SOFTWARE DESIGN AND ANALYSIS ON PGDIT AUTOMATION SYSTEM

SOFTWARE DESIGN AND ANALYSIS ON PGDIT AUTOMATION SYSTEM

Submitted to:

Nadia Nahar

Lecturer

Institute of Information Technology

University of Dhaka

Submitted by:

Jarifa Khatun (0710)

Asadullah Hil Galib (0712)

Satyaki Das (0720)

Moumita Asad (0731)

Shafi ul Alam (0735)

3rd Year, 6th Semester, 7th Batch

Session: 2014-2015

Institute of Information Technology

University of Dhaka

Submission Date: 19th November, 2017

Letter of Transmittal

19th November, 2017

Nadia Nahar

Lecturer

Institute of Information Technology

University of Dhaka

Subject: Submission of design document on “PGDIT Automation System”.

Madam,

We, the team, on which the project on “PGDIT Automation System” was assigned, are submitting our report with due respect. We have tried our best for the report. However, it might lack perfection. So, we therefore, hope that you would be kind enough to accept our report and oblige thereby.

Yours sincerely

Jarifa Khatun (0710)

Asadullah Hil Galib (0712)

Satyaki Das (0720)

Moumita Asad (0731)

Shafi ul Alam (0735)

3rd Year, 6th Semester, 7th Batch

Institute of Information Technology

University of Dhaka

Session: 2014-15

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

Software design encompasses the set of principles, concepts, and practices that lead to the development of a high-quality product. According to Mitch Kapor, the creator of Lotus 1-2-3,"Design is where you stand with a foot in two worlds—the world of technology and the world of people and human purposes—and you try to bring the two together." Design allows to model the system is to be built in such a way that the model can be assessed for quality and improved before code is generated, tests are conducted, and end users become involved in large numbers. Design is the place where software quality is established.

The design model provides detail about software architecture, data structures, interfaces, and components that are necessary to implement the system. Figure 1 shows the elements of design model. This document contains the architectural design, component-level design and interface design of PGDIT Automation System.

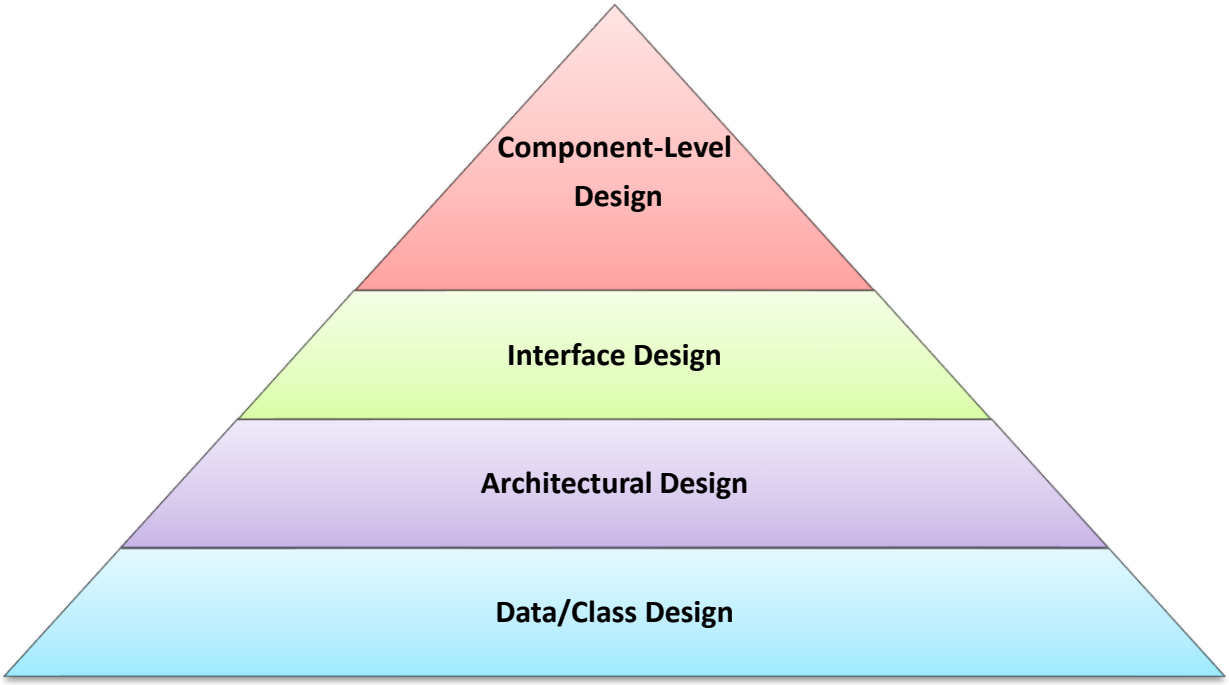


Figure 1 Design Model

Chapter 2. Architectural Design of PGDIT Automation System

The software architecture of a program is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them. Architectural design represents the structure of data and program components that are required to build a computer-based system. It considers the architectural style that the system will take, the structure and properties of the components that constitute the system, and the interrelationships that occur among all architectural components of a system. Architectural design provides big picture of the system.

Figure 2 shows the steps of Architectural design.

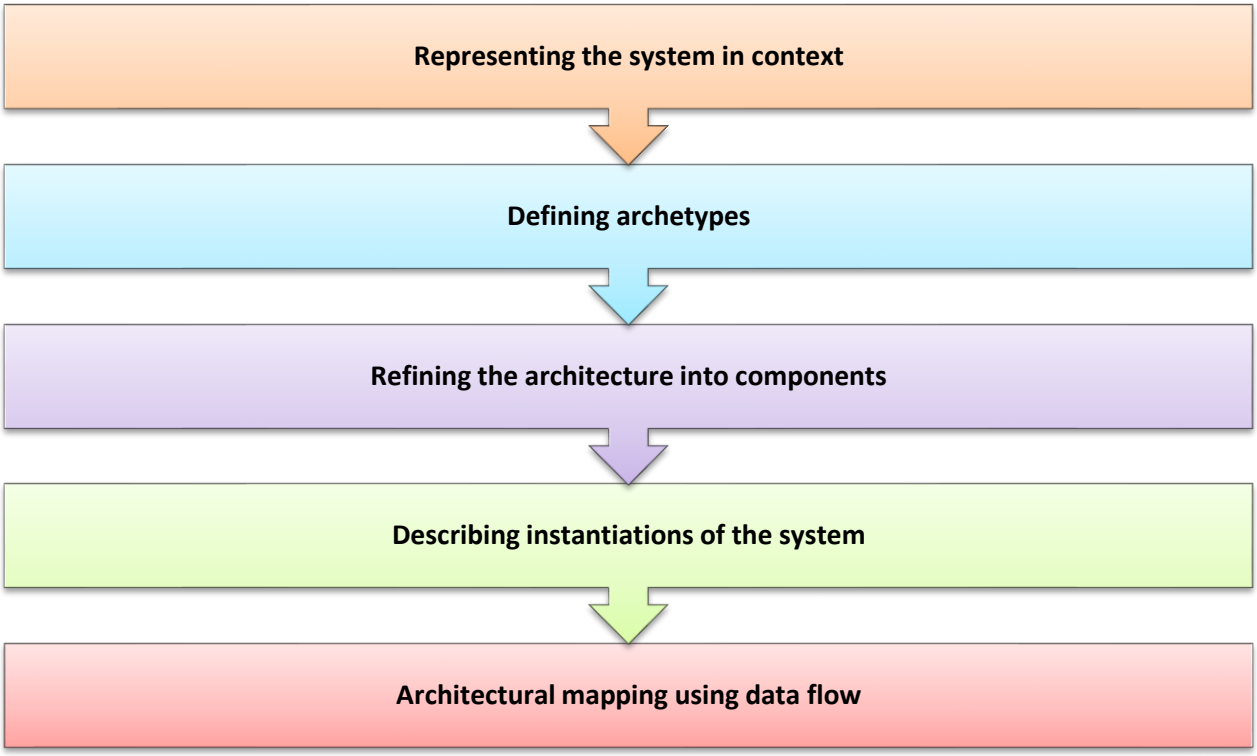


Figure 2 Steps of Architectural Design

2.1 Representing the System in Context

Architectural Context Diagram is used to model the manner in which software interacts with entities external to its boundaries. In architectural context diagram systems that interoperate with the target are represented as

- Super-ordinate systems: Use target system as part of some higher level processing scheme

- Sub-ordinate systems: Used by target system and provide necessary data or processing
- Peer-level systems: Interact on a peer-to-peer basis with target system to produce or consume data
- Actors: People or devices that interact with target system to produce or consume data

Each of these external entities communicates with the target system through an interface.

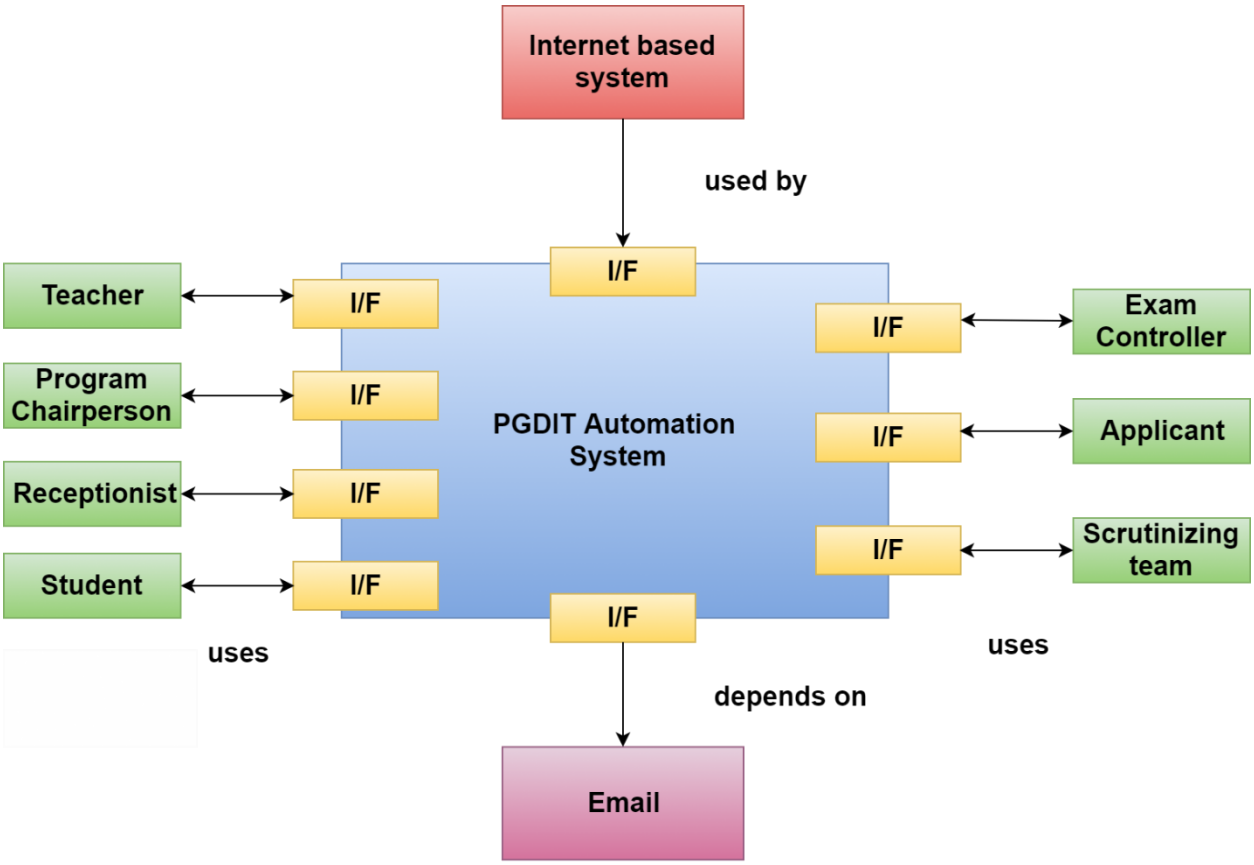


Figure 3 Architectural Context Diagram of PGDIT Automation System

Figure 3 shows architectural context diagram of PGDIT Automation System. The Internet based system is the superordinate system here. Teacher, program chairperson, exam controller, scrutinizing team, student, applicant, and receptionist are actors that are both producers and consumers of information used or produced by the PGDIT Automation System. Finally, email is used by the system and is shown as subordinate to it.

2.2 Defining Archetypes

An archetype is a class or pattern that represents a core abstraction that is critical to the design of an architecture for the target system. They represent stable elements of the architecture. Archetypes can be derived by examining the analysis classes defined as part of the requirements model. The archetypes of PGDIT Automation System are:

- Teacher
- ProgramChairperson
- ExamController
- Scrutinizer
- Student
- Applicant
- Receptionist

The archetypes, their attributes, methods and relationships are illustrated in figure 4.

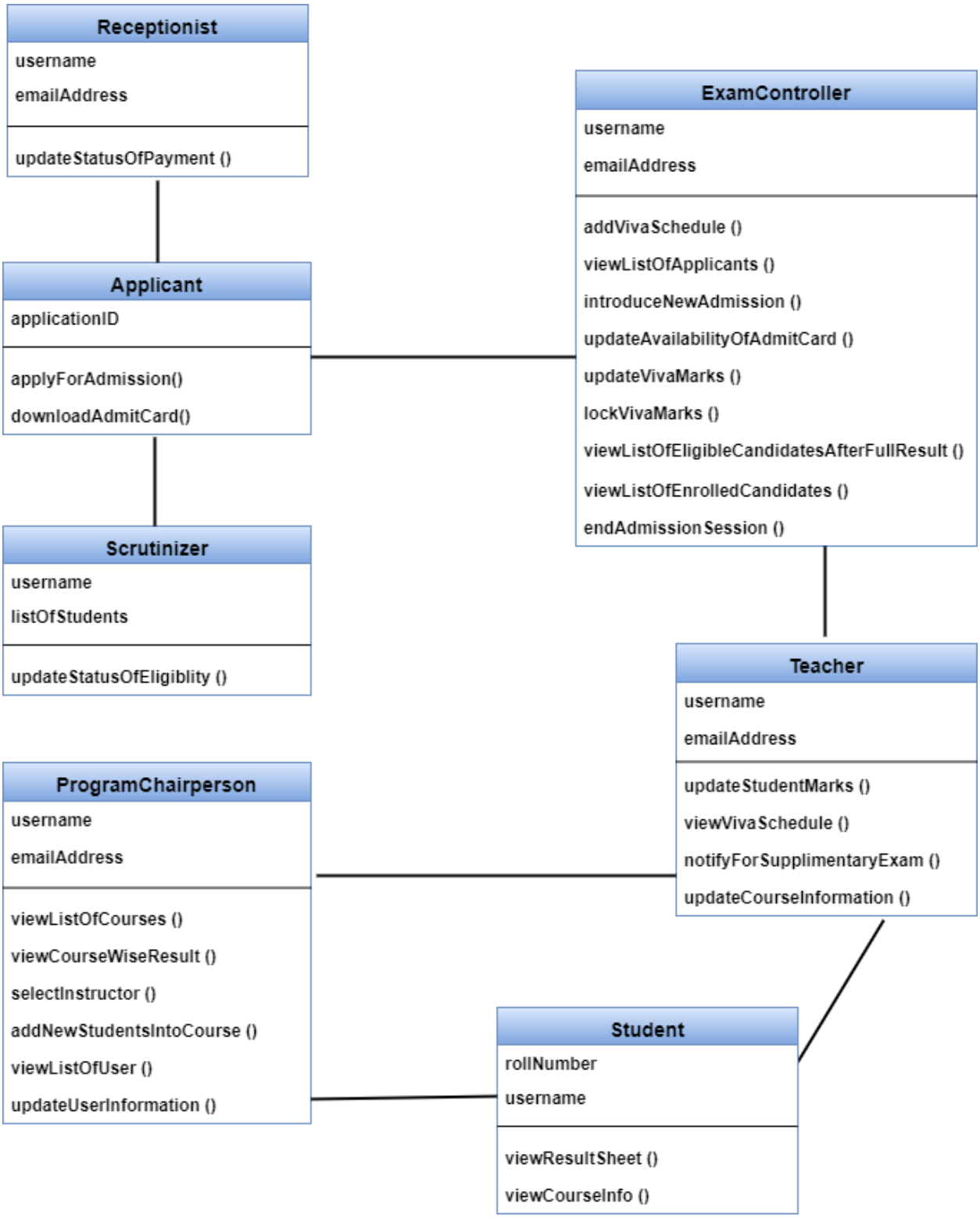


Figure 4 Archetypes

2.3 Refining the Architecture into Components

In this step software architecture is refined into components. These components can be derived from various sources

- Application Domain provides application components.
- Infrastructure Domain provides design components like design classes.
- The interfaces in the ACD imply one or more specialized components that process the data that flow across the interface.

PGDIT Automation System consists of the following top-level components:

- User Account Management: manages works related to user account.
- Admission: handles admission system related activities.
- GUI: manages graphical user interface.
- Course Result Management: handles activities related to course allocation and publishing result.

Figure 5 shows overall architectural structure for PGDIT Automation System with top-level components.

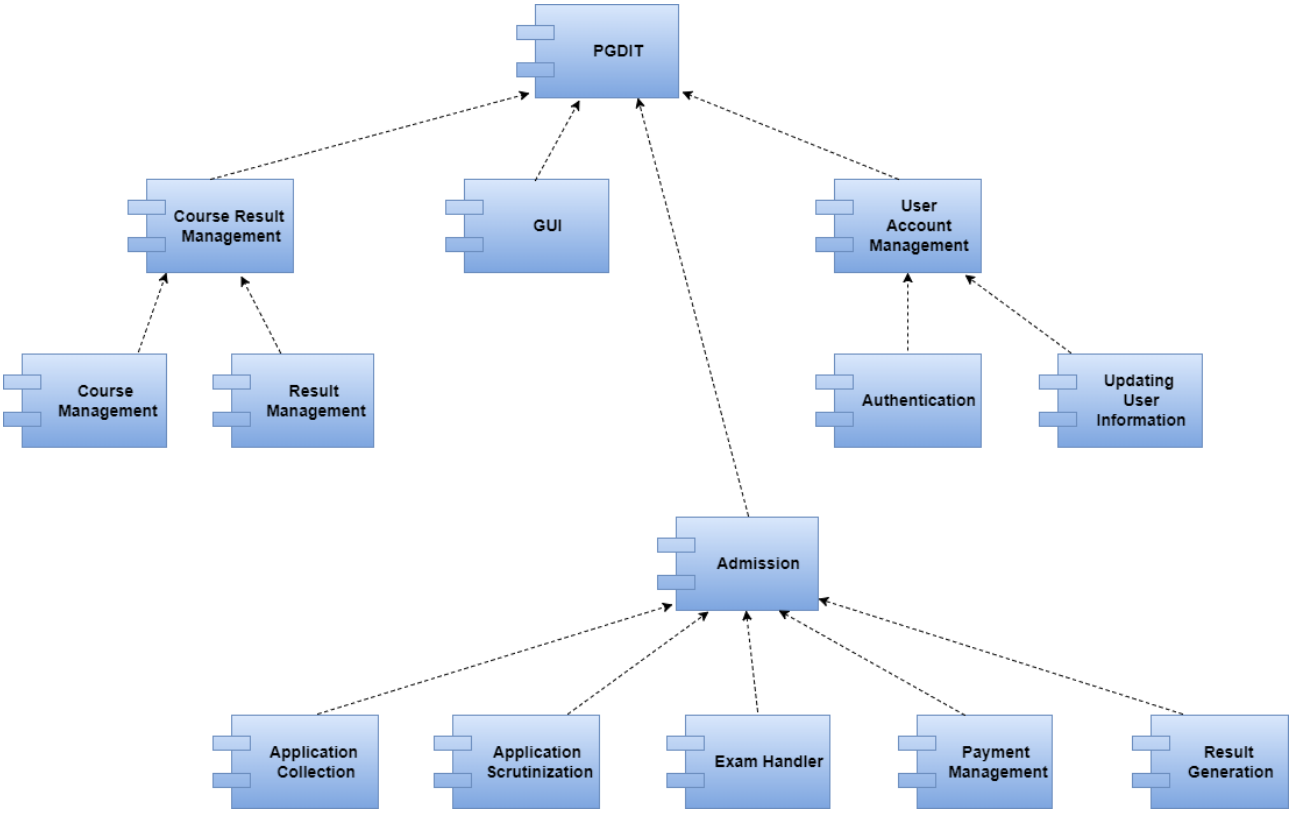


Figure 5 Overall Architectural Structure with Top-Level Components

2.4 Describing Instantiations of the System

Although the major software components have been identified, further refinement is still necessary. To accomplish this, an actual instantiation of the architecture is developed.

Figure 6 illustrates an instantiation of the PGDIT Automation System architecture. Components shown in Figure 5 are elaborated to show additional detail.

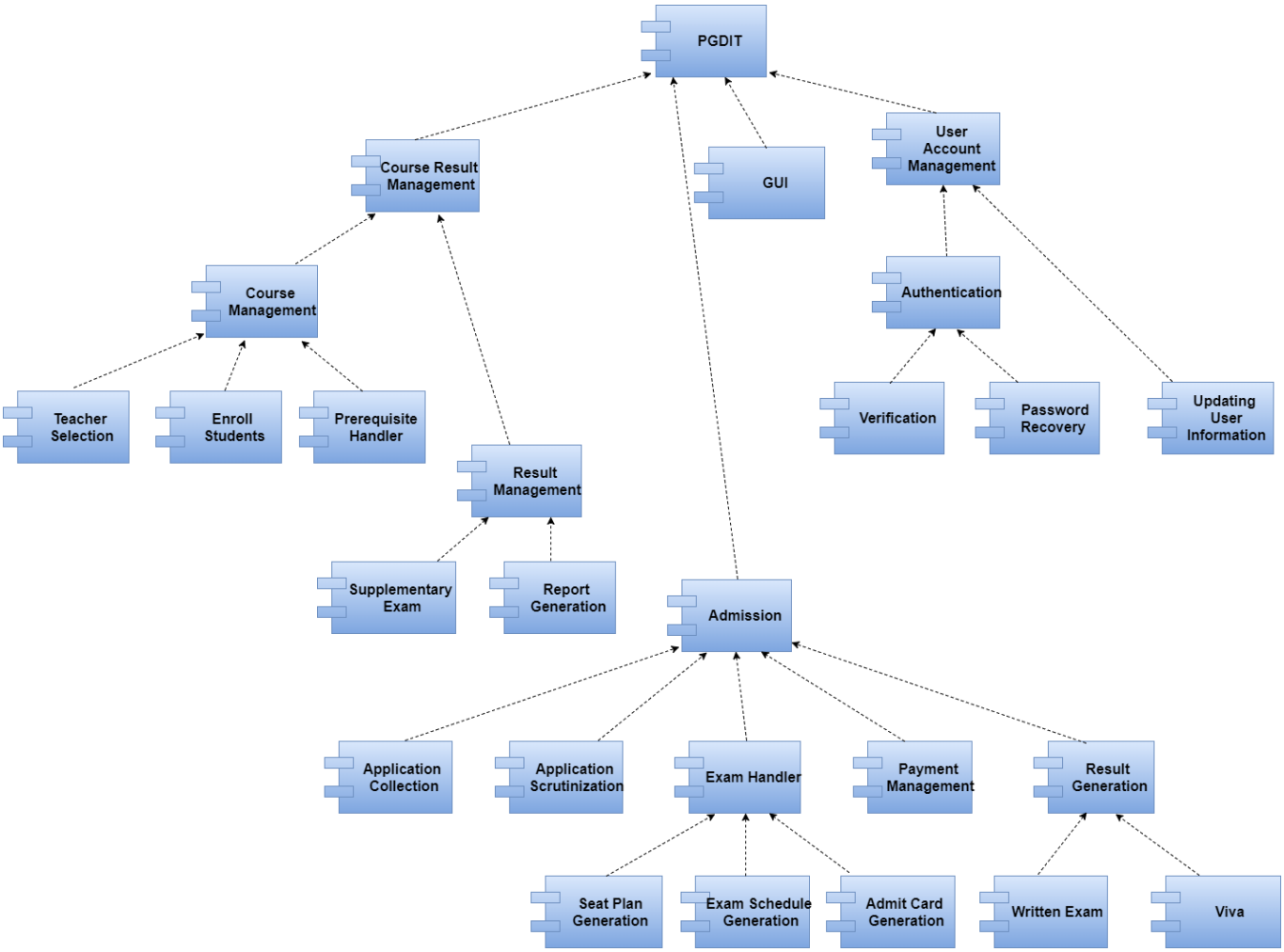


Figure 6 An Instantiation of the PGDIT Automation System Architecture

2.5 Architectural mapping using data flow

A mapping technique called structured design is often characterized as a data flow-oriented design method because it provides a convenient transition from a data flow diagram to software architecture. The transition from information flow to program structure is accomplished as part of a six step process:

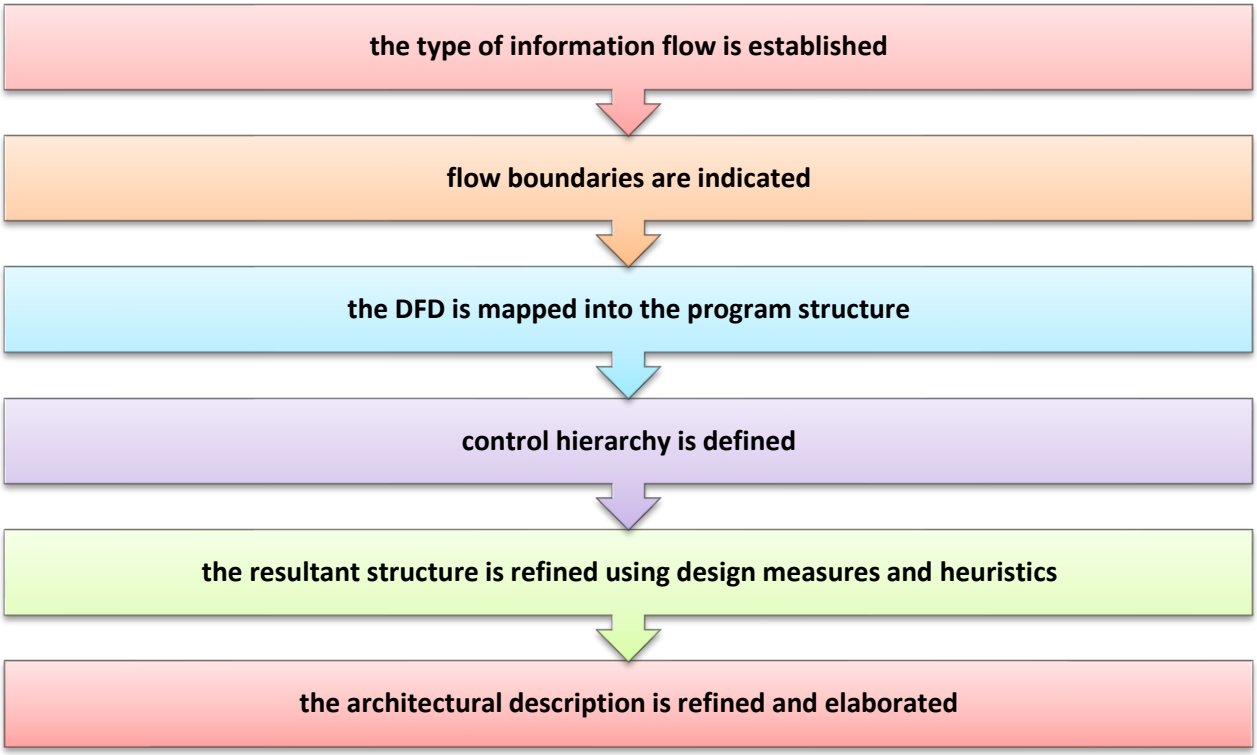


Figure 7 Architectural Mapping

The dataflow diagrams of PGDIT Automation System are shown in figure 8-*.

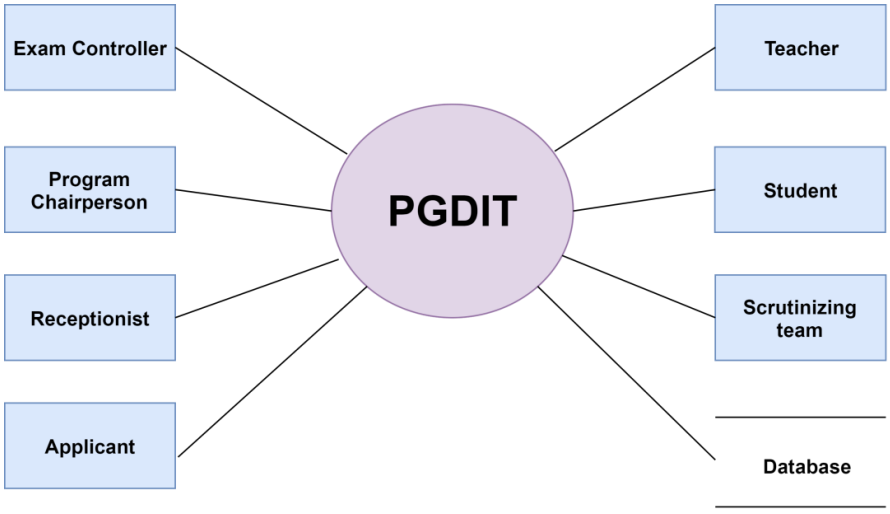


Figure 8 Level 0 Dataflow Diagram

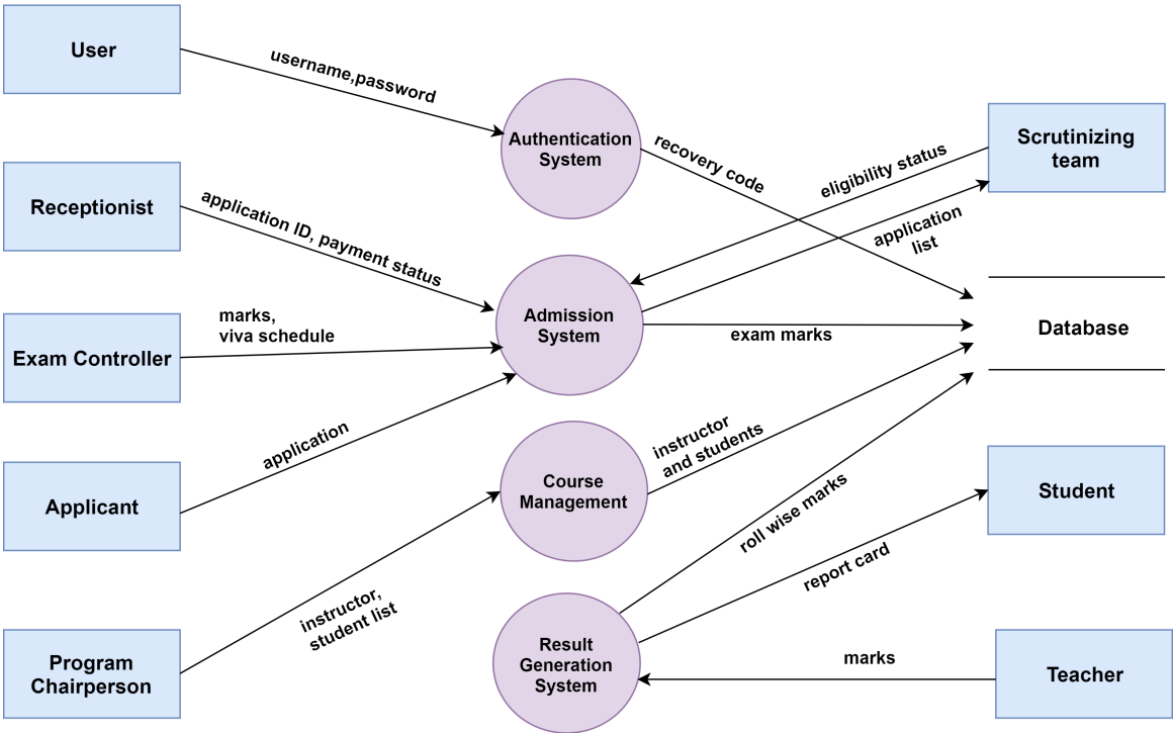


Figure 9 Level 1 Dataflow Diagram

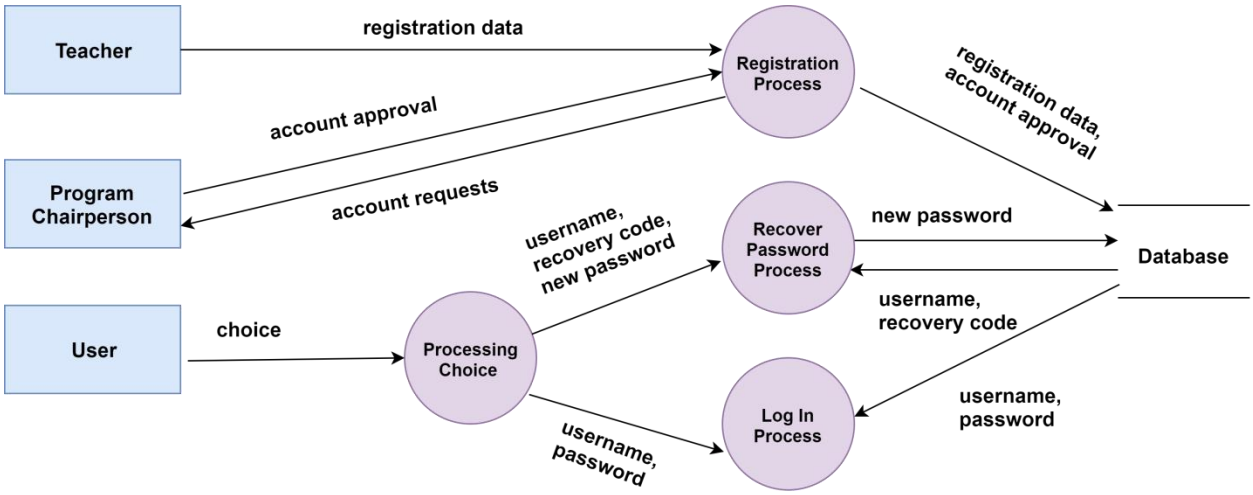


Figure 10 Dataflow Diagram of Authentication and Registration

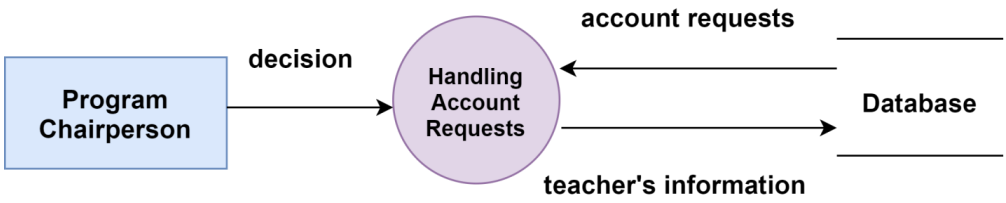


Figure 11 Program Chairperson's Dataflow Diagram of Registration

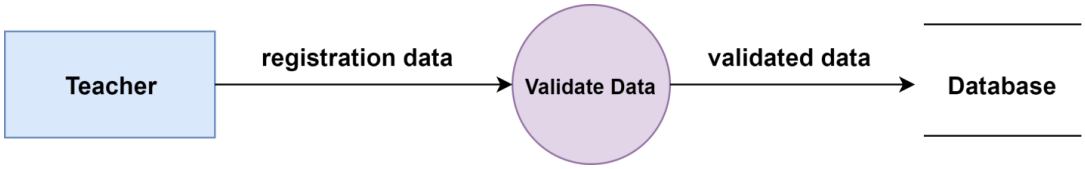


Figure 12 Teacher's Dataflow Diagram of Registration

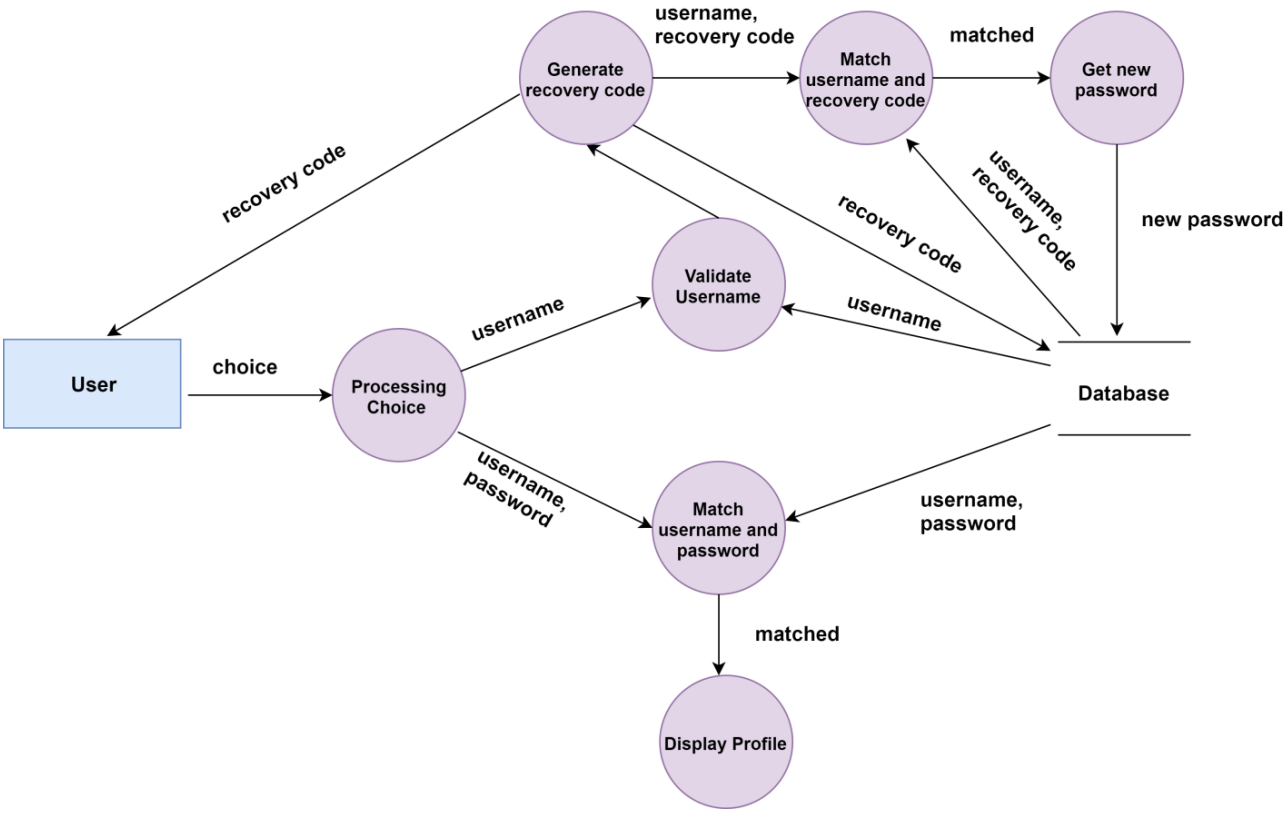


Figure 13 User's Dataflow Diagram of Authentication

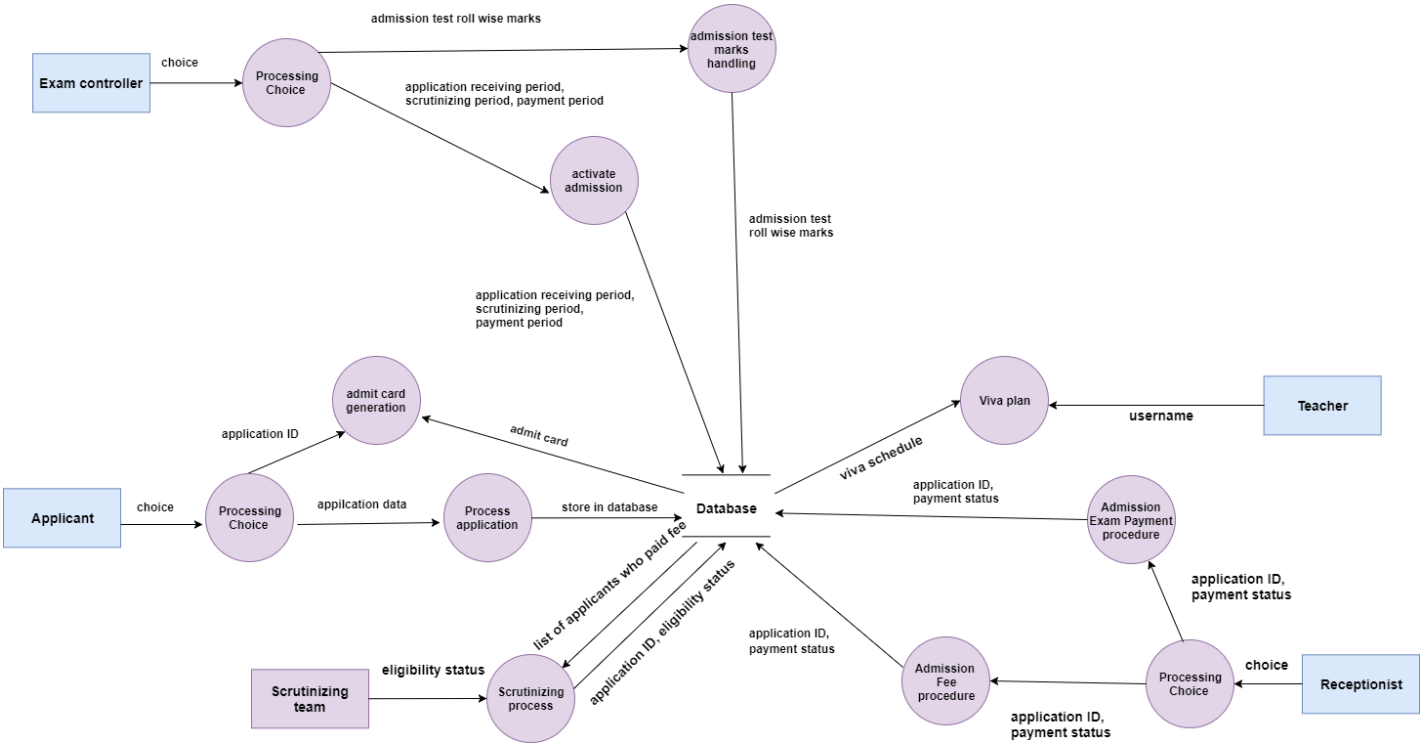


Figure 14 Dataflow Diagram of Admission System

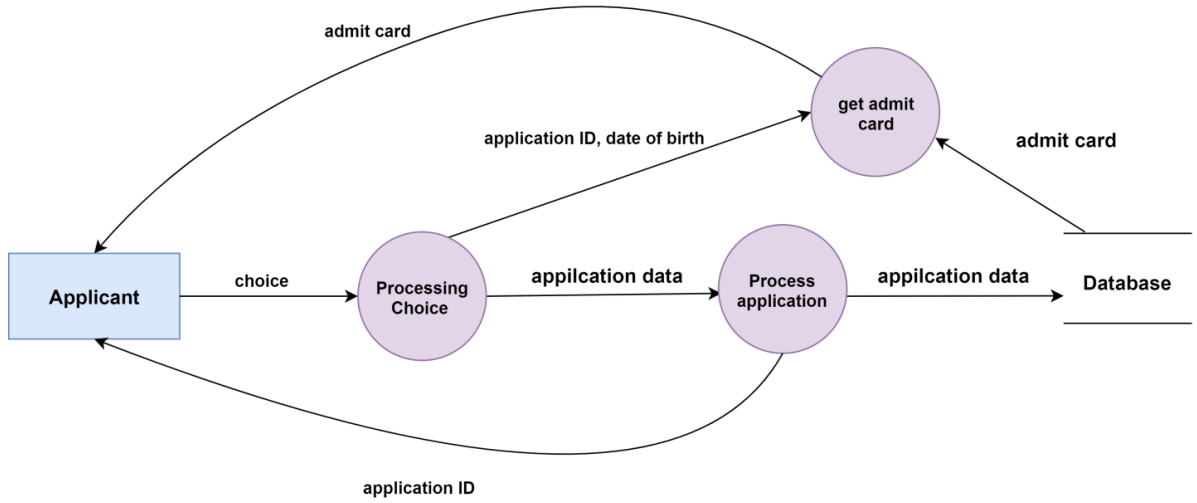


Figure 15 Applicant's Dataflow Diagram of Admission System

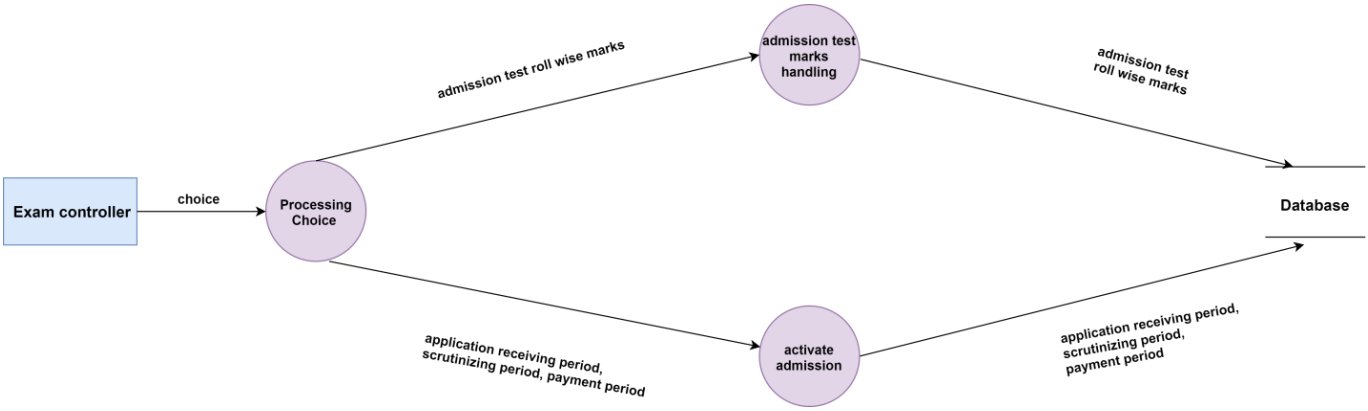


Figure 16 Exam Controller's Dataflow Diagram of Admission System

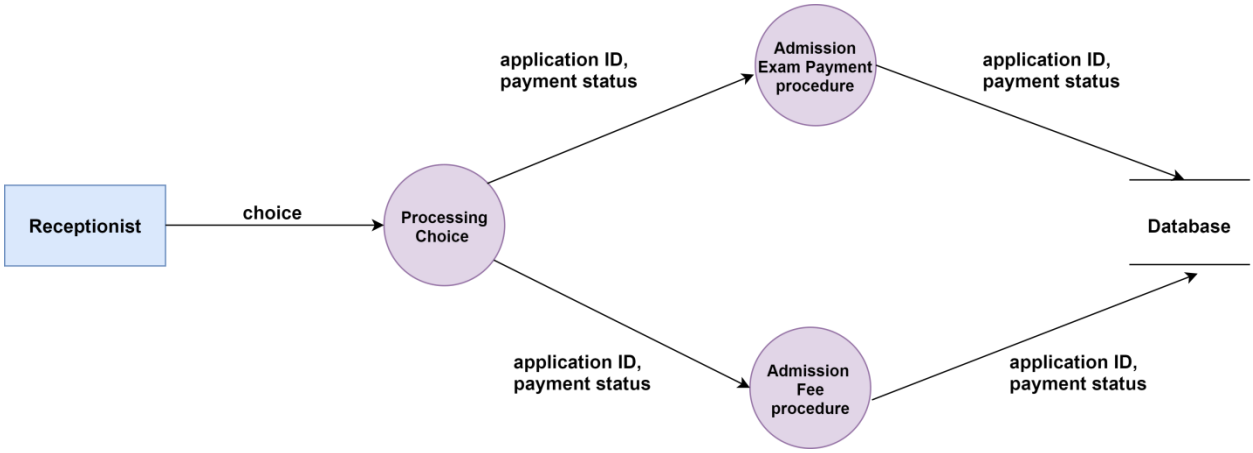


Figure 17 Receptionist's Dataflow Diagram of Admission System

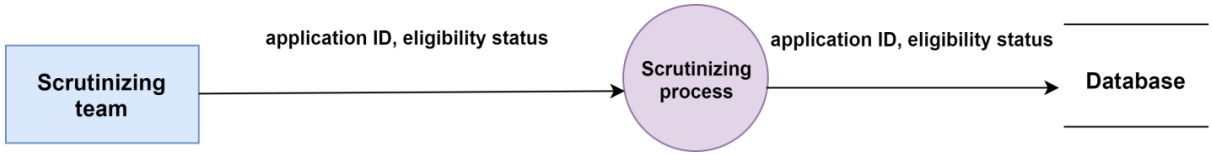


Figure 18 Scrutinizing team's Dataflow Diagram of Admission System

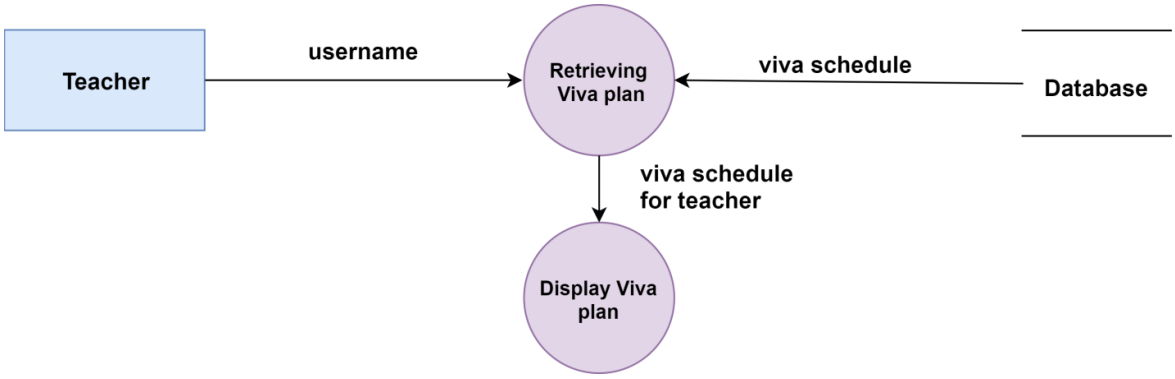


Figure 19 Teacher's Dataflow Diagram of Admission System

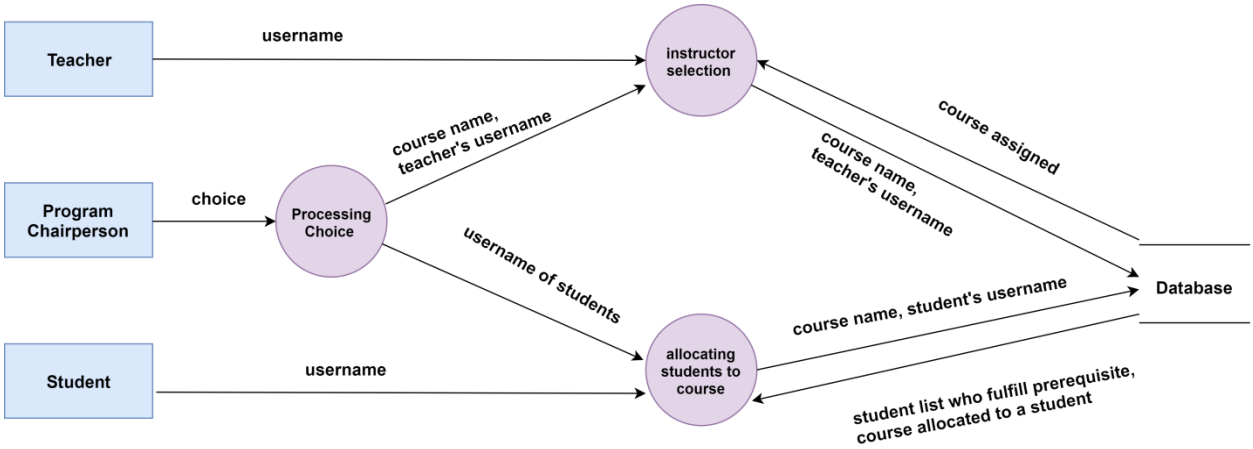


Figure 20 Dataflow Diagram of Course Management System

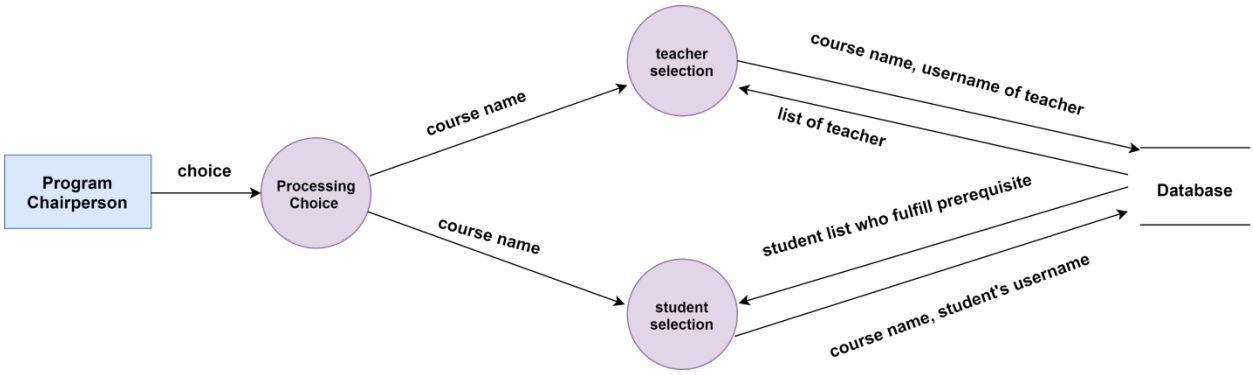


Figure 21 Program Chairperson's Dataflow Diagram of Course Management System

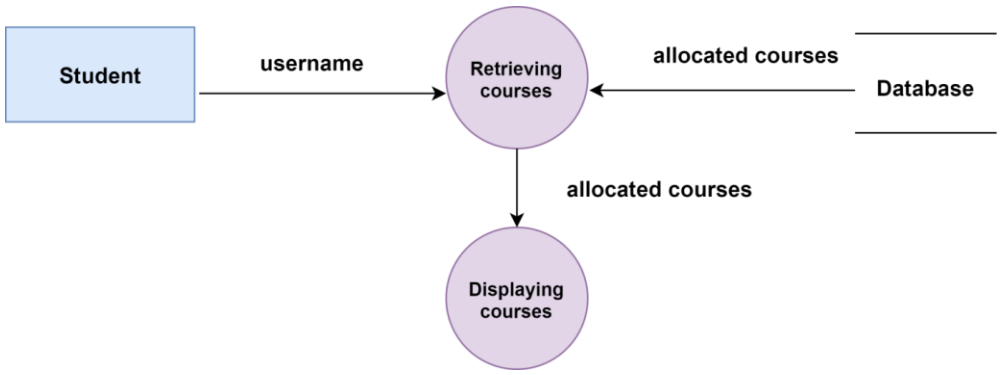


Figure 22 Student's Dataflow Diagram of Course Management System

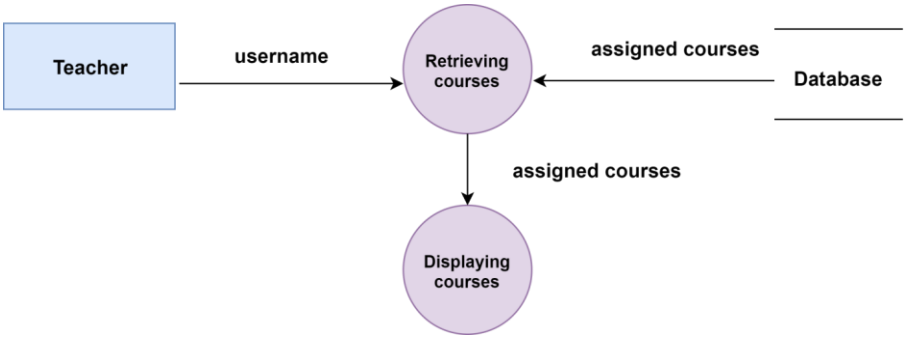


Figure 23 Teacher's Dataflow Diagram of Course Management System

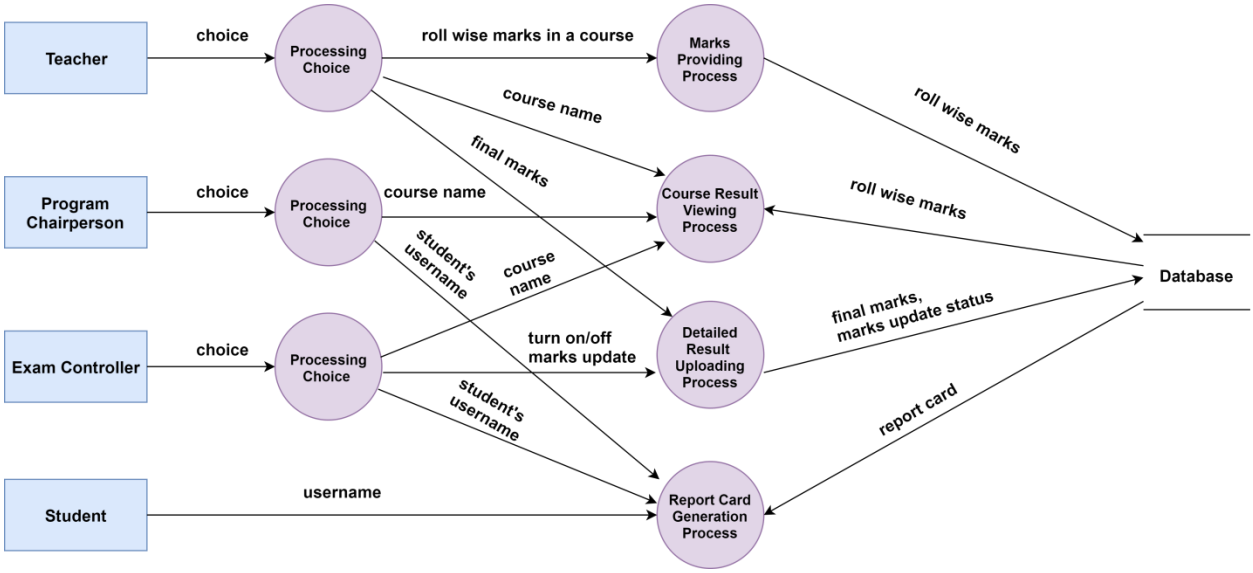


Figure 24 Dataflow Diagram of Result Generation System

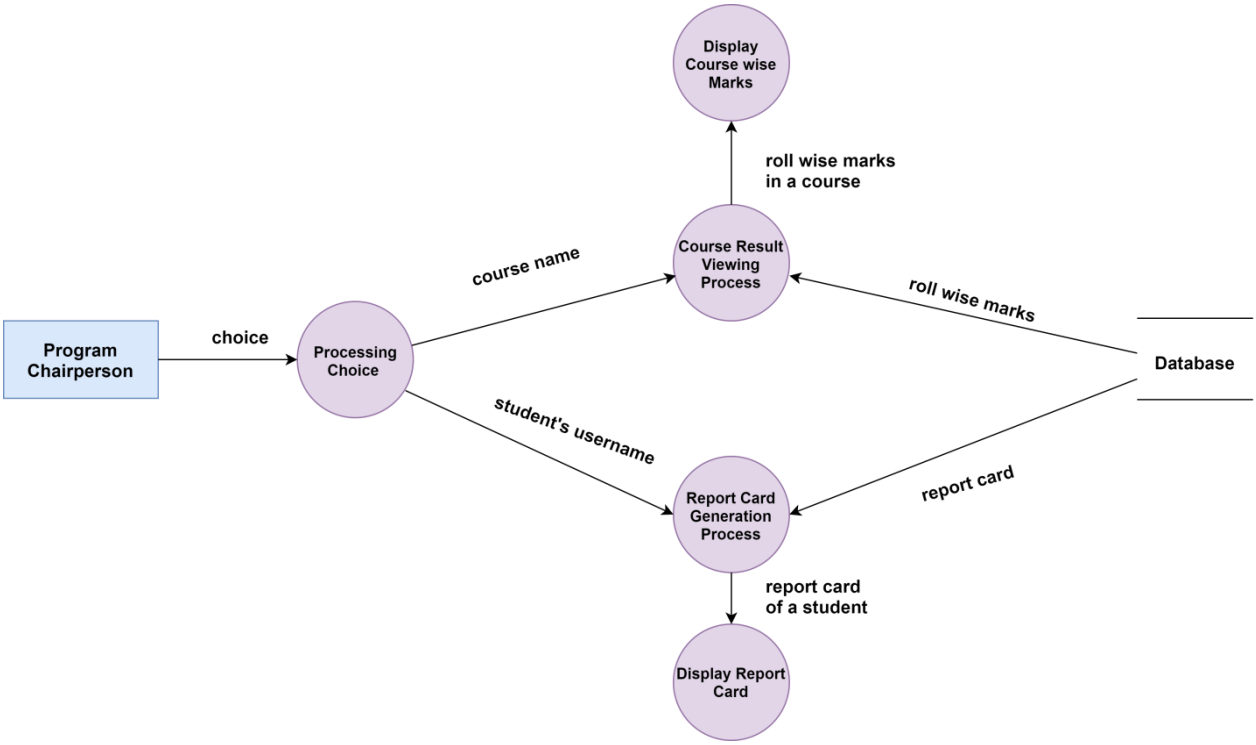


Figure 25 Program Chairperson's Dataflow Diagram of Result Generation System

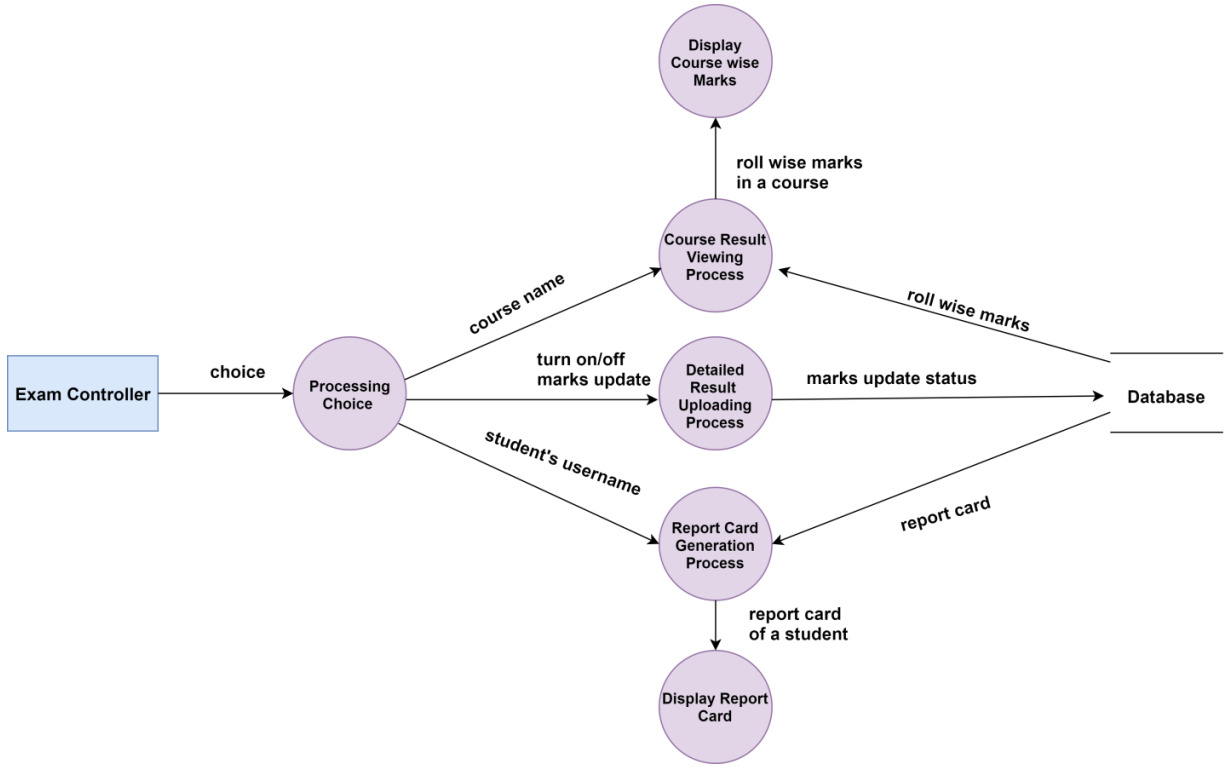


Figure 26 Exam Controller's Dataflow Diagram of Result Generation System

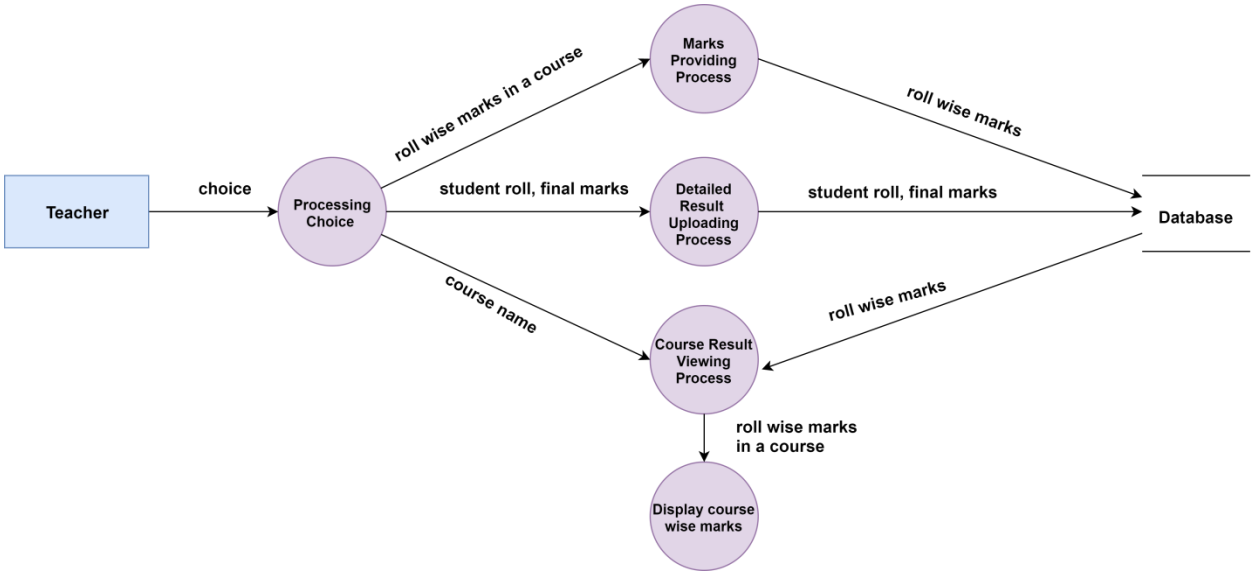


Figure 27 Teacher's Dataflow Diagram of Result Generation System

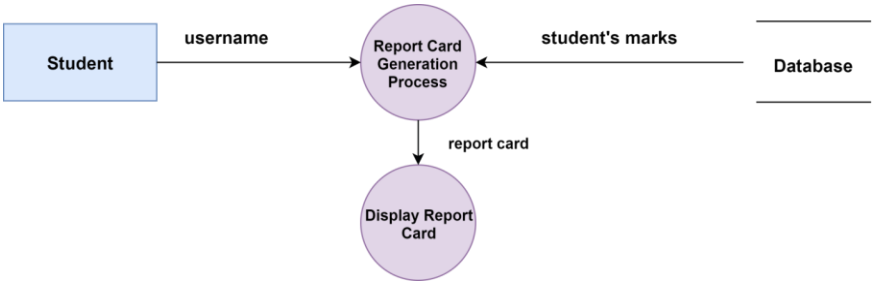


Figure 28 Student's Dataflow Diagram of Result Generation System

Figure 29 shows DFD mapping of Teacher's Dataflow Diagram of Registration.

