Design and Implementation of a Classroom Allocation System Prototype

GRADUATE PROJECT REPORT

Submitted to the Faculty of
The Department of Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, Texas

In Partial Fulfilment of the Requirements for the Degree of
Master of Science in Computer Science

By

Suryakanthi Navuduri
Fall 2016

Committee Members

Dr. Mario Garcia
Committee Chairperson

Dr. David Thomas
Committee Member
ABSTRACT

In a university/school, it is essential for classes to be taught in suitable classrooms. That is, classrooms are expected to give the required computer systems, audio and video hardware, white board, projector equipment, air conditioner and interactive multimedia equipment alongside adequate space for all students who have enrolled for the class. Most universities/schools handle this allocation process with a manual procedure which involves the coordination which all the departmental representatives and staff from the Registration Office. Usually, the manual procedure like this gives way to various challenges and is inclined to mistakes. A better approach to reliably schedule classrooms is to utilize a computer assisted web-based system that will monitor all classrooms on campus taking into consideration the details about those rooms (their capacity) that can automatically recommend effective pairings (Course to classroom pairing) with the courses offered for a given semester. The efficiency will be judged in view of numerous variables, most important being that the extent of every room and its resources is utilized effectually. This project document the details and execution of a web-based system that helps the staff of any university to assign classrooms for a given semester. This system takes into consideration, the most essential elements such as the classroom size, course capacity, preferred building of a department.
# TABLE OF CONTENTS

Abstract .................................................................................................................................................. ii

Table of contents ................................................................................................................................... iii

1. BACKGROUND AND RATIONALE ................................................................................................. 1
   1.1 Introduction ................................................................................................................................. 1
   1.2 Scheduling Algorithm ............................................................................................................... 2
   1.3 Existing Applications ............................................................................................................... 3

2. NARRATIVE ....................................................................................................................................... 4
   2.1 Problem Statement .................................................................................................................... 4
   2.2 Motivation ................................................................................................................................. 4
   2.3 Product Description .................................................................................................................. 5
   2.4 Product Scope ........................................................................................................................... 6

3. APPLICATION DESIGN .................................................................................................................... 7
   3.1 Application Design and Architecture ......................................................................................... 7
   3.2 Design Flow ............................................................................................................................... 8
   3.3 Use Case Diagram ...................................................................................................................... 9
   3.4 Class Diagram ............................................................................................................................ 10
   3.5 Data Flow Diagram .................................................................................................................... 12
   3.6 Context Diagram ....................................................................................................................... 12
   3.7 Sequence Diagram .................................................................................................................... 13
   3.8 State Transition Diagram ........................................................................................................... 14

4. User Interface .................................................................................................................................... 16
   4.1 Login ....................................................................................................................................... 16
   4.2 Dashboard ................................................................................................................................. 17
   4.3 Edit/Preference Courses .......................................................................................................... 18
4.4 Edit/Delete Rooms .................................................................19
4.5 Add Course ........................................................................20
4.6 Add Room ...........................................................................21
4.7 Classroom Allocation .................................................................22
4.8 Erase Data .............................................................................23
5. IMPLEMENTATION ..................................................................................24
5.1 Algorithm ..............................................................................24
5.2 Read Data from Excel ..............................................................25
5.3 Setting Preference for a Course .................................................26
5.4 Delete All Data ...........................................................................27
5.5 Displaying Allocated Classroom Data ...........................................28
5.6 Saving the Allocated Room Data into the Database ......................29
5.7 Course Updating on the Database ...............................................30
6. TESTING AND EVALUATION ..........................................................31
6.1 Testing with Real Time Data .....................................................31
6.2 Login .........................................................................................31
6.3 Edit/Delete/Preference Courses .................................................33
6.4 Edit/Delete Rooms .....................................................................34
6.5 Load Excel Sheet .......................................................................35
6.6 Add Course .................................................................................36
6.7 Add Room ..................................................................................37
6.8 Class Room Allocation ..............................................................38
6.9 Erase All Data ............................................................................41
6.10 Logout .......................................................................................41
LIST OF FIGURES

Figure 3-1. TAMUCC classroom scheduling application Architecture .........................7
Figure 3-2. End User Design Flow ..................................................................................8
Figure 3-3. Use case Diagram for the application ..........................................................10
Figure 3-4. Class Diagram ............................................................................................11
Figure 3-5. Dataflow Diagram ......................................................................................12
Figure 3-6. Context Diagram ........................................................................................13
Figure 3-7. Class Diagram ............................................................................................14
Figure 3-8. State Transition Diagram ..........................................................................15
Figure 4-1. Login ...........................................................................................................16
Figure 4-2. Dashboard ....................................................................................................17
Figure 4-3. Edit/Delete Preference Courses ..................................................................18
Figure 4-4. Edit/Delete Rooms .....................................................................................19
Figure 4-5. Add Course .................................................................................................20
Figure 4-6. Add Room ....................................................................................................21
Figure 4-7. Classroom Allocation ..................................................................................22
Figure 4-8. Delete Data ..................................................................................................23
Figure 5-1. Code Snippet for the Algorithm .................................................................24
Figure 5-2. Code Snippet for Reading the Data from the Excel ....................................25
Figure 5-3. Code Snippet for Setting Preference for a Course .......................................26
Figure 5-4. Code Snippet for Deleting all the Data from the Database .........................27
Figure 5-5. Code Snippet for Displaying the allocated Classrooms in a New HTML Page ........................................................................................................28
Figure 5-6. Code Snippet for Saving the Allocation Data into the Database .................29
Figure 5-7. Code Snippet for Course Updating on the Database.................................30
Figure 6-2. Test Case: Login .................................................................32
Figure 6-3. Test Case: Login Successful ..................................................32
Figure 6-4. Test Case: Wrong Password....................................................33
Figure 6-5. Testing Edit/Delete/Preference Courses ....................................34
Figure 6-6. Test Case: Delete a Room .......................................................34
Figure 6-7. Test Case: Taking up Excel Sheet as an Input .........................35
Figure 6-8. Test Case: Successful Data Upload .........................................35
Figure 6-9. Test Case: Add a Course Successfully .....................................36
Figure 6-10. Failure to specify a field ......................................................36
Figure 6-11. Test Case: Adding a Room ....................................................37
Figure 6-12. Test Case: Failure to specify a field ......................................38
Figure 6-13. Test Case: Result Generation ...............................................39
Figure 6-14. Test Case: Print and save as PDF ........................................40
Figure 6-15. Test Case: Warning for Erase all Data ....................................41
1. BACKGROUND AND RATIONALE

1.1 Introduction

“The issue of allocation of resource to fulfil limitless human needs has been and will continue to be a worldwide phenomenon concerning managers, administrators, businessmen, heads of institutions alike.” [2]

Space allocation can be defined as the distribution of resources to regions of space, for example, rooms, satisfying all the necessities and requirements (Such as the department’s preference for a building) as much as possible. D. B. Varley (1998) defines space allocation as a process of allocating rooms or areas of space for specific functionality [2]. Thus, since it is limited it must in well manage by the faculties towards availability and suitable with the user required.

In this project, Classroom Space Allocation is defined as the accommodation of classroom space among various courses with various sizes of student populace to guarantee the ideal space utilization and the fulfillment of extra necessities and/or conditions. In this, a condition exists: the capacity of the classrooms and the space required by the courses and their timings cannot be modified. A better solution for this space allotment problem is one where every course is assigned a classroom, no space is wasted.
1.2 Scheduling Algorithm

The classroom scheduling algorithm used for the application has been specifically decided upon to be used for a big university as the number of constraints are high. Below is the description of the algorithm:

1: Start
2: For the second row of the Excel sheet to the last row
   {Read the input from every line;
    Assign labels to each classroom as A1, A2, A3.... etc;
    Read the input from each course capacity;
    Group classes as per timings (E.g. Monday 8:00-8:50 AM);
   }
3: Sort classrooms in ascending order of their student capacity
4: Sort courses in descending order of their capacity
5: For Course 0 to CourseMax-1
   {
    If ClassroomCapacity >= CourseCapacity
    {
     Assign Classroom;
    }
    Else
    {
     Keep Searching;
    }
   }
6: Iteratively create empty classrooms list for all timeslots.

