# Classroom Booking System

## TEST REPORT

Version <*1.0*>

*2/4/2018*

# VERSION HISTORY

| Version # | Tested By | Revision Date | Brief Description |
|---|---|---|---|
| 1.0 | Ajat Prabha(B16CS002) and Saksham Banga (B16CS042) | 2/4/2018 | Self Testing report |
| 1.1 | Ajat Prabha(B16CS002) and Saksham Banga (B16CS042) | 9/4/2018 | Function Testing+Self Testing Report |
| 1.2 | Himanshu Dhankar(B16CS008) Ashutosh Yadav(B16CS005) | 10/4/2018 | Cross Testing Report |

## Table of Contents

# 1.0 INTRODUCTION

## 1.1 PURPOSE

This Classroom Booking System *Unit and System* Test Report provides a summary of the results of test performed as outlined within this document. This is done in order to know the fallacies in the code which need to be resolved so that the final release of the software is safe, secure and ready to be directly shipped. The following test report presents the unit and system testing of all the modules defined in the system requirement specification.

Functionality Tests are performed in the following manner:

1. **In-scope testing**
   1. **Login(#Admin and #Faculty View)**
   2. **Create User(#Admin View)**
   3. **Update User(#Admin View)**
   4. **Delete User(#Admin View)**
   5. **List Unseen Slots(and approve them)(#Admin View)**
   6. **Create Room(#Admin View)-similar to <2>**
   7. **Update Room(#Admin View)-similar to <3>**
   8. **Retrieve Room details(#Admin View)**
   9. **Book Slot(#Faculty View)**
   10. **List Empty Rooms for a slot(#Faculty View)**
   11. **Update Slot(#Faculty View)-similar to <3,7>**
   12. **Delete Slot(#Faculty View)-similar to <4>**

2. **Out-Of-scope tests :** Tests related to performance, efficiency and space-time complexities

3. **Items not tested**
   Parallel instantiation of the program is not tested because the data is stored in files and loaded and updated at the start and end of the program respectively.
   Regular Expressions for email have not been added yet
   **Refer to constraints and testing assumptions to be kept in mind while testing (for the testing team) → Appendix B**

# 2.0 TEST PLAN

1. Unit tests are performed for each function.
2. After the unit tests are performed, stable testing for every module(functionality) is carried out.
3. For each unit test control flow testing is followed using predicate coverage and complete coverage.
4. A top down approach will be followed to conduct system testing.

**The Database(File Status, before starting testing is as follows)**

**BaseUser Objects (Full name, email, password, admin status)**

1. **Admin User**
   **admin@iitj.ac.in**
   **password**
   **true**
2. **Professor 1**
   **prof1@iitj.ac.in**
   **password**
   **false**

**Room Objects(Room Number, strength, audio status, video status)**

1. **105**
   **120**
   **true**
   **true**

**Slot Objects(Professor Object, Room object, Time Start, Time End, Reason)**

1. **Professor 1**
   **Room 105**
   **29/2/1904 12:10 - 12:40**
   **Doubt Solving Lecture**
2. **Professor 1**
   **Room 106**
   **29/3/1905 12:10 - 12:50**
   **Tutorial Lecture**

# 3.0 DETAILS OF UNIT TESTING

## 3.1 Login(#Admin and #Faculty View)

### 3.1.3 TEST ITEMS

The unit to be tested here facilitates safe login to the system as an appropriate user.

### 3.1.4 FEATURES TO BE TESTED

3.1.4.1 Valid Credentials are allowed to proceed
3.1.4.2 Invalid Credentials are redirected to the main page

### 3.1.5 ITEM PASS/FAIL CRITERIA

3.1.5.1 The system should not allow user with invalid credentials(username or password)
3.1.5.2 The system should allow user with valid and existing credentials(already made and stored on the file).

### 3.1.4 Test Cases

| Test No. | Username(email) | Password | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|---|
| 3.1.2.1 | ab@iitj.ac.in | abcdef | invalid credentials | invalid credentials | PASS | wrong password+ wrong username |
| 3.1.2.2 | prof1@iitj.ac.in | abcdef | invalid credentials | invalid credentials | PASS | correct username +incorrect password |
| 3.1.2.3 | ab@iitj.ac.in | password | invalid credentials | invalid credentials | PASS | incorrect username +correct password |

| | | | | | | |
|---|---|---|---|---|---|---|
| 3.1.2.4 | admin@iitj.ac.in | password | admin display panel opens | admin display panel opens | PASS | correct username +correct password |

## 3.2 Create user(#Admin View)

### 3.2.3 Test Items

The unit to be tested here facilitates registration of a new user to the system

### 3.2.4 Features To Be Tested

3.2.4.1 Unique username account created(not created previously)
3.2.4.2 Duplicate username flagged

### 3.2.5 Item Pass/Fail Criteria

3.2.5.1 The system should not allow an admin to create a user with duplicate username in the system
3.2.5.2 The system asks for new user details( admin status, email, first name, last name, password).

### 3.2.4 Test Cases

| Test No. | Username(email) | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 3.2.2.1 | prof1@iitj.ac.in | username already exists | username already exists | PASS | non unique username |
| 3.2.2.2 | prof2@iitj.ac.in | account created successfully | account created successfully | PASS | unique username hence account created |

## 3.3 Update User(#Admin view)

### 3.3.3 Test Items

The unit to be tested here verifies the update user view.

