

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Master of Science.

---

Hossein Hakimzadeh, Ph.D.

---

Liguo Yu, Ph.D.

---

Michael R Scheessele, Ph.D.

---

Yu Song, Ph.D.

May 18, 2009

# **IU-ADVISE: A WEB BASED ADVISING TOOL FOR ACADEMIC ADVISORS AND STUDENTS**

Truong Quoc Hung

Submitted to the faculty of the University Graduate School  
in partial fulfillment of the requirements  
for the degree of

Master of Science

in the Department of Computer and Information Science  
Indiana University  
May 18, 2009

Copyright 2009  
Truong Quoc Hung  
ALL RIGHTS RESERVED

# **Abstract**

Academic advising is an important activity of an academic institution. It guides the students to explore potential careers, academic disciplines and opportunities in the college environment. An accurate and full featured advising system can be an effective tool to both students and faculty advisors. The dynamic nature of academic programs, especially in regards to changes in the general education and other degree requirements, poses a continuous challenge to faculty advisors to remain up-to-date. The goal of this thesis is to implement a web-based advising system which facilitates academic advisors in their efforts to providing quality, accurate and consistent advising services to their students. The proposed system was implemented using a set of open source software packages to create a low cost, flexible, and customizable system.

# Acknowledgements

In the completing of this graduate project I have been fortunate to have the help, support and encouragement from many people. I would like to acknowledge them for their cooperation.

I wish to express deep heart-felt gratitude to my project advisor, Dr. Hossein Hakimzadeh, Ph.D., Director of Informatics, for guiding and assisting me through every step of the process with helpful knowledge and support.

I would like to thank Dr. Ligu Yu, Dr. Yu Song and Dr. Michael R Scheessele, my thesis committee members, who showed immense patience and understanding through detailed review and suggestions.

I am grateful to Dr. Robert Batzinger, Ph.D., Informatics Lab Supervisor, for explaining difficult aspects of PHP, web programming and  $\LaTeX$ .

I owe my most sincere gratitude to all staffs and my friends who helped me during my studies and work at IU South Bend.

I would like to thank my grandmother, uncles, aunts, and cousins who have been constantly provided me with various forms of support since the first day I came to the United States.

I owe my loving thanks to Huong Xuan Hoang Thi and her family for their encouragement and understanding.

Lastly and most importantly, I wish to thank my parents and brother, Dieu Hien Thi Nguyen, Thi Truong and Dung Truong. Their support and love that helped me overcome difficulties, enjoy my studies and achieve my goals.

# Contents

List of Tables .....	v
List of Figures .....	vii
1. Introduction .....	1
2. Literature Review .....	3
3. Project Design .....	7
3.1. Privacy Policy .....	7
3.2. Data Model .....	8
3.3. Database Design .....	10
3.4. Process Model .....	34
3.5. User Interface .....	62
4. Implementation Technologies .....	80
4.1. Application Architecture .....	80
4.2. Ajax .....	82
4.3. Apache Web Server .....	85
4.4. Server-Side Scripting with PHP .....	87
4.5. MySQL .....	90
4.6. Design Patterns .....	94
4.7. MVC in Zend Framework Implementation .....	97
5. Conclusion .....	100
Appendix A. Zend Framework .....	102
A.1. MVC Implementation Classes .....	103
A.2. Useful Classes from Zend Framework .....	113
Bibliography .....	118

# List of Tables

Table 3.1. SQL table structure for campus . . . . .	10
Table 3.2. SQL table structure for campus_reqs . . . . .	11
Table 3.3. SQL table structure for admin_role . . . . .	12
Table 3.4. SQL table structure for admin_personnel . . . . .	13
Table 3.5. SQL table structure for college . . . . .	14
Table 3.6. SQL table structure for college_reqs . . . . .	15
Table 3.7. SQL table structure for course_relationship . . . . .	16
Table 3.8. SQL table structure for pre_co_req . . . . .	16
Table 3.9. SQL table structure for degree_type . . . . .	17
Table 3.10. SQL table structure for department . . . . .	18
Table 3.11. SQL table structure for communication_type . . . . .	19
Table 3.12. SQL table structure for access_level . . . . .	19
Table 3.13. SQL table structure for course . . . . .	20
Table 3.14. SQL table structure for academic_program . . . . .	21
Table 3.15. SQL table structure for advisor . . . . .	22
Table 3.16. SQL table structure for advise . . . . .	23
Table 3.17. SQL table structure for advisor_remark . . . . .	24
Table 3.18. SQL table structure for enrollment . . . . .	25
Table 3.19. SQL table structure for satisfied_by . . . . .	26
Table 3.20. SQL table structure for detailed_requirement . . . . .	27
Table 3.21. SQL table structure for requirements . . . . .	28
Table 3.22. SQL table structure for degree_req_categories . . . . .	29
Table 3.23. SQL table structure for declared_program . . . . .	30
Table 3.24. SQL table structure for student . . . . .	31
Table 3.25. SQL table structure for completion_method . . . . .	32
Table 3.26. SQL table structure for semester . . . . .	32
Table 3.27. SQL table structure for actionlog . . . . .	33

Table 4.1. MVC implementations for different languages . . . . . 97



# List of Figures

Figure 2.1. SIS introduction page . . . . .	4
Figure 2.2. Part of a SIS unofficial transcript page . . . . .	5
Figure 2.3. Part of a SIS what-if report . . . . .	5
Figure 3.1. ER Diagram of IU Advisee . . . . .	9
Figure 3.2. Functional Decomposition Diagram . . . . .	35
Figure 3.3. Context Data Flow Diagram of IU-Advise system . . . . .	37
Figure 3.4. Student - View Degree Audit Context DFD . . . . .	38
Figure 3.5. Student - View Degree Audit Detailed DFD . . . . .	39
Figure 3.6. Student - View Unofficial Transcript Context DFD . . . . .	40
Figure 3.7. Student - View Unofficial Transcript Detailed DFD . . . . .	41
Figure 3.8. Student - View Grades Context DFD . . . . .	42
Figure 3.9. Student - View Grades Detailed DFD . . . . .	43
Figure 3.10. Student - View Advisor Remarks Context DFD . . . . .	44
Figure 3.11. Student - View Advisor Remarks Detailed DFD . . . . .	45
Figure 3.12. Advisor - View Advisor Remarks Context DFD . . . . .	46
Figure 3.13. Advisor - View Advisor Remarks Detailed DFD . . . . .	47
Figure 3.14. Advisor - View Degree Audit Context DFD . . . . .	48
Figure 3.15. Advisor - View Degree Audit Detailed DFD . . . . .	49
Figure 3.16. Advisor - View Unofficial Transcript Context DFD . . . . .	51
Figure 3.17. Advisor - View Unofficial Transcript Detailed DFD . . . . .	52
Figure 3.18. Advisor - View Grades Context DFD . . . . .	53
Figure 3.19. Advisor - View Grades Detailed DFD . . . . .	54
Figure 3.20. Advisor - Create Advisor Remark Context DFD . . . . .	55
Figure 3.21. Advisor - Create Advisor Remark Detailed DFD . . . . .	56
Figure 3.22. Advisor - Modify Advisor Remark Context DFD . . . . .	58
Figure 3.23. Advisor - Modify Advisor Remark Detailed DFD . . . . .	59
Figure 3.24. Advisor - Remove Advisor Remark Context DFD . . . . .	60

Figure 3.25. Advisor - Remove Advisor Remark Detailed DFD . . . . .	61
Figure 3.26. Chosen role screen for IU-Advise . . . . .	62
Figure 3.27. Login form for an advisor . . . . .	63
Figure 3.28. Server validation message . . . . .	63
Figure 3.29. Client validation message . . . . .	63
Figure 3.30. Advisor index page . . . . .	64
Figure 3.31. Advisor information tab . . . . .	64
Figure 3.32. Student demographic information tab . . . . .	65
Figure 3.33. Latest admission information tab . . . . .	66
Figure 3.34. Test out information tab . . . . .	66
Figure 3.35. Advisor remark list . . . . .	67
Figure 3.36. Personal advisor remark list . . . . .	68
Figure 3.37. Add new advisor remark form . . . . .	68
Figure 3.38. Choose a user advisor remark . . . . .	69
Figure 3.39. Load data into form . . . . .	70
Figure 3.40. Choose multiple user advisor remark . . . . .	70
Figure 3.41. Only list valid records . . . . .	71
Figure 3.42. Chosen records for activating or deactivating . . . . .	71
Figure 3.43. Degree audit result . . . . .	72
Figure 3.44. Degree audit detail tab . . . . .	72
Figure 3.45. Input for producing degree what-if report . . . . .	73
Figure 3.46. Degree what-if report details . . . . .	74
Figure 3.47. Excluded courses report . . . . .	74
Figure 3.48. Advisor view grades . . . . .	75
Figure 3.49. Unofficial transcript detail . . . . .	76
Figure 3.50. Student index page . . . . .	76
Figure 3.51. Student advisor remark list . . . . .	77
Figure 3.52. Student degree audit . . . . .	77
Figure 3.53. Student degree audit details . . . . .	78
Figure 3.54. Student degree what-if report . . . . .	78

