int dirSize = userDir.size();
        for (int x = 0; x < dirSize; x++) {
                string tmpUsername = userDir[x].returnUsername();
                if (tmpUsername == tmp.returnUsername()) {
                        userDir[x] = tmp;
                }
        }
}

inline void directory::updateExhibit(exhibit tmp)
{
        int dirSize = exhibitDir.size();
        for (int x = 0; x < dirSize; x++) {
                string tmpUsername = exhibitDir[x].returnName();
                if (tmpUsername == tmp.returnName()) {
                        exhibitDir[x] = tmp;
                }
        }
}



inline bool directory::userExists(string username)
{
        string tmpUsername;
        int dirSize = userDir.size();
        for (int x = 0; x < dirSize; x++) {
                tmpUsername = userDir[x].returnUsername();
                if (tmpUsername == username) {
                        return true;
                }
        }
        return false;
}

inline vector<exhibit> directory::returnExhibitDir()
{
        return exhibitDir;
}

//functions for users

inline void user::generatePlan(vector<exhibit> dir, vector<discount> allDiscounts)
{

        vector<exhibit> myExList;
        exhibit exhibitToSet;
        API credit;

        credit.creditValidator();
        cout << "Please eneter your budget(minimum of 15$ is reccomended):    ";
        cin >> budget;
```

```
int arrivalTime;
cout << "what time will you be arriving to the park?: (0-23) EX: 2 = 0200: ):          ";
cin >> arrivalTime;

if (arrivalTime < 0 || arrivalTime > 23) {
        cout << "you have enetered an invalid time, please enter a number between 0-23";
        cin >> arrivalTime;
}
mySchedule.setArrival(arrivalTime);

bool favSet = false;

while (favSet != true) {
        string input;
        int favNum;

        cout << "search for an exhibit you would like to see: ";
        cin >> input;

        //use found to see if user searched correctly
        bool found = false;
        int dirSize = dir.size();
        for (int x = 0; x < dirSize; x++) {
                string tmpUsername = dir[x].returnName();
                if (tmpUsername == input) {
                        exhibitToSet = dir[x];
                        found = true;
```

```
				}
			}
			//bool exists checks if the exhibit is there or not. if not go to next iteration
			cout << "name of exhibit you searched: " << exhibitToSet.returnName() << endl;
			cout << "How important is it you see this exhibit?(1-5) ";
			cin >> favNum;
			exhibitToSet.setFav(favNum);

			myExList.push_back(exhibitToSet);

			char finished;
			cout << "Are you finished choosing exhibits to see? (y/n)";
			cin >> finished;
			if (finished == 'y') {
				favSet = true;
			}

		}

		//now we will generate a schedule based on favourablillity
		//need to loop through fav 1-5 and subtract from budget
		//also need to check that the exhibit schedule and user schedule line up
		//already have mySchedule
		//these will iterate through user preference and schedule more preferred exhibits first then
workt to less preferred
		double happinessRating;
		int tmpCounter;
		double tmpBudget = budget;
```

```
	tmpCounter = checkSchedule(5, myExList);

	// 5 * number of 5 fav levels scheduled
	happinessRating = tmpCounter * 5;

	tmpCounter = checkSchedule(4, myExList);
	//4 * number of fav levels scheduled
	happinessRating += (tmpCounter * 4);

	tmpCounter = checkSchedule(3, myExList);
	happinessRating += (tmpCounter * 3);

	tmpCounter = checkSchedule(2, myExList);
	happinessRating += (tmpCounter * 2);

	tmpCounter = checkSchedule(1, myExList);
	happinessRating += (tmpCounter * 1);



	cout << "If special pass was purchased, please eneter code for discount (for testiing code is:
pass) :";
	string codeName;
	cin >> codeName;
	double discountPercent = 0;
	//search vector of discounts to see if exists
	int dirSize = allDiscounts.size();
	for (int x = 0; x < dirSize; x++) {
        string tmpName = allDiscounts[x].returnName();

        if (tmpName == codeName) {

                discountPercent = allDiscounts[x].returnPercent();

                cout << "discount found, discount percent: " << discountPercent << endl;