10: Continue pairing for every timeslot. For timeslot MWF 8:00-8:50 AM for e.g. all the classrooms would be empty and a fresh allocation starts. When a classroom is already allocated, empty classrooms list is searched and classrooms are assigned.
Using the above algorithm, to effectively create a tool use of best available technologies is required.

1.3 Existing Applications

There are few existing commercial web based applications discussed below.

1.3.1 Classroom Bookings

Classroom bookings is a tool designed to be used by the faculty. Every teacher logs in to the system and updates their availability schedule. The classes are scheduled and corresponding classrooms are assigned to them. This is a messy process as there would be too many users involved, also there is a need to update the roster biweekly.

1.3.2 Visual Classroom Scheduler

This is a paid and costly application. It also comes with several additional and unwanted features. The main drawback of this application lies in the fact that this was designed to be used by a small school setup and not for a university with more than 1000 classes needed to be scheduled.
2. NARRATIVE

2.1 Problem Statement

Currently, at many universities, classroom allocation is done manually as a coordinated effort between the representatives from the Academic Departments (Called representatives from now for convenience) and the staff of the Registration Office. This manual process is time consuming and tedious as well as prone to human errors. In the manual approach currently used, each representative must submit information about the courses their department offers for a given semester to the staff of the Registration Office. These requests may include some specific classrooms. Most departments are given a specific subset of classrooms that they can schedule. The manual process requires a lot of time and coordination before the classroom allocation is finalized [13]. Despite this, the actual enrollment of a class is not yet known until the beginning of its semester; this is an important factor in the selection of a single classroom for a given course and hence a dynamic reallocation of classrooms often happens early on every semester. Compounding this problem, there is no procedure in place to compare alternate allocation strategies; therefore, in most cases the allocation of classrooms is done based on previous semesters’ allocations as well as the expertise of both the representatives and the Registration staff.

2.2 Motivation

To help staff of registration office overcome this problem manual classroom allocation, a web based tool has been created, it gives a better user experience and reduces the time taken by the manual process and also eliminated the possibility of human error.
2.3 Product Description

This web application will allow you to gain access to the administrator, who will be asked for the corresponding password to enable him/her to navigate through. When the username and password are validated the user is to redirected to the home page called the Dashboard where there is an option to upload the duly filled excel sheet whose contents would be the name of the course, department, student capacity of the course, and days and timings of each course.

The following is a brief description of how to navigate through the application.

- User opens the application from a web browser by entering the url.
- The application asks the user to enter the username and the corresponding password.
- On a successful login, the user will be able to navigate to the next screen.
- This screen is called the ‘Dashboard’ which will have a menu which categorizes the app functionality.

The following are the features of the application

Admin:

Upload excel sheet, View the schedule, Download schedule as a PDF

Records:

Add/Delete Buildings, Add/Delete Classrooms, Add/Delete Courses and their details.

The user is also provided with settings screen which has few settings that can be modified by the user and will also be able to logout from the application.
2.4 Product Scope

This app is developed for the staff of the registration office. An internet connection and a web browser are the only things needed to access it. The admin will be given a username and password (which can later be changed) to access the application. This application is compatible for the use on browsers such as Mozilla Firefox, Internet Explorer and Google Chrome.
3. APPLICATION DESIGN

3.1 Application Design and Architecture

Figure 3-1 shows an overview of the architecture of the application. The admin, in this case the registrar of the university, opens the app on a web browser. To start using the app and have access to the data the admin needs to first login to the system using his/her credentials. Only staff at the university registration office can have access to the app as proper credentials are needed for authentication purpose. The admin on his/her part uploads the duly filled excel sheet in to system.

![Image](image_url)

*Figure 3-1. TAMUCC classroom scheduling application Architecture*

The application is web based rendered using Apache Tomcat Server. The user needs to have an internet connection to have access to the data on the database, also it is required to make any necessary changes. Figure 3-1. is a reference image which shows the basic architecture of the application.
3.2 Design Flow

Figure 3.2 shows the basic design flow of all the functionalities the user has. The user logs into the app using his/her id and password. If the user is using the classroom scheduling for the first time i.e., if he/she is a new user, he/she needs is given a option to change the password from the given default password. If the registration is successful, he is redirected to the home page. If the user enters the wrong credentials, the app redirects to the login page until and unless he enters appropriate credentials.

![Figure 3-2. End User Design Flow](image)
If the user forgets his/her password, he/she can choose “forgot password?” option and input his/her university email id, if the university id is valid, the user is redirected to choose a new password. Once the login credentials are validated, the user is taken to the dashboard where he/she can choose an action. At the end, they can logout of the app using the logout option [7].

3.3 Use Case Diagram

The use case diagram (Figure 3-3.) demonstrates the sequence of operations that user will be able to perform with different modules of the application along with the associations. The user able to interact with the application after a successful login. The user will be able to view and update personal information, access class information, schedule history, view and update course and classroom catalogue
3.4 Class Diagram

The class diagram (i.e., Figure 3-4) shows the associations between different classes of the application. After the user logs into the application, a session is initiated and maintained using shared preferences. The application consists of a navigation drawer and which is the super class of many classes during the navigation.
When the user logs in to the app he has options like view/edit profile, add user, etc. In the “myprofile”, the user can view his profile details like the name, university id, email, phone number, mail id etc. He has an option to edit the data and save it. User inputs come in from the GUI layer and get processed by the Object layer; the Object layer is where all of the system’s logic resides. If the Object layer needs any data that is not currently held in its session variables is then calls the Service layer. The Service layer is where all the SOA services reside, these services handle all contact with the databases and provides an API for the Object layer to use.
3.5 Data Flow Diagram

![Data Flow Diagram]

*Figure 3-5. Dataflow Diagram*

3.6 Context Diagram

A system context diagram (SCD) in software engineering and systems engineering is a diagram that defines the boundary between the system, or part of a system, and its
environment, showing the entities that interact with it [2]. This diagram is a high-level view of a system. It is similar to a block diagram.

![Diagram](image)

*Figure 3-6. Context Diagram*

### 3.7 Sequence Diagram

A Sequence diagram is an interaction diagram that shows how objects operate with one another and in what order. It is a construct of a message sequence chart.
3.8 State Transition Diagram

State diagrams are used to give an abstract description of the behavior of a system. This behavior is analyzed and represented as a series of events that can occur in one or more possible states. Hereby "each diagram usually represents objects of a single class and track the different states of its objects through the system". 
Figure 3-8. State Transition Diagram
4. User Interface

4.1 Login

Figure 4-1. shows the screen which is shown when the app opens. It is the login screen in which the user needs to give in his credentials to login to the app. Initially only the admin can access the system. The login screen now shows Texas A and M University for reference purposes.

![Figure 4-1. Login](image-url)
4.2 Dashboard

Upon successful login, the dashboard of the application is displayed where there are a number of tools available to the user. Dashboard is shown in the Figure 4-2. below.

![Dashboard Image]

*Figure 4-2. Dashboard*
4.3 Edit/Preference Courses

During the process of manual classroom allocation, the universities face problems with a few important courses missing out on having a room allocated [17]. Hence, the user has been enabled with an option to turn on/off preference for a course. When the preference is turned on for a course, it is sure to get a classroom and also it would be the first to get one. When multiple courses are turned on for preference, the first one is assigned a classroom first followed by the second, third and so on in the order in which preference has been turned on (row by row). In Figure 4-3. It can clearly be seen how all the courses are displayed and how the preference can be turned on or off.

