CS3520 Final Project Design

Jack Dreifus & Alan Eng

The Community Center Management System is set to be designed in an object-oriented manner. Our design is composed of ten classes to meet the requirements of this project while maintaining a well-structured and clear codebase. Here's an overview of our classes:

1. User: This is the parent class for all users in the system. It is responsible for logging in and out of our system and maintaining the property of unique usernames. Also, it is responsible for editing user information.
2. FacilityManager: This is a child class of User. There is only one FacilityManager and their login information will be stored in a text file for the TA. This class will handle an abundance of validation to make sure everything runs smoothly in the system. The FacilityManager class will validate event requests as well as ticket requests by creating instances of those objects when proper payment is received and all event/ticket properties hold. When process requests this class gives priority to City Events. This class is also able to view the event schedule.
3. Organization and City: These are child classes of User. They both are able to request events and cancel them. Both classes can request events as long as they are not weddings. However, they will be at slightly different price points and have separate event limits. They can also view complete event schedules as well their event schedule.
4. Citizen: This is a child class of User and is composed of both residents and non-residents (each object will know if its a resident or not). Citizens have similar abilities to Organization's and City's, but they can also purchase and cancel tickets for public events. Citizens will be able to view the complete event schedule, their events, and their tickets.
5. Facility: This class holds a list of all confirmed events and can neatly display them for Users. This class will always collaborate with the Facility Manager to update its events.
6. Event: Each event can be public or private; private events do not have any tickets. Events will hold a list of all their attendees and will have many fields to describe their event: format, price, time etc. Events can neatly display themselves by showing all of their information as well as their attendees.
7. Ticket: A ticket has a price, a time, a name, and an event. Tickets will be able to display themselves in a ticket format.
8. ReservationRequest: This class will be able to reserve and cancel events as well as tickets. When a request is made this class will collaborate with the payment class to pay for the request as well as the FacilityManger to turn this request into an event/ticket.
9. Payment: This class is responsible for handling user payment information for specific requests.
10. TimeUtils: This will not be a class but is worth writing a note about. Time is a key factor for our reservation system, for that reason we will implement utility header and cpp files in order to track time for our system. Time will play a role in reservation priority, refund status, event scheduling etc.
11. FileIO: Again not a class but is worth a note about. In order to implement state persistence we will implement a file input/output header and cpp files to handle reading

and writing to an outfile logic. This will be used to save data both within the facility as well as specific user data.

CRC Cards:

Class: User (Parent)	
Responsibilities	**Collaborators**
- Login - Logout - Edit user information - Makes sure usernames are unique	

Class: FacilityManager (Child of user)	
Responsibilities	**Collaborators**
- Process refunds on request - Makes sure facility is not overbooked - Check for payments and booking by user - Ensure City reservations get priority - Views Schedule - Creates instance of event if valid event - Creates instance of ticket if valid ticket - When created a ticket makes sure event is not full - Refund tickets	ReservationRequest

Class: Organization (Child of user)		
Responsibilities		**Collaborators**
- Request Event - Cancel Event - Views Schedule		Facility

Class: City (Child of user)	
Responsibilities	**Collaborators**
- Request Event - Cancel Event - Has reservation priority - Views Schedule	Facility

Class: Citizen (Child of user)	
Responsibilities	**Collaborators**
- Request Event - Cancel Event - Knows if its a resident or non-resident - Purchase tickets - Cancel tickets	Facility

Class: Facility	
Responsibilities	**Collaborators**
- Hold a lists of confirmed events - Neatly displays the upcoming schedule showing the names of people on each reservation - Knows when it's open	FacilityManager