### 3.3.4 Features To Be Tested

3.3.4.1 The entered username is searched for, if the corresponding user exists, update the information

3.3.4.2 If the username entered is invalid, an error is flagged

### 3.3.5 Item Pass/Fail Criteria

3.3.5.1 For correct username, update the user successfully

3.3.5.2 For incorrect username, display an error and redirect to the main page

### 3.3.4 Test Cases

| Test No. | Username(email) | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 3.3.2.1 | ab@iitj.ac.in | user with these credentials doesn't exist | user with these credentials doesn't exist | PASS | username does not exist |
| 3.3.2.2 | prof1@iitj.ac.in | update user details | update user details | PASS | correct |

## 3.4 Delete User(#Admin View)

### 3.4.3 Test Items

The unit to be tested here facilitates safe deletion of users(both admin/faculty)

### 3.4.4 Features To Be Tested

3.4.4.1 If the username of the user to be deleted is incorrect, error is flagged.

3.4.4.2 If the username of the user to be deleted is same as the current logged in user, error is flagged.

3.4.4.3 If the username of the user is not in the above 2 categories, the user is removed from the system.

### 3.4.5 Item Pass/Fail Criteria

3.4.5.1 The system should not allow user with incorrect credentials username.

3.4.5.2 The system should not allow current logged in user to be deleted.

3.4.5.3 The system deletes a user if the username is not from the above 2 categories.

### 3.4.4 Test Cases

| Test No. | Username(email) | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 3.4.2.1 | ab@iitj.ac.in | invalid credentials | invalid credentials | PASS | invalid username |
| 3.4.2.2 | admin@iitj.ac.in | You cannot delete yourself | You cannot delete yourself | PASS | current logged in |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | user tried to be deleted |
| 3.4.2.3 | prof1@iitj.ac.in | user deleted | user deleted | PASS | user deleted successfully |

## 3.5 List Unseen Slots(#Admin view)

### 3.5.3 Test Items

The unit to be tested here lists unseen slot requests given by faculty.

### 3.5.4 Features To Be Tested

3.5.4.1 If unseen slot requests exist in the system, they are displayed in a list format with necessary details

3.5.4.2 If no unseen slot request exists, a message is displayed instead of the list.

### 3.5.5 Item Pass/Fail Criteria

3.5.5.1 For existing slot unseen requests, system returns a list of those slot requests

3.5.5.2 For empty list display a message

### 3.5.4 Test Cases

| Test No. | Slot list | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 3.5.2.1 | 3 unseen slot objects exist | #1 Prof1 slot details<br>#2 Prof2 slot details<br>#3 Prof3 slot details | #1 Prof1 slot details<br>#2 Prof2 slot details<br>#3 Prof3 slot details | PASS | List displayed |
| 3.5.2.2 | 0 unseen slot objects exist | Your list is empty | Your list is empty | PASS | Empty List |

## 3.6 Create Room(#Admin View)

### 3.6.3 Test Items

The unit to be tested here facilitates registration of a new room to the system

### 3.6.4 Features To Be Tested

3.6.4.1 Unique room created(not created previously)
3.6.4.2 Duplicate room number flagged

### 3.6.5 Item Pass/Fail Criteria

3.6.5.1 The system should not allow an admin to create a room with duplicate room number in the system
3.6.5.2 The system asks for new room details( room number, strength, audio(yes/no), video(yes/no)).

### 3.6.6 Test Cases

The test cases of this view are similar to those of view 3.2 hence they test cases are omitted.

## 3.7 Update room(#Admin View)

### 3.7.3 Test Items

The unit to be tested here facilitates updation of an existing room

### 3.7.4 Features To Be Tested

3.7.4.1 The entered room number is searched for, if the corresponding room exists, update the information

3.7.4.2 If the room number entered is invalid, an error is flagged

### 3.7.5 Item Pass/Fail Criteria

3.7.5.1 For correct room number, update the room successfully

3.7.5.2 For incorrect room number, display an error and redirect to the main page

### 3.7.6 Test Cases

The test cases of this view are similar to those of view 3.3 hence they test cases are omitted.

## 3.8 Retrieve Room Details(#Admin view)

### 3.8.3 Test Items

The unit to be tested here displays room details for a given room number.

### 3.8.4 Features To Be Tested

3.8.4.1 If the room number entered does not exist, display an error message.

3.8.4.2 If the room number exists, display necessary details and redirect to main page.

### 3.8.5 Item Pass/Fail Criteria

3.8.5.1 The system displays an error message for non existent rooms

3.8.5.2 The system displays accurate room details for valid room numbers.

### 3.8.4 Test Cases

| Test No. | Room Number | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 3.8.2.1 | 106(Non existent) | Room does not exist | Room does not exist | PASS | No room exists with entered room number |
| 3.8.2.2 | 105(Existent) | Room #105<br>Strength:120<br>Audio: y<br>Video:y | Room #105<br>Strength:120<br>Audio: y<br>Video:y | PASS | Room Details displayed |

## 3.9 Book Slot(#Faculty view)

### 3.9.3 Test Items

The unit to be tested here is used to create a slot request by a faculty which will be approved/unapproved by an admin.

