**College Management System – Case study**

**Problem Statement:**

Any college will have a diverse pool of departments which include course-handling faculties and students from multiple-disciplines. So, it becomes very crucial to maintain the information of all the course-handling faculty and the students of the college. Assuming we do not have a well-structured system in place to manage the information, we’ll be forced to use the age-old paper-and-pen system of maintaining information. This leads us to an array of difficulties.

Firstly, in an environment where updates are frequently made to existing information, the pen-and-paper approach is going to be extremely painful. Secondly, if we need to locate one particular student/faculty from thousands of them, it will take a lot of time to do the same. This leads to wastage of time and energy. Lastly, security becomes a big issue when information is not stored properly as there are high chances of it landing up in the wrong hands. In order to avoid all such and much more problems, a well-planned, organized and structured system becomes essential.

The main objective of college management system is to automate all functionalities of a college or university. There are a lot of advantages of such a system over a haphazard, unorganized system. Some of them are:

1. Inconsistency of data is drastically reduced as there are a number of checks in place before a piece of information can be added or updated into the system, which can’t be done on the former way of managing information.
2. Data security is a big uncertainty in other types of management systems because the information can be easily accessed by unauthorized persons. But, in an automated system, there are a number of security checks and firewalls that block such unwanted users from accessing information.
3. The end product is going to be far easier to access to the users of the system because everything is in place, whereas in other forms of managing data, several discrepancies may occur while using the system and it will cause a lot of ambiguity and discomfort to the users.

Using this system, we call carry-out all college-management work like fees payment, attendance and result announcement, Sports House allocation, Borrowing and returning books from Library, course registration, hosteller check-in and check-out details, access to student details and student feedback in a precise way.

There are three types of users in our system. Namely, **Student, Faculty, Admin.**

**Student** can pay fees, view marks and attendance, house Leader board, borrow and return books, register for new courses and view his/her personal details.

**Staff** can provide marks and attendance to student, borrow and return book from library and modify the house leader board.

**Admin** can view student, staff and hosteller details and add check-in and check-out details for hosteller.

The various tasks carried out by the actors are discussed in detail in the Use cases below.

**Actors:**

**Primary:**

* Student
* Staff
* Admin

**Secondary:**

* Hosteller
* Library
* Sports House
* User credential database
* Fees management database
* Leader Board
* Student details database
* Staff details database
* Hosteller details database
* Attendance and mark registry database
* House details database
* Library record database
* Course registration database
* Hostel registry database

**Tasks:**

* Log in
* Pay fees
* Allocate and View marks & Attendance
* Borrow and return book
* Allocate house
* Modify and view Leader board
* Add and remove course
* View student, staff and hostel details
* Check in and check out

**Use Cases (for each task):**

**Login and signup:**

**Description:**

Users like Student, Teachers, Admin can sign up or login to their account to access the management system.

**Actors:  
Primary:**

* Any

**Secondary:**

* User credentials Database

**Trigger**:

This module is triggered when the user enters this management system.

**Preconditions:**

* The user doesn’t have an existing account for signup.
* The user must have an existing account for login.
* The user has active internet connection

**Basic Flow: (Sunny Day Scenario)**

1. The user selects the "Signup” option if they do not have an existing account.
2. Enter the Username and the password that he/she wants for the account
3. Enter his/her designation (Admin, Student, Exam cell, Faculty)
4. A new tuple is created in the User credentials database containing the login credentials of the new user
5. Displays Sign Up successful.
6. After signup they can login to their account using username, password and with their designation.
7. Login Credential Database will check the login credential of the user is valid.
8. If the credentials are invalid then it asks the user to try again
9. After successful tries it displays Login successful
10. According to their designation the details will be displayed

**Alternates:**

If user forgets his/her login credentials they can contact college for retrieving their account

**Exceptions: Nil**

**Level:**

* Create a new user account - blue level
* User login into their account - blue level