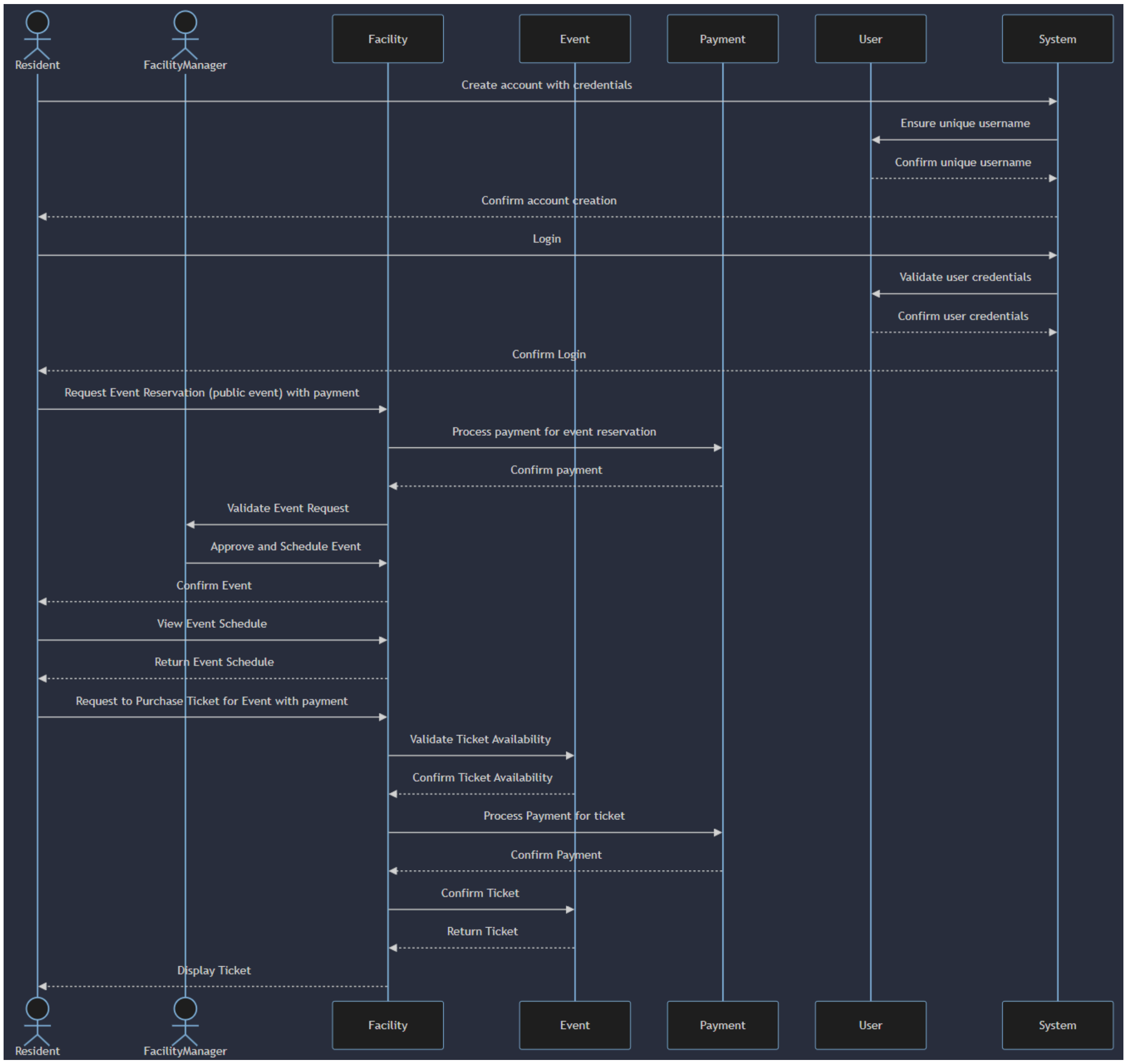
Class: Event	
Responsibilities	**Collaborators**
- Knows if its public or private - Knows who the event is for (residents and/or non res) - Knows it format - Holds a list of attendees - Has a details and basic information for users - Create tickets for this event - Store a waitlist for this event	Facility User

Class: Ticket	
Responsibilities	**Collaborators**
- Display ticket is a ticket format - Update ticket information(sells the ticket)	Facility User

