

Classroom Booking System

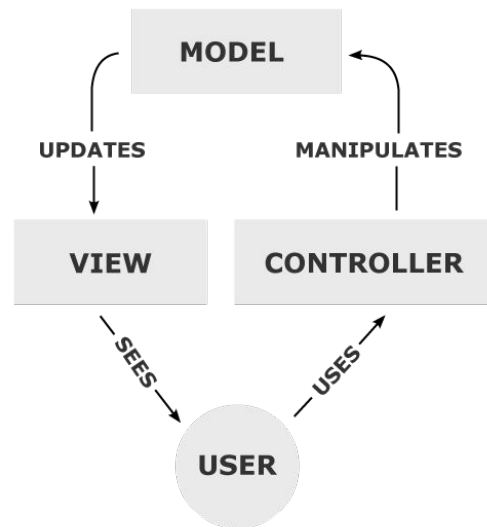
[CS223] Software Engineering Course Project

Ajat Prabha (B16CS002)
Saksham Banga (B16CS042)

Under Guidance of:
Dr. Chiranjoy Chattopadhyay

MVC Paradigm

- Provides clear separation of concerns for business logic and presentation.
- Helps make code modular and maintainable.
- Adding features is very convenient.



Templatized Code (down to core)

```
class Model<T>
```

- Enables object saving, deleting, etc very easy.
- Convenient file handling
- Provides easy search throughout the application
- One interface between all Classrooms and Database(File System)
- Thus the model class acts as the dynamic loader and the file handler for all types of classes

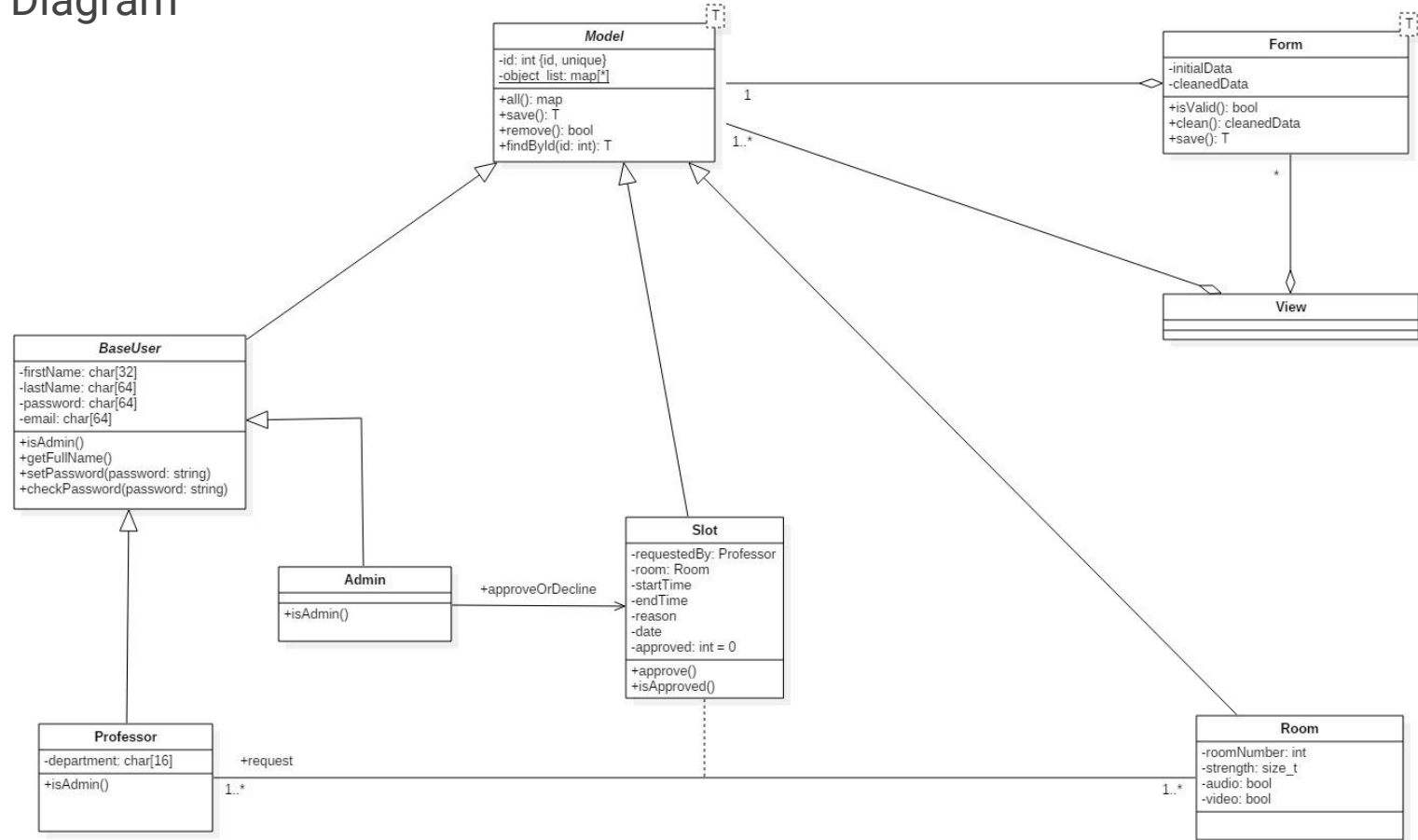
```
Admin("Admin", "user",  
"admin@iitj.ac.in",  
"admin@123").save();
```