Figure 3.55. Student view grade report . . . . .	79
Figure 3.56. Student view unofficial transcript . . . . .	79
Figure 4.1. Architecture of the proposed system . . . . .	81
Figure 4.2. AJAX (right) and traditional model (left) . . . . .	83
Figure 4.3. AJAX (bottom) and traditional model (top) . . . . .	84
Figure 4.4. Server Share among the Million Busiest Sites, March 2009 . . . . .	86
Figure 4.5. Structure of MVC model . . . . .	95
Figure 4.6. Structure of Student class . . . . .	95
Figure 4.7. Zend implementation of MVC design pattern . . . . .	98
Figure A.1. Dispatching cycle of Zend Framework . . . . .	103
Figure A.2. Logical model of Factory design pattern . . . . .	108

# 1. Introduction

The main objective of academic advising is to guide, motivate and support students as they explore their potential and make precise academic choices in order to satisfy students' needs and comply with academic policies [1]. To achieve these objectives, IU South Bend employs the direct communication between advisors and students as the main advising system. Advisors are typically faculty or professional advisors employed by an academic unit. A normal advising session consists of meetings between an advisor and a student. On the basis of these meetings, the student makes decisions about class schedules, choosing an academic major or minor, planning for graduation and many other academic related activities. These important decisions are made based on information about previously completed courses, degree requirements, academic policies, and offered courses in the upcoming semester provided by advisors balanced against the student's work schedule and other interests or commitments.

The current academic advising system, however, has encountered some problems [2]. Among these, there are some noticeable points. First, academic advisors serve as major and comprehensive resources for students to utilize, and therefore need to spend time understanding and updating their knowledge about degree requirements and academic policies as well as familiarizing themselves with students' progress toward academic degrees prior to any advising period. This is a time-consuming task for any advisor especially when students far outnumber their advisors. Second, a faculty advisor may not be able to keep up with new academic policies, programs and/or degree requirements as they may have several duties during an advising period adding to the difficulties in updating information. This situation can lead to inconsistent information among advisors. Third, most of the time, advisors answer recurrent questions about trivial class scheduling. In fact, these questions could be answered easily by students themselves, if useful information about class schedules and previously completed courses were available and easy to access. Accordingly,

there should be a tool for helping students to take advantage of authorized part of academic information before coming to their advisors. Finally, computer-literate students would frequently like to have more electronic interaction for advising. This is an important factor that needs to be taken into consideration because the students are customers of the services both directly and indirectly [3].

Although there are some problems in the current advising system, faculty mentors and advisors cannot be replaced completely by a computer-based system. The reason is that the academic advising process requires professional knowledge of academic disciplines to satisfy questions about a specific course structure, teaching methods, etc. Moreover, the students do not come to the advisors' offices for only course selection, but also for recommendation while they decide their majors and careers. For these types of questions, an academic advisor with intensive and proficient knowledge about a specific field of study is the best source of valuable information in this regard. However, this information cannot be stored and interpreted from static data held in a database.

This thesis attempts to implement a web-based advising system known as "IU-Advise". The system supports following activities for an authenticated and authorized student or advisor:

- View or print an unofficial transcript;
- View or print degree audit that shows the progress toward a degree and identifies unmet requirements;
- View or print students' advising records;
- Add or modify advising information;
- View the degree requirements for a given program in a given academic year;
- View what-if report which shows how previously completed courses would fit into a new degree program;

It is hoped that the proposed system would become a useful tool for both advisors and students and will facilitate the advising process.

## 2. Literature Review

Many universities and colleges are applying computer technology to create useful tools and complete systems that improve the traditional academic advising system. There have been many levels and means for applying computer technology. The basic level is to develop a simple tool that can produce reports or cover a simple task of the advising process. The higher level is the complete software system that supports the complete advising process.

For the basic level, one useful and common tool is a degree audit reporting system (DARS). DARS, which was developed by Miami University in 1985 [4], produces a degree audit which shows all the requirements of a specific academic degree, the courses that satisfy those requirements and the progress of a student toward the degree. This is a small and effective tool for both advisors and students in the advising process. DARS has been purchased and adapted by many academic institutions such as Ohio University [5], University of San Diego[6], University of Missouri - St. Louis [7] and etc.

In the higher level, the AdvisorTrac of RedRock Software [8] is a commercial software that supports scheduling, reviewing, cancelling advising appointments, storing demographic records and keeping track of advising records. AdvisorTrac is currently being used by a number of universities, among of which are the University of Louisville[9], Western Kentucky University[10], Indiana University-Purdue University Fort Wayne [11] and other colleges.

At the software system level, redLatern, an auxiliary of Miami University [12], developed a commercial software solution based on the DARS for both students and advisors. The solution includes 3 components: u.select, u.direct, and u.achieve. This education software solution provides tools for most advising activities, such as planning courses, selecting courses, keeping track of grades, generating degree audits, and other common advising tasks with unified and consistent information throughout an academic institution.

In addition to these specialized software solutions, most comprehensive administrative software systems such as PeopleSoft, SCT Banner and Oracle also have an academic advisement module. The current OneStart application portal of IU South Bend provides an access to Student Self Service. Student Self Service, based on PeopleSoft's Human Resource Management System (HRMS) and Student Information System (SIS), provides two advising services: View My Advisors and View My Advisement Reports [13]. Figures 2.1, 2.2 and 2.3 show some of current SIS reports. The View My Advisement Reports function allows a student to view a degree audit with or without a unofficial transcript. It also produces a what-if report that shows how current enrolled courses could be applied to a new major when a student wants to change his or her major. A course list what-if report of this service determines if selected courses fit into any requirement of a given degree program. These two reports need to be polished and reorganized in order to provide more useful information than the current version. The View My Advisor offers a list of advisors of a given student and allows the student to notify one or many advisors.

**Figure 2.1.** *SIS introduction page*

Degree Progress Report & Advising Transcript  
Indiana University South Bend  
Name : ██████████  
Student ID : ██████████  
Print Date : 02-27-2009  
Request Nbr : 007099631

----- Academic Program History -----  
South Bend Program : Grad Sch-Liberal Arts & Sci  
2006-02-24 : Appl Math & Computer Sci MS Major

----- Beginning of Graduate Record -----  
Fall 2006 South Bend

Course	Description	GPA	Hours	Earned	Grade	Points
CSCI-A 594	DATA STRUCTURES		3.00			
ENG-G 13	ACADEMIC WRITING GRAD STUDENTS		3.00			
ENG-G 20	COMM SKLS GRAD STDNTS & ITA'S		3.00			
PROGRAM GPA:			TERM TOTALS :	9.00		
PROGRAM CUM GPA:			CUM TOTALS :	9.00		

Spr 2007 South Bend

Course	Description	GPA	Hours	Earned	Grade	Points
CSCI-A 340	AN INTRO TO WEB PROGRAMMING		3.00			
CSCI-A 593	COMPUTER STRUCTURES		3.00			
CSCI-C 442	DATABASE SYSTEMS		3.00			
MATH-M 575	SIMULATION MODELING		3.00			
PROGRAM GPA:			TERM TOTALS :	12.00		
PROGRAM CUM GPA:			CUM TOTALS :	21.00		

Summer2007 South Bend

**Figure 2.2.** Part of a SIS unofficial transcript page

**Minor: Mathematics Plan**

Requirements Not Satisfied

**MATH MINOR CORE AND ELECTIVE REQUIREMENTS - 1998 (RG 12110)**

Requirement Not Satisfied -

GPA (required/actual): 2.000/Unknown

Units (required/actual/needed): 18.00/0.00/18.00

**Math minor Requirement. (RQ 13093)**

Requirement Not Satisfied -

GPA (required/actual): 2.000/Unknown

Units (required/actual/needed): 18.00/0.00/18.00

**Student must complete MATH-M215 and MATH-M216**

Requirement Not Satisfied -

GPA (required/actual): 2.000/Unknown

Units (required/actual/needed): 10.00/0.00/10.00

Courses (required/actual/needed): 2.00/0.00/2.00

**Figure 2.3.** Part of a SIS what-if report

A commercial software package is not the only way to meet the needs of improving the quality of advising services. There are many computer software systems developed in-house. In 1999, the Department of Computer Science and Engineering (CSE) of Florida Atlantic University (FAU) started working on a project of a web-based advising system. This system is composed of three components [2]:



- The FAQ component archives and presents the most common questions that can be answered without intensive knowledge about policies, requirements and courses.
- The Course component is used by advisors and administrators to maintain course information.
- The Advising component captures all the information of the advising appointments between advisors and students. It then combines with the data in the database to produce a final result of advising.