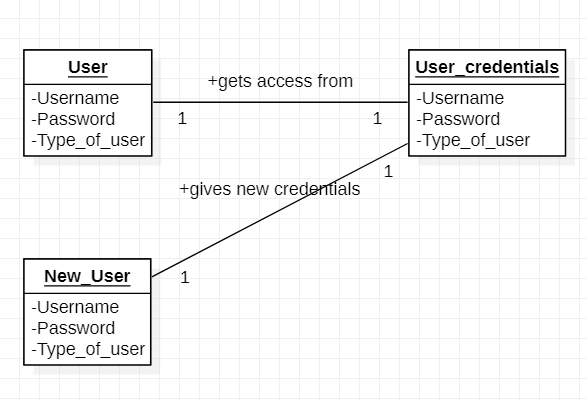
**Post condition:**

The account is created successfully  
Then Login successful

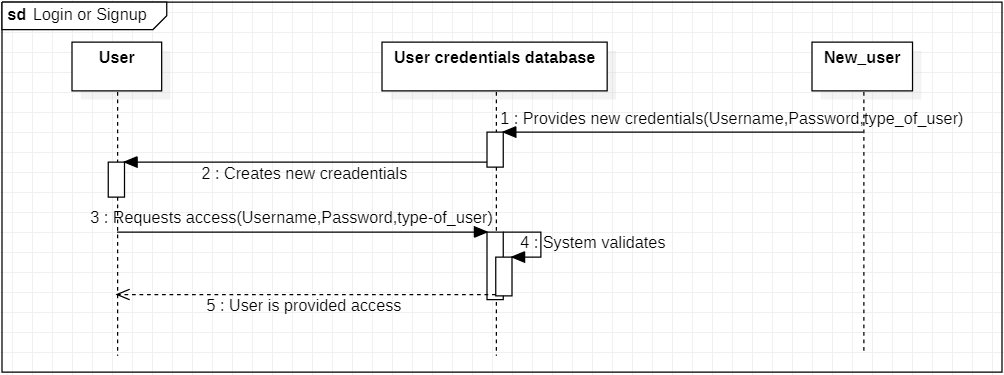
**Stakeholders:**

* User

**Object Diagram:**

****

**Time-sequence Diagram:**



**Allocate House:**

**Description:**

Initially when the student is relatively new to the system, this module uses Randomize function to allocate a house to the student on a random basis.

**Actors:**

**Primary:**

* Student
* Sports House

**Secondary:**

* House details record

**Trigger:**

This use case is triggered when the student selects the “Allocate House” option.

**Precondition:**

* The student should have a stable internet connection.
* The student is not allocated any house until then.

**Basic flow: (Sunny day scenario)**

* Student selects the “Allocate House” option.
* Randomize function is called which chooses a random number between 1 to 4.
* Each number is mapped with one house and according to the random number, the student is given a House.
* The name of the House is updated along with the student roll no in the House details record.

**Alternates: Nil**

**Exceptions:**

* In case the student loses internet connection, the module is halted in between and any partial records updated in the table are deleted.

**Level:**

* Allocating a House – Blue Level
* Sports House Management – White Level

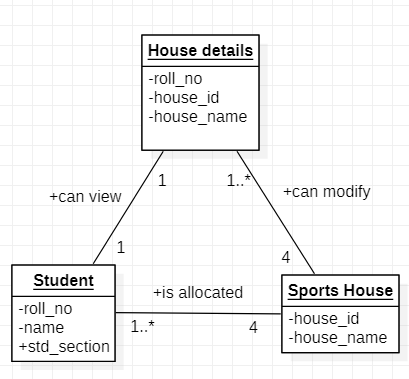
**Postcondition:**

* The student is allocated a House.

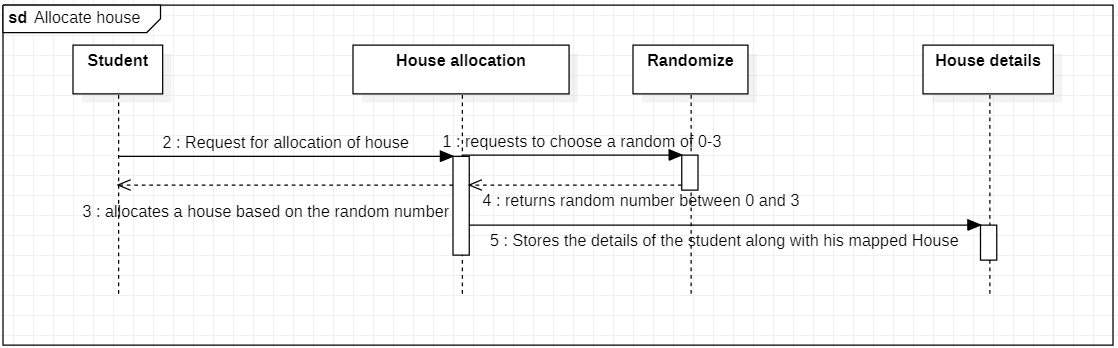
**Stakeholders:**

* Student
* Sports House

**Object diagram:**



**Time-sequence Diagram:**



**Pay fees:**

**Description:**

Students who have not paid their fees can pay their fees through this module.

**Actors:**

**Primary:**

* Student

**Secondary:**

* Fees management Database
* Admin

**Trigger:**

This use case is triggered when the student selects “Pay fees” option.

**Precondition:**

* The student has a stable internet connection.
* The student has not paid the fees till then.
* The student has access to a bank credit/debit card to pay his/her fees.

**Basic Flow: (Sunny Day Scenario)**

1. The student selects the “Pay Fees” option.
2. The amount to be paid by the student is displayed.
3. The student selects the “Proceed to payment” option.
4. The student is asked to enter the card number, pin, cardholder name and bank name.
5. The student is informed that the fees payment was successful/Incomplete.
6. The admin can inspect whether the student has paid the fees using the fees management database.

**Alternate:**

1. As an alternate, the student can directly visit the campus to make an offline fees payment.

**Exception:**

1. If the student loses internet connection in between the transaction, the transaction is cancelled and the money is refunded.

**Level:**

* Fees Payment – Blue level

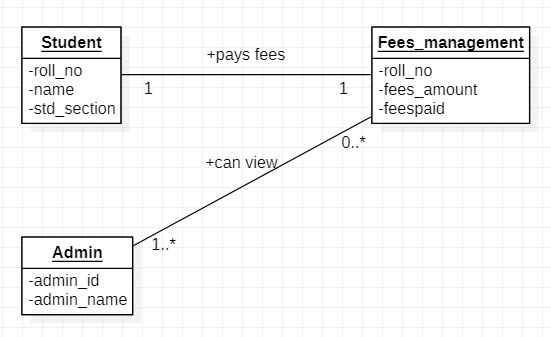
**Postcondition:**

1. The student is displayed a receipt that his/her fees payment was successful.
2. In case the process is halted in between, the fees paid by the student is refunded.

