Addis Ababa University

Department of Computer Science

Postgraduate program

Software Architecture and design

SDD

Of

Student Activity Manager (SAM)

Submitted by: Awoke Bagodaw

Submitted to: Ayalew Belay (PhD)

Submission date: 06-06-2021

Table of Contents page no_

1. Introduction

As System Design Document (SDD) is one of the main part in software development life cycle process and as we will use it in implementation, project monitoring, verification and validation, we preparing it as an independent but basic component of our work. We are going to prepare it by referring Requirement Analysis Document (RAD) which includes detail formats and other information of the current working environment of the existing system. The SDD is a foundation for the system functional components, code development, system testing and deployment. The description of these functions will assist with the implementation strategy to ensure that the system meets the defined requirements.

This document will describe the design goals expected from the system, subsystem decomposition, hardware/software mapping, persistent data management, access control and in the last section the object model will described using class interface.

1.1.Purpose of the Document

The purpose of this system design document is to describe the solution domain of the problem domain stated on the RAD of Student Activity Manager (SAM). System Design Document is to provide a description for how the new SAM will be constructed. The SDD document tracks the necessary information required to effectively define architecture and system design of the proposed system in order to give the development team guidance on the architecture of the system to be developed.

1.2. Design Goals

Design goals identify the qualities that SAM will focus on as they are derived from the nonfunctional requirements in the application domain. The qualities of the design goals are selected from the following design criteria's which are grouped in to: Dependability, Performance, maintenance and end user.

Dependability Criteria

I. **Security:** the system should not allow non-authorized users using a form based authentication.

II. **Reliability:** in order to maintain the difference between specified and observed system behavior we will try to test it as much as possible.

III. **Availability:** the system should be available for any legitimate users as long as the service provider is available or it is not shut down by the system administrator.

End User Criteria

I. **Usability:** The system should be user friendly, and easy to learn and use. The user interface form should be clearly designed.

Performance Criteria

I. **Responsive time:** The system should provide as fast response as possible. In order to minimize the time it takes to provide response, powerful application server is used.

II. **Throughput:** The system should be able to support a number of users at a time using the available bandwidth of the system.

Maintenance Criteria

I. **Modifiability:** the system should be easily extensible to the need of the SMS and to add new functionalities to the system.

1.3. Design Trade-off

Reliability Vs. Performance

In order to make the system reliable, we need to lock data on concurrent user. As this will reduce the overall performance of the system.

Security Vs usability

To make the system more secure, we will use time session which lock when the user is idle. This will ask the user to enter his credential frequently.

2. Proposed System Design Model

2.1. Subsystem Decomposition with Services

A system is decomposed in to subsystems in order to decrease complexity of a system so that it will be manageable for implementation. SAM is decomposed in to sub systems as shown in figure 1.

As a result:

- Highly related classes/objects or objects identified in one use case were assigned in to the same subsystem

- Objects used for moving data among sub systems will be placed in a dedicated subsystem
- The system will have minimal number of associations crossing subsystems boundaries
- All objects in the same subsystem should be functionally related

These criteria in general bring high cohesion and low coupling which enhances performance of the system. The sub systems are:

- Schedule Management
- User Management
- Event information management
- Internship Job management
- Report Management

For the library managemet componenet which is included in the RAD of the proposed system, we believe that it is better to integrate it with the unversity's lirary management system else the libray management system is broad by it selef.