Or

```
temp = new Slot(requestedBy, room,  
startTime, endTime, reason, approved);  
return temp->save();
```



Class Diagram



List of Use-cases implemented

Admin Panel View Use-cases

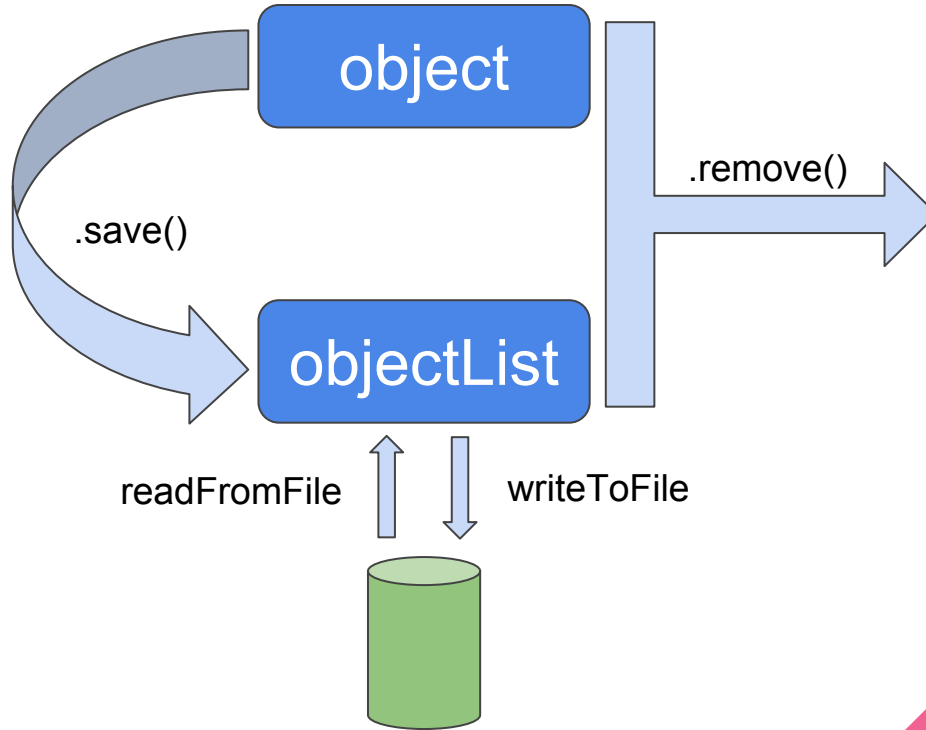
1. List Unseen Slot Requests
2. Display Room Details
3. Delete Room
4. Update Room Details
5. Create Professor/Admin
6. Delete Professor/Admin

Faculty Panel View Use-cases

1. Update User
2. Delete Slot
3. Update Slot
4. Create Slot
5. Slots status list
6. Empty Class List

Database Lifecycle

Model<T>



Templatized Code (down to core)

```
class ModelForm<T> : public Form
```

- Provides a dynamic and robust validation method
- Managing errors is easy
- Easily create and update Model<T> instances through this class

```
void RoomCreateUpdateForm::clean() {  
    if (!instance) {  
        for (auto &i : Room::all()) {  
            if (i.second.getRoomNumber() == roomNumber)  
                addError("A room with this room number  
                           already exists.");  
        }  
    }  
}
```