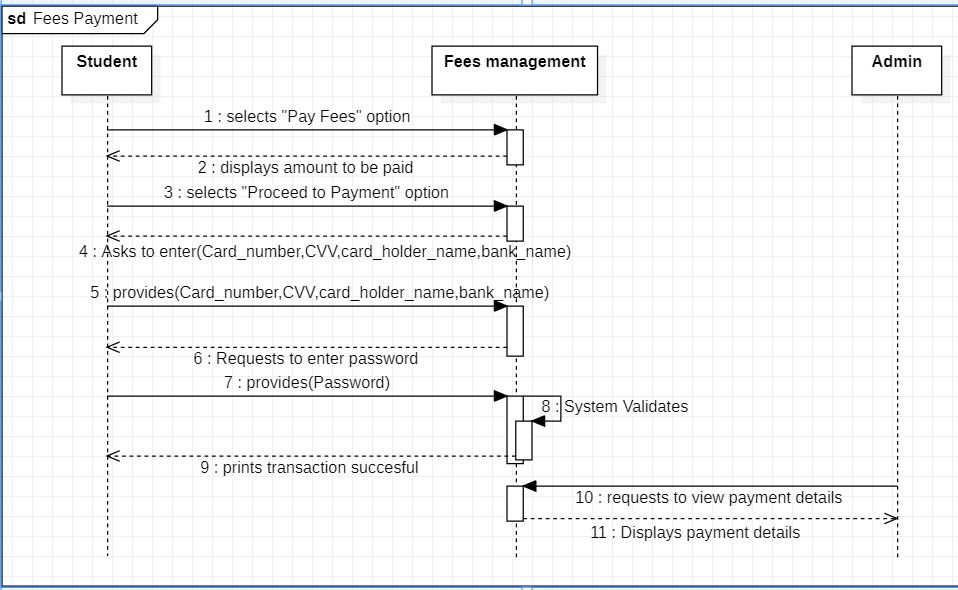
**Stakeholders:**

1. College Management
2. Student

**Object Diagram:**



**Time-sequence diagram:**



**Modify and View Leader board:**

**Description:**

The staffs can view and modify the Sports House Leader Board according to the points of each house and the students can view the Leader board from their system.

**Actors:**

**Primary:**

* Student
* Staff

**Secondary:**

* Leader Board database

**Trigger:**

This use case is triggered when the staff selects “Modify the Leader Board” option.

**Precondition:**

* The student/staff should have a stable internet connection.
* All the events corresponding to the sports houses should be completed before modifying the Leader Board.

**Basic Flow: (Sunny Day Scenario)**

1. The staff selects “Modify the Leader Board” option.
2. The staff is prompted to choose one of the four houses.
3. The staff is allowed to change the chosen House’s points.
4. If the staff wants to continue, the module starts again from step 2.
5. Else the changes made by the staff is updates in the Leader Board database.
6. The students can view the Leader Board by selecting the “View Leader Board” option.

**Alternates: Nil**

**Exception:**

1. If the staff is unable to modify the Leader Board, they can reach out to the admin and directly access the backend of the system to update the Leader Board database.

**Level:**

* Leader Board Modification – Blue level
* View Leader Board – Blue Level
* Sports House Management – White Level

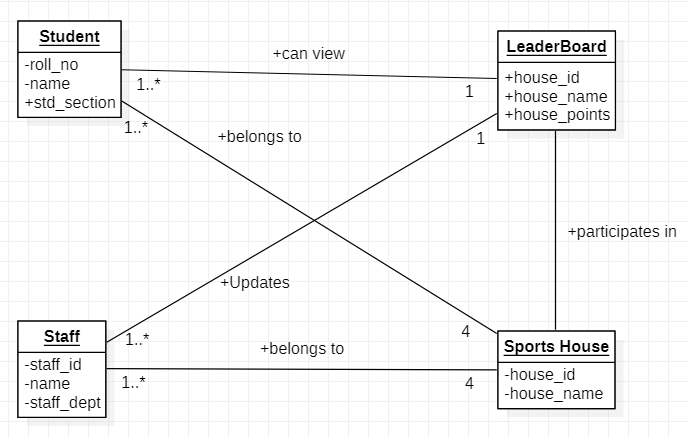
**Postcondition:**

* The Leader Board is modified and can be viewed by the students.