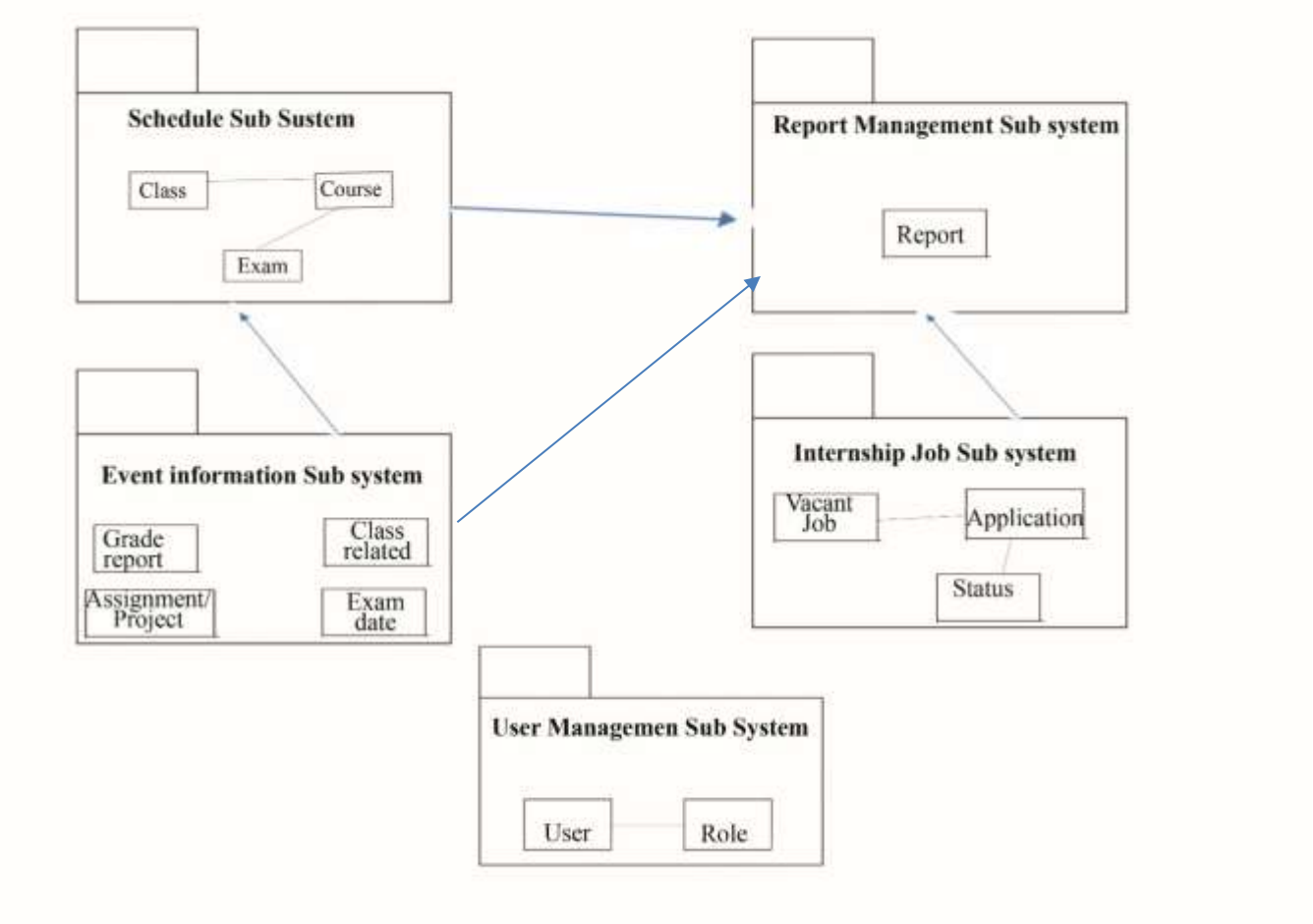

Figure 1. Subsystem Decomposition of SAM

Table 1. Subsystem Description table

Sub System	Description
Schedule subsystem	This subsystem enables to manages information about detail schedule information. It provides methods to add, update and search information for the mentioned activities and services.
Event information management subsystem	This subsystem enables manage the information to be transferd to students/instructors, enable post an event, update, view, search including online chat communication

Internship Job management sub system	Internship Job management sub system is a module to mange internship job related activities like post/add job, update, delete, view and search for an internship job including applying for it and follow its progress.
User Management	The subsystem manages information about user registration, role registration, assign role and give permission. The subsystem provides methods to add, update and search information for the mentioned activities and services
Report Management sub system	This subsystem generates different reports from schedule, event information and internship job management subsystems

2.2. Hardware/Software Mapping

This section shows architecture of the system. The system contains three hardware components the client machine, the application server machine and the database server machine. The client machine contains the web browser that enables system users to access the application server according to their privilege through HTTPs. The application server or the web server contains the application logic and uses Glass Fish Server to handle requests send from users through HTTPs. And finally there is the database server that is used to perform tasks such as data analysis, storage, data manipulation, archiving, and other non-user specific tasks. The database server uses the relational database management system which will be Oracle Database server. The data access from the database server using the application server is performed by using Java. The selected architecture increases maintainability and dependability of the system as shown in figure 2.