Figure 4-3. Edit/Delete Preference Courses
4.4 Edit/Delete Rooms

Lot of times it may so happen that a room number is changed or the building is remodeled to suit the newer necessities of the universities. Hence, user must be able to edit the details of a room and also be able to delete a room if need be. Figure 4-4 shows the edit/delete rooms tab.

![Edit/Delete Rooms](image)

*Figure 4-4. Edit/Delete Rooms*
4.5 Add Course

When a new course is introduced into a department, it can be added to the database using this functionality. As shown in Figure 4-5, the department can be easily selected from the drop down list that has been provided. The user additionally needs to enter “Course Name” which is the actual title of the course, “Course Number” which is an alphanumeric identifier of the course, “Capacity” which is the number of students a course allows to register, “Time” the exact timeslot of the course during the week.

Figure 4-5. Add Course
4.6 Add Room

Whenever a new room is added to a building the same can be added to our database using the Add Room module.

*Figure 4-6. Add Room*
4.7 Classroom Allocation

As shown in Figure 4-7 previously allocated list of classrooms can be seen here. Latest can be viewed just by clicking the “Generate New” button provided.

![Classroom Allocation](image)

*Figure 4-7. Classroom Allocation*
4.8 Erase Data

All the existing data is erased from the database. Figure 4-8 shows that by a simple click of the button ‘Delete Data” every detail pertaining to courses names, capacities and timings is erased. There is no way to retrieve this information, hence a warning is displayed before proceeding to erase the data. This functionality has been added to the application so as to reduce the data burden on the database.

Figure 4-8. Delete Data
5. IMPLEMENTATION

5.1 Algorithm

The code snippet in the Figure 5-1 shows how our algorithm works to search for a room for a course. Every course is searched for a room in the building that belongs to the department. When there is no room available in the preferred building, move forward to the next available room in other building.

```
parentloop: for (Map.Entry<String, List<RoomDto>> entry: copyroomsdatamap.entrySet())
{
    List<RoomDto> value = entry.getValue();
    System.out.println("KEY " + entry.getKey() + " COUNT" + entry.getValue().size());

    roomsloop: for (RoomDto roomDto : value)
    {
        System.out.println("SEARCHING IN ROOM" + roomDto.getRoomcode() + ""
        + roomDto.getDepartment());
        if (roomDto.getCapacity() < courseDto.getCourseCapacity())
        {
            System.out.println("No capacity");
            continue;
        }
    }
}
```

Figure 5-1. Code Snippet for the Algorithm
5.2 Read Data from Excel

The data from the input excel file is read row by row and is inserted into the database. Every row is incremented with the statement “Row row=rowIterator.next()” statement as seen in the Figure 5-2.

```java
while (rowIterator.hasNext()) {
    Row row = rowIterator.next();

    if (rowCount < 1) {
        rowCount++;
        continue;
    }

    RoomDto roomDto = new RoomDto();
    int counter = 0;

    if (row.getCell(counter) != null
        && row.getCell(counter).getCellType() != Cell.CELL_TYPE_BLANK) {

        String string = row.getCell(counter).getStringCellValue().trim();
        // System.out.println("LAST"+string.lastIndexOf(" "));
        String building = string.substring(0, string.lastIndexOf(" ")).trim();
    }
}
```

*Figure 5-2. Code Snippet for Reading the Data from the Excel*
5.3 Setting Preference for a Course

Every course that has the preference turned on for it is assigned tokens like 1, 2, 3, etc. and all these courses move to the top of the course list in the database so that the classroom can first be allotted to them.

```java
public void editPreference(String courseid, int preference) {
    try {
        Connection con = DB.getConnection();
        String query = "update course set preference=? where id=?";
        PreparedStatement coursestatement =
            con.prepareStatement(query,
            Statement.RETURN_GENERATED_KEYS);
        coursestatement.setInt(1, preference);
        coursestatement.setString(2, courseid);
        coursestatement.executeUpdate();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

*Figure 5-3. Code Snippet for Setting Preference for a Course*
5.4 Delete All Data

Using the statement “removeAll()” all the data can be serially removed from the database. Figure 5-4 shows how CoursesDB, RoomsDB and DepartmentsDB is deleted from the RoomAllocationDB.

```java
public class DeleteDataService {

  public void deleteAllData() {

    RoomAllocationDB roomAllocationDB = new RoomAllocationDB();
    roomAllocationDB.removeAll();

    CoursesDB coursesDB = new CoursesDB();
    coursesDB.removeAll();

    RoomsDB roomsDB = new RoomsDB();
    roomsDB.removeAll();

    DepartmentsDB departmentsDB = new DepartmentsDB();
    departmentsDB.deleteDepartments();

    // Figure 5-4. Code Snippet for Deleting all the Data from the Database
  }
}
```
5.5 Displaying Allocated Classroom Data

After the classrooms have been allocated. We need the list to be displayed in a new
page so that at any point the data can be used as a reference by the staff of the registration
office. Using the `new_Win_document_write(divToPrint_outerHTML)` statement the
output is shown in a new HTML page and from here it can be downloaded in a PDF form.

```javascript
<script>
    function printData() {
        var divToPrint =
                        document.getElementById("printTable");
        newWin = window.open("");

        newWin.document.write(divToPrint.outerHTML);
        newWin.print();
        newWin.close();
    }

    function generateNewAllocation() {
        document.getElementById("loading").style.display = 'block';
        $('#generateNewAllocation').click();
    }
</script>

Figure 5-5. Code Snippet for Displaying the allocated Classrooms in a New HTML Page
5.6 Saving the Allocated Room Data into the Database

Once all the rooms have been allocated the data needs to be saved to the database, so that a copy can be downloaded at any point of time. The snippet shown in Figure 5-6 performs this very functionality.

```java
public void addAll(List<RoomAllocationDto> roomallocations) {
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        System.out.println("SIZE" + roomallocations.size());
        for (RoomAllocationDto roomallocation : roomallocations) {
            String query =
                "insert into roomallocation (courseid, roomid)
                values(" + roomallocation.getCourseid()
                + "," + roomallocation.getRoomid() + ")";
            // System.out.println(query);
            statement.addBatch(query);
        }
        statement.executeBatch();
        statement.close();
        con.close();
    }
```

*Figure 5-6. Code Snippet for Saving the Allocation Data into the Database*
5.7 Course Updating on the Database

When the details of a particular course are updated, the same changes are made to its entry in the database using the code shown in Figure 5-7.

```java
public void editCourse(CourseDto courseDto) {
    try {
        Connection con = DB.getConnection();

        String query = "update course set department=?,
                        course=?,capacity=?,timing=? where id=?";

        PreparedStatement coursestatement =
            con.prepareStatement(query,
            Statement.RETURN_GENERATED_KEYS);

        coursestatement.setString(1,
          courseDto.getDepartment());
        coursestatement.setString(2,
          courseDto.getCourse());
        coursestatement.setInt(3,
          courseDto.getCapacity());
        coursestatement.setInt(5,
          courseDto.getId());
        coursestatement.setString(4,
          courseDto.getTiming());
        coursestatement.executeUpdate();
    }
}
```

Figure 5-7. Code Snippet for Course Updating on the Database
6.. TESTING AND EVALUATION

This application is tested on a laptop running on Windows 10 Operating System. The application starts and runs smoothly. The navigation of the application’s user interface has been tested and it is running fine. Testing needs to be done where the user input is necessary for the application.

6.1 Testing with Real Time Data

The application has been tested for its efficiency with the data of courses offered in our university in Spring 2017. The university’s current procedure yielded 754 (out of 1976) courses for which a class has not been allocated [7]. By using the application, the count has reduced to 216.