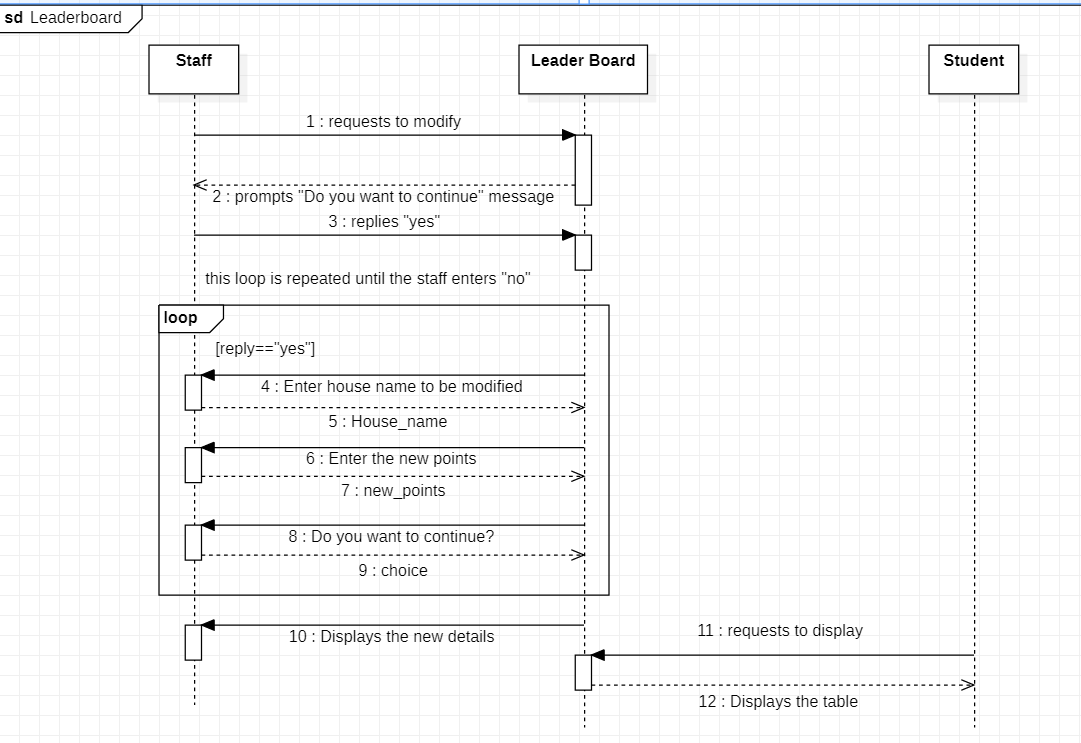
**Stakeholders:**

* Sports house

**Object Diagram:**



**Time-sequence Diagram:**



**View Student, Staff and Hostel Details:**

**Description:**

This module enables the admin to access the details of students, staffs and hosteller.

**Actors:**

**Primary:**

* Admin

**Secondary:**

* Student
* Staff
* Hosteller

**Trigger:**

This module is triggered when the admin selects the option “View student/staff/hosteller” details.

**Precondition:**

* The admin has a stable internet connection.

**Basic Flow: (Sunny Day Scenario)**

1. The admin selects the option “View student/staff/hosteller details”.
2. The admin is asked to enter the roll no or teacher id and the corresponding data of the user will be displayed to the admin.

**Alternates: Nil**

**Exceptions:**

1. If the module is not working due to technical glitches, the admin can access the details directly from the database.

**Level:**

* Viewing student/staff/hosteller details – Blue Level

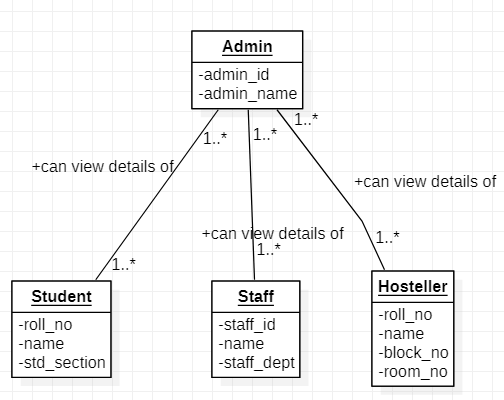
**Postcondition:**

* The requested details are displayed to the Admin.

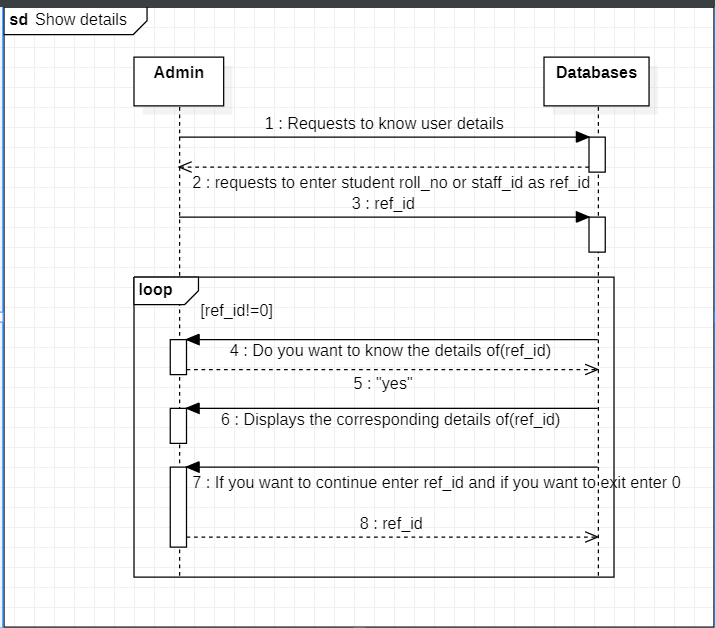
**Stakeholders:**

* Admin

**Object Diagram:**



**Time-Sequence Diagram:**

****

**Allocate and View marks and attendance:**

**Description:**

Staffs can allocate marks and Attendance for each student and the students can view their own marks and attendance.

**Actors:**

**Primary:**

* Student
* Staff

**Secondary:**

* Attendance registry database
* Marks registry database

**Trigger:**

The use case is triggered when the staff select “Allocate marks/Attendance” option.

**Precondition:**

* The user(staff/student) should have a stable internet connection
* All the exams/classes corresponding to the mark/Attendance publication need to be completed before the result publication.