In the same manner, Indiana University developed and used Indiana Student Information Transaction Environment [2, 14] - INSITE on some Indiana University campuses before OneStart [2, 14] was employed. INSITE offered the following features:

- Producing an advising report for a student's current major;
- Producing an advising report for a different major;
- Producing an advising report for a special purpose program;
- Viewing how in-progress courses would apply to a student's advising report;
- Adding future courses, grades, and hours to determine affects on the advising report;

## 3. Project Design

In order to develop an application, a conceptual design was produced and reviewed carefully. A conceptual design included two components: the **data model** and the **process model**. The data model determined what information was needed, how it was to be organized and stored. For IU-Advise, we use Entity-Relationship model for organizing and storing data. The process model was used to convey the structure of an application, how its components operate and interact with each other. It also shows how data flows among processes to get the final result. In addition to these conceptual models, the designing process was used to develop a layout for the **user interface model** which represents the interaction of users with the system. The designs of these models and the information privacy policy for sharing advisor remarks among advisors will be the topics for the remainder of this chapter.

### 3.1. Privacy Policy

Privacy is always the an important concern of users. Sharing advisor remarks among advisors help advisors understand the personal needs of a student. However, information sharing without proper controls could lead to the violation of privacy policies, regulations or laws. Therefore, it is important to have a mechanism for restricting access to advisor remarks as well as promoting information sharing. In order to maintain the privacy of students and advisors, the IU-Advise system provides three privacy levels which can be applied to advisor remarks. These levels are Advisor, Advisor Group and Public.

- Advisor level: advisor remarks belonging to this type can only be viewed and manipulated by its owner or the creator. The user can delete and edit his or her own private remark;
- Advisor Group level: advisor remarks in this group can be viewed and shared among advisors but not students. Only the owner or creator of an advisor remark

can deactivate or activate it. When an advisor remark is deactivated, it is not deleted. It is just invisible to other advisors;

- Public level: these remarks can be viewed by both students and advisors using the system. Similar to Advisor Group type advisor remarks, these remarks can only be activated and deactivated by the owner or creator of the advisor remark;

These levels ensure that advisor remarks are only disclosed in a way that do not violate user's privacy or create unexpected difficulties for either students or advisors.

## 3.2. Data Model

Figure 3.1 shows the conceptual data model for the IU-Advise project [15]. We use an Entity-Relationship Diagram (ERD) for modeling the entities (or objects) and relationships among these entities in the application's domain. In the ERD notation, a rectangle symbolizes a data source or data table in the physical implementation. A diamond symbolizes a relationship between two entities. A diamond wrapped by a rectangle indicates that a relationship has its own identification information and is promoted to a data table in the implementing phase. The Entity-Relationship Diagram provides the layout for implementation of the database.

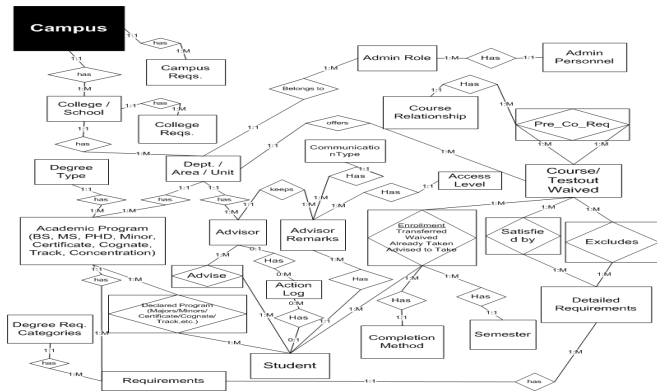


### 3.3. Database Design

This section lists all the tables in the database implementation. As mentioned in the previous section, these tables present essential entities and relationships among them in the IU-Advise system as shown in Figure 3.1. In the section belows, each table is defined with its location and relationships with other tables highlighted in the Entity-Relationship Diagram.

#### campus

Given that Indiana University has multiple campuses, the *campus* table allows our system to maintain information about each campus and it further allows us to extract specific information maintained in the system such as student, faculty, degree requirements, etc by specifying the proper campus.

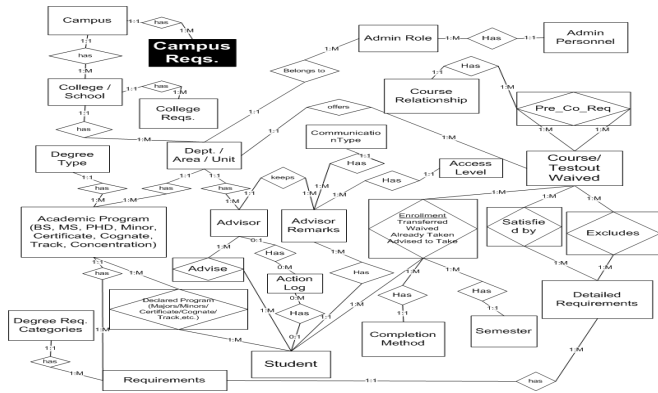


Column name	Data type	Length	Description
<u>CampusID</u>	varchar	10	Primary key-Unique identification string of a campus
Name	varchar	50	Full name of a campus
Address1	varchar	50	Building number, street name
Address2	varchar	50	
City	varchar	25	
State	char	2	
Zip	varchar	50	Usually 5 digits
Phone	varchar	12	
URL	varchar	255	Web page of a campus
DefaultRetentionTemplateID	varchar	10	

Table 3.1. SQL table structure for campus

## campus\_reqs

The *campus\_reqs* table contains information about the requirements that a student has to satisfy before officially being admitted to a campus.

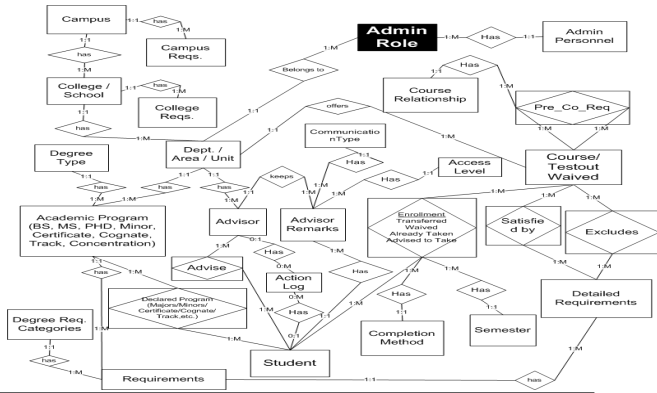


Column name	Data type	Length	Description
<u>CampusID</u>	varchar	10	Primary key-Unique identification string of the campus ask for the requirements
<u>CampusReqID</u>	int		Primary key-Unique identification string of a requirement
<u>StartAcademicTerm</u>	varchar	4	Primary key-Year and semester in which the requirement is applied
OrderOfAppearance	tinyint		
RequirementText	varchar	250	Content of the requirement
MiscNotes	varchar	250	More information or instruction about the requirement

Table 3.2. SQL table structure for *campus\_reqs*

## admin\_role

The *admin\_role* table contains information about the administrators and their administrative access to one or more departments. In addition, this table allows our system to enforce a time period during which access is granted.