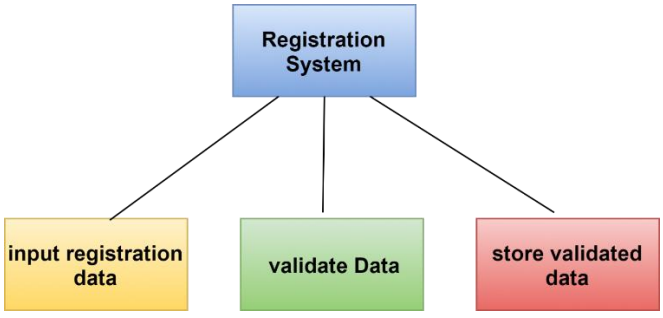


Figure 29 DFD mapping of Teacher's Dataflow Diagram of Registration

Figure 30 shows DFD mapping of Program chairperson's Dataflow Diagram of Registration.

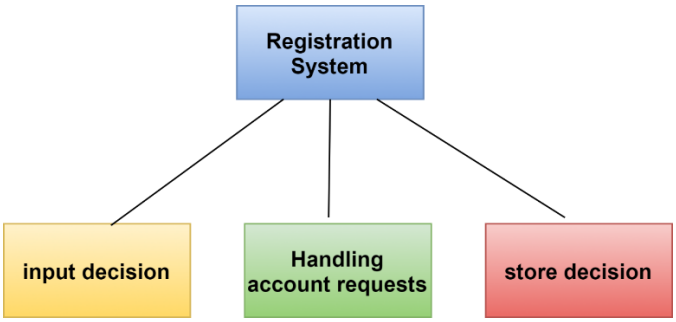


Figure 30 DFD mapping of Program chairperson's Dataflow Diagram of Registration

Figure 31 shows DFD mapping of User's Dataflow Diagram of Authentication.

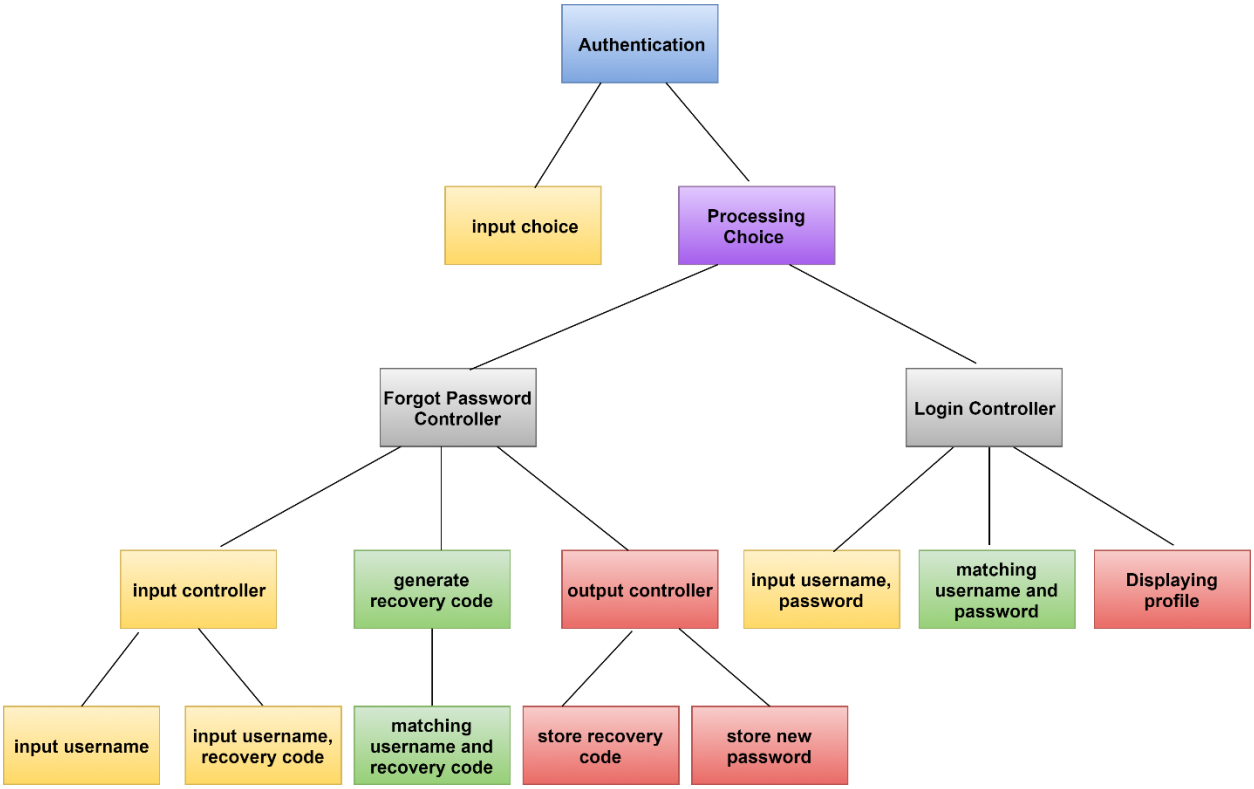


Figure 31 DFD mapping of User's Dataflow Diagram of Authentication

Figure 32 shows DFD mapping of Applicant's Dataflow Diagram of Admission System.

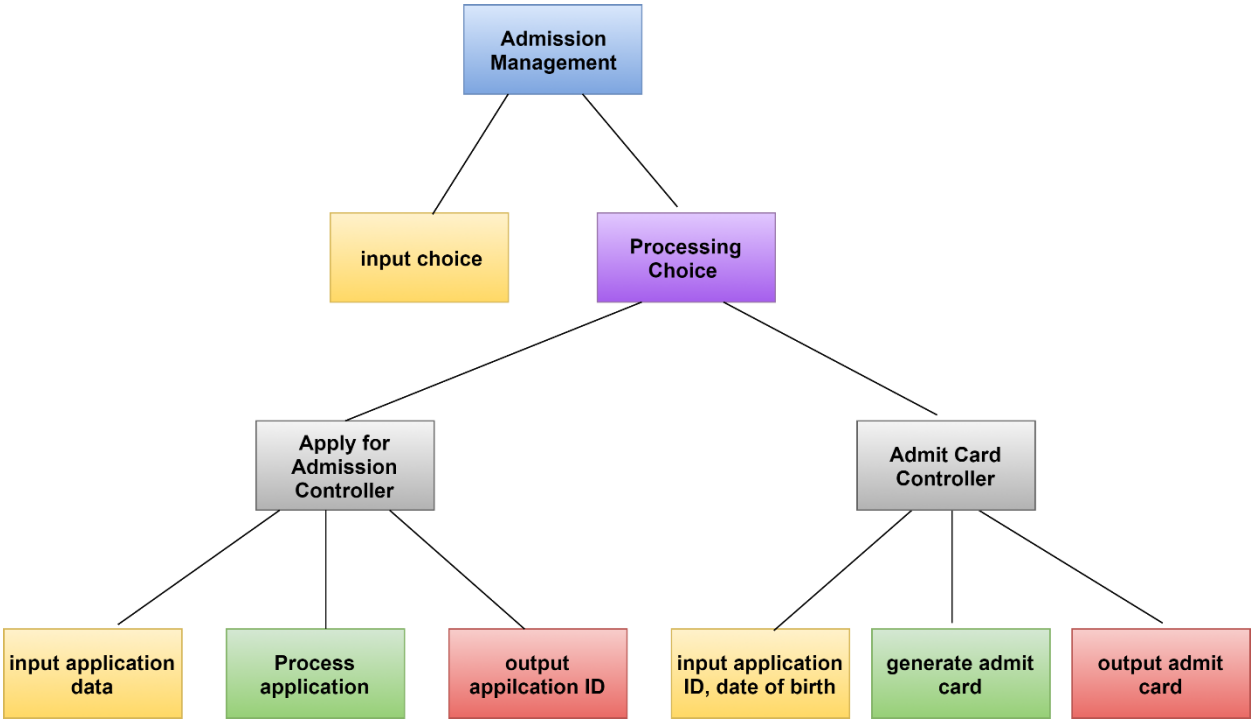


Figure 32 DFD mapping of Applicant's Dataflow Diagram of Admission System

Figure 33 shows DFD mapping of Exam Controller's Dataflow Diagram of Admission System.

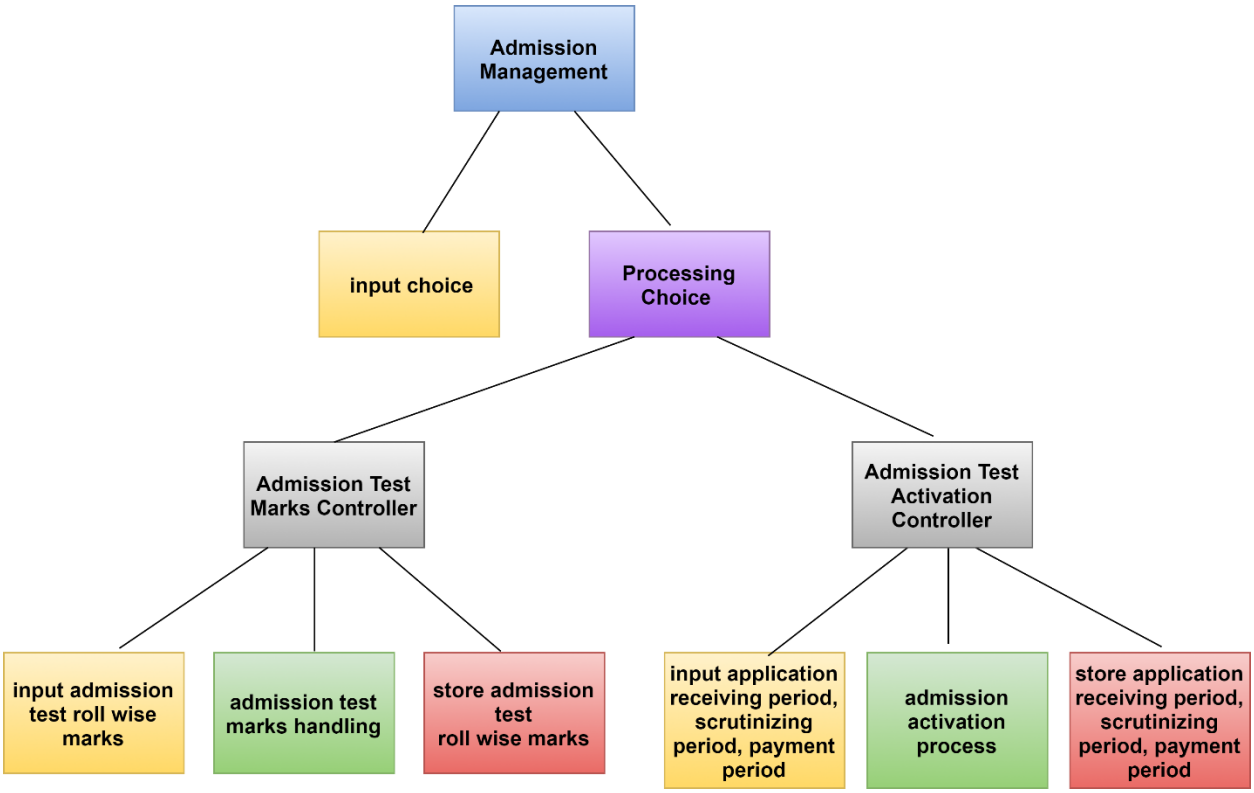


Figure 33 DFD mapping of Exam Controller's Dataflow Diagram of Admission System

Figure 34 shows DFD mapping of Receptionist's Dataflow Diagram of Admission System.

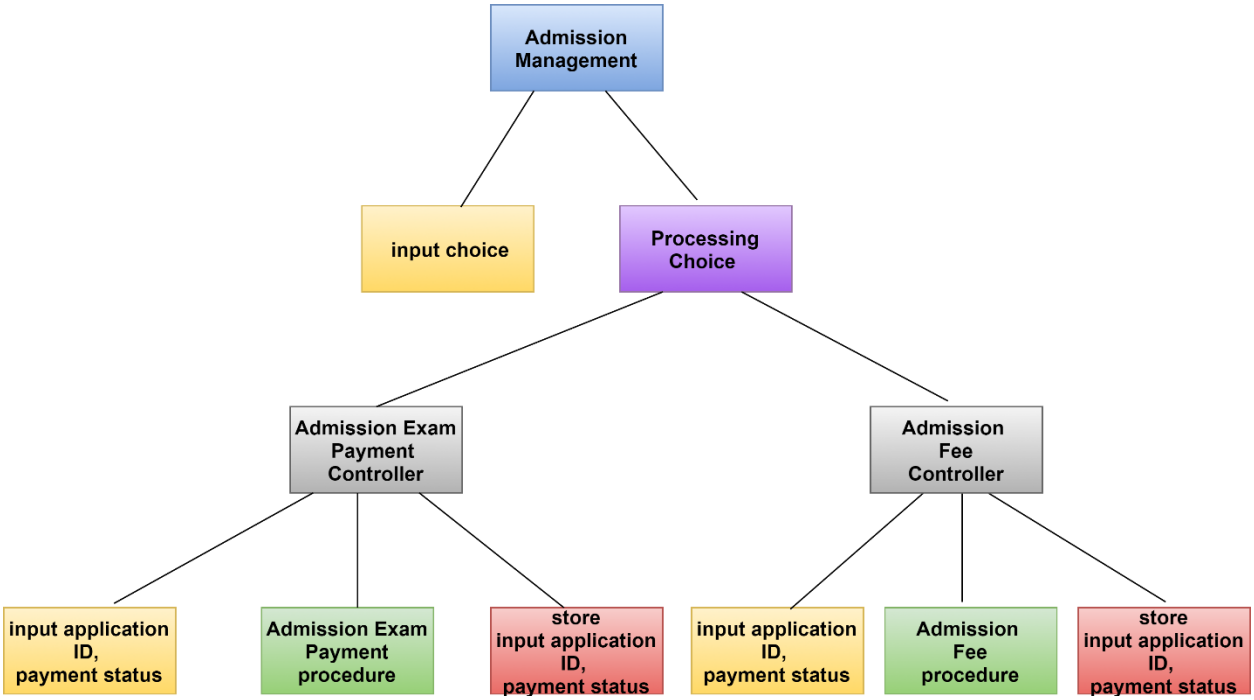


Figure 34 DFD mapping of Receptionist's Dataflow Diagram of Admission System

Figure 35 shows DFD mapping of Scrutinizer's Dataflow Diagram of Admission System.

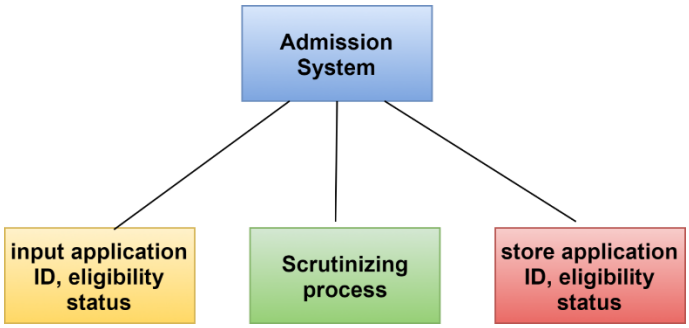


Figure 35 DFD mapping of Scrutinizer's Dataflow Diagram of Admission System

Figure 36 shows DFD mapping of Teacher's Dataflow Diagram of Admission System.

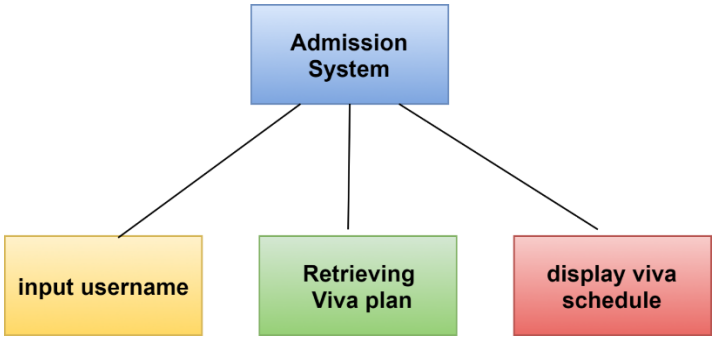


Figure 36 DFD mapping of Teacher's Dataflow Diagram of Admission System

Figure 37 shows DFD mapping of Program Chairperson's Dataflow Diagram of Course Management System.

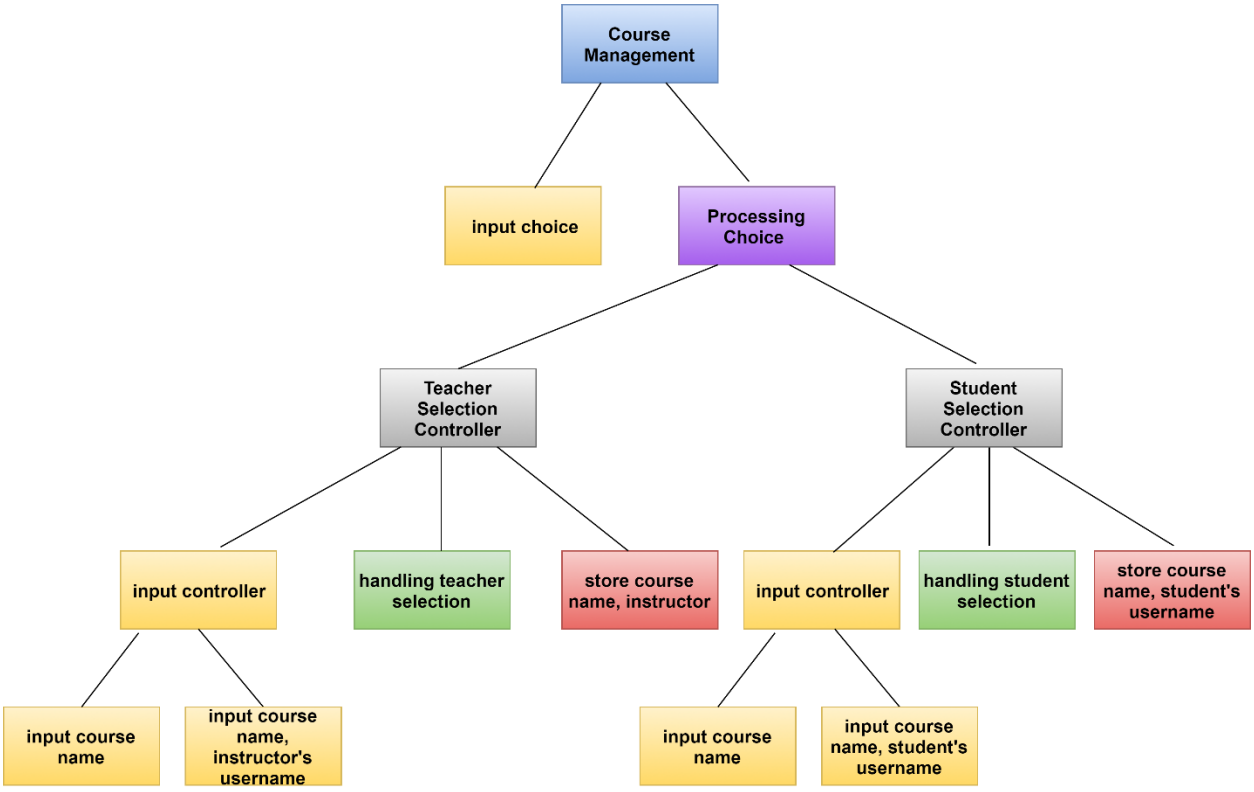


Figure 37 DFD mapping of Program Chairperson's Dataflow Diagram of Course Management System.

Figure 38 shows DFD mapping of Teacher's Dataflow Diagram of Course Management System.

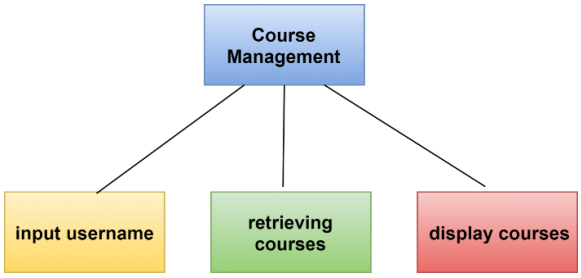


Figure 38 Teacher's Dataflow Diagram of Course Management System

Figure 39 shows DFD mapping of Student's Dataflow Diagram of Course Management System.

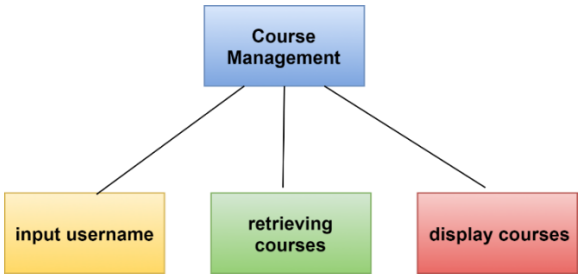


Figure 39 DFD mapping of Student's Dataflow Diagram of Course Management System

Figure 40 shows DFD mapping of Program Chairperson's Dataflow Diagram of Result Generation System.

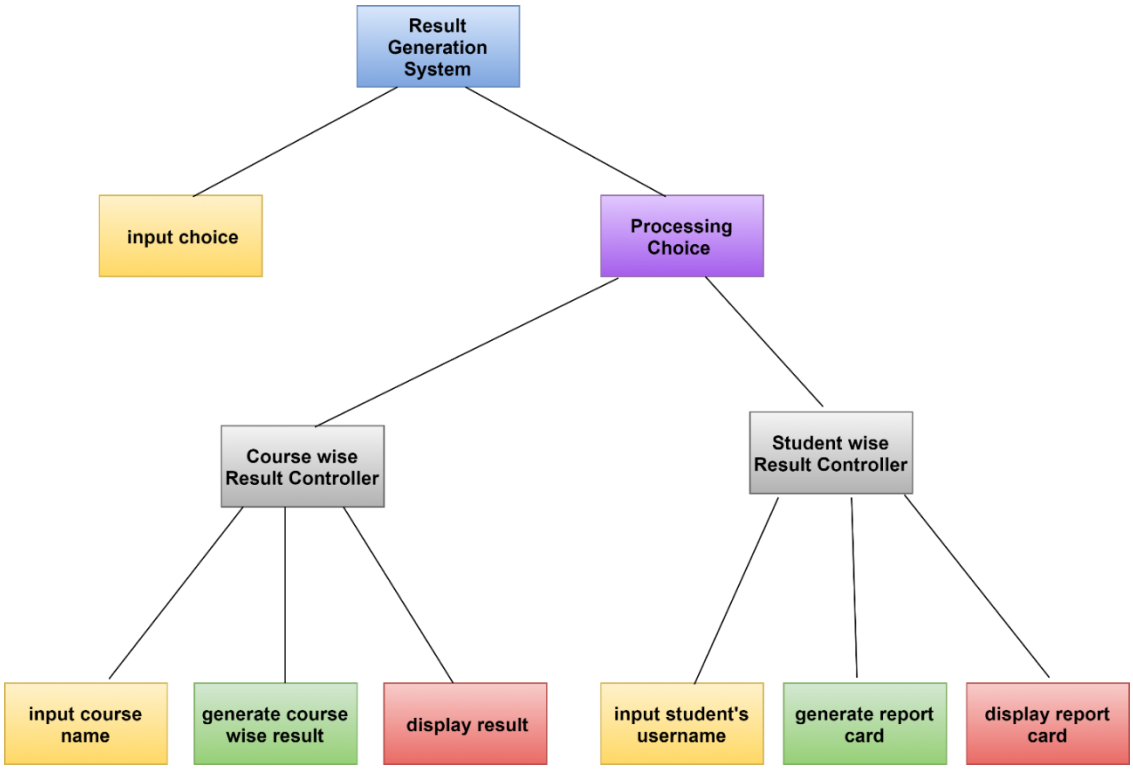


Figure 40 DFD mapping of Program Chairperson's Dataflow Diagram of Result Generation System

Figure 41 shows DFD mapping of Exam Controllers's Dataflow Diagram of Result Generation System.

Figure 41 DFD mapping of Exam Controllers's Dataflow Diagram of Result Generation System

Figure 42 shows DFD mapping of Teacher's Dataflow Diagram of Result Generation System.

Figure 42 DFD mapping of Teacher's Dataflow Diagram of Result Generation System

Figure 43 shows DFD mapping of Student's Dataflow Diagram of Result Generation System.

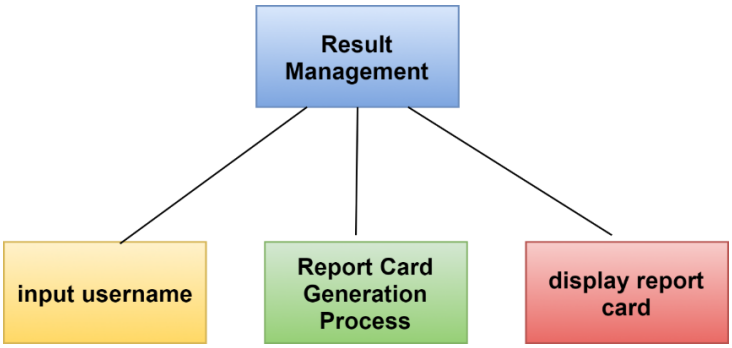


Figure 43 DFD mapping of Student's Dataflow Diagram of Result Generation System

Chapter 3. Component-Level Design of PGDIT Automation System

A component is a modular building block for computer software. According to OMG Unified Modeling Language Specification [OMG03a] component is a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces. Component-level design defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each software component. Component-level design helps to determine whether the software will work before building it. The component-level design represents the software in a way that allows to review the details of the design for correctness and consistency with other design representations. It provides a means for assessing whether data structures, interfaces, and algorithms will work.

Component-level design comprises of the following steps:

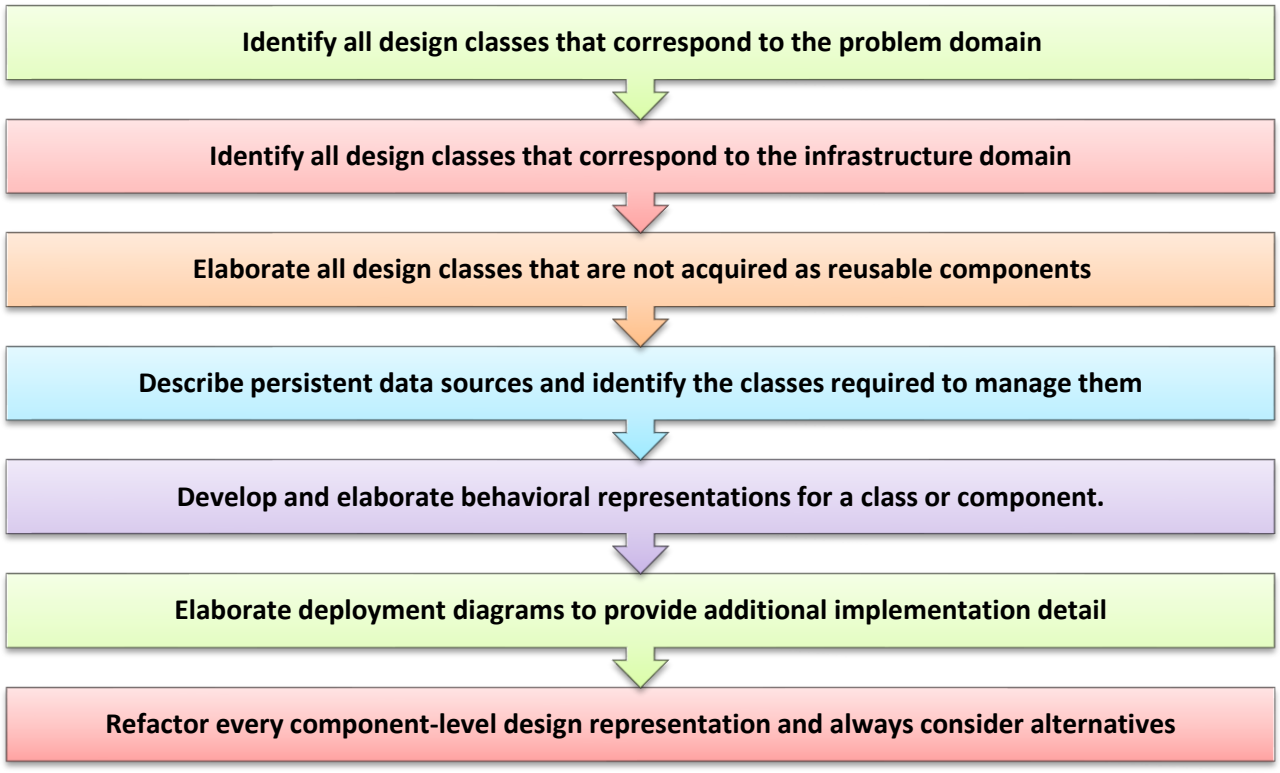


Figure 44 Steps of Component-Level Design

3.1 Identify All Design Classes that Correspond to the Problem Domain

PGDIT Automation System has the following analysis class.

- Teacher
- Student

- ProgramChairperson
- ExamController
- Scrutinizer
- Applicant
- Receptionist

All analysis class and their attributes and methods are shown in figure 45.

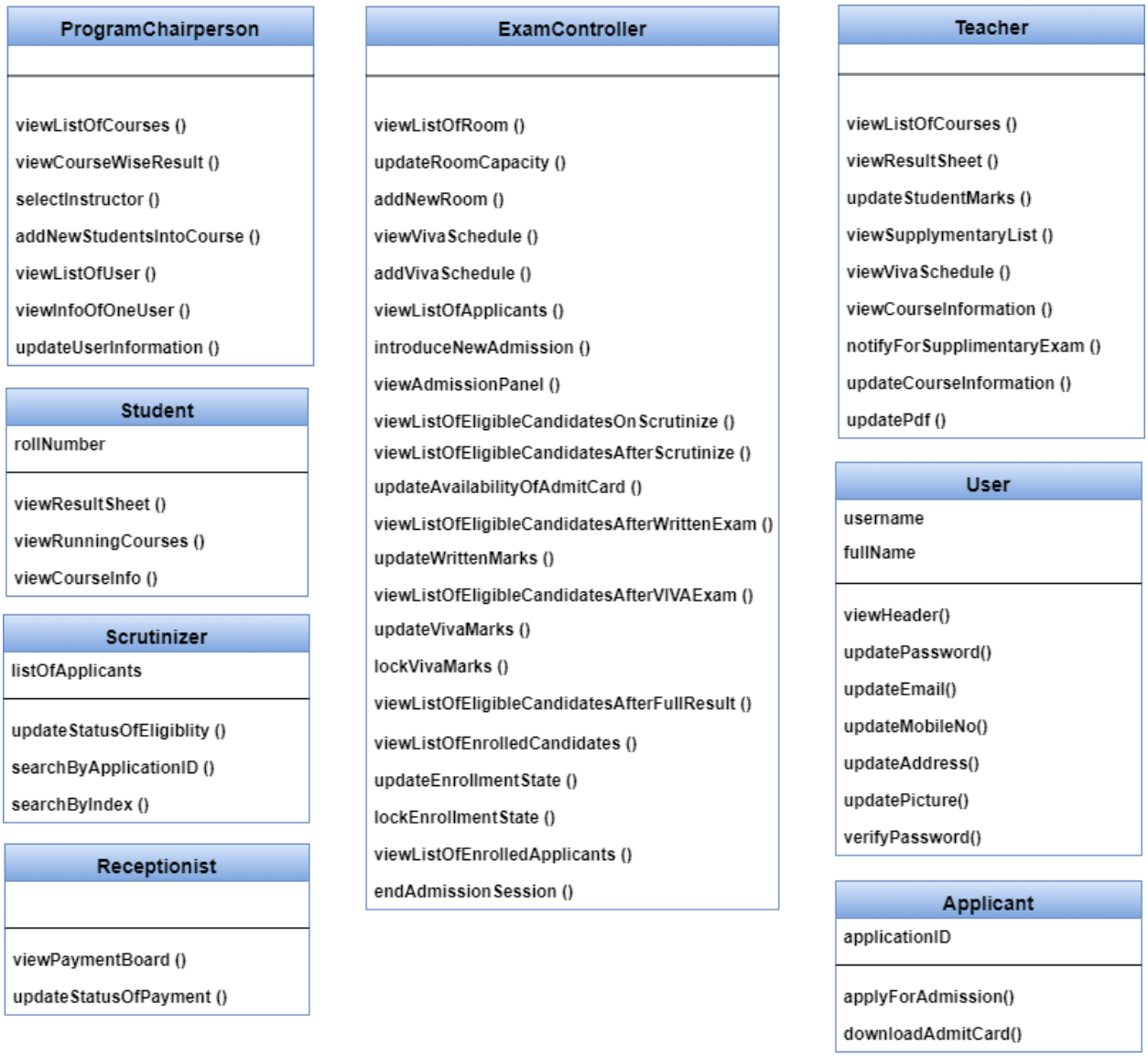


Figure 45 Analysis classes

3.2 Identify All Design Classes that Correspond to the Infrastructure Domain

There are some classes that are not described in the requirements model and are often missing from the architecture model. These classes have been described here. We have found 2 infrastructure domain classes.

- DatabaseHandler: for handling database operations(insert, update, delete, retrieve)
- QueryGenerator: for generationg sql query using javascript object.

Figure 46 shows infrastructure domain classes, their attributes and methods.

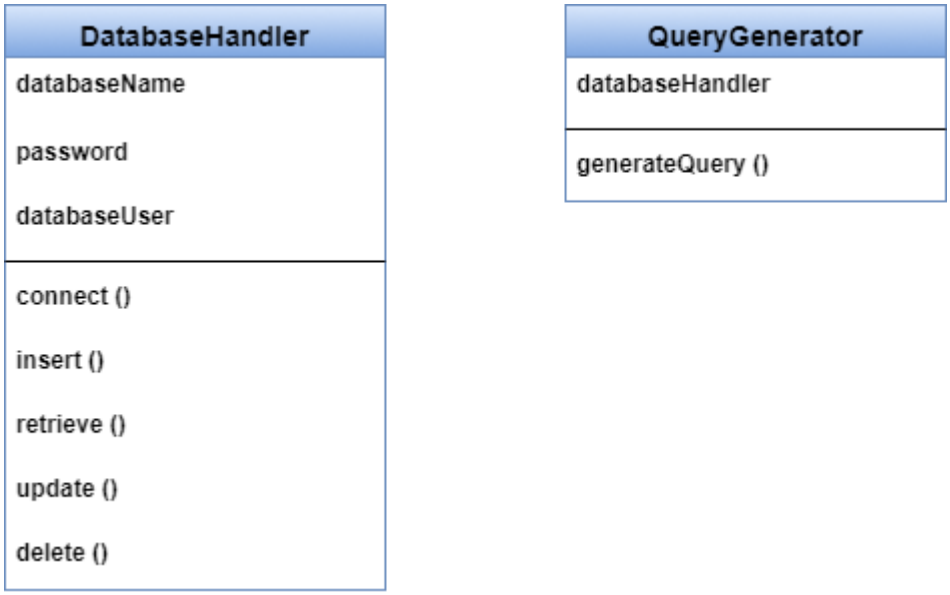


Figure 46 Design classes

3.3 Elaborate All Design Classes that are not Acquired as Reusable Components

Elaboration requires that all interfaces, attributes, and operations necessary to implement the class be described in detail.

Figure 47-54 show elaborated design classes. The attributes and operations defined during requirement analysis are noted at the top of the figures. The interfaces of these design classes are represented using the “lollipop” symbols shown to the left of the component boxes.

Figure 47 shows elaborated Applicant class. It has two interfaces.

Figure 47 Elaborated Applicant class

Figure 48 shows elaborated User class. It has ten interfaces.

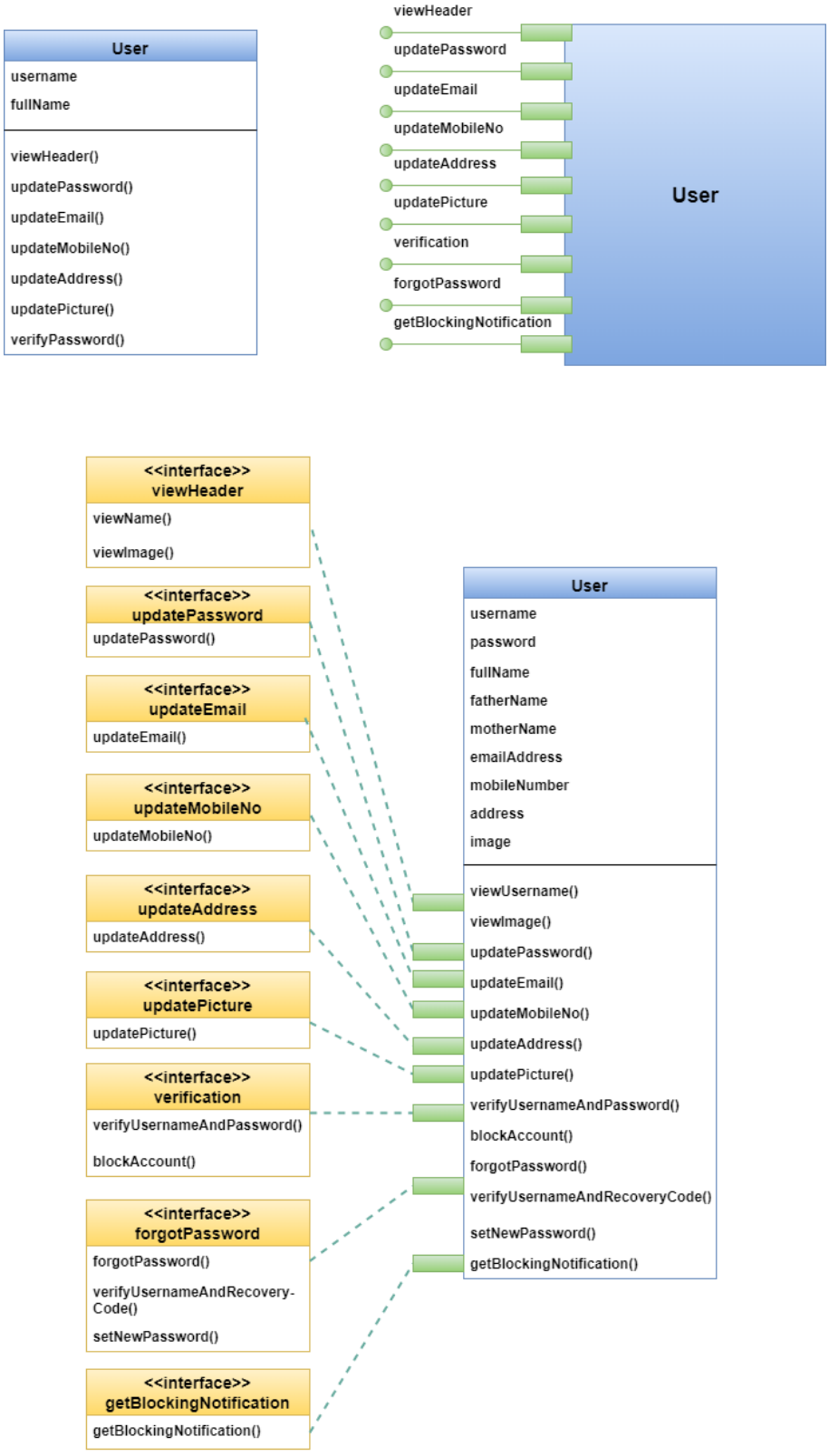


Figure 48 Elaborated User class

Figure 49 shows elaborated ProgramChairperson class. It has eleven interfaces.

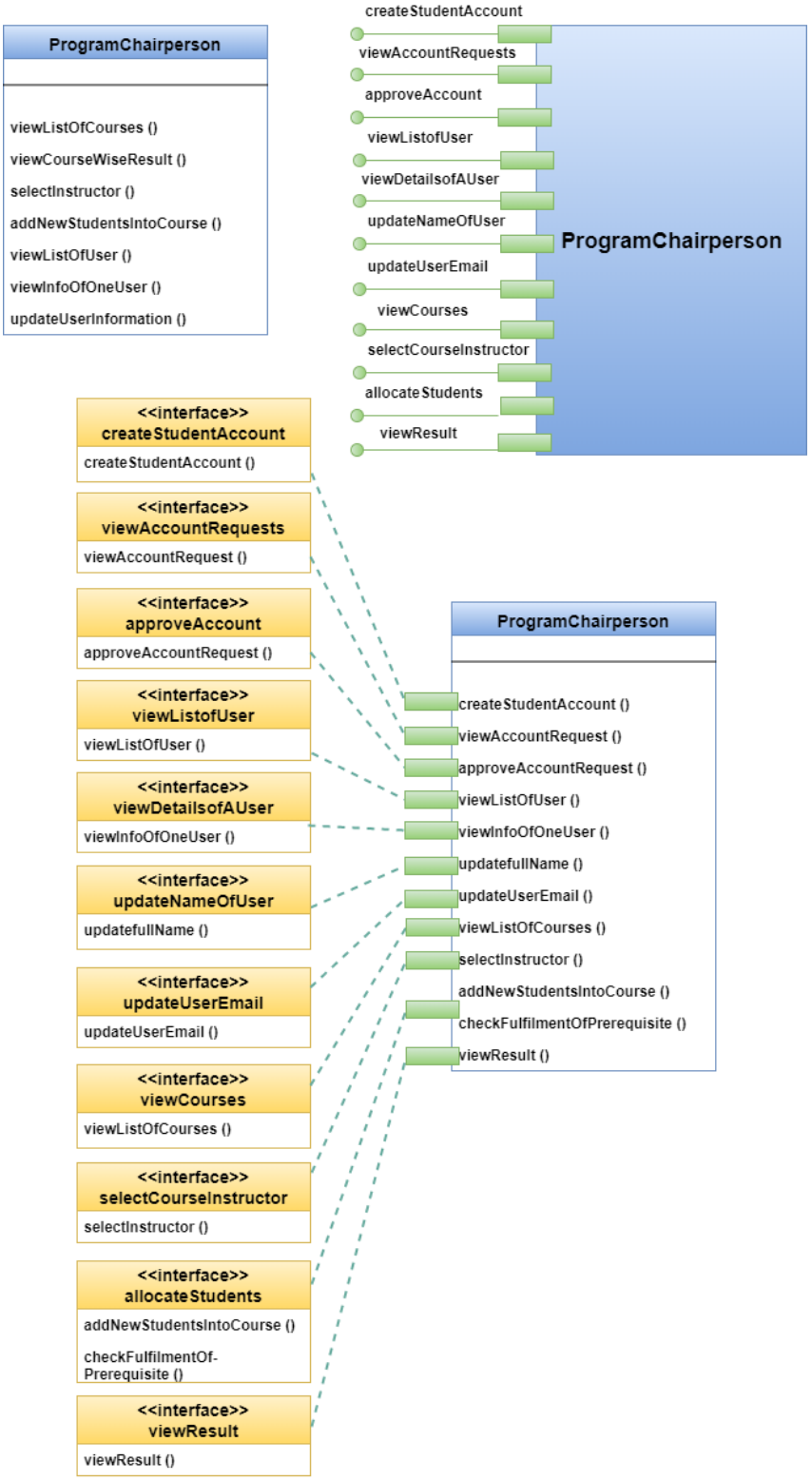


Figure 49 Elaborated ProgramChairperson class

Figure 50 shows elaborated Scrutinizer class. It has four interfaces.

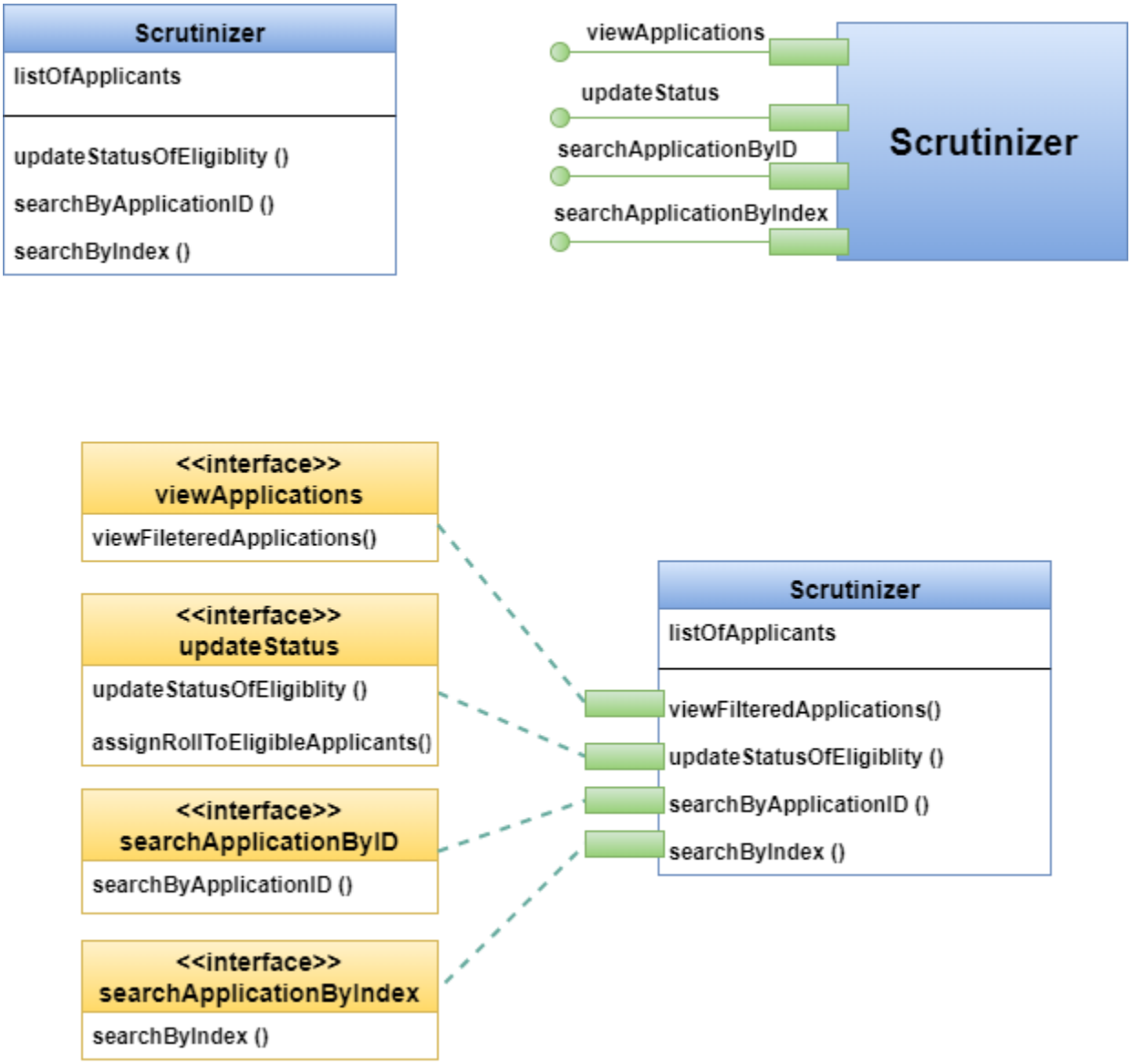


Figure 50 Elaborated Scrutinizer class

Figure 51 shows elaborated Receptionist class. It has two interfaces.

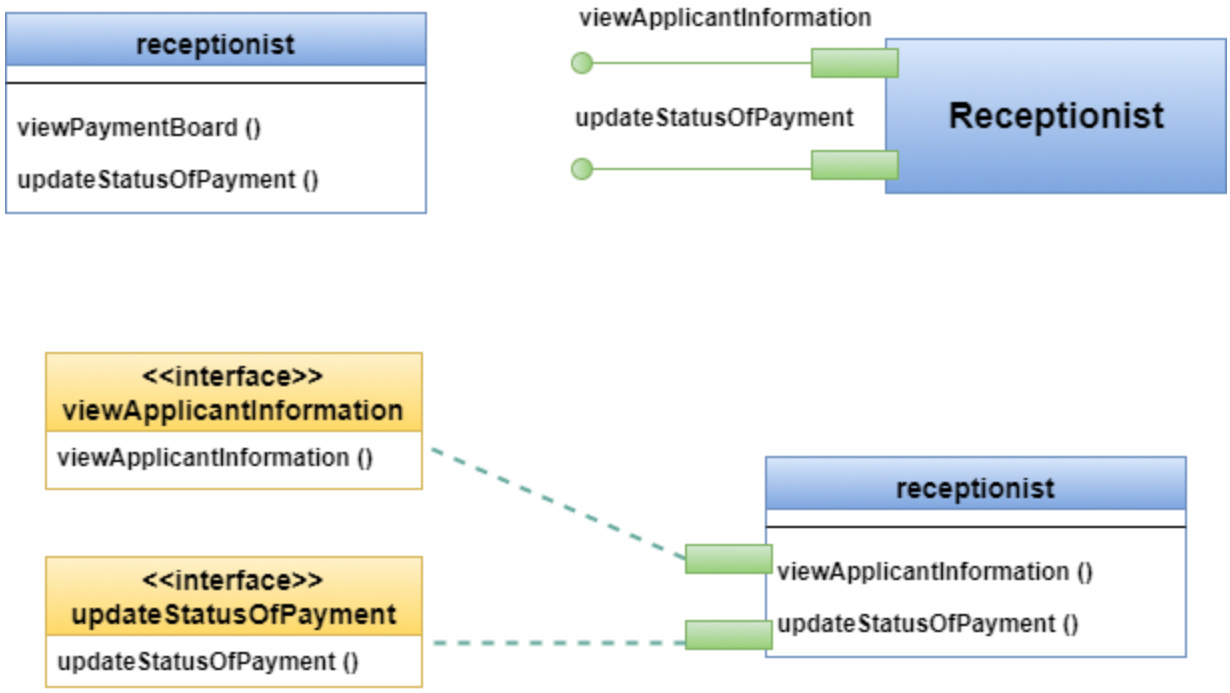


Figure 51 Elaborated Receptionist class

Figure 52 shows elaborated ExamController class. It has twenty four interfaces.


ExamController
viewListOfRoom()
updateRoomCapacity()
addNewRoom()
viewVivaSchedule()
addVivaSchedule()
introduceNewAdmission()
updateAdmissionInfo()
viewAdmissionPanel()
viewListOfApplicants()
viewListOfEligibleCandidates-OnScrutinize()
viewListOfEligibleCandidates-AfterScrutinize()
updateAvailabilityOfAdmitCard()
viewListOfEligibleCandidates-AfterWrittenExam()
updateWrittenMarks()
lockWrittenMarks()
viewListOfEligibleCandidates-AfterVIVAExam()
updateVivaMarks()
lockVivaMarks()
viewListOfEligibleCandidates-AfterFullResult()
viewListOfEnrolledCandidates()
updateEnrollmentState()
lockEnrollmentState()
endAdmissionSession()
viewRooms
updateRoomCapacity
addNewRoom
generateSeatPlan
viewVivaSchedule
updateVivaSchedule
addApplicationTime
addPaymentTime
addScrutinizationTime
updateAdmissionInfo
viewAdmissionPanel
viewApplicants
viewEligibleCandidatesOnScrutinize
viewEligibleCandidatesAfterScrutinize
updateAvailabilityOfAdmitCard
viewEligibleCandidatesAfterWrittenExam
updateWrittenMarks
lockWrittenMarks
viewEligibleCandidatesAfterVIVAExam
updateVivaMarks
lockVivaMarks
viewEligibleCandidatesAfterFullResult
viewEnrolledCandidates
updateEnrollmentState
lockEnrollmentState
endAdmissionSession
ExamController

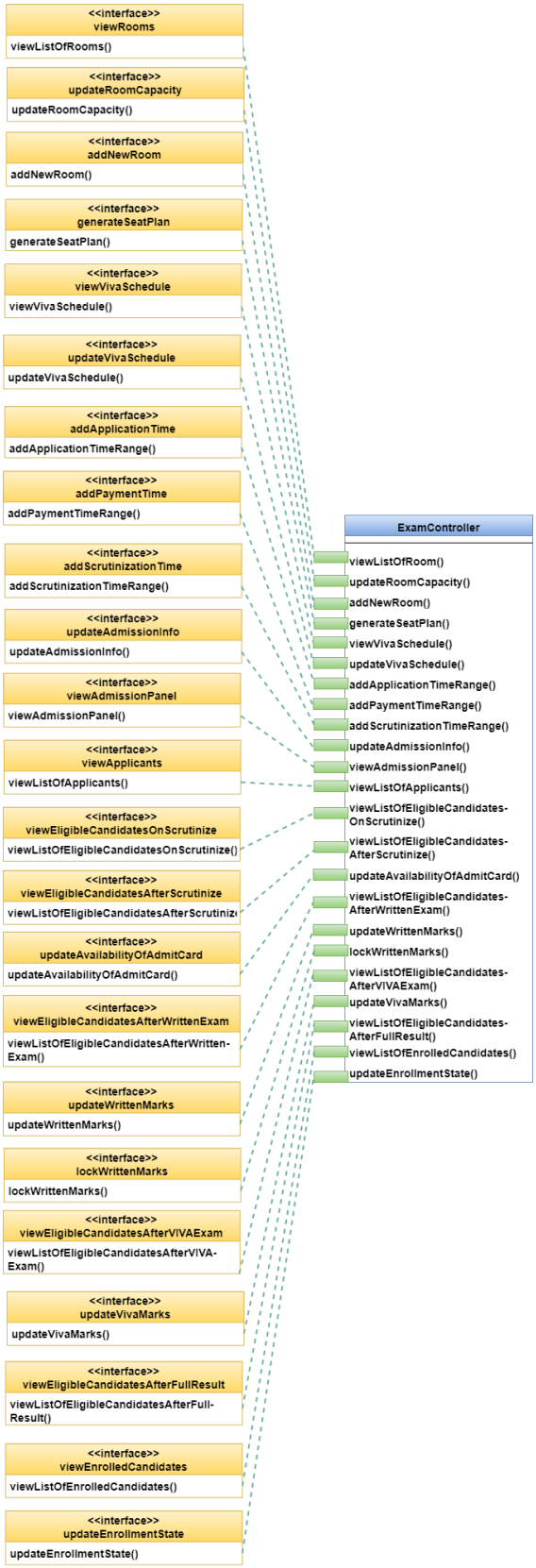
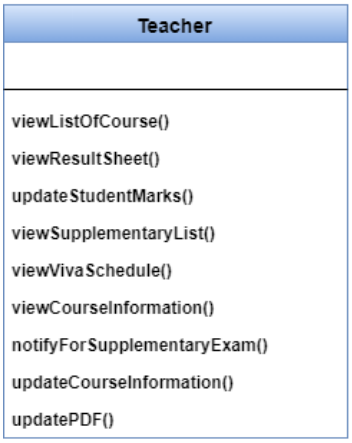
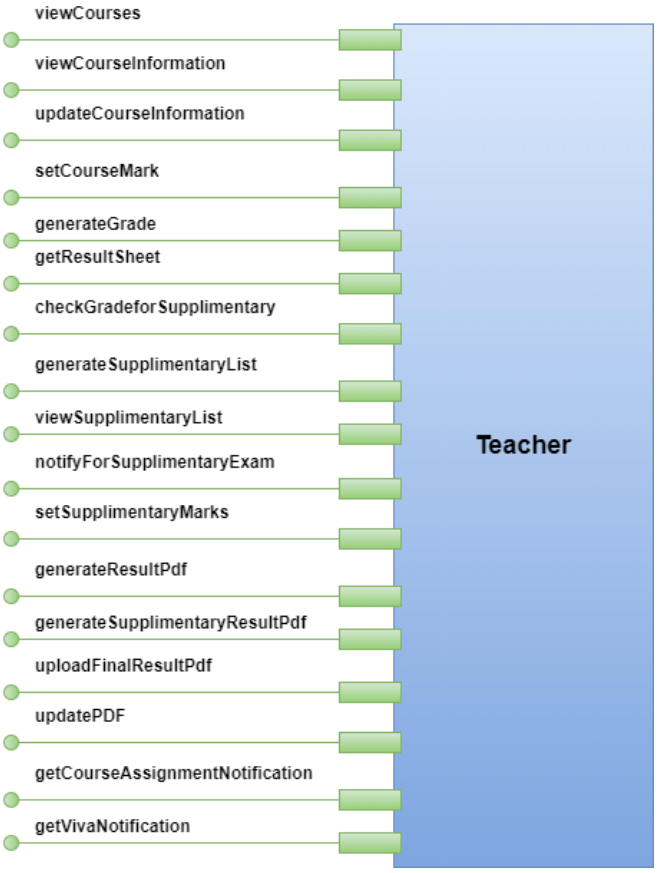


Figure 52 Elaborated ExamController class

Figure 53 shows elaborated Teacher class. It has seventeen interfaces.





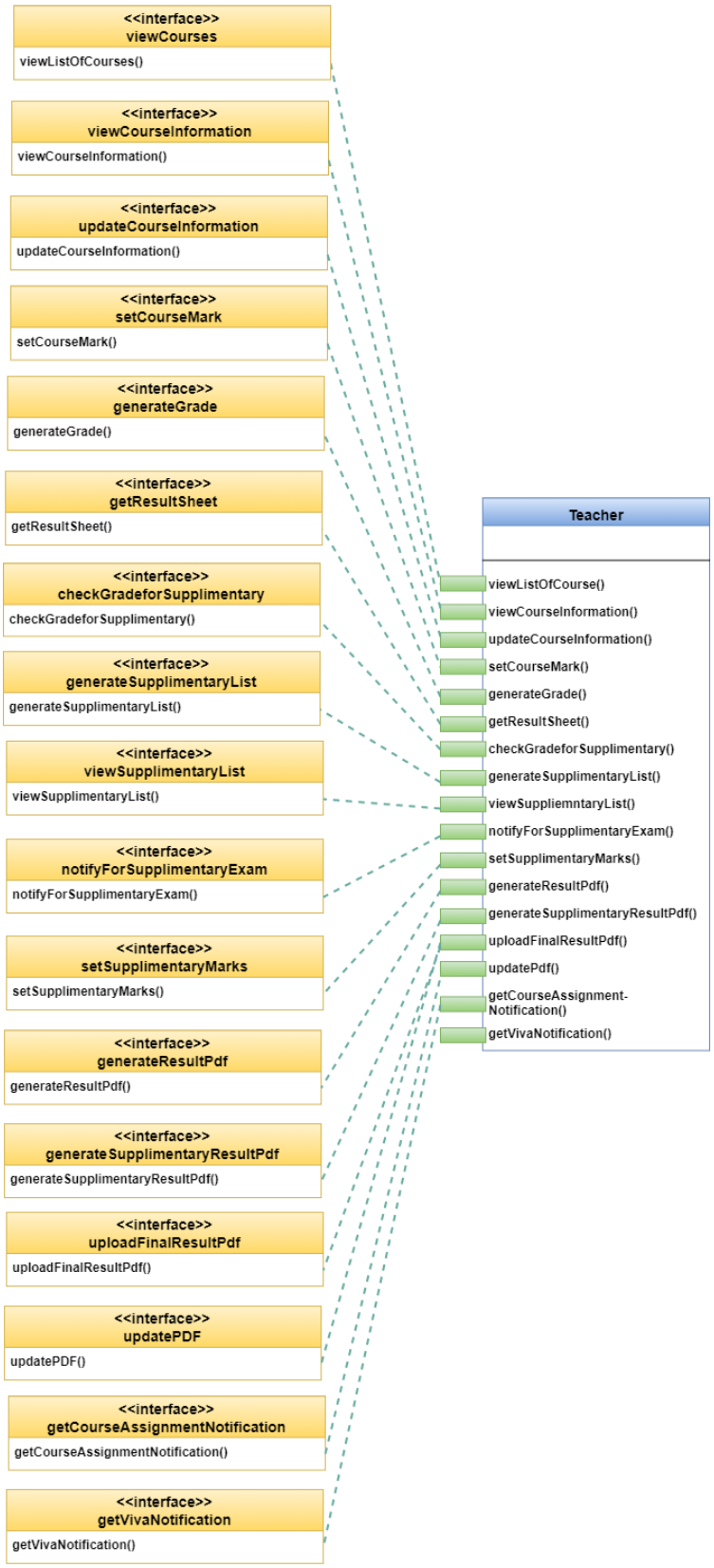


Figure 53 Elaborated Teacher class

Figure 54 shows elaborated Student class. It has four interfaces.