And

```
Room &RoomCreateUpdateForm::save() {  
    Room *temp;  
    temp = new Room(roomNumber, strength, audio, video);  
    if (instance) {  
        int id = instance->getId();  
        instance->remove();  
        temp->setId(id);  
    }  
    temp->save();  
    return *temp;  
}
```

Templatized Code (down to core)

Mixins

- Provides easy hooks to add objects into another class
- Methods can be overridden for custom behaviour
- Types:
 - `MultipleObjectMixin<T>`
 - `SingleObjectMixin<T>`

```
template<class T>
vector<T> &MultipleObjectMixin<T>::getQueryset() {
    vector<T> objects{};
    return objects;
}
```

And

```
template<class T>
T &SingleObjectMixin<T>::getObject() {
    return object;
}
```



Templatized Code (down to core)

Generic Class Based Views

- For easily providing **CRUD** features to views
- Provides a systematic approach to implementing views
- Views are templatized for easy addition of Model<T> instance logic
- Types:
 - CreateView<T>
 - UpdateView<T> uses SingleObjectMixin<T>
 - DetailView<T> uses SingleObjectMixin<T>
 - DeleteView<T> uses SingleObjectMixin<T>
 - ListView<T> uses MultipleObjectMixin<T>

Example:

```
class SlotNotificationListView : public ListView<Slot>

vector<Slot> &SlotNotificationListView::getQueryset() {
    for (auto &i : objectList) {
        if (i.second.getRequestedBy() ==
            static_cast<Professor &>(*context.user))
            objects.push_back(i.second);
    }
    return objects;
}

void SlotNotificationListView::display() {
    for (auto &slot : getQueryset()) {
        cout << slot.getId() << ". "
              << slot.getRoom().getRoomNumber();
    }
}
```

Design Patterns Used

Singleton Class

- **class** `Application`
 - Private constructor
- **class** `Controller`
 - Private constructor

Example:

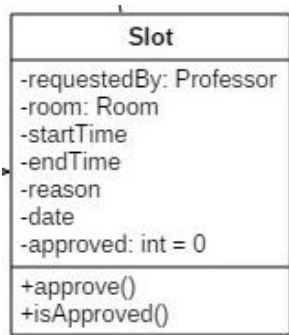
```
int main() {  
    Application app = Application::getInstance();  
    app.start();  
    return 0;  
}
```



Association Class

```
class Slot : public Model<Slot>
```

- Contains information for both Professor and Room
- Additional association data is also stored
 - Start/End Time
 - Reason
 - Date
 - Approval status, etc.



Example:

Used in **class** `SlotNotificationListView` for `ProfessorPanelView`

Used in **class** `UnseenSlotRequestListView` for `AdminPanelView`

Utility Classes

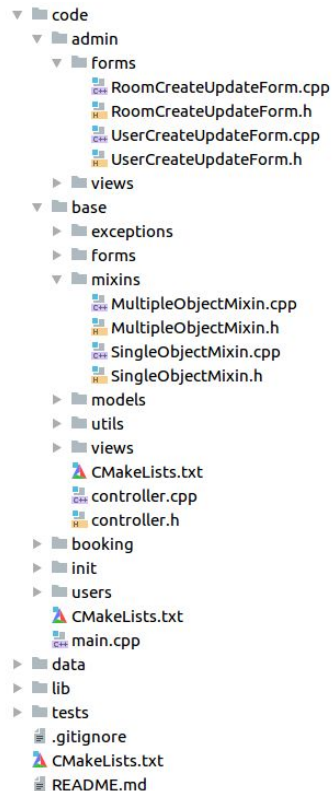
- **class Input**
 - Provides input methods throughout the application
 - **Regular expression** based input validation
- **class DateTime**
 - Able to detect leap years
 - Can compare two date instances for equality and inequality
- **class ViewPattern**
 - Used by Controller class to maintain Views
- **class Context**
 - Passed along while calling a view to pass stateful information
- **class Response**
 - Used to return contextual information back from a View

```
string Input::getEmail() {  
    regex  
    emailRegex(R"((\w+)(\.|_)?(\w*)@(\w+)(\.(?!\w+))+))");  
    return regexInputValidate(emailRegex, "Enter a  
                                valid email!");  
}
```