**Basic Flow: (Sunny Day Scenario)**

* The staff selects the “Allocate Marks/Attendance” option.
* The staff is asked to enter the course name, student roll no and the marks he/she secured in that exam.
* The staff is also asked to enter the attendance percentage of that particular student.
* The values entered by the staff are stored in the marks and attendance registry databases respectively.
* The student selects the “View marks/Attendance” option.
* The respective marks and the Attendance percentage is displayed to the student.

**Alternative: Nil**

**Exception:**

* If the staff is unable to enter the marks/Attendance, they can reach out to the admin and directly access the backend of the system to enter the marks/Attendance.

**Level:**

* Marks/Attendance management – White level
* Enter Marks/Attendance - Blue Level
* View Marks/Attendance - Blue Level

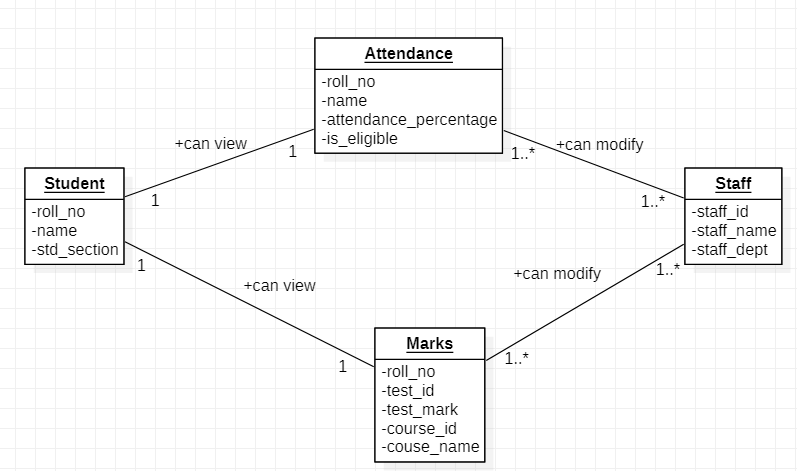
**Postcondition:**

* The marks and attendance are updated in the database and can be viewed by the students.

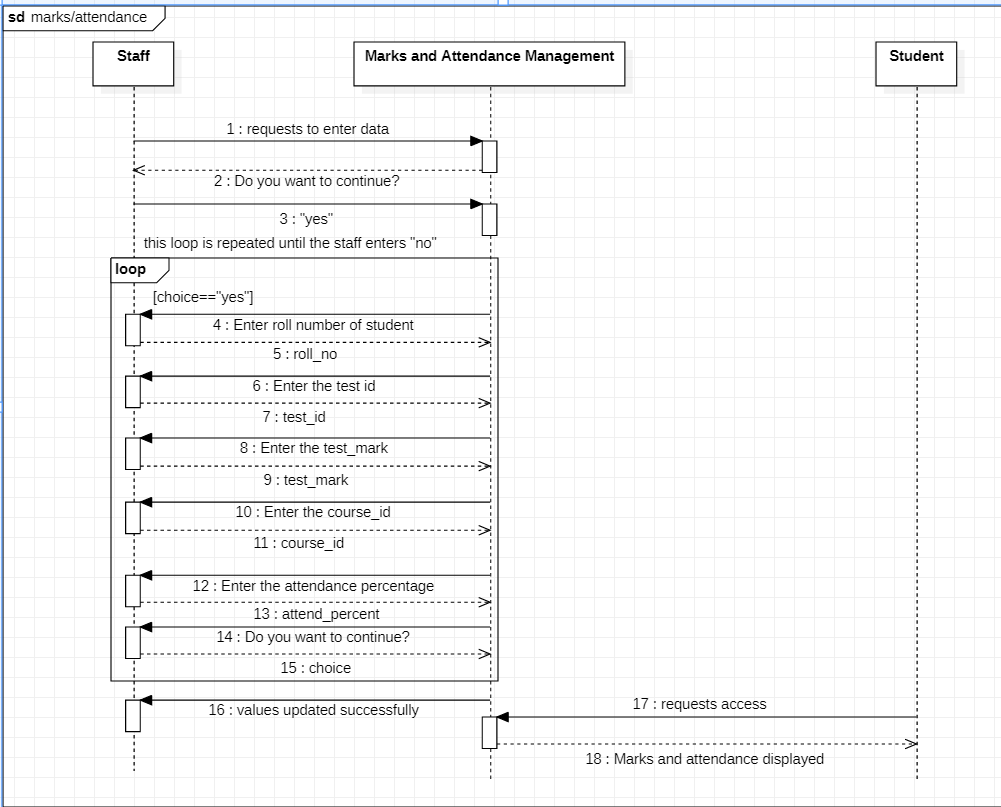
**Stakeholders:**

* Student
* Staff

**Object Diagram:**



**Time-sequence Diagram:**



**Add and remove course**

**Description:**

A student will be able to add/remove when he passes on to the next semester.

**Actors**:  
**Primary:**

* Student

**Secondary:**

* Course Registration Database

**Trigger:**  
When the student clears current semester and goes to the next semester

**Preconditions:**

* Student should unenroll from previous semester courses
* The user has active internet connection

**Basic Flow: (Sunny Day Scenario)**

1. Student login to their account and navigates to course details.
2. Student removes previous semester courses.
3. Student enrols in the courses allotted for the next semester.
4. The registered courses are displayed.