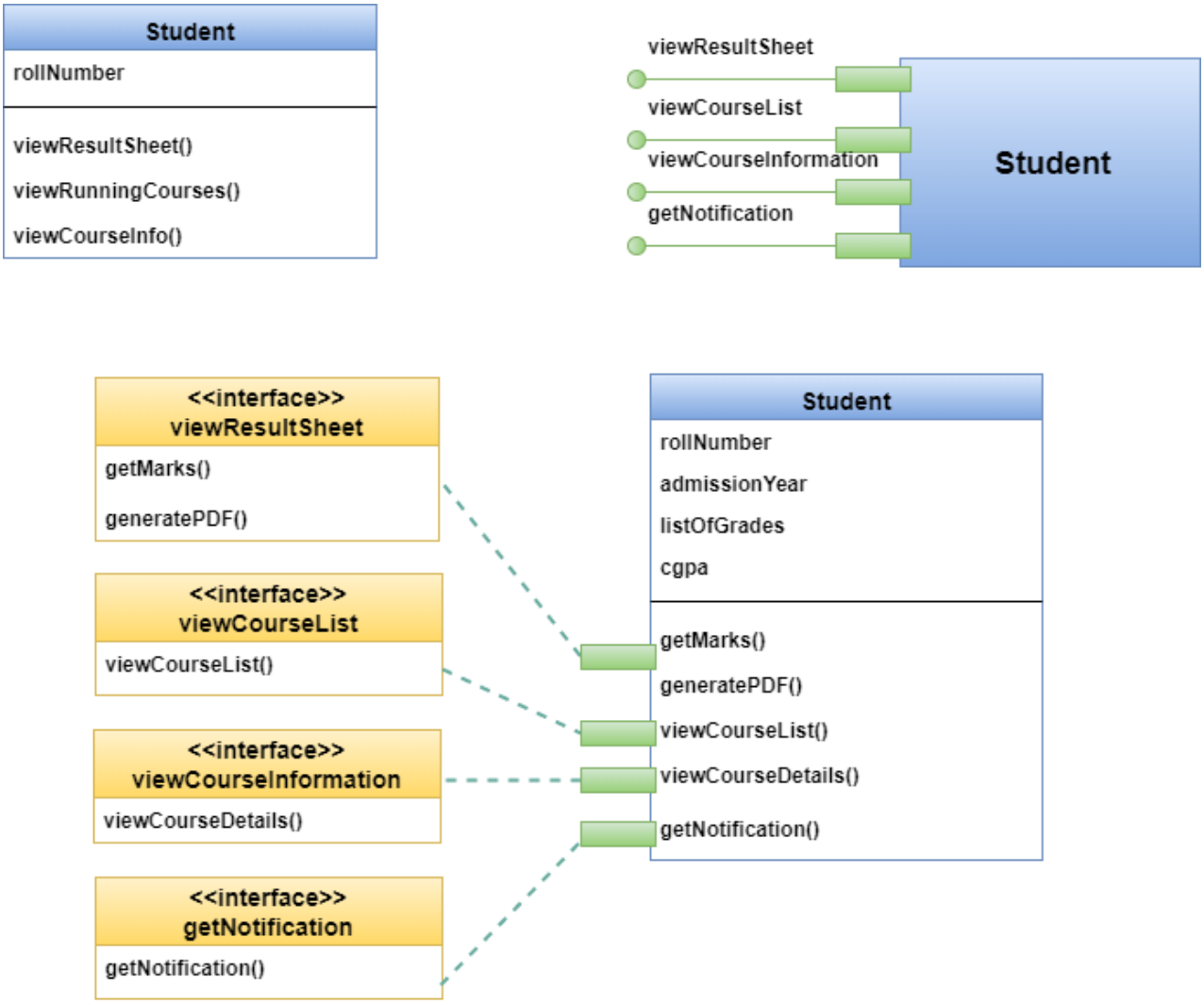


Figure 54 Elaborated Student class

3.3.1 Specify Message Details when Classes or Components Collaborate

The requirements model makes use of a collaboration diagram to show how analysis classes collaborate with one another. As component-level design proceeds, it is sometimes useful to show the details of these collaborations by specifying the structure of messages that are passed between objects within a system. This design activity shows how components within the system communicate and collaborate.

Figure 55-56 shows collaboration diagrams for Applicant class.

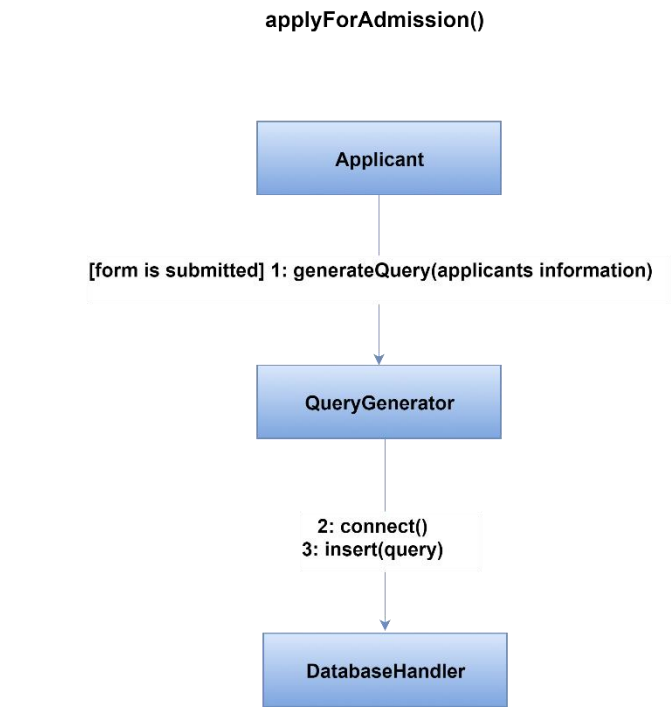


Figure 55 Method applyForAdmission() of Applicant Class

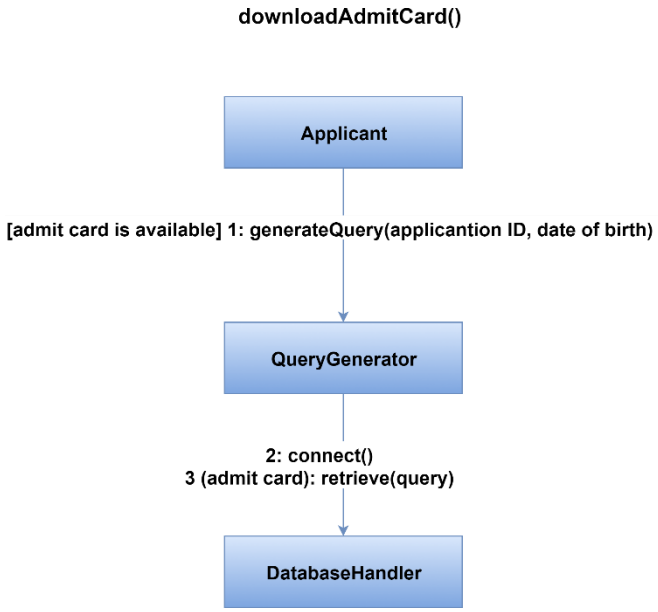


Figure 56 Method downloadAdmitCard() of Applicant Class

Figure 57-62 shows collaboration diagrams for ProgramChairperson class.

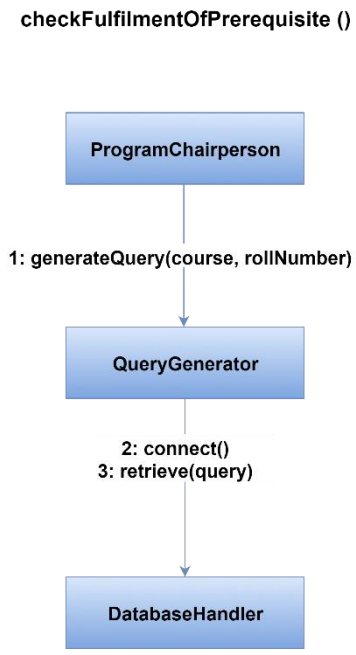


Figure 57 Method checkFulfillmentOfPrerequisites() of ProgramChairperson Class

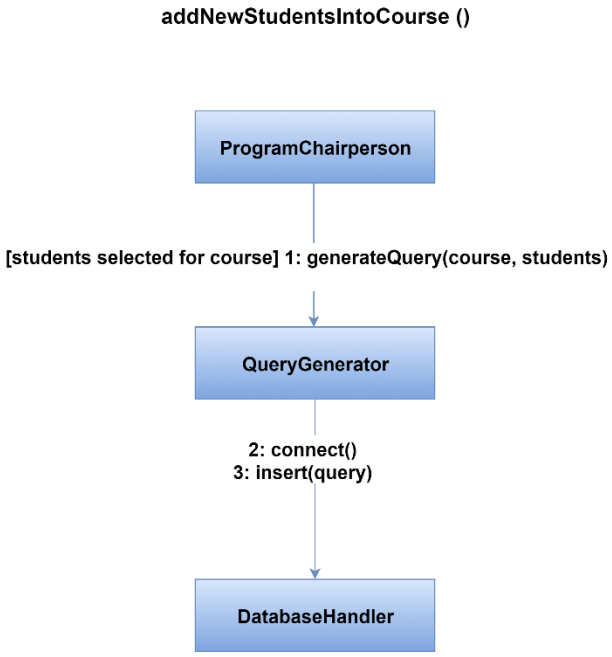


Figure 58 Method addNewStudentsIntoCourse() of ProgramChairperson Class

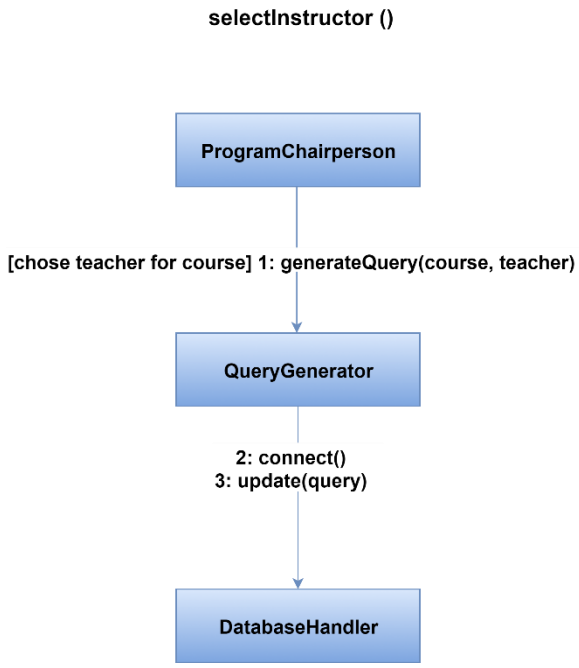


Figure 59 Method selectInstructor() of ProgramChairperson Class

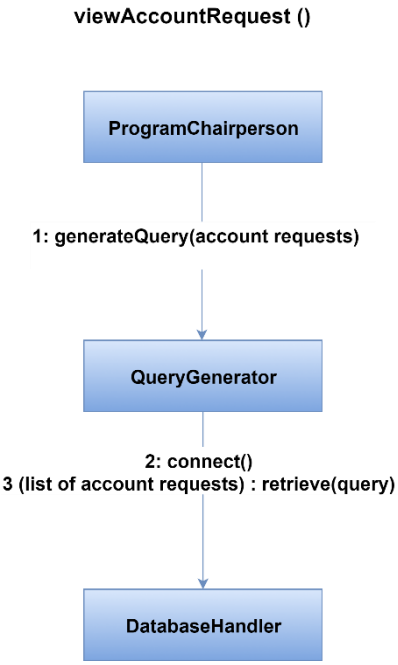


Figure 60 Method viewAccountRequest() of ProgramChairperson Class

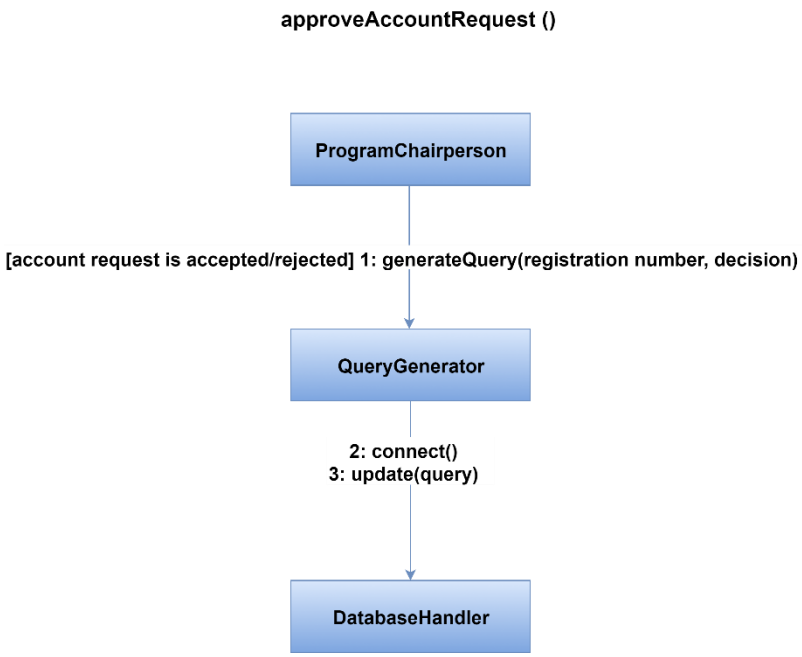


Figure 61 Method approveAccountRequest() of ProgramChairperson Class

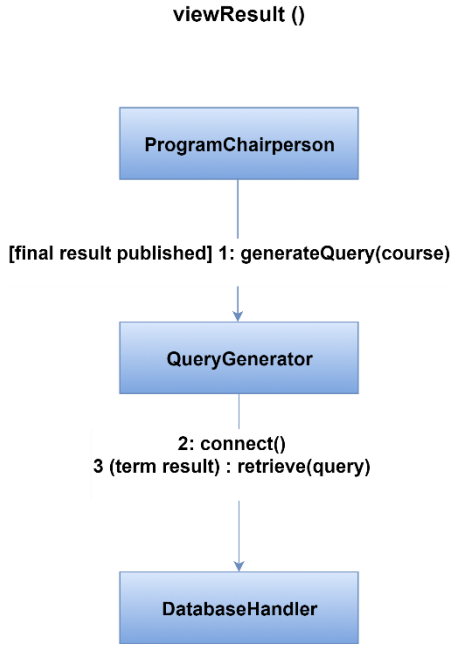


Figure 62 Method viewResult() of ProgramChairperson Class

Figure 63-64 shows collaboration diagrams for Scrutinizer class.

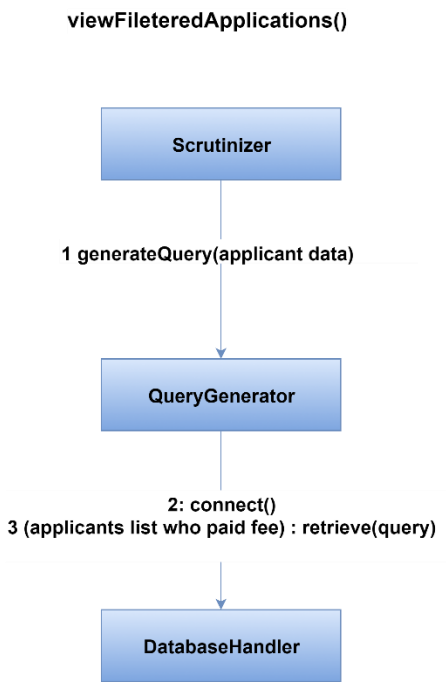


Figure 63 Method viewFilteredApplications() of Scrutinizer Class

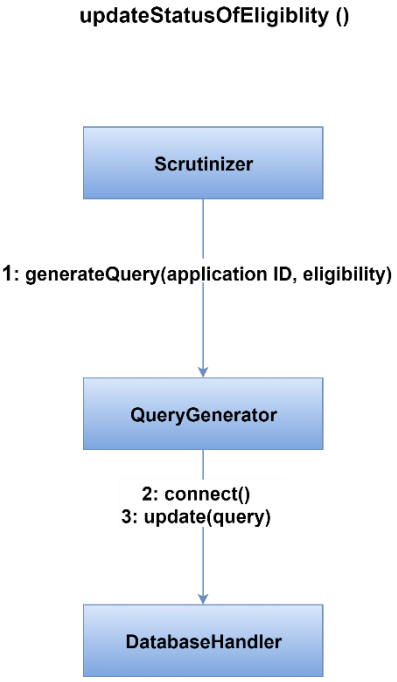


Figure 64 Method updateStatusOfEligibility() of Scrutinizer Class

Figure 65-66 shows collaboration diagrams for Receptionist class.

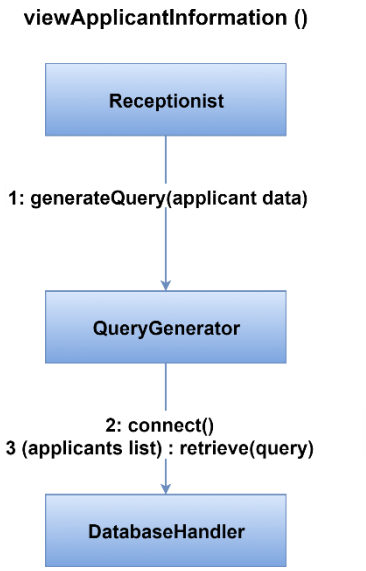


Figure 65 Method viewApplicantInformation() of Receptionist Class

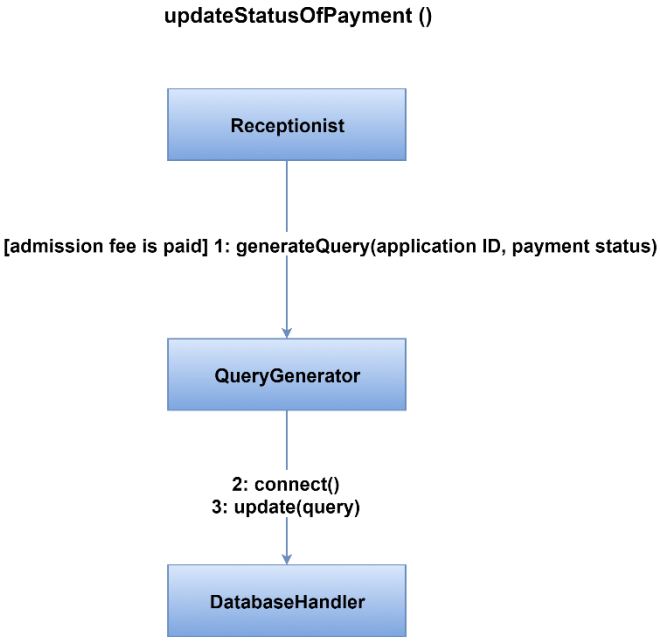


Figure 66 Method updateStatusOfPayment() of Receptionist Class

Figure 67-73 shows collaboration diagrams for ExamController class.

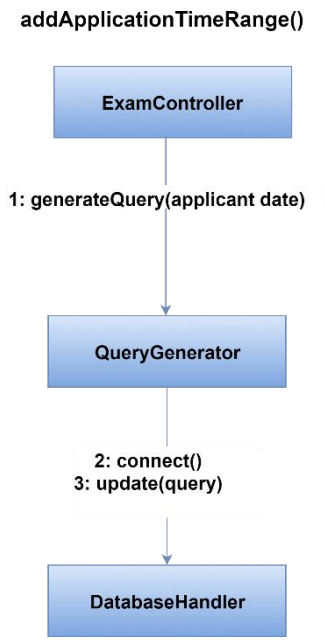


Figure 67 Method addApplicationTimeRange() of ExamController Class

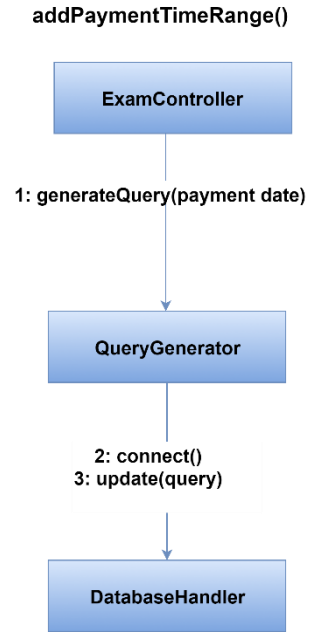


Figure 68 Method addPaymentTimeRange() of ExamController Class

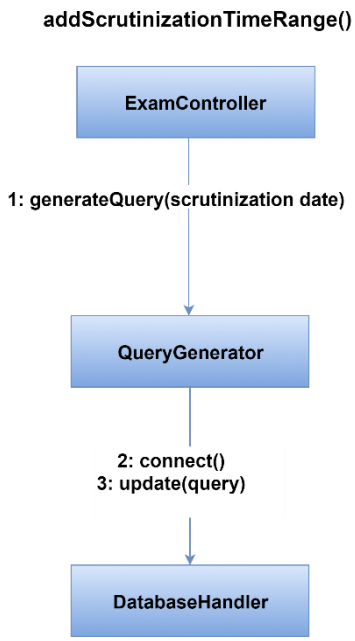


Figure 69 Method addScrutinizationTimeRange() of ExamController Class

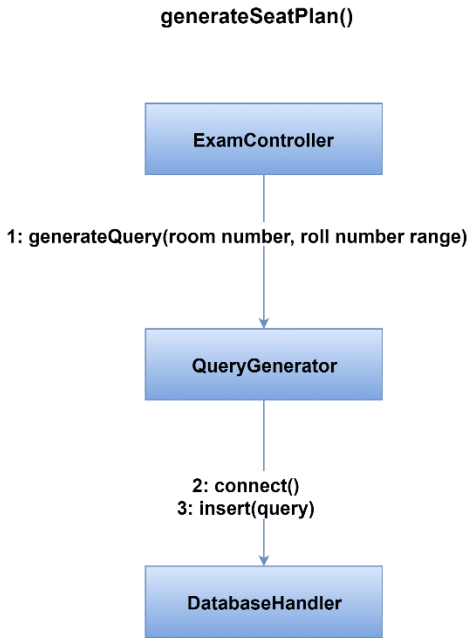


Figure 70 Method generateSeatPlan() of ExamController Class

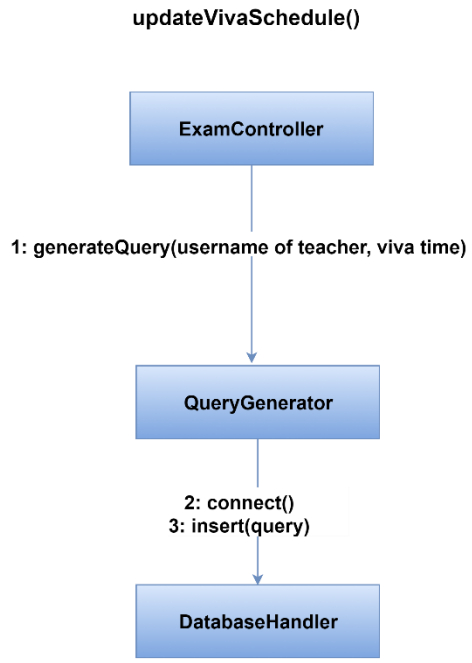


Figure 71 Method updateVivaSchedule() of ExamController Class

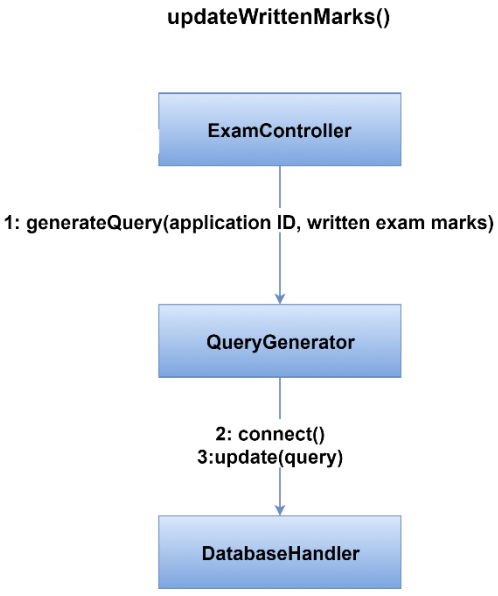


Figure 72 Method updateWrittenMarks() of ExamController Class

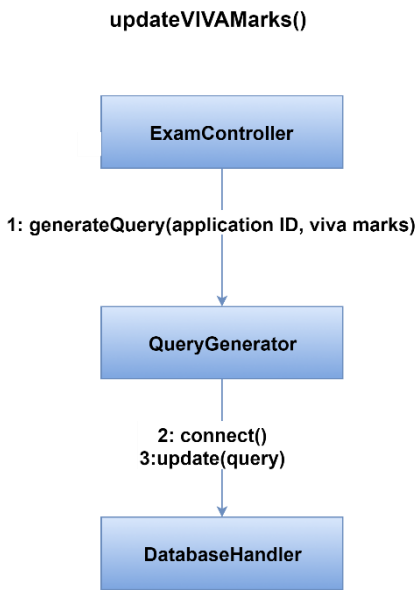


Figure 73 Method updateVIVAMarks() of ExamController Class

Figure 74-86 shows collaboration diagrams for Teacher class.

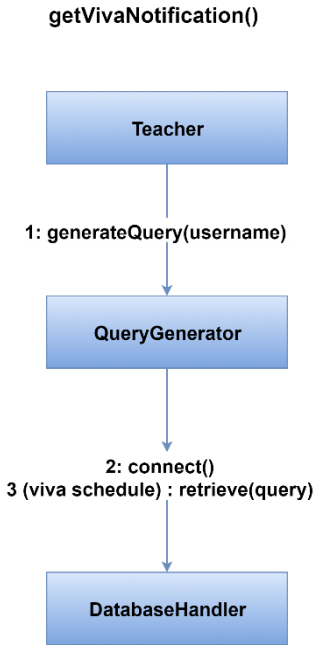


Figure 74 Method getVivaNotification() of Teacher Class

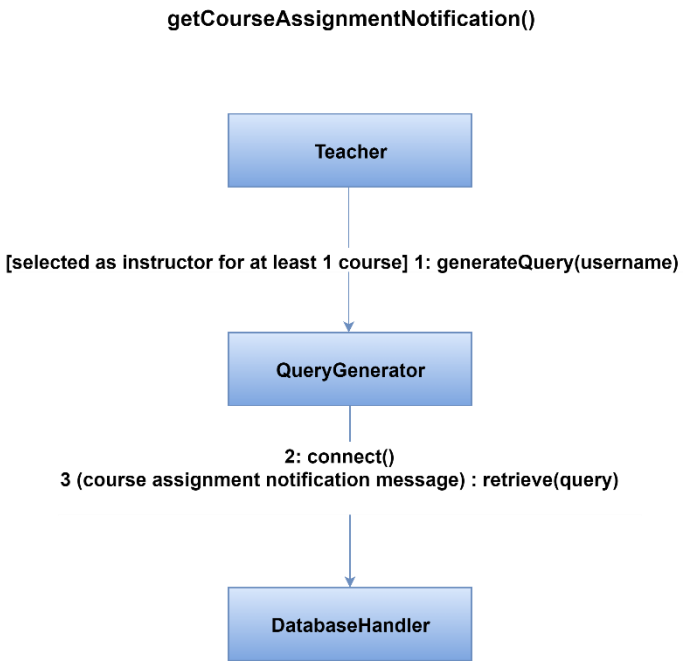


Figure 75 Method getCourseAssignmentNotification() of Teacher Class

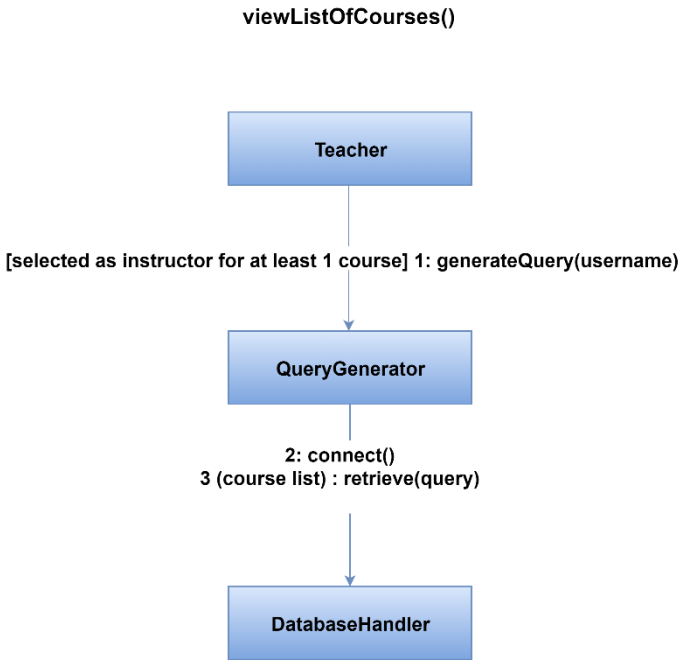


Figure 76 Method viewListOfCourses() of Teacher Class

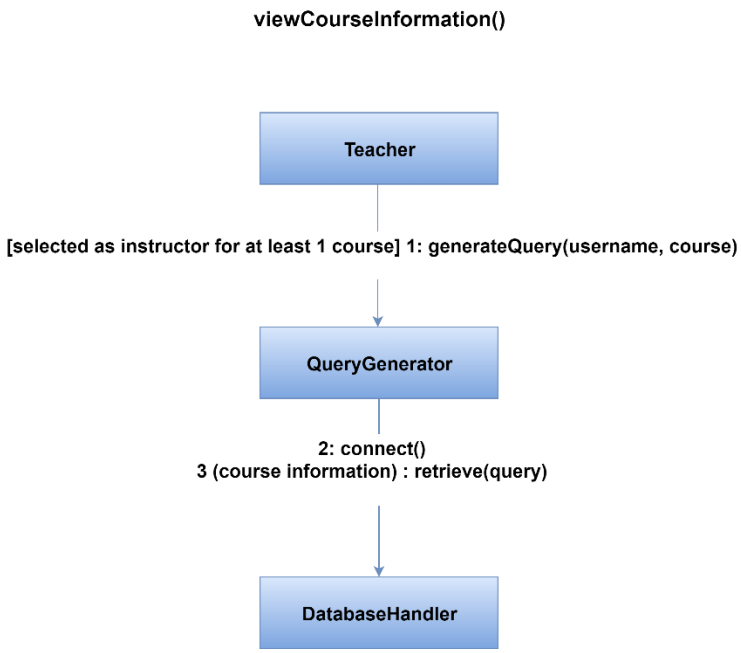


Figure 77 Method viewCourseInformation() of Teacher Class

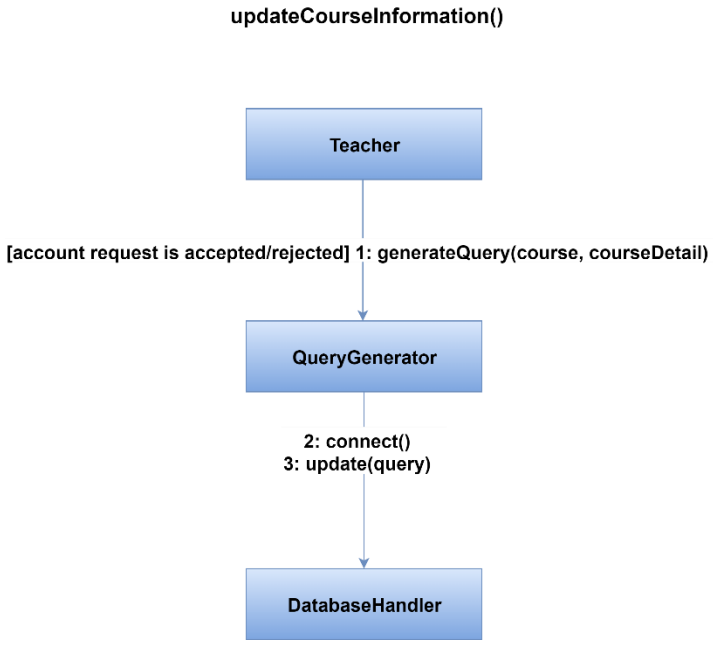


Figure 78 Method updateCourseInformation() of Teacher Class

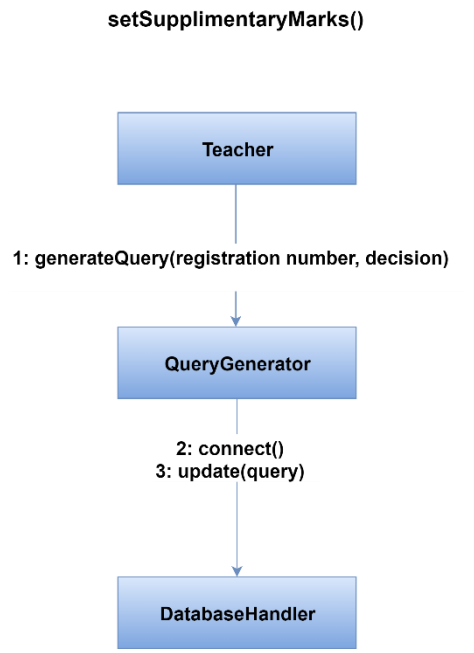


Figure 79 Method setSupplementaryMarks() of Teacher Class

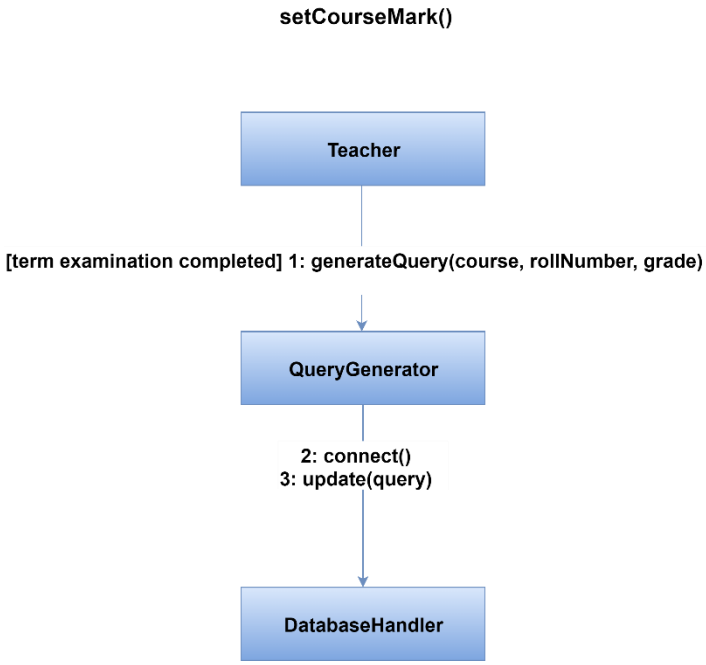


Figure 80 Method setCourseMark() of Teacher Class

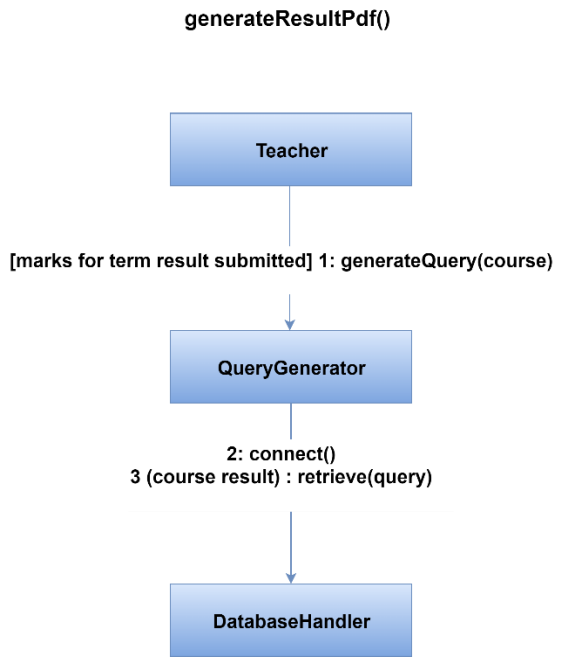


Figure 81 Method generateResultPdf() of Teacher Class

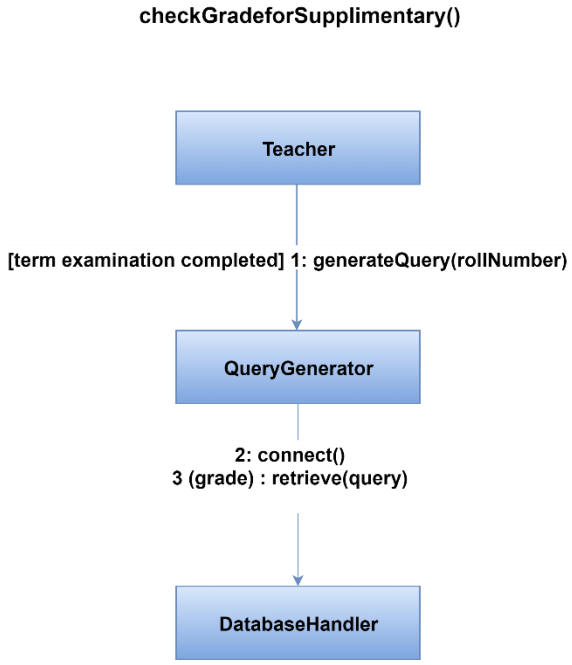


Figure 82 Method checkGradeForSupplementary() of Teacher Class

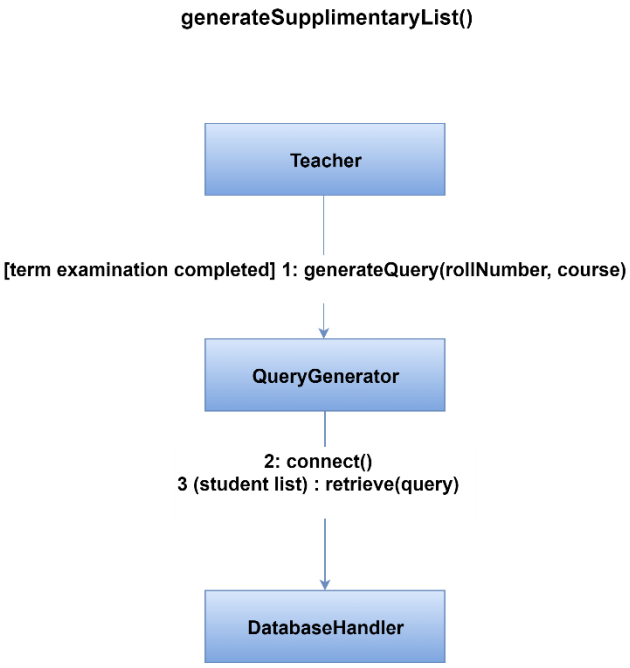


Figure 83 Method generateSupplementaryList() of Teacher Class

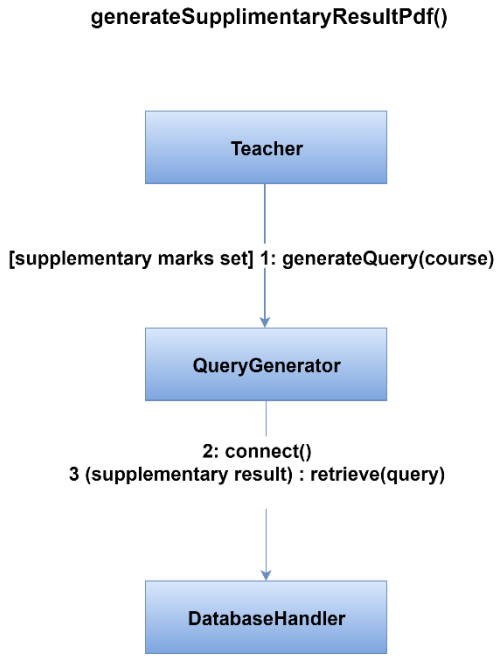


Figure 84 Method generateSupplementaryResultPdf() of Teacher Class

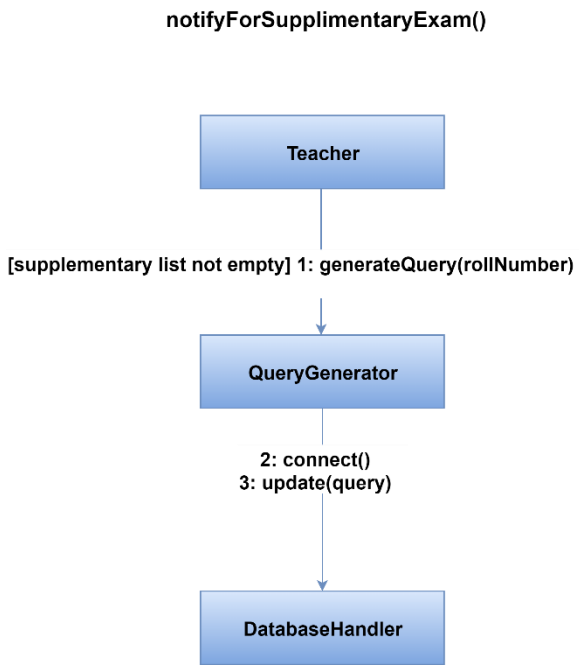


Figure 85 Method notifyForSupplementaryExam() of Teacher Class

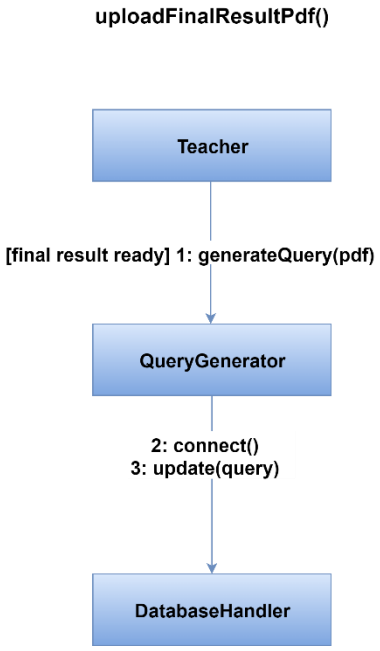


Figure 86 Method uploadFinalResultPdf() of Teacher Class

Figure 87-90 shows collaboration diagrams for Student class.

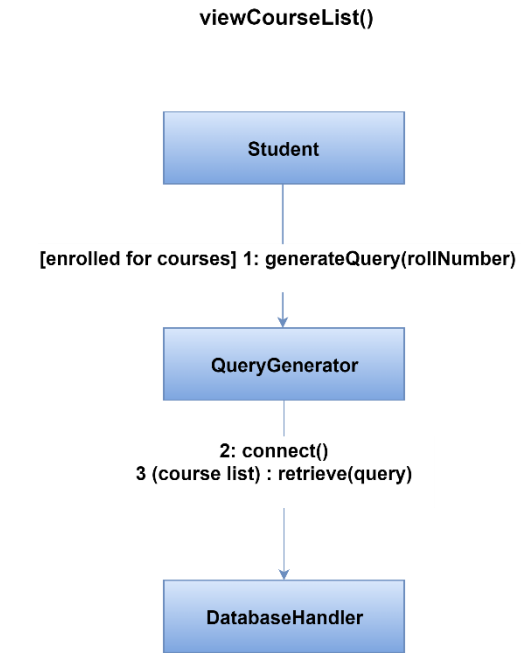


Figure 87 Method viewCourseList() of Student Class

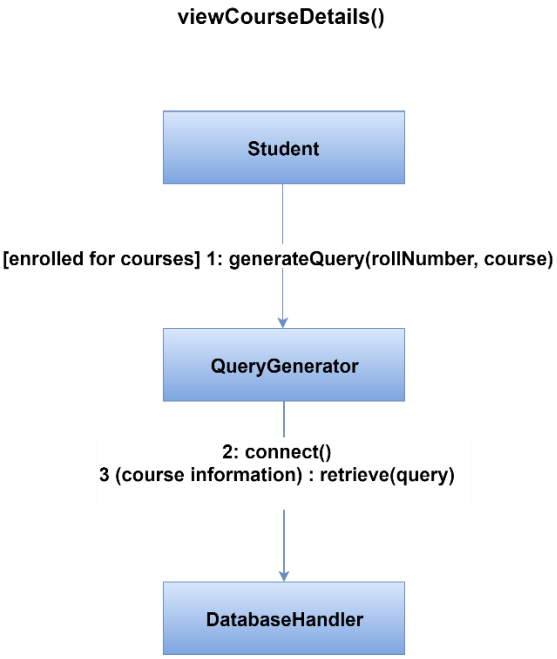


Figure 88 Method viewCourseDetails() of Student Class

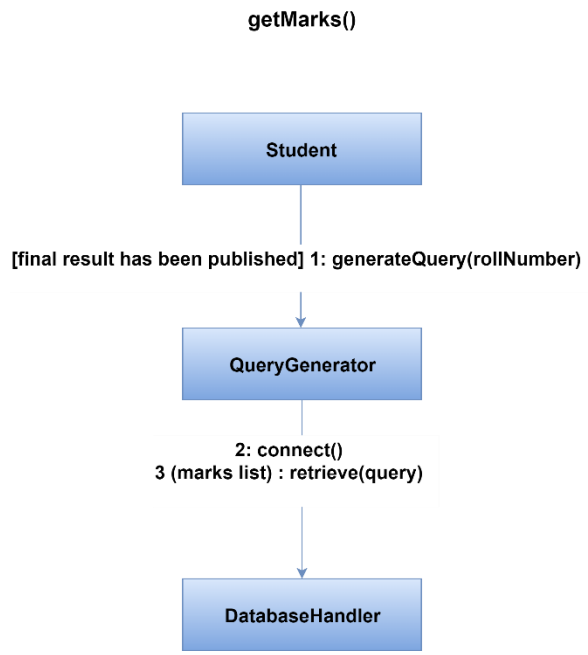


Figure 89 Method getMarks() of Student Class

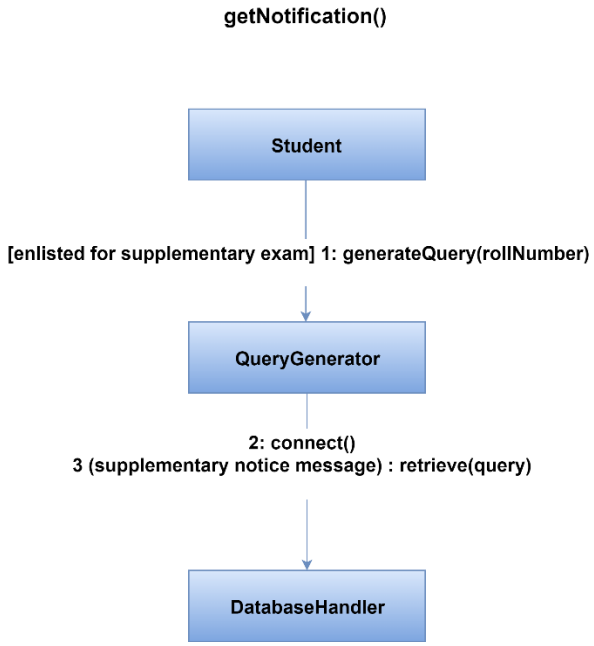


Figure 90 Method getNotification() of Student Class

3.3.2 Identify Appropriate Interfaces for Each Component

Within the context of component-level design, a UML interface is a group of externally visible operations.

User class has some interfaces named verification, forgot password which deal with authentication related activities. But it is not the responsibility of User class to handle

authentication. So we have defined a new class named Authentication which will be responsible for handling authentication. Figure 91 shows Authentication class.

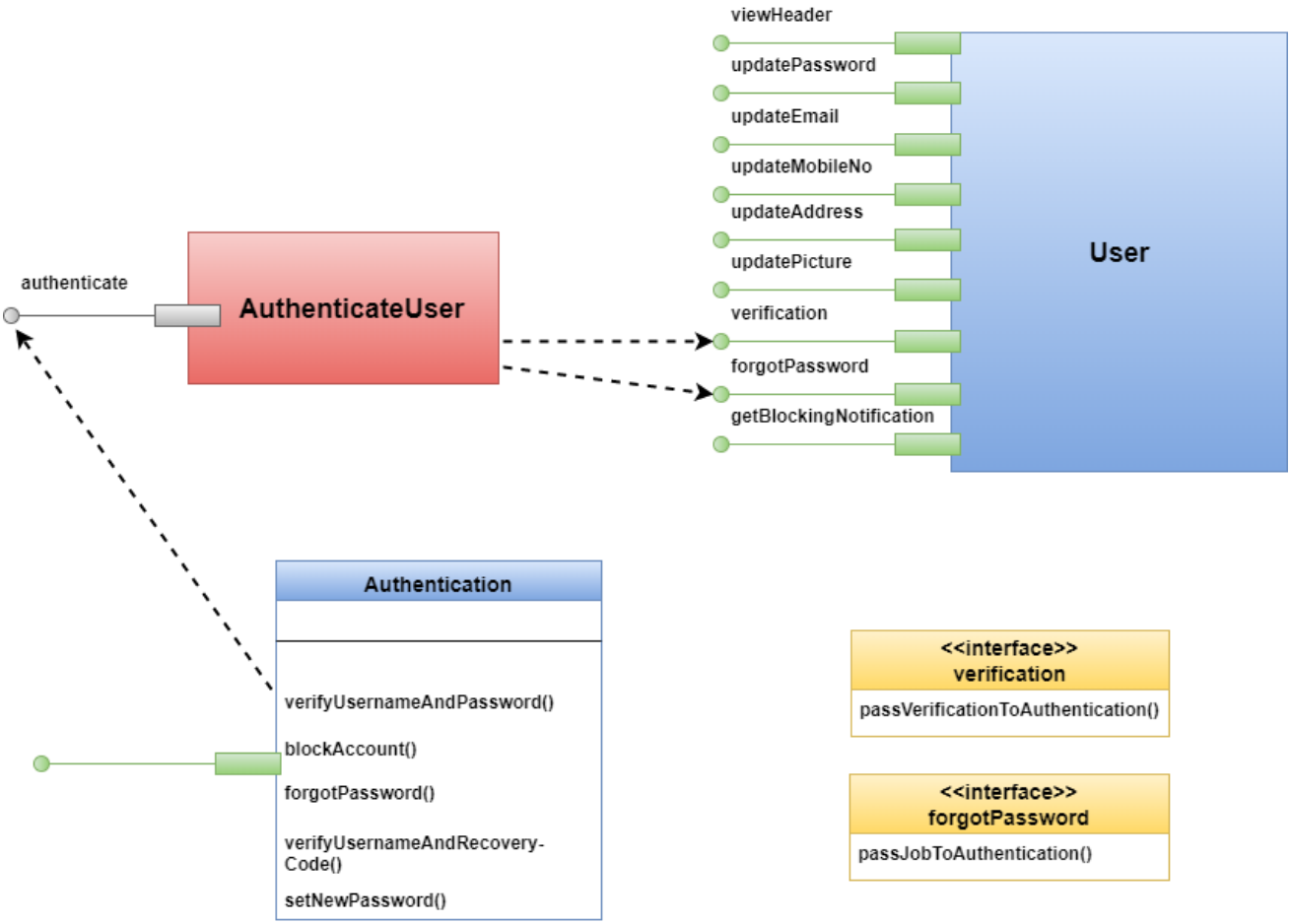


Figure 91 Authentication class

It is not the responsibility of Scrutinizer class to generate roll number for applicants. So a new class RollGenerator has been defined for handling this activity. Figure 92 shows RollGenerator class.

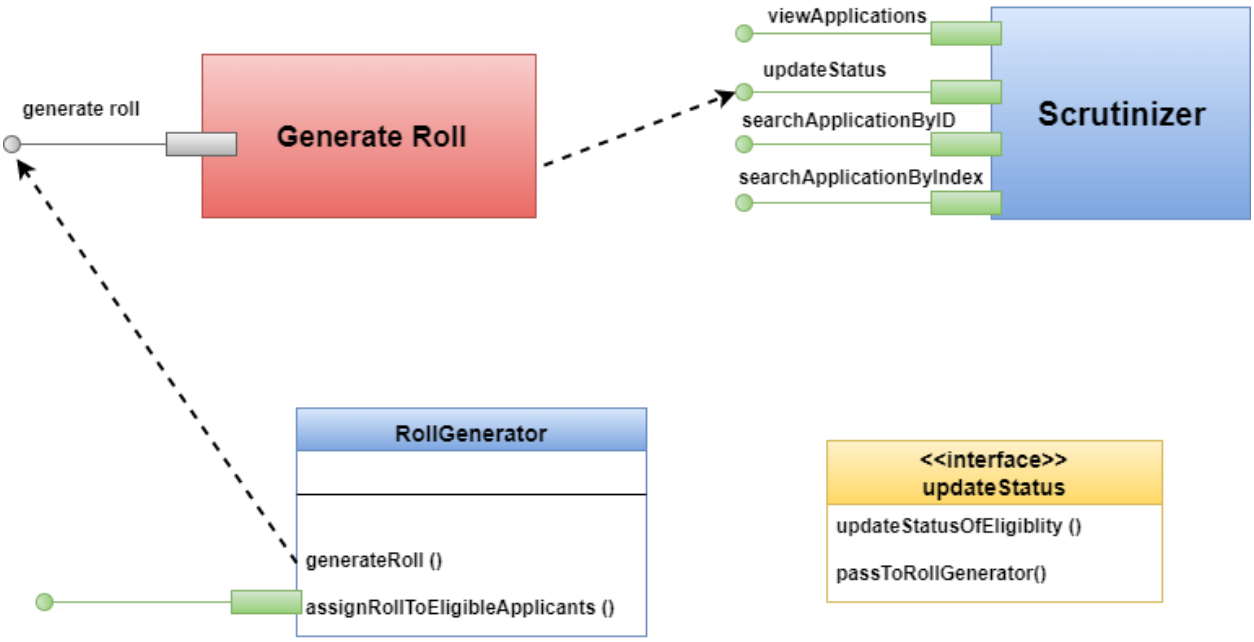


Figure 92 RollGenerator class

Student class has an interface named viewResultSheet which deals with result geenration process. But it is not the responsibility of Student class. So we have defined a new class named ReportCard which will be responsible for generation report card. Figure 93 shows ReportCard class.

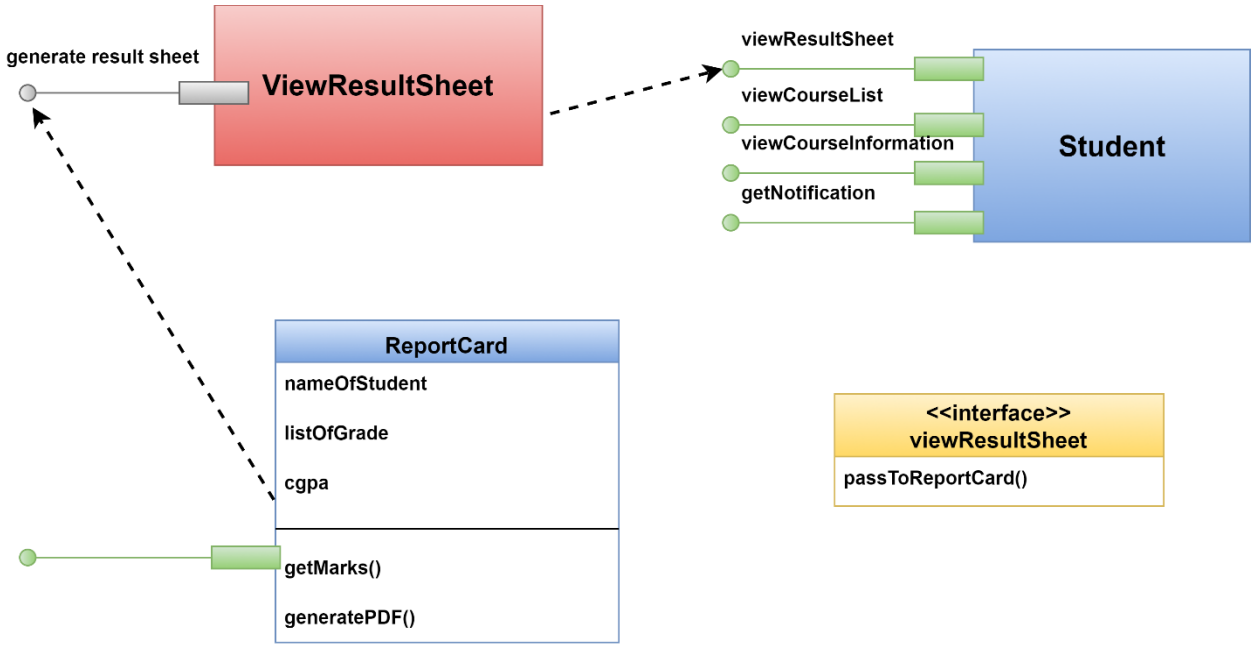


Figure 93 ReportCard class

It is not the responsibility of ExamController class to generate seat plan for applicants. So a new class RoomHandler has been introduced for handling this activity. Figure 94 shows RoomHandler class.

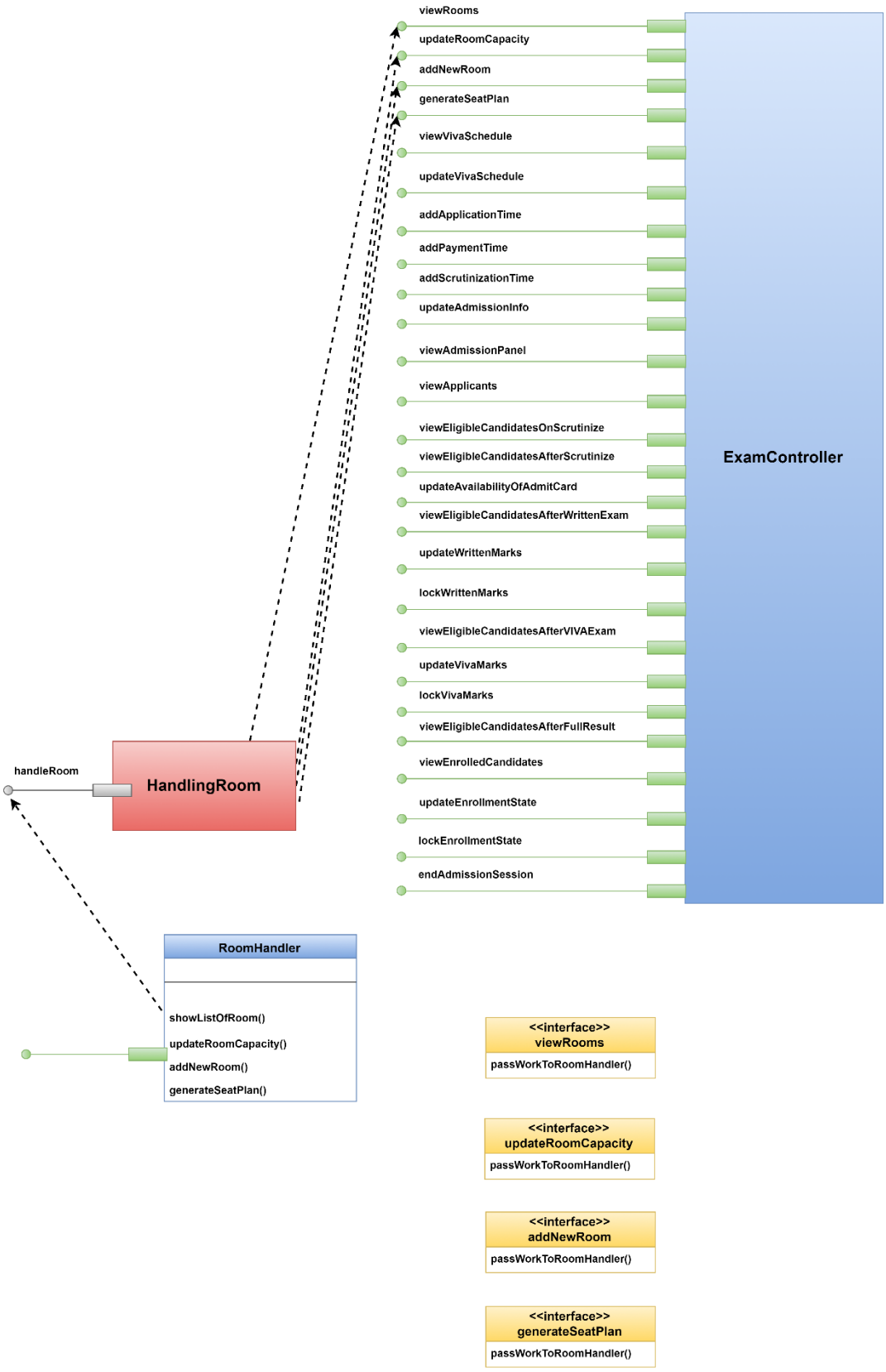


Figure 94 RoomHandler Class

It is not the responsibility of User class to retrieve and display name and image. So a new class DisplayHeader has been introduced for handling this activity. Figure 95 shows DisplayHeader class.

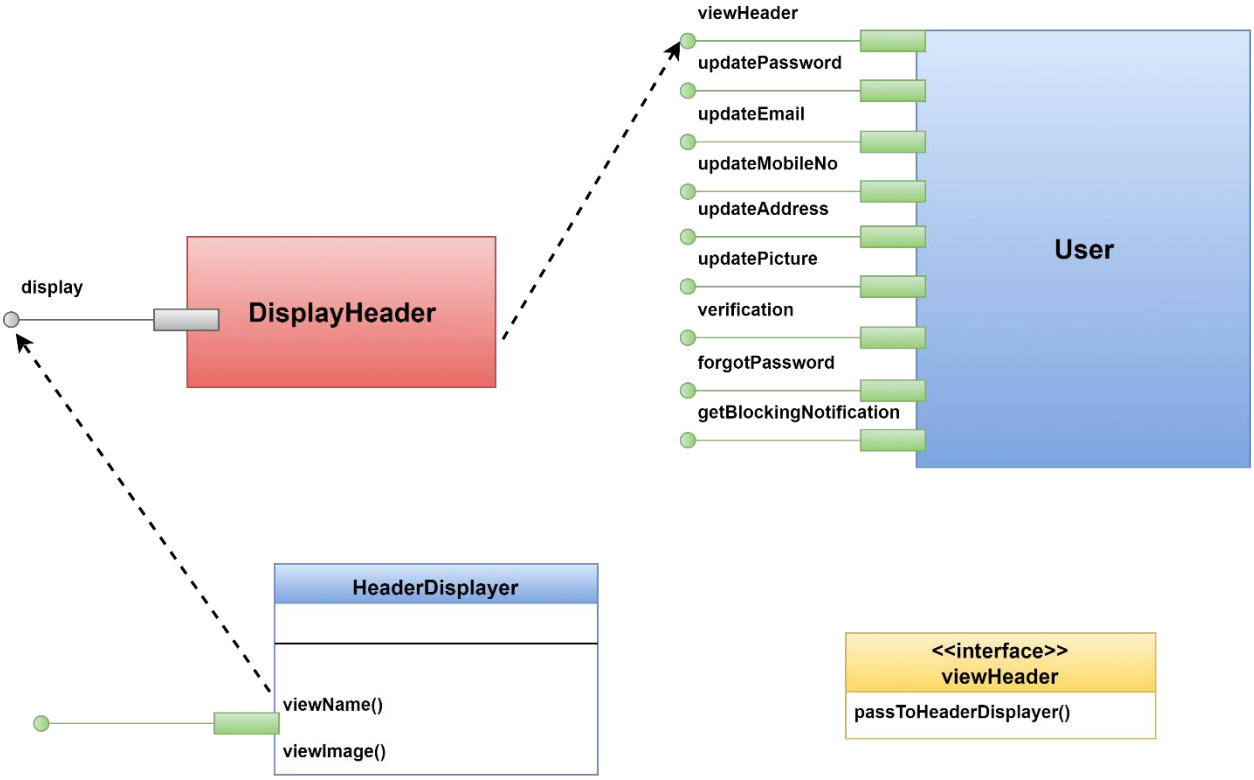


Figure 95 HeaderDisplayer class

It is not the task of Teacher class to generate PDF. So PDFHandler class has been introduced which is shown in figure 96.