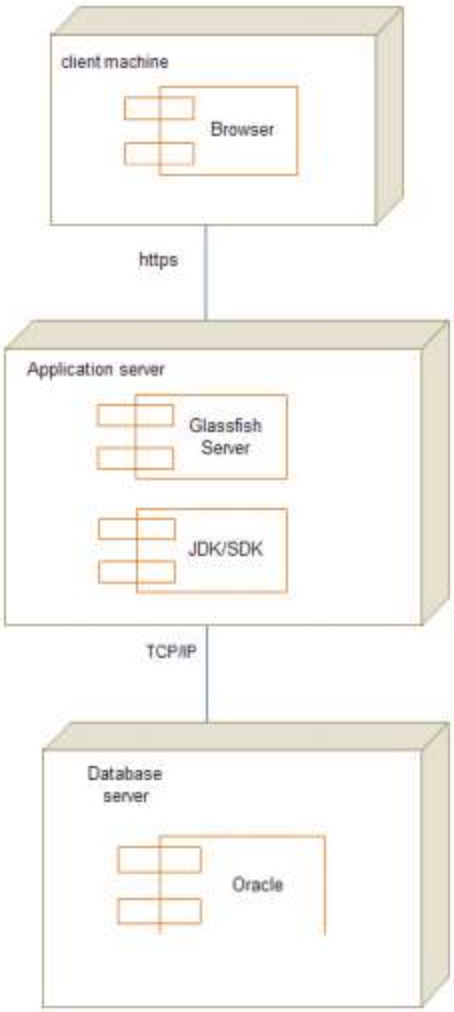

Figure. 2.

2.3.Persistent Data Management

As persistent data management deals with how the persistent data are stored and managed, information related to student activitiy, and other related information which are expected as persistent data are stored in a database management system. In order to store data persistently in a database, those entity classes illustrated in class diagram in the analysis model should have to be mapped into tables and the attributes into fields to the respective tables as shown in Figure 3.

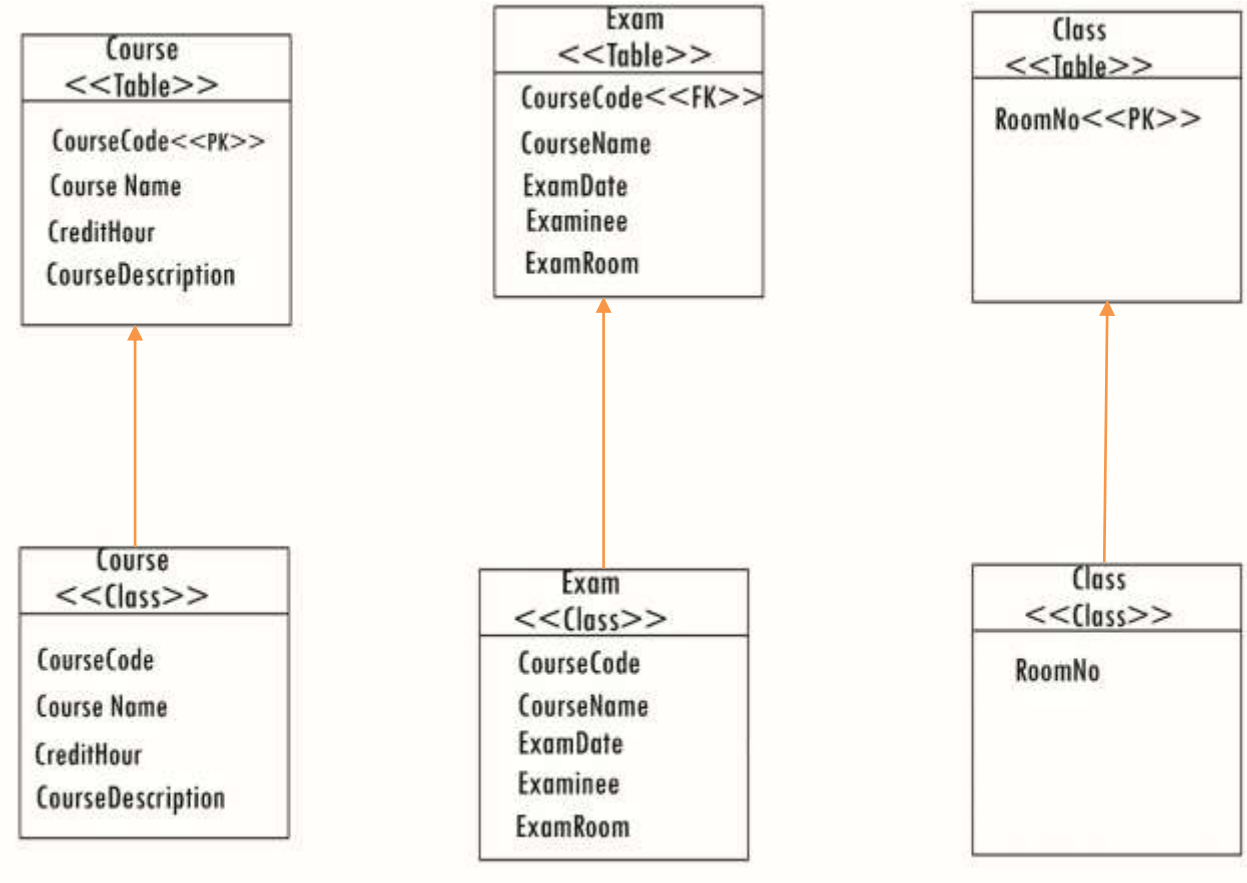
Course
<<Table>>
CourseCode<<PK>>
Course Name
CreditHour
CourseDescription
Exam
<<Table>>
CourseCode<<FK>>
CourseName
ExamDate
Examinee
ExamRoom
Class
<<Table>>
RoomNo<<PK>>
Course
<<Class>>
CourseCode
Course Name
CreditHour
CourseDescription
Exam
<<Class>>
CourseCode
CourseName
ExamDate
Examinee
ExamRoom
Class
<<Class>>
RoomNo