### 3.9.4 Features To Be Tested

3.9.4.1 If empty rooms exist in the system, they are displayed in a list formatted with necessary details.

3.9.4.2 If no empty rooms exists, a message is displayed instead of the list.

### 3.9.5 Item Pass/Fail Criteria

3.9.5.1 If a professor or room doesn't exist, or slot request clashes with an existing slot, display an error message.

### 3.9.4 Test Cases

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 3.9.2.1 | Room corresponding to request doesn't exist. | Show error message | Show error message | PASS | Error message shown |
| 3.9.2.2 | A clashing slot request exists | Show error message | Show error message | PASS | Error message shown |

## 3.10 List Empty Rooms for a slot(#Faculty View)

### 3.10.3 Test Items

The unit to be tested here is displaying a list of rooms that can be requested between two given timestamps.

### 3.10.4 Features To Be Tested

3.10.4.1 A room and professor should exist in order to create a slot request.

3.10.4.2 Unique slot request is created(duplicate/clashing slot requests are flagged).

### 3.10.5 Item Pass/Fail Criteria

3.10.5.1 For available empty rooms, system returns a list of those rooms.

3.10.5.2 For empty list display a message.

### 3.10.6 Test Cases

| Test No. | Room list | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 3.10.2.1 | 2 empty room objects exist | #1 Room 2 details<br>#2 Room 2 details | #1 Room 2 details<br>#2 Room 2 details | PASS | List displayed |
| 3.10.2.2 | 0 empty rooms exist | No empty rooms exist | No empty rooms exist | PASS | Empty List |

## 3.11 Update Slot(#Faculty view)

### 3.11.3 Test Items

The unit to be tested here updates the slot request details

### 3.11.4 Features To Be Tested

3.11.4.1 The entered slot id is searched for, if the corresponding user exists, update the information

3.11.4.2 If the slot id entered is invalid, an error is flagged

### 3.11.5 Item Pass/Fail Criteria

3.11.5.1 For correct slot id, update the user successfully

3.11.5.2 For incorrect slot id, display an error and redirect to the faculty panel view

### 3.11.6 Test Cases

The view is similar to update user shown in 3.3 and 3.7 and hence the test cases are updated here(The backend is templatized, hence tests for either of them should suffice).

## 3.12 Delete Slot(#Faculty View)

### 3.12.3 Test Items

The unit to be tested here facilitates safe deletion of slot requests

### 3.12.4 Features To Be Tested

3.12.4.1 If the slot id of the user to be deleted is incorrect, error is flagged.

3.12.4.2 If the username of the user to be deleted is same as the current logged in user, error is flagged.

3.12.4.3 If the username of the user is not in the above 2 categories, the user is removed from the system.

### 3.12.5 Item Pass/Fail Criteria

3.12.5.1 The system should not allow user with incorrect credentials username.

3.12.5.2 The system should not allow current logged in user to be deleted.

3.12.5.3 The system deletes a user if the username is not from the above 2 categories.

### 3.12.6 Test Cases

This view is similar to the delete users view shown above in 3.4 hence the test cases for this view are omitted. (The templatized view calls DeleteView for both the views, hence test cases for one of them suffice).

## 4.0 DETAILS OF SYSTEM TESTING

| Step | TEST STEP | Result Intended | Actual Output | Pass/ Fail |
|---|---|---|---|---|
| 1 | Login → Admin Panel View | Admin Panel Opens and presents various options | Admin Panel Opens and presents various options | PASS |
| 2 | Login → Faculty Panel View | Faculty Panel Opens and presents various options | Faculty Panel Opens and presents various options | PASS |
| 3 | Admin Panel View → List Unseen Slot Requests | Displays a list of slot requests which are neither unapproved nor approved | Displays a list of slot requests which are neither unapproved nor approved | PASS |
| 4 | Admin Panel View → Create New User | Creates a new user by validating and receiving valid input | Creates a new user by validating and receiving valid input | PASS |
| 5 | Admin Panel View → Create New Room | Creates a new room by validating and taking valid input | Creates a new room by validating and taking valid input | PASS |
| 6 | Admin Panel View → Update Existing User | Updates a user whose username is entered, and its updated information is validated | Updates a user whose username is entered, and its updated information is validated | PASS |
| 7 | Admin Panel View → Update Existing Room | Updates a room whose room number is entered, and its updated | Updates a room whose room number is entered, and its updated | PASS |

