**Community Center Management System**

**Overview**

The Community Center Management System is designed to manage users, events, and reservations for a community center. It allows different types of users (City, Resident, Non-Resident, and Organization) to log in, create events, make reservations, and view schedules.

**Features**

* User management (create, login, logout)
* Event creation and management
* Reservation requests and approval
* View daily and weekly schedules
* View and manage user reservations

**How to Run**

**Prerequisites**

* Ensure you have a C++ compiler installed (e.g., g++ for GCC).

**Steps**

1. **Clone the repository**:

git clone <repository-url>  
cd community-center-management

1. **Compile the program**: Use the provided Makefile to compile the project. Simply run:

make

1. **Run the executable**:

./main

1. **Follow the on-screen menu** to navigate through the system.

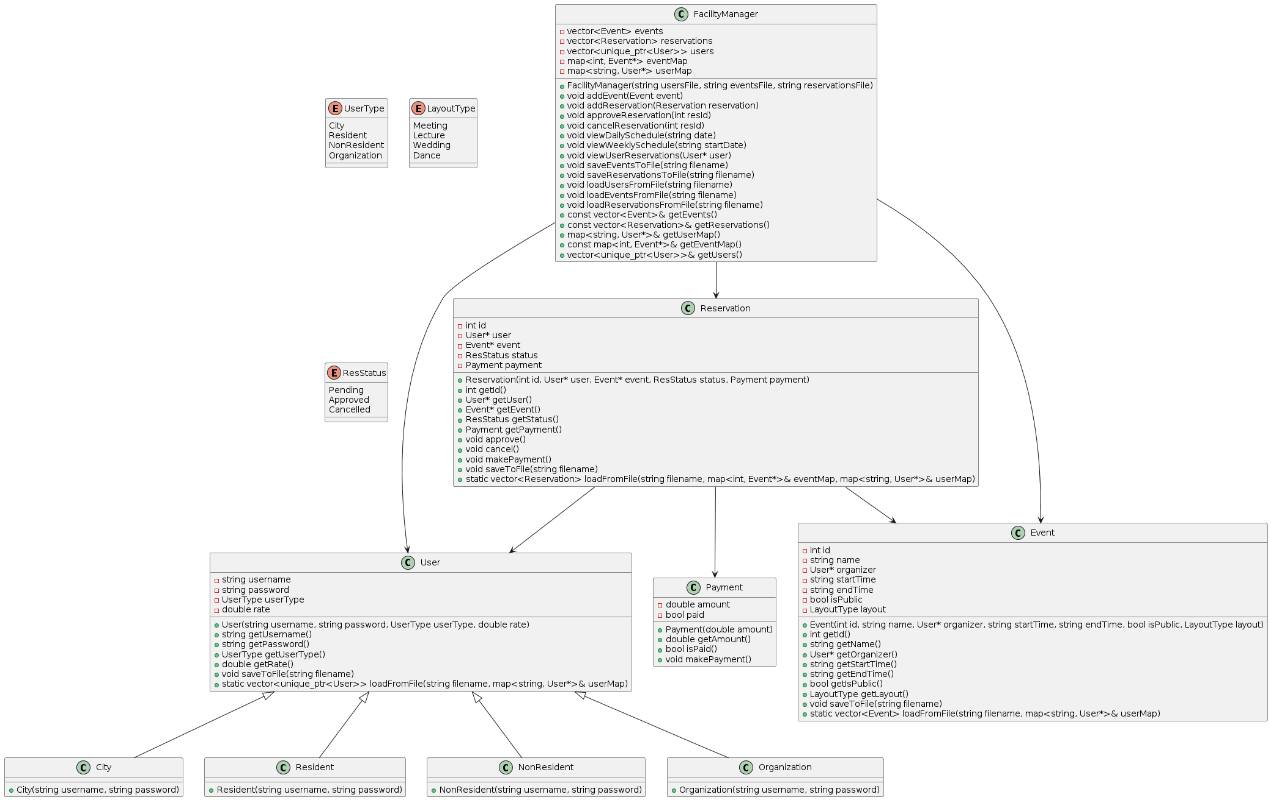
**Makefile**

The provided Makefile simplifies the build process. It automatically handles the compilation of all source files and links them into the final executable. Here is a brief explanation of each section in the Makefile:

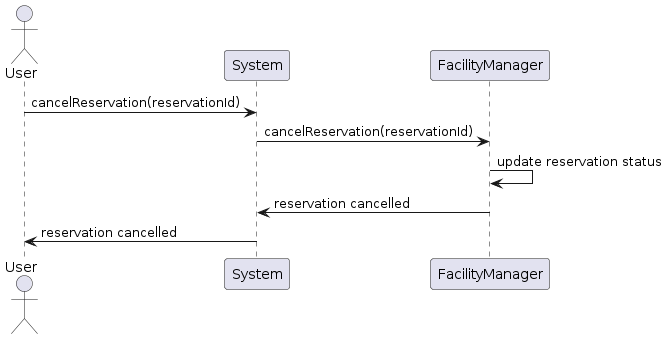
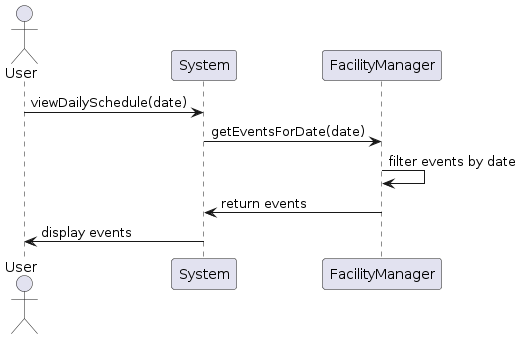
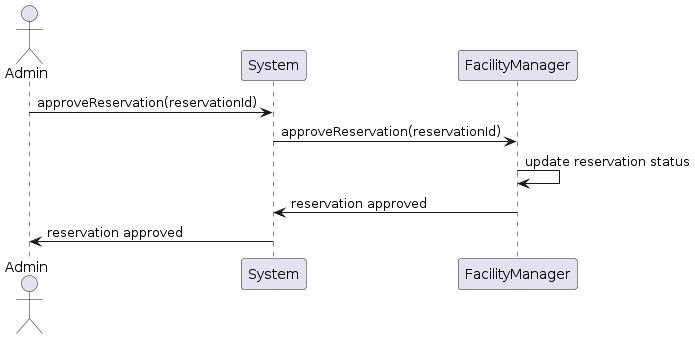
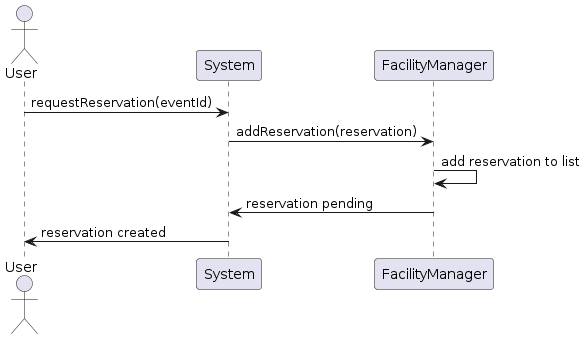
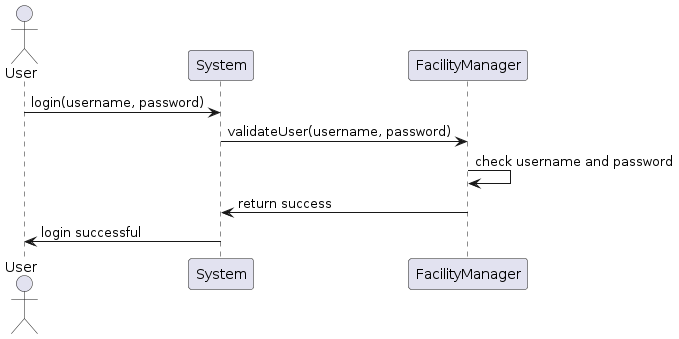
# Compiler  
CXX = g++  
​  
# Compiler flags  
CXXFLAGS = -std=c++11 -Wall -g  
​  
# Target executable  
TARGET = main  
​  
# Source files  
SOURCES = main.cpp user.cpp event.cpp reservation.cpp facility\_manager.cpp  
​  
# Object files  
OBJECTS = $(SOURCES:.cpp=.o)  
​  
# Header files  
HEADERS = user.h event.h reservation.h facility\_manager.h  
​  
# Default target  
all: $(TARGET)  
​  
# Rule to link the target executable  
$(TARGET): $(OBJECTS)  
$(CXX) $(CXXFLAGS) -o $@ $^  
​  
# Rule to compile source files into object files  
%.o: %.cpp $(HEADERS)  
$(CXX) $(CXXFLAGS) -c $< -o $@  
​  
# Clean up build files  
clean:  
rm -f $(TARGET) $(OBJECTS)  
​  
# Rebuild everything  
rebuild: clean all  
​  
.PHONY: all clean rebuild