Figure 3.

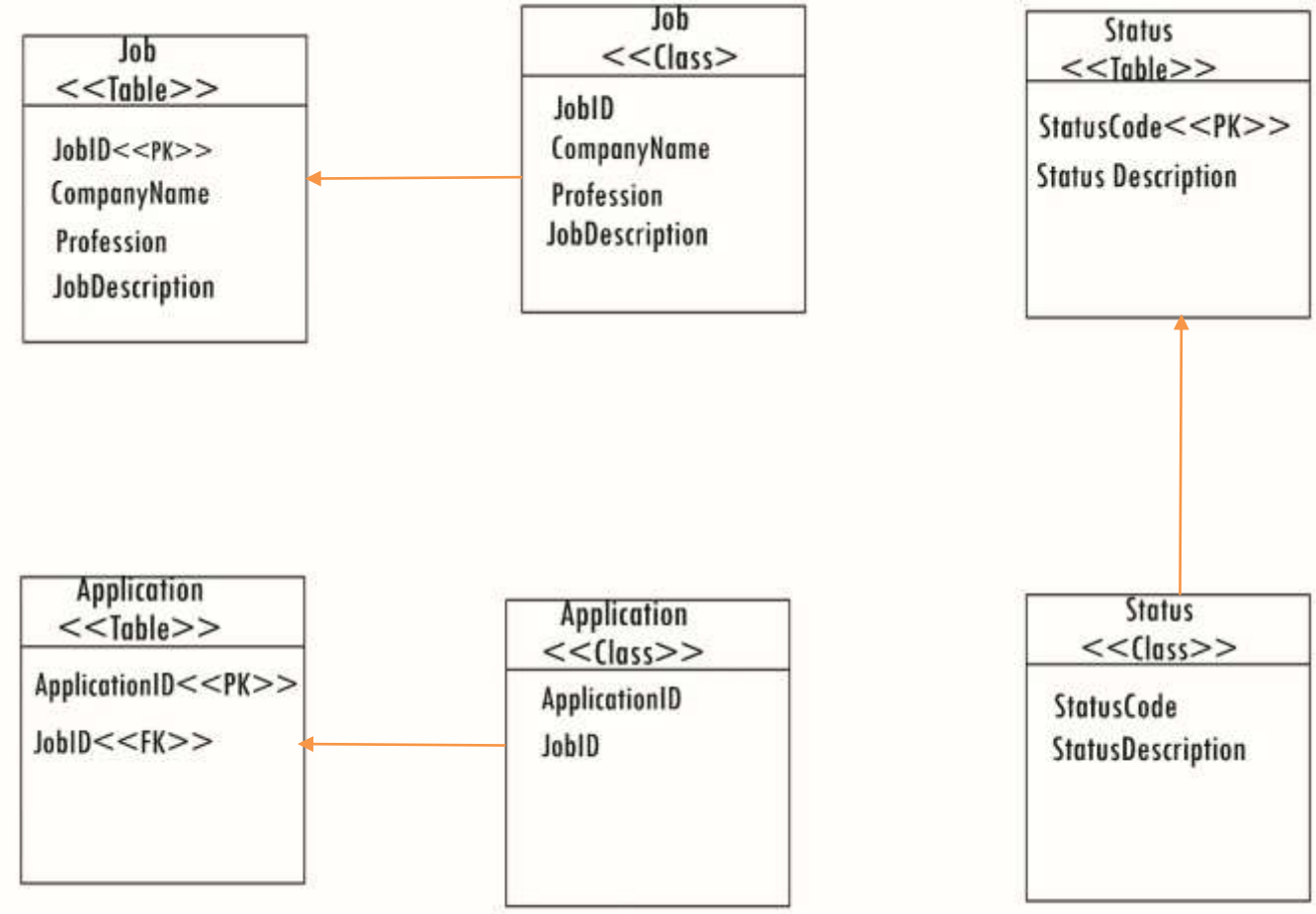
Job
<<Table>>
JobID<<PK>>
CompanyName
Profession
JobDescription
Job
<<Class>
JobID
CompanyName
Profession
JobDescription
Status
<<Table>>
StatusCode<<PK>>
Status Description
Application
<<Table>>
ApplicationID<<PK>>
JobID<<FK>>
Application
<<Class>>
ApplicationID
JobID
Status
<<Class>>
StatusCode
StatusDescription