| | | | | |
|---|---|---|---|---|
| | | information is validated | information is validated | |
| 8 | Admin Panel View → Delete User | Deletes existing user objects | Deletes existing user objects | PASS |
| 9 | Admin Panel View → Delete Room | Deletes existing room objects | Deletes existing room objects | PASS |
| 10 | Admin Panel View → View Room Details | Displays Room details for requested room number | Displays Room details for requested room number | PASS |
| 11 | Faculty Panel View → Update Profile | Updates the profile of current logged in user and its updated information is validated | Updates the profile of current logged in user and its updated information is validated | PASS |
| 12 | Faculty Panel View → List Empty Rooms | Displays list of empty rooms between a given time slot and other filters like room strength, audio video capabilities | Displays list of empty rooms between a given time slot and other filters like room strength, audio video capabilities | PASS |
| 13 | Faculty Panel View → Book Slot | Books a new slot request by taking in a room number and time details for a professor | Books a new slot request by taking in a room number and time details for a professor | PASS |
| 14 | Faculty Panel View → Check status | Checks status of self made slot requests, to know their current status(approved/unapproved/unseen) | Checks status of self made slot requests, to know their current status(approved/unapproved/unseen) | PASS |
| 15 | Faculty Panel View → Update Slot | Updates the slot of the current user by taking and its id and the updated information is validated | Updates the slot of the current user by taking and its id and the updated information is validated | PASS |
| 16 | Faculty Panel View → Delete Slot | Deletes existing slot as per entered id | Deletes existing slot as per entered id | PASS |
| 17 | Panel View → Logout | Logs out the currently logged in user and redirects to Splash View | Logs out the currently logged in user and redirects to Splash View | PASS |

| | | | | |
|---|---|---|---|---|
| 18 | Panel View → Exit | Exits from the program | Exits from the program | PASS |

## 5. Critical Function Test Report

**(For all major functions in the code)**

### *Plan:*

The whole code follows inheritance on 3 levels, thus every function which is virtual may be called virtually across 3 levels. The Functionality testing signifies testing of the 3rd layer of the invocation of such functions. Thus for complete test coverage, it is sufficient to test the functions in the first 2 layers of the inheritance.

**We have a BASE directory which contains several C++ files:**

1. **Models.h**
   - 1.1. **Model Class**
     - 1.1.1. **virtual T &save()**
     - 1.1.2. **virtual T &remove()**
     - 1.1.3. **static T *findById(int)**
     - 1.1.4. **static writeToFile(const string)**
     - 1.1.5. **static readFromFile(const string)**

2. **Views.h**
   - 2.1. **View Class**
     - 2.1.1. **display()**
     - 2.1.2. **call(Context)**
     - 2.1.3. **populateMenu()**
     - 2.1.4. **callAction(int)**
   - 2.2. **CreateView:View(All virtual functions overridden)**
   - 2.3. **UpdateView:View(All virtual functions overridden)**
   - 2.4. **ListView:View, MultipleObjectMixin(pure virtual class)**
   - 2.5. **DeleteView:View, SingleObjectMixin**
     - 2.5.1. **display()**
   - 2.6. **DetailView:View, SingleObjectMixin(pure virtual class)**

3. **Mixins.h**
   - 3.1. **Single Object Mixin**
     - 3.1.1. **static T* getObject(int id)**
   - 3.2. **Multiple Object Mixin**

- 3.2.1. static vector<T> getQuerySet()(pure Virtual function)

4. Forms.h
   - 4.1. Form
     - 4.1.1. isValid()
     - 4.1.2. addError(string)
     - 4.1.3. clean()
     - 4.1.4. printErrors()
   - 4.2. ModelForm:Form
     - 4.2.1. save()

**The ADMIN, BOOKING, USER directories, use the BASE directory, in some or the other way via inheritance**

1. ADMIN contents:
   - 1.1. Forms
     - 1.1.1. UserCreateUpdateForm:ModelForm<BaseUser>
     - 1.1.2. RoomCreateUpdateForm:ModelForm<Room>
   - 1.2. Views
     - 1.2.1. AdminPanelView: public View(All functionalities corresponding to Admin invoked here, which is covered in functionality testing)
2. BOOKING contents:
   3. Forms
      - 3.1. SlotCreateUpdateForm:ModelForm<Slot>
        - 3.1.1. clean()
        - 3.1.2. save()
   4. Views
      - 4.1. FacultyPanelView: public View(All functionalities corresponding to Faculty invoked here, which is covered in functionality testing)
   5. Models
      - 5.1. Room:Model<Room>: Tests for Model are given above, hence Room class can use Model as a stub
      - 5.2. Slot:Model<Slot>: Tests for Model are given above, hence Slot class can use Model as a stub.

1. USER contents:
   - 1.1. Models
     - 1.1.1. BaseUser
       - 1.1.1.1. static BaseUser *findByEmail(string &)
     - 1.1.2. Admin: BaseUser
     - 1.1.3. Professor: BaseUser
   - 1.2. Views
     - 1.2.1. UserCreateView:CreateView<BaseUser>
       - 1.2.1.1. display()
     - 1.2.2. UserUpdateView:UpdateView<BaseUser>
       - 1.2.2.1. display()
     - 1.2.3. LoginView:View

**1.2.3.1. display()**

**1.2.4. LogoutView:View**

**1.2.4.1. display()**

## 1. BASE → Models

### a. Model(T = Room) Room &save

#### I. FUNCTION

virtual Room &save()

- This function saves a room object to the map<int, Room> with the id auto incrementing
- The function is invoked by a Room Object when constructed.

  ie Room r105;
- r105.save() → saves r105 in the map by auto-assigning a primary key(id).

#### II. TEST ITEMS

The unit will be used to create a new room entity in the map.

Thus the same shall be tested.