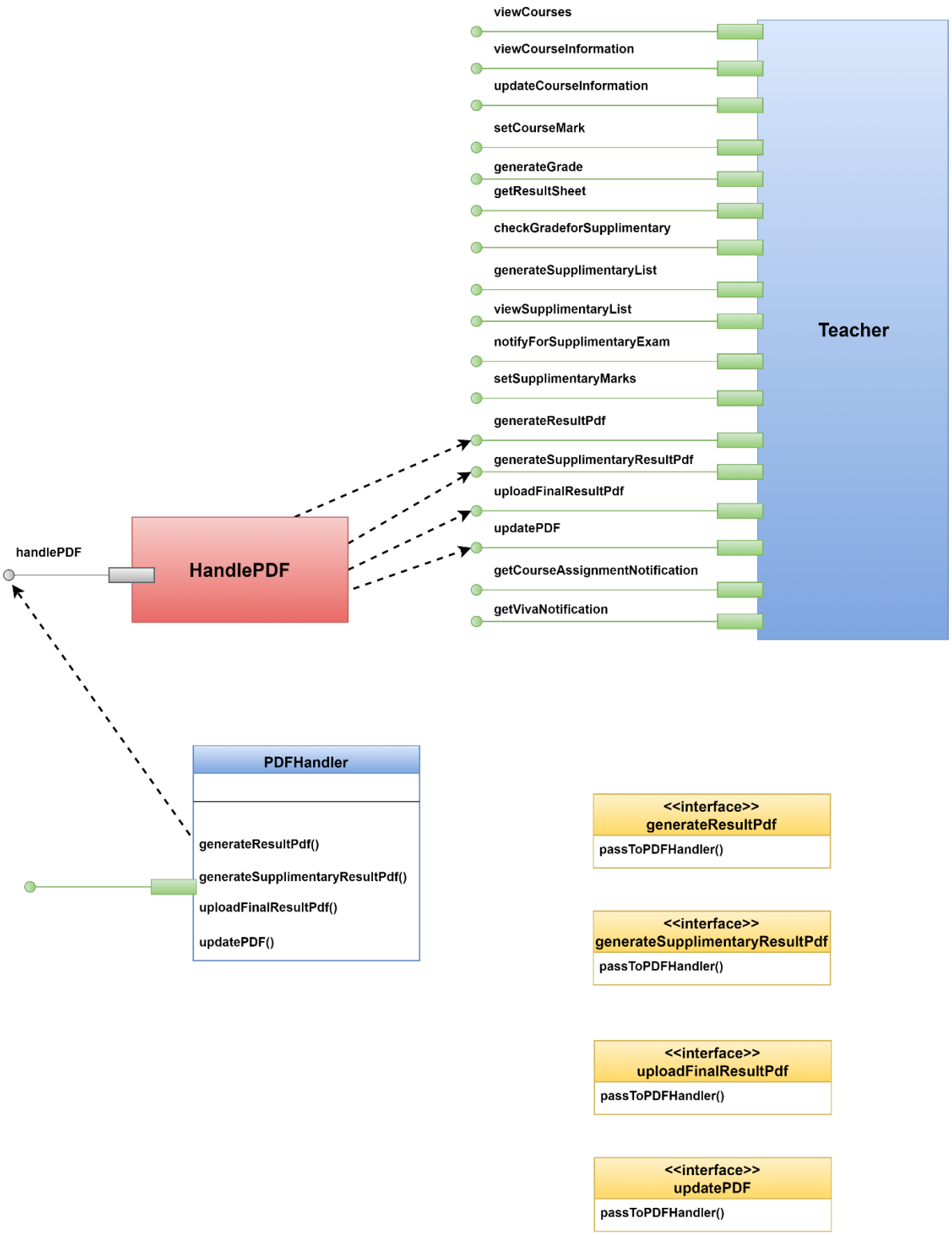


Figure 96 PDFHandler class

3.3.3 Elaborate Attributes and Define Data Types and Data Structures required to Implement Them

Generally data structures and types used to define attributes are defined within the context of the programming language that is to be used for implementation.

Table 1 elaborates all attributes of design classes.

Table 1 Elaborated Attributes of Design classes

No	Attribute Name	Class	Data Type	Initial Value	Property String
1	username	User	String	null	[a-zA-Z]{1,20}
2	password	User	String	null	^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,30}$
3	fullName	User, Applicant, ReportCard	String	null	[a-z A-Z]{1,50}
4	fatherName	User, Applicant	String	null	[a-z A-Z]{1,50}
5	motherName	User, Applicant	String	null	[a-z A-Z]{1,50}
6	mobileNumber	User, Applicant	String	null	[0-9]{11}
7	emailAddress	User, Applicant	String	null	Maximum length = 50 characters
8	address	User, Applicant	String	null	Maximum length = 250 characters
9	rollNumber	Student	int	0	[0-9]{6}
10	year	Student, Applicant	int	2017	[0-9]{4}
11	listOfGrade	ReportCard	Associative array	0.00	0.00 ≤ grade of a course ≤ 4.00
12	cgpa	ReportCard	float	0.00	0.00 ≤ cgpa ≤ 4.00
13	applicationID	Applicant	String	null	[0-9]{6}
14	dateOfBirth	Applicant	Date	Current date	Minimum date 01-01-1990
15	nationality	Applicant	String	null	[a-zA-Z]{1,20}
16	schoolName	Applicant	String	null	[a-zA-Z]{1,50}
17	schoolBoard	Applicant	String	null	[a-zA-Z]{1,50}

18	schoolGroup	Applicant	String	null	[a-zA-Z]{1,50}
19	schoolYear	Applicant	int	2010	[0-9]{4}
20	schoolGPA	Applicant	float	0.00	0.00 ≤ value ≤ 5.00
21	collegeName	Applicant	String	null	[a-zA-Z]{1,50}
22	collegeBoard	Applicant	String	null	[a-zA-Z]{1,50}
23	collegeGroup	Applicant	String	null	[a-zA-Z]{1,50}
24	collegeYear	Applicant	int	2012	[0-9]{4}
25	collegeGPA	Applicant	float	0.00	0.00 ≤ value ≤ 5.00
26	universityName	Applicant	String	null	[a-zA-Z]{1,50}
27	departmentName	Applicant	String	null	[a-zA-Z]{1,50}
28	universityYear	Applicant	int	2016	[0-9]{4}
29	universityGPA	Applicant	float	0.00	0.00 ≤ value ≤ 4.00
30	algebraInHSC	Applicant	boolean	false	Either true or false
31	algebraIn Bachelor	Applicant	boolean	false	Either true or false
32	mathInHSC	Applicant	boolean	false	Either true or false
33	mathPercentage InHSC	Applicant	float	0.00	0.00 ≤ value ≤ 100.00
34	mathInBachelor	Applicant	boolean	false	Either true or false
35	mathPercentage InBachelor	Applicant	float	0.00	0.00 ≤ value ≤ 100.00
36	eligibilityForViva	Applicant	boolean	false	Either true or false
37	enrollmentStatus	Applicant	boolean	false	Either true or false
38	applicationFee PaymentStatus	Applicant	boolean	false	Either true or false
39	eligibilityOf Application	Applicant	boolean	false	Either true or false
40	roomNo	Applicant	int	0	[0-9]{3}
41	writtenMarks	Applicant	float	0.00	0.00 ≤ value ≤ 100.00

42	eligibilityForVIVA	Applicant	boolean	false	Either true or false
43	VIVAmarks	Applicant	float	0.00	0.00 ≤ value ≤ 100.00
44	eligibilityFor Admission	Applicant	boolean	false	Either true or false
45	admissionFee PaymentStatus	Applicant	boolean	false	Either true or false
46	enrollmentStatus	Applicant	boolean	false	Either true or false

3.3.4 Describe Processing Flow within Each Operation in Detail

This may be accomplished using UML activity diagram.

Figure 97-100 show UML activity diagrams for Authentication class.

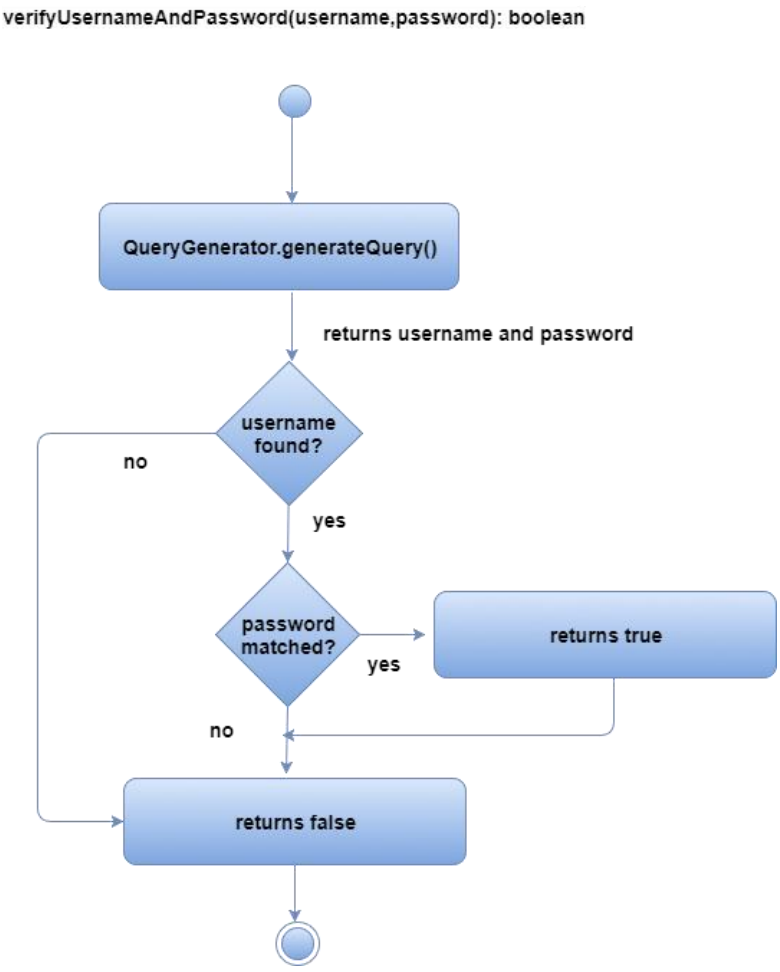


Figure 97 verifyUsernameAndPassword() of Authentication

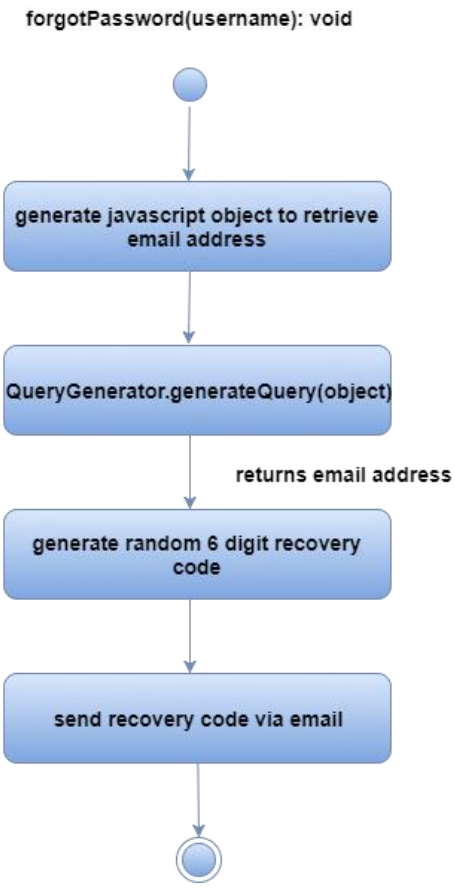


Figure 98 forgotPassword() of Authentication

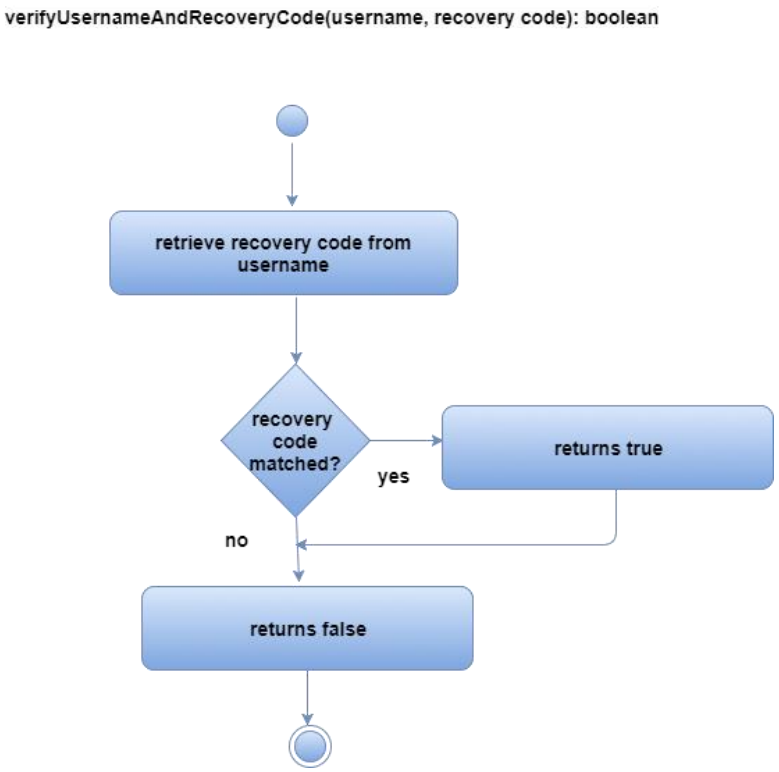


Figure 99 verifyUsernameAndRecoveryCode() of Authentication

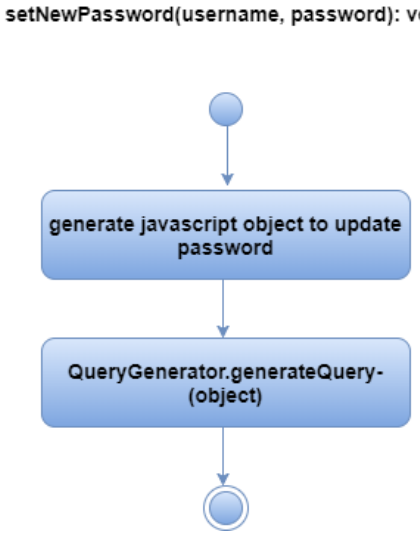


Figure 100 setNewPassword() of Authentication

Figure 101-103 show UML activity diagrams for Applicant class.

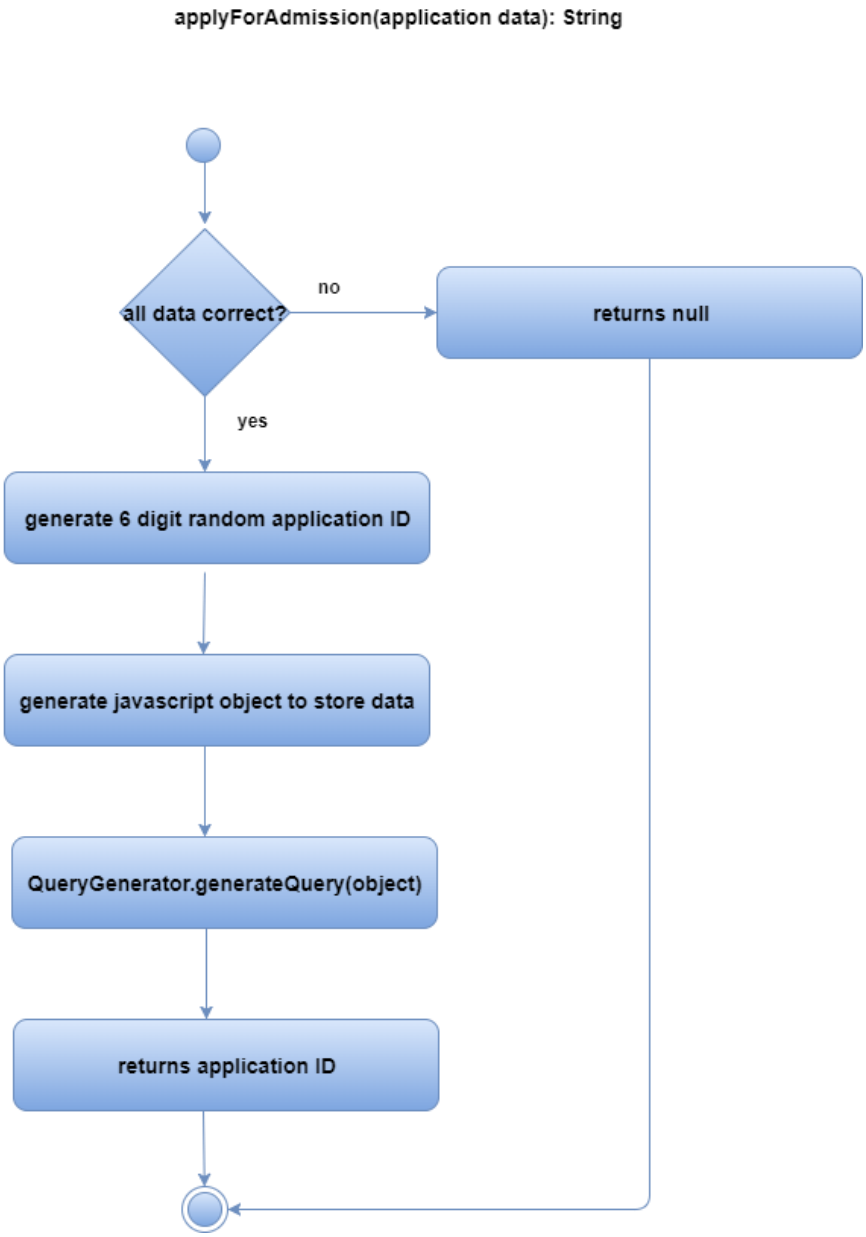


Figure 101 applyForAdmission() of Applicant

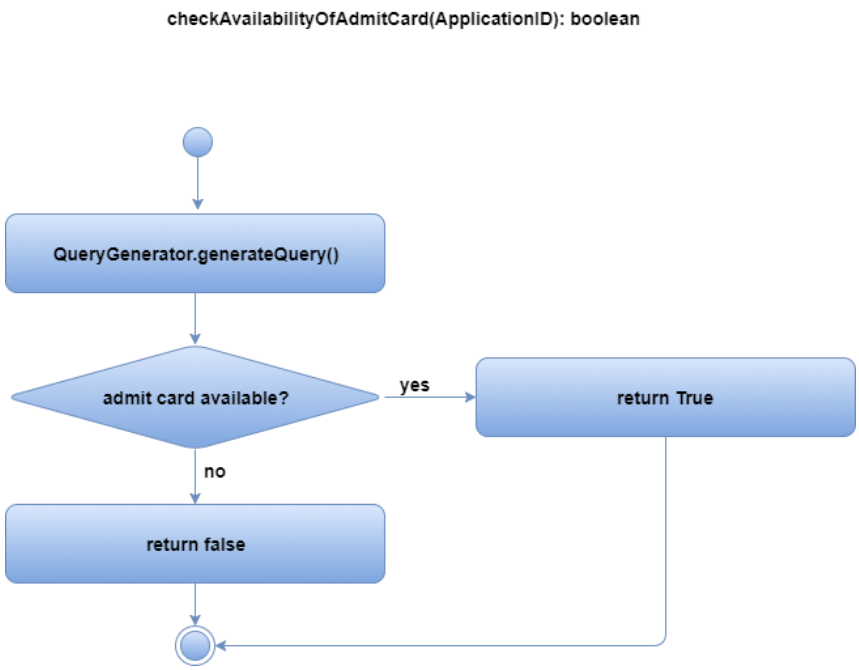


Figure 102 checkAvailabilityOfAdmitCard() of Applicant

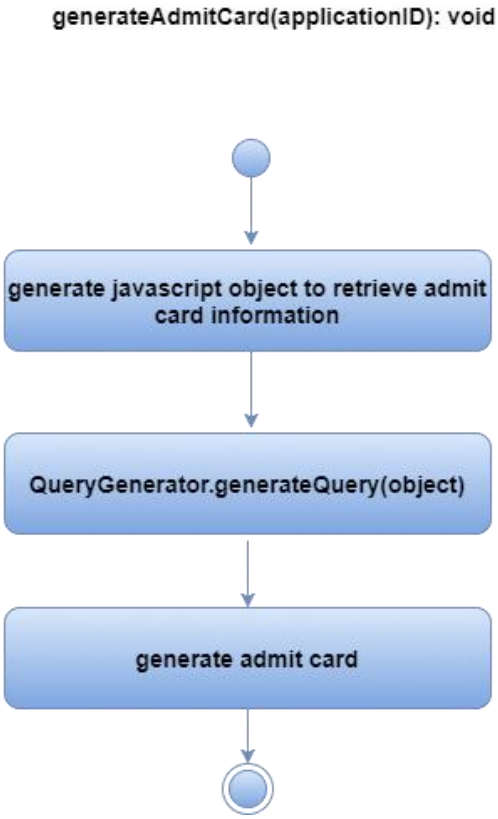


Figure 103 generateAdmitCard() of Applicant

Figure 104 shows UML activity diagram for QueryGenerator class.

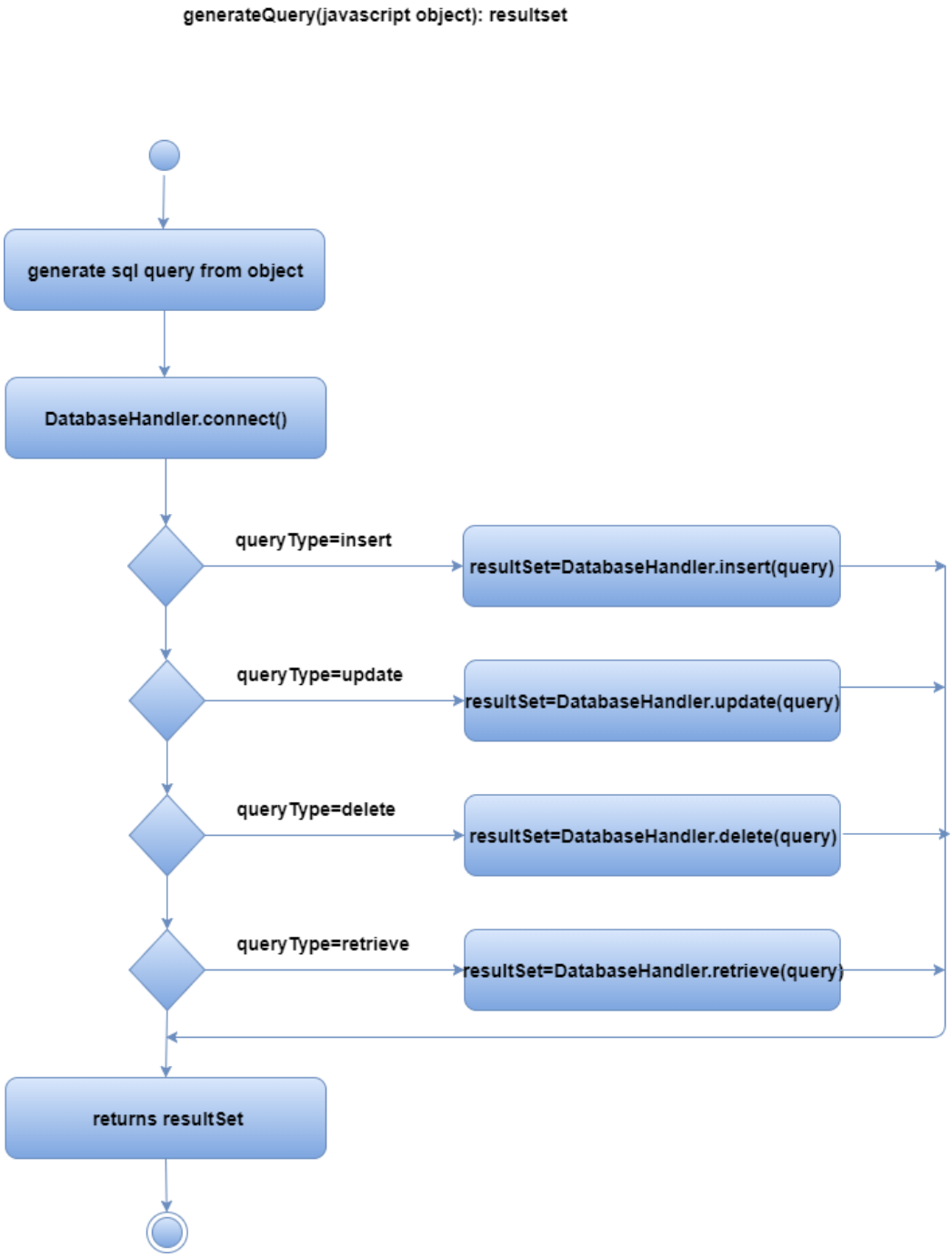


Figure 104 generateQuery() of QueryGenerator

Figure 105-108 show UML activity diagrams for User class.

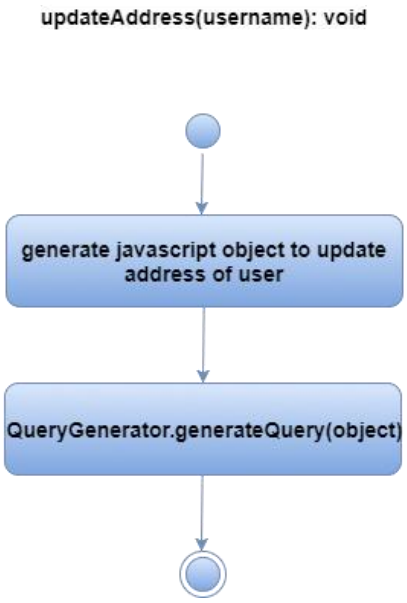


Figure 105 updateAddress() of User

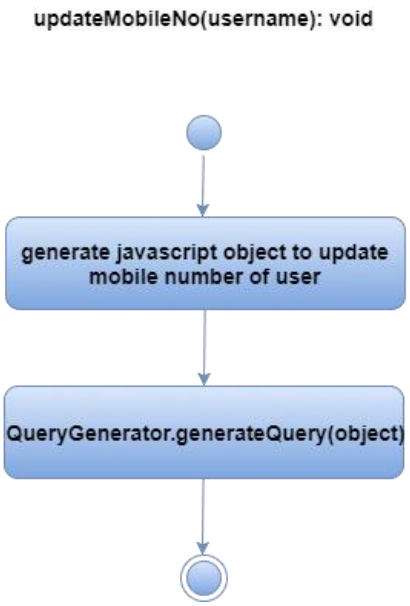


Figure 106 updateMobileNo() of User

updatePicture(username): void

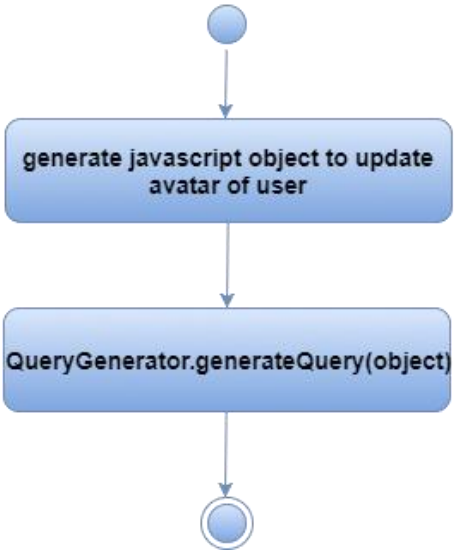


Figure 107 updatePicture() of User

getBlockingNotification(): notificationMessage

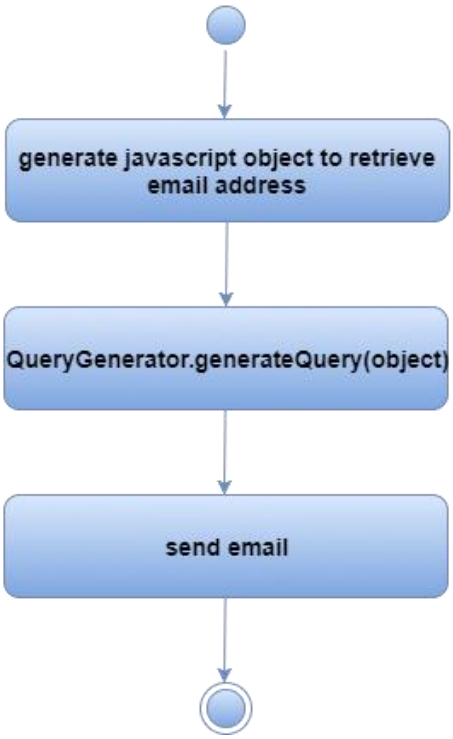


Figure 108 getBlockingNotification() of User

Figure 109-110 show UML activity diagrams for HeaderDisplayer class.

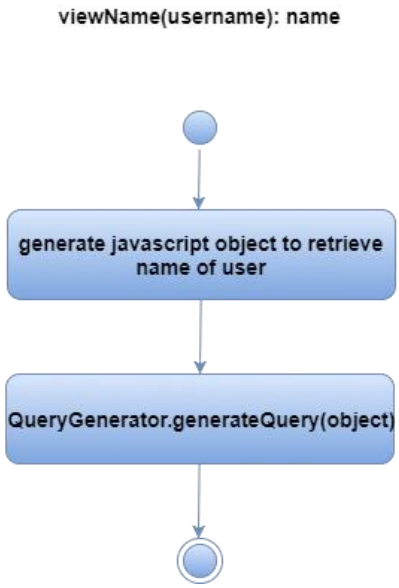


Figure 109 viewName() of HeaderDisplayer

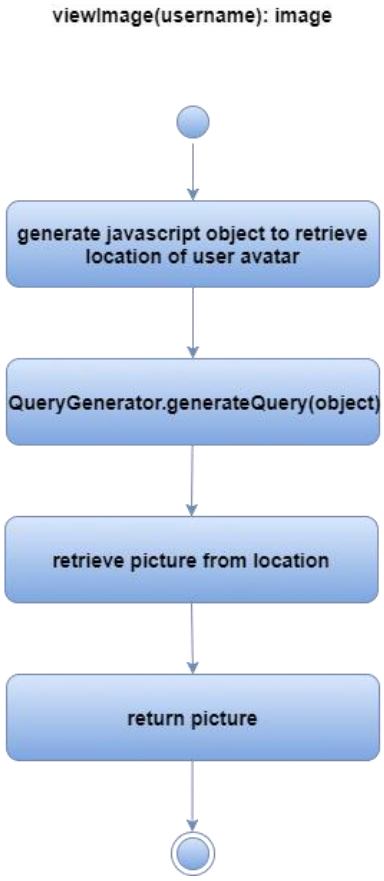


Figure 110 viewImage() of HeaderDisplayer

Figure 111-121 show UML activity diagrams for ProgramChairperson class.

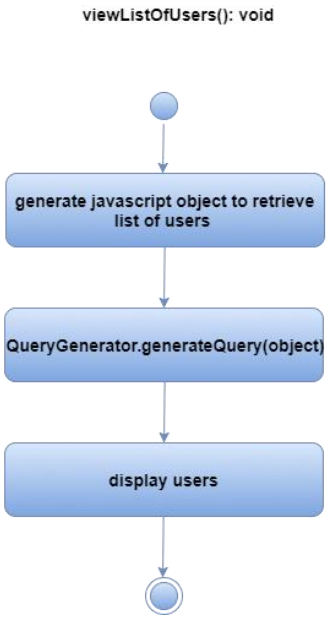


Figure 111 viewListOfUsers() of ProgramChairperson

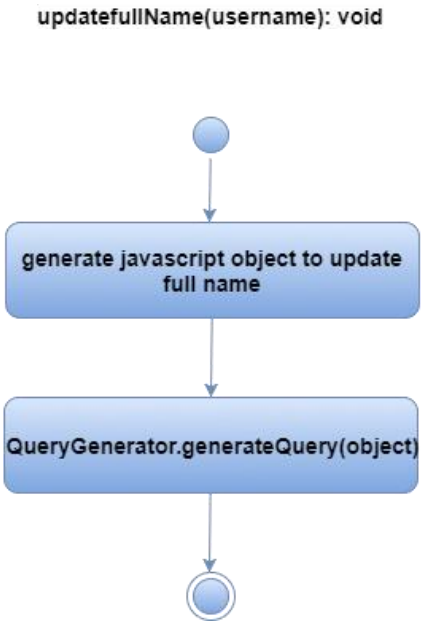


Figure 112 updateFullName() of ProgramChairperson

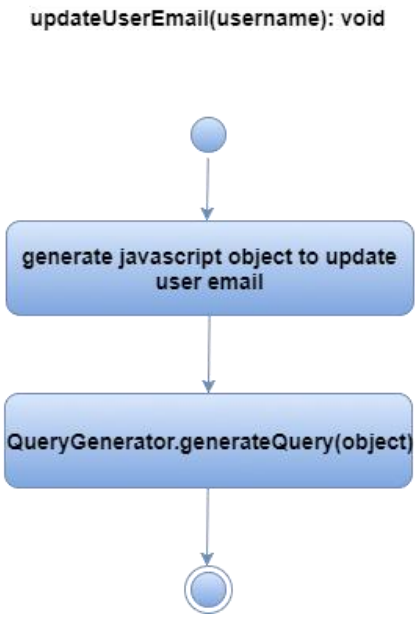


Figure 113 updateUserEmail () of ProgramChairperson

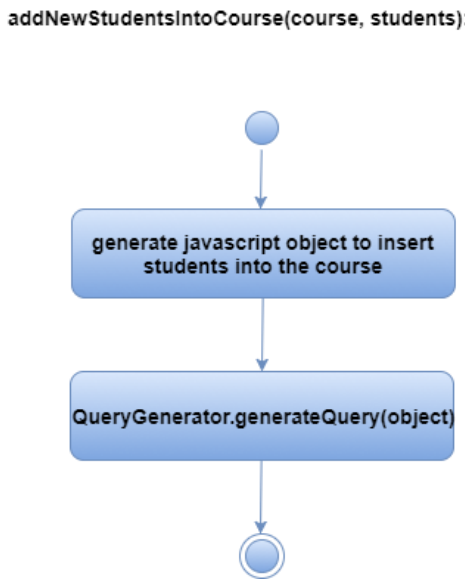


Figure 114 addNewStudentsIntoCourse() of ProgramChairperson

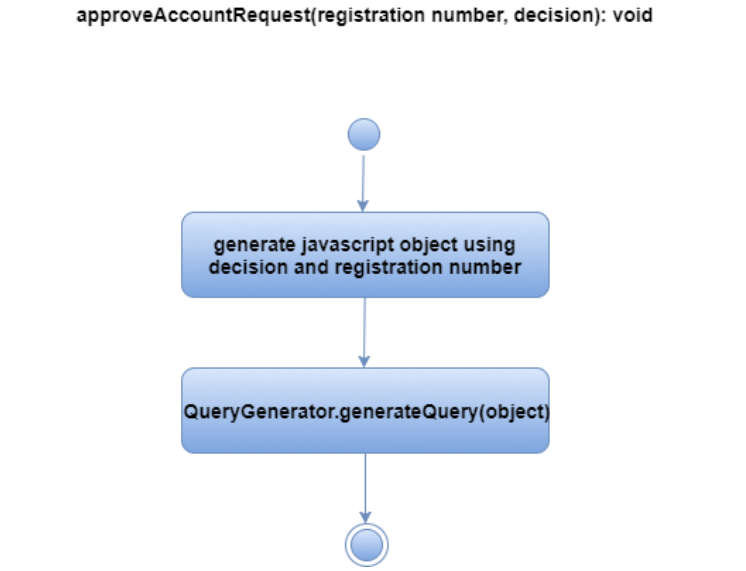


Figure 115 approveAccountRequest() of ProgramChairperson

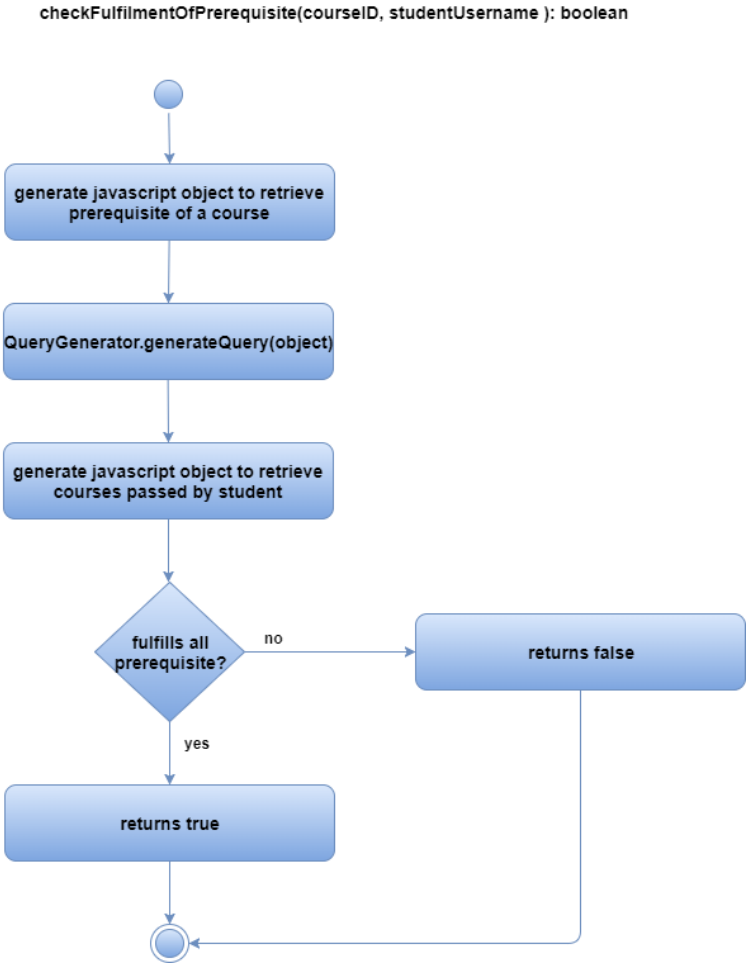


Figure 116 checkFulfillmentOfPayment() of ProgramChairperson

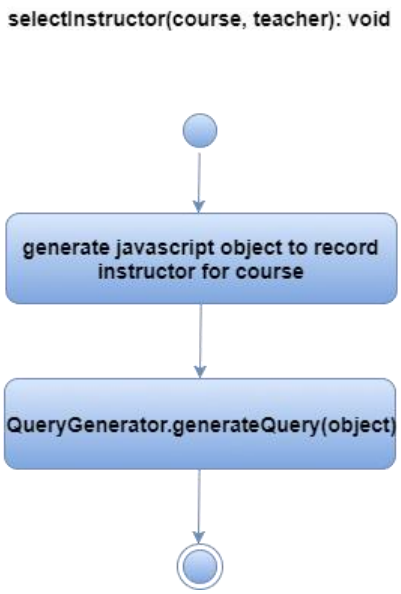


Figure 117 selectInstructor() of ProgramChairperson

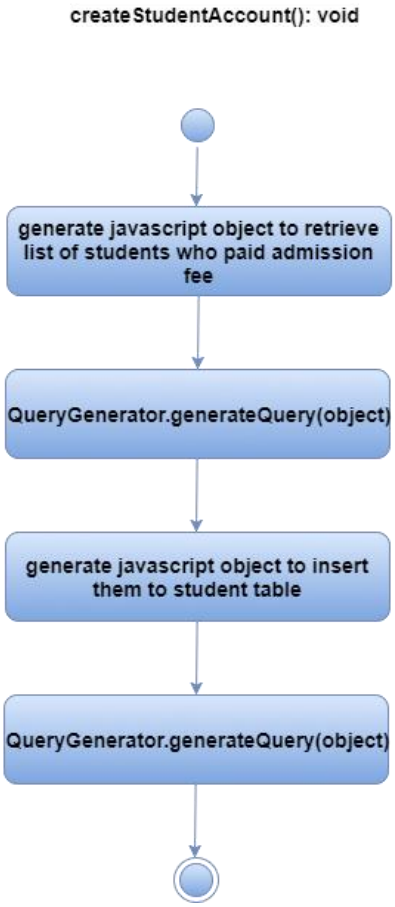


Figure 118 createStudentAccount() of ProgramChairperson

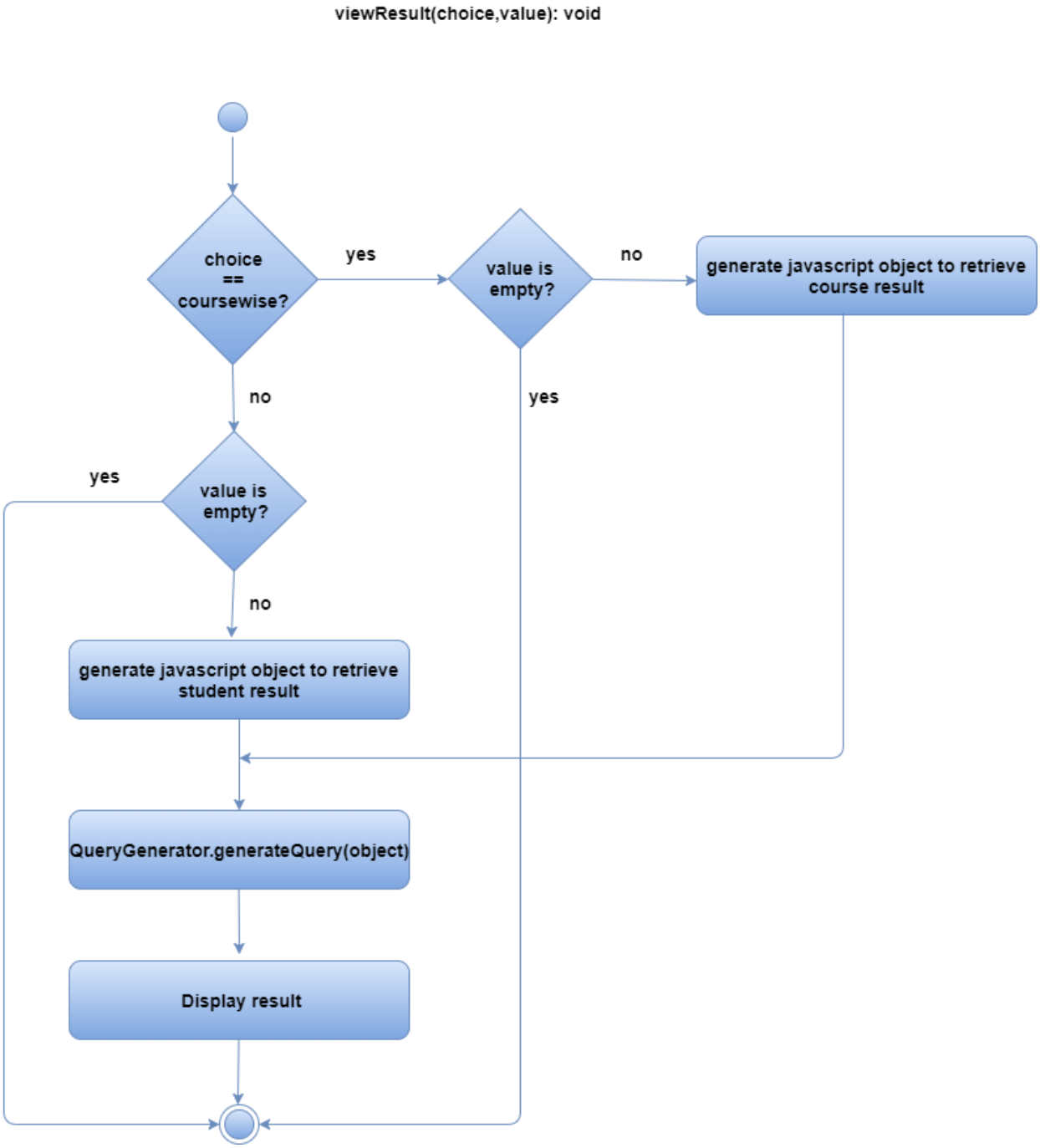


Figure 119 viewResult() of ProgramChairperson

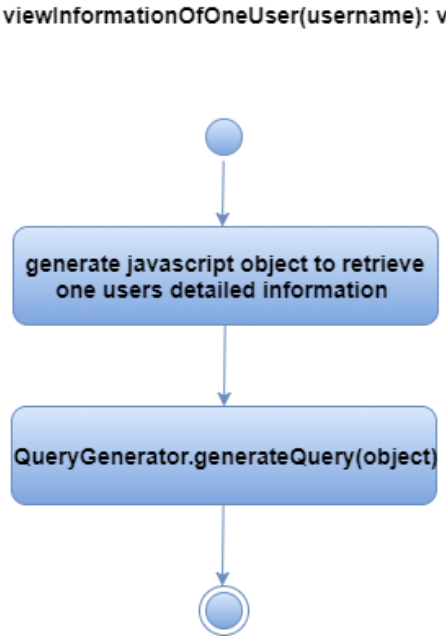


Figure 120 viewInformationOfOneUser() of ProgramChairperson

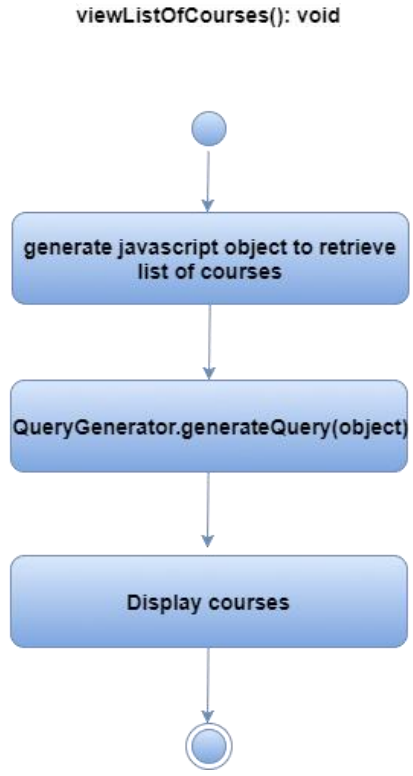


Figure 121 viewListOfCourses () of ProgramChairperson

Figure 122-134 show UML activity diagrams for Teacher class.

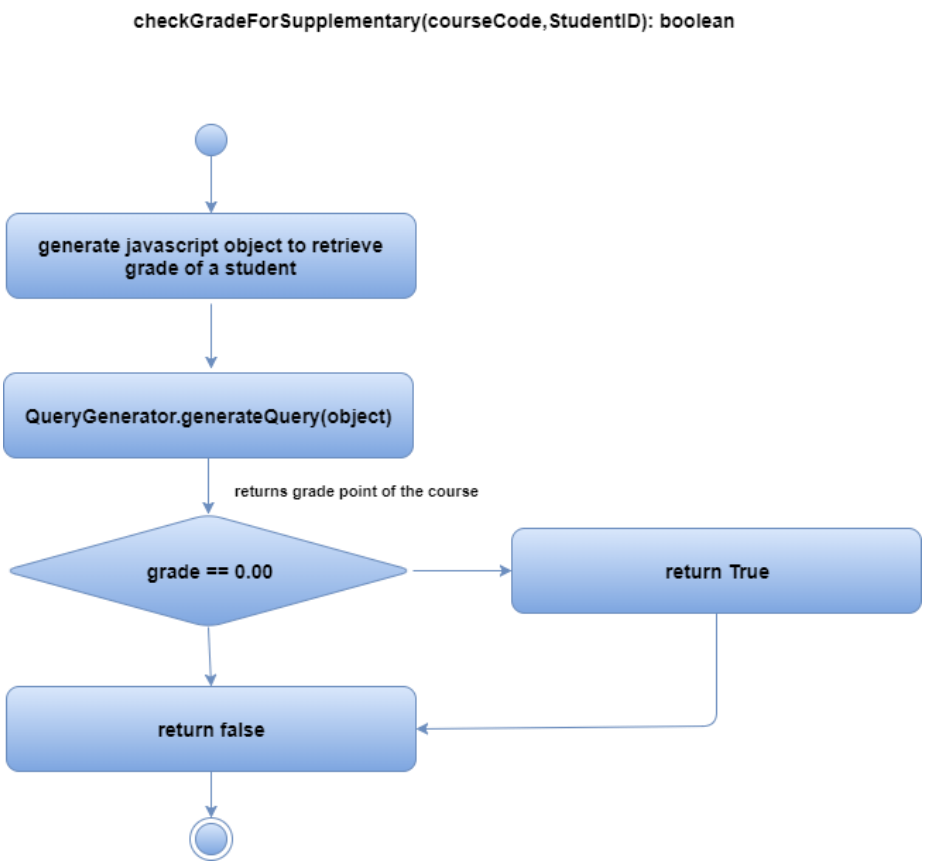


Figure 122 checkGradeForSupplementary() of Teacher

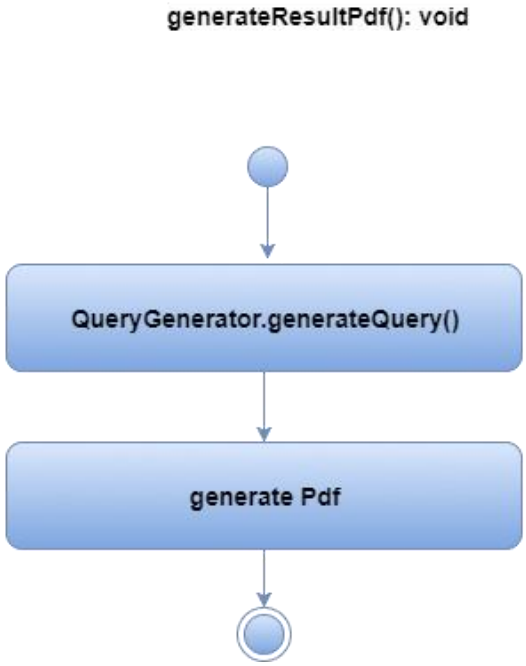


Figure 123 generateResultPdf() of Teacher

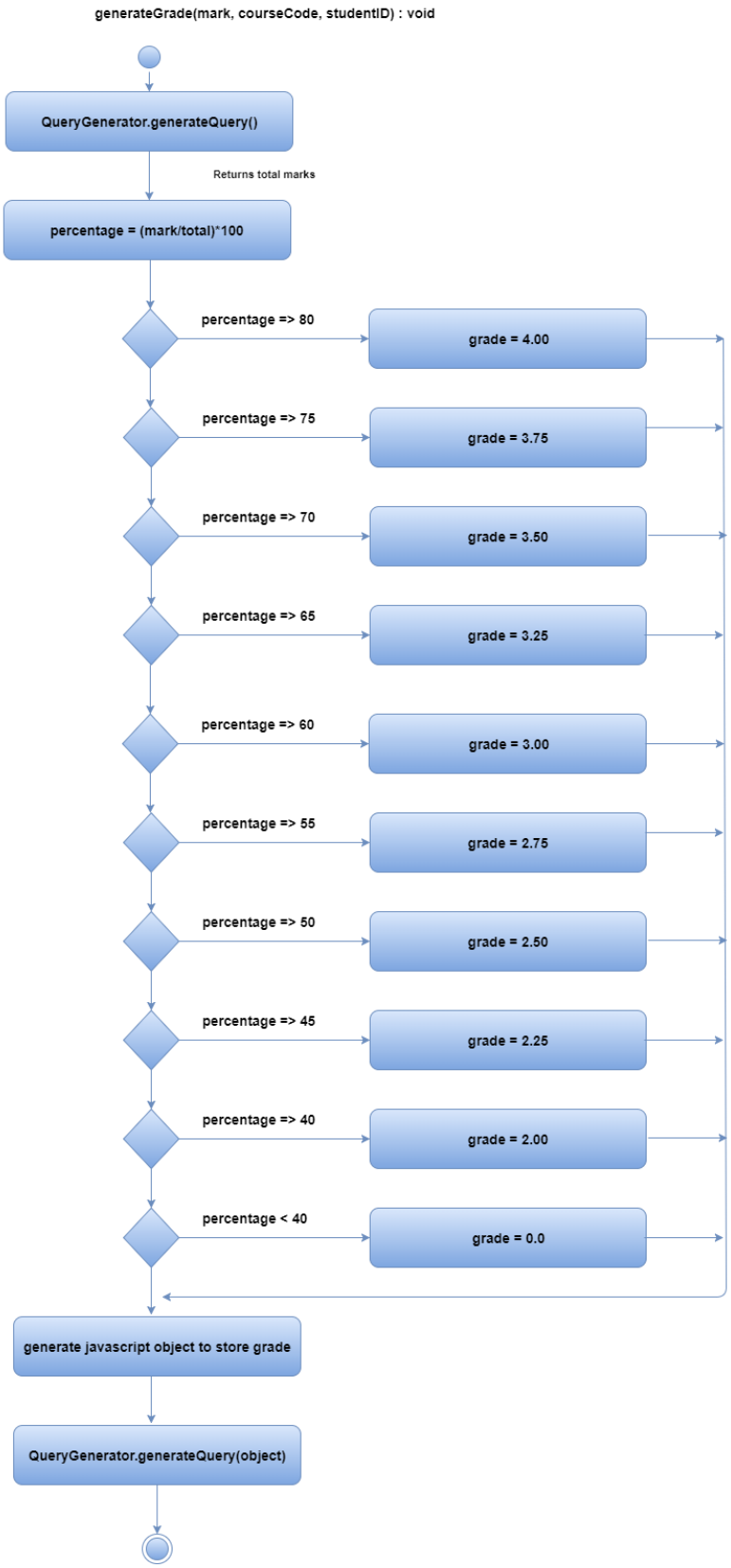


Figure 124 generateGrade() of Teacher

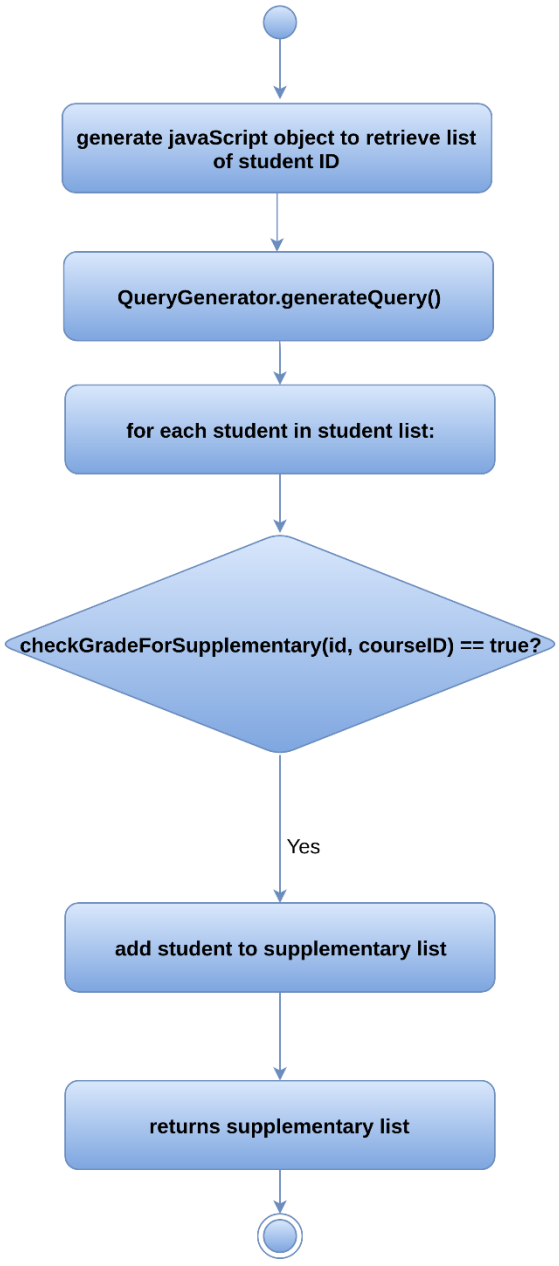


Figure 125 generateSupplementaryList() of Teacher

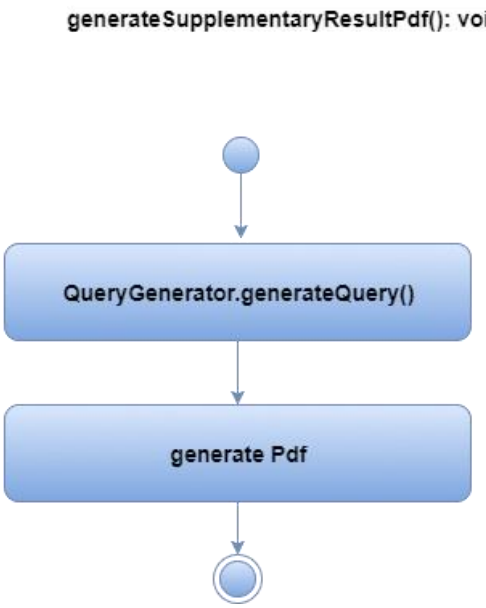


Figure 126 generateSupplementaryResultPdf() of Teacher

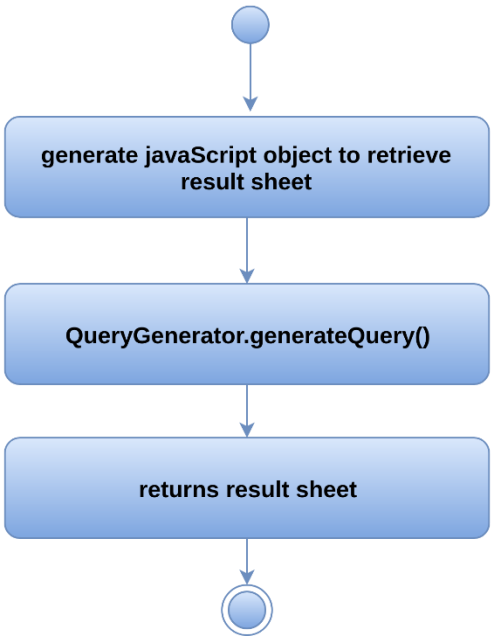


Figure 127 getResultSheet() of Teacher

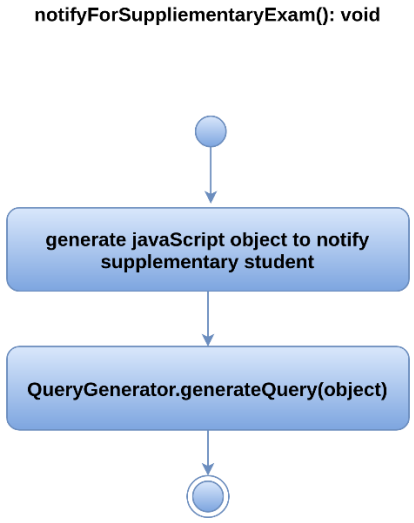


Figure 128 notifyForSupplementaryExam() of Teacher

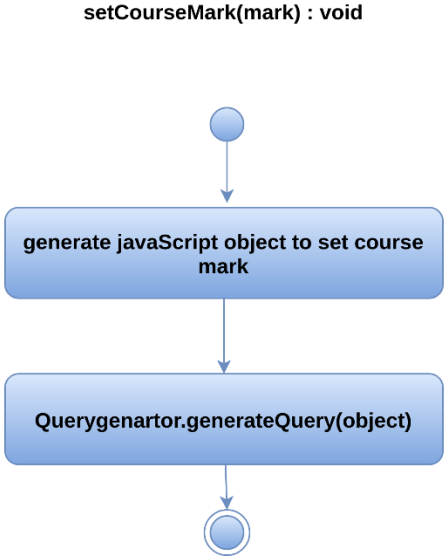


Figure 129 setCourseMark() of Teacher

setSupplementaryMakr(mark): void

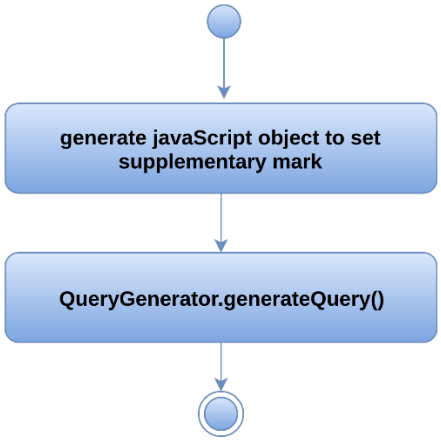


Figure 130 setSupplementaryMark() of Teacher

updateCourseInformation(course ID, information):void

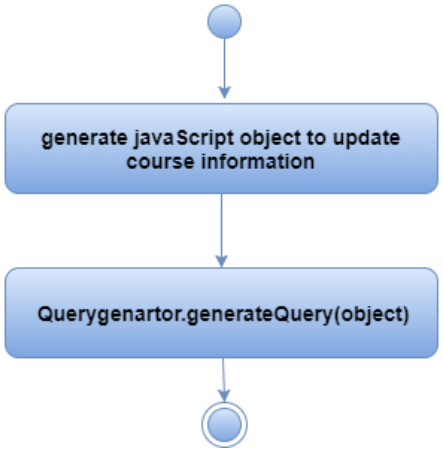


Figure 131 updateCourseInformation() of Teacher

viewCourseInformation(courseCode): courseInformation

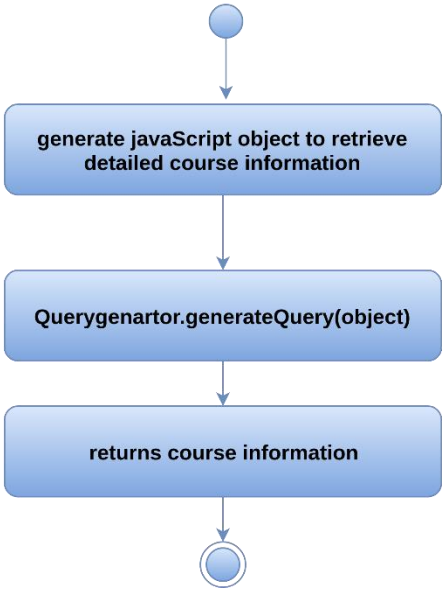


Figure 132 viewCourseInformation() of Teacher

viewListOfCourse(): courseList

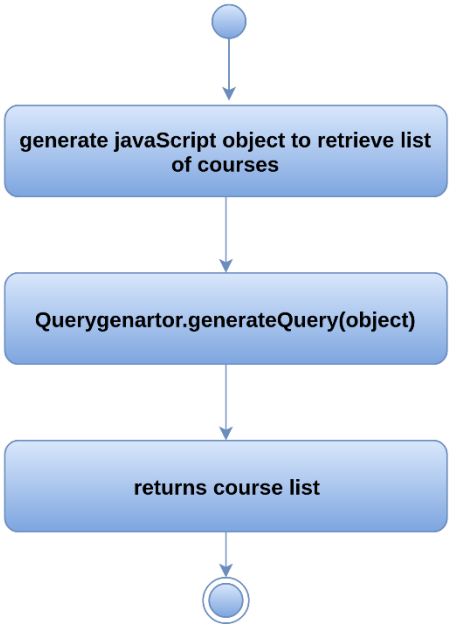


Figure 133 viewListOfCourse() of Teacher

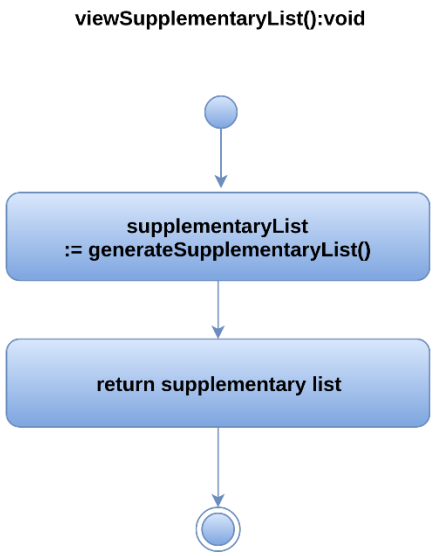


Figure 134 viewSupplementaryList() of Teacher

Figure 135-138 show UML activity diagrams for PDFHandler class.