        }

}


//lets user know if discount didnt exist

if (discountPercent == 0) {

        cout << "No Discount Found For That Code" << endl;

}


//must calculate money spent with discount

//tmp budget carries original budget amount

double moneySpent;

moneySpent = tmpBudget - budget;


if (discountPercent != 0) {

        //discount percent will now hold actual percent of discount

        discountPercent = 100 - discountPercent;

        discountPercent = discountPercent / 100.00;


        //moneySpent is now correct

        moneySpent = moneySpent * discountPercent;


        //making budget correct

        budget = tmpBudget - moneySpent;

}
```

```
		//calculate final hapiness rating
		happinessRating = happinessRating - moneySpent;


		cout << "Your happiness score for this schedule is: " << happinessRating << endl;
		cout << "Your Schedule for the day" << endl;
		mySchedule.printSchedule();


		//log event
		//logEvent("Plan Generated");
		logging test;
		test.logEvent("Plan Generated ");


	}
	inline int user::checkSchedule(int favNum, vector<exhibit> ExList)
	{
		//loops and checks the favorbillity or number that was passed.
		int count = 0;
		exhibit tmp;
		int dirSize = ExList.size();
		for (int x = 0; x < dirSize; x++) {
			int tmpFav = ExList[x].returnFav();
			if (tmpFav == favNum) {
				tmp = ExList[x];


				//checks if user can affird to schedule this exhibit before scheduling
				double budgetTest = budget - tmp.returnPrice();
				if (budgetTest >= 0) {
```

```
                    string exhibitName = tmp.returnName();

                    setSchedules(mySchedule, tmp.returnSchedule(), exhibitName);

                    count++;

                    budget = budget - tmp.returnPrice();


                }

                else { return count; }

            }

        }

        return count;


}


//will check schedule compadibillity then set time

inline void user::setSchedules(schedule userSchedule, schedule ExhibitSchedule, string name)

{

        for (int i = 0; i < 24; i++) {


            if (userSchedule.checkTime(i) == false && ExhibitSchedule.checkTime(i) == false) {


                mySchedule.addTime(i);

                mySchedule.addTimeName(name, i);

                break;

            }


        }


}
```

```
inline vector<exhibit> user::setFav(vector<exhibit>)

{

		return vector<exhibit>();

}


string user::returnUsername()

{

		return username;

}


inline string user::returnPassword()

{

		return password;

}


inline string user::returnRole()

{

		return role;

}


inline void user::setUsername(string str)

{

		username = str;

}


inline void user::setPassword(string str)

{

		password = str;
}

inline void user::setBudget(double num)
{
	budget = num;
}

inline void user::setArrival(int time)
{
	mySchedule.setArrival(time);
}

inline void user::setRole(string newRole)
{
	role = newRole;
}

inline void user::manageProfile()
{
	string choice;
	cout << "what would you like to change? :" << endl;
	cout << "1.) change budget" << endl;
	cout << "2.) change password" << endl;
	cin >> choice;
	if (choice == "1") {
		double newBudget;
		cout << "please enter a new budget amount: ";
		cin >> newBudget;
		setBudget(newBudget);

		cout << "new budget amount set" << endl;

	}

	else if (choice == "2") {

		string newPassword;

		cout << "please enter a new password: ";

		cin >> newPassword;

		setPassword(newPassword);

		cout << "new password has been set" << endl;

	}

	else { cout << "invalid input" << endl; }


}



//exhibit functions


exhibit::exhibit()

{

	name = "null";

	details = "details have not been set";

	price = 100000;

	exhibitSchedule;

	favorabillity = 0;

}


inline void exhibit::setDetails(string text)