6.2 Login

This test case checked the smooth login process of the application. The application accepted the default password of “admin” (Figure 6-2) and redirected to the homepage quickly(Figure 6-3). When the password entered is wrong the login page keeps refreshing by emptying the contents of the password area(Figure 6-4)
Figure 6-1. Test Case: Login

Figure 6-2. Test Case: Login Successful
6.3 Edit/Delete/Preference Courses

The next functionality of our application is “Edit/Delete/Preference Courses” here the user can edit a course details, delete a course and/or give it preference during classroom allotment. Since deleting a course is a risky aspect, it has been checked for the warning and the application generates a warning very gently asking for confirmation (Figure 6-5).
6.4 Edit/Delete Rooms

When trying to delete a course the application reacted positively by generating a warning.
6.5 Load Excel Sheet

The important functionality of the application is to load the excel sheet. It gets totally uploaded with 3 minutes.

Figure 6-6. Test Case: Taking up Excel Sheet as an Input

Figure 6-7. Test Case: Successful Data Upload
6.6 Add Course

Next comes the add course functionality. Adding a course is simple and a confirmation is generated if the course has been added (Figure 6-9). If there is a missing field a gentle warning to fill it is generated (Figure 6-10).
6.7 Add Room

Next functionality is the Add Room functionality. When all the fields are duly filled, the application notifies the user that the room has been added (Figure 6-11). When a field is missing a warning is generated to fill the field (Figure 6-12).

![Figure 6-10. Test Case: Adding a Room](image-url)
Next is the classroom allocation functionality of the application. When the user clicks generate new button immediately the newly added/deleted/edited courses and classroom are read along with the old ones existing in the database. Then the classes and courses are mapped and the output is generated (Figure 6-15). A proper output in the required format is being displayed and the option to print the result is also working fine (Figure 6-14)
Figure 6-12. Test Case: Result Generation

Figure 6-13. Desired Input through the Excel Sheet
Figure 6-13. Test Case: Print and save as PDF
6.9 Erase All Data

Since the Erase all data functionality deletes all the data in the database a warning is desired. The application has generated a warning to confirm if the user really wants to erase all the data (Figure 6-15).

![Figure 6-14. Test Case: Warning for Erase all Data](image)

6.10 Logout

The logout functionality is working fine. The user is immediately redirected to the login page.

6.11 User Feedback

Several students were contacted to receive feedback, on the usability of the application. The students liked the outlook of the app, also they opinioned that it is user friendly and that it would be very helpful for the registrar of the university. Most of them
also thought that most their classes have been allotted in the classrooms belonging to their
department which is a timesaver.
7. CONCLUSION AND FUTURE WORK

This main aim of developing this application was to provide the staff of the registration office a web based tool for classroom allocation to save time and energy as using this application the process takes less than 5 minutes [17]. Every room now holds 19 classes each week on an average, as opposed to 5 to 10 classes a week [7], which is a better utilization of space. This application now allows the users:

- To allot classrooms for most of the courses.
- To change a schedule based on dynamic situations
- To view the empty classrooms at any given timeslot
- Effectively utilize every classroom.

The matching algorithm can handle different considerations such as: closest fit for class size (as described in the algorithm), departmental building preferences, and being able to prioritize courses. Some of the future enhancements that can be done are:

- User testing: Staff members who are currently involved in the classroom allocation of a University might want to test this application and give their feedback.
- The system can be upgraded to support the allocation of labs for courses that need them.
- There can be an update which takes into consideration the personal choice and comfort of the respective faculty.
- An automatic email system can be developed so that all the faculty members and students are notified of the allocated rooms immediately.
BIBLIOGRAPHY AND REFERENCES


This chapter has been included to make the future work easier. The following are the details of the technologies that have been used to create the application.

7.1.1 Tomcat Server

The Apache Tomcat® software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. The Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket specifications are developed under the Java Community Process. The Apache Tomcat Server has the following components:

7.1.2 Catalina

Catalina is Tomcat's servlet container. Catalina implements Sun Micro systems' specifications for servlet and JavaServer Pages (JSP). In Tomcat, a Realm element represents a "database" of usernames, passwords, and roles (similar to Unix groups) assigned to those users. Different implementations of Realm allow Catalina to be integrated into environments where such authentication information is already being created and maintained, and then use that information to implement Container Managed Security as described in the Servlet Specification.

7.1.3 Coyote

Coyote is a Connector component for Tomcat that supports the HTTP 1.1 protocol as a web server. This allows Catalina, nominally a Java Servlet or JSP container, to also act as a plain web server that serves local files as HTTP documents. [4]
Coyote listens for incoming connections to the server on a specific TCP port and forwards the request to the Tomcat Engine to process the request and send back a response to the requesting client. Another Coyote Connector, Coyote JK, listens similarly but instead forwards its requests to another web server, such as Apache, using the JK protocol.\[5\] This usually offers better performance.

### 7.1.4 Jasper

Jasper is Tomcat's JSP Engine. Jasper parses JSP files to compile them into Java code as servlets (that can be handled by Catalina). At runtime, Jasper detects changes to JSP files and recompiles them.

As of version 5, Tomcat uses Jasper 2, which is an implementation of the Sun Microsystems's JSP 2.0 specification. From Jasper to Jasper 2, important features were added:

- **JSP Tag library pooling** - Each tag markup in JSP file is handled by a tag handler class. Tag handler class objects can be pooled and reused in the whole JSP servlet.

- **Background JSP compilation** - While recompiling modified JSP Java code, the older version is still available for server requests. The older JSP servlet is deleted once the new JSP servlet has finished being recompiled.

- **Recompile JSP when included page changes** - Pages can be inserted and included into a JSP at runtime. The JSP will not only be recompiled with JSP file changes but also with included page changes.

- **JDT Java compiler** - Jasper 2 can use the Eclipse JDT (Java Development Tools) Java compiler instead of Ant and javac.
7.1.4.1 High availability

A high-availability facilitates the scheduling of system upgrades (e.g. new releases, change requests) without affecting the live environment. This is done by dispatching live traffic requests to a temporary server on a different port while the main server is upgraded on the main port. It is very useful in handling user requests on high-traffic web applications.

7.1.4.2 Web application

Tomcat has a feature for user- as well as system-based web applications to add support for deployment across the variety of environments. It also tries to manage sessions as well as applications across the network.

7.1.5 J2EE

The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications. J2EE extends the Java Platform, Standard Edition (Java SE), providing an API for object-relational mapping, distributed and multi-tier architectures, and web services. The platform incorporates a design based largely on modular components running on an application server. Software for J2EE is primarily developed in the Java programming language.

7.1.6 MySQL Server

MySQL is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL). SQL is the most popular language for adding, accessing and managing content in a database. It is most noted for its quick processing, proven reliability, ease and flexibility of use. It is used for three main reasons:
• High Performance: A unique storage-engine architecture allows database professionals to configure the MySQL database server specifically for particular applications, with the end result being amazing performance results.

• High Availability: MySQL offers a variety of high-availability options from high-speed master/slave replication configurations, to specialized Cluster servers offering instant failover, to third party vendors offering unique high-availability solutions for the MySQL database server.

• Open Source: Since most of its source code is available online for free. A user can modify it for his specific requirements and use effectively.

7.1.7 JDBC

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is part of the Java Standard Edition platform, from Oracle Corporation. It provides methods to query and update data in a database, and is oriented towards relational databases. Our system would use JDBC to connect and communicate with the database.

7.1.8 Front End

The following technologies are used to develop the front end that is the user interface.

• HTML 5: HTML5[note 1] is a markup language used for structuring and presenting content on the World Wide Web. It is the fifth and current version of the HTML standard.

• CSS3: Cascading Style Sheets (CSS) is a style sheet language used for
describing the presentation of a document written in a markup language (In our case it is HTML3).

- **jQuery**: jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML.

- **BootStrap Framework**: Bootstrap is a free and open-source front-end web framework for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Unlike many web frameworks, it concerns itself with front-end development only.

## 7.2 Sites Built Using Frameworks

### 7.2.1 Tomcat Server:

Tomcat is the most popular webserver available today. Linkedin.com, ebay.com, Zillow.com, snapdeal.com, aliexpress.com are all built with Tomcat.