#### III. FEATURES TO BE TESTED

All the current attributes of the invoking room object are stored in the map

#### IV. ITEM PASS/FAIL CRITERIA

This function should always save the object in the map. No fail criteria as such(unless dynamic cast doesn't work)

#### V. TEST CASES

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| 1. | Room Object(Number → 105, strength → 120, audio → y, video→ y).save() called | Room Object saved | Room Object saved | PASS | Room object correctly saved in map |

### b. Model(T = Room) Room &remove

#### I. FUNCTION

virtual Room &remove()

- This function removes a room object from the map<int, Room>
- The function is invoked by the Room Object when needed to be deleted

  ie Room r105;
- r105.remove() → removes r105 from the map

#### II. TEST ITEMS

The unit will be used to remove a room entity from the map.
Thus the same shall be tested.

#### IV. FEATURES TO BE TESTED

The room is successfully deleted or not.

#### V. ITEM PASS/FAIL CRITERIA

If the system manages to delete the room, test passes else fails

#### VI. TEST CASES

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | Room Object(Number → 105, strength → 120, audio → y, video→ y).remove() called | Room Object deleted | Room Object deleted | PASS | Room object correctly deleted from map |

### c. Model(T = Room) static Room &findById(int)

#### i. Function

virtual Room &findById(int)

- This function takes in an id and if an object with such an id exists, it returns such an object.
- This function can be invoked by any function operating on Room objects since the function is static

#### iii. Test Items

- The unit will be used to obtain a Room object, with given id

#### v. Features To Be Tested

- The room is retrieved successfully.
- If the room doesn't exist, return a nullptr

#### vi. Item Pass/Fail Criteria

- If the system manages to return the correct room object, test passes
- If the room requested for doesn't exist, function must return a nullptr.

#### vii. Test cases

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | Room* room = Room::findById(1) called | Room Object with id 1(105) returned | Room Object with id 1(105) returned | PASS | Room object correctly retrieved from map |
| 2. | Room* room = | returns a | returns a | PASS | Room Object |

| | | | | | |
|---|---|---|---|---|---|
| | Room::findById(7) called(There is no object with id 7) | nullptr | nullptr | | doesn't exist, hence nullptr returned. |

### d. Model(T = Room) static Model<Room>::writeToFile(const string&)

#### I. FUNCTION

void writeToFile(const string&)

- This function writes contents of the map to the file, whose filename is passed in as parameters. The sizeof object which is being written to the file is taken from the type of function(The T value) calling the function.

#### IV. TEST ITEMS

The unit will be used to write the object data to file.

#### VI. FEATURES TO BE TESTED

All the current attributes of all the objects on the Room map are copied to file

#### VII. ITEM PASS/FAIL CRITERIA

This function write the map contents onto the file. If it does so, test is successful

#### viii. TEST CASES

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | Model<Room>::writeToFile("RoomData.dat") is called | All Room objects copied to file "RoomData.dat" | All Room objects copied to file "RoomData.dat" | PASS | Room objects correctly written to file |

### e. Model(T = Room) static Model<Room>::readFromFile(const string&)

#### I. FUNCTION

void readFromFile(const string&)

- This function reads contents of the file and loads them on the map, whose filename is passed in as parameters. The sizeof object which is being written to the file is taken from the type of function(The T value) calling the function.

#### V. TEST ITEMS

The unit will be used to load object data from files onto maps

#### VII. FEATURES TO BE TESTED

All the current attributes of all the objects on the Room map are loaded from file

#### VIII. ITEM PASS/FAIL CRITERIA

This function loads the map contents from the file. If it does so, test is successful

#### IX. TEST CASES

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | Model<Room>::readFromFile("Room Data.dat") is called | All Room objects loaded onto the objectList map | All Room objects loaded onto the objectList map | PASS | Room objects correctly loaded from file |

## 2. BASE → View(Abstract Class)

**a. display() :** Since this is a virtual function, it has no definition in View Class, the tests corresponding to display function. where view has been inherited will be listed later.

**b. call(Context)**

#### I. Function

- This function takes in a Context Object(containing the current logged in user information, and the object id to be operated upon)
- It invokes the display function of the corresponding View type being displayed to the user.
- The display function, by the virtue of virtual functions, invokes the correct display() corresponding to the view(eg CreateView) invoked
- A new Response object(Containing redirect view information) is also created here which may be updated by the display().

#### II. Test Items

- If the call() for a view is called, call() calls appropriate display() and updates response

#### III. Features To Be Tested

- calling call() from a specific View should call its appropriate display() and must update response

#### IV. Item Pass/Fail Criteria

- If the call() updates and returns the correct response*, the test is successful

#### V. Test Cases

- **For testing this function, assume the usage of an example CreateView class.**
- The CreateView Class is used to create new objects which are to be saved to the database. Thus if CreateView::call() is called and the context passed contains the admin BaseUser object, -1 as object id, the display of CreateView is invoked, which is further overridden by the View which calls it, eg SlotCreateView, RoomCreateView, UserCreateView.