**Alternates:**

Students registers for the course manually at the administration office.

**Exceptions: Nil**

**Level:**

* Unenrolls course - blue level
* Enrolls course – Blue level

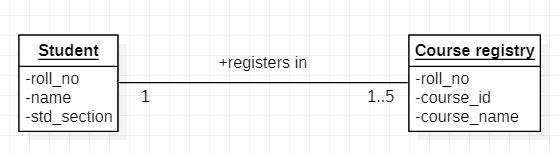
**Post condition:**

* The student is enrolled in the selected courses.

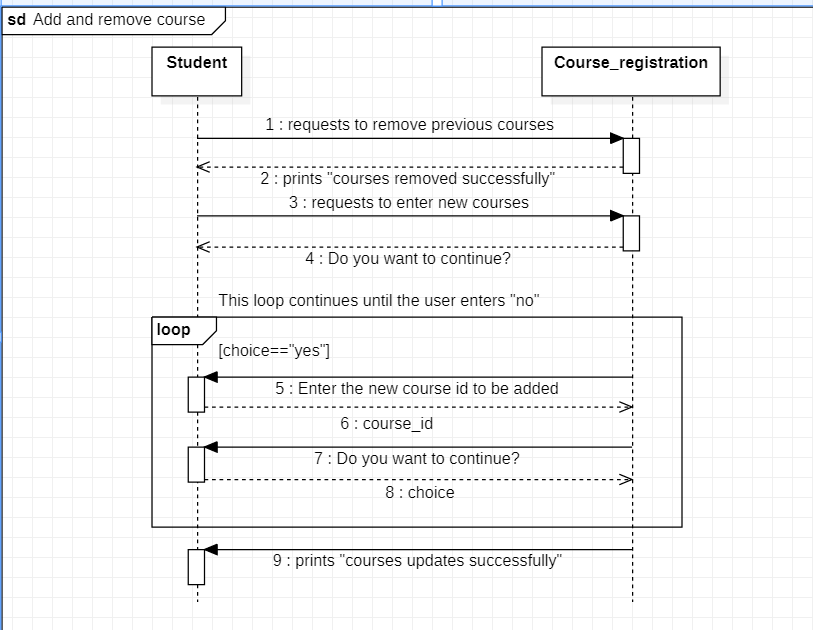
**Stakeholders:**

* Student

**Object diagram:**

****

**Time-Sequence Diagram:**



**Borrow and return book**

**Description:**

Staff and students add details of borrowed library book in this module.

**Actors:**

**Primary:**

* Staff
* Students

**Secondary:**

* Library Database

**Trigger:**

* This module is triggered when a staff or students borrows a book.

**Preconditions:**

* The user has active internet connection
* The user must not borrow two books at a time

**Basic Flow:**

* The user first adds their details and then details of the book
* Then the user enters the date and time of borrow
* While returning the book user will add the date and time of return using their pre-existing borrowed details

**Alternates: Nil**

**Exceptions: Nil**

**Level:**

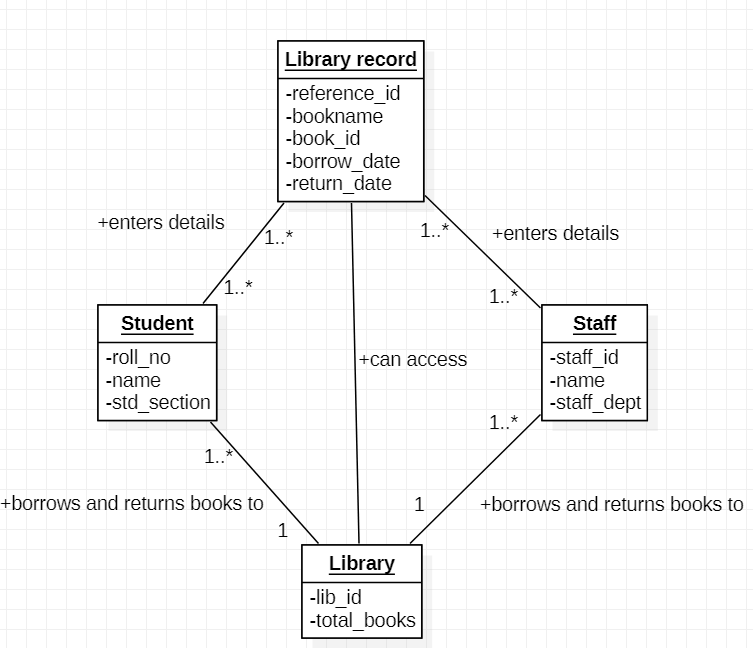
* User enter borrowed book details – Blue level
* User enter return time details - blue level