### 7.2.2 J2EE:

Stack Overflow, jGuru, Java Lobby, JDocs, Devx, Developer.com

### 7.2.3 MySQL Server:

Pinterest, Verizon, Zappos.com, YouTube

All of these frameworks are open source. They can be easily modified for our requirements and adopted for better user experience and to get the job done easily and quickly. Classroom allocation which now takes a day to get done would take less than 5 minutes using our web based system.
APPENDIX II

DepartmentDB.java

package com.classroom.database;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
public class BuildingsDB {
    public ArrayList<String> buildings() {
        ArrayList<String> buildings = new ArrayList<>();
        try {
            Connection con = DB.getConnection();
            Statement statement = con.createStatement();
            String SQL = "SELECT DISTINCT building FROM `departments` ";
            ResultSet resultSet = statement.executeQuery(SQL);
            while (resultSet.next()) {
                buildings.add(resultSet.getString("building"));
            }
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return buildings;
    }
}

CoursesDB.java

package com.classroom.database;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import com.classroom.dto.CourseDto;
import com.classroom.dto.TimingDto;
import com.classroom.service.DateUtil;
import com.mysql.jdbc.Util;
public class CoursesDB {
    public void addCourses(ArrayList<CourseDto> courses) {
        try {
            Connection con = DB.getConnection();
            ...
Statement timingstatement = con.createStatement();
for (CourseDto courseDto : courses) {
    addCoursetoDB(con, timingstatement, courseDto);
}

timingstatement.close();
con.close();
} catch (Exception ex) {
    ex.printStackTrace();
}

public void addCourse(CourseDto courseDto) {
    try {
        Connection con = DB.getConnection();
        Statement timingstatement = con.createStatement();
        addCoursetoDB(con, timingstatement, courseDto);
        timingstatement.close();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

public void updateCourse(CourseDto courseDto) {
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

public void addCoursetoDB(Connection con, Statement timingstatement, CourseDto courseDto)
    throws Exception {
    String query =
        "insert into course (department, course,capacity,courseno,timing) values(?,?," + courseDto.getCoursecapacity() + "?,?)";
    System.out.println(query);
    PreparedStatement coursestatement = con.prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
    coursestatement.setString(1,  courseDto.getCoursedepartment());
    coursestatement.setString(2,   courseDto.getCourse());
    coursestatement.setString(3,   courseDto.getCourseno());
    coursestatement.setString(4,   courseDto.getTiminglabel());
    coursestatement.executeUpdate();
}
coursestatement.executeUpdate();
ResultSet rs = coursestatement.getGeneratedKeys();
boolean hasnext = rs.next();
System.out.println(hasnext);
int courseid = rs.getInt(1);
for (TimingDto timingDto : courseDto.getTimings()) {
    query = 
        "insert into timing (courseid, day, starttime, endtime) values( " + courseid + "," 
            + timingDto.getDay() + "," + timingDto.getStarttime() + "," 
            + timingDto.getEndtime() + ");"
    timingstatement.executeUpdate(query);
}

public List<CourseDto> readAllCourseInfo() {
    List<CourseDto> courses = new ArrayList<CourseDto>();
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        String SQL = "select * from course order by preference desc, capacity desc ";
        System.out.println(SQL);
        ResultSet resultSet = statement.executeQuery(SQL);
        while (resultSet.next()) {
            CourseDto course = new CourseDto();
            int id = resultSet.getInt("id");
            course.setCoursedepartment(resultSet.getString("department"));
            course.setCourse(resultSet.getString("course"));
            course.setCoursecapacity(resultSet.getInt("capacity"));
            course.setId(id);
            course.setPreference(resultSet.getInt("preference"));
            course.setCourseno(resultSet.getString("courseno"));
            Statement statement2 = con.createStatement();
            String SQL2 = " SELECT * FROM timing where courseid = " + id;
            ResultSet resultSet2 = statement2.executeQuery(SQL2);
            ArrayList<TimingDto> timings = new ArrayList<>();
            while (resultSet2.next()) {
                TimingDto timingDto = new TimingDto();
                timingDto.setCourseid(id);
                timingDto.setDay(resultSet2.getString("day"));
                timingDto.setStarttime(resultSet2.getInt("starttime"));
                timingDto.setEndtime(resultSet2.getInt("endtime"));
                timings.add(timingDto);
            }
            course.setTimings(timings);
            course.setTiminglabel(resultSet.getString("timing"));
            courses.add(course);
        }
        con.close();
    } catch (Exception e) {
public List<CourseDto> readAll() {
    List<CourseDto> courses = new ArrayList<CourseDto>();
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        String SQL = "select * from course order by id asc ";
        System.out.println(SQL);
        ResultSet resultSet = statement.executeQuery(SQL);
        while (resultSet.next()) {
            ReadCourse readCourse = new ReadCourse();
            CourseDto course = readCourse.getCourseByResultSet(resultSet, con);
            courses.add(course);
        }
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return courses;
}

public void deleteCourse(String courseid) {
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        String query = "DELETE FROM course where id = " + courseid;
        statement.executeUpdate(query);
        statement.close();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

public void removeAll() {
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        String query = "DELETE FROM course";
        statement.executeUpdate(query);
        statement.close();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
DB.java
package com.classroom.database;
import java.sql.Connection;
import java.sql.DriverManager;
public class DB {
    final static String USER = "root";
    final static String PASS = "";
    final static String url = "jdbc:mysql://localhost:3306/classroomallocation";
    public static Connection getConnection() {
        Connection connection = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            connection = DriverManager.getConnection(url, USER, PASS);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return connection;
    }
}

DepartmentsDB.java
package com.classroom.database;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import com.classroom.dto.CourseDto;
import com.classroom.dto.Department;
import com.classroom.dto.RoomDto;
public class DepartmentsDB {
    public void addDepartments(ArrayList<Department> departments) {
        try {
            Connection con = DB.getConnection();
            String query = "insert into departments (department, departmentcode,building)
            values(?,?,?)";
            PreparedStatement statement = con.prepareStatement(query,
                Statement.RETURN_GENERATED_KEYS);
            for (Department department : departments) {
                statement.setString(1, department.getDepartment());
                statement.setString(2, department.getCode());
                statement.setString(3, department.getBuilding());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
public void deleteDepartments() {
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        String query = "DELETE FROM departments";
        statement.executeUpdate(query);
        statement.close();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

public Department readDepartment(String id) {
    Department department = new Department();
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        String SQL = "select * from departments where id = " + id;
        System.out.println(SQL);
        ResultSet resultSet = statement.executeQuery(SQL);
        if (resultSet.next()) {
            department.setBuilding(resultSet.getString("building"));
            department.setDepartment(resultSet.getString("department"));
            department.setCode(resultSet.getString("departmentcode"));
            department.setId(resultSet.getInt("id"));
        }
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return department;
}

public void editDepartment(Department department) {
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        String query = "UPDATE departments set building = ?, department = ?, departmentcode = ? where id = ?";
        statement.setInt(4, department.getId());
        statement.setString(1, department.getBuilding());
        statement.setString(2, department.getDepartment());
        statement.setString(3, department.getCode());
        statement.executeUpdate(query);
        statement.close();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
String query = "update departments set department=?,
departmentcode=?,building=? where id=?";
PreparedStatement coursestatement =
    con.prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
coursestatement.setString(1, department.getDepartment());
coursestatement.setString(2, department.getCode());
coursestatement.setString(3, department.getBuilding());
coursestatement.setInt(4, department.getId());
coursestatement.executeUpdate();
con.close();
} catch (Exception ex) {
    ex.printStackTrace();
}

public List<Department> readAll() {
    List<Department> departments = new ArrayList<Department>();
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        String SQL = "select * from departments";
        ResultSet resultSet = statement.executeQuery(SQL);
        while (resultSet.next()) {
            Department department = new Department();
            department.setId(resultSet.getInt("id"));
            department.setDepartment(resultSet.getString("department"));
            department.setCode(resultSet.getString("departmentcode"));
            department.setBuilding(resultSet.getString("building"));
            departments.add(department);
        }
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return departments;
}

public void deleteDepartment(String id) {
    try {
        Connection con = DB.getConnection();
        String query = "delete from departments where id = " + id;
        System.out.println(query);
        PreparedStatement coursestatement =
            con.prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
        coursestatement.executeUpdate();
        con.close();
    } catch (Exception ex) {
EditCourseDB.java
package com.classroom.database;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.Statement;
import com.classroom.dto.CourseDto;
import com.classroom.dto.TimingDto;
public class EditCourseDB {
    public void editCourse(CourseDto courseDto) {
        try {
            Connection con = DB.getConnection();