- The testing is also carried out by creating a new ViewTestClass which simply inherits View class, and the display() just outputs a success message. The tests hence **PASS** and are successful

### c. populateMenu()

#### I. Function

- This function populates menu for the View to display a list of menu options that can be performed under the header of this View

#### II. Test Items

- The function, when called should display title of the View, and should display list of available options which can be performed by this view

#### III. Features To Be Tested

- The title and The list should be correctly displayed, After which user should be prompted to enter desired option and then callAction() should be invoked

#### IV. Item Pass/Fail Criteria

- If the execution of all the Features listed above occurs successfully, the function has passed the test

#### V. Test Cases(Consider AdminPanelView)

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | AdminPanelView Object calls populateMenu() | A list of all options which the Admin User can perform is displayed, user is prompted to enter an option and then the correspondin g id is passed when callAction(id) is called | A list of all options which the Admin User can perform is displayed, user is prompted to enter an option and then the correspondin g id is passed when callAction(id) is called | PASS | List of Admin Portal functions displayed |

### d. callAction(int)

#### I. Function

- This function is invoked from the populateMenu() of a View object.
- The function calls the function pointer corresponding to the number passed as parameter

#### II. Test Items

- If the number passed is mapped to a function, the function is called
- If the number passed is not mapped to a function, an error message is displayed

#### III. Features To Be Tested

- A valid function number is passed and the corresponding function should be invoked.
- An invalid function number should be detected and an error message saying invalid choice should be displayed, prompting for input again.

#### IV. Item Pass/Fail Criteria

- The features listed above must be performed correctly to pass the tests.

#### V. Test Cases

Just like in PopulateMenu() consider the AdminPanelView

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | callAction(2) | The function CreateNewUser in AdminPanel View is called. | The function CreateNewUser in AdminPanel View is called. is called | PASS | callAction() calls the 2nd option in the list |

| | | | | | |
|---|---|---|---|---|---|
| 2. | callAction(200) | Invalid choice, please try again | Invalid choice, please try again | PASS | callAction() displays error message since no function with given id exists |

### 3. BASE → DeleteView

#### a. display()

##### I. Function

- This function invokes the remove() function for a object whose id is given.

##### II. Test Items

- If the display() for a view is invoked, the id corresponding to the object id in context is passed to the SingleObjectMixin's getObject function
- The function returns a pointer to the object which is to be removed.
- Finally the display() calls remove() for the corresponding object so returned(If the object to be deleted exists)

##### III. Features To Be Tested

- calling DeleteView::display() should appropriately delete the object whose id is passed in the context.

##### IV. Item Pass/Fail Criteria

- If the display() removes the object(if it exists), the test is successful
- If the object doesn't exist, an error message must be displayed.

##### V. Test Cases

- **ASSUME SlotDeleteView invoked From FacultyPanelView**

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | display() **[Context→ {Faculty User, id = 2}]** | Successfully Deleted (Slot with ID: 2 deleted) | Successfully Deleted | PASS | display() for valid object id |
| 2. | display() **[Context→ {Faculty User, id = 200}]** | Not Found | Not Found | PASS | display() for invalid object id |

## 4. BASE → Forms(Abstract Class)

### a. clean()

#### I. Function

- This method is a pure virtual method and needs to be implemented in the derived class.
- Any kind of validation required is done in this method, errors are added to `errors` private variable.

#### II. Test Items

- Any error added to `errors` is reflected when calling errors.size()

#### III. Features To Be Tested

- Custom validation can be added on individual fields.

#### IV. Item Pass/Fail Criteria

- When calling this method, validation is performed.

#### V. Test Cases

- For testing this, define any custom validation, and when you call isValid() on form object, it should run this method and return a boolean.

### b. isValid()

#### I. Function

- This method first calls the `clean()` method and then returns `errors.empty()` which will be cast to bool depending upon validation.

#### II. Test Items

- When calling this, validation is done and errors are added by clean() method.

#### III. Features To Be Tested

- Calling this method should run the validation and return true or false.

IV. ITEM PASS/FAIL CRITERIA
- After calling, it should return true if no errors are there during validation.

V. TEST CASES
- For testing this method, define some custom validation and if any of the validation is violated it'll return false else true.

### c. addError(string)

I. FUNCTION
- This method is used to add error to the `errors` variable.

II. TEST ITEMS
- It should add a string typed error to form.

III. FEATURES TO BE TESTED
- After calling this, isValid() should return false because

IV. ITEM PASS/FAIL CRITERIA
- After calling this method, size of `errors` should increase.

V. TEST CASES
- For testing this method, call this during the clean() method and then upon calling isValid(), false should be returned.

### d. printErrors()

I. FUNCTION
- This method is used to print all the errors stored in `errors` variable.

II. TEST ITEMS
- It should print all the errors when called

III. FEATURES TO BE TESTED
- It prints nothing when no errors, else it prints all the errors.

IV. ITEM PASS/FAIL CRITERIA
- Errors are printed or not based on isValid() return value.

V. TEST CASES
- Add some errors via addError() method, and then on calling printErrors, they should be printed.

### e. ModelForm<T>::save()

This is a pure virtual method and must be implemented by derived class.