**Post condition:**1. User borrow details added successfully

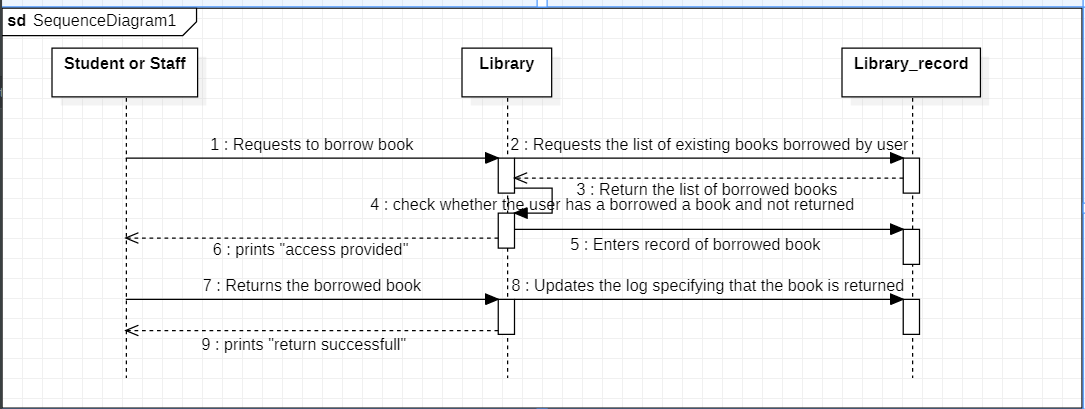
**Stakeholders:**

* Library
* Student
* Staff

**Object Diagram:**

****

**Time-sequence Diagram:**



**Check in and check out**

**Description:**

Admin enter check-in and check-out date and time of a student entering and leaving hostel.

**Actors:**  
**Primary:**

* Hosteller
* Admin

**Secondary:**

* Hostel management Database

**Trigger:**

When the student enters or leaves the hostel

**Preconditions:**

1. The user has active internet connection
2. Student must have been allocated a hostel room earlier

**Basic Flow:**

1. While checking in admin will enter Student Id, hostel details and check-in date and time
2. While checking out admin enters check-out date and time using check-in details

**Alternates:**

Manually enter hostel check-in and check-out details

**Exceptions: Nil**

**Level:**

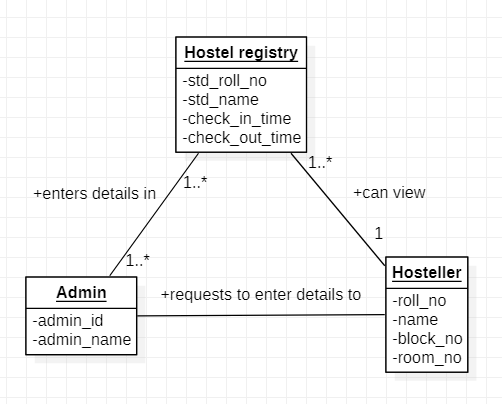
* Enter check-in and check-out details – Blue level

**Post condition:**  
Student’s check-in and check-out details are updated in the registry successfully.

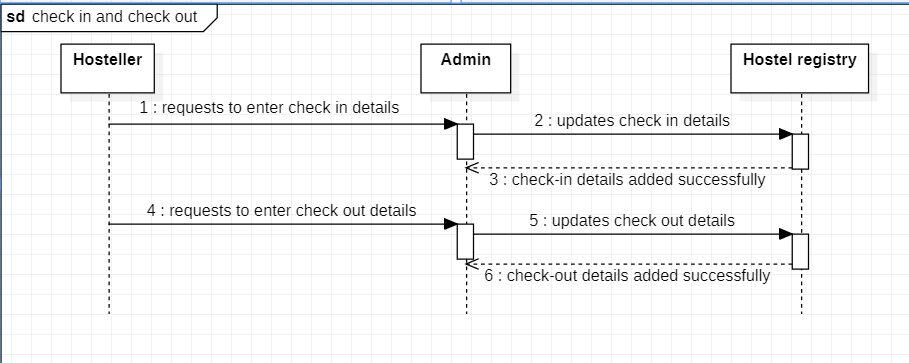
**Stakeholders:**

* Student
* Admin
* Hostel

**Object Diagram:**



**Time-sequence Diagram:**

****

**Conceptual class list:**

* **Login**
* **Username** is a string that denotes the user’s username.
* **Password** is a string that denotes the user’s password
* **Type\_of\_user** is a string that denotes the user’s designation.
* **Student**
* **Roll\_no** is a string that denotes the student’s unique roll no.
* **Name** is a string that denotes the student’s name.
* **Std\_section** is a string denotes the student’s classroom section.
* **Std\_campus** is a static string that denotes the student’s college campus
* **No\_of\_cert\_courses** is a integer that denotes the number of certified courses done by the UG student
* **No\_of\_research\_paper** is an integer that denotes the number of research papers published by a PG student
* **Staff**
* **Staff\_id** is a string that denotes the staff’s unique id.
* **Name** is a string that denotes the staff’s name.
* **Staff\_dept** is a string that denotes the staff’s department.
* **Staff\_pay** is a float that denotes the staff’s total pay per annum.
* **Staff\_incentive** is a static float that denotes the incentive amount provided to all the teachers.
* **Subscription\_allotment** is a float that denotes the educative subscription allotment provided to teaching staffs.
* **Overtime\_pay** is a float that denotes the extra pay provided to support staff for working overtime.
* **Admin**
* **Admin\_id** is a string that denotes the admin’s unique id.
* **Admin\_name** is a string that denotes the admin’s name.
* **Course\_registration**
* **Roll\_no** is a string that denotes the student’s unique roll no.
* **Course\_id** is a string that denotes a course’s unique id
* **Course\_name** is a string that denotes the title of the course.
* **Sports\_House**
* **House\_id** is a string that denotes the house’s unique id
* **House\_name** is a string that denotes the house’s name
* **House\_points** is an integer that denotes the house’s current points
* **Max\_points** is a static and integer that denotes the total possible points available for a house.
* **Library**
* **User\_ref\_id** is a string that denotes the user’s unique id
* **User\_name** is a string that denotes the user’s name
* **Premium\_fees** are a static and float that denotes the premium to be paid to library for subscription.
* **Hosteller:**
* **Roll\_no:** is a string that denotes the hosteller’s unique roll no.
* **Block\_no** is a string that denotes the hosteller’s block number.
* **Room\_no** is a string that denotes the hosteller’s room number.
* **Hostel\_fees** is a static and float that denotes the hosteller’s fees.
* **Boys\_hostel\_rules** is a string that denotes the rules in boys’ hostel.
* **Girls\_hostel\_rules** is a string that denotes the rules in girls’ hostel.
* **Address:**
* **Street** is a string that denotes the residential street name of the user.
* **City** is a string that denotes the residential city of the user.
* **Pin code** is an integer that denotes the residential pin code of the user.