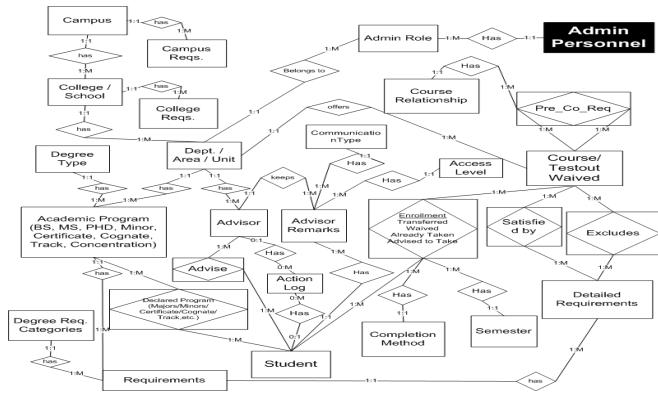


Column name	Data type	Length	Description
<u>LoginID</u>	varchar	8	Primary key-Username of the admin person
<u>DeptID</u>	varchar	10	Primary key-Unique identification string of the department administered by the admin person
<u>CollegeID</u>	varchar	10	Primary key-Unique identification string of the college to which the department belongs
<u>CampusID</u>	varchar	10	Primary key-Unique identification string of the campus to which the department belongs
StartDate	date		Begin date
EndDate	date		Expired date

**Table 3.3.** SQL table structure for *admin\_role*

## admin\_personnel

The *admin\_personnel* provides the credentials of an administrator.



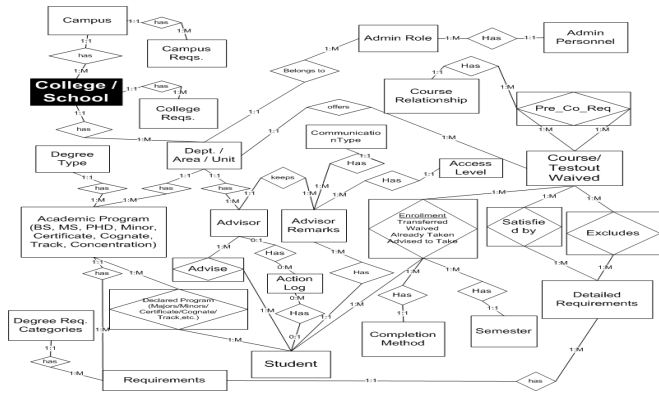
Column name	Data type	Length	Description
<u>LoginID</u>	varchar	8	Primary key-Username of admin personnel
Password	varchar	50	Secret word used for logging in
LastName	varchar	50	Family name
FirstName	varchar	50	
StartDate	date		Begin date
EndDate	date		Expired date
AccessLevel	varchar	50	Default value: USER
SiteStyle	varchar	50	Page format applied for user-Default value: iusb.css

**Table 3.4.** SQL table structure for *admin\_personnel*



## college

Similar to the *campus* table, the *college* table allows our system to maintain information about each college (or school) and it further allows us to extract specific information maintained in the system such as student, faculty, degree requirements, etc by specifying the proper college.

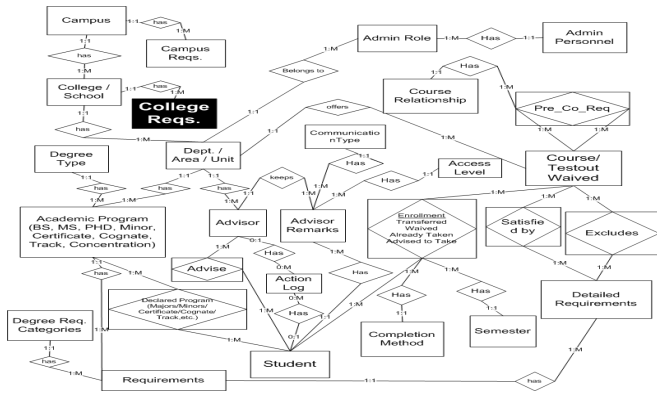


Column name	Data type	Length	Description
<u>CollegeID</u>	varchar	10	Primary key-Unique identification string of a college
<u>CampusID</u>	varchar	10	Primary key-Unique identification string of the campus to which the college belongs
Name	varchar	50	Name of the college
Address1	varchar	50	Building number, street name
Address2	varchar	50	
City	varchar	50	
State	char	2	
Zip	varchar	50	Usually 5 digits
Phone	varchar	12	
URL	varchar	255	Web page of a college

**Table 3.5.** SQL table structure for college

## college\_reqs

The *college\_reqs* table lists all the requirements of a specific college. It also provides the beginning academic term in which a requirement is applied.

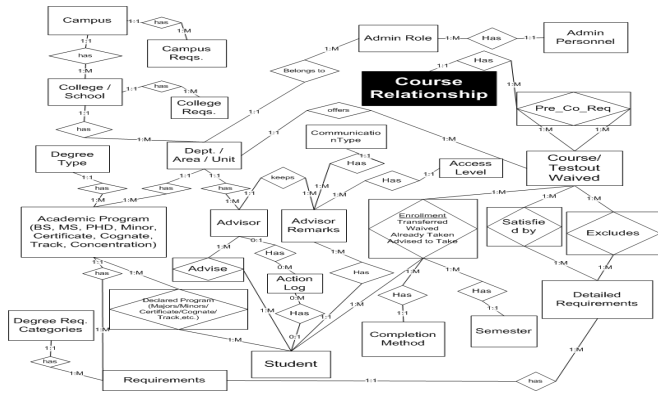


Column name	Data type	Length	Description
<u>CampusID</u>	varchar	10	Primary key-Unique identification string of the campus to which the college belongs
<u>CollegeID</u>	varchar	10	Primary key-Unique identification string of the college that ask for the requirement
<u>CollegeReqID</u>	int		Primary key-Unique identification string of the college requirement
<u>StartAcademicTerm</u>	varchar	4	Primary key-Year and semester in which the requirement is applied
OrderOfAppearance	tinyint		
RequirementText	varchar	250	Content of the requirement
MiscNotes	varchar	250	Information or instruction about the requirement

**Table 3.6.** SQL table structure for college\_reqs

### course\_relationship

This data table contains the descriptive text for a specific relationship between two courses.

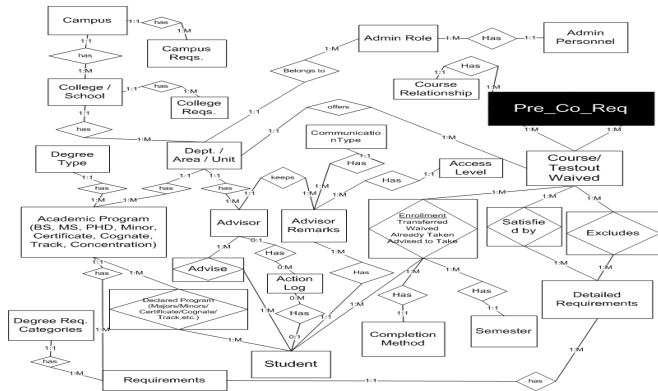


Column name	Data type	Length	Description
<u>RelationshipID</u>	varchar	10	Primary key-Unique identification string of a relationship
Description	varchar	50	Descriptive information about the relationship

Table 3.7. SQL table structure for course\_relationship

### pre\_co\_req

The pre\_co\_req table contains information about the prerequisites, equivalents and co-requisites of a particular course.

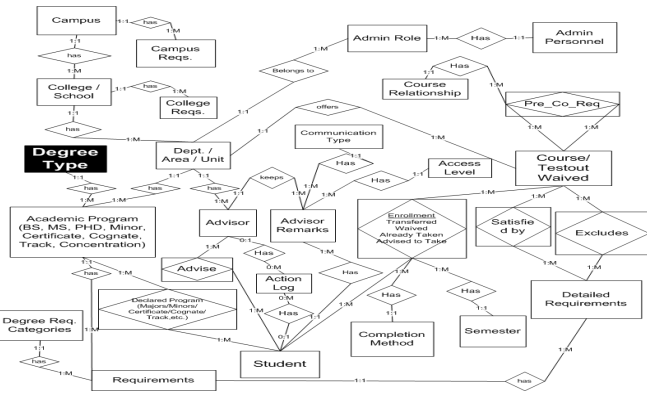


Column name	Data type	Length	Description
<u>CourseID</u>	varchar	6	Primary key-Unique identification string of a course
<u>PreCoReqCourseID</u>	varchar	6	Primary key-Unique identification string of a course
Relationship	varchar	10	Unique identification string of the relationship-Default value: PREREQ

Table 3.8. SQL table structure for pre\_co\_req

## degree\_type

This data table contains the meaningful description of a particular degree program ID.