## 5. BASE → Single Object Mixin(Abstract Class)

### a. T* getObject(int)

#### I. Function

- This function returns a pointer to the object whose id is passed(if it exists), else returns a nullptr

#### II. Test Items

- If the display() for a view is invoked, the id corresponding to the object id in context is passed to the SingleObjectMixin's getObject() function
- The function returns a pointer to the object which is to be operated upon

#### III. Features To Be Tested

- calling getObject() should return a pointer to the object whose id is passed in as parameter
- If the object doesn't exist, returns a nullptr

#### IV. Item Pass/Fail Criteria

- If all the above features work, ie the getObject() function works correctly for the case when object asked for exists, and for the case when it does not exist, the Tests will said to be successful

#### V. Test Cases

- **ASSUME SingleObjectMixin for AdminDetailView**

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | getObject(2) | returns pointer to user whose id is 2 | returns pointer to user whose id is 2 | PASS | getObject() for valid id |
| 2. | getObject(200) | returns | returns | PASS | getObject() |

| | | | | | |
|---|---|---|---|---|---|
| | | nullptr | nullptr | | for invalid id |

### 6. ADMIN → UserCreateUpdateForm : ModelForm<BaseUser>

#### a. save()

##### I. FUNCTION

- This method is called after the isValid() method on base form is called.
- It saves a Model<T> instance in the system, in this case a Model<BaseUser> and returns a down type-casted object T of it.

##### II. TEST ITEMS

- It should save the instance to the iterable objectList.

##### III. FEATURES TO BE TESTED

- No errors occur while saving.
- Validation is done properly.

##### IV. ITEM PASS/FAIL CRITERIA

- After a user is saved, one should be able to interact with it in the system.

##### V. TEST CASES

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | form1->save(); [{"Ajat", "Prabha", "prabha.1@iitj.ac.in", "password", true}] | Object is saved and reference to it is returned. | Object is saved and reference to it is returned. | PASS | This was an object create scenario |
| 2. | form2->save(); [{"Saksham", "Banga", "banga.1@iitj.ac.in", "password", false, | Object is updated and reference to it is returned. | Object is updated and reference to it is returned. | PASS | This was an object update scenario |

| | | | | | |
|---|---|---|---|---|---|
| | BaseUser::findByEmail("banga.1@iitj.ac.in")}] | | | | |
| 3. | form3->save(); [{"Saksham", "Banga", "prabha.1@iitj.ac.in", "password", true}] | Error message printed. | Error message printed. | PASS | Since, "prabha.1@iitj.ac.in" already exists, validation adds error to the form. |

### 7. ADMIN → RoomCreateUpdateForm : ModelForm<Room>

#### a. save()

##### I. FUNCTION

- This method is called after the isValid() method on base form is called.
- It saves a Model<T> instance in the system, in this case a Model<BaseUser> and returns a down type-casted object T of it.

##### II. TEST ITEMS

- It should save the instance to the iterable objectList.

##### III. FEATURES TO BE TESTED

- No errors occur while saving.
- Validation is done properly.

##### IV. ITEM PASS/FAIL CRITERIA

- After a room is saved, one should be able to interact with it in the system.

##### V. TEST CASES

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | form1->save(); [{105, 180, 'y', 'y'}] | Object is saved and reference to it is returned. | Object is saved and reference to it is returned. | PASS | This was an object create scenario |
| 2. | form2->save(); [{105, 110, 'y', 'n', Room::findByRoomNumber(105))}] | Object is updated and reference to it is returned. | Object is updated and reference to it is returned. | PASS | This was an object update scenario |

| | | | | | |
|---|---|---|---|---|---|
| 3. | form3->save(); [{105, 100, 'n', 'n'}] | Error message printed. | Error message printed. | PASS | Since, a room numbered 105 already exists, validation adds error to the form. |

## 8. BOOKING → SlotCreateUpdateForm : ModelForm<Slot>

### a. clean()

#### I. FUNCTION

- This method is used to validate the slot details and clashes with other existing requested slots.
- It saves a Model<T> instance in the system, in this case a Model<BaseUser> and returns a down type-casted object T of it.

#### II. TEST ITEMS

- It should save the instance to the iterable objectList.

#### III. FEATURES TO BE TESTED

- No errors occur while saving.
- Slot is unique and doesn't clash with any other slot.
- Validation is done properly.

#### IV. ITEM PASS/FAIL CRITERIA

- After a slot is requested and saved, one should be able to interact with it in the system.
-

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | Slot(Room: 105, Strength: 120, Audio: y, Video: y Start: 20/11/2018 16 00 End: 20/11/2018 17 00) calls clean() | Slot requested successfully | .Slot #2 requested successfully | PASS | The inputs are correct hence Slot requested successfully |
| 2. | Slot(Room: 107, | Error: Room | Error: Room | PASS | The Room |