            String query = "update course set department=?, course=?,capacity=?,timing=? where id=?";

            PreparedStatement coursestatement =
            con.prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
            coursestatement.setString(1, courseDto.getCoursedepartment());
            coursestatement.setString(2, courseDto.getCourse());
            coursestatement.setInt(3, courseDto.getCoursecapacity());
            coursestatement.setInt(5, courseDto.getId());
            coursestatement.setString(4, courseDto.getTiminglabel());
            coursestatement.executeUpdate();
            Statement deletestatement = con.createStatement();
            query = "delete from timing where courseid=" + courseDto.getId();
            deletestatement.executeUpdate(query);
            Statement statement = con.createStatement();
            for (TimingDto timingDto : courseDto.getTimings()) {
                query = 
                "insert into timing (courseid,day, starttime,endtime) values(" + 
                courseDto.getId() + "," + timingDto.getDay() + "," + timingDto.getStarttime() + "," + 
                timingDto.getEndtime() + ");"
                statement.executeUpdate(query);
            }
        }
        con.close();
    }
}
package com.classroom.database;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.Statement;
import com.classroom.dto.RoomDto;
import com.classroom.Statement.RooDto;

public class EditorDeleteRoomDB {

public void editRoom(RoomDto roomdto) {
    try {
        Connection con = DB.getConnection();
        String query = "update room set department=?, capacity=?, building=? where id=?";
        System.out.println(query);
        System.out.println(roomdto.getCapacity());
        PreparedStatement coursestatement =
        con.prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
        coursestatement.setString(1, roomdto.getDepartment());
        coursestatement.setInt(2, roomdto.getCapacity());
        coursestatement.setString(3, roomdto.getBuilding());
        coursestatement.setInt(4, roomdto.getId());
        coursestatement.executeUpdate();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

public void deleteRoom(String id) {
    try {
        Connection con = DB.getConnection();
        String query = "delete from room where id = " + id;
        System.out.println(query);
        PreparedStatement coursestatement =
        con.prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
        coursestatement.executeUpdate();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
}
PreferenceDB.java
package com.classroom.database;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.Statement;
public class PreferenceDB {
    public void editPreference(String courseid, int preference) {
        try {
            Connection con = DB.getConnection();

            String query = "update course set preference=? where id=?";
            PreparedStatement coursestatement =
                con.prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
            coursestatement.setInt(1, preference);
            coursestatement.setString(2, courseid);
            coursestatement.executeUpdate();
            con.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}

ReadCourse.Java
package com.classroom.database;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import com.classroom.dto.CourseDto;
import com.classroom.service.DateUtil;
public class ReadCourse {
    public CourseDto getCourseById(String id) {
        CourseDto courseDto = null;
        try {
            Connection con = DB.getConnection();

            CourseDto courseDto = null;
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
Statement statement = con.createStatement();
String SQL = "select * from course where id = " + id;

System.out.println(SQL);
ResultSet resultSet = statement.executeQuery(SQL);

while (resultSet.next()) {
    courseDto = getCourseByResultSet(resultSet, con);
}
con.close();
} catch (Exception e) {
e.printStackTrace();
}
return courseDto;

public boolean getconcatanatedtiming = false;

public CourseDto getCourseByResultSet(ResultSet resultSet, Connection con) throws Exception {
    CourseDto course = new CourseDto();
    int id = resultSet.getInt("id");
    course.setCourseDepartment(resultSet.getString("department"));
    course.setCourse(resultSet.getString("course"));
    course.setCourseCapacity(resultSet.getInt("capacity"));
    course.setId(id);
    course.setPreference(resultSet.getInt("preference"));
    course.setTimingLabel("" + resultSet.getString("timing"));
    course.setCourseNo(resultSet.getString("courseno"));
    // System.out.println("TEST");

    if (getconcatanatedtiming) {
        Statement statement2 = con.createStatement();
        String SQL2 =
            " SELECT id, GROUP_CONCAT( DAY SEPARATOR '' ) as timing,
            starttime, endtime FROM timing GROUP BY courseid HAVING courseid =" + id;
ResultSet resultSet2 = statement2.executeQuery(SQL2);
if (resultSet2.next()) {

    String time = "";
    time = time + resultSet2.getString("timing");
    String starttime = DateUtil.calculateTime(resultSet2.getInt("starttime"), false);
    String endtime = DateUtil.calculateTime(resultSet2.getInt("endtime"), true);
    time = time + " " + starttime + "-" + endtime;
    System.out.println(time);
    course.setTimingLabel(time);
}

return course;
}

RoomAllocationDB.java
package com.classroom.database;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import com.classroom.dto.RoomAllocationDto;
import com.classroom.dto.RoomDto;
import com.classroom.service.DateUtil;
public class RoomAllocationDB {
    public void removeAll() {
        try {
            Connection con = DB.getConnection();
            Statement statement = con.createStatement();

            String query = "DELETE FROM roomallocation";
            statement.executeUpdate(query);
            statement.close();
            con.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    public void addAll(List<RoomAllocationDto> roomallocations) {
        try {
            Connection con = DB.getConnection();
            Statement statement = con.createStatement();
            System.out.println("SIZE" + roomallocations.size());
        }
    }
}
for (RoomAllocationDto roomallocation : roomallocations) {
    String query =
        "insert into roomallocation (courseid, roomid) values(" +
    roomallocation.getCourseid()
        + "," + roomallocation.getRoomid() + ")";
    // System.out.println(query);
    statement.addBatch(query);
}
statement.executeBatch();
statement.close();
con.close();
} catch (Exception ex) {
    ex.printStackTrace();
}
}
public List<RoomAllocationDto> readAll() {
    List<RoomAllocationDto> roomallocations = new
    ArrayList<RoomAllocationDto>();
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        String SQL =
            "select C.timing as timinglabel,
                RA.courseid,RA.roomid,R.roomnumber,R.building,R.capacity,C.course,C.department
                as coursedepartment,C.courseno,C.capacity as coursecapacity from room
                R,roomallocation RA,course C where RA.courseid =C.id and RA.roomid = R.id and
                RA.roomid<>-1 order by RA.id asc";
        ResultSet resultSet = statement.executeQuery(SQL);
        while (resultSet.next()) {
            RoomAllocationDto roomAllocationDto = new RoomAllocationDto();
            roomAllocationDto.setCourseid(resultSet.getInt("courseid"));
            roomAllocationDto.setRoomid(resultSet.getInt("roomid"));
            RoomDto roomDto = new RoomDto();
            roomDto.setRoomnumber(resultSet.getString("roomnumber"));
            roomDto.setDepartment(resultSet.getString("building"));
            roomDto.setCapacity(resultSet.getInt("capacity"));
            roomAllocationDto.setRoomDto(roomDto);
            roomAllocationDto.setCourse(resultSet.getString("course"));
            roomAllocationDto.setCoursedepartment(resultSet.getString("coursedepartment") + "+" +
                resultSet.getString("courseno"));
            // Statement statement2 = con.createStatement();
            // String SQL2 =
                // " SELECT id, GROUP_CONCAT( DAY SEPARATOR " ) as timing,
                // starttime, endtime FROM timing GROUP BY courseid HAVING courseid ="
            RoomAllocationDto roomAllocationDto = new RoomAllocationDto();
        }
    }
}
20