**Identified classes from conceptual class**

**1. Abstract Class list:** Student, Staff

**2. Interface List:** Hosteller

**3. Inheritance relationship:** Student-UG student, Student-PG student, Staff-teaching staff, Staff-support staff, Hosteller-Boy Hosteller, Hosteller-Girl Hosteller

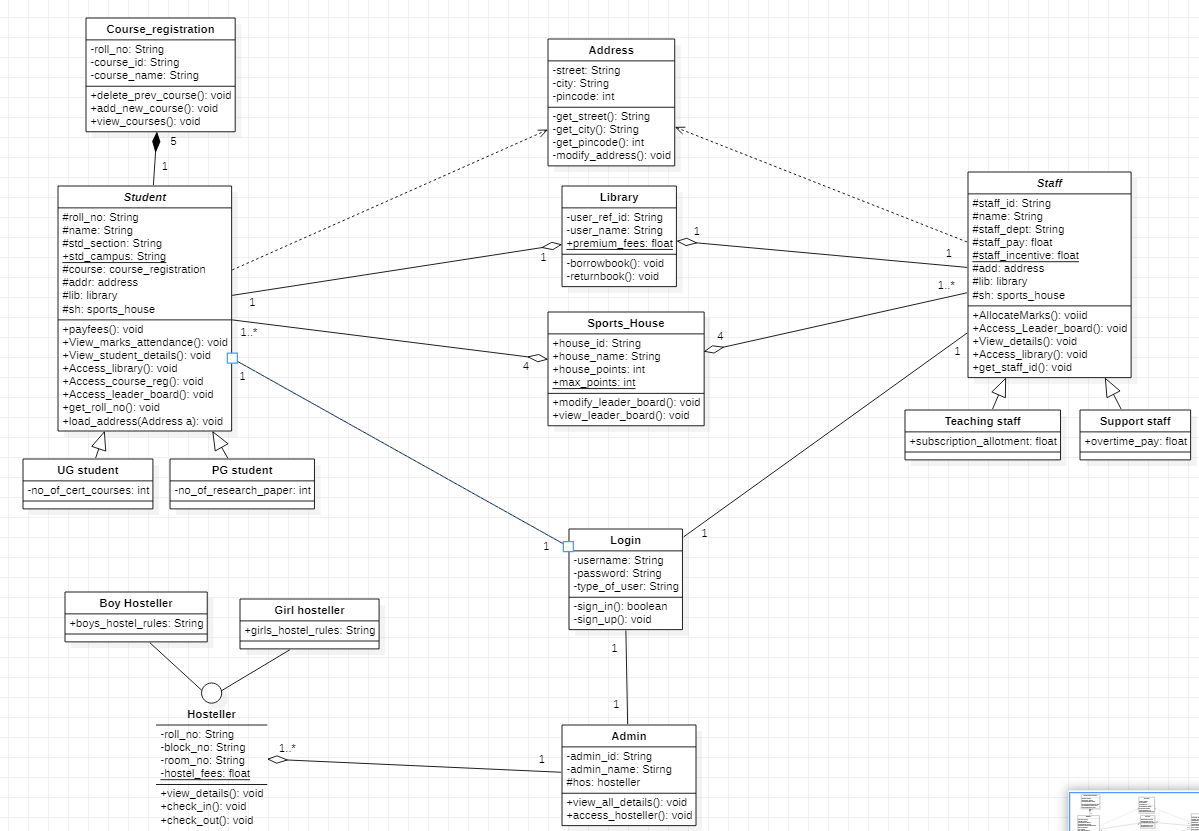
**4. Static variables:** std\_campus, staff\_incentive, premium\_fees, max\_points

**5. Aggregation:** Student-Library, Student-Sports\_house, Staff-Library, Staff-Sports\_house, Admin-Hosteller

**6. Composition:** Student-Course\_registration

**7. Dependency:** Student-Address, Staff-Address

**Class Diagram:**



**Task Done by Each member:**

|  |  |
| --- | --- |
| **Name** | **Task** |
| Pradeep Karthik M [CB.EN.U4CSE20447] | Pay fees, Modify and view leader board |
| Praveen Kumar M [CB.EN.U4CSE20449] | Allocate and view marks/attendance, Add and remove course |
| S Adithya [CB.EN.U4CSE20403] | Login and signup and Allocate house |
| S R Logeswaran [CB.EN.U4CSE20435] | Borrow and return book, check-in and check out |
| Combined | Problem statement, actors and tasks list, Identifying conceptual classes and class diagram, View student,staff,hosteller details (use case). |