| | | | | | |
|---|---|---|---|---|---|
| | Strength: 120,<br>Audio: y,<br>Video: y<br>Start: 20/11/2018 16 00<br>End: 20/11/2018 17 00) calls clean() | Number does not exist | Number does not exist | | number for which slot is requested does not exist |
| 3. | Slot(Room: 105,<br>Strength: 125,<br>Audio: y,<br>Video: y<br>Start: 20/11/2018 16 00<br>End: 20/11/2018 17 00) calls clean() | Error: Strength requirements does not match | Error: Strength requirements does not match | PASS | Since room capacity is 120, 125 students cannot be accomodated |
| 4. | Slot(Room: 105,<br>Strength: 120,<br>Audio: n,<br>Video: y,<br>Start: 20/11/2018 16 00<br>End: 20/11/2018 17 00) calls clean() | Slot Requested successfully | Slot #2 Requested successfully | PASS | Audio requirement is not required→ slot can be accomodated. |
| 5. | Slot(Room: 105,<br>Strength: 120,<br>Audio: n,<br>Video: y,<br>Start: 20/11/2018 16 00<br>End: 20/11/2018 17 00) calls clean() | Slot Requested successfully | Slot #2 Requested successfully | PASS | Video requirement is not required→ slot can be accomodated. |
| 6. | Slot(Room: 105,<br>Strength: 120,<br>Audio: n,<br>Video: n,<br>Start: 20/11/2018 16 00<br>End: 20/11/2018 17 00) calls clean() | Slot Requested successfully | Slot #2 Requested successfully | PASS | Video and Audio requirement is not required→ slot can be accomodated. |
| 7. | **Assume that any one of the above slot is created before this**<br>Slot(Room: 105,<br>Strength: 120, | Error: Slot overlapping for requested class | Error: Slot overlapping for requested class | PASS | 2 Slots requesting overlapping slots cannot be requested |

| | | | | | |
|---|---|---|---|---|---|
| | Audio: y,<br>Video: n,<br>Start: 20/11/2018 16 00<br>End: 20/11/2018 17 00) calls clean() | | | | |
| 8. | **Assume that any one of the above slot is created before this**<br>Slot(Room: 105, Strength: 120,<br>Audio: y,<br>Video: n,<br>Start: 20/11/2018 16 30<br>End: 20/11/2018 17 00) calls clean() | Error: Slot overlapping for requested class | Error: Slot overlapping for requested class | PASS | 2 Slots requesting overlapping slots cannot be requested |

**b. save()**

**I. FUNCTION**

- This method is called after the isValid() method on base form is called.
- It saves a Model<T> instance in the system, in this case a Model<BaseUser> and returns a down type-casted object T of it.

**II. TEST ITEMS**

- It should save the instance to the iterable objectList.

**III. FEATURES TO BE TESTED**

- No errors occur while saving.
- Validation is done properly.

**IV. ITEM PASS/FAIL CRITERIA**

- After a slot is requested and saved, one should be able to interact with it in the system.

**V. TEST CASES**

| Test No. | Test Case | Expected Output | Actual Output | Test Verdict | Comments |
|---|---|---|---|---|---|
| 1. | form1->save();<br>[{prof1, room105, startTime1, endTime1, reason, 0}] | Object is saved and reference to it is returned. | Object is saved and reference to it is returned. | PASS | This was an object create scenario |

| | | | | | |
|---|---|---|---|---|---|
| 2. | form2->save(); [{prof1, room105, startTime2, endTime2, reason, 0, Slot::findById(1)}] | Object is updated and reference to it is returned. | Object is updated and reference to it is returned. | PASS | This was an object update scenario |
| 3. | form3->save(); [{prof1, room105, startTime1, endTime1, reason, 0}] | Error message printed. | Error message printed. | PASS | Since, a slot request with similar parameters already exists, validation adds error to the form. |

**Testing Team Report:**

**All Tests Reported in the Self Testing Report are verified and validated. There was no test which was found to be invalid during this testing procedure.**

**Other Tests performed by the Testing Team along with the Tester Details are listed below:**

| Test No. | Tested By | Summary of Defect spotted | Resolved before Release of Software? | Why not resolved(If applicable)? |
|---|---|---|---|---|
| 1. | Himanshu | **Email Not validated when entered(123456 also works as valid email)** | | |
| 2. | Ashutosh | **Error info message not displayed when wrong input on main screen however program doesn't crashes.**<br>**1. login**<br>**2. exit** | | |
| 3. | Himanshu | **If a update is carried out and then terminal is shut down without logging out, then newly created/updated objects will not be stored.** | | |
| 4 | Himanshu | **If more than required inputs are entered, the program takes the** | | |

| | | | | |
|---|---|---|---|---|
| | | **extra inputs for the next input prompts. This must be resolved for all occurences in the program.** | | |

## Appendix A : Commonly Used Terms and their technical reference

| Term | Term Reference |
|---|---|
| user | BaseUser object(faculty/professor) |
| View | Inherited View from MVC framework. In general, this is the class responsible for presenting the data on the terminal/command prompt |
| Validate | A generic term which depends on context wherever used. eg Validation in Room creation means identifying duplication of 2 objects. |
| Controller | The interface between Views and Models(File/Database handler) |
| flagged | Error message displayed(Informative) and redirected to appropriate view(The view opened just before this) |
| Slots | Slot Class objects which signify slot |

| | |
|---|---|
| | requests by a Professor, for a given room from a given time(Start Time) to a given time(End Time) |
| username | unique identifier for any model instance for eg for user, username → email id of the user |
| Stub | The unit, once tested, are assumed to be working correctly and hence can be considered as a working abstraction to carry out System integration. |

## Appendix B

### Constraints and Assumptions to be kept in mind while Testing(For the Testing Team)

1. Since the Entire code is templatized, some views are not tested twice(to avoid redundancy) because they have been inherently tested for 1 class and thus by the virtue of templates, they are expected to work for all classes who inherit the same template.
2. Regular Expressions for email inputs have not been applied so any invalid email, as long as it is a string will be accepted.