 + roomAllocationDto.getCourseid();

 //
 //        ResultSet resultSet2 = statement2.executeQuery(SQL2);
 //        if (resultSet2.next()) {
 //          String time = "";
 //          time = time + resultSet2.getString("timing");
 //          String starttime = DateUtil.calculateTime(resultSet2.getInt("starttime"), false);
 //          String endtime = DateUtil.calculateTime(resultSet2.getInt("endtime"), true);
 //          time = time + " " + starttime + ":" + endtime;
 //          roomAllocationDto.setTiminglabel("" + time);
 //        }
 //        roomallocations.add(roomAllocationDto);
 //      }
 //      con.close();
 //    } catch (Exception e) {
 //      e.printStackTrace();
 //    }
 //    return roomallocations;

 public List<RoomAllocationDto> readUnallotted() {
  List<RoomAllocationDto> roomallocations = new ArrayList<RoomAllocationDto>();
  try {
    Connection con = DB.getConnection();
    Statement statement = con.createStatement();
    String SQL = "select C.timing as timinglabel, RA.courseid, RA.roomid, R.roomnumber, R.building, R.capacity, C.course, C.department as coursedepartment, C.courseno, C.capacity as coursecapacity from room R, roomallocation RA, course C where RA.courseid = C.id and RA.roomid = R.id and RA.roomid = -1 order by RA.id asc";
    ResultSet resultSet = statement.executeQuery(SQL);
    while (resultSet.next()) {
      RoomAllocationDto roomAllocationDto = new RoomAllocationDto();
      roomAllocationDto.setCourseid(resultSet.getInt("courseid"));
      roomAllocationDto.setRoomid(resultSet.getInt("roomid"));
      RoomDto roomDto = new RoomDto();
      roomDto.setRoomnumber(resultSet.getString("roomnumber"));
      roomDto.setDepartment(resultSet.getString("building"));
      roomDto.setCapacity(resultSet.getInt("capacity"));
      roomAllocationDto.setRoomDto(roomDto);
      roomAllocationDto.setCourse(resultSet.getString("course"));
      roomAllocationDto.setCoursecapacity(resultSet.getInt("coursecapacity"));
      roomAllocationDto.setCoursedepartment(resultSet.getString("coursedepartment"));
      roomAllocationDto.setCourseno(resultSet.getInt("courseno"));
      roomAllocationDto.setTiminglabel(resultSet.getString("timinglabel"));
      roomallocations.add(roomAllocationDto);
    }
  } catch (Exception e) {
    e.printStackTrace();
  }
  return roomallocations;
}
roomAllocationDto.setCoursedepartment(resultSet.getString("coursedepartment")+-
"+resultSet.getString("courseno");
    roomAllocationDto.setTiminglabel(resultSet.getString("timinglabel");

    roomallocations.add(roomAllocationDto);
}  
    con.close();
} catch (Exception e) {
    e.printStackTrace();
}
    return roomallocations;
}

RoomsDB.java
package com.classroom.database;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import com.classroom.dto.CourseDto;
import com.classroom.dto.RoomDto;
import com.classroom.service.DateUtil;
public class RoomsDB {
    public void addRooms(ArrayList<RoomDto> roomslist) {
        try {
            Connection con = DB.getConnection();
            String query =
                    "insert into room (roomnumber,
department,capacity,roomcode,building,timing) values (?,?,?,?,?,?)";
            PreparedStatement statement = con.prepareStatement(query,
Statement.RETURN_GENERATED_KEYS);
            for (RoomDto roomDto : roomslist) {
                statement.setString(1, roomDto.getRoomnumber());
                statement.setString(2, roomDto.getDepartment());
                statement.setInt(3, roomDto.getCapacity());
                statement.setString(4, roomDto.getRoomcode());
                statement.setString(5, roomDto.getBuilding());
                statement.setString(6, roomDto.getTiming());
                System.out.println(statement.toString() + ";");
                statement.addBatch();
            }
            statement.executeBatch();
            statement.close();
        }
    }
}
public RoomDto readRoomById(String id) {
    RoomDto room = new RoomDto();
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        String SQL = "select * from room where id =" + id;
        ResultSet resultSet = statement.executeQuery(SQL);
        if (resultSet.next()) {
            room.setDepartment(resultSet.getString("department"));
            room.setRoomnumber(resultSet.getString("roomnumber"));
            room.setCapacity(resultSet.getInt("capacity"));
            room.setRoomcode(resultSet.getString("roomcode"));
            room.setId(Integer.parseInt(id));
        }
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return room;
}

public List<RoomDto> readAll() {
    List<RoomDto> rooms = new ArrayList<RoomDto>();
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();
        String SQL = "select * from room where id <>-1 order by capacity desc";
        ResultSet resultSet = statement.executeQuery(SQL);
        while (resultSet.next()) {
            RoomDto room = new RoomDto();
            int id = resultSet.getInt("id");
            room.setDepartment(resultSet.getString("department"));
            room.setRoomnumber(resultSet.getString("roomnumber"));
            room.setCapacity(resultSet.getInt("capacity"));
            room.setTiming(resultSet.getString("timing"));
            room.setBuilding(resultSet.getString("building"));
            room.setRoomcode(resultSet.getString("roomcode"));
            room.setId(id);
            rooms.add(room);
        }
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
public void removeAll() {
    try {
        Connection con = DB.getConnection();
        Statement statement = con.createStatement();

        String query = "DELETE FROM room where id <> -1";
        statement.executeUpdate(query);
        statement.close();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

RoomAllocationService.java
package com.classroom.service;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import com.classroom.database.CoursesDB;
import com.classroom.database.DepartmentsDB;
import com.classroom.database.RoomAllocationDB;
import com.classroom.database.RoomsDB;
import com.classroom.dto.CourseDto;
import com.classroom.dto.Department;
import com.classroom.dto.RoomAllocationDto;
import com.classroom.dto.RoomDto;
import com.classroom.dto.TimingDto;

public class RoomAllocationService {

    public void generateNewAllocation() {
        RoomAllocationDB roomAllocationDB = new RoomAllocationDB();
        roomAllocationDB.removeAll();

        CoursesDB coursesDB = new CoursesDB();
        List<CourseDto> coursesdata = coursesDB.readAllCourseInfo();

        ArrayList<RoomAllocationDto> roomallocations = new ArrayList<RoomAllocationDto>();

        // Code for generating new allocation...
    }
}
RoomsDB roomsDB = new RoomsDB();
List<RoomDto> roomsdata = roomsDB.readAll();
roomsdata = Util.filterrooms(roomsdata);
System.out.println("ROOMS SIZE"+roomsdata.size());

DepartmentsDB departmentsDB = new DepartmentsDB();
List<Department> departments = departmentsDB.readAll();
Map<String, String> departmentsmap = new HashMap<>();
for (Department department : departments) {
    departmentsmap.put(department.getCode(), department.getBuilding());
}
Map<String, List<TimingDto>> allotedroomtimings = new HashMap<>();
for (CourseDto courseDto : coursesdata) {
    RoomDto allotedRoom = findAvailableRoom(roomsdata, allotedroomtimings, courseDto, departmentsmap.get(courseDto.getCoursedepartment()));
    RoomAllocationDto roomAllocationDto = new RoomAllocationDto();
    roomAllocationDto.setCourseid(courseDto.getId());
    if (allotedRoom != null) {
        courseDto.setRoomDto(allotedRoom);
        if (allotedroomtimings.get(allotedRoom.getRoomcode()) == null) {
            allotedroomtimings.put(allotedRoom.getRoomcode(), courseDto.getTimings());
            System.out.println(courseDto.getTimings() + "allotedRoom.getRoomcode()" + allotedroomtimings.get(allotedRoom.getRoomcode()));
        } else {
            List<TimingDto> roomreservedtimings = allotedroomtimings.get(allotedRoom.getRoomcode());
            roomreservedtimings.addAll(courseDto.getTimings());
            allotedroomtimings.put(allotedRoom.getRoomcode(), roomreservedtimings);
        }
        roomAllocationDto.setRoomid(allotedRoom.getId());
    } else {
        roomAllocationDto.setRoomid(-1);
    }
    roomallocations.add(roomAllocationDto);
}