`base` directory compiled into static library

- Directory under `code/base` has been compiled as a library and used in main source code and tests
- Provides portability and out-of-the-box usage
- BaseLib.a file can be linked to an executable





Testing

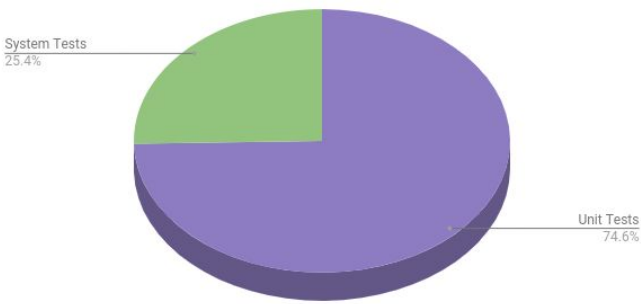
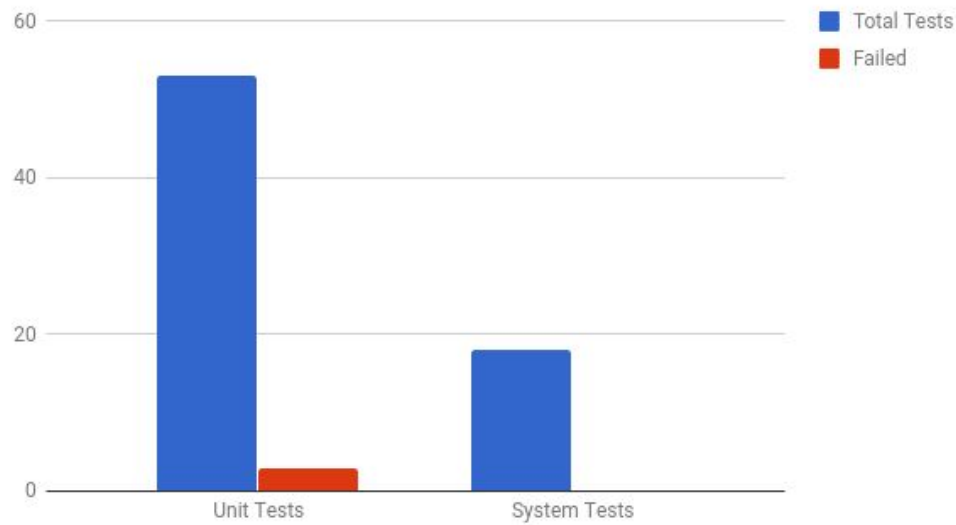
Statistics of Tests performed

Nature	Unit Tests	System Tests
Failed	3	0
Total	53	18
% Failure	5.66	0

All the above errors were resolved during the allotted enhancement period. However, the issue of Exiting system abruptly is not resolved after enhancement.



Total Tests and Failed



Automated Testing

- Did automated testing using Google's Google-Test Suite unit testing framework
- Followed Test-Driven-Development (TDD)

Example:

```
class Mock : public Model<Mock> { };
```

```
TEST_F(ModelTestFixture,  
        testSizeZeroAfterSaveAndDelete) {  
    Mock object = Mock();  
    object.save();  
    ASSERT_EQ(Mock::all().size(), 1);  
    object.remove();  
    ASSERT_EQ(Mock::all().size(), 0);  
}
```

Errors not resolved

- If the Application is abruptly closed from the terminal itself, the model instances edited after the application started, would not be saved to file and hence the track would be lost. This can be resolved if file handling is invoked every time a new object is created/updated/deleted but it decreases the efficiency of the application since files are read/written entirely.

Alternative: using a SQL database for dynamic updates



Cross Testing

Group 11: Sports Equipment Management

List of Use-cases tested

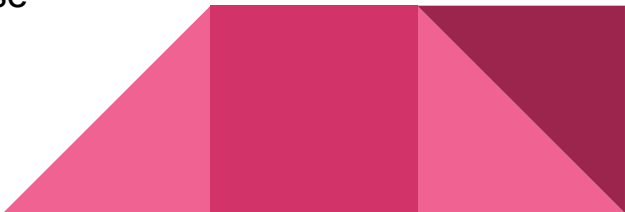
1. Login
2. Issue Equipment
3. Check Availability
4. Verification
5. Check Dues
6. Return Equipment
7. Update Database
8. Pay dues
9. Add Equipment
10. View List of Equipments
 - a. By equipment Category
 - b. By outstanding filter