Figure 4

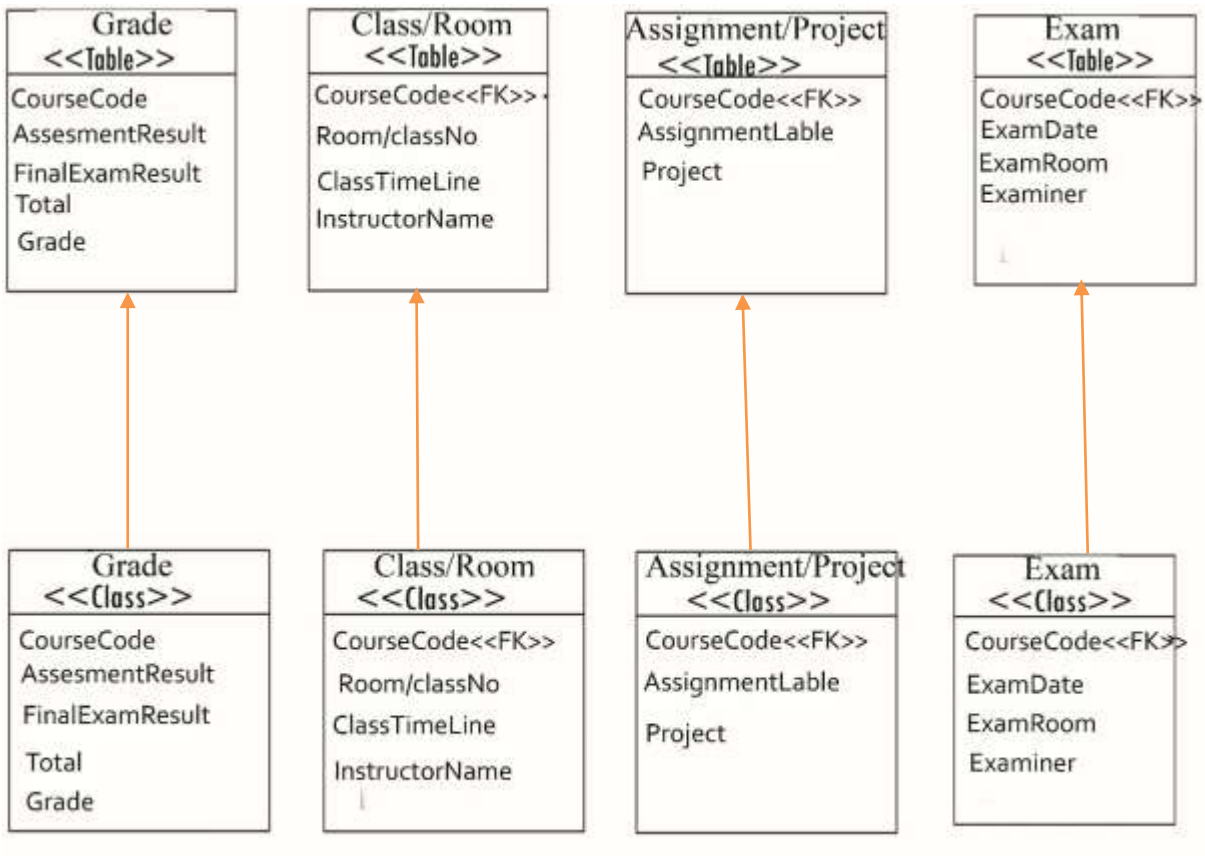
Grade
<<Table>>
CourseCode
AssesmentResult
FinalExamResult
Total
Grade
Class/Room
<<Table>>
CourseCode<<FK>>
Room/classNo
ClassTimeLine
InstructorName
Assignment/Project
<<Table>>
CourseCode<<FK>>
AssignmentLable
Project
Exam
<<Table>>
CourseCode<<FK>>
ExamDate
ExamRoom
Examiner
Grade
<<Class>>
CourseCode
AssesmentResult
FinalExamResult
Total
Grade
Class/Room
<<Class>>
CourseCode<<FK>>
Room/classNo
ClassTimeLine
InstructorName
Assignment/Project
<<Class>>
CourseCode<<FK>>
AssignmentLable
Project
Exam
<<Class>>
CourseCode<<FK>>
ExamDate
ExamRoom
Examiner