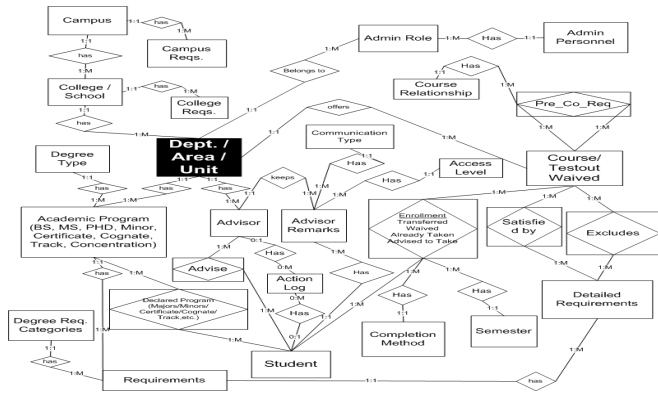


Column name	Data type	Length	Description
<u>DegreeTypeID</u>	varchar	10	Primary key-Unique identification string of the degree type
DegreeTypeDescription	varchar	250	Information or instruction about the degree type

**Table 3.9.** SQL table structure for degree\_type

## department

Similar to the *campus* and *college* table, the *department* table allows our system to maintain information about each department and it further allows us to extract specific information maintained in the system such as student, faculty, degree requirements, etc by specifying the proper department.

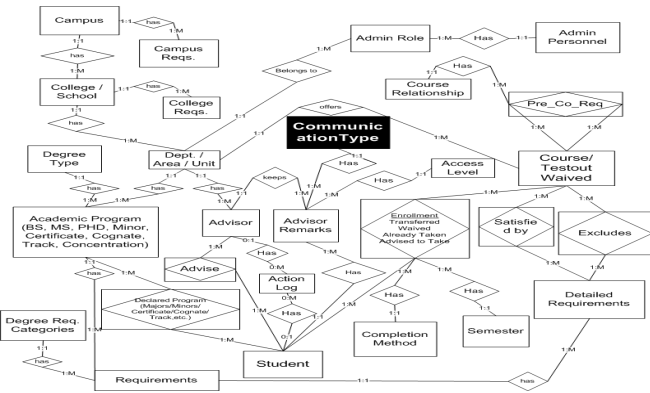


Column name	Data type	Length	Description
<u>DeptID</u>	varchar	10	Primary key-Unique identification string of the department
<u>CollegeID</u>	varchar	10	Primary key-Unique identification string of the college to which the department belongs
<u>CampusID</u>	varchar	10	Primary key-Unique identification string of the campus to which the department belongs
Name	varchar	50	Full name of the department
Address1	varchar	50	Building number, street name
Address2	varchar	50	
City	varchar	50	
State	char	2	
Zip	varchar	50	Usually 5 digits
Phone	varchar	12	
URL	varchar	255	Web page of the department
DefaultRetentionTemplateID	varchar	10	

**Table 3.10.** SQL table structure for department

### communication\_type

This table lists all the methods for contacting an advisee.

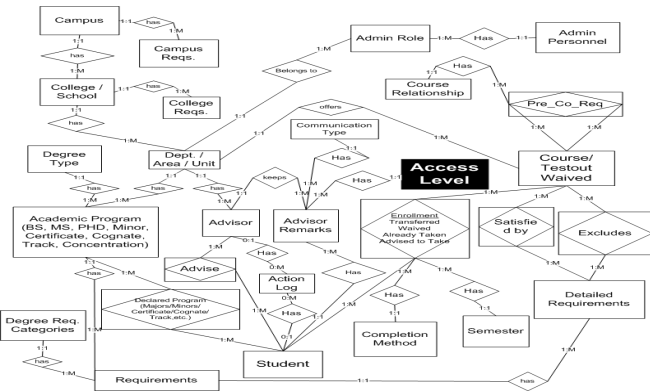


Column name	Data type	Length	Description
<u>CommTypeID</u>	varchar	7	Primary key-Unique identification string of communication type
CommTypeName	varchar	50	Name of communication type
CommTypeDescription	varchar	255	Description of communication type
DateCreated	datetime		Date and time of the record creation
UserCreated	varchar	50	Who created the record

Table 3.11. SQL table structure for communication\_type

### access\_level

The access\_level table models the privacy hierarchy for advisor remarks of IU-Advise system. It provides detail description about the scope of each level.

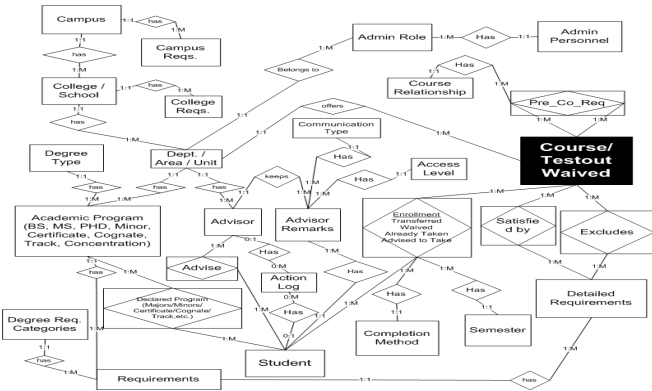


Column name	Data type	Length	Description
<u>LevelID</u>	varchar	6	Primary key-ID string for an access level
LevelDescription	varchar	255	Detail information about the level
CreationUser	varchar	10	Who created the record
CreationDate	datetime		Date of the creation

Table 3.12. SQL table structure for access\_level

**course**

This table lists all courses offered by a university. It also allows an administrator to specify the maximum and minimum number of credit hours of a course that can be counted toward a degree program's course work.

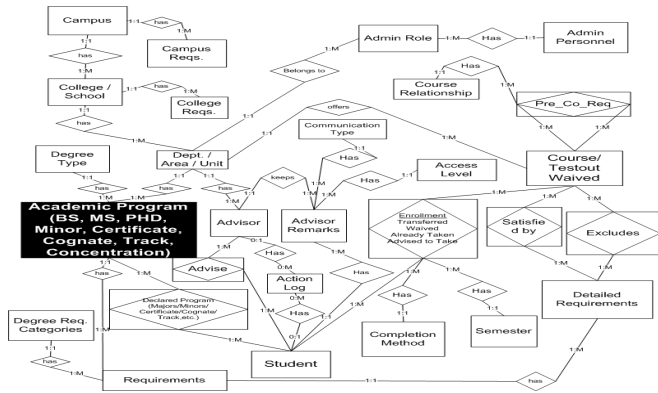


Column name	Data type	Length	Description
<u>CourseID</u>	varchar	6	Primary key-Unique identification string of a course
CourseNo	varchar	10	Course number
SubjectArea	varchar	8	The subject to which the course belongs (CSCI, etc)
CourseTitle	varchar	50	Full name of the course
DeptID	varchar	10	Unique identification string of the department which offers the course
CollegeID	varchar	10	Unique identification string of the college to which the department belongs
CampusID	varchar	10	Unique identification string of the campus to which the department belongs
MinCredits	float		Minimum of number of credit hours that a course can be counted toward a degree-Default value: 1
MaxCredits	float		Maximum of number of credit hours that a course can be counted toward a degree-Default value: 6

**Table 3.13.** SQL table structure for course

## academic\_program

The *academic\_program* table provides information about a degree program. It supplies the department ID to which the degree program belongs to and the beginning academic term that the degree is offered. The course work and GPA requirements are also specified here.



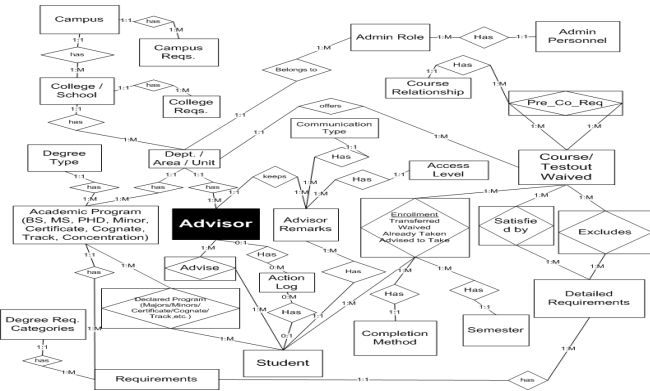
Column name	Data type	Length	Description
<u>DegreeID</u>	varchar	50	Primary key-Unique identification string of a degree program
<u>CampusID</u>	varchar	10	Primary key-Unique identification string of the campus to which the college belongs
<u>CollegeID</u>	varchar	10	Primary key-Unique identification string of the college to which the department belongs
<u>DeptID</u>	varchar	10	Primary key-Unique identification string of the department which offers the degree program
<u>StartAcademicTerm</u>	varchar	4	Primary key-Year, semester that the degree program is offered
DegreeDescription	varchar	250	Descriptive information for a degree program
DegreeTypeID	varchar	10	Unique identification number for a degree type
Credits_Required	tinyint		Number of credit hours is required for a degree
MinGPARequired	float		The minimum GPA required for a degree program.