{
	details = text;
}

inline void exhibit::setPrice(double num)
{
	price = num;
}

inline void exhibit::setName(string text)
{
	name = text;
}

inline void exhibit::setFav(int num)
{
	favorabillity = num;
}

inline string exhibit::returnName()
{
	return name;
}

inline double exhibit::returnPrice()
{
	return price;
}
```

```
inline int exhibit::returnFav()

{

	return favorabillity;

}


inline schedule exhibit::returnSchedule()

{

	return exhibitSchedule;

}


inline string exhibit::returnDetails()

{

	return details;

}



//schedule functions



schedule::schedule()

{



	for (int i = 0; i < 24; i++) {

		times[i] = false;

		timeName[i] = " ";

	}
```

```
		//this is hard coded for testing purposes for phase two
		//remove once added apropriate functionallity
		//srand(time(NULL));
		//times[rand() % 24] = true;
		//times[rand() % 24] = true;
		//times[rand() % 24] = true;
		//times[rand() % 24] = true;
		//times[rand() % 24] = true;
}

inline void schedule::addTime(int x)
{
		if (times[x] == true) {
				cout << "ERROR time taken" << endl;
				return;
		}
		else { times[x] = true; }

}

inline void schedule::addTimeName(string name, int x)
{
		timeName[x] = name;
}

inline void schedule::printSchedule()
{
		for (int i = 0; i < 24; i++) {
```

```
		cout << i << "00:  ";

		cout << times[i] << "  " << timeName[i] << endl;

	}


}


inline bool schedule::checkTime(int x)

{

	return times[x];

}


inline void schedule::setArrival(int time)

{

	for (int i = 0; i < time; i++) {

		times[i] = true;

	}

}


//functions for logging service

//make these stand alone functions

//ofstream myfile and logfile is the name of the txt file

inline void logging::getDate(ofstream & myfile)

{

	time_t tt;


	struct tm * ti;


	// Applying time()
```

```
        time(&tt);

        // Using localtime()
        ti = localtime(&tt);

        myfile << "DATETIME: " << asctime(ti);
        cout << "DATETIME: " << asctime(ti);
        myfile.close();
}
void logging::logEvent(string str)
{
        ofstream myfile;

        myfile.open("../log/log_DATETIME.txt", std::ofstream::app);

        myfile << setw(20) << str << "                              ";
        cout << setw(20) << str << "                                ";

        getDate(myfile);
        myfile.close();
}



//input scenario functions
```

```
//NEEDS DONE



#endif
```

```
#pragma warning(disable : 4996) //_CRT_SECURE_NO_WARNINGS

/*------------------------------------------------------------------
System Name: Zoinkers!
Artifact Name: Zoinkers.cpp
Create Date: May 1, 2018
Author: Matt Skeins
Version: 3.0
-----------------------------------------------------------------*/


#include <iostream>
#include <ctime>
#include <stdlib.h>
#include "zoinkers.h"
#include <vector>
```

```
}
```

```
#include <fstream>
#include <string>
using namespace std;

//global functions
ifstream in;

int main() {



        testing testClient;
        testClient.start();
        //bool login = testClient.logIn();

        //if (login == true) {

        testClient.systemStart();

        //}
        //else {}

        //zoinkers mainSystem;
        //mainSystem.systemStart();
        cout << "Now exiting the system" << endl;

        system("PAUSE");
        return 0;
}
```

```
void testing::start()
{
	in.open("xt.txt");
	if (in.fail())
	{
		cout << "\Error!";
		in.clear();
	}
}

void directory::INPUTcreateExhibit()
{
	exhibit newExhibit;
	string name;
	double price;

	cout << "enter name for exhibit: \n";
	//fileinput
	string inStr;
	getline(in, inStr);
	//price = stoi(inStr);
	name = inStr;
	//cin >> name;

	cout << "enter price for exhibit: \n";
	getline(in, inStr);
```

```
        price = atof(inStr.c_str());
        //cin >> price;

        newExhibit.setName(name);
        newExhibit.setPrice(price);

        exhibitDir.push_back(newExhibit);
        cout << "Number of exhibits: " << exhibitDir.size() << endl;