**UML Diagrams**

**Class Diagram**



**Sequence Diagram**



**CRC Cards**

| **Class** | **Responsibilities** | **Collaborators** |
| --- | --- | --- |
| **User** | - Hold user details (username, password, user type, rate) | FacilityManager |
|  | - Save and load user data |  |
|  | - Provide methods for accessing user details |  |
|  |  |  |
| **City** | - Inherits from User | User |
|  | - Set specific rate for City users |  |
|  |  |  |
| **Resident** | - Inherits from User | User |
|  | - Set specific rate for Resident users |  |
|  |  |  |
| **NonResident** | - Inherits from User | User |
|  | - Set specific rate for Non-Resident users |  |
|  |  |  |
| **Organization** | - Inherits from User | User |
|  | - Set specific rate for Organization users |  |
|  |  |  |
| **Event** | - Hold event details (ID, name, organizer, start time, end time, is public, layout type) | User, FacilityManager |
|  | - Save and load event data |  |
|  | - Provide methods for accessing event details |  |
|  |  |  |
| **Reservation** | - Hold reservation details (ID, user, event, status, payment) | User, Event, Payment, |
|  | - Save and load reservation data | FacilityManager |
|  | - Provide methods for accessing reservation details |  |
|  | - Approve and cancel reservations |  |
|  | - Process payments |  |
|  |  |  |
| **Payment** | - Hold payment details (amount, paid status) | Reservation |
|  | - Process payment |  |
|  | - Provide methods for accessing payment details |  |
|  |  |  |
| **FacilityManager** | - Manage users, events, and reservations | User, Event, Reservation |
|  | - Load and save data for users, events, and reservations |  |
|  | - Provide methods for viewing schedules and reservations |  |
|  | - Approve and cancel reservations |  |

**Discussion of Design**

The design follows object-oriented principles to ensure modularity and separation of concerns. Each class is responsible for its own data and operations, promoting encapsulation.

The FacilityManager class acts as a central manager, coordinating interactions between users, events, and reservations. It handles the loading and saving of data from files, ensuring persistence across sessions.

By using inheritance, the User class is extended to create specific user types, each with its own rate. This design allows for easy expansion and maintenance of user types.

The program ensures data integrity and security by handling user authentication and authorization, particularly for reservation approvals which are restricted to city admin users.

**File Organization**

community-center-management/  
├── user.h  
├── user.cpp  
├── event.h  
├── event.cpp  
├── reservation.h  
├── reservation.cpp  
├── facility\_manager.h  
├── facility\_manager.cpp  
├── main.cpp  
├── users.txt  
├── events.txt  
└── reservations.txt

**Comments and Naming Conventions**

* **Header Blocks**: Each file starts with a header block describing its purpose.
* **Indentation**: Consistent 4-space indentation is used.
* **Naming Conventions**: Classes and methods use CamelCase, while variables use snake\_case.
* **Comments**: Adequate comments are provided to explain the functionality of methods and important code sections.