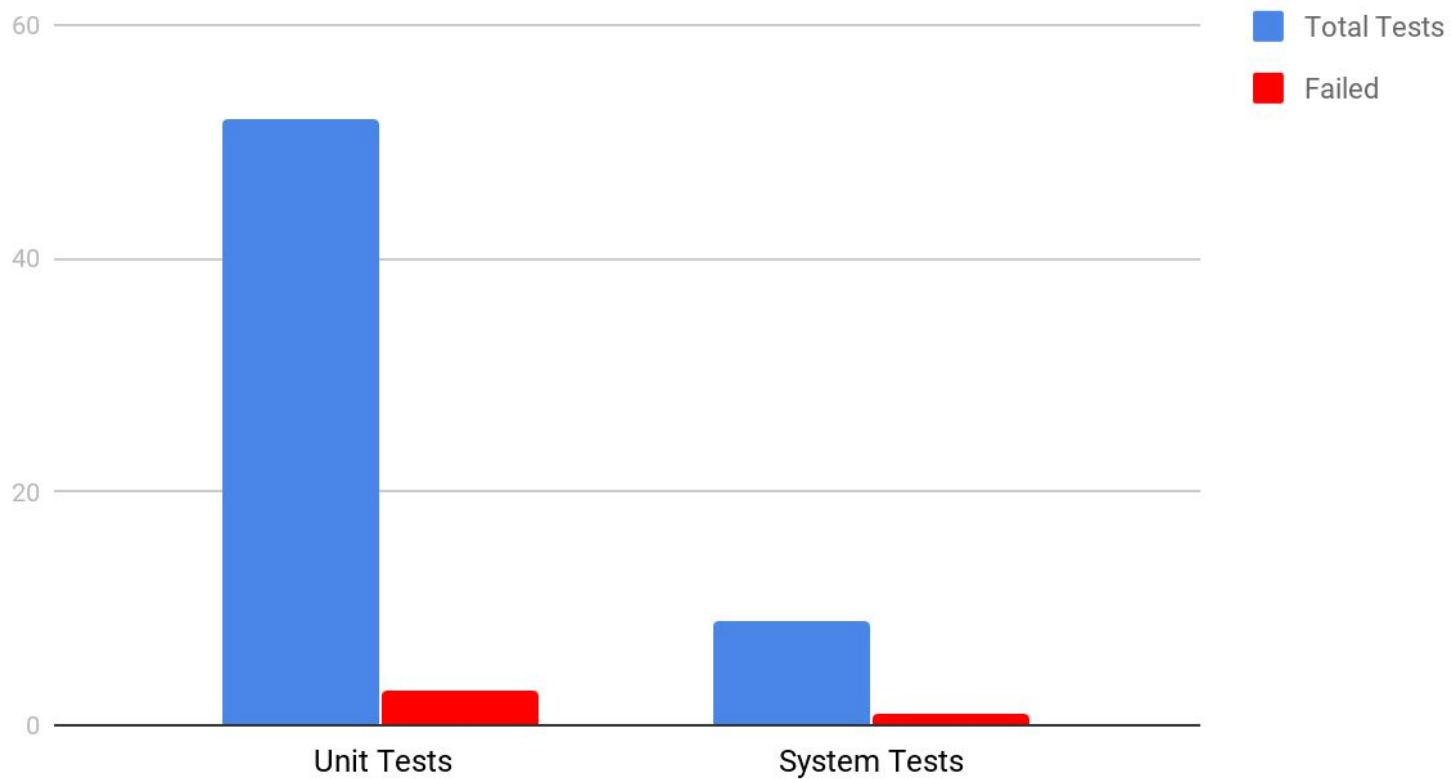
Statistics of Tests performed

Nature	Unit Tests	System Tests
Failed	3	1
Total	52	9
% Failure	5.7	11.1

No major error (eg wrong application of business rules etc.) was observed, however minor input/output error handling was not done during testing phase



Test Statistics



Conclusion

- A stand alone, stable C++ application for Classroom Booking System is developed.
- The application, was prepared keeping all steps of Software Engineering in mind ie. Software Requirements Analysis, Implementation, Testing, Refactoring and Maintainability.
- Thus the application is now ready to be released.

Code Demo



Thank You!