**Table 3.14.** SQL table structure for *academic\_program*



# advisor

This table contains contact information of an advisor.

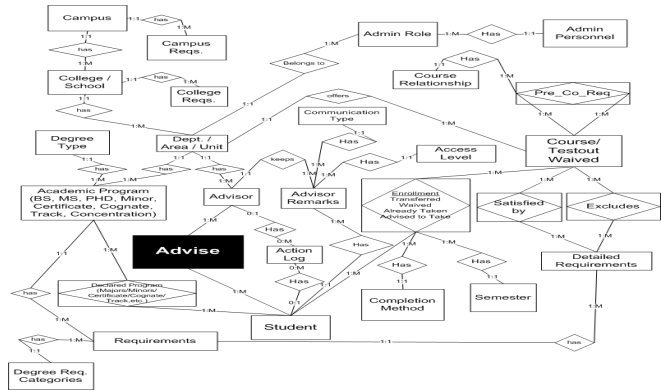


Column name	Data type	Length	Description
<u>AdvisorID</u>	varchar	8	Primary key-Unique identification string of an advisor
<u>CampusID</u>	varchar	10	Primary key-Unique identification string of the campus to which the advisor belongs
<u>CollegeID</u>	varchar	10	Primary key-Unique identification string of the college to which the advisor belongs
<u>DeptID</u>	varchar	10	Primary key-Unique identification string of the department to which the advisor belongs
LastName	varchar	50	Family name
FirstName	varchar	50	
Email	varchar	50	Primary email (usually IU South Bend email)
Password	varchar	50	Secret word of an advisor in order to login

**Table 3.15.** SQL table structure for advisor

advise

The *advise* table maintains information about advisors and their advisees. A given advisor may be paired to more than one advisee. Similarly, a given advisee may have more than one advisor.

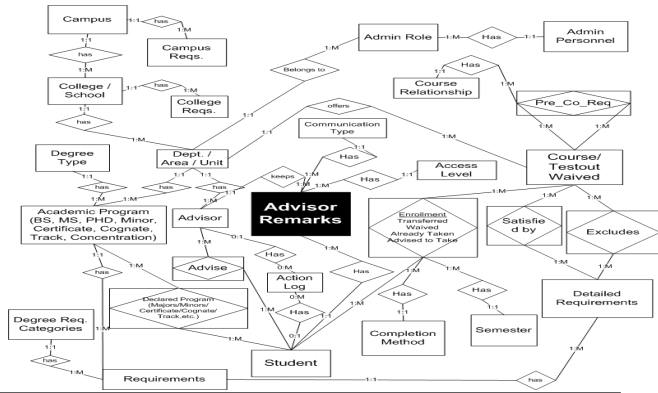


Column name	Data type	Length	Description
<u>StudentID</u>	varchar	15	Primary key-Unique identification string of a student
<u>AdvisorID</u>	varchar	8	Primary key-Unique identification string of the advisor who advises the student who advises the student
<u>DeptID</u>	varchar	10	Primary key-Unique identification string of the department to which the advisor belongs
<u>AcademicTerm</u>	varchar	4	Primary key-Academic term in which the advisor is assigned to a student
CollegeID	varchar	10	Unique identification string of the college to which the advisor belongs
CampusID	varchar	10	Unique identification string of the campus to which the advisor belongs

Table 3.16. SQL table structure for advise

## advisor\_remark

The *advisor\_remark* table contains all the remarks that an advisor may make about their advisees. After each advising session, the advisor's remarks, the privacy level, and the status of the remark are saved to this table.

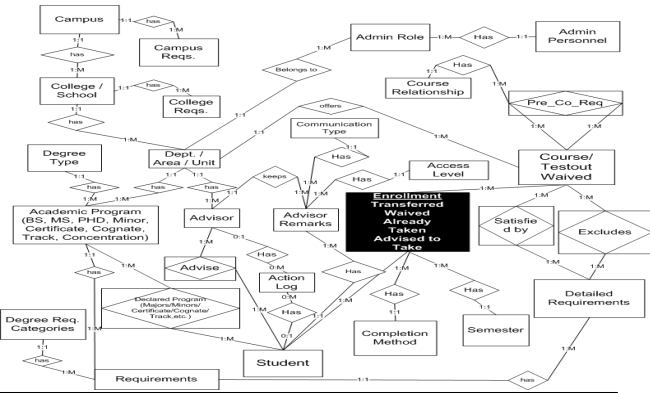


Column name	Data type	Length	Description
<u>AdvisorID</u>	varchar	8	Primary key-Unique identification string of each advisor
<u>DeptID</u>	varchar	10	Primary key-Unique identification of the department to which the advisor belongs
<u>CollegeID</u>	varchar	10	Primary key-Unique identification of the college to which the advisor belongs
<u>CampusID</u>	varchar	10	Primary key-Unique identification string of the campus to which the advisor belongs
<u>StudentID</u>	varchar	15	Primary key-Unique identification string for a student
<u>AdvisingDate</u>	datetime		Primary key-Date of advising session-Default value: 0000-00-00 00:00:00
AdvisorRemark	text	65535	Content of the notes
CommunicationType	varchar	7	The way a student uses to contact an advisor-Default value: comm03
Active	int		Is a public remark is active-Default value: 1
Level	varchar	6	Authorization level of an advisor remark

Table 3.17. SQL table structure for *advisor\_remark*

# enrollment

The *enrollment* table captures all the courses that a student has taken. It also maintains the method by which the student was able to complete the course (passing grade, test-out, etc.)

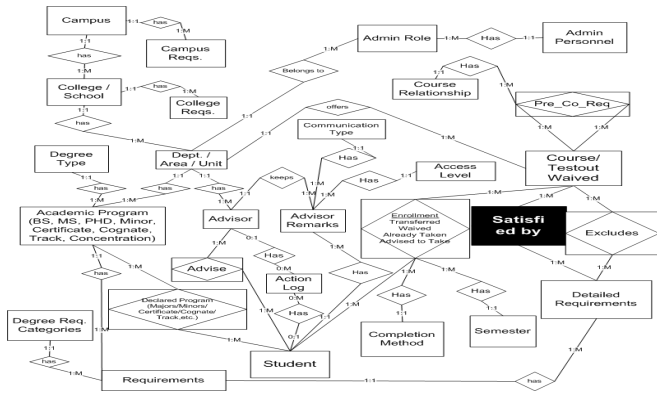


Column name	Data type	Length	Description
<u>StudentID</u>	varchar	15	Primary key-Unique identification string of the student
<u>CourseID</u>	varchar	10	Primary key-Unique identification string of the course
<u>AcademicTerm</u>	varchar	4	Primary key-Year and semester the student take/took/will take the course
Grade	varchar	4	Final grade -Default value: TBD
CompletionMethodID	varchar	25	How the student complete the course
Explanation	varchar	250	More information about the enrollment

**Table 3.18.** SQL table structure for enrollment

satisfied\_by

This table presents the relationship between a detailed degree requirements and courses. It shows a list of courses that can be used for satisfying a particular detailed requirement.

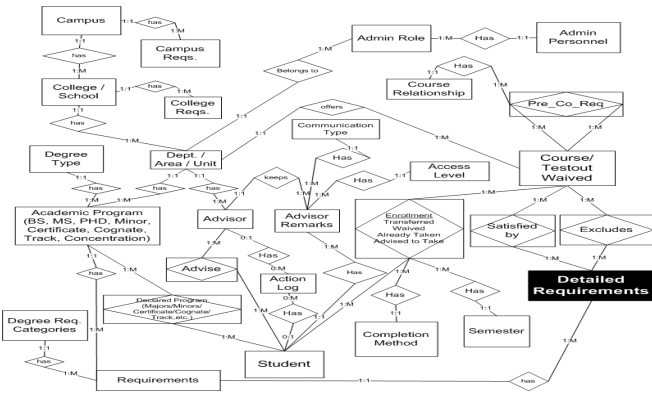


Column name	Data type	Length	Description
<u>DetailedRequirementID</u>	int		Primary key
<u>CourseID</u>	varchar	6	Primary key
MinGradeRequired	varchar	10	Default value: NA
OtherRequirements	varchar	250	
Notes	varchar	250	

Table 3.19. SQL table structure for satisfied\_by

## detailed\_requirement

The *detailed\_requirement* lists all the requirements of a degree requirement category. User can specify the minimum and maximum number of credit hours, notes and the descriptive text for each detailed requirement.