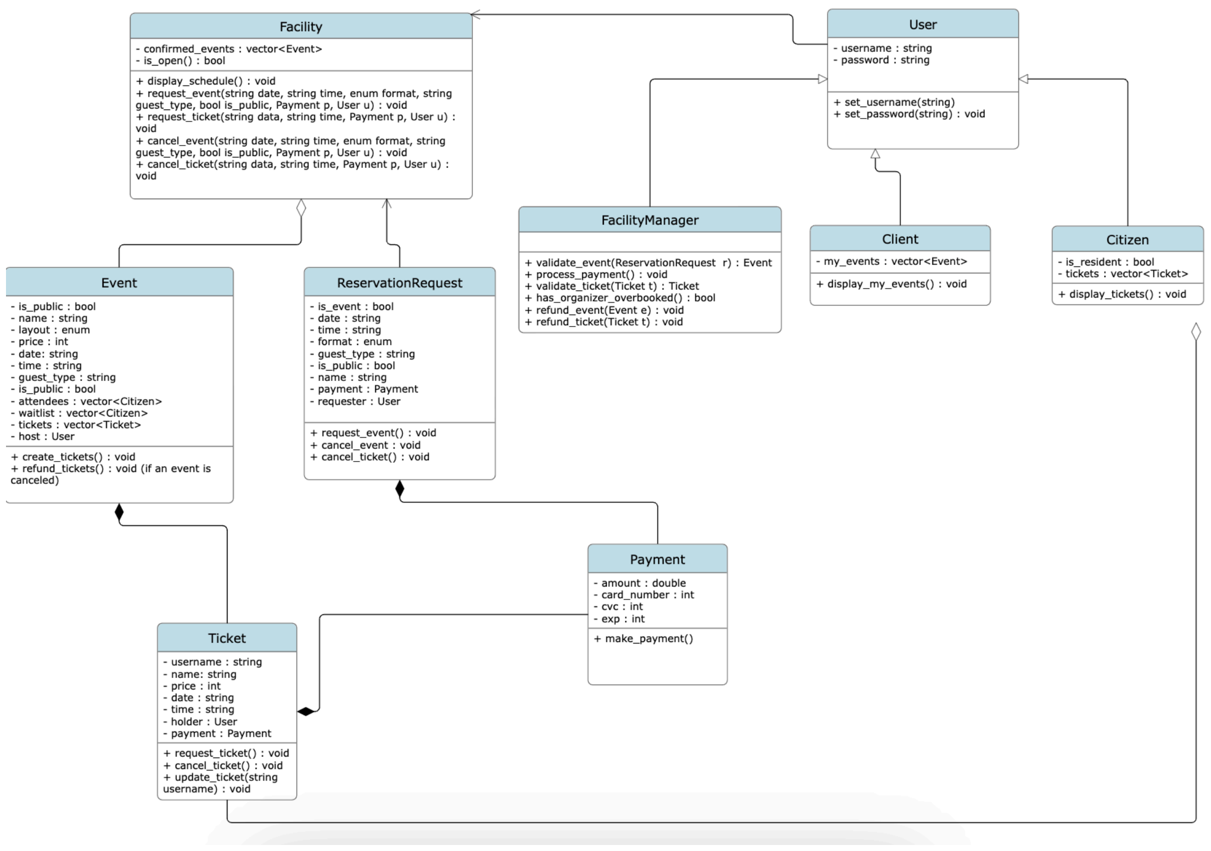
Class: ReservationRequest	
Responsibilities	**Collaborators**
- Request Event - Cancel Event - Purchase ticket - Cancel ticket - Waitlist ticket	FacilityManager User

Class: Payment	
Responsibilities	**Collaborators**
- Holds payment information - Pay amount for x	ReservationRequest

Sequence Diagram:



UML Diagram:


Facility
- confirmed_events : vector<Event>
- is_open() : bool
+ display_schedule() : void
+ request_event(string date, string time, enum format, string guest_type, bool is_public, Payment p, User u) : void
+ request_ticket(string data, string time, Payment p, User u) : void
+ cancel_event(string date, string time, enum format, string guest_type, bool is_public, Payment p, User u) : void
+ cancel_ticket(string data, string time, Payment p, User u) : void
User
- username : string
- password : string
+ set_username(string)
+ set_password(string) : void
FacilityManager
+ validate_event(ReservationRequest r) : Event
+ process_payment() : void
+ validate_ticket(Ticket t) : Ticket
+ has_organizer_overbooked() : bool
+ refund_event(Event e) : void
+ refund_ticket(Ticket t) : void
Client
- my_events : vector<Event>
+ display_my_events() : void
Citizen
- is_resident : bool
- tickets : vector<Ticket>
+ display_tickets() : void
Event
- is_public : bool
- name : string
- layout : enum
- price : int
- date: string
- time : string
- guest_type : string
- is_public : bool
- attendees : vector<Citizen>
- waitlist : vector<Citizen>
- tickets : vector<Ticket>
- host : User
+ create_tickets() : void
+ refund_tickets() : void (if an event is canceled)
ReservationRequest
- is_event : bool
- date : string
- time : string
- format : enum
- guest_type : string
- is_public : bool
- name : string
- payment : Payment
- requester : User
+ request_event() : void
+ cancel_event : void
+ cancel_ticket() : void
Payment
- amount : double
- card_number : int
- cvc : int
- exp : int
+ make_payment()
Ticket
- username : string
- name: string
- price : int
- date : string
- time : string
- holder : User
- payment : Payment
+ request_ticket() : void
+ cancel_ticket() : void
+ update_ticket(string username) : void

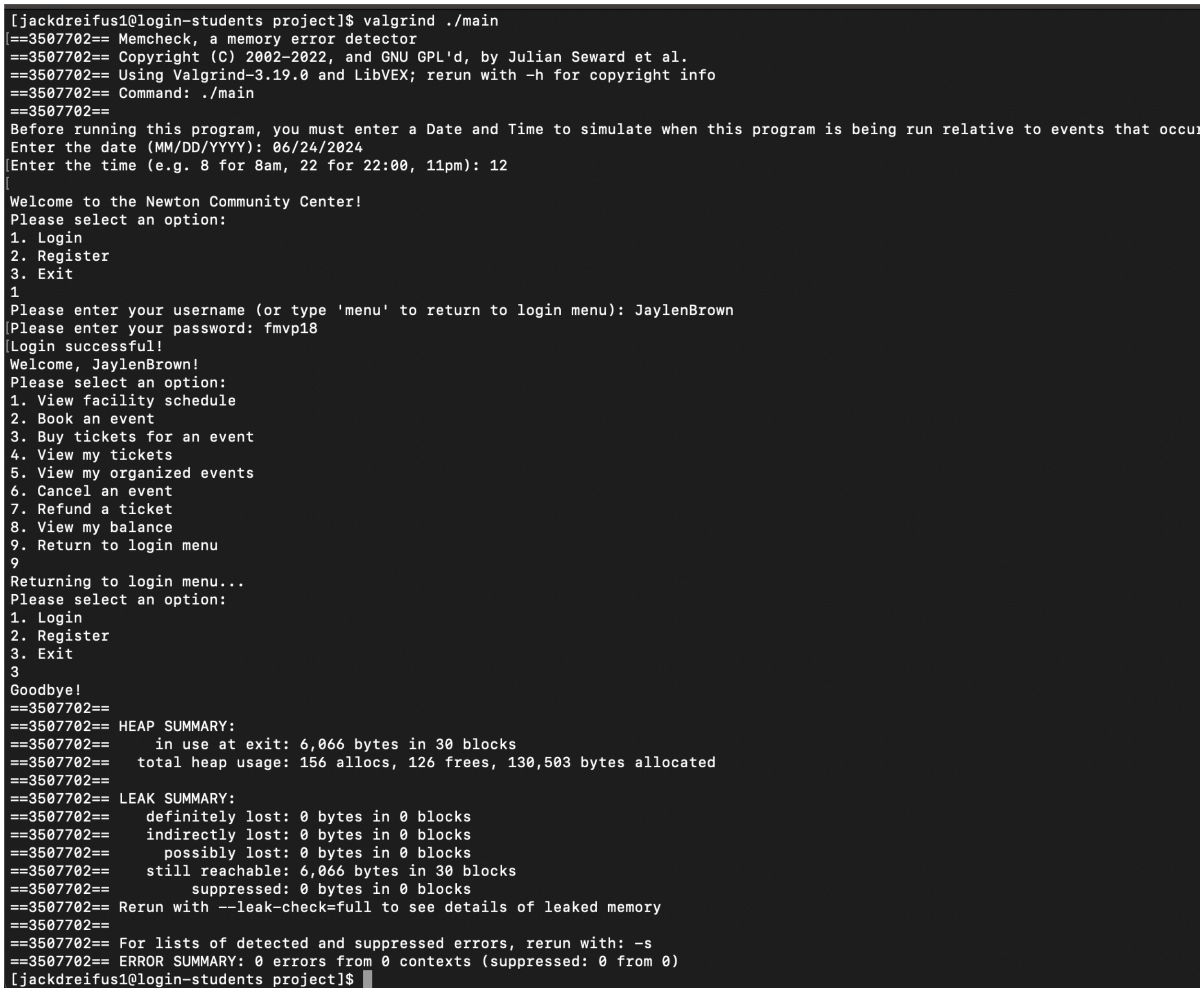
Data Structures:

- Vector: vectors are used extensively in our project due to their dynamic ability to manage a collection of items. Additionally, allowing access to vectors through indexing has been very beneficial to match usernames, for example to find the user who is the facility manager.
- Queue: a queue was implemented to handle the waitlist functionality for overbooked events. Since a queue uses FIFO ordering it makes managing the waitlist very straightforward as this aligns perfectly with required order of our waitlist.
- Shared_ptr: shared pointers are used in our project to safely handle the memory manager for all of our users and events in the facility system. Shared pointers also allow shared ownership over a single object which is extremely useful as there are multiple classes within this program that require access to the same objects.

Files:

- users.csv: holds the data for all of the previously created users including: username, password, and user-type.
- pending_events: holds the data for all events waiting for approval by the Facility Manager.
- confirmed_events: holds the data for all the events that have been approved by the Facility Manager. This file includes the tickets for each event too.

Valgrind SS:



```
[jackdreifus1@login-students project]$ valgrind ./main
==3507702== Memcheck, a memory error detector
==3507702== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==3507702== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==3507702== Command: ./main
==3507702==
Before running this program, you must enter a Date and Time to simulate when this program is being run relative to events that occu
Enter the date (MM/DD/YYYY): 06/24/2024
Enter the time (e.g. 8 for 8am, 22 for 22:00, 11pm): 12

Welcome to the Newton Community Center!
Please select an option:
1. Login
2. Register
3. Exit
1
Please enter your username (or type 'menu' to return to login menu): JaylenBrown
Please enter your password: fmvp18
Login successful!
Welcome, JaylenBrown!
Please select an option:
1. View facility schedule
2. Book an event
3. Buy tickets for an event
4. View my tickets
5. View my organized events
6. Cancel an event
7. Refund a ticket
8. View my balance
9. Return to login menu
9
Returning to login menu...
Please select an option:
1. Login
2. Register
3. Exit
3
Goodbye!
==3507702==
==3507702== HEAP SUMMARY:
==3507702==     in use at exit: 6,066 bytes in 30 blocks
==3507702==   total heap usage: 156 allocs, 126 frees, 130,503 bytes allocated
==3507702==
==3507702== LEAK SUMMARY:
==3507702==    definitely lost: 0 bytes in 0 blocks
==3507702==    indirectly lost: 0 bytes in 0 blocks
==3507702==      possibly lost: 0 bytes in 0 blocks
==3507702==    still reachable: 6,066 bytes in 30 blocks
==3507702==         suppressed: 0 bytes in 0 blocks
==3507702== Rerun with --leak-check=full to see details of leaked memory
==3507702==
==3507702== For lists of detected and suppressed errors, rerun with: -s
==3507702== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[jackdreifus1@login-students project]$
```