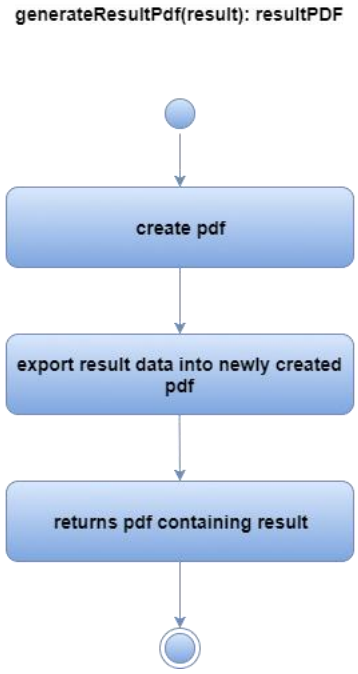


Figure 135 generateResultPdf() of PDFHandler

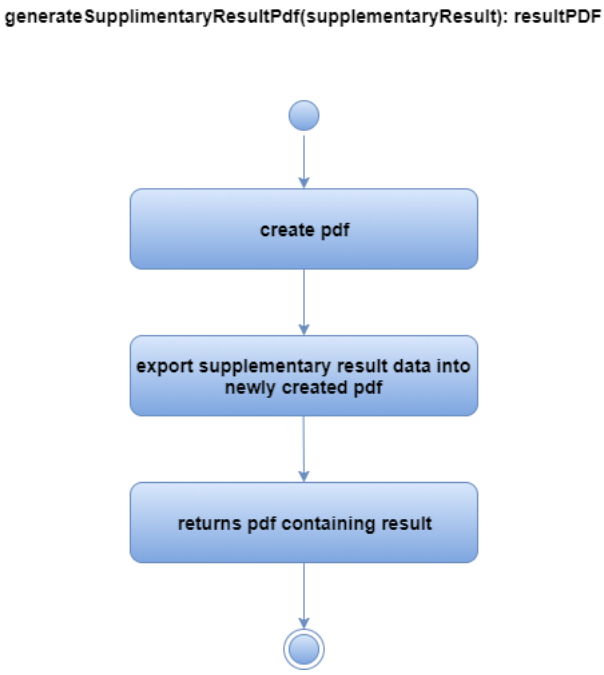


Figure 136 generateSupplementaryResultPdf() of PDFHandler

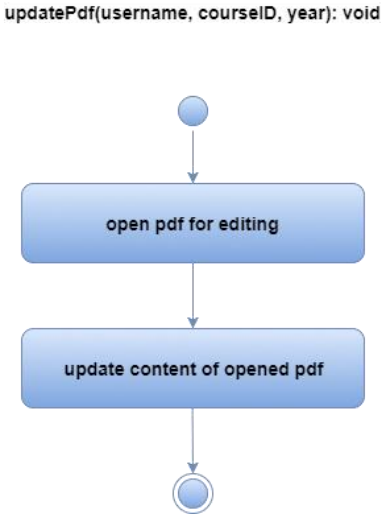


Figure 137 updatePdf() of PDFHandler

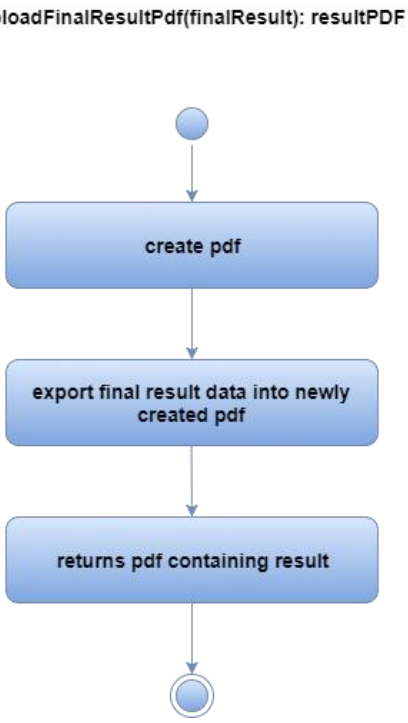


Figure 138 uploadFinalResultPdf() of PDFHandler

Figure 139-141 show UML activity diagrams for Student class.

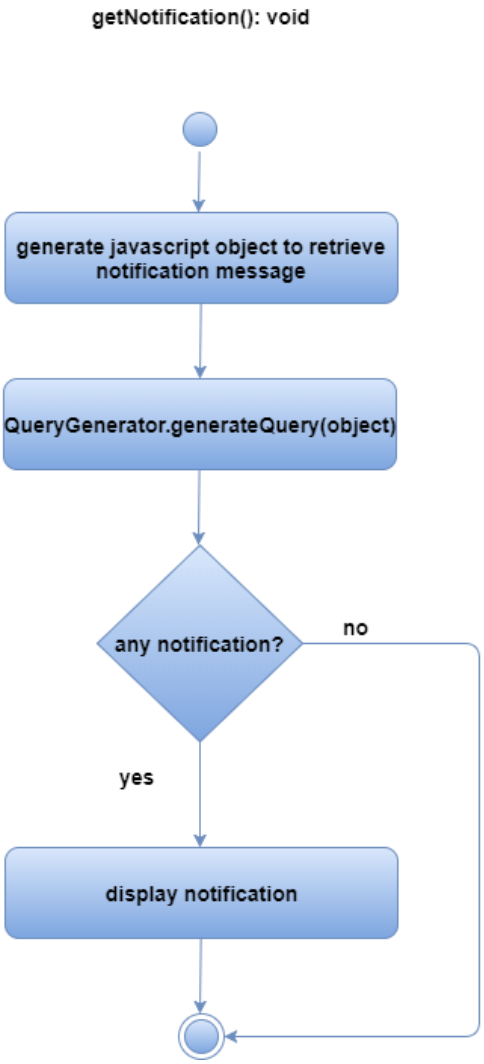


Figure 139 getNotification() of Student

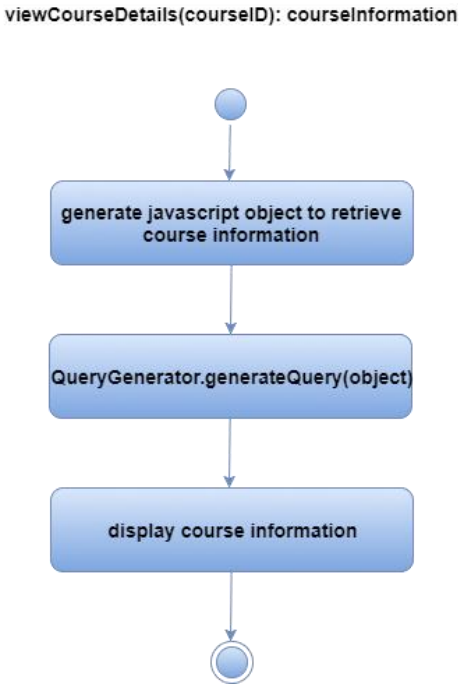


Figure 140 viewCourseDetails() of Student

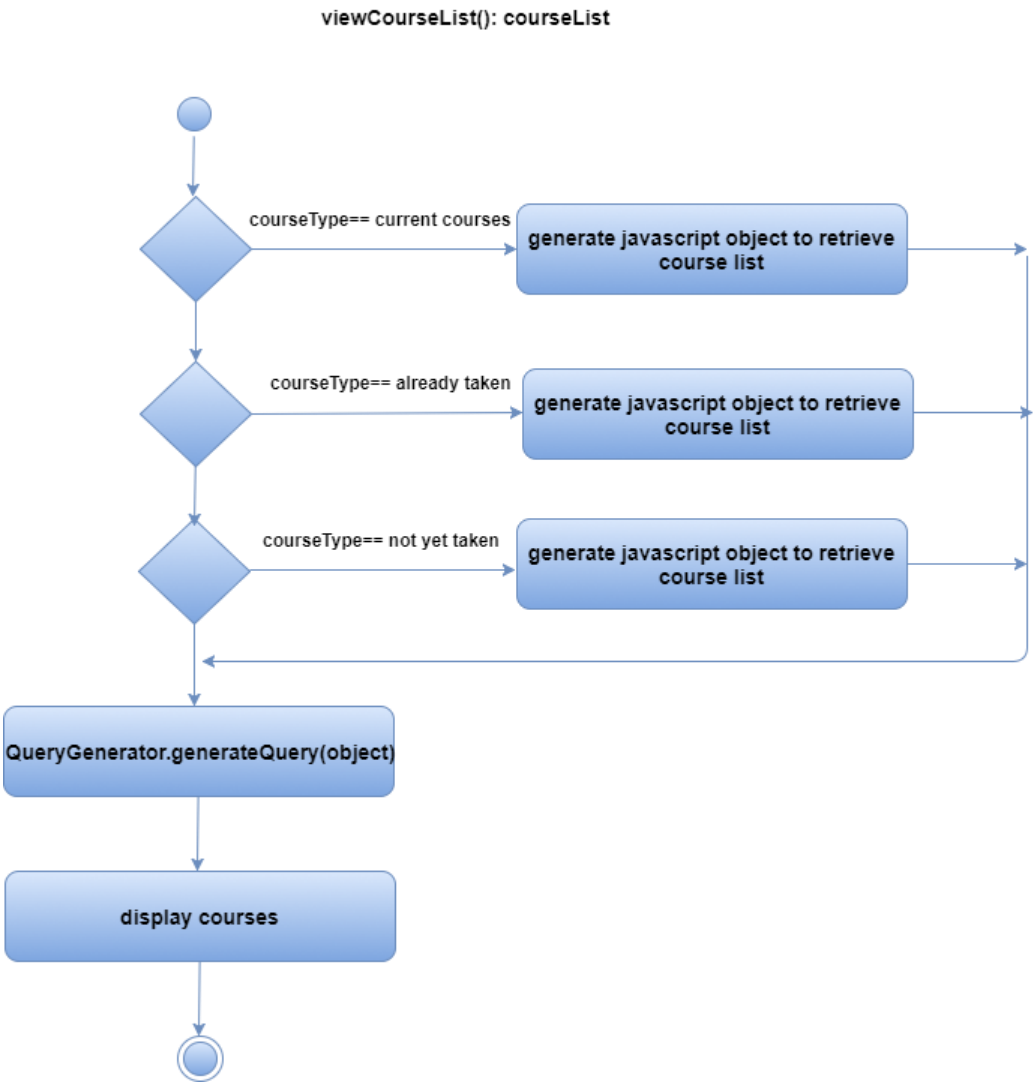


Figure 141 viewCourseList() of Student

Figure 142-143 show UML activity diagrams for ReportCard class.

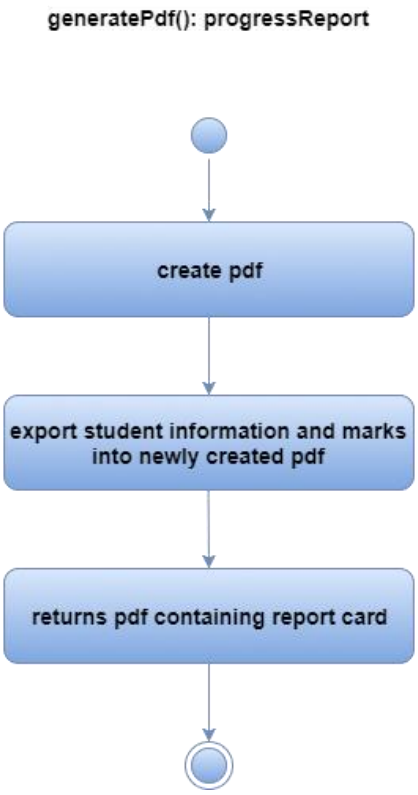


Figure 142 generatePdf() of ReportCard

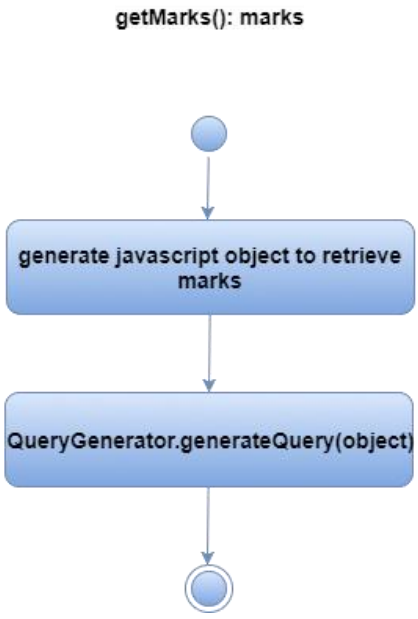


Figure 143 getMarks() of ReportCard

Figure 144-156 show UML activity diagrams for ExamController class.

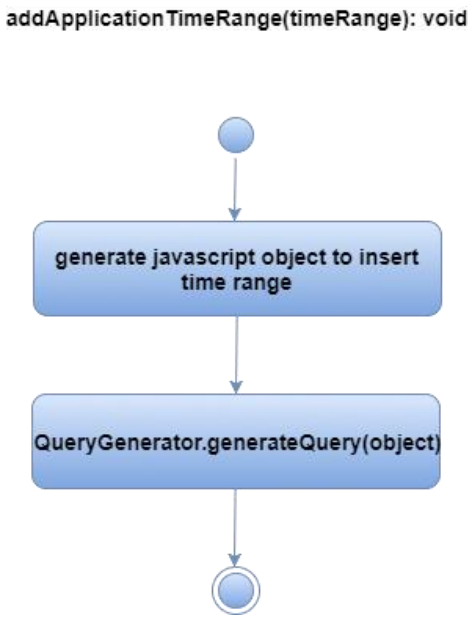


Figure 144 addApplicationTimeRange() of Exam Controller

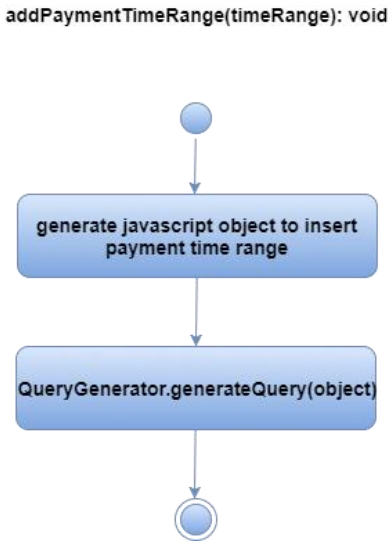


Figure 145 addPaymentTimeRange() of Exam Controller

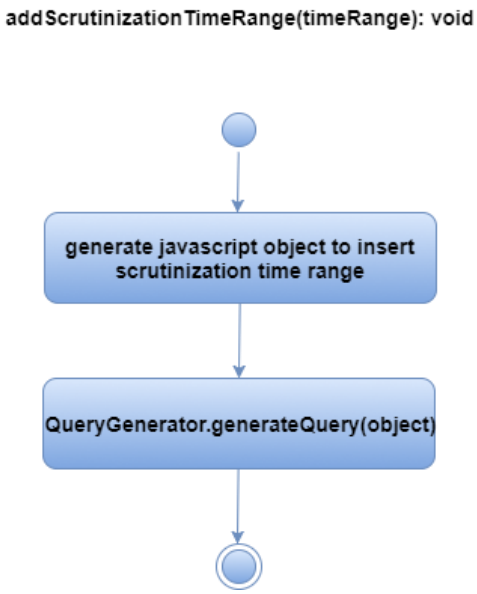


Figure 146 addScrutinizationTimeRange() of Exam Controller

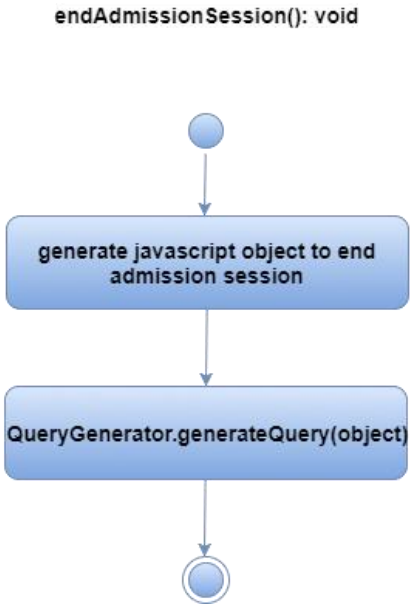


Figure 147 endAdmissionSession() of ExamController

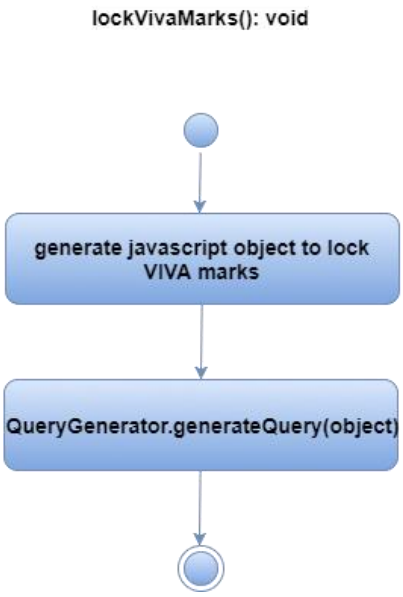


Figure 148 lockVivaMarks() of ExamController

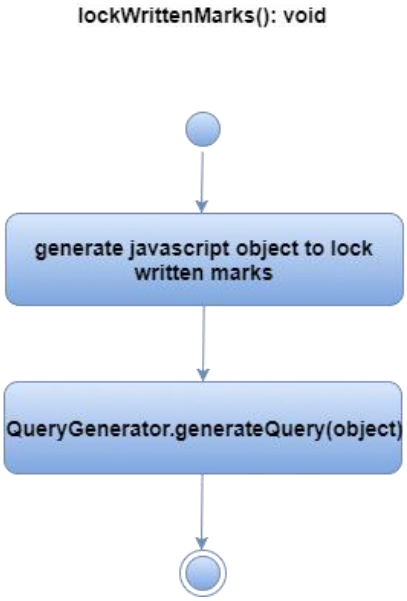


Figure 149 lockWrittenMarks() of ExamController

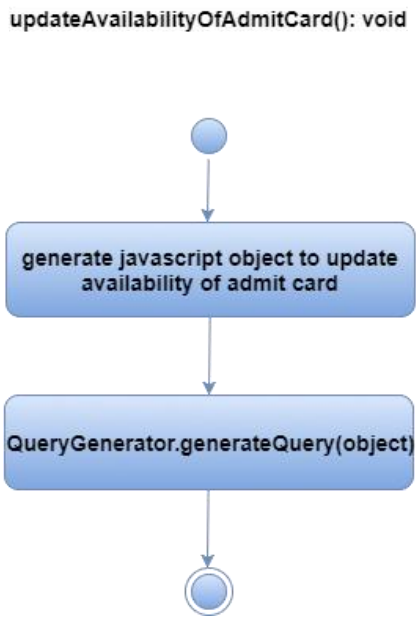


Figure 150 updateAvailabilityOfAdmitCard() of ExamController

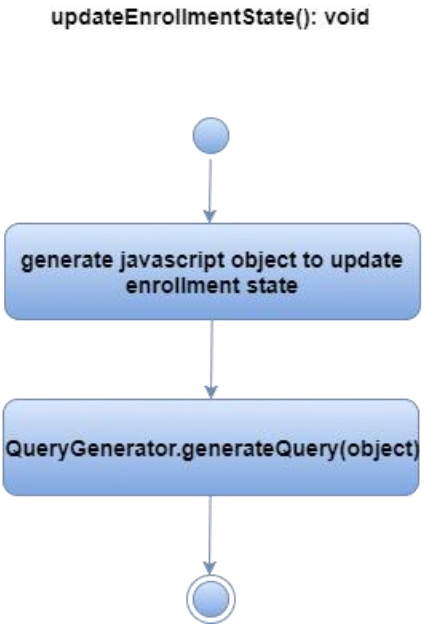


Figure 151 updateEnrollmentState() of ExamController

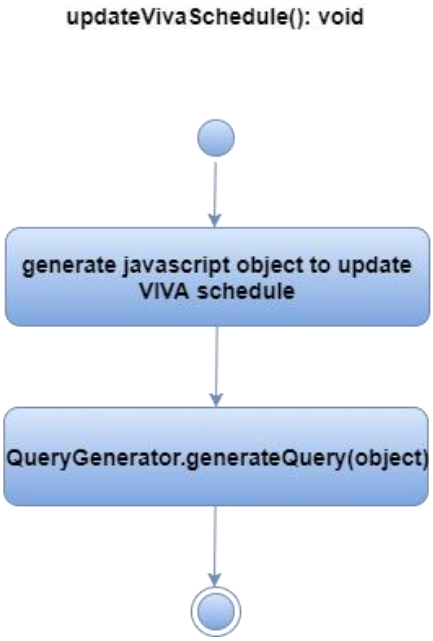


Figure 152 updateVivaSchedule() of ExamController

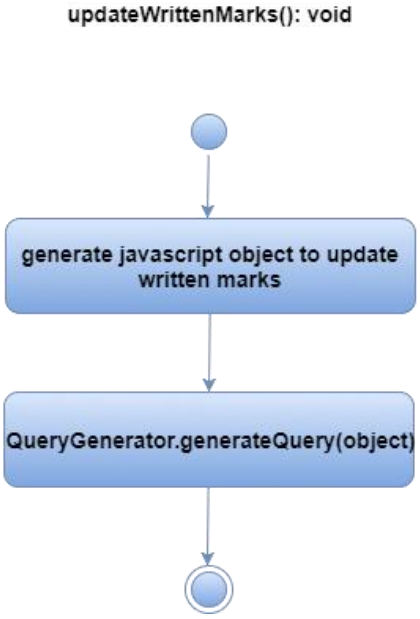


Figure 153 updateWrittenMarks() of ExamController

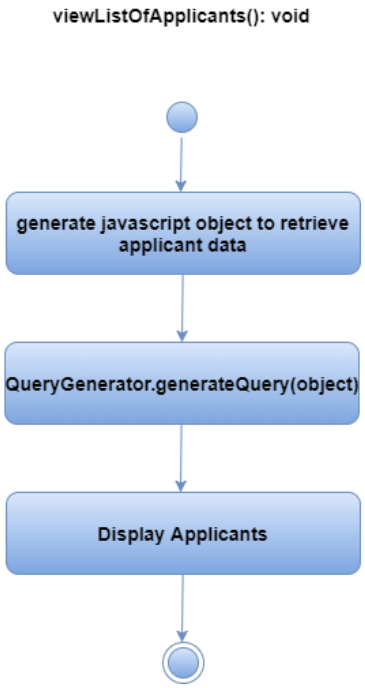


Figure 154 viewListOfApplicants() of ExamController

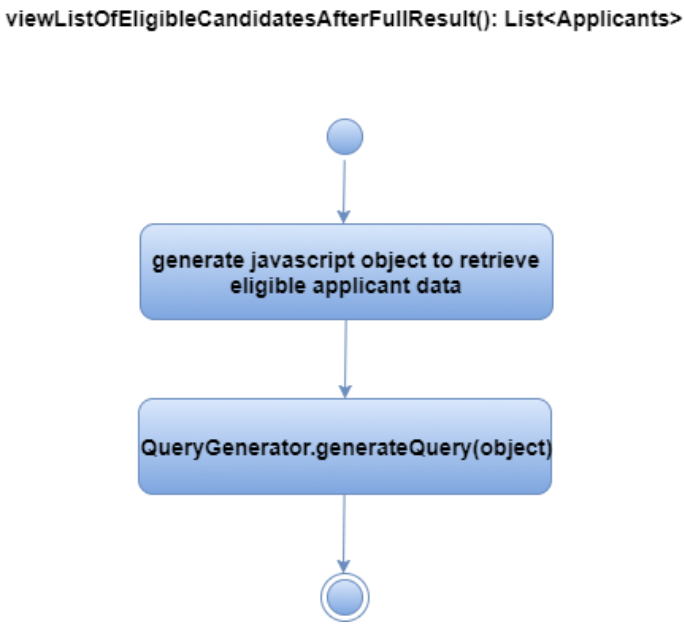


Figure 155 viewListOfEligibleCandidateAfterFullResult() of ExamController

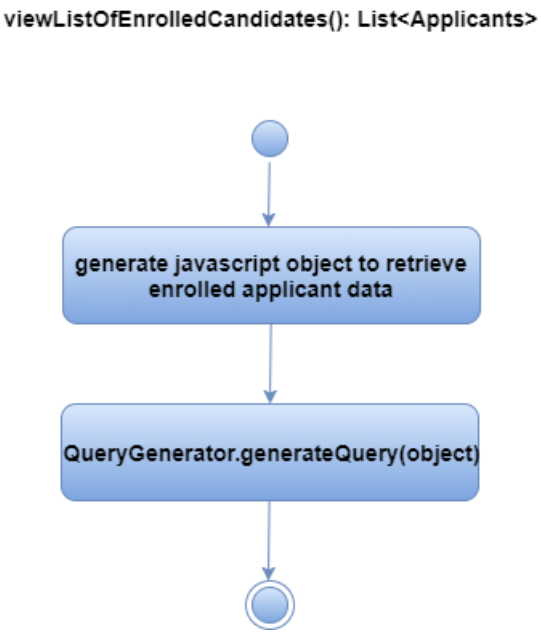


Figure 156 viewListOfEnrolledCandidates() of ExamController

Figure 157-160 show UML activity diagrams for RoomHandler class.

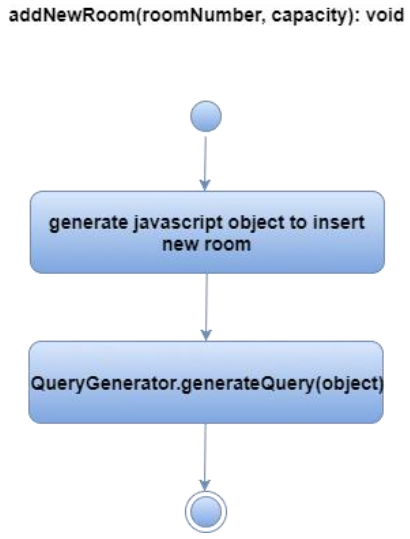


Figure 157 addNewRoom() of RoomHandler

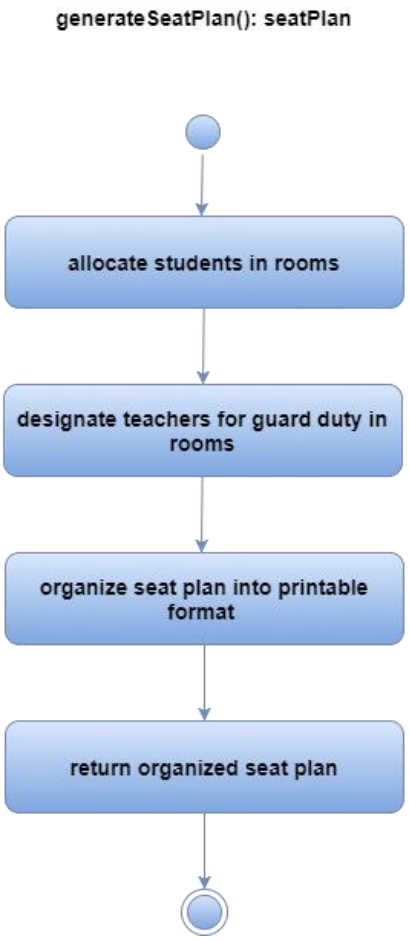


Figure 158 generateSeatPlan() of RoomHandler

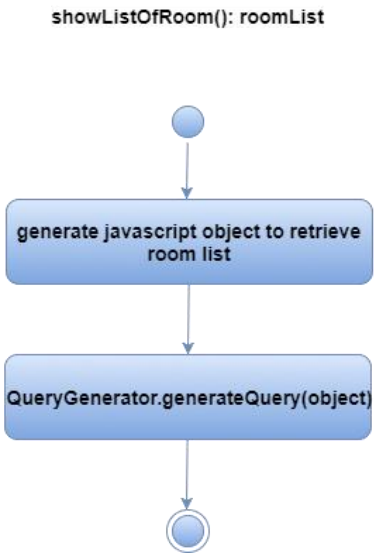


Figure 159 showListOfRoom() of RoomHandler

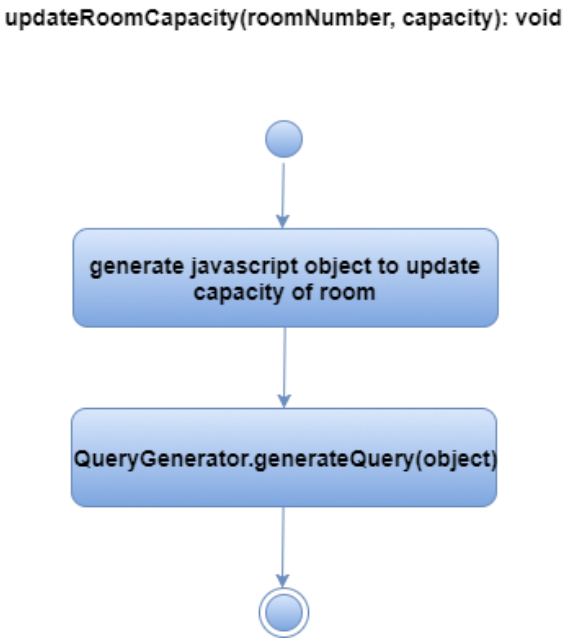


Figure 160 updateRoomCapacity() of RoomHandler

Figure 161-163 show UML activity diagrams for Scrutinizer class.

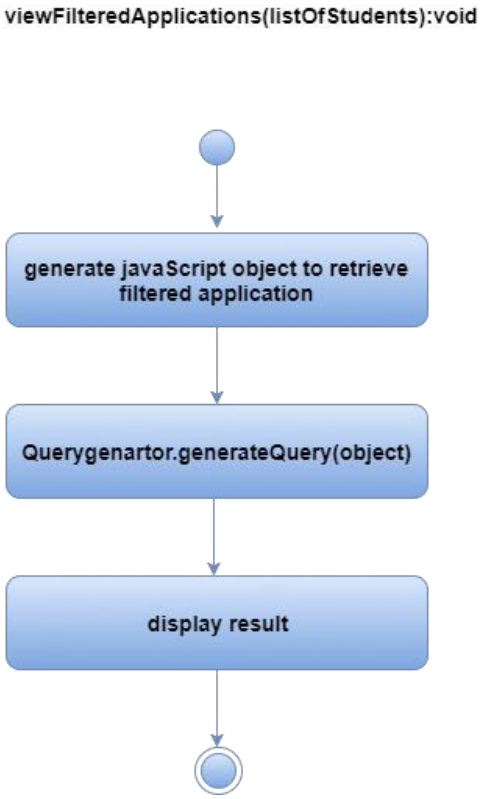


Figure 161 viewFilteredApplcations() of Scrutinizer

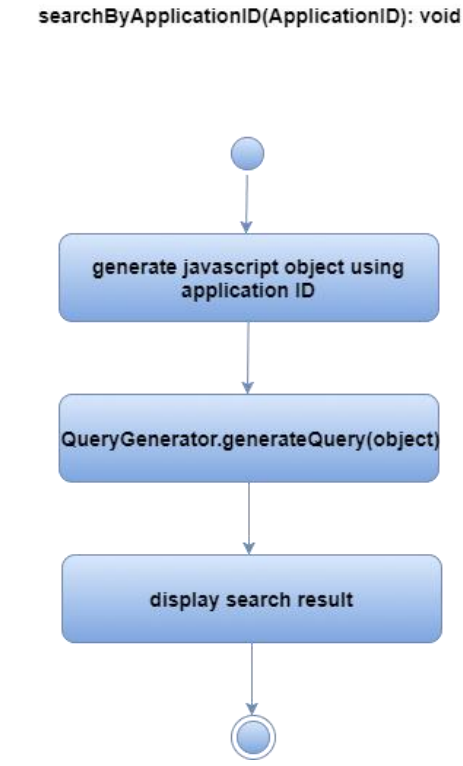


Figure 162 searchByApplicationID () of Scrutinizer

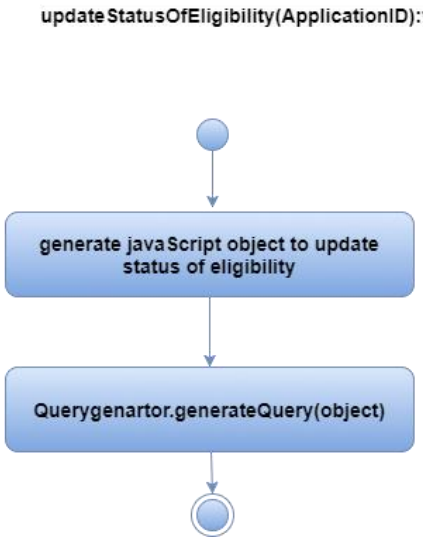


Figure 163 updateStatusOfEligibility() of Scrutinizer

Figure 164-165 show UML activity diagrams for RollGenerator class.

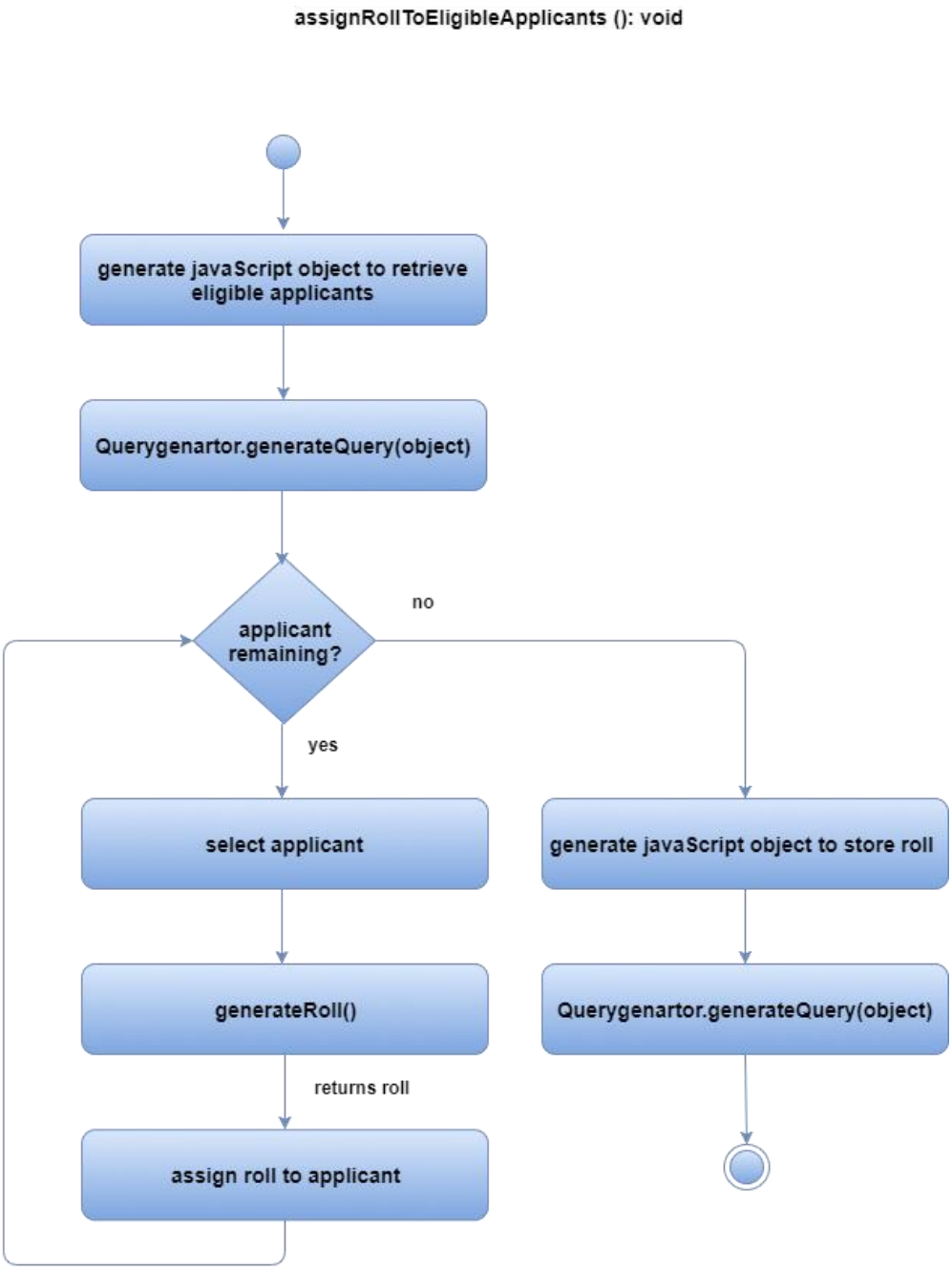


Figure 164 assignRollToEligibleApplicants() of RollGenerator

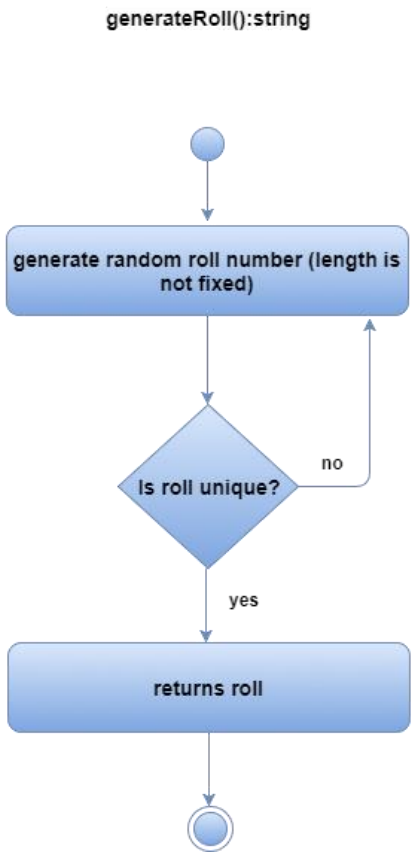


Figure 165 generateRoll() of RollGenerator

Figure 166-167 show UML activity diagrams for Receptionist class.

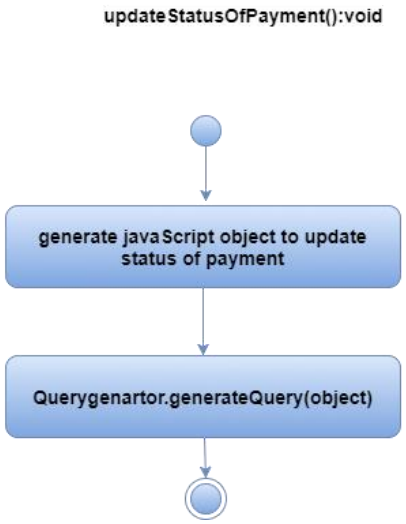


Figure 166 updateStatusOfPayment() of Receptionist

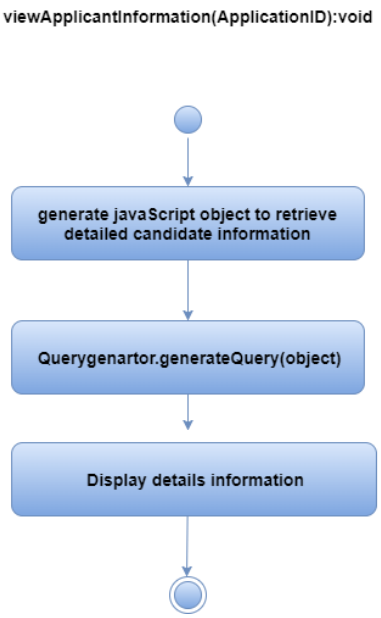


Figure 167 viewApplicantInformation() of Receptionist

3.4 Describe Persistent Data Sources and Identify the Classes required to Manage Them

For PGDIT Automation System we will use MySQL Database as data source and DatabaseHandler class is required to manage the database (shown in figure 168).

Figure 168 DatabaseHandler class

3.5 Develop and Elaborate Behavioral Representations for a Class or Component

During component-level design, it is sometimes necessary to model the behavior of a design class. For this purpose state diagram can be used. A state diagram is an illustration of the states an object can attain as well as the transitions between those states

Figure 169 shows state diagram of the Applicant class.

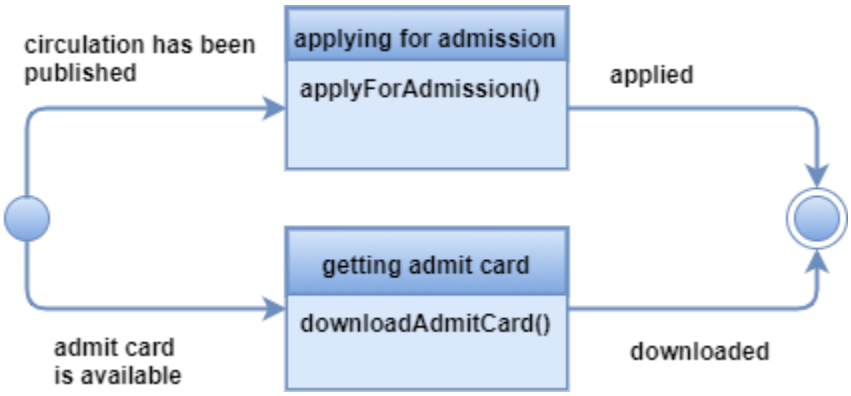


Figure 169 State Diagram of the Applicant class

Figure 170 shows state diagram of the Authentication class.

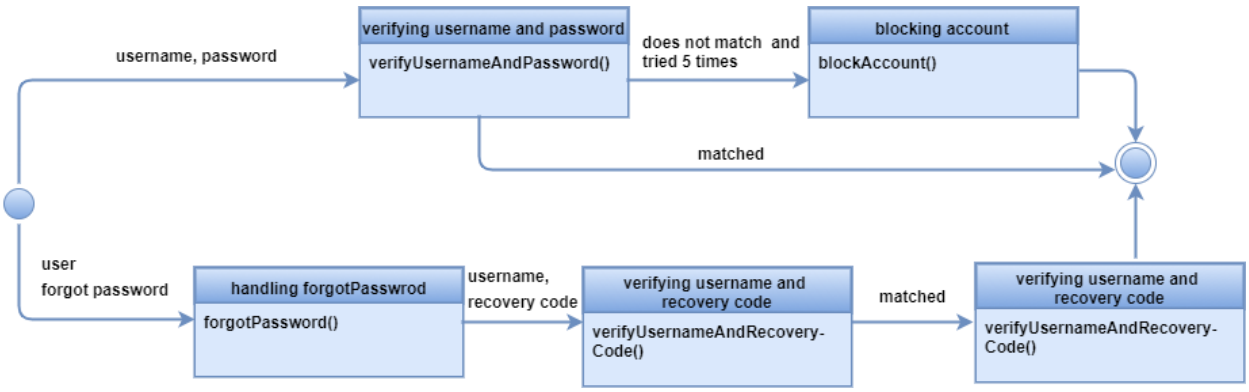


Figure 170 State Diagram of the Authentication class

Figure 171 shows state diagram of the User class.

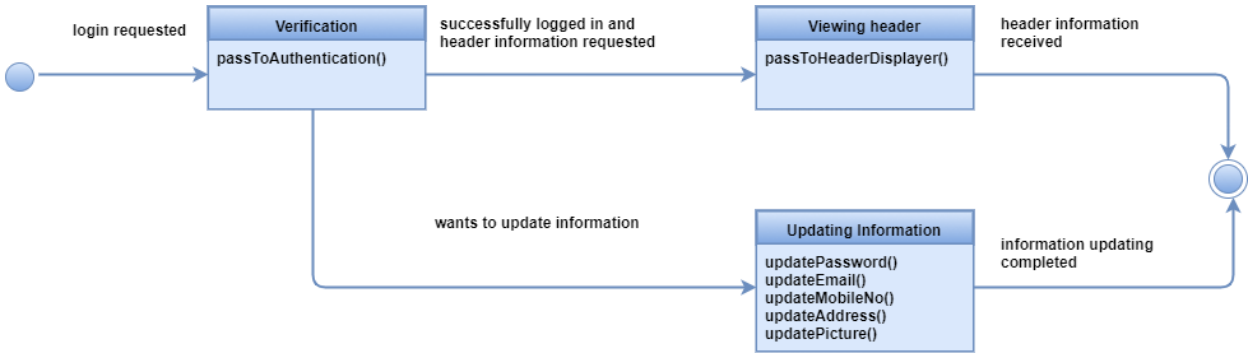


Figure 171 state diagram of the Authentication class

Figure 172 shows state diagram of the HeaderDisplayer class.

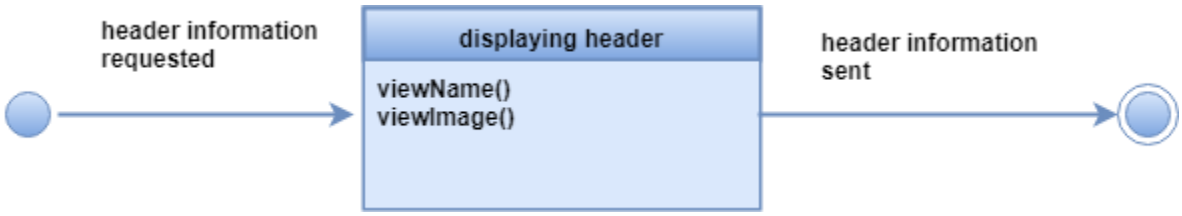


Figure 172 State Diagram of the HeaderDisplayer class

Figure 173 shows state diagram of the ProgramChairperson class.

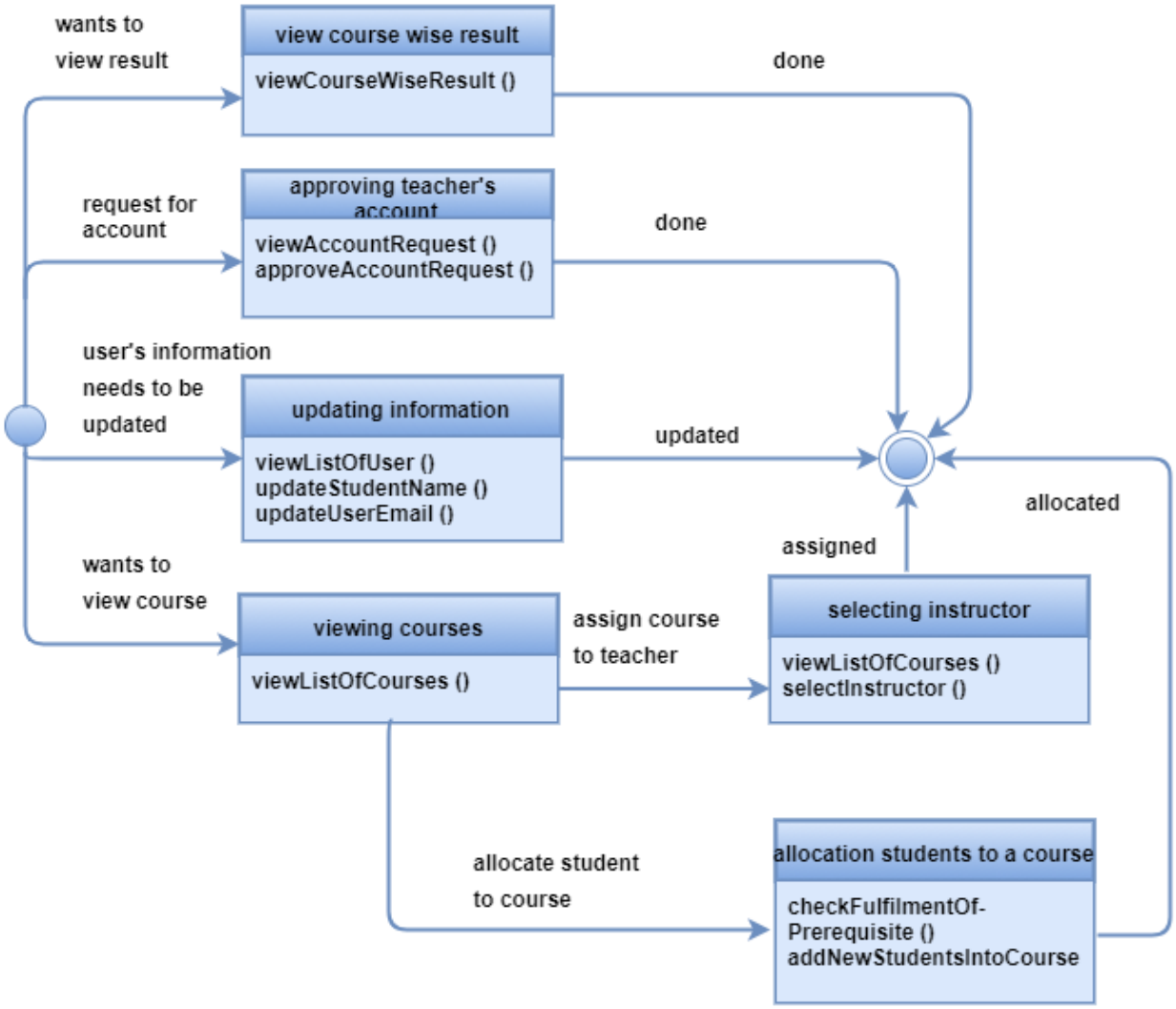


Figure 173 State Diagram of the ProgramChairperson class

Figure 174 shows state diagram of the ExamController class.

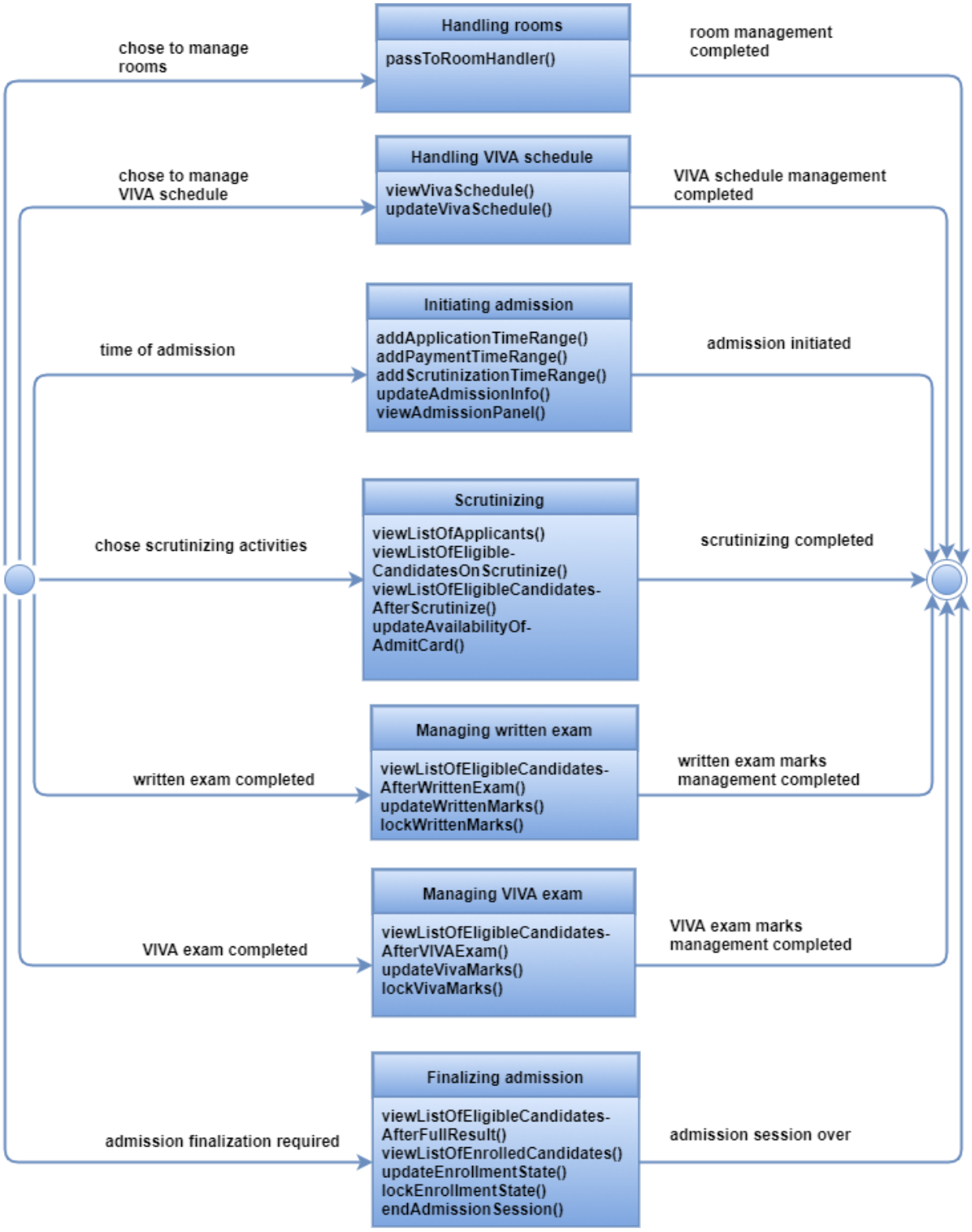


Figure 174 State Diagram of the ExamController class

Figure 175 shows state diagram of the RoomHandler class.

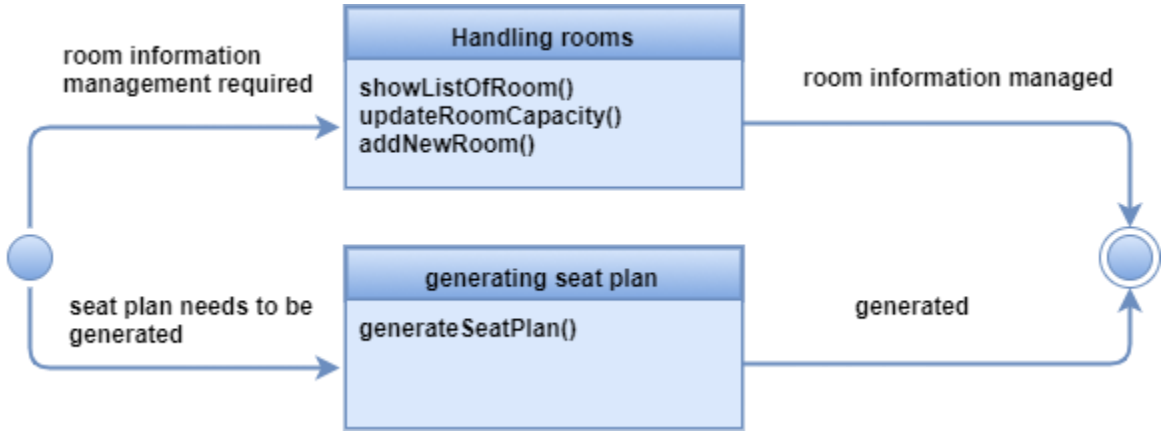


Figure 175 State Diagram of the RoomHandler class

Figure 176 shows state diagram of the Receptionist class.

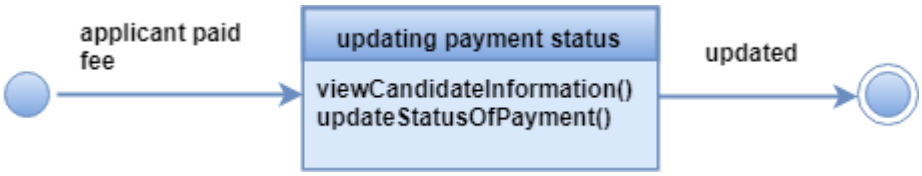


Figure 176 State Diagram of the Receptionist class

Figure 177 shows state diagram of the Scrutinizer class.

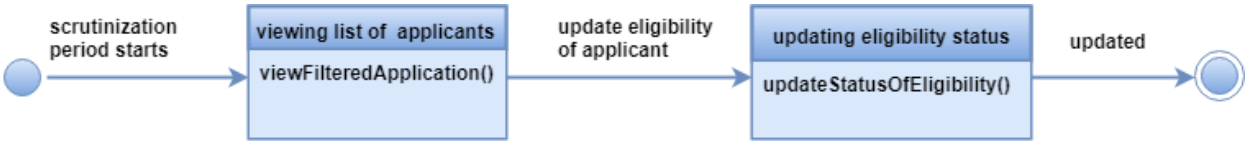


Figure 177 State Diagram of the Scrutinizer class

Figure 178 shows state diagram of the RollGenerator class.

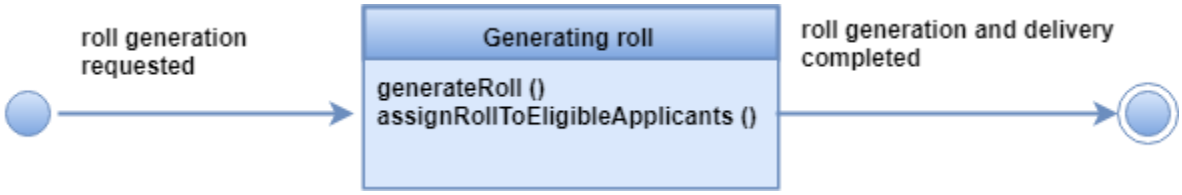


Figure 178 State Diagram of the RollGenerator class

Figure 179 shows state diagram of the Teacher class.

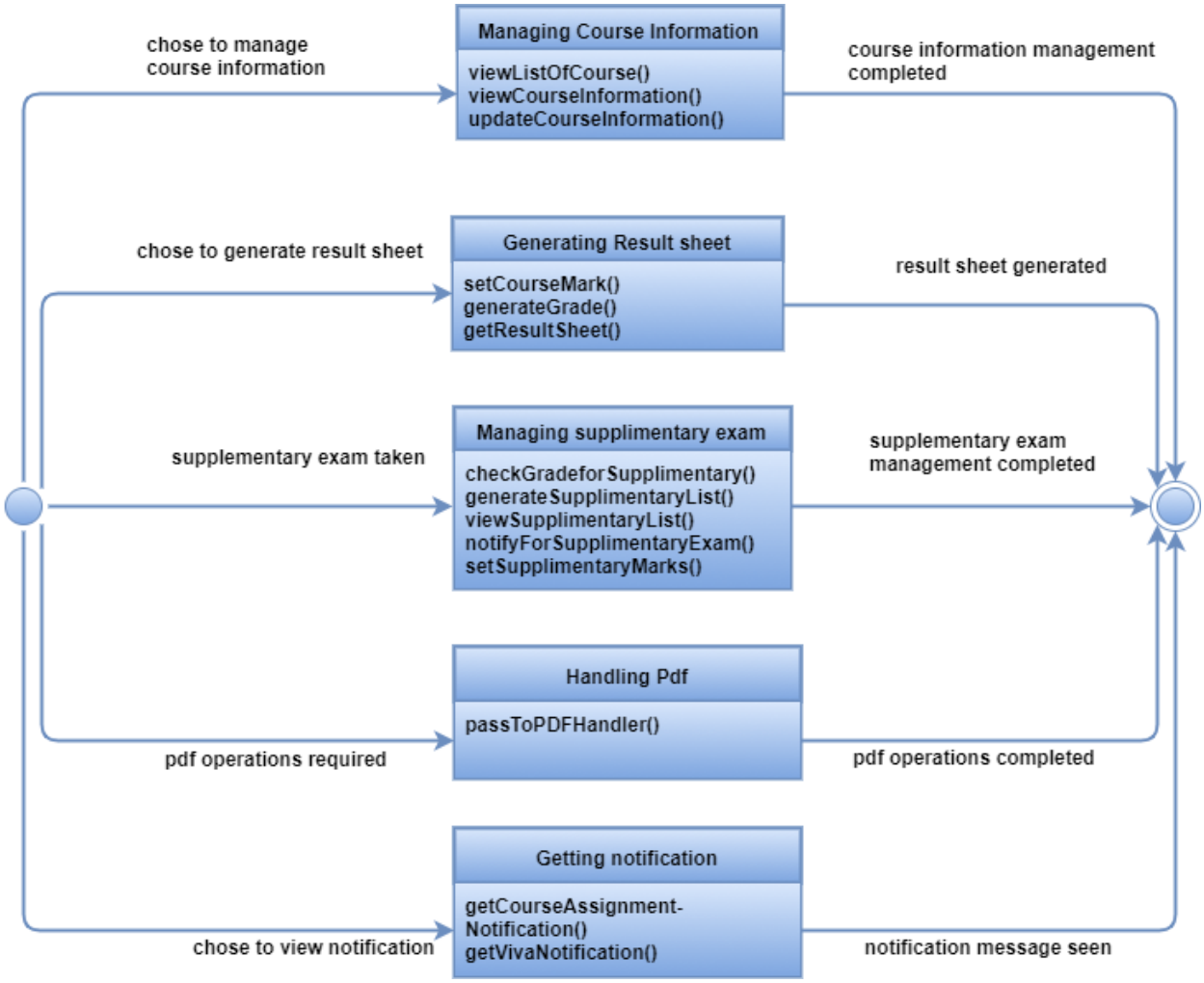


Figure 179 State Diagram of the Teacher class

Figure 180 shows state diagram of the PDFHandler class.

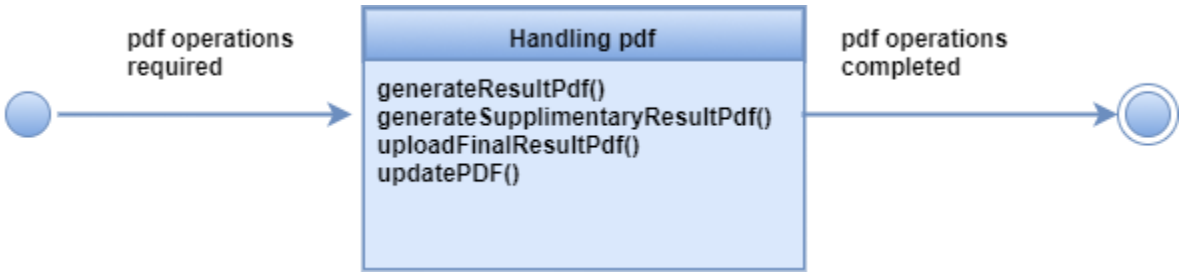


Figure 180 State Diagram of the PDFHandler class

Figure 181 shows state diagram of the Student class.

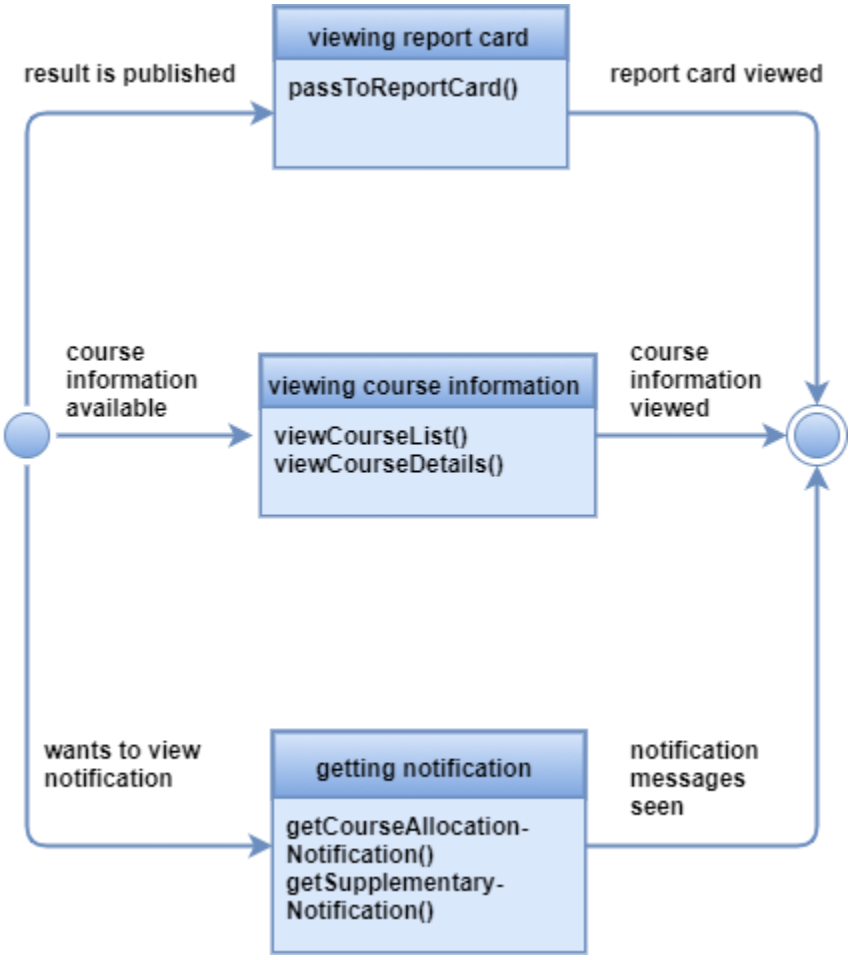


Figure 181 State Diagram of the Student class

Figure 182 shows state diagram of the ReportCard class.