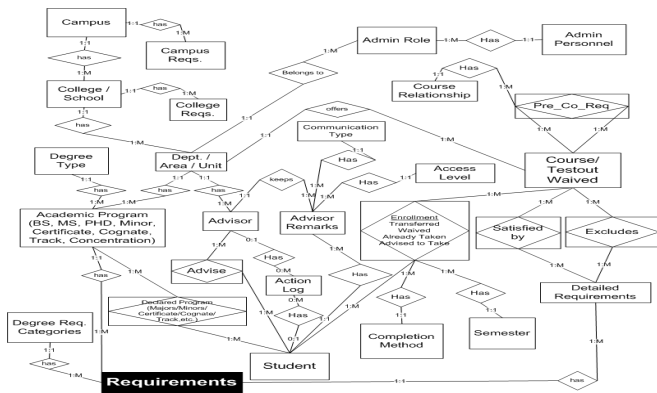


Column name	Data type	Length	Description
<u>DetailedRequirementID</u>	int		Primary key-Unique identification string of the detailed requirement
RequirementID	int		Unique identification string of the requirement to which the detailed requirement belongs
OrderOfAppearance	tinyint		
DetailedRequirementText	varchar	50	Content of the detailed requirement
MinCreditsRequired	tinyint		Minimum of number of credit hours required
MaxCreditsRequired	tinyint		Maximum of number of credit hours that student can take
MiscNotes	varchar	250	Information or instruction about the detailed requirement

**Table 3.20.** SQL table structure for *detailed\_requirement*

## requirements

The *requirements* table lists all the degree requirements for a particular academic program. It also provides the starting semester when the requirement must be enforced as well as total number of credits needed to complete the degree.

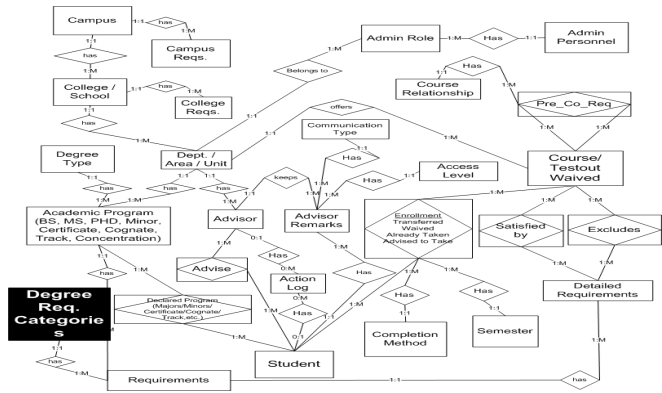


Column name	Data type	Length	Description
<u>RequirementID</u>	int		Primary key-Unique identification string of the requirement
OrderOfAppearance	tinyint		
CategoryID	varchar	25	Unique identification string of the category to which the requirement belongs
RequirementText	varchar	50	Description about the requirement
MinCreditsRequired	tinyint		Minimum of number of credit hours required by the requirement
MaxCreditsRequired	tinyint		Maximum of number of credit hours required by the requirement
MiscNotes	varchar	250	Information or instruction about the requirement
CampusID	varchar	10	Unique identification string of the campus to which the department belongs
CollegeID	varchar	10	Unique identification string of the college to which the department belongs
DeptID	varchar	10	Unique identification string of the department which offers the degree program
DegreeID	varchar	50	Unique identification string of the degree program to which the requirement belongs
StartAcademicTerm	varchar	4	Year and semester of beginning of applying

**Table 3.21.** SQL table structure for requirements

## degree\_req\_categories

The *degree\_req\_categories* shows the broad requirement categories for a particular degree. Such categories may include General Education, Language, Math, Science, etc.



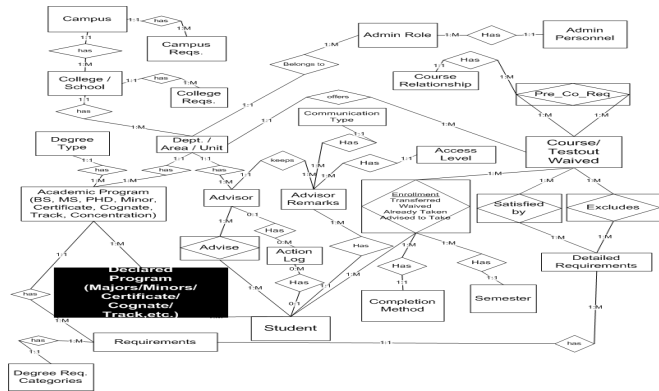
Column name	Data type	Length	Description
<u>CategoryID</u>	varchar	25	Primary key-Unique identification string of the category
CategoryText	varchar	250	Content of the category
OrderOfAppearance	tinyint		
MiscNotes	varchar	250	Information or instruction about the category

**Table 3.22.** SQL table structure for *degree\_req\_categories*



### declared\_program

This *declared\_program* table captures the information about a student's declared major or majors. This information is necessary to ensure the proper degree requirements are selected and applied to this student.

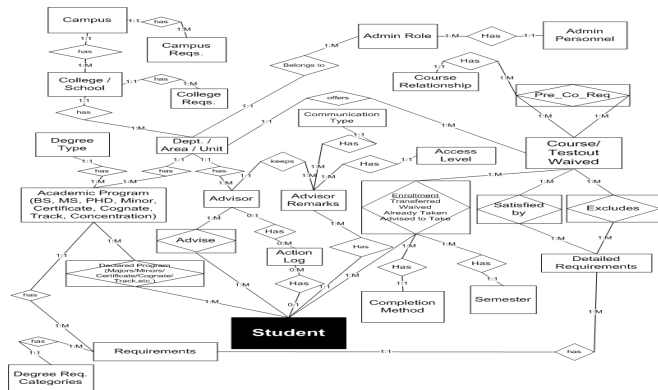


Column name	Data type	Length	Description
<u>StudentID</u>	varchar	15	Primary key-Unique identification string of a student
<u>DegreeID</u>	varchar	50	Primary key-Unique identification string of the degree program that the student wants to pursue
<u>CampusID</u>	varchar	10	Primary key-Unique identification string of the campus to which the department belongs
<u>CollegeID</u>	varchar	10	Primary key-Unique identification string of the college to which the department belongs
<u>DeptID</u>	varchar	10	Primary key-Unique identification string of the department that offers the degree program
DeclaredAcademicTerm	varchar	4	Year and semester of beginning of offering

**Table 3.23.** SQL table structure for *declared\_program*

# student

The *student* table captures demographic information about a student. It also maintains the student's picture and admission status.

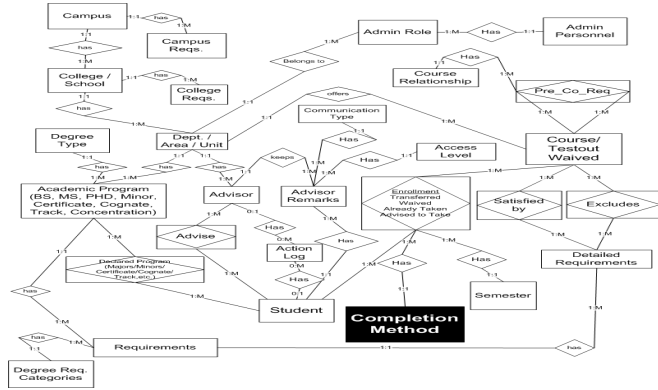


Column name	Data type	Length	Description
<u>StudentID</u>	varchar	15	Primary key-Unique identification string of a student
NetworkID	varchar	10	IU username of the student
Password	varchar	50	Secret word used for logging in
LastName	varchar	50	Family name
FirstName	varchar	50	
Email	varchar	50	Primary email (IU South Bend email)
AlternateEmail	varchar	50	Secondary email address (beside IU email address)
Address1	varchar	50	Building/house number, street name
Address2	varchar	50	Apartment number
City	varchar	25	
State	char	2	
Zip	varchar	50	Usually 5 digits
Phone	varchar	12	Primary phone number (usually home phone)
MobilePhone	varchar	12	Cellular phone number
DeptID	varchar	10	Unique identification string of the department to which the student is admitted to (it may differ from the department of declared program)
CollegeID	varchar	10	Unique identification string of the college to which the department belongs
CampusID	varchar	10	Unique identification string of the campus to which the department belongs
DegreeID	varchar	50	Unique identification of the degree program to which the student is admitted
TermAdmitted	varchar	4	The year and semester of admission
BirthYear	int		Year of birth of student
AdmissionStatus	enum	4	Admission condition of student: AFQL - admitted fully qualified, APRS - admitted probation-enum('AFQL','APRS')
ImageUrl	longtext	2147483647	Link to student picture

**Table 3.24.** SQL table structure for student

## completion\_method

This table provides description for the method by which a course is completed.