System.out.println("ROOM L"+roomallocations.size());
roomAllocationDB.addAll(roomallocations);

public RoomDto findAvailableRoom(List<RoomDto> roomsdata,
Map<String, List<TimingDto>> allotedroomtimings, CourseDto courseDto, String building) {
    System.out.println("Alloting : " + courseDto.getCourseno());
    System.out.println("Course department : " + courseDto.getCoursedepartment());
    ArrayList<RoomDto> copyroomsdata = new ArrayList<RoomDto>(roomsdata);
    System.out.println("Building"+building);
    Map<String, List<RoomDto>> copyroomsdatamap =
        Util.groupByDepartment(copyroomsdata, building);
    RoomDto allotedRoom = null;
    ArrayList<TimingDto> timings = courseDto.getTimings();
    parentloop: for (Map.Entry<String, List<RoomDto>> entry :
        copyroomsdatamap.entrySet()) {
        List<RoomDto> value = entry.getValue();
        System.out.println(" KEY " + entry.getKey() + " COUNT" +
            entry.getValue().size());
        roomsloop: for (RoomDto roomDto : value) {
            System.out.println("SEARCHING IN ROOM" + roomDto.getRoomcode() + " "
                + roomDto.getDepartment());
            if (roomDto.getCapacity() < courseDto.getCoursecapacity()) {
                System.out.println("No capacity");
                continue;
            }
            if (allotedroomtimings.get(roomDto.getRoomcode()) == null) {
                System.out.println("Alloting room"+roomDto.getRoomcode());
                allotedRoom = roomDto;
                break parentloop;
            } else {
                List<TimingDto> roomreservedtimings =
                    allotedroomtimings.get(roomDto.getRoomcode());
                boolean isavailable = false;
                for (TimingDto timingDto : timings) {
                    int starttime = timingDto.getStarttime();
                    int endtime = timingDto.getEndtime();
                    String day = timingDto.getDay();
                    for (TimingDto roomreservedtimingDto : roomreservedtimings) {
                        System.out.println(roomreservedtimingDto.getDay() + " " + day);
                        if (roomreservedtimingDto.getDay().equals(day)) {
                            // System.out.println(starttime);
                            // System.out.println(endtime);
                            // System.out.println(roomreservedtimingDto.getStarttime());
                            // System.out.println(roomreservedtimingDto.getEndtime());
                            // System.out.println((starttime >= roomreservedtimingDto.getStarttime() &&
                                endtime <=
                                // roomreservedtimingDto
                                // .getStarttime()));
                            endtime <=
                        }
                    }
                }
            }
        }
    }
}
if ((starttime < roomreservedtimingDto.getStarttime() && endtime <= roomreservedtimingDto.getStartTime()) || (starttime >= roomreservedtimingDto.getEndtime())) {
    isavailable = true;
} else {
    System.out.println("Not available");
    isavailable = false;
    continue roomsloop;
}
} else {
    isavailable = true;
}
}
if (isavailable) {
    allottedRoom = roomDto;
    break parentloop;
}
}
}
System.out.println("\n\n");
return allottedRoom;
}
public static void main(String[] args) {
    new RoomAllocationService().generateNewAllocation();
}

DateUtil.java
package com.classroom.service;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import com.classroom.dto.TimingDto;
public class DateUtil {

    public static String adjustCourseTimings(String time) {
        String newtime = time;
        newtime = newtime.replaceAll(":00", "");
        if (newtime.charAt(newtime.indexOf(" ") + 1) == '0') {
            newtime = newtime.substring(0, newtime.indexOf(" ") + 1) + "" + newtime.substring(newtime.indexOf(" ") + 2);
        }
        if (newtime.charAt(newtime.indexOf(".") + 1) == '0') {
            
        } else {
            
        }
        }
    }
    }
    System.out.println("\n\n");
    return allottedRoom;
}
public static void main(String[] args) {
    new RoomAllocationService().generateNewAllocation();
}

DateUtil.java
package com.classroom.service;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import com.classroom.dto.TimingDto;
public class DateUtil {

    public static String adjustCourseTimings(String time) {
        String newtime = time;
        newtime = newtime.replaceAll(":00", "");
        if (newtime.charAt(newtime.indexOf(" ") + 1) == '0') {
            newtime = newtime.substring(0, newtime.indexOf(" ") + 1) + "" + newtime.substring(newtime.indexOf(" ") + 2);
        }
        if (newtime.charAt(newtime.indexOf(".") + 1) == '0') {
            
        } else {
            
        }
        }
    }
    }
    System.out.println("\n\n");
    return allottedRoom;
}
newtime = newtime.substring(0, newtime.indexOf("-") + 1) + "+" + newtime.substring(newtime.indexOf("-") + 2);
}
return newtime;

public static String adjustRoomTimings(String roomtiming) {

String timing = "";
if (!roomtiming.endsWith("M"))
    roomtiming = roomtiming + "M";
return roomtiming;
}

public static String calculateTime(long seconds, boolean appendmeridian) {
    long minutes = seconds % 60;
    long hours = seconds % 3600 / 60;
    String formattedhours = String.format("%02d", hours);
    String formattedminutes = String.format("%02d", minutes);
    String time = formattedhours + ":" + formattedminutes;

    try {
        final SimpleDateFormat sdf = new SimpleDateFormat("H:mm");
        final Date dateObj = sdf.parse(time);
        String format = "K:mm";
        if (appendmeridian) {
            format = "K:mma";
        }
        return new SimpleDateFormat(format).format(dateObj);
    } catch (final ParseException e) {
        e.printStackTrace();
        return null;
    }
}

public static ArrayList<TimingDto> stringtoTime(String timerange) {
    ArrayList<TimingDto> timingDtolist = new ArrayList<TimingDto>();
    String weeks = timerange.split(" ")[0];
    String time = timerange.split(" ")[1];
    String a = "AM";
if (time.contains("PM")) {
    a = "PM";
}

time = time.replaceAll("AM", "").replaceAll("PM", "");
String startsat = time.split("-")[0];
String endsat = time.split("-")[1];

int starttime = timetoMinutes(startsat + " " + a);
int endtime = timetoMinutes(endsat + " " + a);

for (int i = 0; i < weeks.length(); i++) {
    char c = weeks.charAt(i);
    TimingDto timingDto = new TimingDto();
    timingDto.setDay("" + c);
    timingDto.setStarttime(starttime);
    timingDto.setEndtime(endtime);
    timingDtolist.add(timingDto);
}

return timingDtolist;
}

public static int timetoMinutes(String time) {
    Calendar cal = Calendar.getInstance();
    SimpleDateFormat sdf = new SimpleDateFormat("hh:mm a");
    Date date = null;
    try {
        date = sdf.parse(time);
        System.out.println(date);
    } catch (ParseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    cal.setTime(date);
    int mins = cal.get(Calendar.HOUR_OF_DAY) * 60 + cal.get(Calendar.MINUTE);
    return mins;
}

public static void main(String[] args) {
    // ArrayList<TimingDto> timingslist = stringtoTime("TR 01:00-01:50PM");
    // for (TimingDto timingDto : timingslist) {
    //    System.out.println(timingDto.getDay());
    //    System.out.println(timingDto.getStarttime());
    //    System.out.println(timingDto.getEndtime());
    // }
    System.out.println(calculateTime(1125, false));
}

}