        //log event
        //logEvent("Exhibit Created");
        logging test;
        test.logEvent("Exhibit Created ");
}

 void user::INPUTgeneratePlan(vector<exhibit> dir)
{

        string inStr;

        vector<exhibit> myExList;
        exhibit exhibitToSet;
        cout << "Please eneter your budget(minimum of 15$ is reccomended):    ";

        //file input
        getline(in, inStr);
        budget = atof(inStr.c_str());
        //cin >> budget;
```

```
int arrivalTime;

cout << "what time will you be arriving to the park?: (0-23) EX: 2 = 0200: ):          ";

//file input

getline(in, inStr);

arrivalTime = atoi(inStr.c_str());


//cin >> arrivalTime;

mySchedule.setArrival(arrivalTime);


bool favSet = false;


while (favSet != true) {

        string input;

        int favNum;


        cout << "search for an exhibit you would like to see: ";


        //file input

        getline(in, inStr);

        input = inStr;

        //cin >> input;


        //use found to see if user searched correctly

        bool found = false;

        int dirSize = dir.size();

        for (int x = 0; x < dirSize; x++) {

                string tmpUsername = dir[x].returnName();
        if (tmpUsername == input) {
                exhibitToSet = dir[x];
                found = true;
        }
}
//bool exists checks if the exhibit is there or not. if not go to next iteration
cout << "name of exhibit you searched: " << exhibitToSet.returnName() << endl;
cout << "How important is it you see this exhibit?(1-5) ";

//file input
getline(in, inStr);
favNum = atoi(inStr.c_str());
//cin >> favNum;

exhibitToSet.setFav(favNum);

myExList.push_back(exhibitToSet);

char finished;
cout << "Are you finished choosing exhibits to see? (y/n)";

//file input
getline(in, inStr);
finished = inStr[0];
//cin >> finished;

if (finished == 'y') {
        favSet = true;
```

```
		}

	}


	//these will iterate through user preference and schedule more preferred exhibits first then
workt to less preferred

	double happinessRating;

	int tmpCounter;

	double tmpBudget = budget;


	tmpCounter = checkSchedule(5, myExList);


	// 5 * number of 5 fav levels scheduled

	happinessRating = tmpCounter * 5;


	tmpCounter = checkSchedule(4, myExList);

	//4 * number of fav levels scheduled

	happinessRating += (tmpCounter * 4);


	tmpCounter = checkSchedule(3, myExList);

	happinessRating += (tmpCounter * 3);


	tmpCounter = checkSchedule(2, myExList);

	happinessRating += (tmpCounter * 2);


	tmpCounter = checkSchedule(1, myExList);

	happinessRating += (tmpCounter * 1);


	//must calculate money spent
```

```
        tmpBudget = tmpBudget - budget;
        //calculate final hapiness rating
        happinessRating = happinessRating - tmpBudget;

        cout << "Your happiness score for this schedule is: " << happinessRating << endl;
        cout << "Your Schedule for the day" << endl;
        mySchedule.printSchedule();

        //log event
        //logEvent("Plan Generated");
        logging test;
        test.logEvent("Plan Generated ");

}

//NEEDS CHANGED
 void user::INPUTmanageProfile()
{
        string choice;
        string inStr;
        cout << "what would you like to change? :" << endl;
        cout << "1.) change budget" << endl;
        cout << "2.) change password" << endl;

        //starts line 21
        getline(in, inStr);
        choice = inStr;
	if (choice == "1") {
		double newBudget;
		cout << "please enter a new budget amount: ";

		getline(in, inStr);
		newBudget = atof(inStr.c_str());

		setBudget(newBudget);
		cout << "new budget amount set" << endl;
	}
	else if (choice == "2") {
		string newPassword;
		cout << "please enter a new password: ";