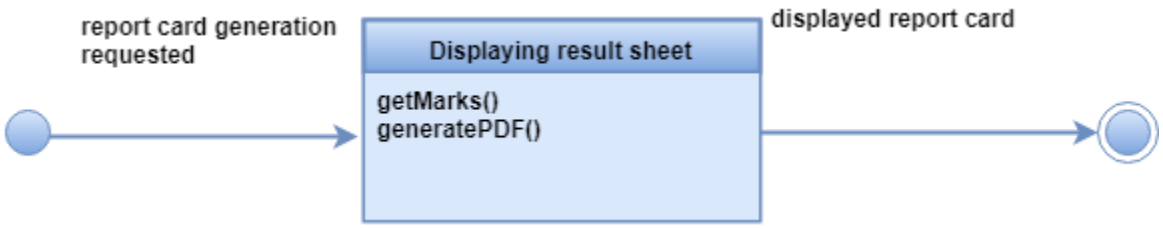


Figure 182 State Diagram of the ReportCard class

3.6 Elaborate Deployment Diagrams to Provide Additional Implementation Detail

Deployment diagrams are represented in descriptor form. In this form, major system functions are represented within the context of the computing environment that will house them.

Figure 183 shows deployment diagram of PGDIT Automation System.

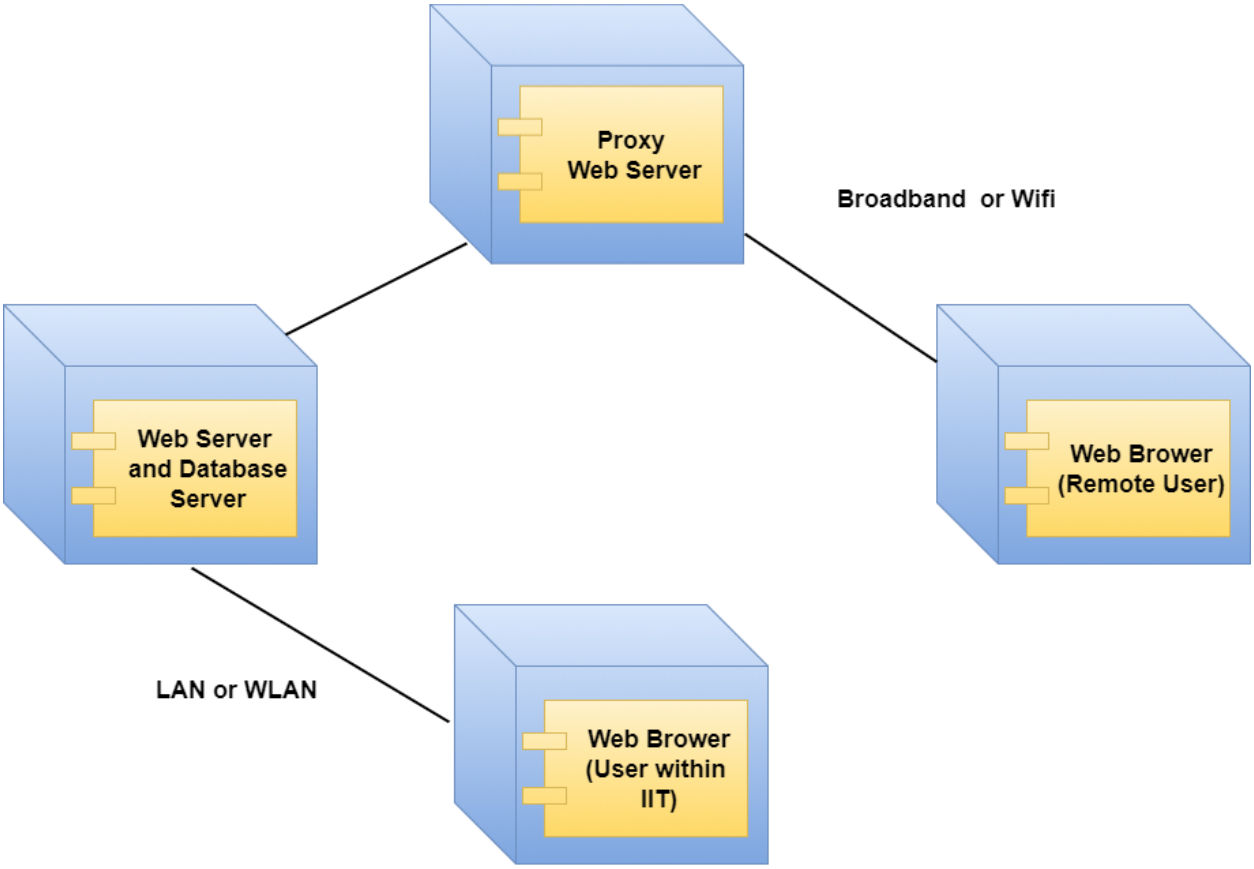


Figure 183 Deployment Diagram of PGDIT Automation System

Chapter 4. User Interface Design

User interface design creates an effective communication medium between a human and a computer. User Interface (UI) Design focuses on anticipating what users might need to do and ensuring that the interface has elements that are easy to access, understand, and use to facilitate those actions.

4.1 Interface Analysis

We divide interface analysis into following parts:

1. User Analysis
2. Task Analysis

4.1.1 User Analysis

User analysis helps us to know who the end users are, what is likely to motivate and please them, how they can be grouped into different user classes or profiles, what their mental models of the system are, and how the user interface must be characterized to meet their needs. From the requirements specification, we have found the following users for PGDIT Automation System:

- Teacher
- Student
- Program Chairperson
- Applicant
- Exam Controller
- Scrutinizer
- Receptionist

Table 2 Analyzing Teacher

Teacher	
Work type	Trained Professionals
Educational Level	Masters
Learning capability	Training
Skills	Above Average
Age	25-65
Domain expert	Yes
Application expert	No
Office hour	Normal
Frequency of use	Occasionally
Consequence of a mistake	Low

General computer experience	Yes
Want to know about technology that sits behind the interface	No

Table 3 Analyzing Student

Student	
Educational Level	Honors
Learning capability	Training
Skills	Average
Age	22-40
Application expert	No
Frequency of use	Occasionally
Consequence of a mistake	Low
General computer experience	Yes
Want to know about technology that sits behind the interface	No

Table 4 Analyzing Program Chairperson

Program Chairperson	
Work type	Trained Professionals
Educational Level	Masters
Learning capability	Training
Skills	Above Average
Age	25-65
Domain expert	Yes
Application expert	No
Office hour	Normal
Frequency of use	Occasionally
Consequence of a mistake	Low
General computer experience	Yes
Want to know about technology that sits behind the interface	No

Table 5 Analyzing Applicant

Applicant	
Educational Level	Honors
Learning capability	Training
Skills	Average

Age	22-40
Application expert	No
Frequency of use	Occasionally
Consequence of a mistake	Low
General computer experience	Yes
Want to know about technology that sits behind the interface	No

Table 6 Analyzing Exam Controller

Exam Controller	
Work type	Trained Professionals
Educational Level	Masters
Learning capability	Training
Skills	Above Average
Age	25-65
Domain expert	Yes
Application expert	No
Office hour	Normal
Frequency of use	Occasionally
Consequence of a mistake	High
General computer experience	Yes
Want to know about technology that sits behind the interface	No

Table 7 Analyzing Scrutinizer

Scrutinizer	
Work type	Trained Professionals
Educational Level	Masters
Learning capability	Training
Skills	Above Average
Age	25-65
Domain expert	Yes
Application expert	No
Office hour	Normal
Frequency of use	Occasionally
Consequence of a mistake	Medium
General computer experience	Yes
Want to know about technology that sits behind the interface	No

Table 8 Analyzing Receptionist

Receptionist	
Work type	Clerical
Educational Level	Masters
Learning capability	Training
Skills	Average
Age	25-50
Domain expert	Yes
Application expert	No
Office hour	Normal
Frequency of use	Occasionally
Consequence of a mistake	Low
General computer experience	Yes
Want to know about technology that sits behind the interface	No

4.1.2 Task Analysis

In this step we identify and analyze the tasks of every users separately.

- Applicants: The tasks of applicants are given below:

1. Apply for admission
   Goal: Apply for admission by filling up application form.
   Precondition: Submission of application has been activated.
   Sub-tasks:
   I. fill up form
   II. receive application ID

2. Download admit card
   Goal: Download admit card of admission test.
   Precondition:
   I. eligible for admission test
   II. application ID and date of birth are valid
   III. admit card is available
   Sub-tasks:
   I. provide application ID and date of birth
   II. get admit card

- Program Chairperson: Program Chairperson has following tasks:

1. Create student account

Goal: Create account for new students.
Precondition: Must be logged in as program chairperson.
Sub-tasks:

I. view list of students who paid admission fee
II. create account

2. Enroll student in course
   Goal: Allocate course to a student.
   Precondition: Must be logged in as program chairperson.
   Sub-tasks:
   I. check students who fulfill prerequisite
   II. select students
   III. notify selected students

3. Assign course to teacher
   Goal: Select a teacher as instructor of a course.
   Precondition: Must be logged in as program chairperson.
   Sub-tasks:
   I. view list of teachers
   II. select teacher
   III. notify teacher

4. Update user's email
   Goal: Update any user's email address.
   Precondition: Must be logged in as program chairperson.
   Sub-tasks:
   I. select user
   II. update email

5. Update student's name
   Goal: Update any student's name.
   Precondition: Must be logged in as program chairperson.
   Sub-tasks:
   I. select student
   II. update name

6. Approve teacher's account
   Goal: Approve teacher's account request.
   Precondition: Must be logged in as program chairperson.
   Sub-tasks:
   I. check account requests
   II. approve requests

7. View result
   Goal: View course wise marks or result of a particular student.
   Precondition: Must be logged in as program chairperson.
   Sub-tasks:
   I. select category
   II. view result

- Exam Controller: The tasks of Exam Controller are given below:

1. Introduce new admission
   Goal: Setup new admission session and publish circular.
   Precondition: Must be logged in.
   Sub-tasks:
   I. provide timeline for applying online
   II. provide timeline for payment
   III. provide timeline for scrutinization

2. Manage room information
   Goal: Keep information about rooms of the institution up-to-date.
   Precondition: Must be logged in.
   Sub-tasks:
   I. view information of all rooms
   II. change capacity of a room
   III. add new room

3. Initiate written admission exam
   Goal: Generate seat plan and make admit cards available for download.
   Precondition: Must be logged in.
   Sub-tasks:
   I. generate seat plan for written exam
   II. turn off scrutinizing process
   III. make admit cards available for download

4. End written admission exam
   Goal: Prepare result of written admission exam.
   Precondition: Must be logged in.
   Sub-tasks:
   I. provide marks of written exam
   II. lock marks of written exam

5. Initiate VIVA admission exam
   Goal: Generate seat plan and VIVA schedule.

Precondition: Must be logged in.
Sub-tasks:

I. generate seat plan for VIVA exam
II. generate VIVA schedule
III. notify teachers about VIVA shift

6. End VIVA admission exam
   Goal: Prepare result of VIVA admission exam.
   Precondition: Must be logged in.
   Sub-tasks:
   I. provide marks of VIVA exam
   II. lock marks of VIVA exam

7. Finalize admission exam
   Goal: Finalize admission examination and complete enrollment of students.
   Precondition: Must be logged in.
   Sub-tasks:
   I. publish merit and waiting list
   II. enroll students
   III. end enrollment session
   IV. end admission session

8. Finalize term examination result
   Goal: Finalize term examination results.
   Precondition: Must be logged in.
   Sub-tasks:
   I. lock marks providing session
   II. upload final result for Exam Controller and Program Chairperson

- Teacher: Teacher has following tasks:

1. Edit Course Information

   Goal: Change existing information or introduce new information about the course.
   Precondition: Must be logged in.
   Sub-tasks:
   I. view course list
   II. select Course
   III. change information
   IV. lock course information

2. Generate Result Sheet
   Goal: Generate result sheet of students with marks obtained in the respective teacher's course.

Precondition: Must be logged in.
Sub-tasks:

I. input marks obtained by individual students
II. generate Result sheet
III. lock result sheet

3. Take Supplementary Exam
   Goal: Provide supplementary Exam results
   Precondition: Must be logged in. Exam result sheet must be available.
   Sub-tasks:
   I. set-up a criteria for supplementary exam.
   II. view supplementary candidates' list
   III. notify candidates
   IV. input marks of supplementary exam
   V. generate supplementary result sheet
   VI. lock supplementary result sheet

4. Upload Result Sheet PDF
   Goal: Upload pdf of result sheet in respective course and make it available for download.
   Precondition:
   I. Must be logged in.
   II. Exam result sheet must be available.
   III. Supplementary result sheet are to be available if any student had been required to appear.
   Sub-tasks:
   I. generate Result Sheet PDF
   II. generate Supplementary Result Sheet PDF
   III. upload PDF to the server
   IV. make PDF's available for download

- Student: Student has following tasks:

1. View result sheet
   Goal: View report card of semesters completed so far.
   Precondition: Must be logged in.
   Sub-tasks:
   I. invoke generation of report card
   II. download report card

2. View course information
   Goal: View information of all courses in the program.
   Precondition: Must be logged in.
   Sub-tasks:

I. view list of courses
II. view information of a specific course

3. Get notifications
   Goal: Get course allocation and supplementary notification
   Precondition: Must be logged in
   Sub-tasks:
   I. get course allocation notification
   II. get supplementary notification

- Receptionist: The tasks of Receptionist are given below:

1. Update payment status
   Goal: Update payment status of applications
   Precondition: Must be logged in
   Sub-tasks:
   I. retrieve applications
   II. update payment status of applications

2. View candidate information
   Goal: View information of an application
   Precondition: Must be logged in
   Sub-tasks:
   I. retrieve application
   II. view details of application

4.2 Interface Design Steps

Interface design steps are shown in figure 184.

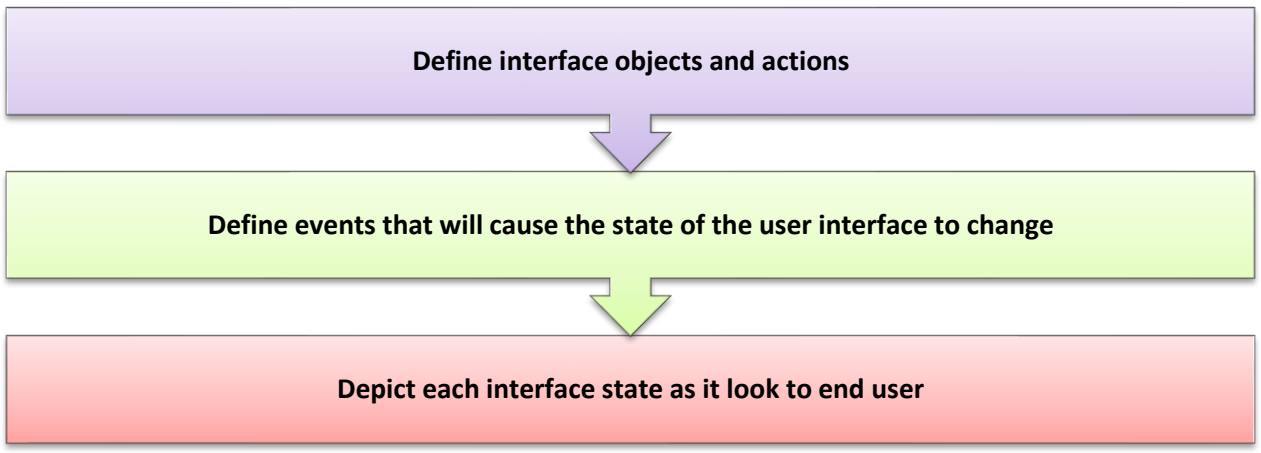


Figure 184 Interface Design Steps

4.2.1 Define Interface Objects and Actions

We identified following objects and actions for the user interface.