Figure 5

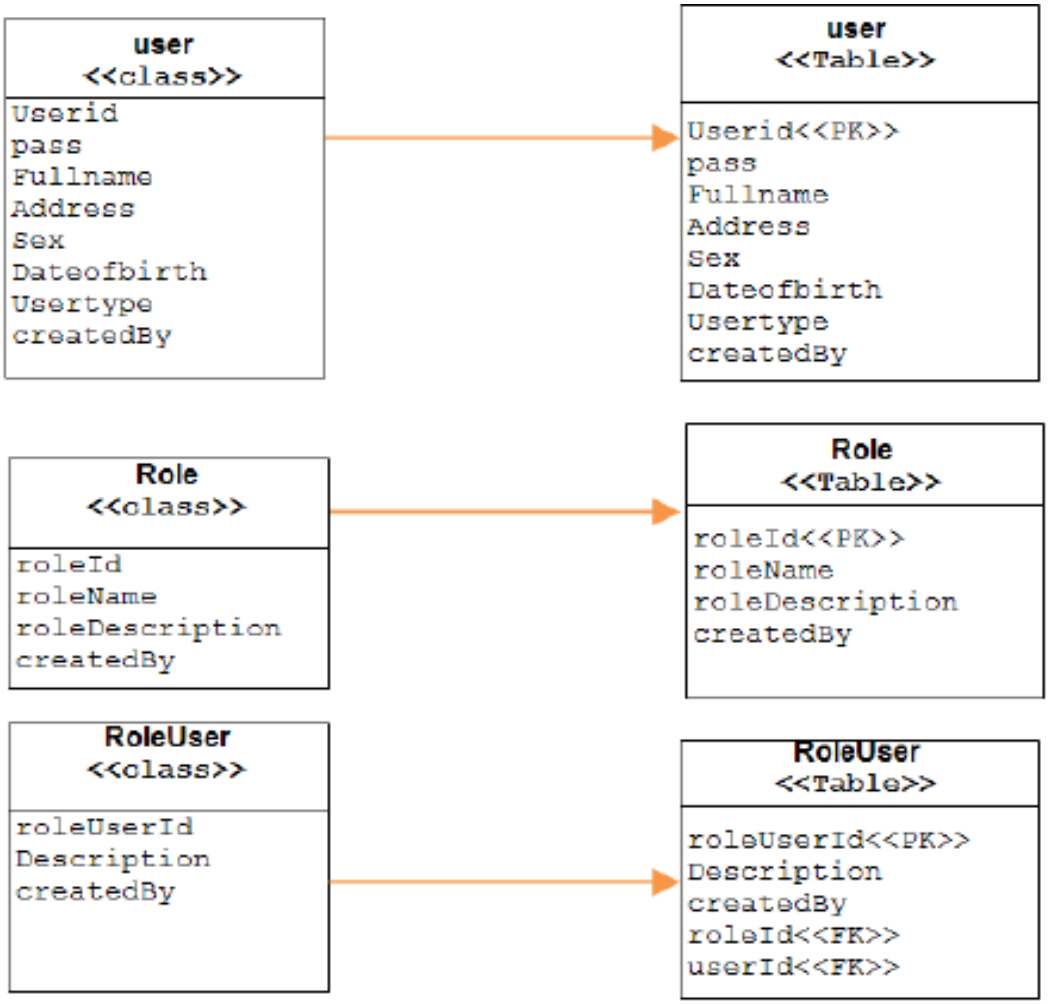

Figure 6

2.4.Access Control & Security

Access control is a security technique that can be used to regulate who or what can view or use resources in SAM.