		//line 25 start
		getline(in, inStr);
		newPassword = inStr;

		setPassword(newPassword);
		cout << "new password has been set" << endl;
	}
	else { cout << "invalid input" << endl; }
}


 void directory::INPUTmanageUsers()
{
	string choice;
```

```
cout << "What would you like to do?" << endl;

cout << "1.) Print Users\n";

cout << "2.) Edit User\n";

cout << "3.) Delete User\n";

cout << "4.) Create User\n";


string inStr;

//starting line 30

getline(in, inStr);

choice = inStr;

if (choice == "1") {

            printUsers();

}

else if (choice == "2") {

            //starts line 32

            //ends 33

            user tmpUser;

            string userName;

            cout << "what user would you like to edit: ";


            getline(in, inStr);

            userName = inStr;


            tmpUser = searchUser(userName);

            updateUser(tmpUser);

            cout << "Account successfully updated" << endl;

}

else if (choice == "3") {
```

```
                //starts line 35 text.txt
                string userDelete;
                cout << "enter user to be deleted" << endl;
                getline(in, inStr);
                userDelete = inStr;
                deleteUser(userDelete);
        }
        else if (choice == "4") {
                //starts line 38
                INPUTcreateUser();
        }
        else { cout << "invalid input"; }
}

 void directory::INPUTcreateUser()
{

        user newUser;
        string username;
        string password;

        cout << "please enter a username: \n";
        string inStr;
        //fileinput
        //starts line 39
        getline(in, inStr);
        //price = stoi(inStr);
        username = inStr;
```

```
//cin >> username;




newUser.setUsername(username);

cout << "please enter a password: \n" << endl;;

//fileinput

//line 40

getline(in, inStr);

//price = stoi(inStr);

password = inStr;

//cin >> password;

newUser.setPassword(password);




newUser.setRole("keeper");



userDir.push_back(newUser);


cout << "\n username:   " << userDir.at(0).returnUsername() << "\n password:     " <<
userDir.at(0).returnPassword() << endl;

cout << "account creation successful" << endl;


cout << "current size of user directory:   " << userDir.size() << endl;


//log event
```

```
        //logEvent("User Created");
        logging test;
        test.logEvent("User Created ");
}


 void directory::INPUTmanageExhibits()
{
        string choice;
        string inStr;
        cout << "What would you like to do?" << endl;
        cout << "1.) Print Exhibits\n";
        cout << "3.) Delete Exhibit\n";
        cout << "4.) Create Exhibit\n";
        getline(in, inStr);
        choice = inStr;
        if (choice == "1") {
                //starts line 42
                printExhibits();
        }
        else if (choice == "3") {
                string exhibitToDelete;
                cout << "input exhibit to delete";
                //input from file for exhibit deletion
                //starts line 44
                getline(in, inStr);
                exhibitToDelete = inStr;
                deleteExhibit(exhibitToDelete);
	}
	else if (choice == "4") {
		//starts line 47
		//creates zebra price 7
		INPUTcreateExhibit();
	}
	else { cout << "invalid input"; }
}

 void directory::INPUTcreateDiscount()
{
	discount newDiscount;
	string newName;
	int disPercent;
	string inStr;

	cout << "eneter code for new discount: ";
	//line 51
	getline(in, inStr);
	newName = inStr;

	cout << "eneter percent of discount: ";

	//test.txt line 52
	getline(in, inStr);
	disPercent = atof(inStr.c_str());

	newDiscount.setName(newName);
```

```
        newDiscount.setPercent(disPercent);


        allDiscounts.push_back(newDiscount);

        cout << "Number of discounts: " << allDiscounts.size() << endl;


        logging test;

        test.logEvent("Discount Created ");

}




//NEEDS DONE

void directory::INPUTmenuAction(user &currentUser, bool &loggedStatus)

{


        string inStr;

        string choice;


        if (currentUser.returnRole() == "customer") {

                cout << "What would you like to do?" << endl;

                cout << "1.) Generate Order\n";

                cout << "2.) Manage Account\n";

                cout << "3.) Log Out\n";