- External
  - Home

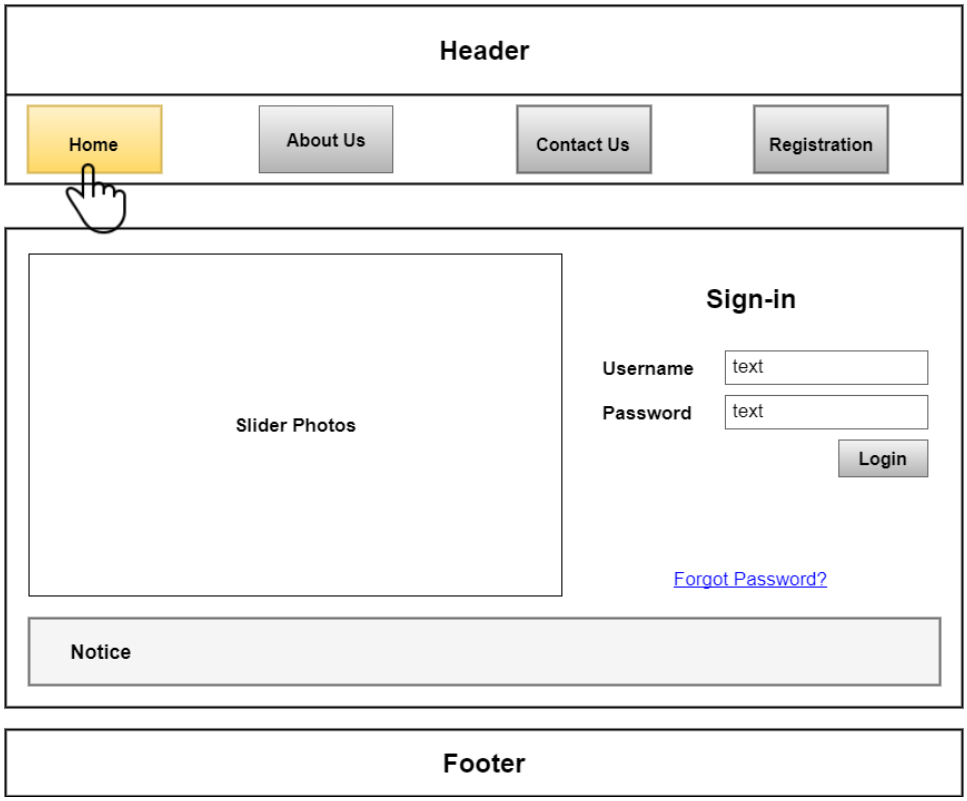


Figure 185 Home

  - About us

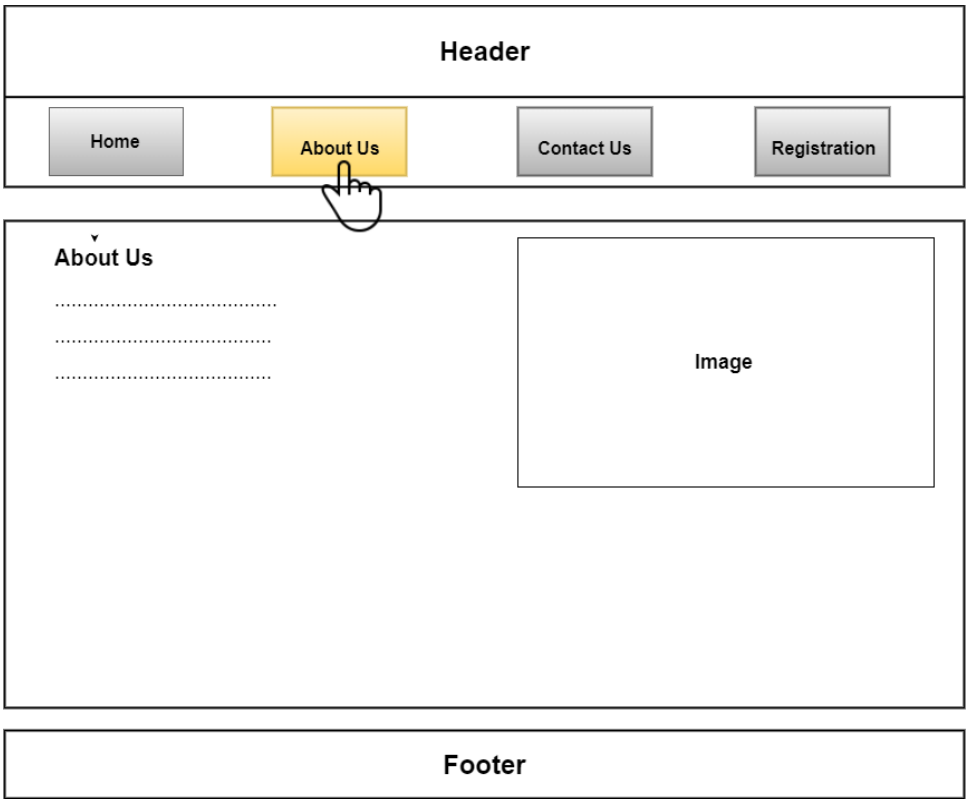


Figure 186 About us

- Contact us

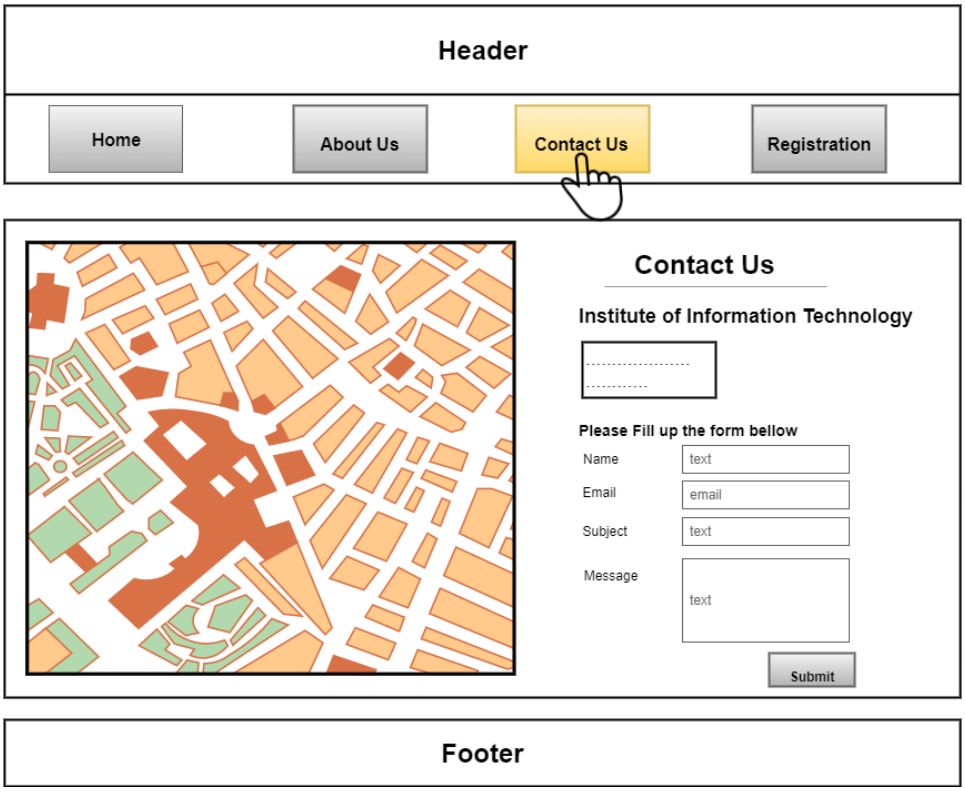


Figure 187 Contact us

- Registration

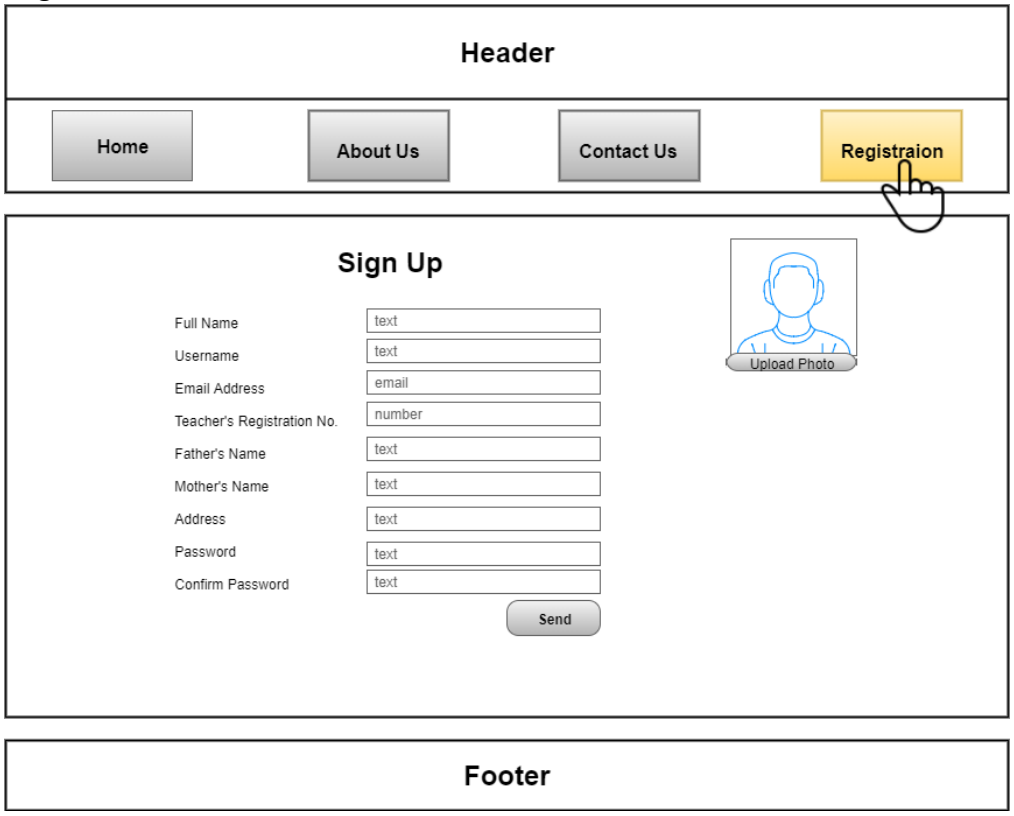


Figure 188 Registration

- Forgot Password

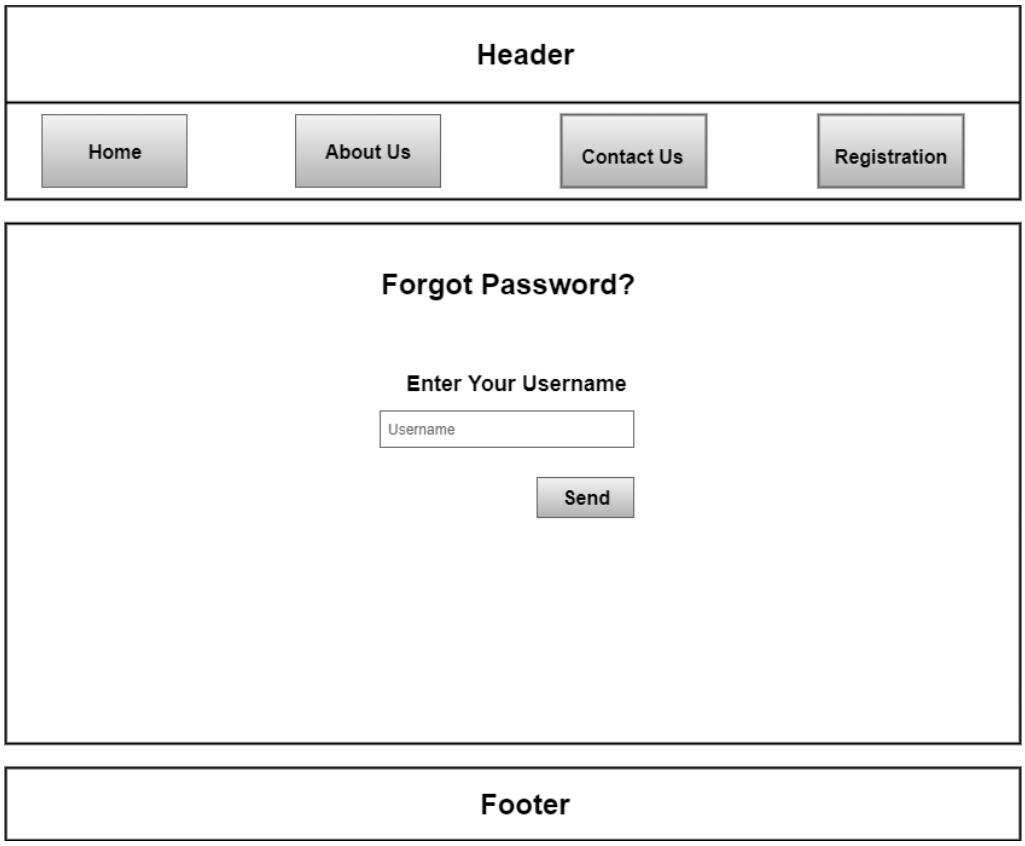


Figure 189 Forgot Password

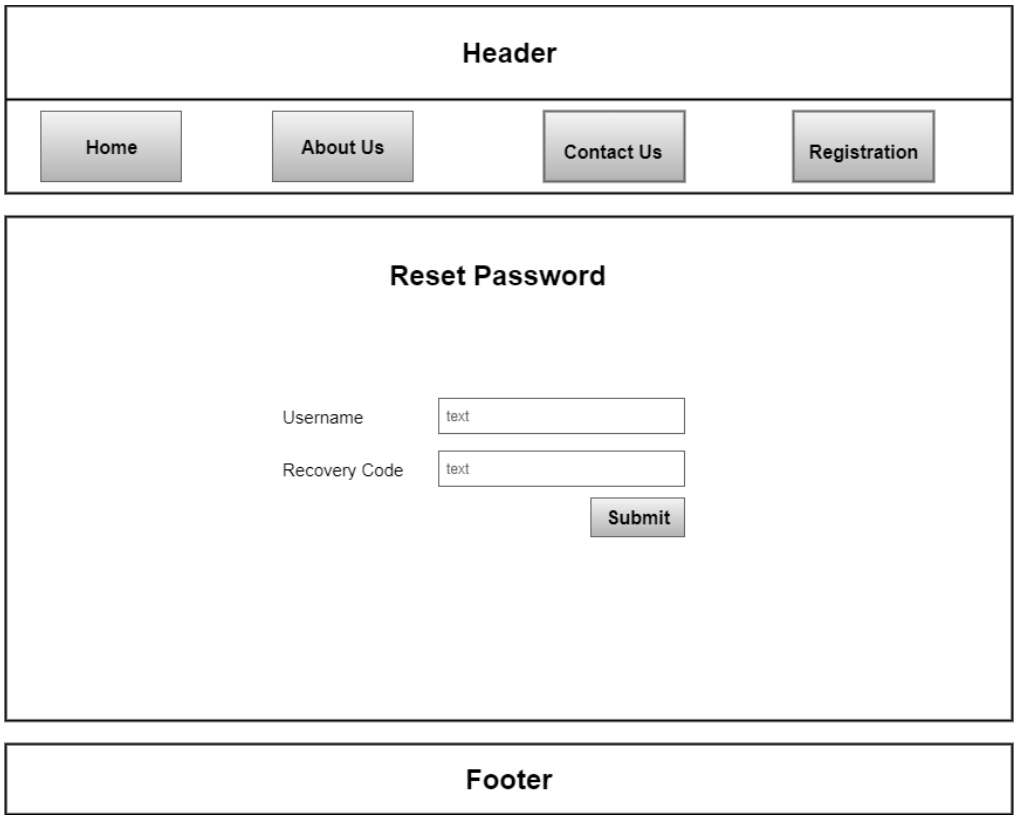


Figure 190 Reset Password

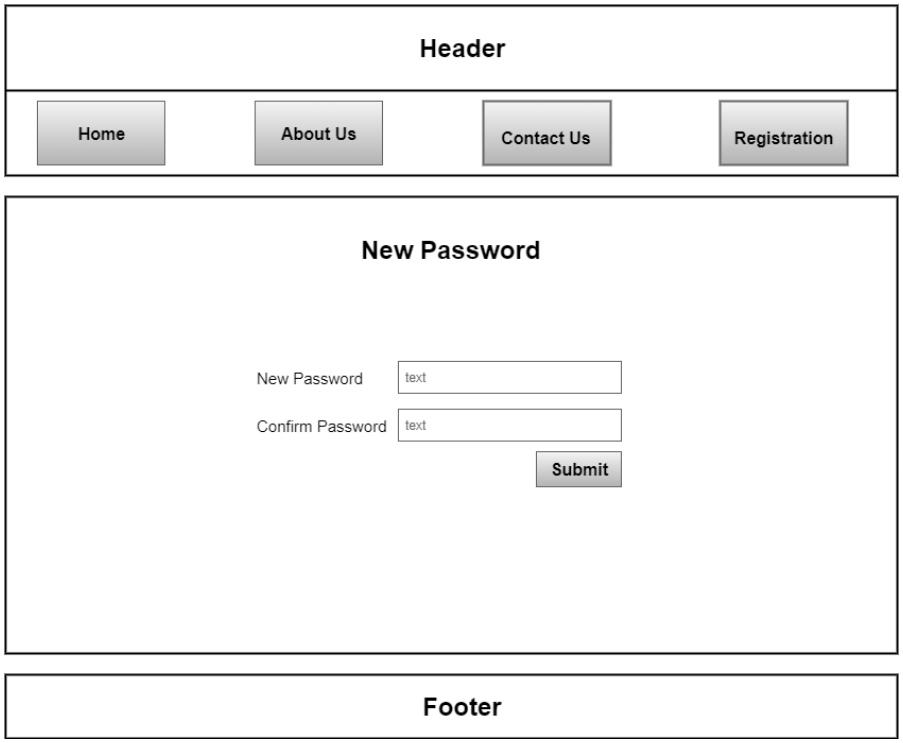


Figure 191 New Password

- Apply For Admission

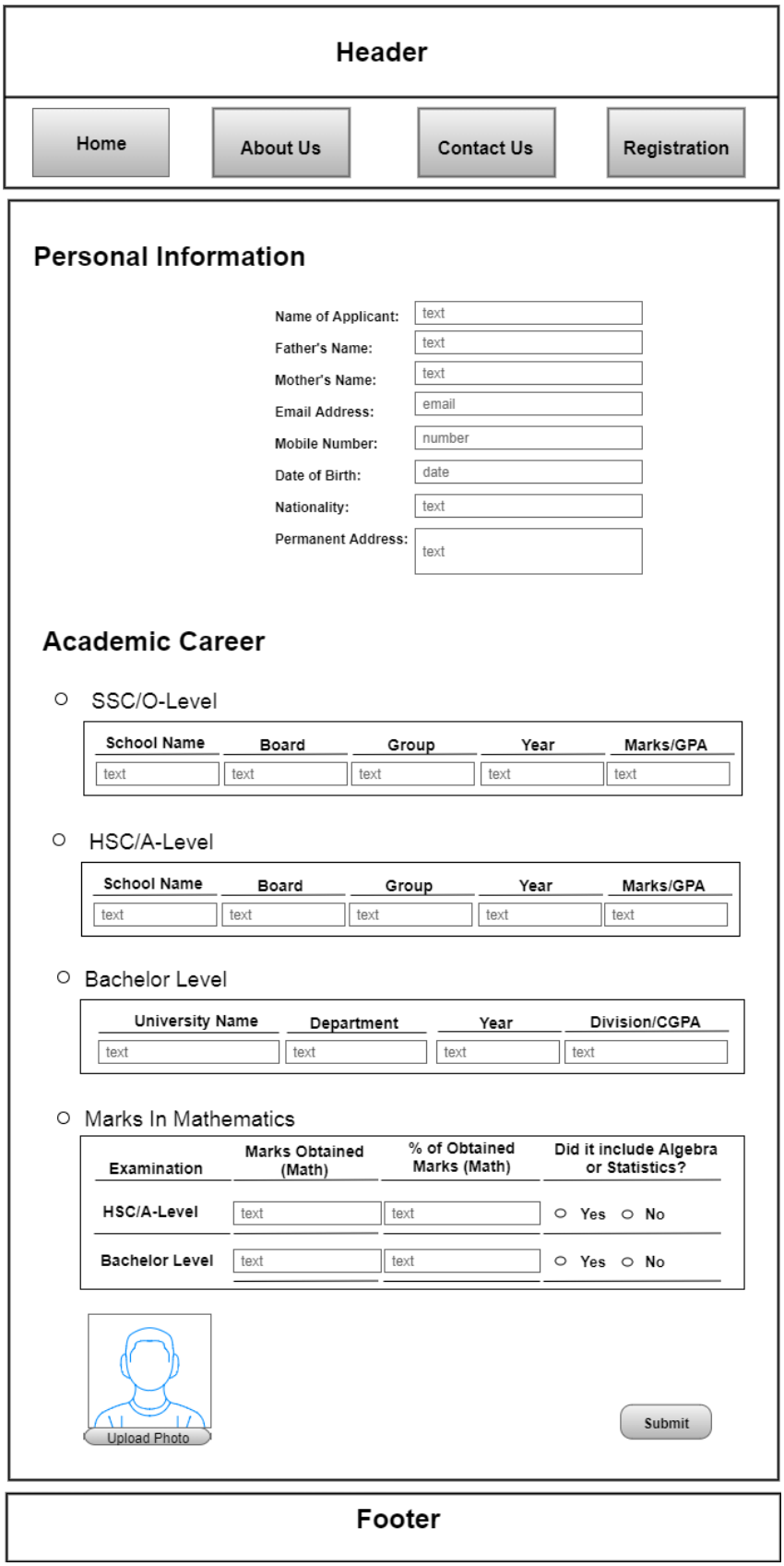


Figure 192 Admission form

- Download Admit card

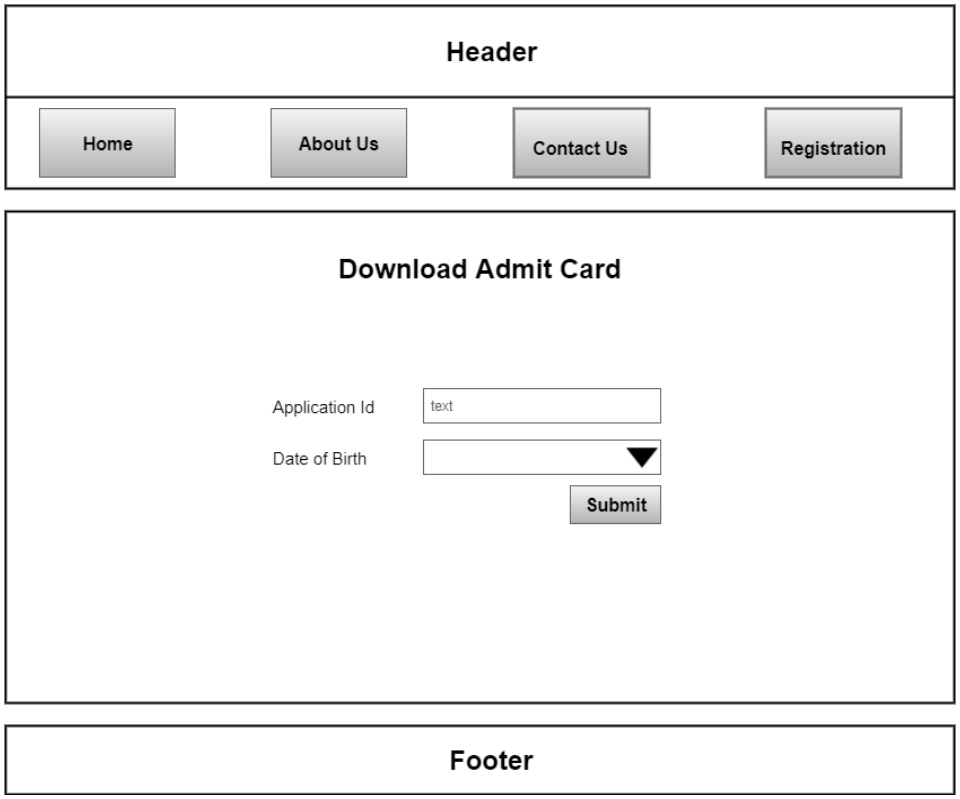


Figure 193 Download Admit Card

- Internal
  - Program Chairperson
    - Profile

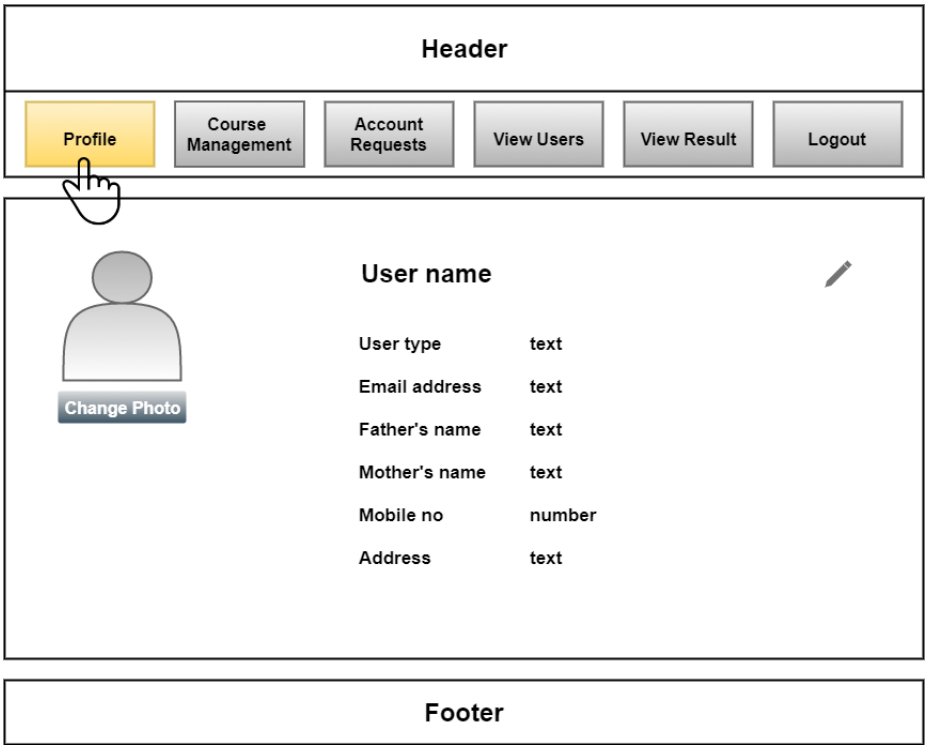


Figure 194 Profile of Program Chairperson

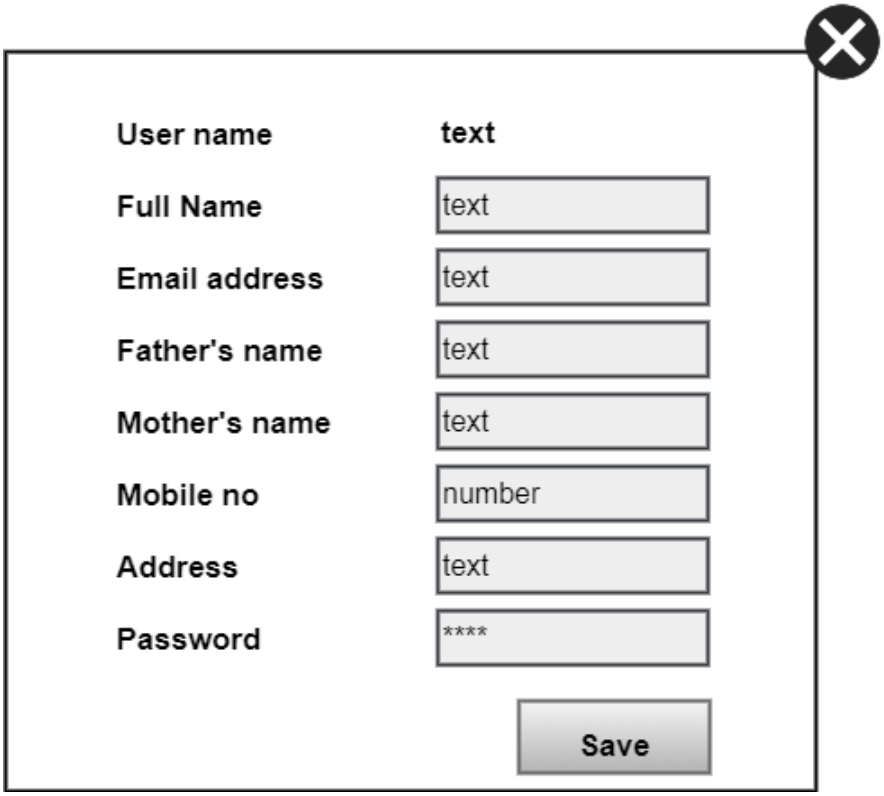


Figure 195 Update Profile

- Course Management

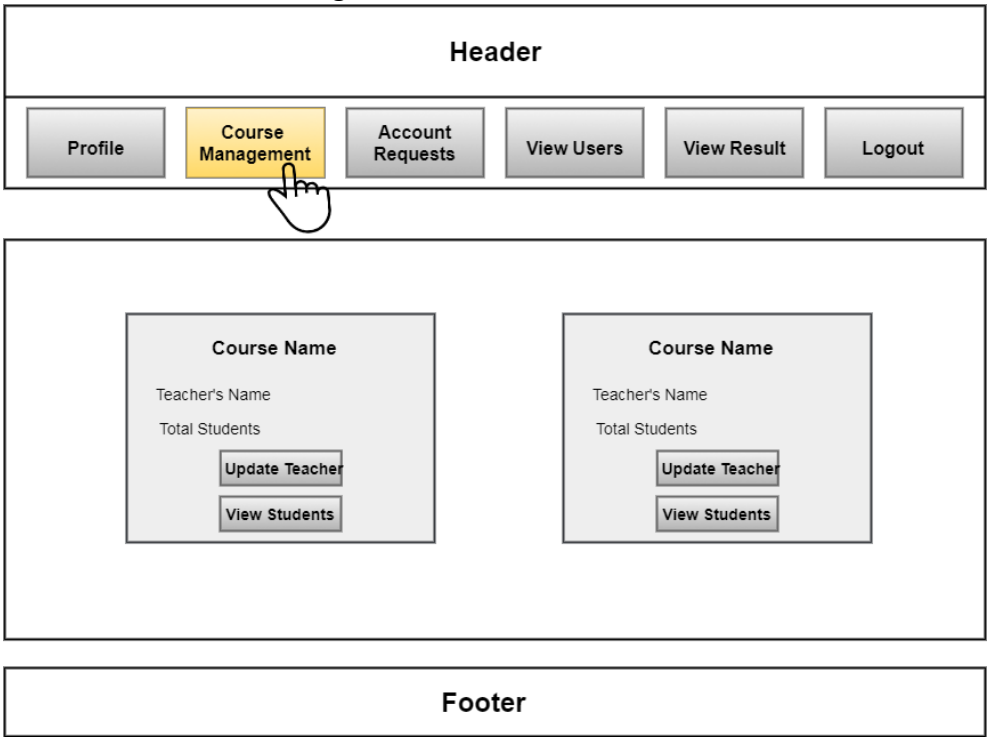


Figure 196 Course Management

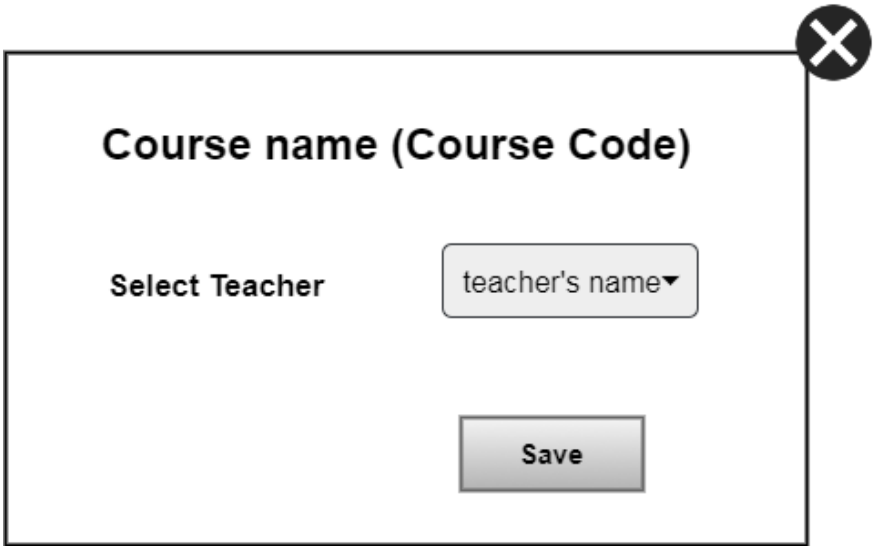


Figure 197 Select Teacher

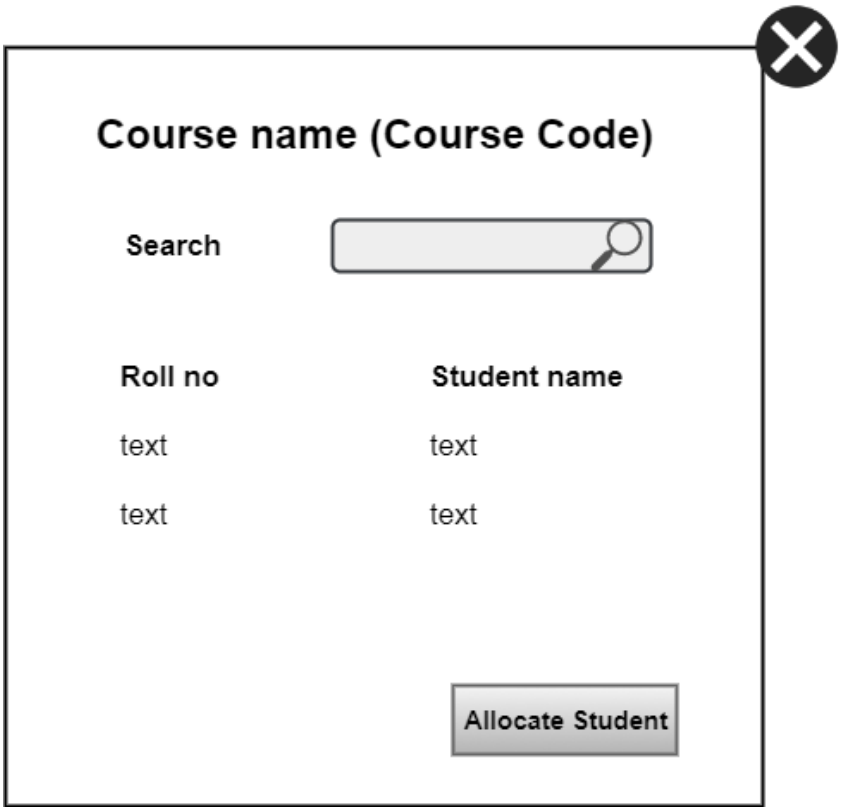


Figure 198 View Students of a Course

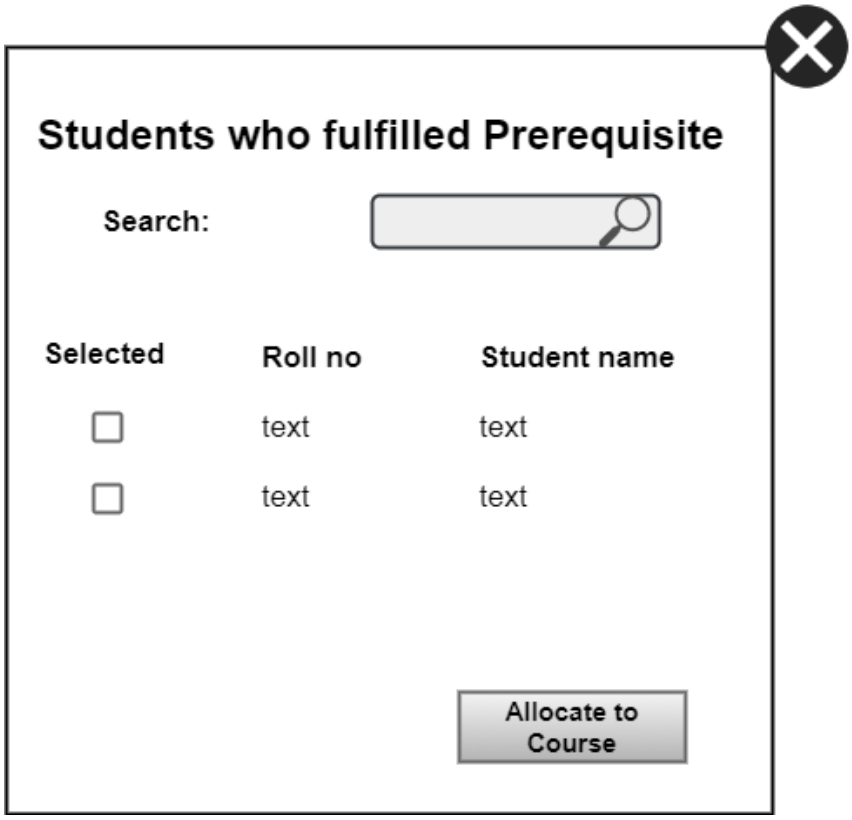


Figure 199 Allocate Students

- Account Requests

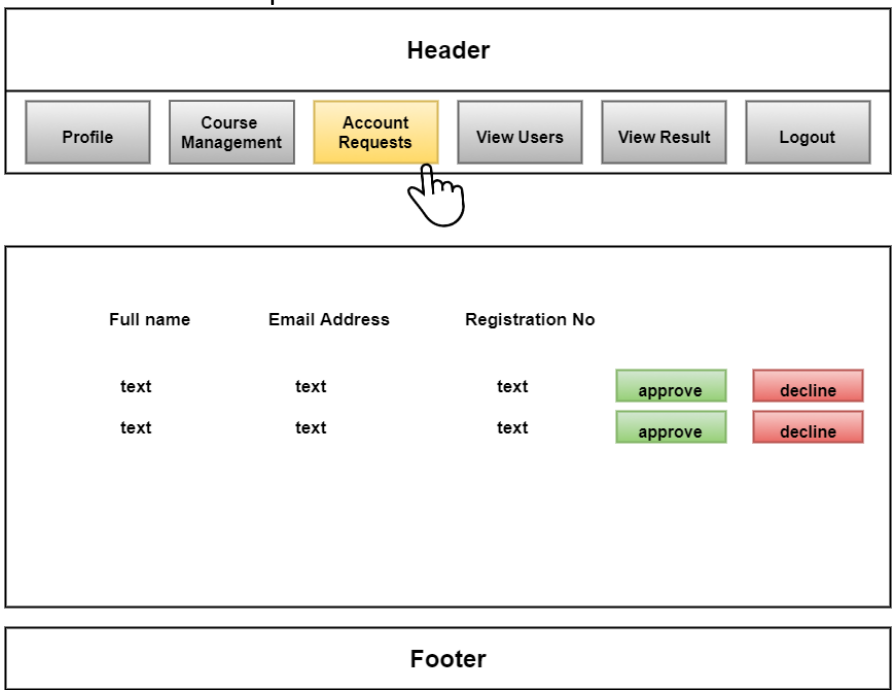


Figure 200 Account Requests

- View Users

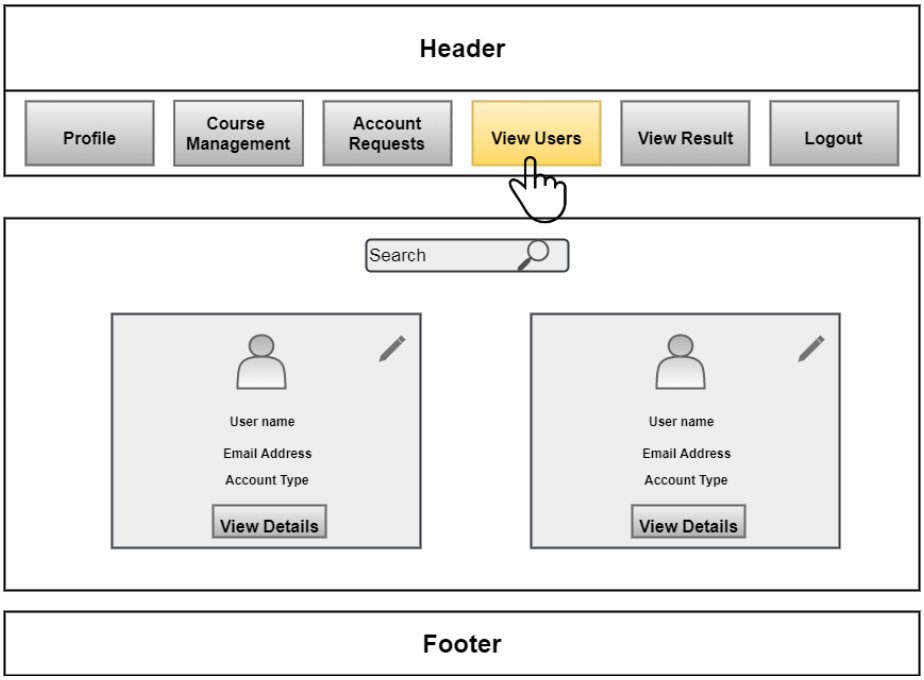


Figure 201 View Users

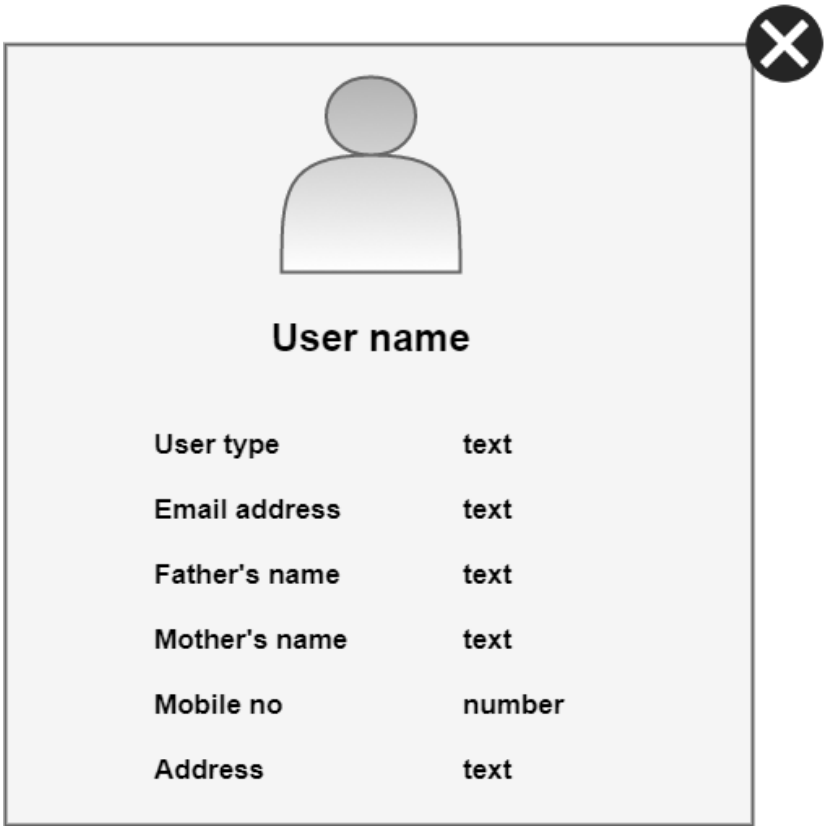


Figure 202 View User Details

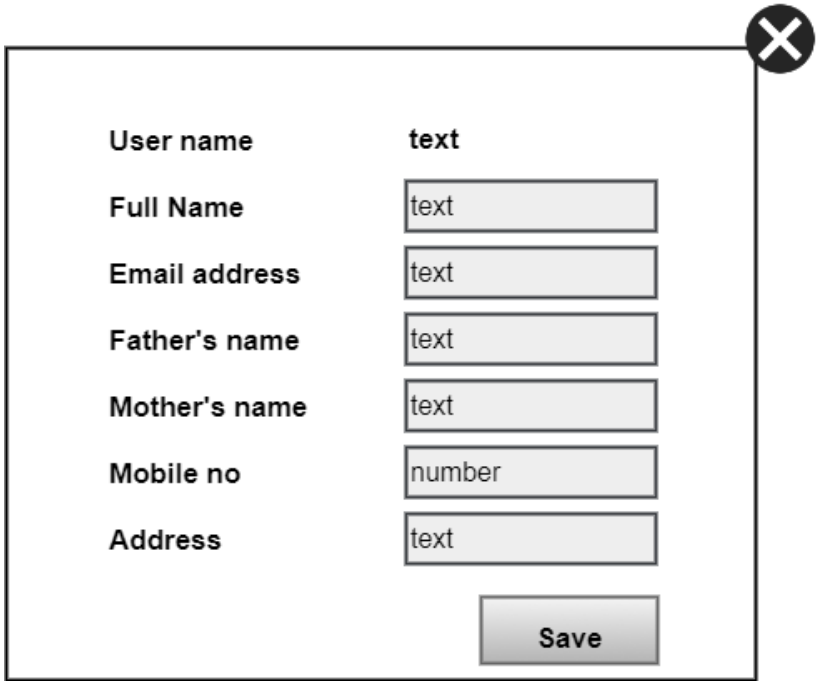


Figure 203 Edit User Information

- View Result
  - Course wise Result

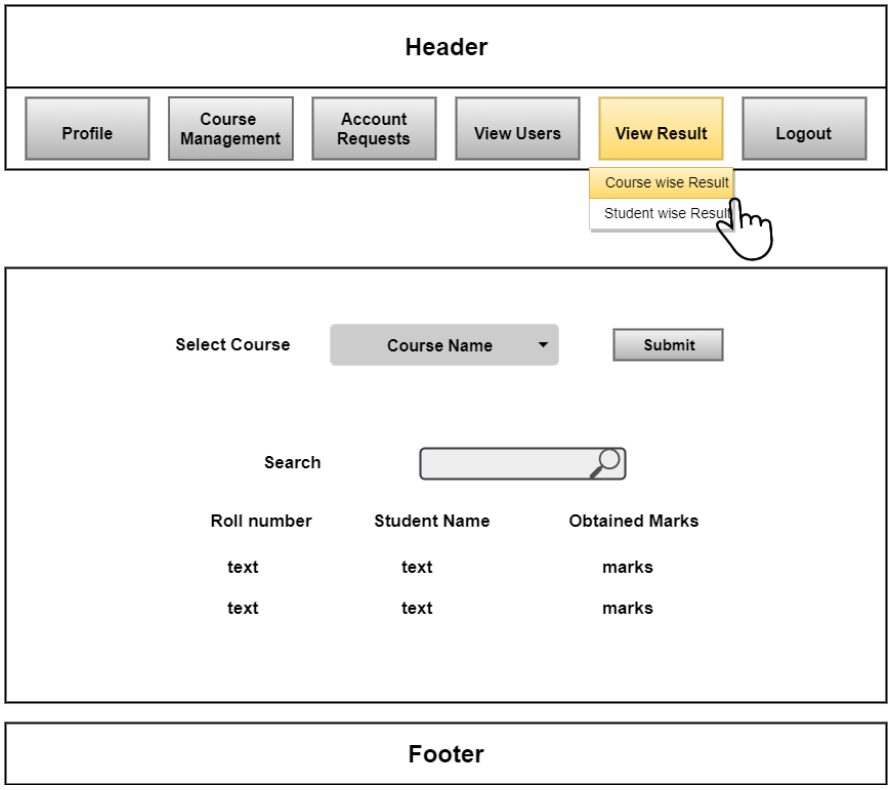


Figure 204 Course wise Result

- Student wise Result

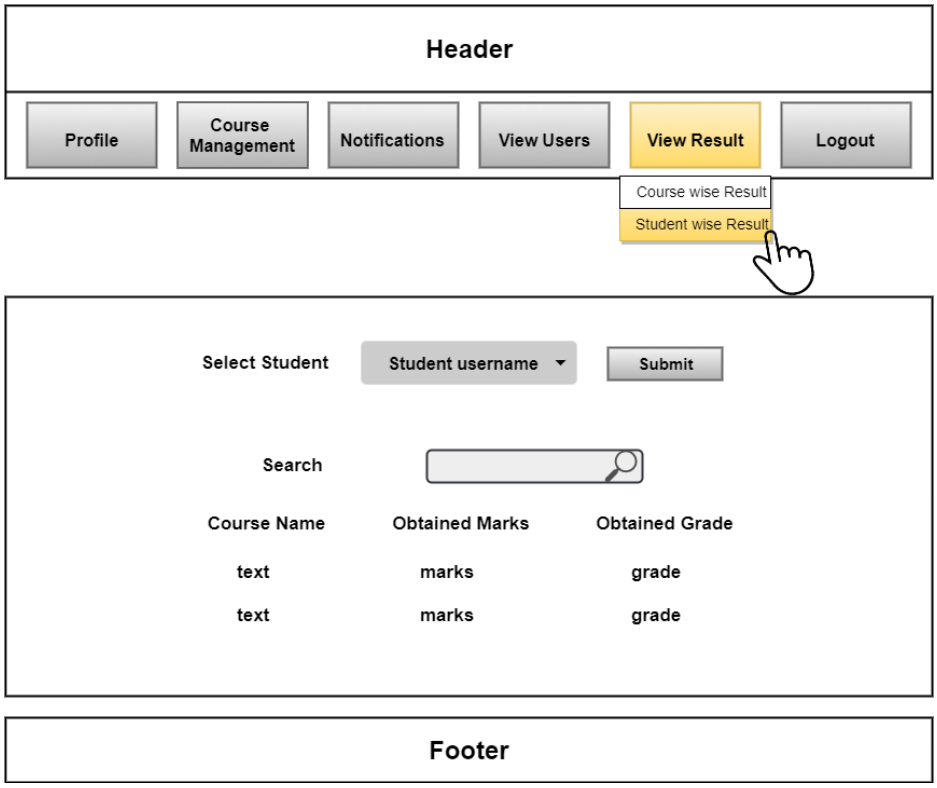


Figure 205 Student wise Result

- Teacher
  - Profile

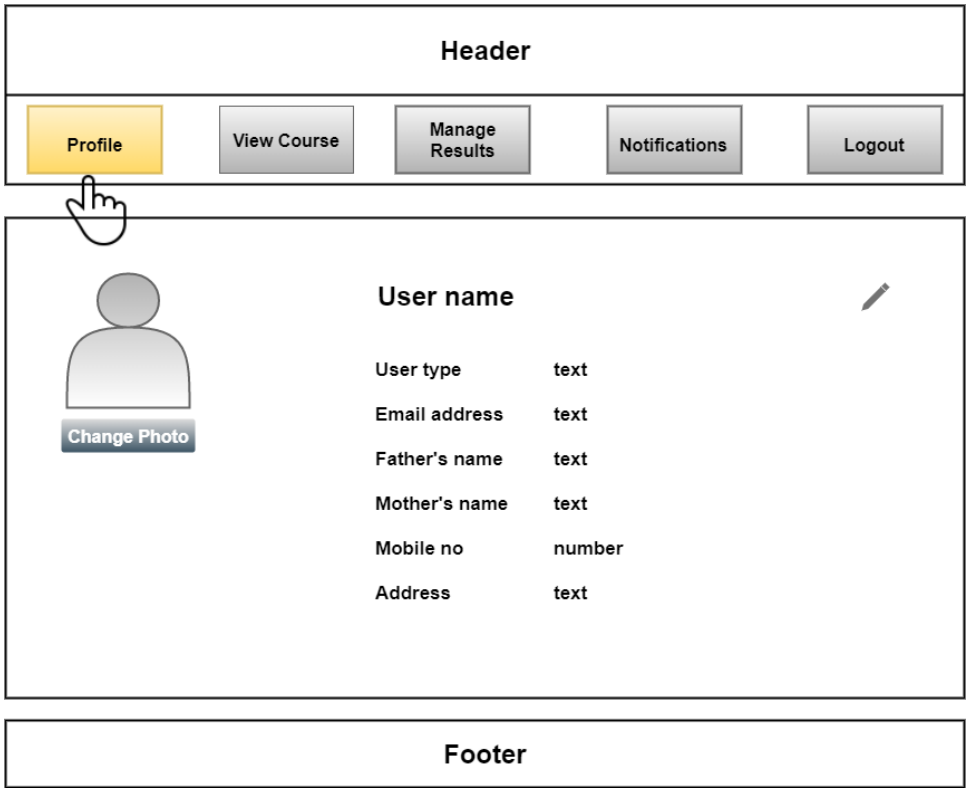


Figure 206 Teacher's Profile

- View Course

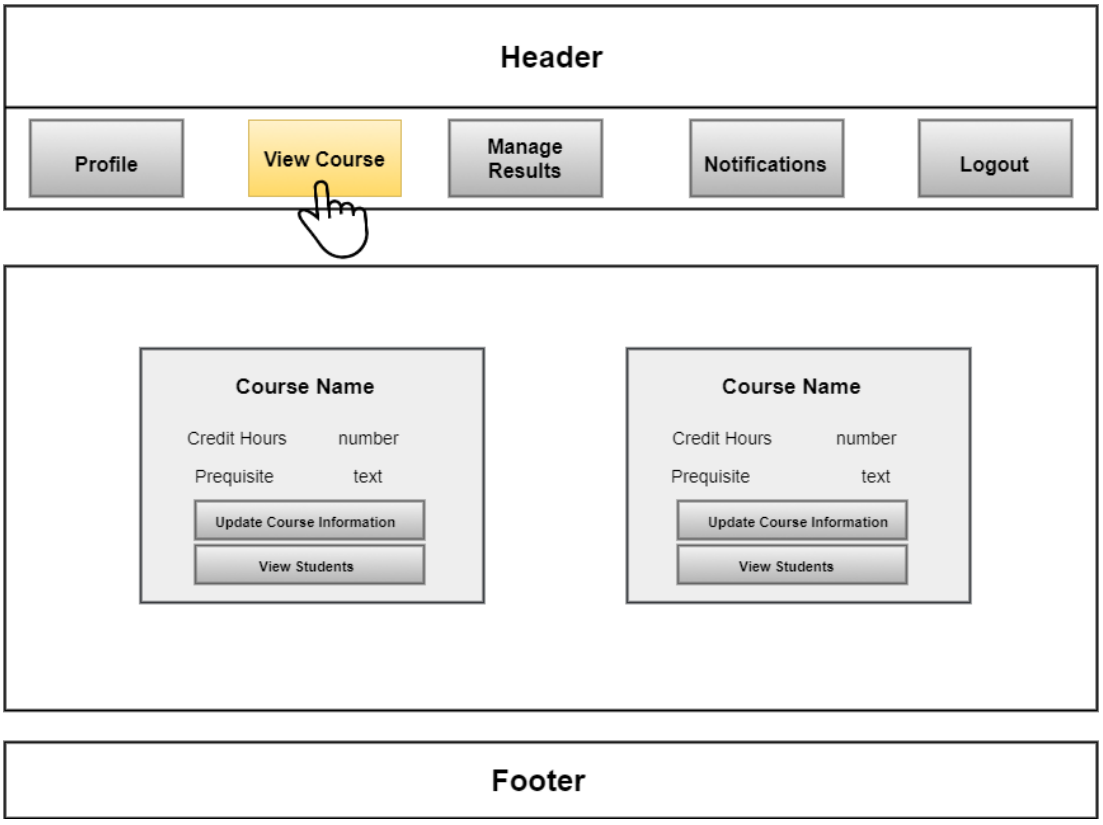


Figure 207 View Courses

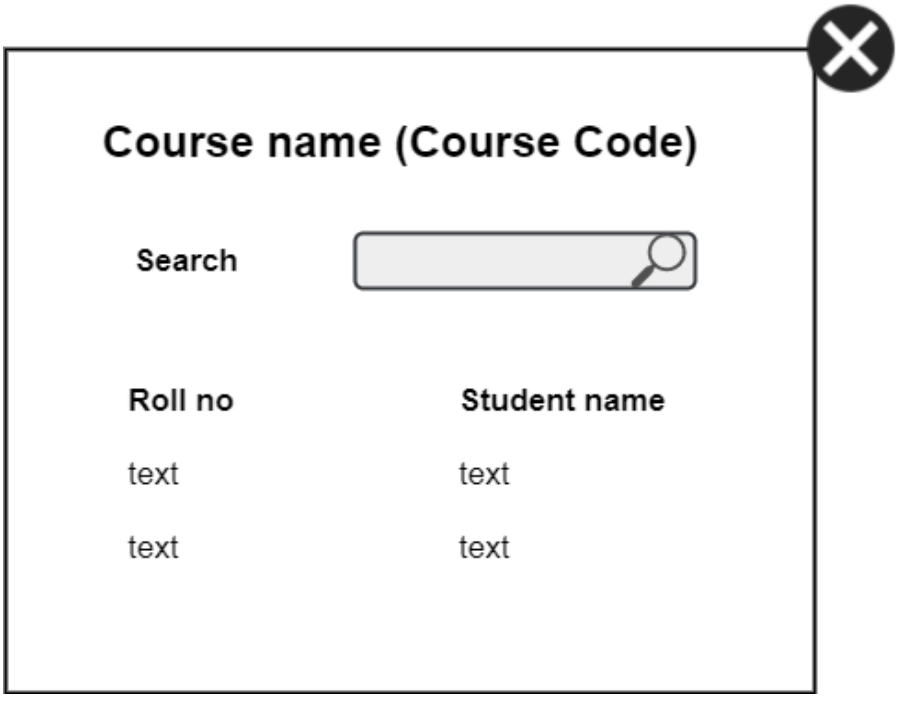


Figure 208 View Students

- Manage Results

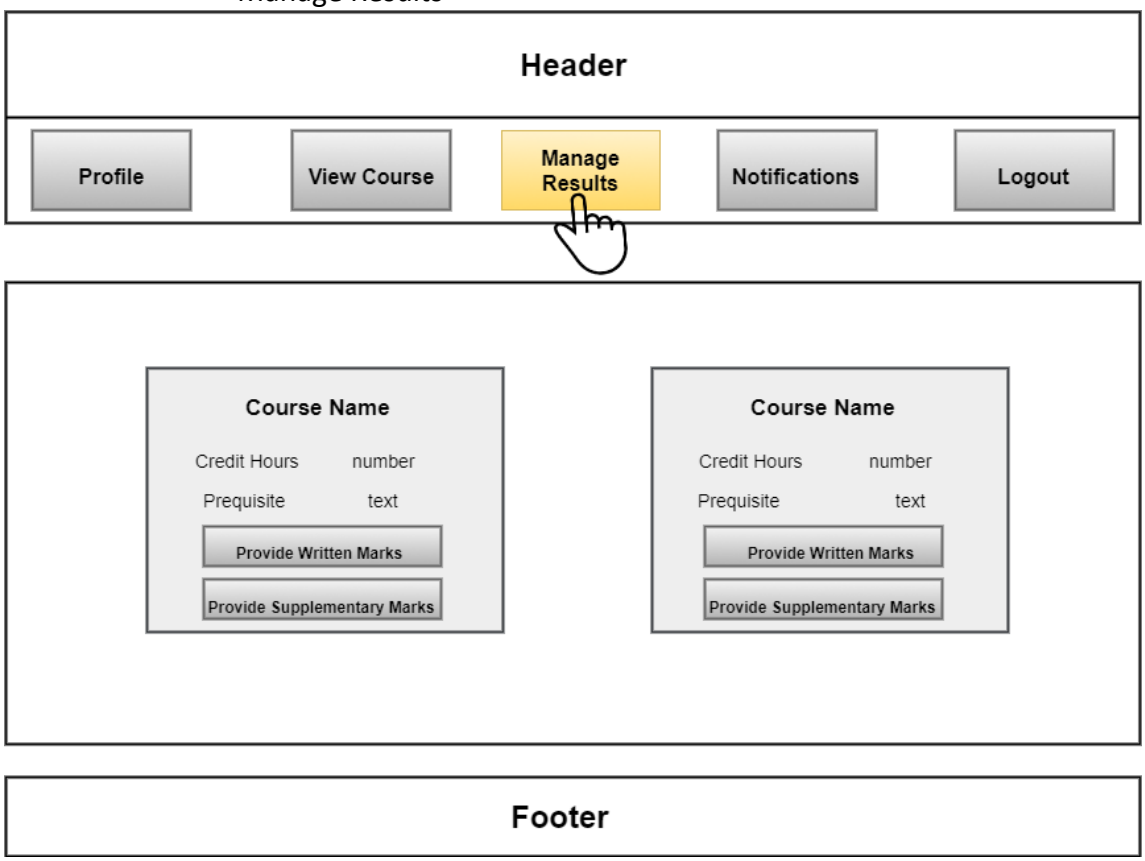


Figure 209 Manage Result

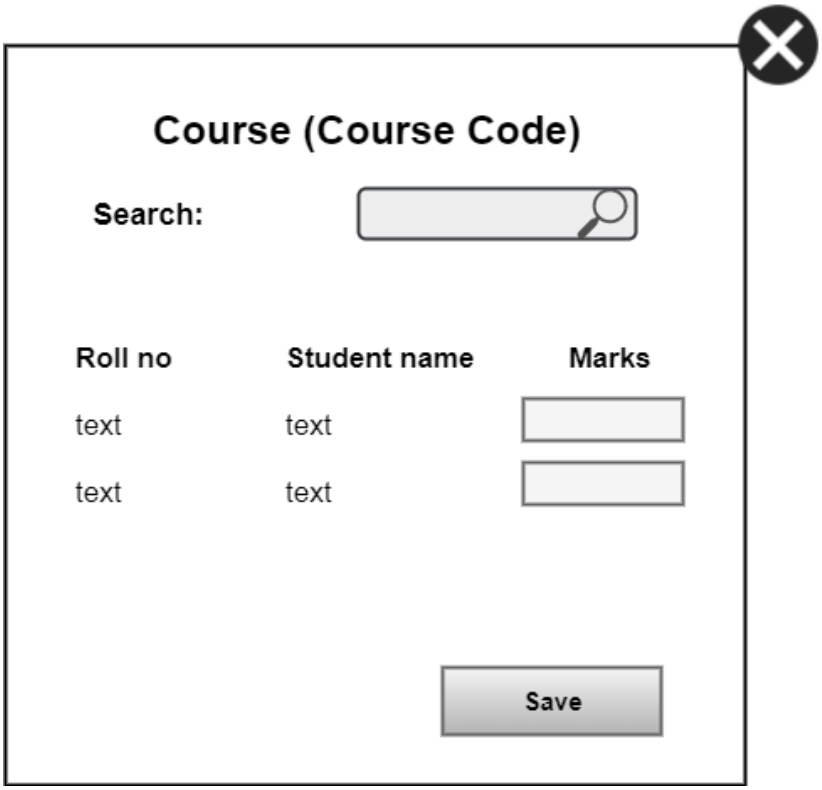


Figure 210 Provide Marks

- Notifications

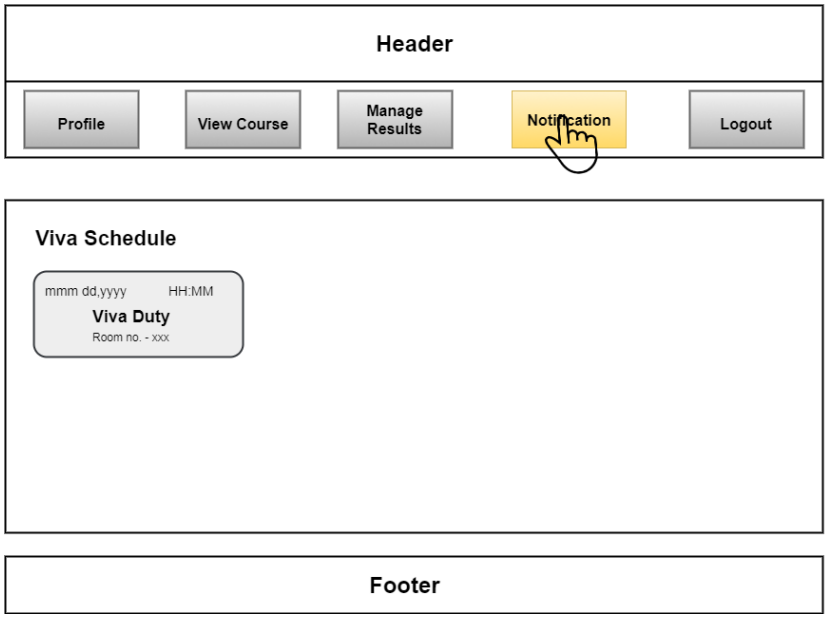


Figure 211 Notifications

- Student
  - Profile

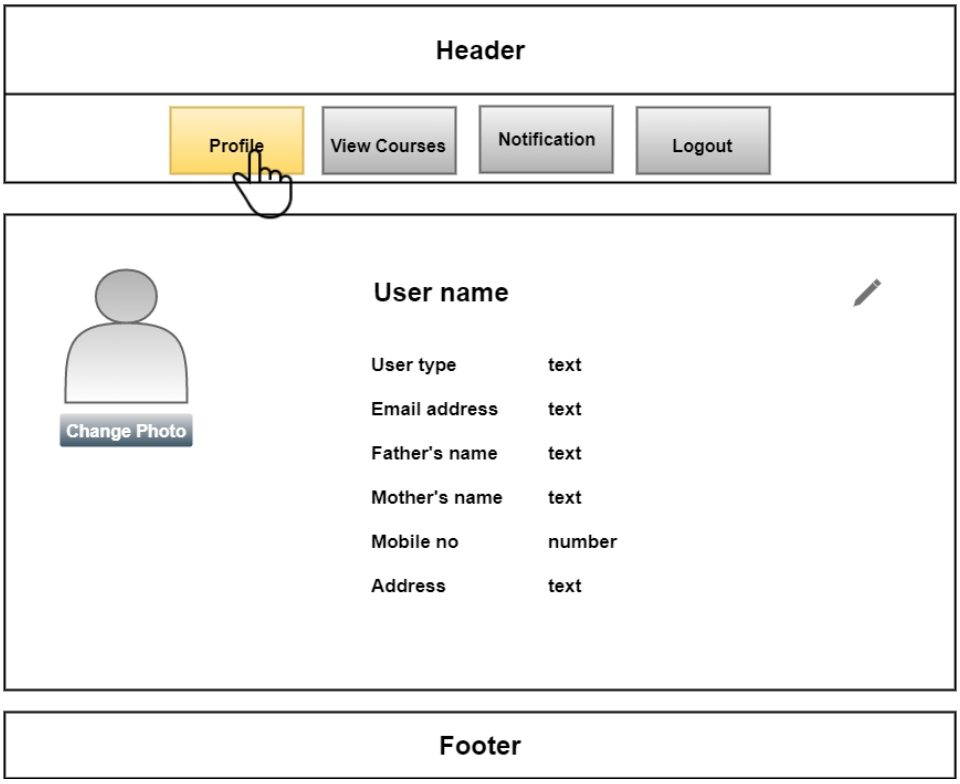


Figure 212 Student Profle

  - Download Report Card

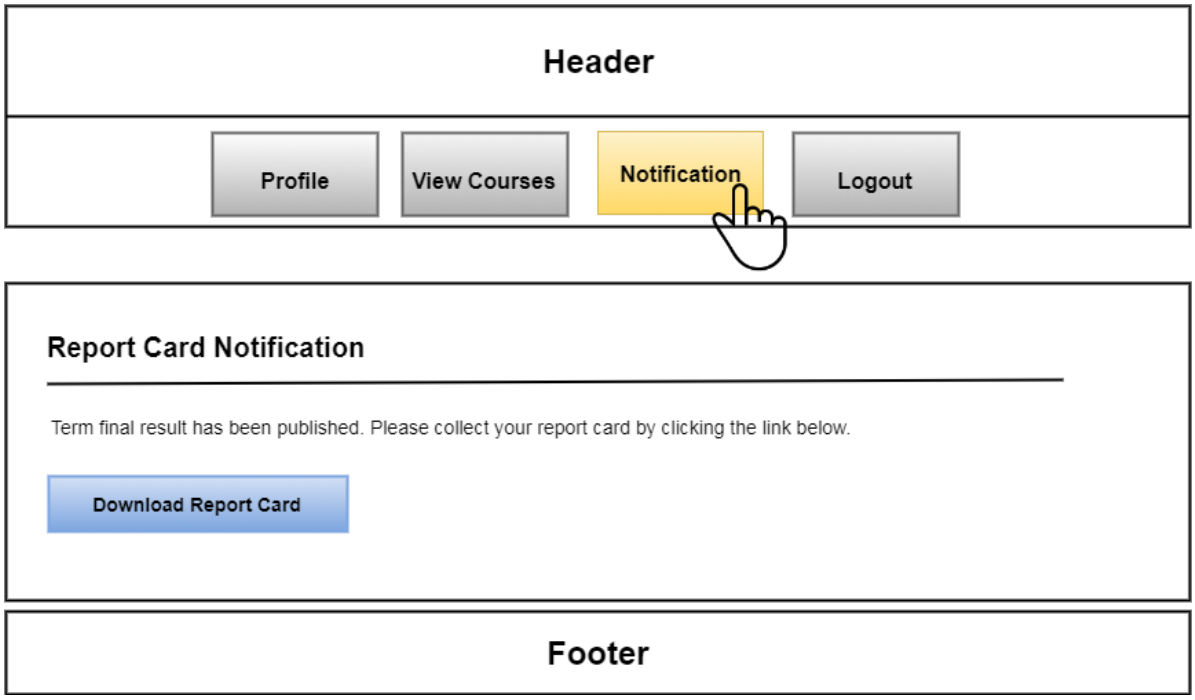


Figure 213 Download Report Card

- View Courses

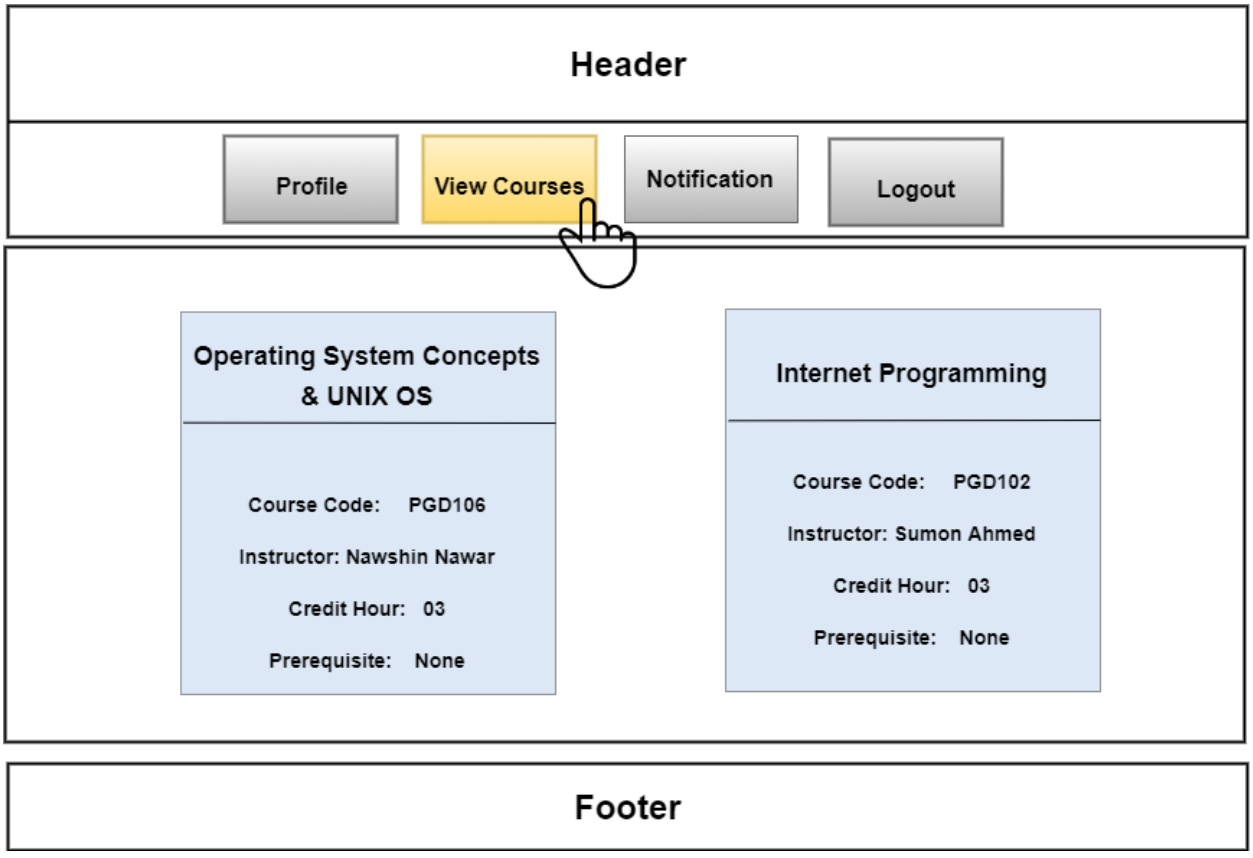


Figure 214 View Courses

- Exam Controller
  - Profile

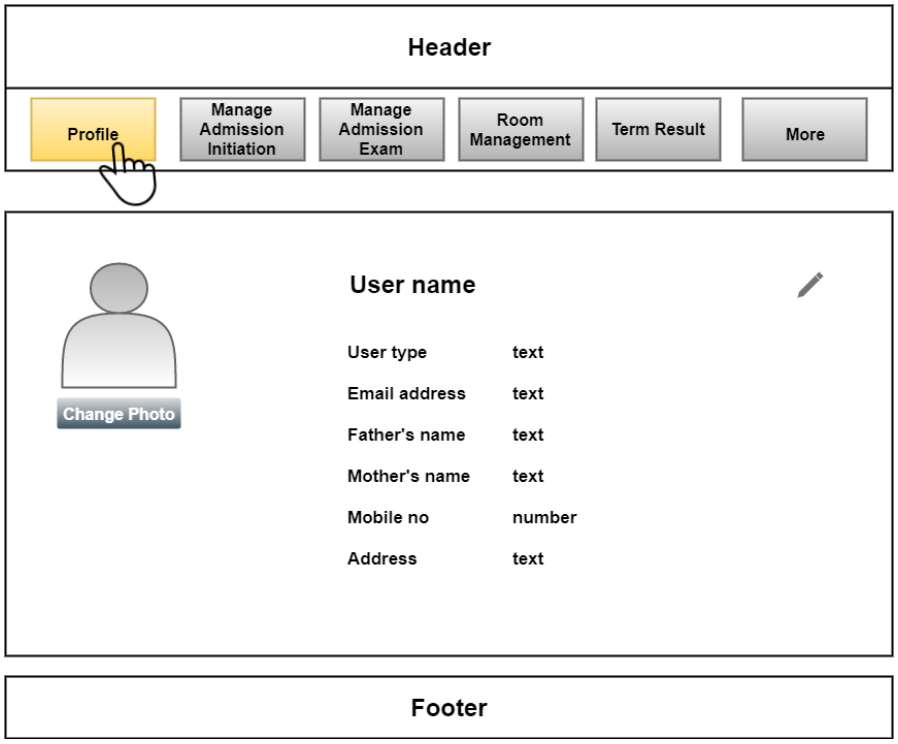


Figure 215 Exam Controller's Profile

  - Manage Admission Initiation

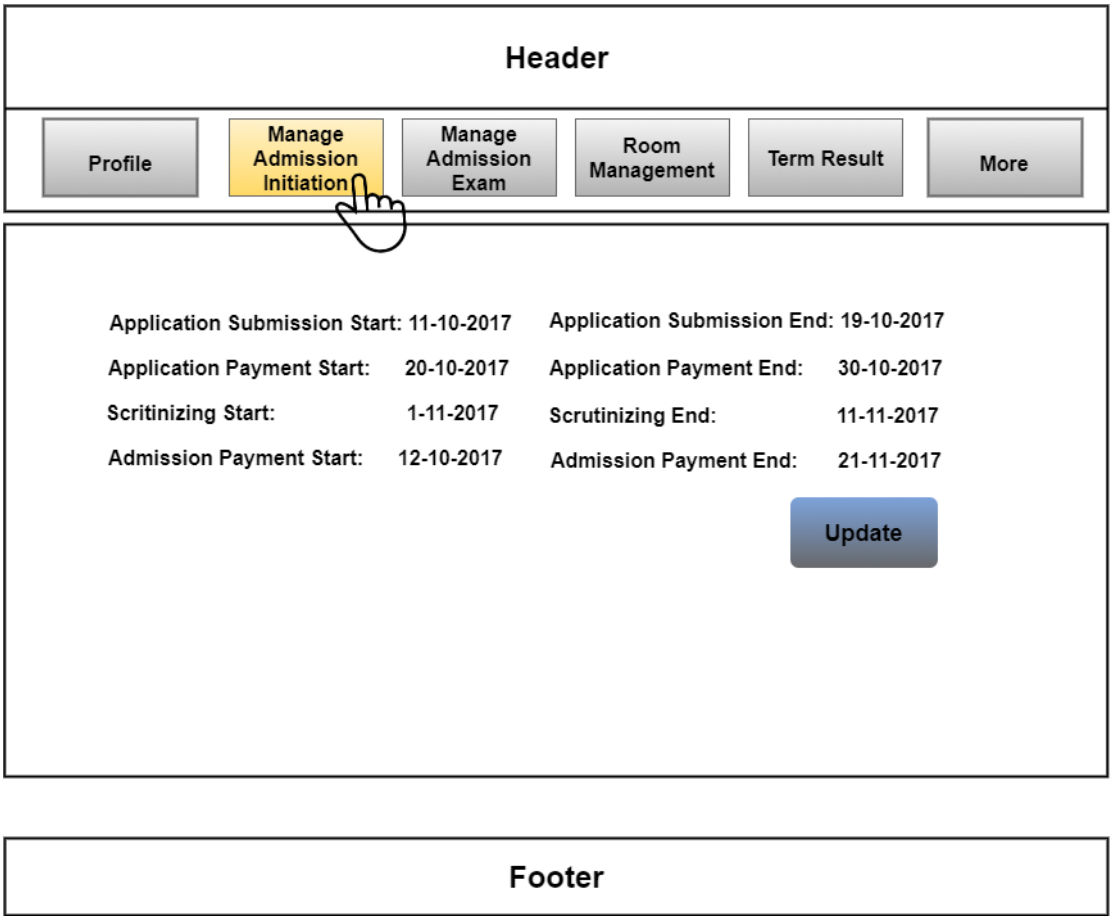


Figure 216 Manage Admission Initiation

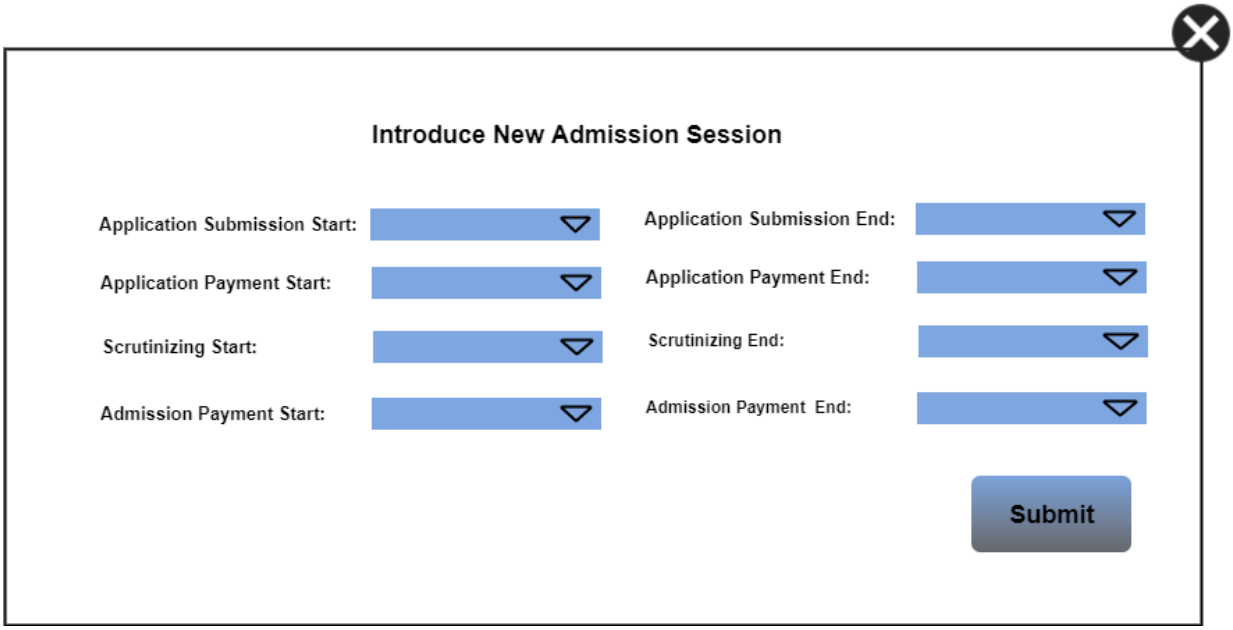


Figure 217 Setting New Admission

- Manage Admission Exam

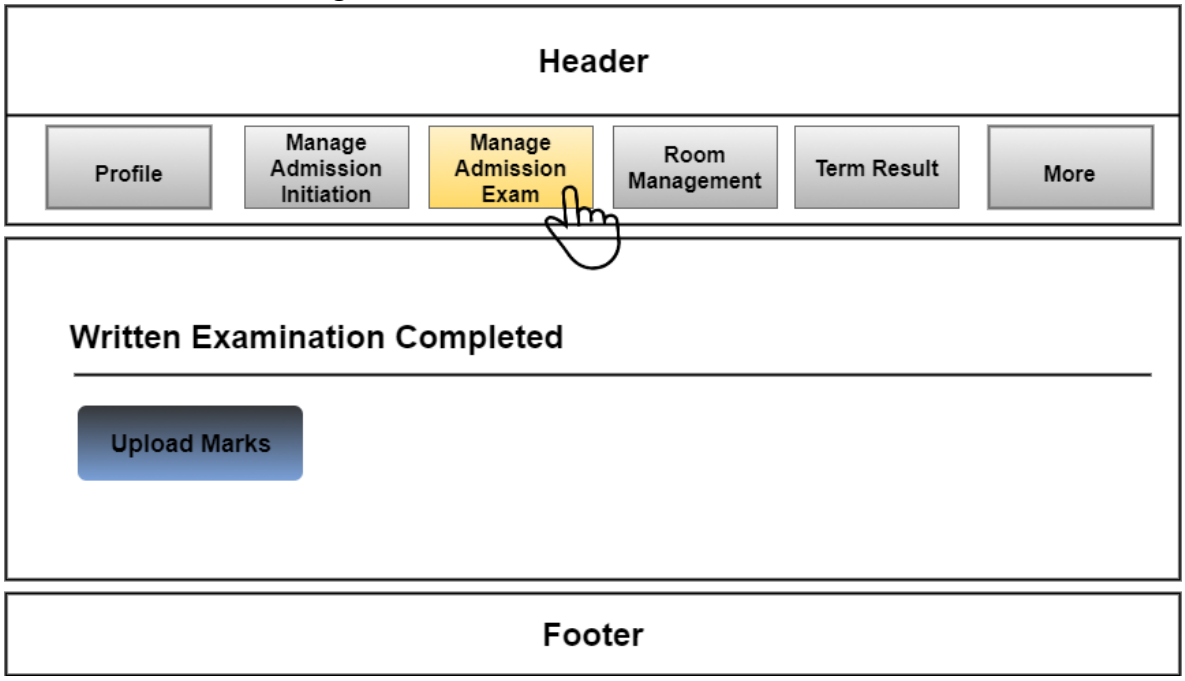


Figure 218 Manage Admission Exam

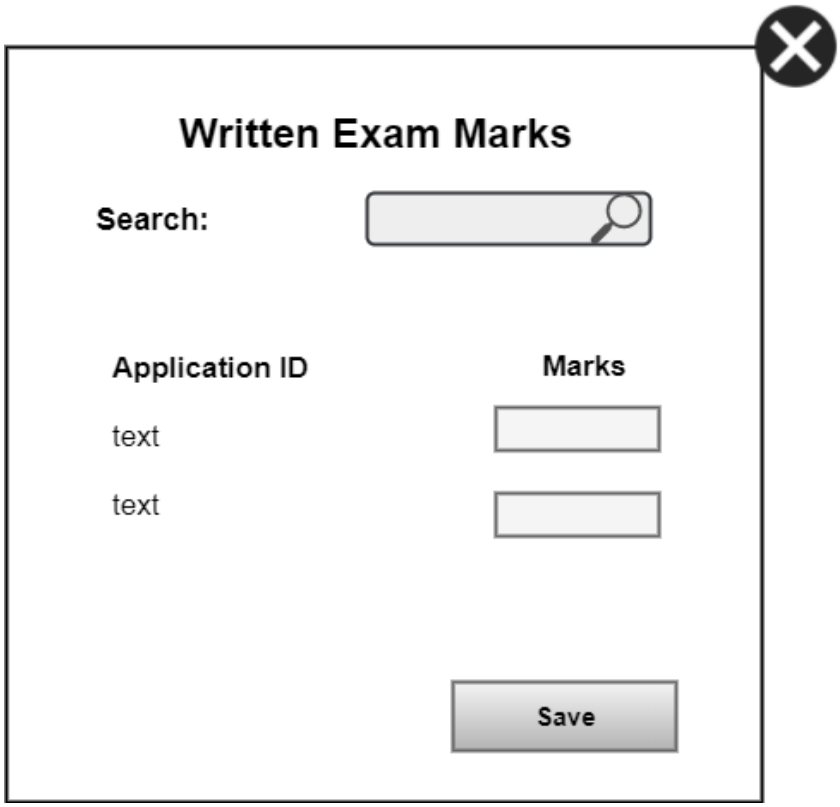


Figure 219 Provide Written Marks

- Room Management

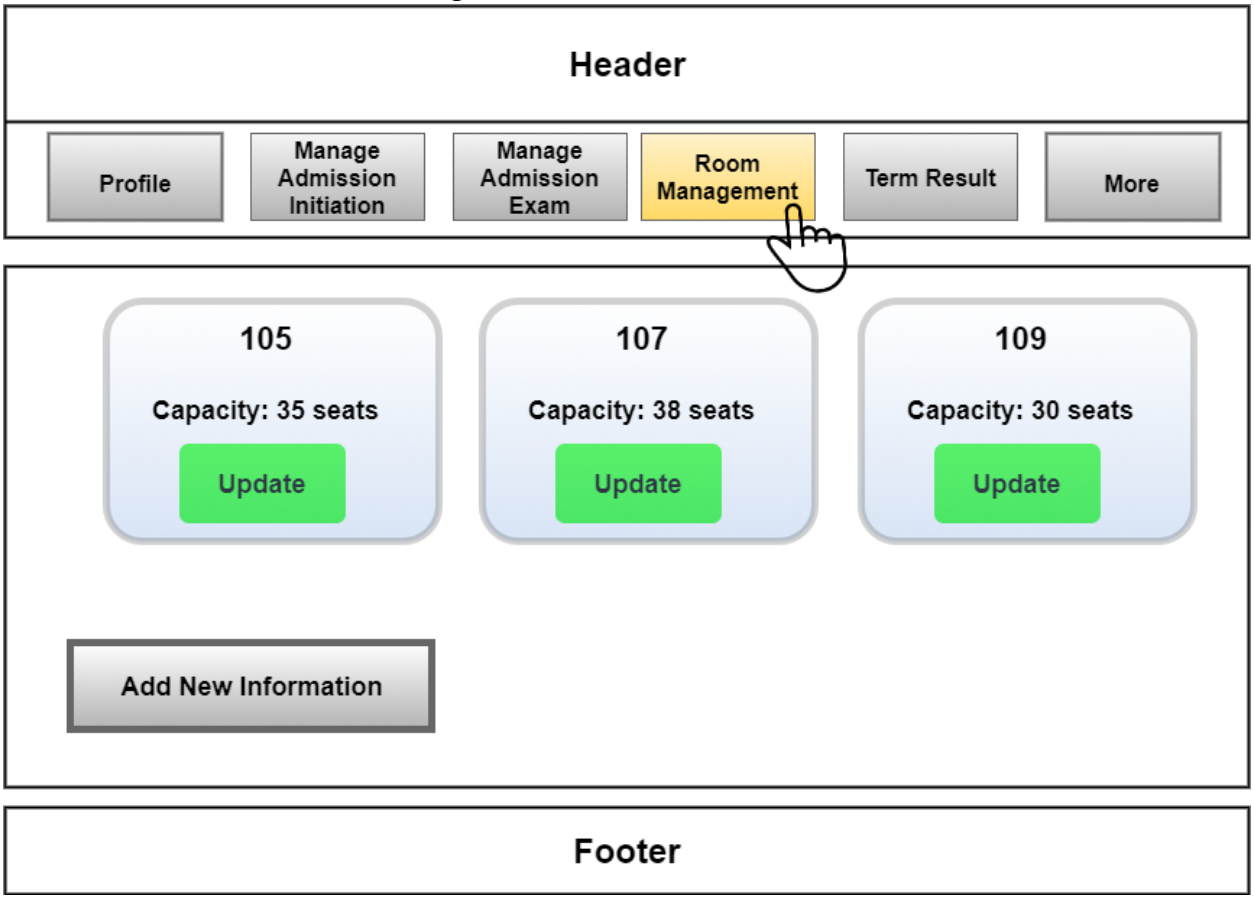


Figure 220 Room Management

- Term Result

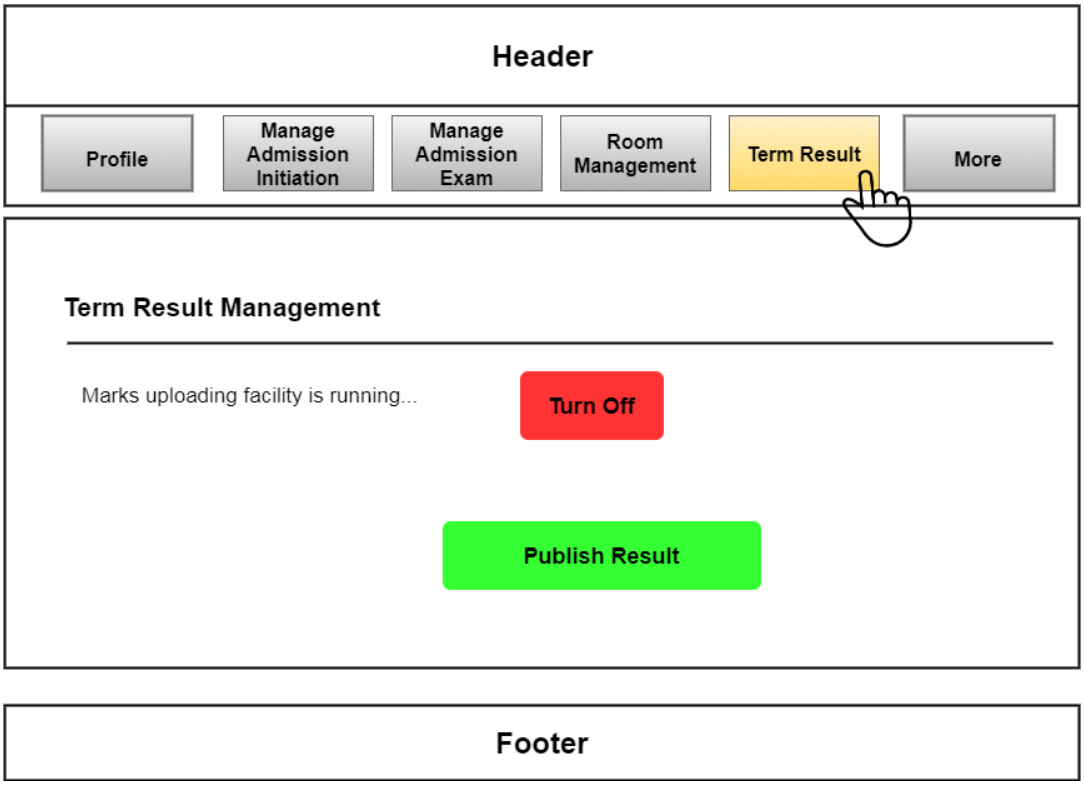


Figure 221 Term Result

- Update Signature

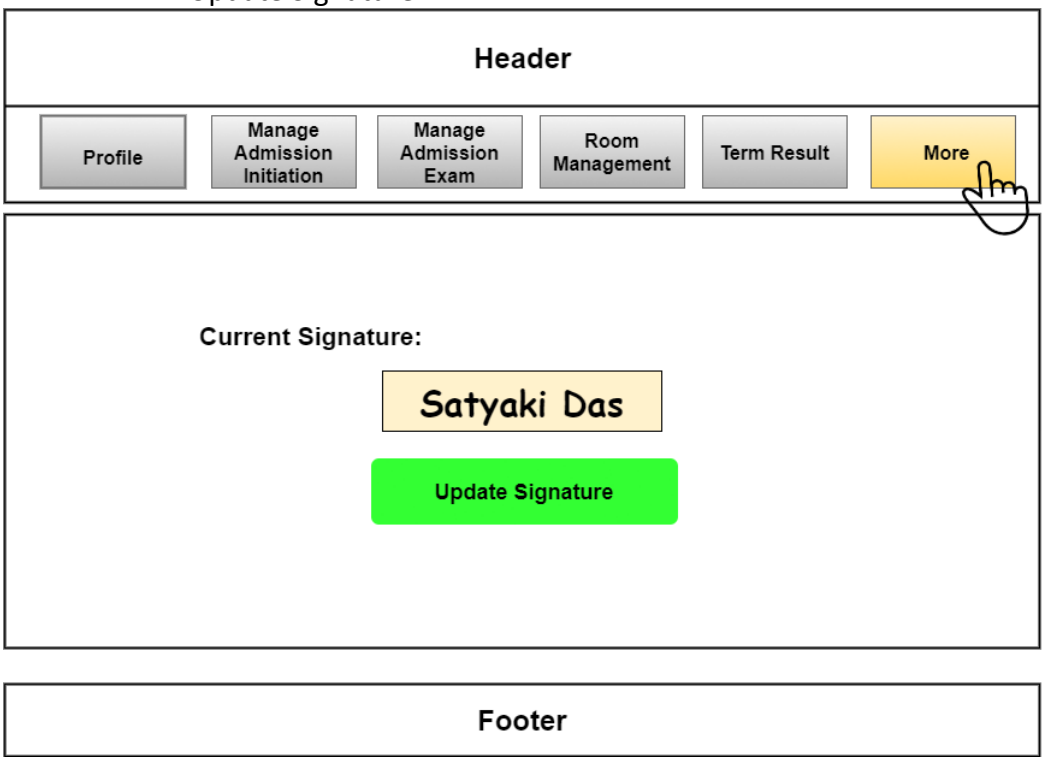
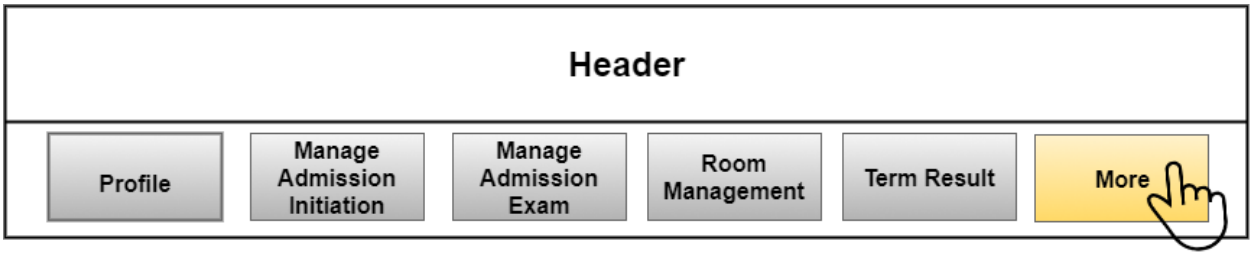
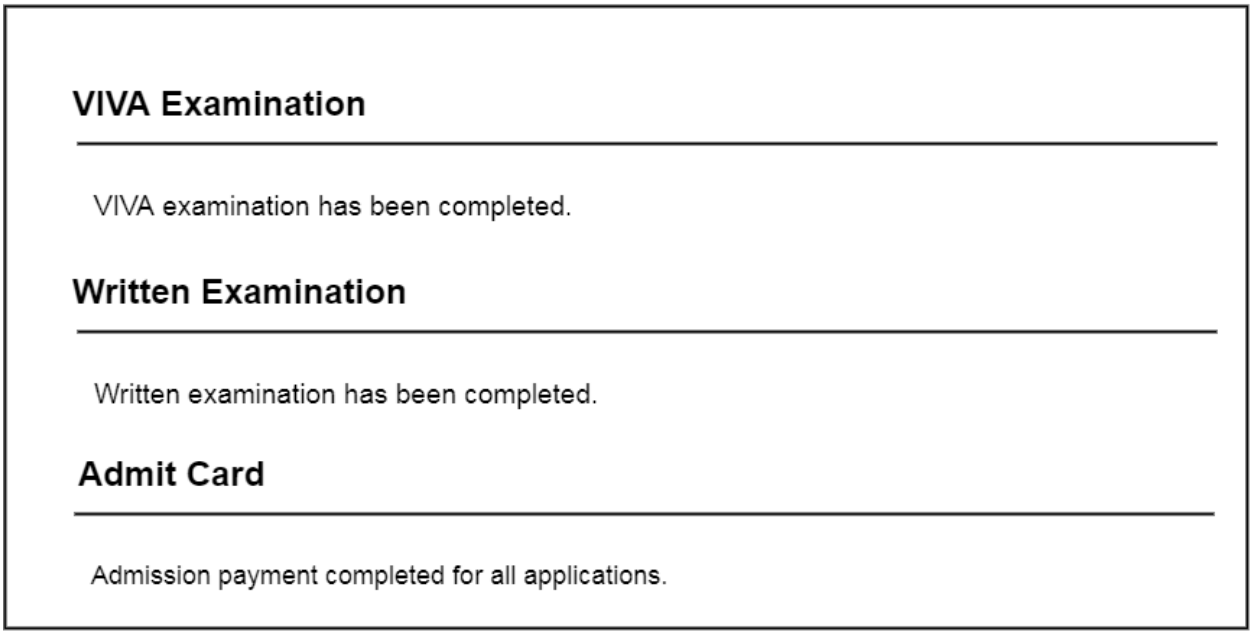


Figure 222 Update Signature





- Receptionist
  - Profile

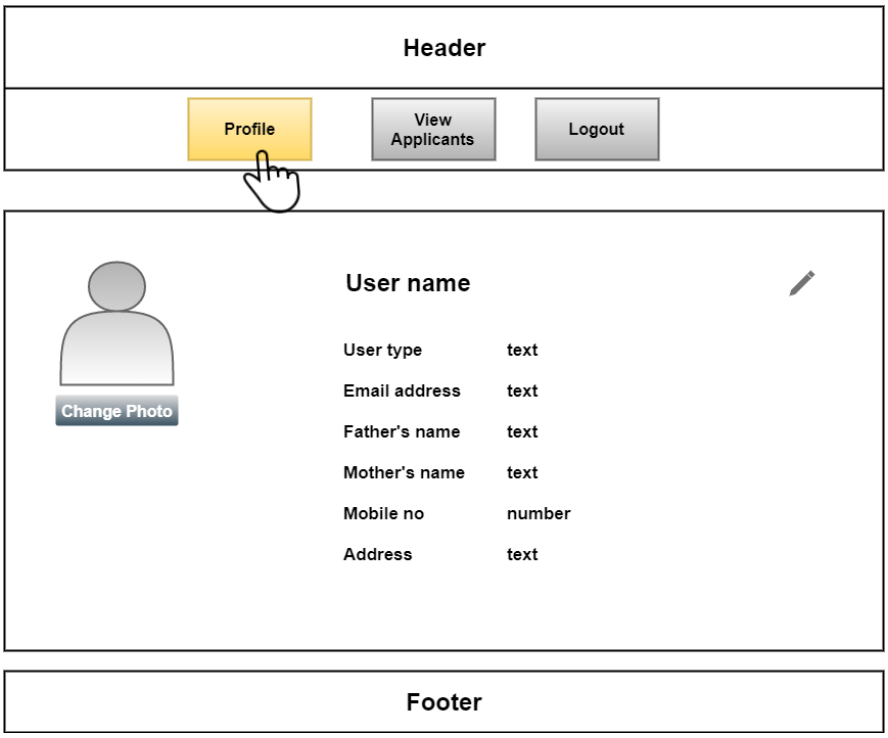


Figure 223 Receptionist Profile

- View Applicants

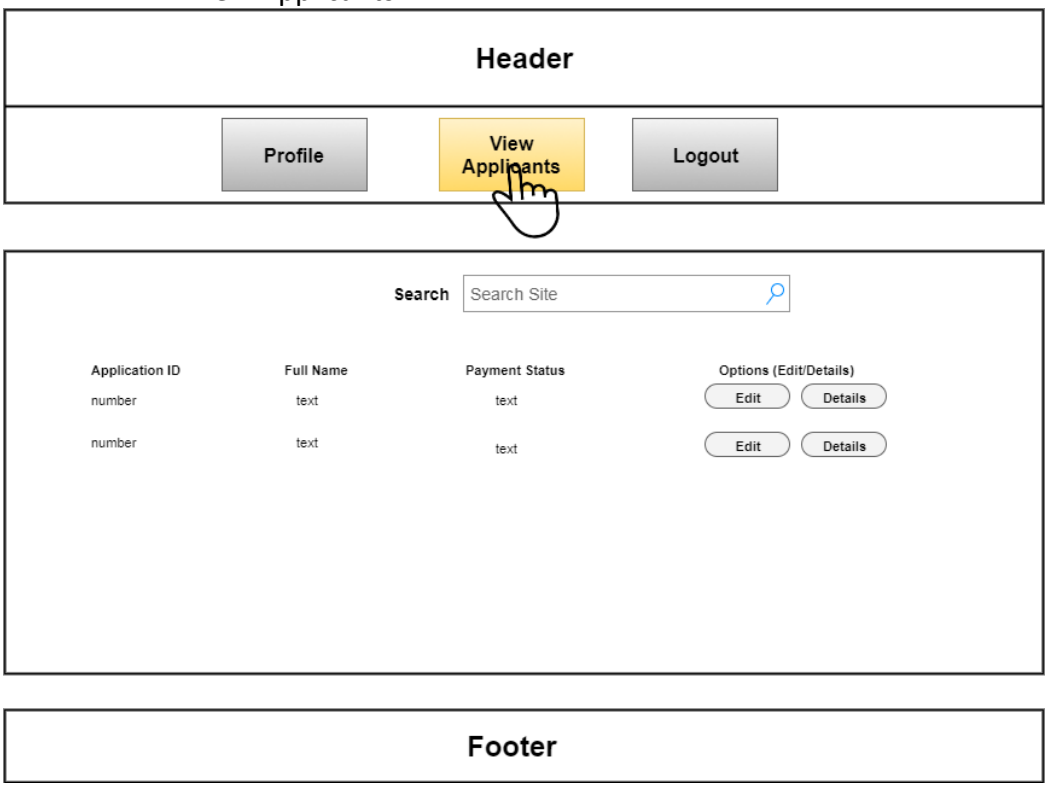


Figure 224 View Applicants

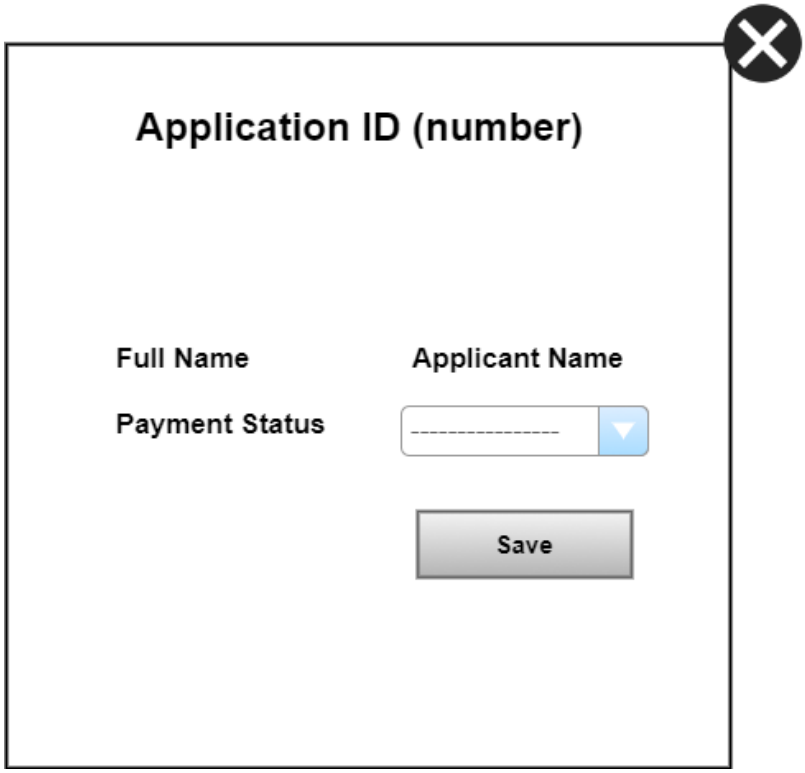


Figure 225 Update Payment Status

- Scrutinizer
  - View Applicants

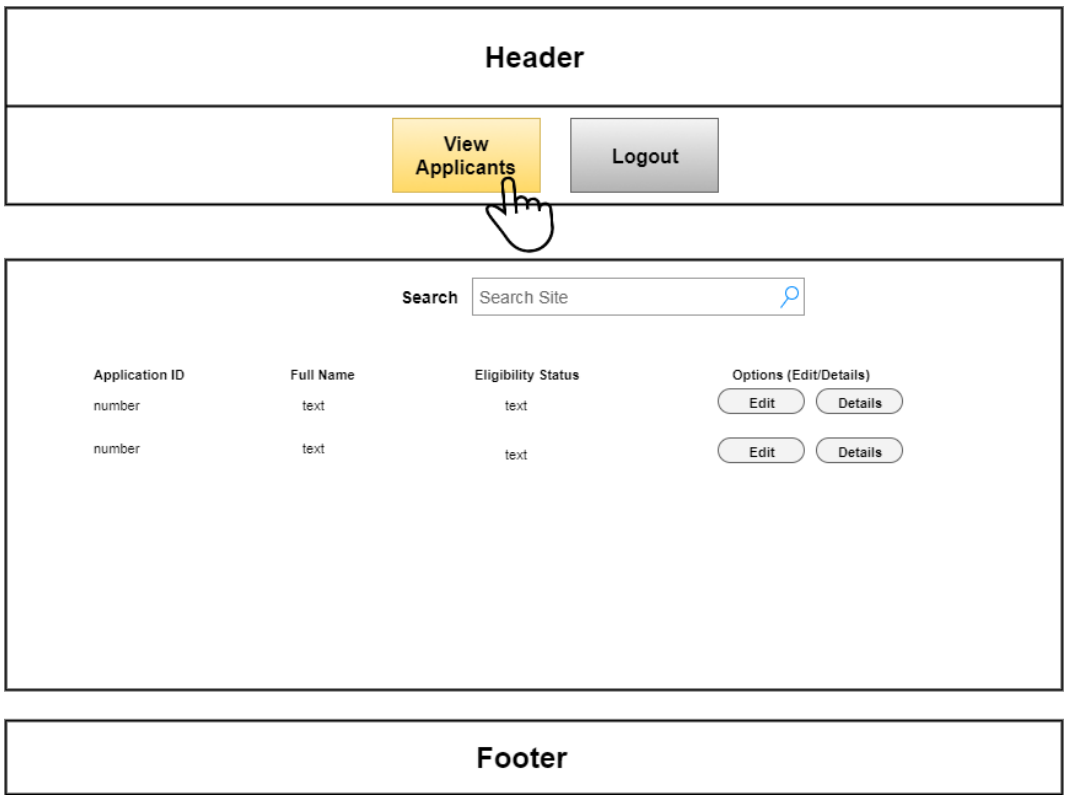


Figure 226 View Applicants (Scrutinizer)