Table 2. Access Control Matrix for SAM

Subsystem	**Class**	**Opration**	**Actoras**		
			Associated Dean	**Instructor**	**System Admin**
	Course	AddCoursesOfSemister()	✓		
		AssignsClassesForCourses()	✓		
		AssignTimlineForCourses()	✓		

Schedule subsystem		DefineCourseNumberForCourse()	✓		
		AssignLectureForCourse()	✓		
		Update()	✓		
	Exam	ScheduleExamForCourse()	✓		
		PrepareExamForACourse()		✓	
		AssignExaminerForFinalExam()	✓		
		ScheduleExamTimeForCourse()	✓		
		AssignExamRoomForCourse()	✓		
	Class	AssignLearnnigRoomForCourse()	✓		
Event information subsystem	Grade report	AssesStudentsForACourse()		✓	
		PostStudentGrade()	✓		
		UpdateGrade()		✓	
		SearchGrade	✓	✓	✓
	Class related event	PostClassTimeChange()	✓	✓	
	Assignment/project	PostProjectSubbmitionDate()		✓	
	Exam date	PostExamDateAndTime()	✓	✓	
		PostInternshipJob()	✓	✓	
Internship job subsystem	Vacant job	PostVacantjob()	✓	✓	
	Application	ApplyForAjob()			✓
	Status	Viewstatus()			✓
Report Subsystem	Report	GenerateScheduleReport	✓		
		GenerateInternshipJobReport	✓	✓	
		GenerateEventInformationReport	✓	✓	
User management subsystem	User	AddUser()	✓		
		DeleleteUser()	✓		
		SearchUser()	✓		
		EditUser()	✓		
	Role	AddRole()	✓		
		AssignRole	✓		
		EditRole()	✓		

		DeleteRole()	✓		

3. Object Design

3.1. Class interface

Table 3 course class interface

Class	Attribute	Invariant
Course	(-)CourseCode:string	Should be unique
	(-)CourseName:string	
	(-)CreditHour:number	Should not be null
	(-)CourseDescription:string	
	Methods	
	(+)AddCoursesOfSemister(Course:course): void	**Precondition:-** course information should be valid and yet not be registered before.
	(+)AssignsClassesForCourses(CourseCode:string):course	**Precondition:-** The class/ room should not be assigned for other course at the specified time. **Post condition:-** a course have assigned defined room/class
	(+)AssignTimlineForCourses(CourseCode:String):course	**Precondition:-**The schedule time should not be clash with the class/ room and the instructor who is assigned for the course. **Post condition:-** a course have assigned defined timeline
	(+)AssignLectureForCourses(CourseCode:String):course	**Precondition:-** The course should not yet assigned instructor and the instructors timeline should not be clashed with other course. **Post condition:-** a course have assigned lecture
	(+)Update(Course:course): void	**Precondition:-** The course information should exist **Post condition:-** Course information updated

Table 4 Exam class interface

Class	Attribute	Invariant
Exam	(-)CourseCode:string	Should be unique
	(-)CourseName:string	
	(-)ExamDate:date	Should not be null
	(-)Examinee:string	Should not be null
	(-)Room:number	Should not be null
	Methods	
	(+)AssignExamineeForFinalExam (Course:CourseCode): course	**Precondition:-** course should be valid and examinee should not yet assigned
	(+)ScheduleExamTimeForCourse(CourseCode:string):course	**Precondition:-** The course should be valid and the time should not be clashed with other exam time **Post condition:-** Exam time for a course is assigned
	(+)AssignExamRoomForCourse(CourseCode:String):course	**Precondition:-**The exam should not be assigned for other course at the same time **Post condition:-** a course have assigned defined exam room/class
	(+)Update(Exam:Exam): void	**Precondition:-** The exam information should exist and valid **Post condition:-** Exam information updated

Table 5 Class/room class interface

Class	Attribute	Invariant
Class/ Room	(-)RoomNo:number	Should be unique
	Methods	
	(+)AssignLearnnigRoomForCourse(class/room:class/room): class/room	**Precondition:-** class/room should exist and not yet assigned for other course at the same time **Post condition:-** Learning class/ room assigned
	(+)Update(class/room:class/room): void	**Precondition:-** The class/room information should valid **Post condition:-** class/room information updated