                //starts line 14 of test.txt

                //choice 2 starts 20

                getline(in, inStr);

                choice = inStr;

                if (choice == "1") {

                        currentUser.INPUTgeneratePlan(exhibitDir);
```

```
		}
		else if (choice == "2") {
			currentUser.INPUTmanageProfile();
		}
		else if (choice == "3") {
			//will cause system start to exit loop
			loggedStatus = false;
			cout << "LOG OUT SUCCESSFUL" << endl;
		}
		else { cout << "invalid input"; }
	}
	else if (currentUser.returnRole() == "manager") {
		cout << "What would you like to do?" << endl;
		cout << "1.) Manage Users\n";
		cout << "2.) Manage Exhibits\n";
		cout << "3.) Add Discount\n";
		cout << "4.) Log Out\n";

		//starting line 29
		getline(in, inStr);
		choice = inStr;

		if (choice == "1") {
			INPUTmanageUsers();
		}
		else if (choice == "2") {
			INPUTmanageExhibits();
		}
```

```
        else if (choice == "3") {
                //starts line 50
                INPUTcreateDiscount();
        }
        else if (choice == "4") {
                //will cause system start to exit loop
                loggedStatus = false;
                cout << "LOG OUT SUCCESSFUL" << endl;
        }
        else { cout << "invalid input"; }
}
else if (currentUser.returnRole() == "keeper") {
        cout << "What would you like to do?" << endl;
        cout << "1.) Add Exhibit\n";
        cout << "2.) Remove Exhibit\n";
        cout << "3.) Add Details\n";
        cout << "4.) Log Out\n";
        //cin >> choice;
        //input from file
        getline(in, inStr);
        choice = inStr;
        if (choice == "1") {
                INPUTcreateExhibit();
        }
        else if (choice == "2") {
                string exhibitToDelete;
                cout << "input exhibit to delete";
                //input from file for exhibit deletion
        getline(in, inStr);

        exhibitToDelete = inStr;

        deleteExhibit(exhibitToDelete);

}

else if (choice == "3") {

        //need to be able to add details

        string exhibitToChange;

        exhibit tmpExhibit;


        cout << "eneter exhibit you wish to add details to:  ";

        getline(in, inStr);

        exhibitToChange = inStr;

        tmpExhibit = searchExhibit(exhibitToChange);


        string details;

        cout << "enter correct details for the exhibit:      ";


        //takes test details for details line 10

        getline(in, details);

        tmpExhibit.setDetails(details);


        updateExhibit(tmpExhibit);

        cout << "exhibit updated" << endl;


        //delete this

        cout << tmpExhibit.returnDetails() << endl;


}
```

```
                else if (choice == "4") {
                        //will cause system start to exit loop
                        loggedStatus = false;
                        cout << "LOG OUT SUCCESSFUL" << endl;
                }
                else { cout << "invalid input"; }
        }

}

user directory::INPUTLogin()
{
        user currentUser;
        user empty;
        string user;
        string password;

        string inStr;
        //fileinput
        getline(in, inStr);

        cout << "Please enter your username: \n";
        user = inStr;

        currentUser = searchUser(user);
        cout << currentUser.returnUsername() << "    " << currentUser.returnPassword() << "\n";
        cout << "please enter your password: \n";

        //fileinput

        getline(in, inStr);

        password = inStr;

        string currentPassword = currentUser.returnPassword();


        if (password != currentPassword) {

                cout << "Log in failed, incorrect password" << endl;


                logging test;

                test.logEvent("User Log In (failed) ");

                return empty;

        }

        else {

                cout << "LOG IN SUCCESSFUL" << endl;

                //log event

                //logEvent("User Log In (success)");

                logging test;

                test.logEvent("User Log In (success) ");

                return currentUser;

        }



}
```

```
zoinkers: zoinkers.o
        g++ zoinkers.o -o zoinkers

zoinkers.o: zoinkers.cpp zoinkers.h
```

```
        g++ -c zoinkers.cpp

clean:
        rm *.o zoinkers
```