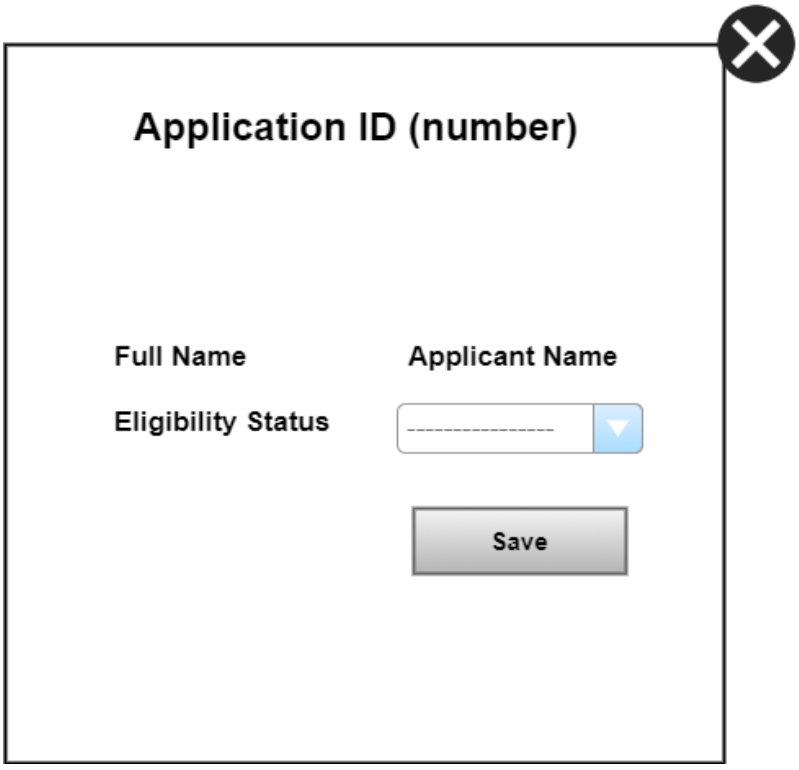


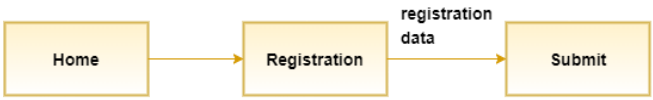
Figure 227 Update Eligibility Status

4.2.2 Define Events that will Cause the State of the User Interface to Change

External Object

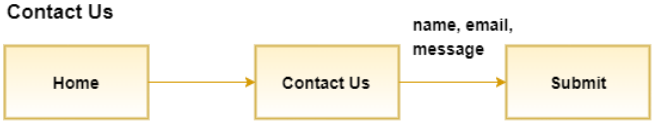
Sign-in

Registration



Recover Passowrd

Contact Us



About us

Scrutinizer

View Applicants Scrutinizer

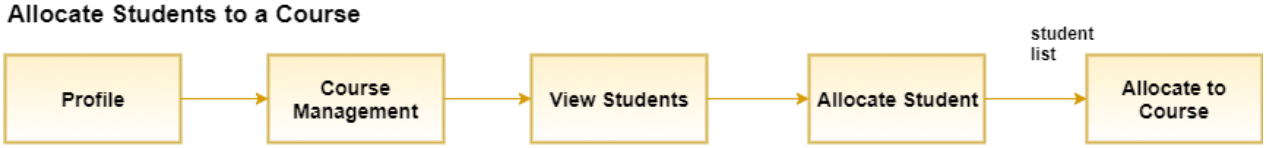
Edit Applicant

Program Chairperson

Update Profile

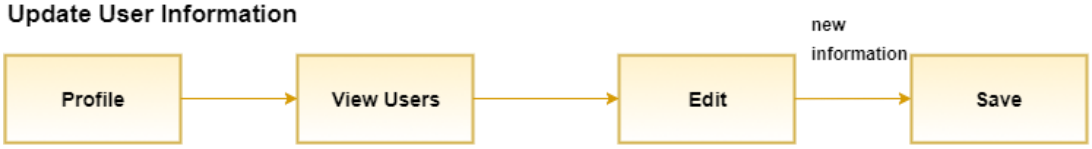
Set Course Teacher

Allocate Students to a Course

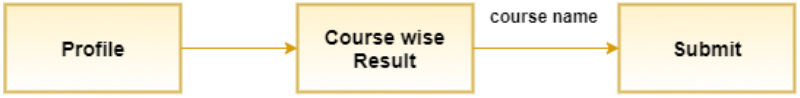


Approve New Account

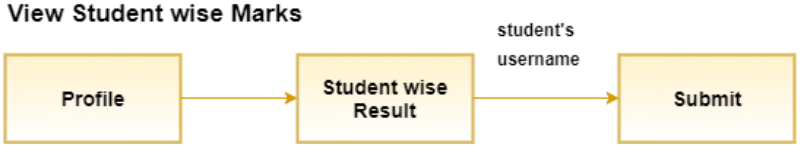
Update User Information



View Course wise Marks



View Student wise Marks



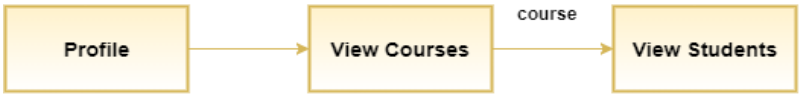
Receptionist

View Applicants

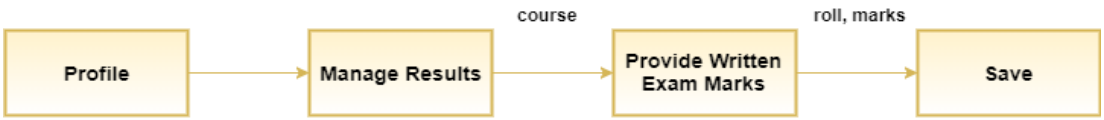
Edit Applicant

Teacher

View Students of a Course



Provide Written Exam Marks

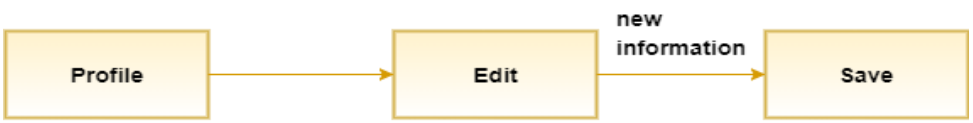


Provide Supplementary Exam Marks

Notification

Student

Update Profile



View Courses

View Notification

Download Report Card

Exam Controller

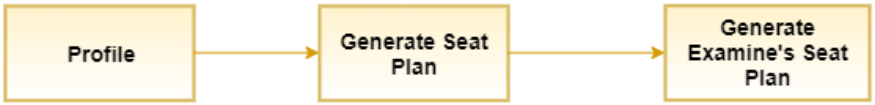
Update Profile

Introduce New Admission

Update Current Admission Session

Upload Marks

Generate Examine's Seat Plan



Generate VIVA Schedule and Notify Teachers

Enroll Students to Program

Update Room Information

Add New Room Information

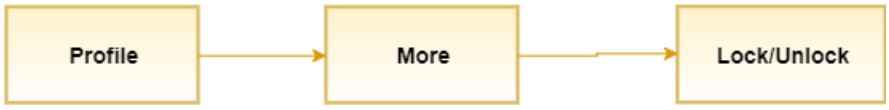
Term Result Management

Upload Signature

View Notifications

Make Admit Card Available

Scrutinizing Session



4.2.3 Depict Each Interface State as It Look to End User

Home Page

Figure 228 Home Page

Registration Page

Figure 229 Registration Page

About us Page

Figure 230 About us Page

Contact us Page

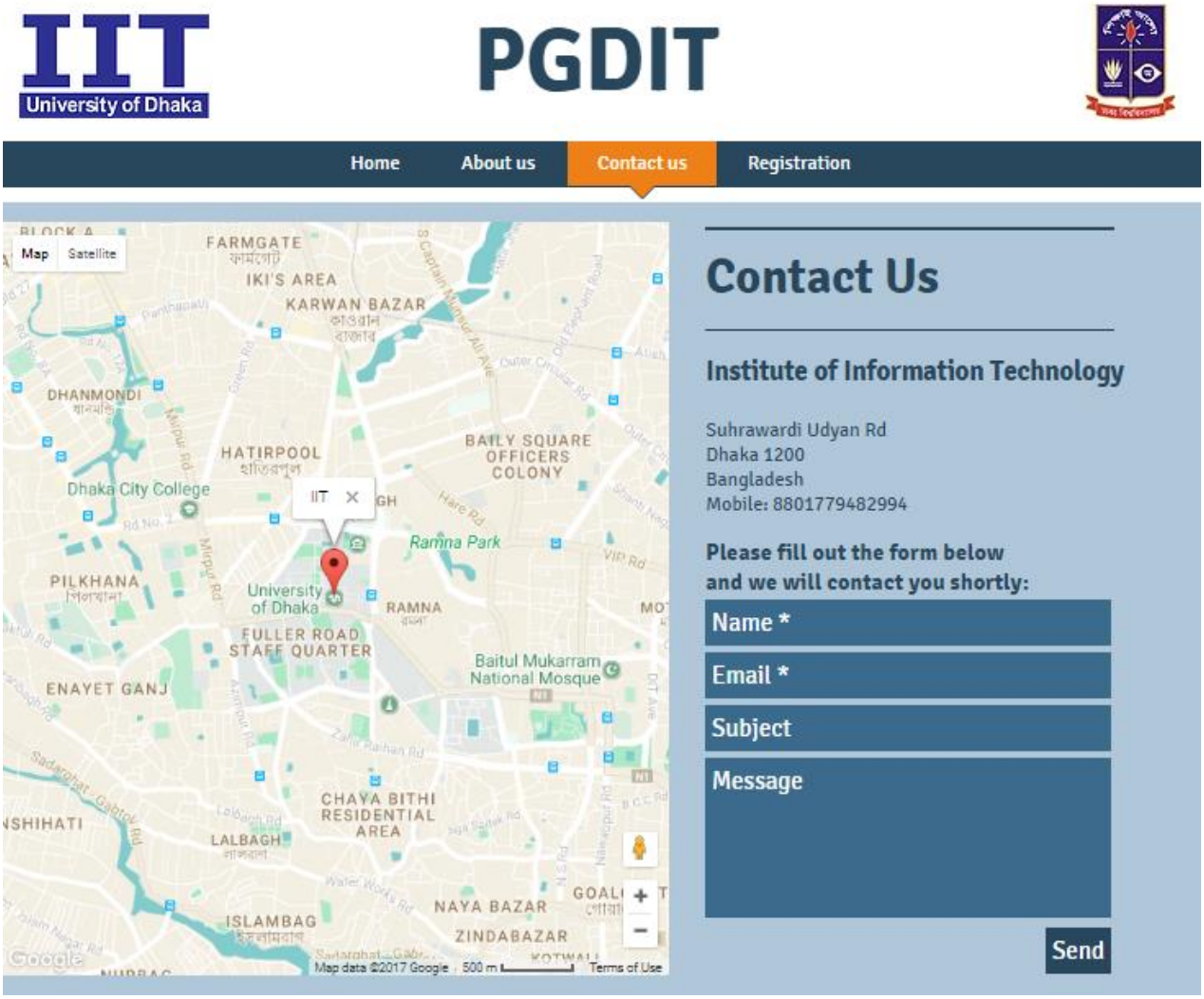


Figure 231 Contact us Page

Program Chairperson: Profile

Figure 232 Program Chairperson:Profile

Program Chairperson: Course Management

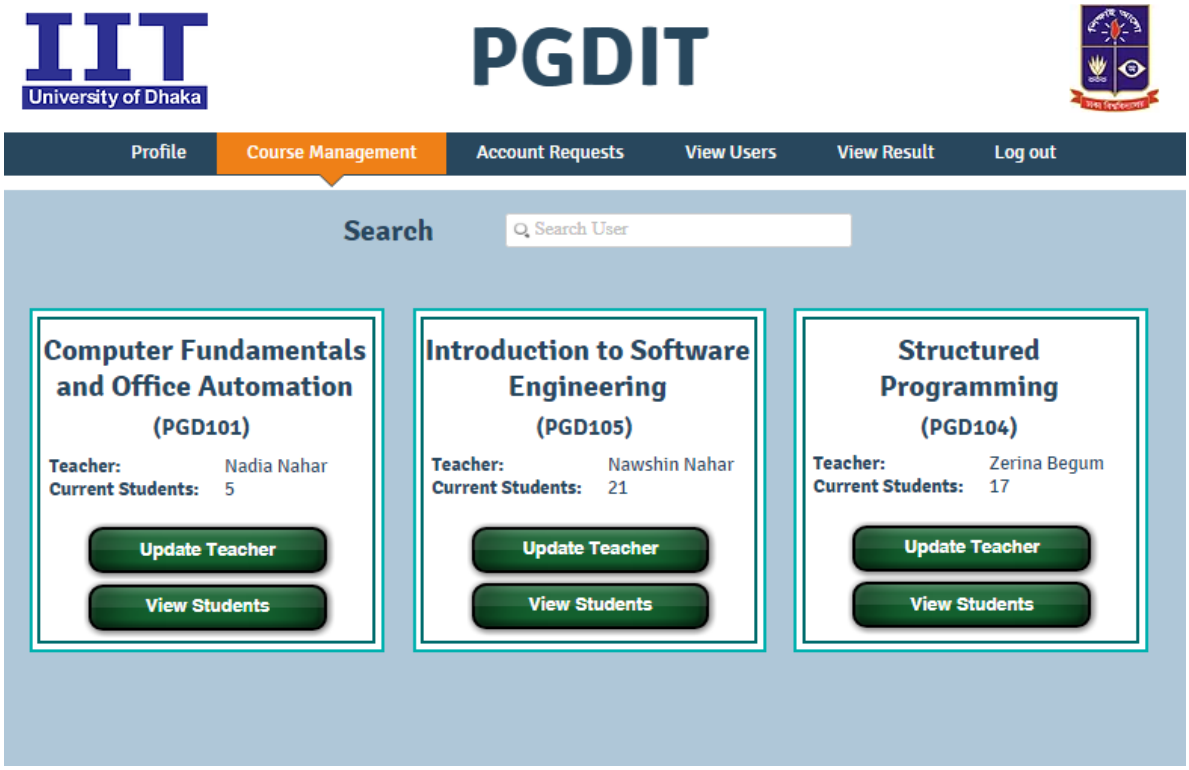


Figure 233Program Chairperson: Course Management

Program Chairperson: Account Requests

Figure 234 Program Chairperson: Account Requests

Program Chairperson: View Users

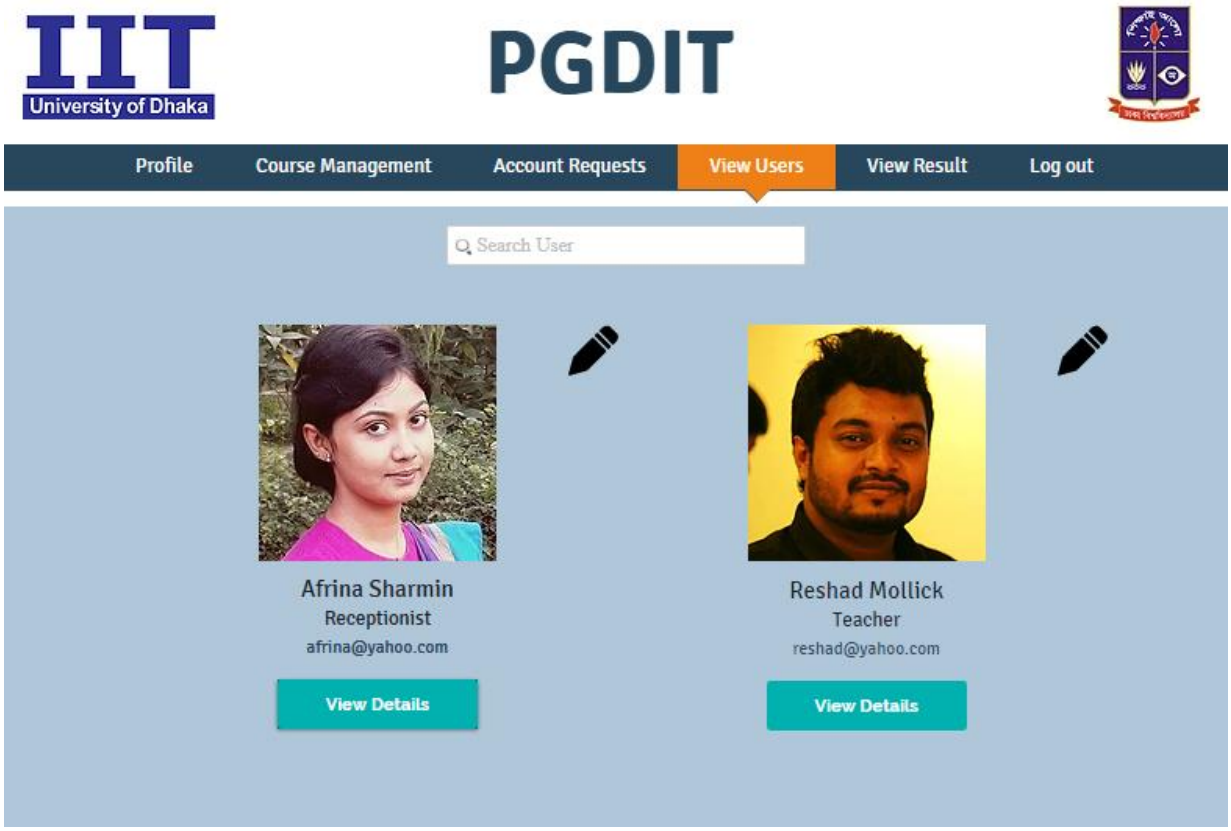


Figure 235 Program Chairperson: View Users

Teacher: Profile

Figure 236 Teacher: Profile

Teacher: View Course

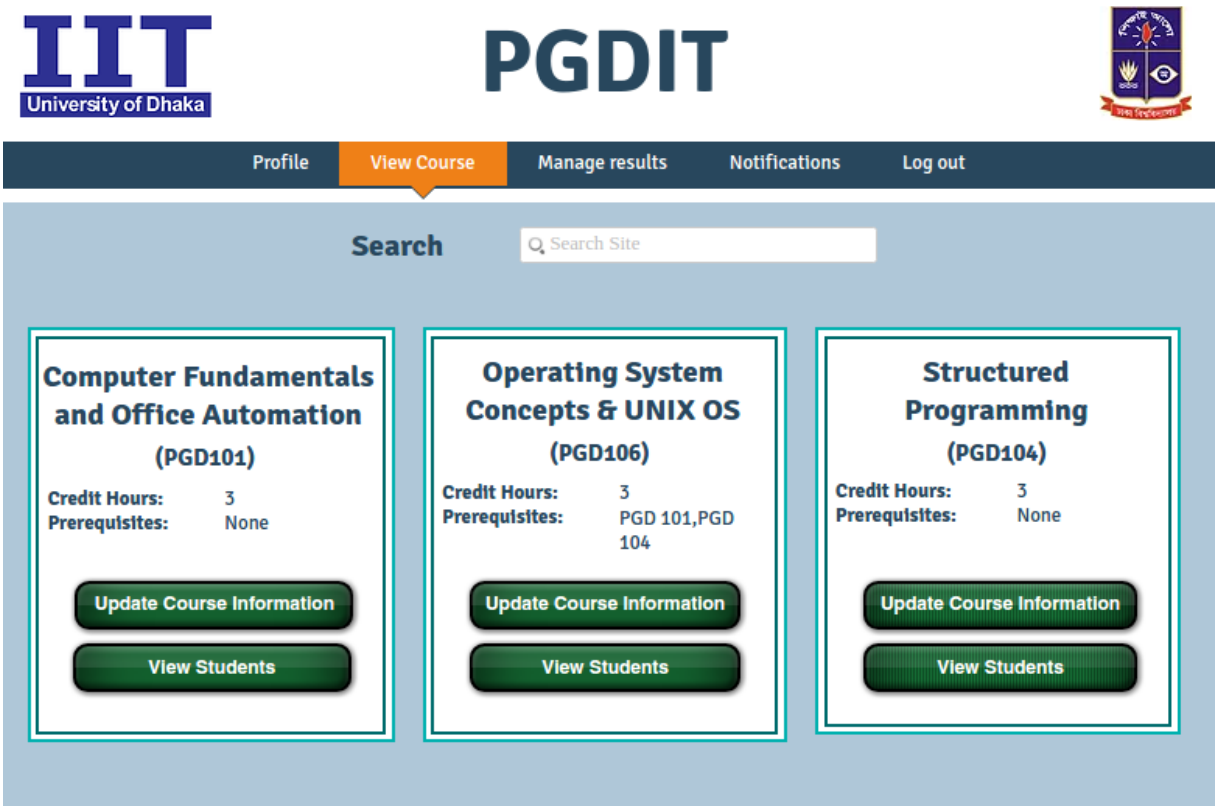


Figure 237 Teacher: View Course

Teacher: Manage Results

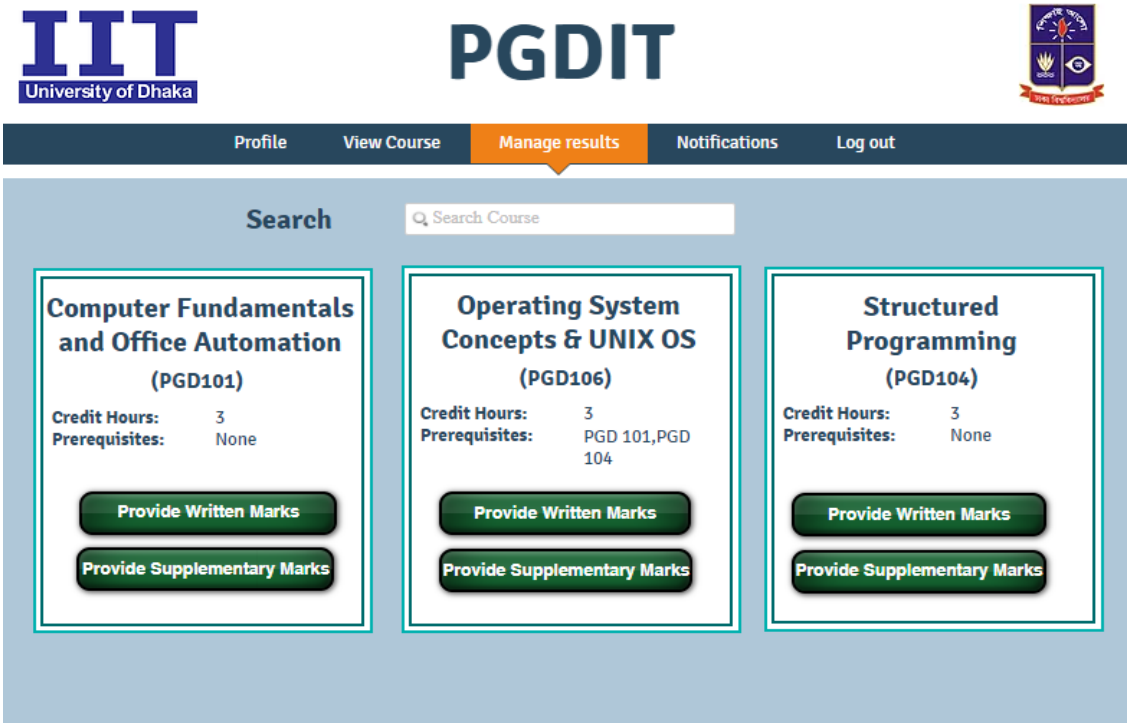


Figure 238 Teacher: Manage Results

Teacher: Notifications

Figure 239 Teacher: Notifications

Exam Controller: Profile

Figure 240 Exam Controller: Profile

Exam Controller: Introduce New Admission

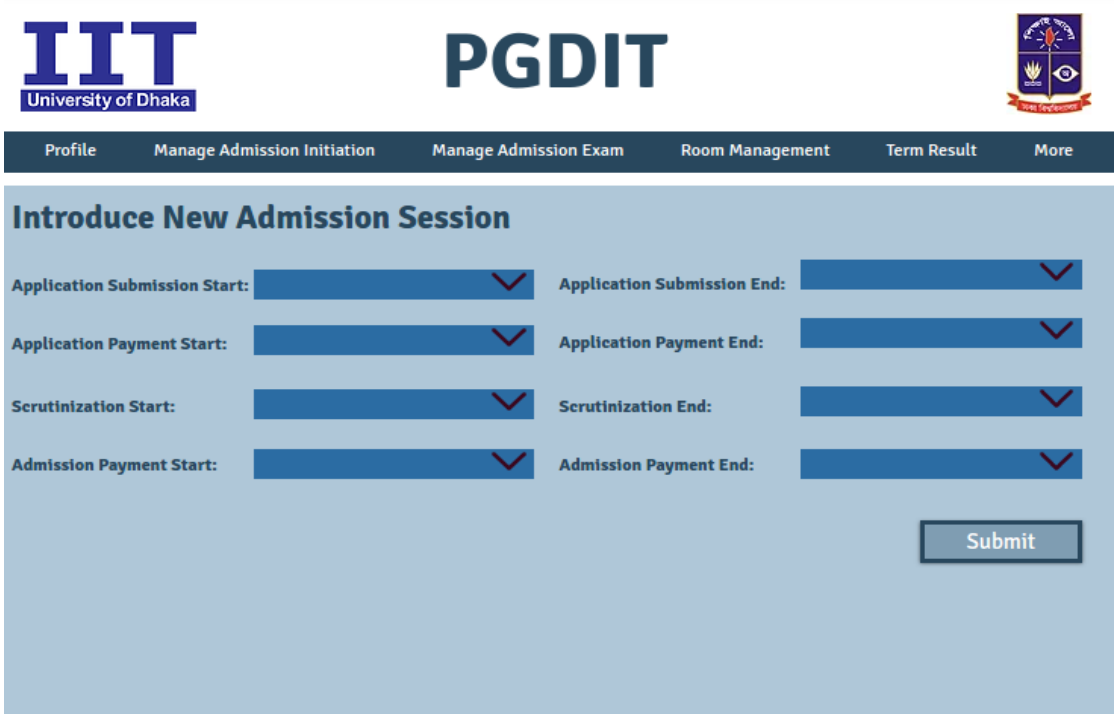


Figure 241 Exam Controller: Introduce New Admission

Exam Controller: Update Admission Information

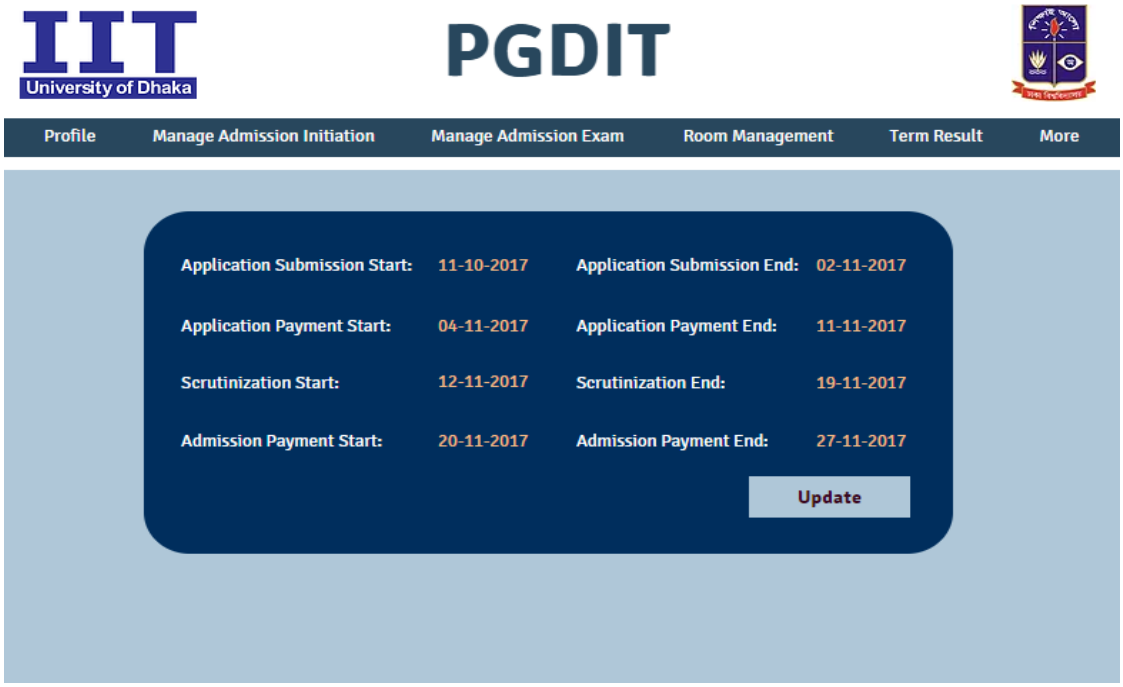


Figure 242Exam Controller: Update Admission Information

Exam Controller: Additional Admission Initiation Activities

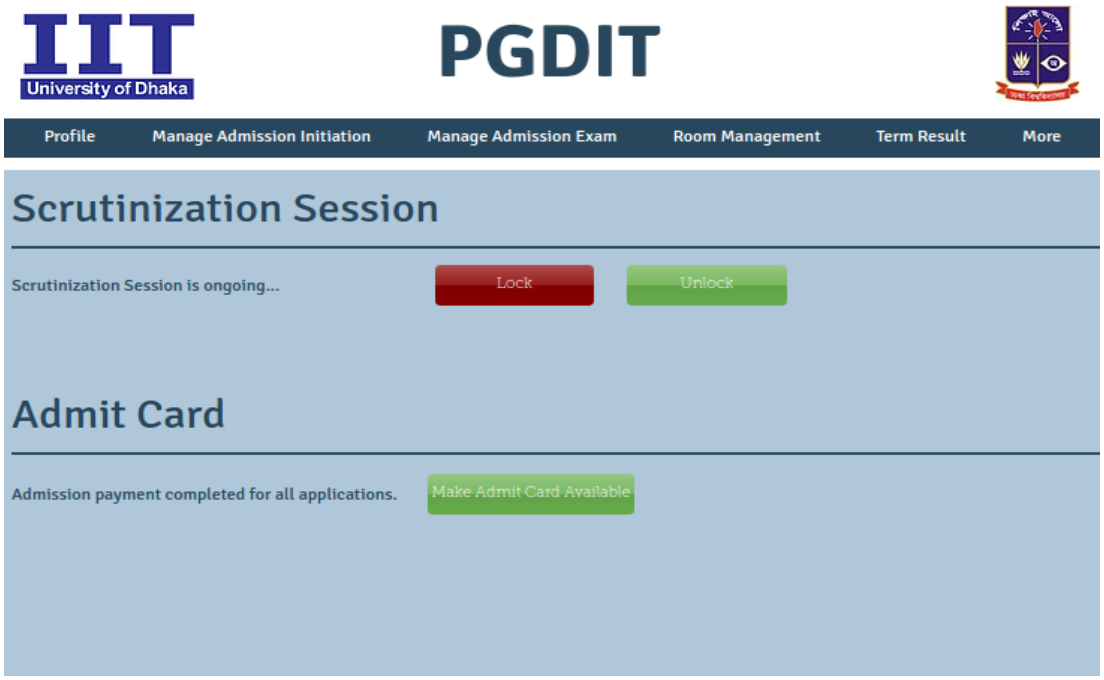


Figure 243 Exam Controller: Additional Admission Initiation Activities

Exam Controller: Upload Marks written/VIVA

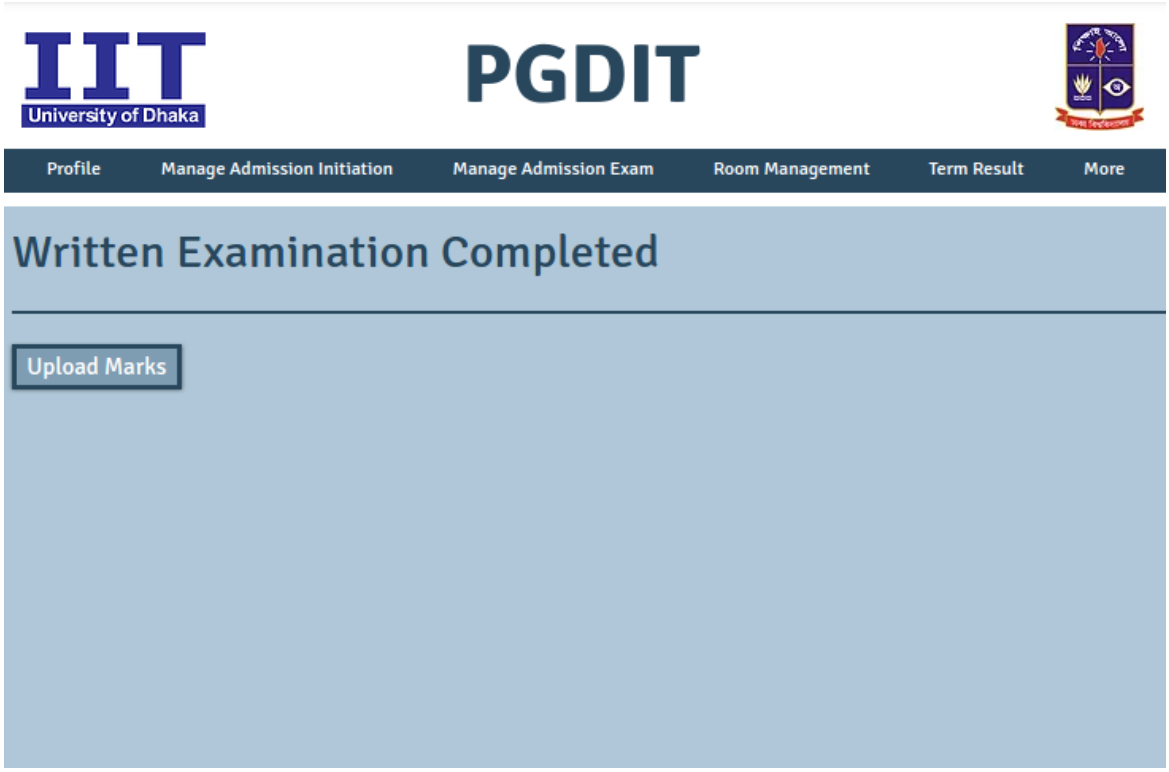


Figure 244 Exam Controller: Upload Marks written/VIVA

Exam Controller: Generate Seat Plan

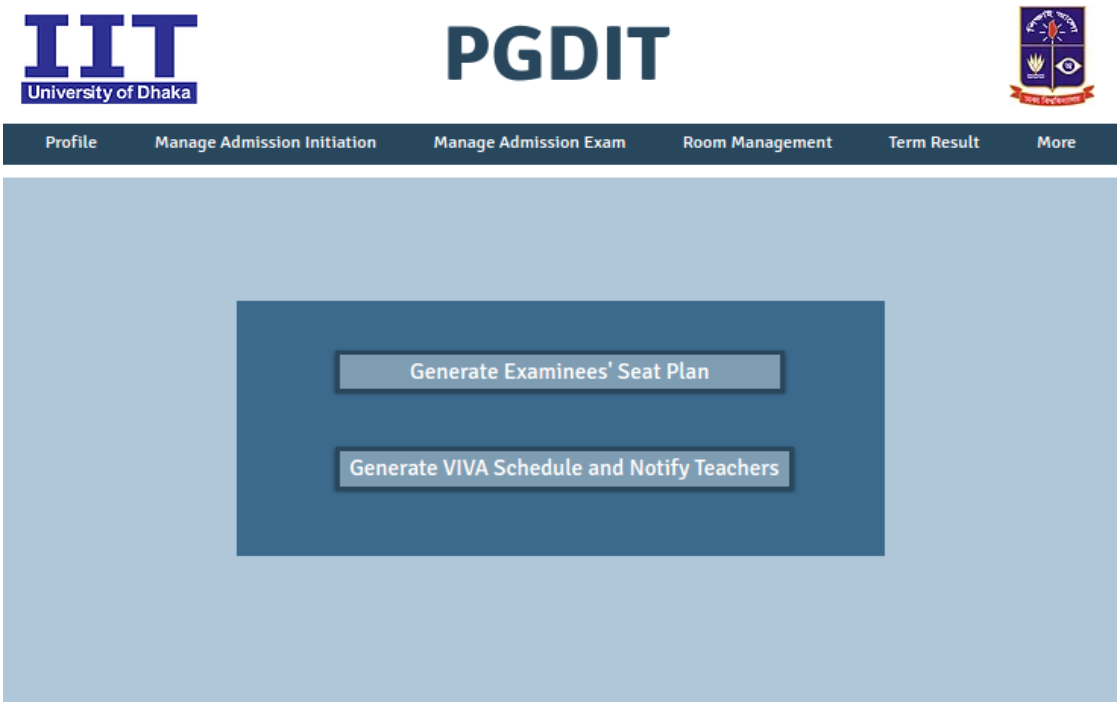


Figure 245 Exam Controller: Generate Seat Plan

Exam Controller: Update Enrollment State

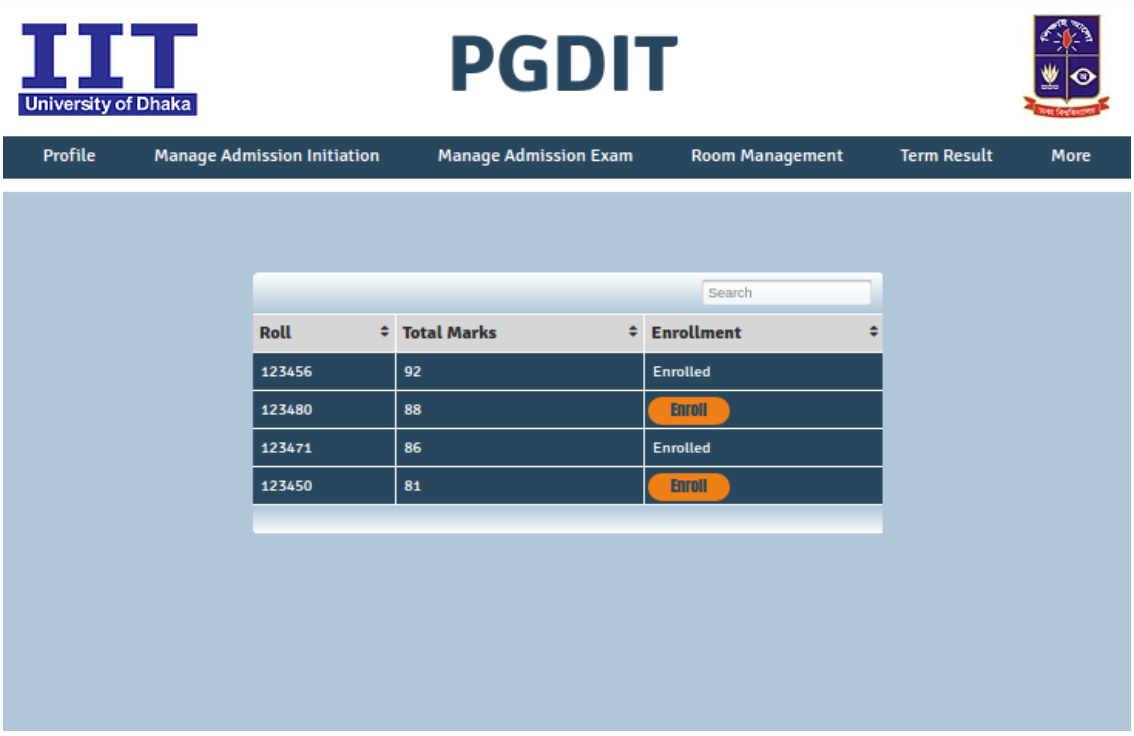


Figure 246 Exam Controller: Update Enrollment State

Exam Controller: Manage Rooms

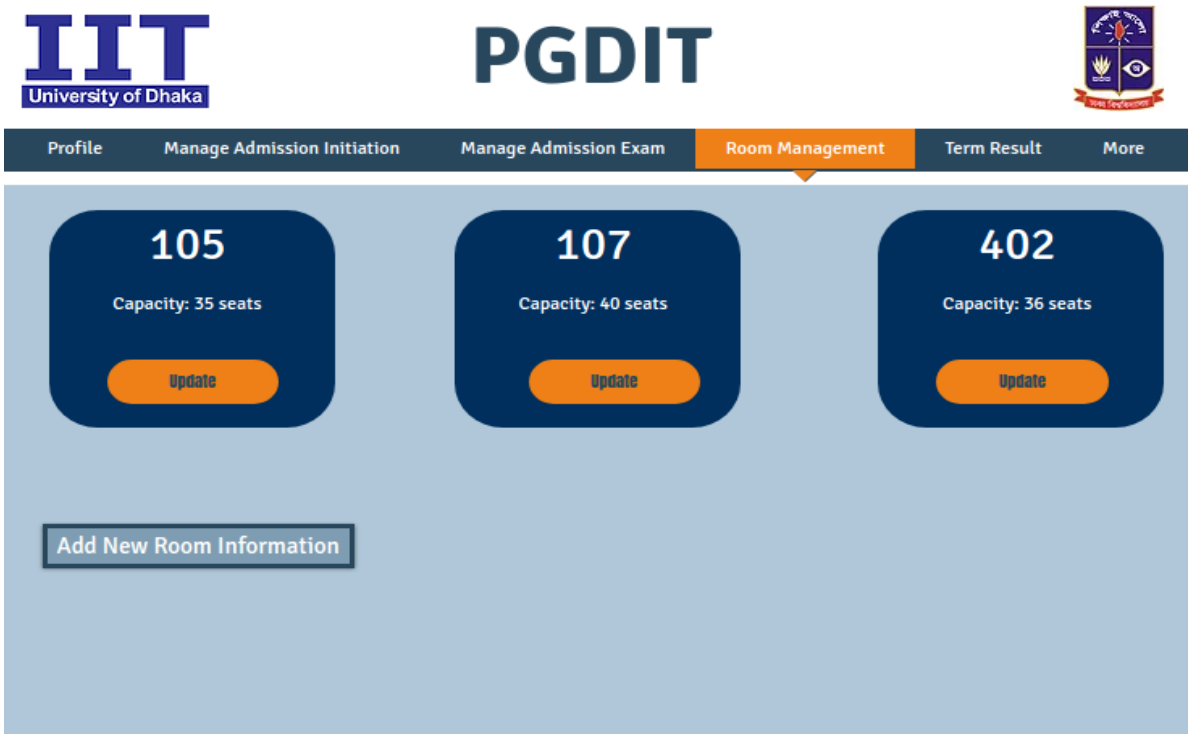


Figure 247 Exam Controller: Manage Rooms

Exam Controller: Term Result Management

Figure 248 Exam Controller: Term Result Management

Exam Controller: Update Director's Signature

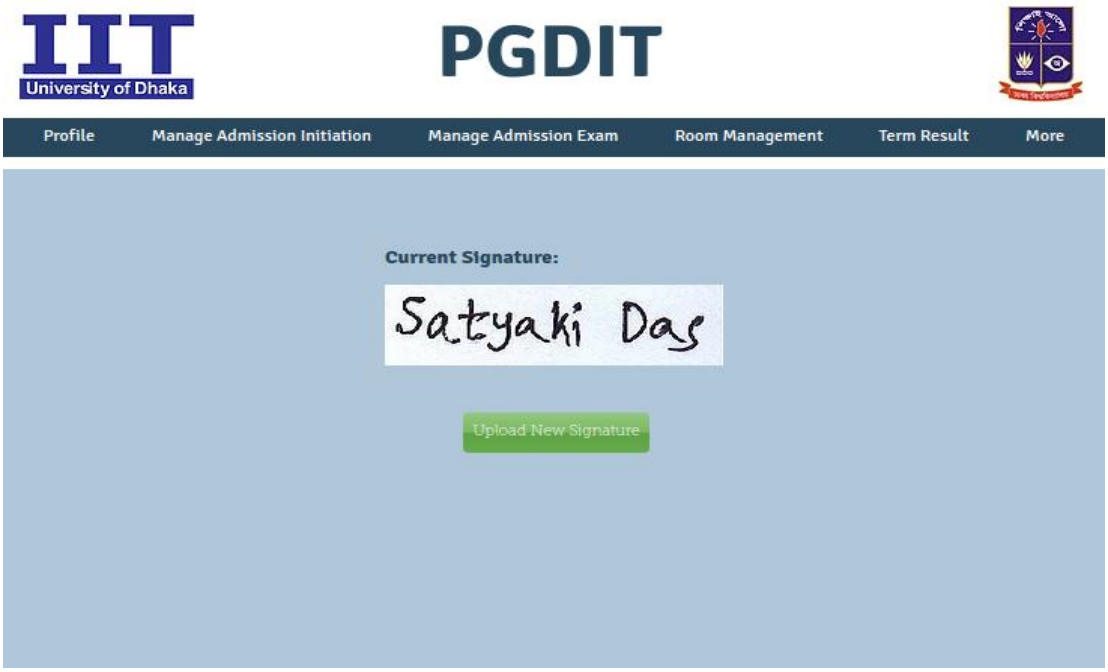


Figure 249 Exam Controller: Update Director's Signature

Exam Controller: Notifications

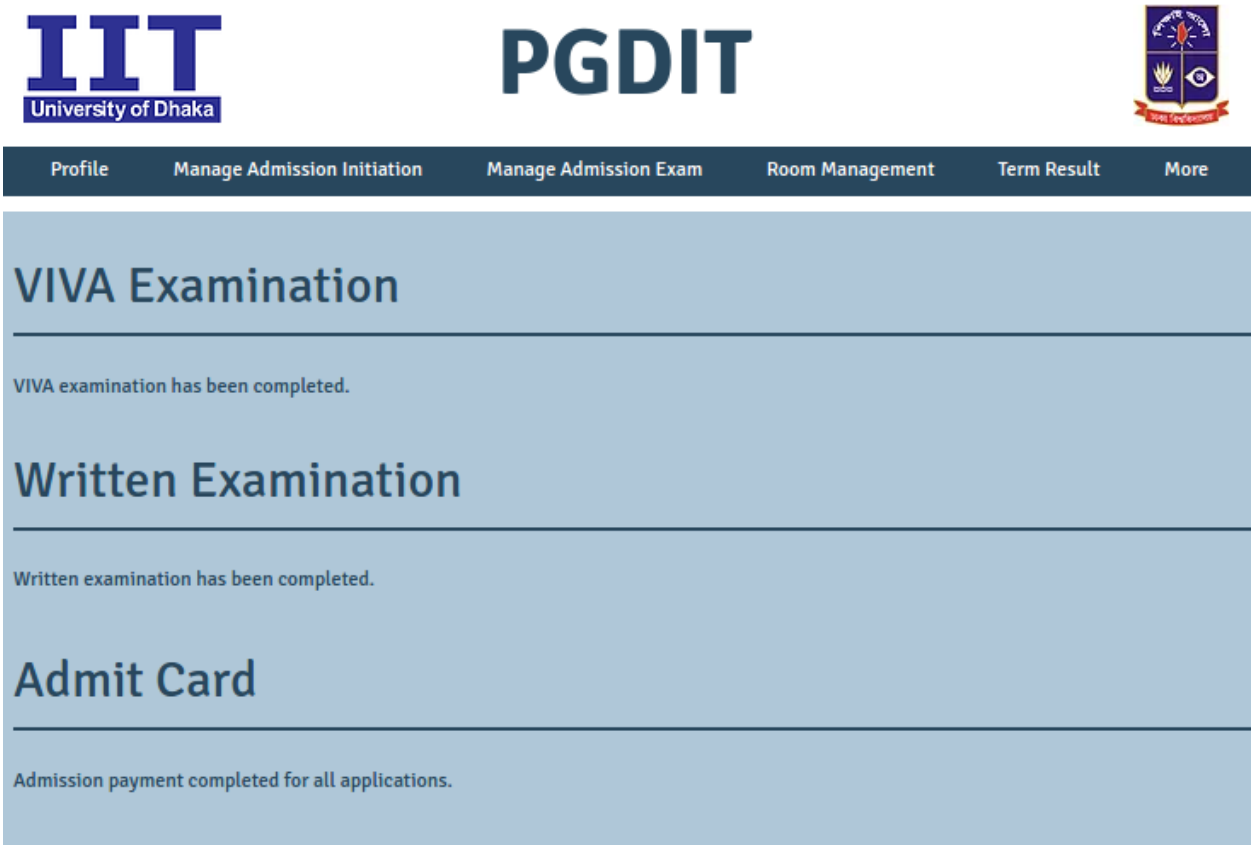


Figure 250 Exam Controller: Notifications

Student: Profile

Figure 251 Student: Profile

Student: View Running Courses

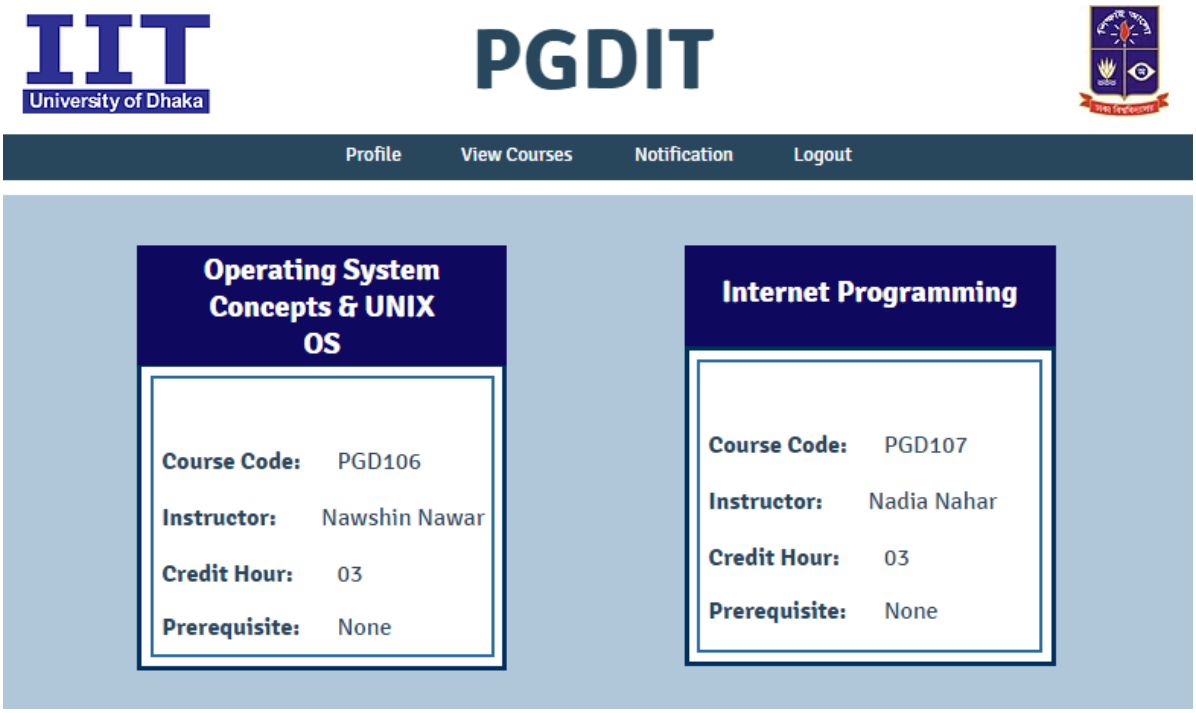


Figure 252 Student: View Running Courses

Student: View Courses Not Yet Taken

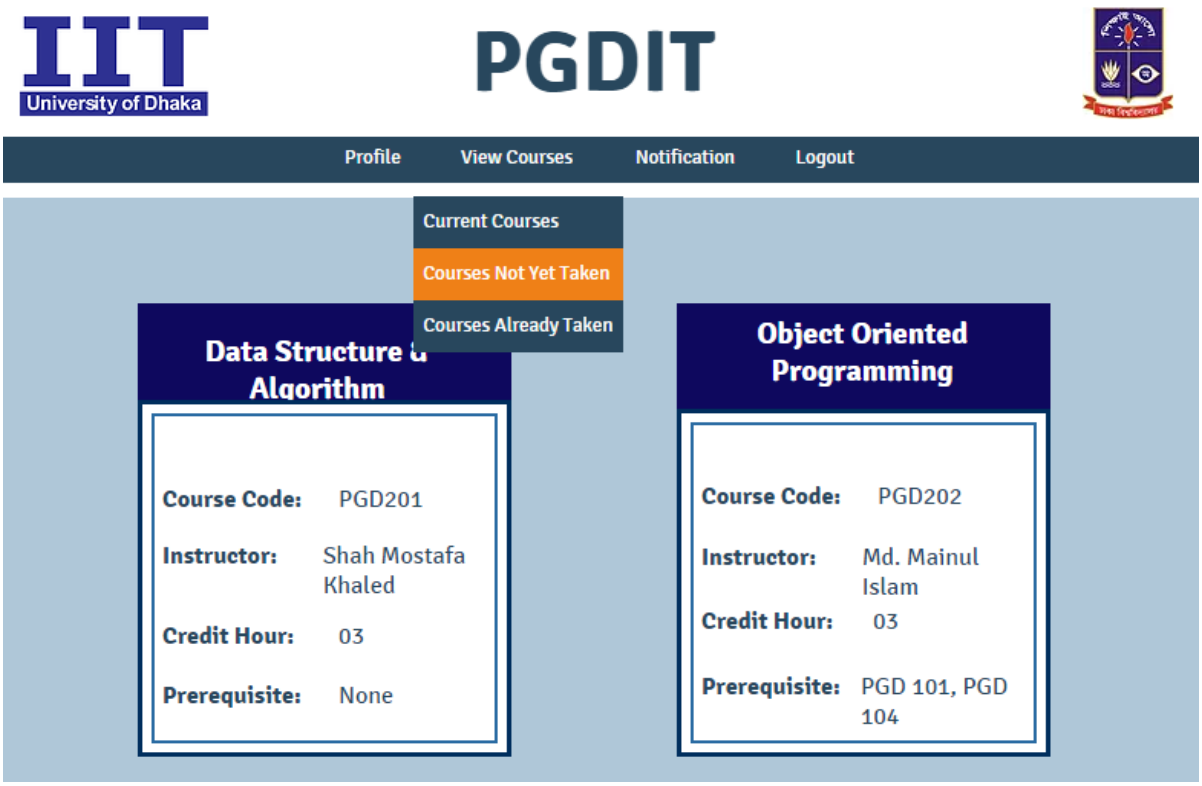


Figure 253 Student: View Courses Not Yet Taken

Student: View Courses Already Taken

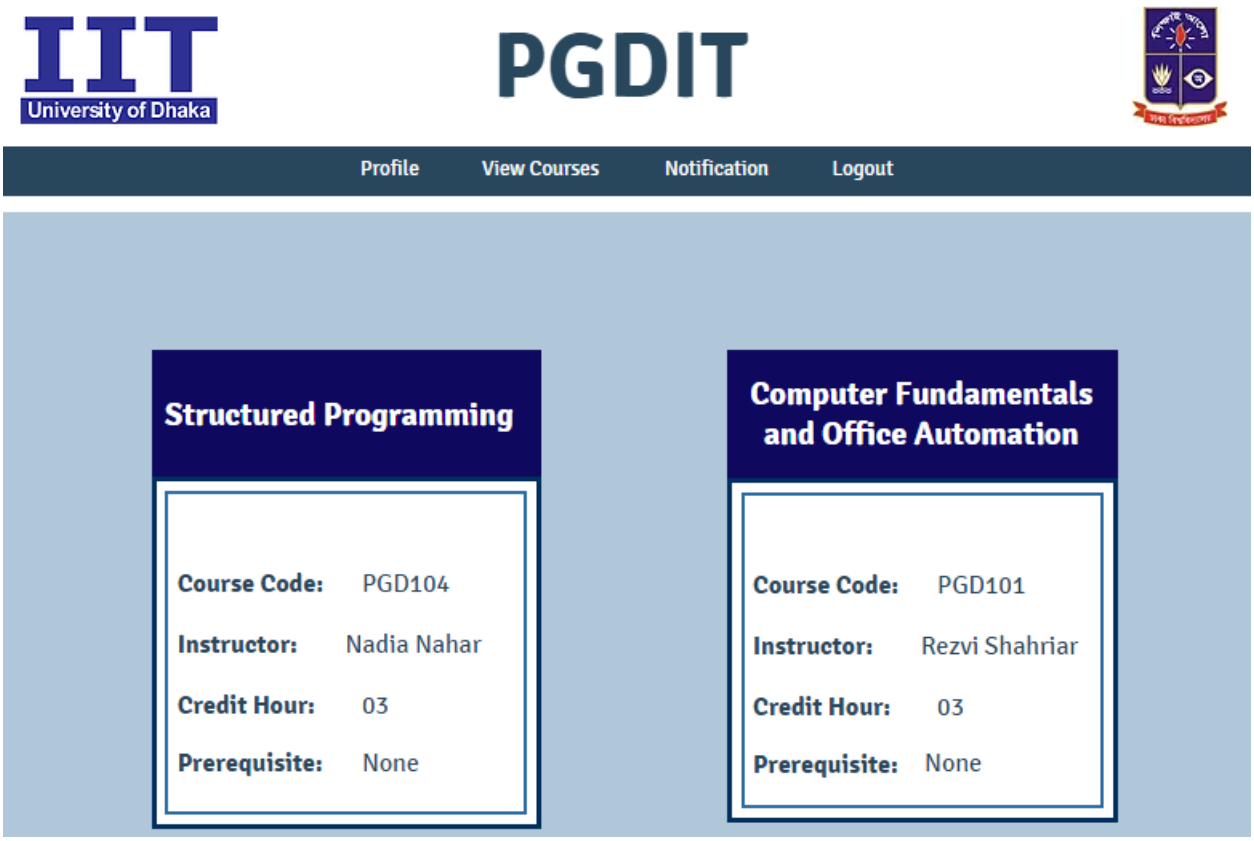


Figure 254 Student: View Courses Already Taken

Student: Notifications

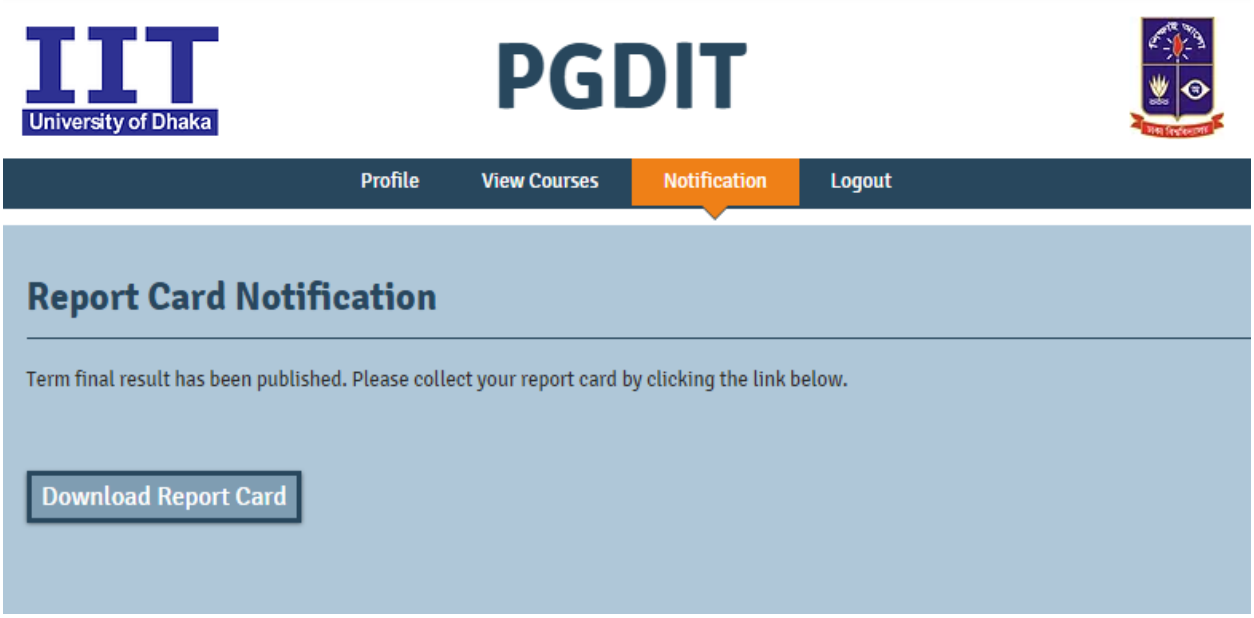


Figure 255 Student: Notifications

Scrutinizer: View Applicants

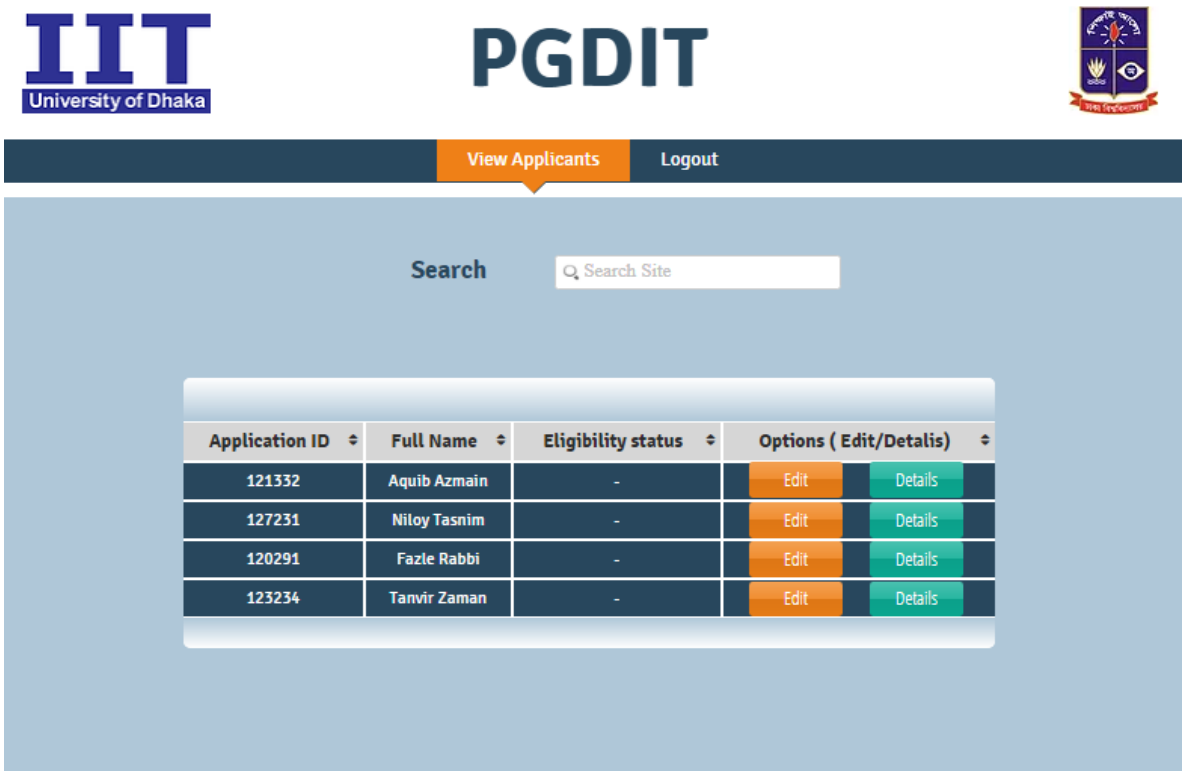


Figure 256 Scrutinizer: View Applicants

Receptionist: Profile

Figure 257 Receptionist: Profile

Receptionist: View Applicants

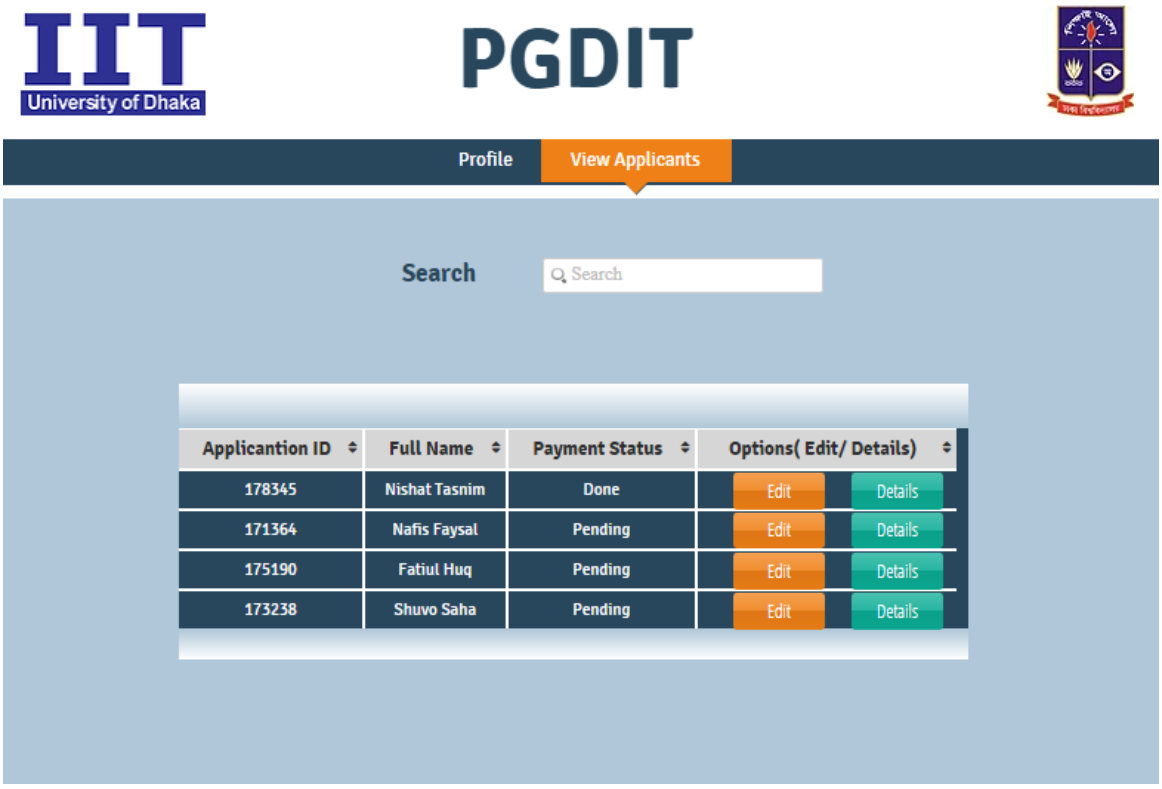


Figure 258 Receptionist: View Applicants

References

- Pressman, Roger S. Software Engineering: A Practitioner's Approach (7th Edition)
- User Interface Design Basics, https://www.usability.gov/what-and-why/user-interface-design.html, last accessed on: 10.11.2017
- What is state diagram (state machine diagram or statechart diagram), http://searchsoa.techtarget.com/definition/state-diagram, last accessed on 14.11.2017