Table 6 Grade class interface

Class	Attribute	Invariant
Grade	(-)CourseCode:string	Should be unique
	(-)AsseementResult:number	Should not be null
	(-)FinalExamResult:number	Should not be null
	(-)Total:number	Should not be null
	(-)Grade:number	Should not be null
	Methods	
	(+)AddStudentsAssesment (grade:grade): void	**Precondition:-** Assessment result should exist and not yet added **Post condition:-** Assessment result added
	(+)PostStudentGrade(grade:CourseCode):grade	**Precondition:-** Assessment and final exam result should exist **Post condition:-** Grade posted
	(+)UpdateGrade (Student:StudentID): grade	**Precondition:-** The class/room information should valid **Post condition:-** class/room information updated
	(+)SearchGrade(Student:StudentID):grade	**Precondition:-** Grade should exist **Post condition:-** Grade retrieved

Table 7 event class interface

Class	Attribute	Invariant
Event	(-)CourseCode:string	Should be unique
	(-)Room/ClassNo:number	
	(-)ClassTime:date	Should not be null
	(-)InstructorName:string	Should not be null
	Methods	
	(+)PostClassTimeChange (event:event): void	**Precondition:-** Class timeline should be pre-sated **Post condition:-** class time change event information posted
	(+)PostProjectSubbmitionDate (event:CourseCode):event	**Precondition:-** project must be given previously **Post condition:-** Project/assignment submission date posted
	(+)PostExamDateAndTime (event:CourseCode): event	**Precondition:- T**he course Exam should exist **Post condition:-** Exam date posted
	(+)PostInternshipJob (Event:JobId):event:job	**Precondition:-** Internship job should exist **Post condition:-** Internship job posted

Table 8 job class interface

Class	Attribute	Invariant
Job	(-)JobID:string	Should be unique
	(-)CompanyName:string	Should not be null
	(-)Profession:string	Should not be null
	(-)JobDescription:string	
	Methods	
	(+)AddVacantjob(job:job): void	**Precondition:-** Vacant internship job should exist **Post condition:-** Vacant internship job added
	(+)ApplyForAjob (job:JobID):job	**Precondition:-** Vacant internship job should be added **Post condition:-** applied for internship job
	(+)PostExamDateAndTime (event:CourseCode): event	**Precondition:- T**he course Exam should exist **Post condition:-** Exam date posted
	(+)PostInternshipJob (Event:JobId):event:job	**Precondition:-** Internship job should exist **Post condition:-** Internship job posted

Table 9

Class	Attribute	Invariant
Application	(-)ApplicationID:number	Should be unique
	(-)JobID:string	Should not be null
	Methods	
	(+)ApplyForAjob (job:JobID):job	**Precondition:-** Vacant internship job should be added **Post condition:-** applied for internship job

Table 10

Class	Attribute	Invariant
Status	(-)JobID:string	Should be unique
	(-)StatusCode:string	Should not be null
	(-)StatusDescription:string	
	Methods	

	(+)Viewstatus(Status: StatusCode):status	**Precondition:-** prior internship job application should be exist **Post condition:-** application status reviewed

Table 11 user class interface

Class Name	Attributes	Invariant
User	(-)userId:number	Should be unique
	(-)userName:string	Should not be null
	(-)Password:string	Should be Alphanumeric and should not be null
	(-)address:string	
	(-)sex:string	Should not be null and specified by either "M" or "F"
	(-)dateOfbirth:date	Should be the valid date format
	(-)userType:string	
	Methods	
	(+)addUser(User:User):void	➢ **Precondition** :user information should be valid and there should not be exist. ➢ **Postcondition** :user information added.
	(+)searchUser(userId:numb er):User	➢ **Precondition** : user information should exist ➢ **Post condition** : userinformation retrieved
	+deleteUser(userId:number):void	➢ **Precondition** : user information should exist ➢ **Post condition** : userinformation deleted
	editUser(User:User):void	➢ **Precondition** : user information should exist ➢ **Post condition** : userinformation edited

Table 16 user Role class interface

Class Name	Attributes	Invariant
Role	(-)roleId:number	Should be unique
	(-)roleName:string	Should not be null
	Methods	
	(+)addUserRole (UserRole :UserRole):void	➢ **Precondition** :user role information should be valid and there should not be exist. ➢ **Postcondition** :user role information added.
	(+)search**UserRole** (roleId:number):**UserRole**	➢ **Precondition** : user role information should exist ➢ **Post condition** : user role information retrieved
	+delete**UserRole** (roleId:number):void	➢ **Precondition** : user role information should exist ➢ **Post condition** : user roleinformation deleted
	(+)editUserRole(**UserRole**: **UserRole**):void	➢ **Precondition** : user role information should exist ➢ **Post condition** : user role information edited
	(+)assignUserRole(**UserRole**:**UserRole**):void	➢ **Precondition** : user role information should exist ➢ **Post condition** : user role information assigned to the user