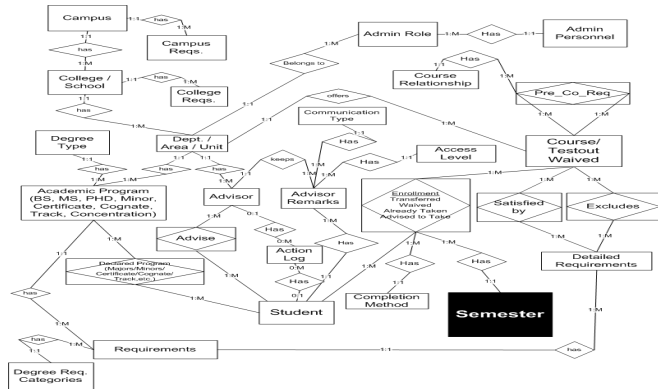


Column name	Data type	Length	Description
<u>CompletionMethodID</u>	varchar	25	Primary key-Unique identification string of the completion method
Description	varchar	250	Descriptive information of the completion method

Table 3.25. SQL table structure for completion\_method

## semester

The *semester* table supplies detail information of an academic term which includes the academic semester and year.

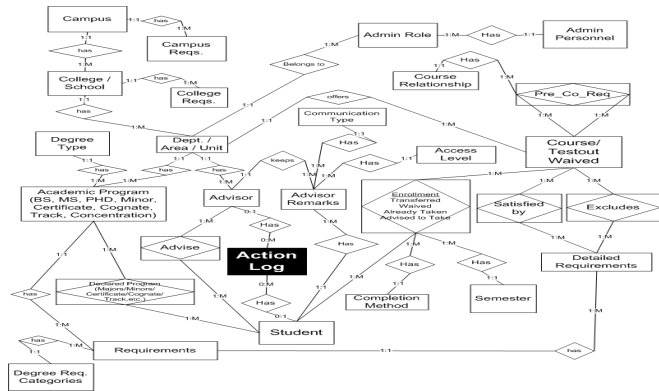


Column name	Data type	Length	Description
<u>AcademicTerm</u>	varchar	4	Primary key
Year	int		
Semester	varchar	10	
BeginOfSemesterDate	date		Default value: 0000-00-00
EndOfSemesterDate	date		Default value: 0000-00-00
CurrentTerm	tinyint		

Table 3.26. SQL table structure for semester

**actionlog**

This table maintains a log of all login attempts to the IU-Advise system. The information in this table is used to limit the number of login attempts a user can make.



Column name	Data type	Length	Description
<u>ActionID</u>	bigint		Primary key-ID for an action
ActionDescription	varchar	255	What user does
LoggedDateTime	datetime		When the action happens
LoggedUser	varchar	50	Who invokes the action
Result	varchar	20	Result of the action

**Table 3.27.** SQL table structure for actionlog

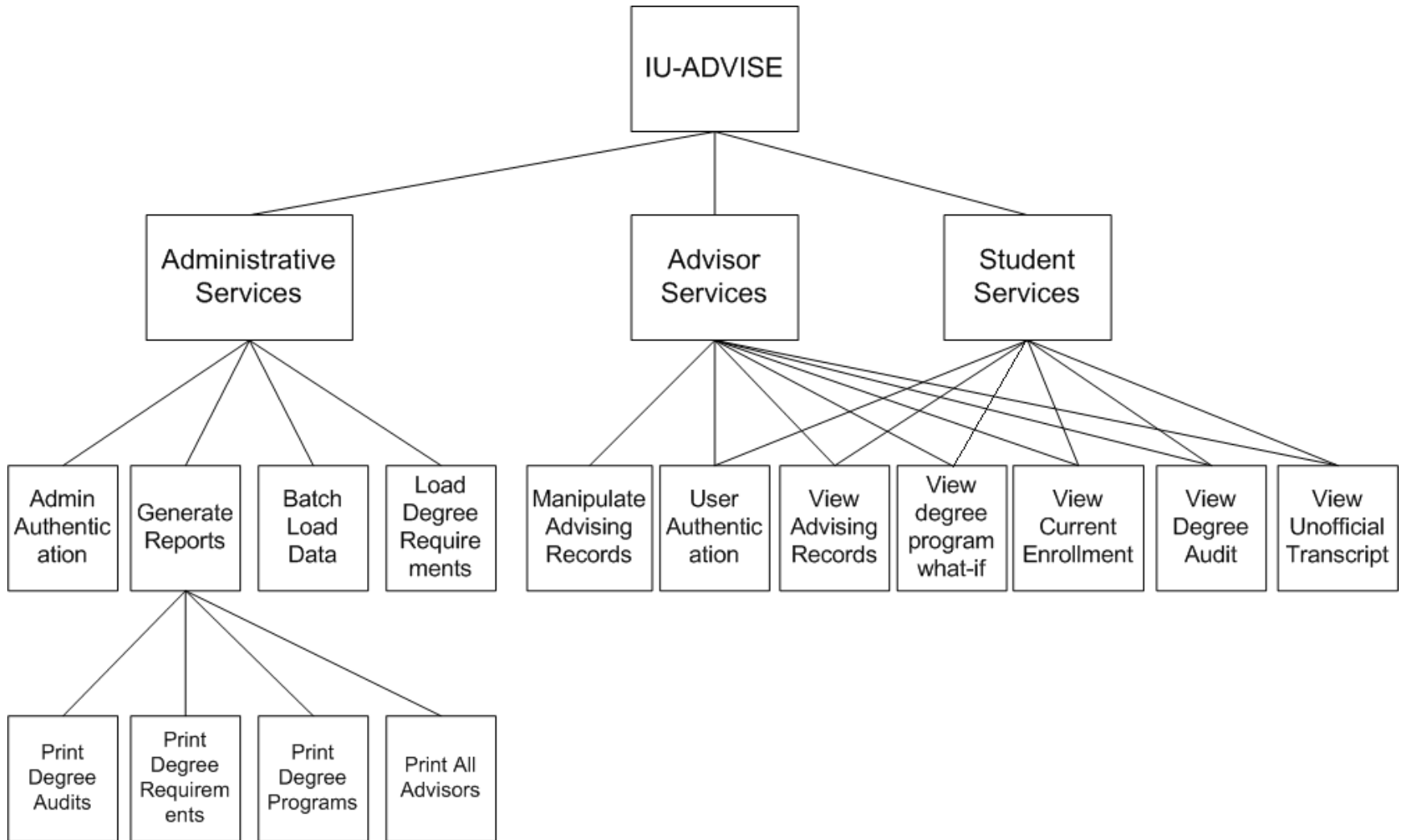
## 3.4. Process Model

### 3.4.1. Functional Decomposition Diagram

Figure 3.2 shows the functionalities supported by the IU-Advise system [15]. The system is divided into three subsystems: Student services, Advisor services, and Administrative services. The scope of this thesis is limited to the functionalities provided under advisor and student services.

Before providing accesses to functionalities, a user must be authenticated and granted a valid role and privileges. Once the user is authenticated and authorized as a student, he or she has access to degree audit, unofficial transcript, and current enrollment information reports. The user can use the “what-if” report feature to determine how his or her current courses fit with other degree requirements. An authenticated advisor has similar access, however, advisors have additional capability to create and store advisor remarks.

In the following sections, we will look into how these functionalities are decomposed and implemented by developing a series of data flow diagrams (DFD).

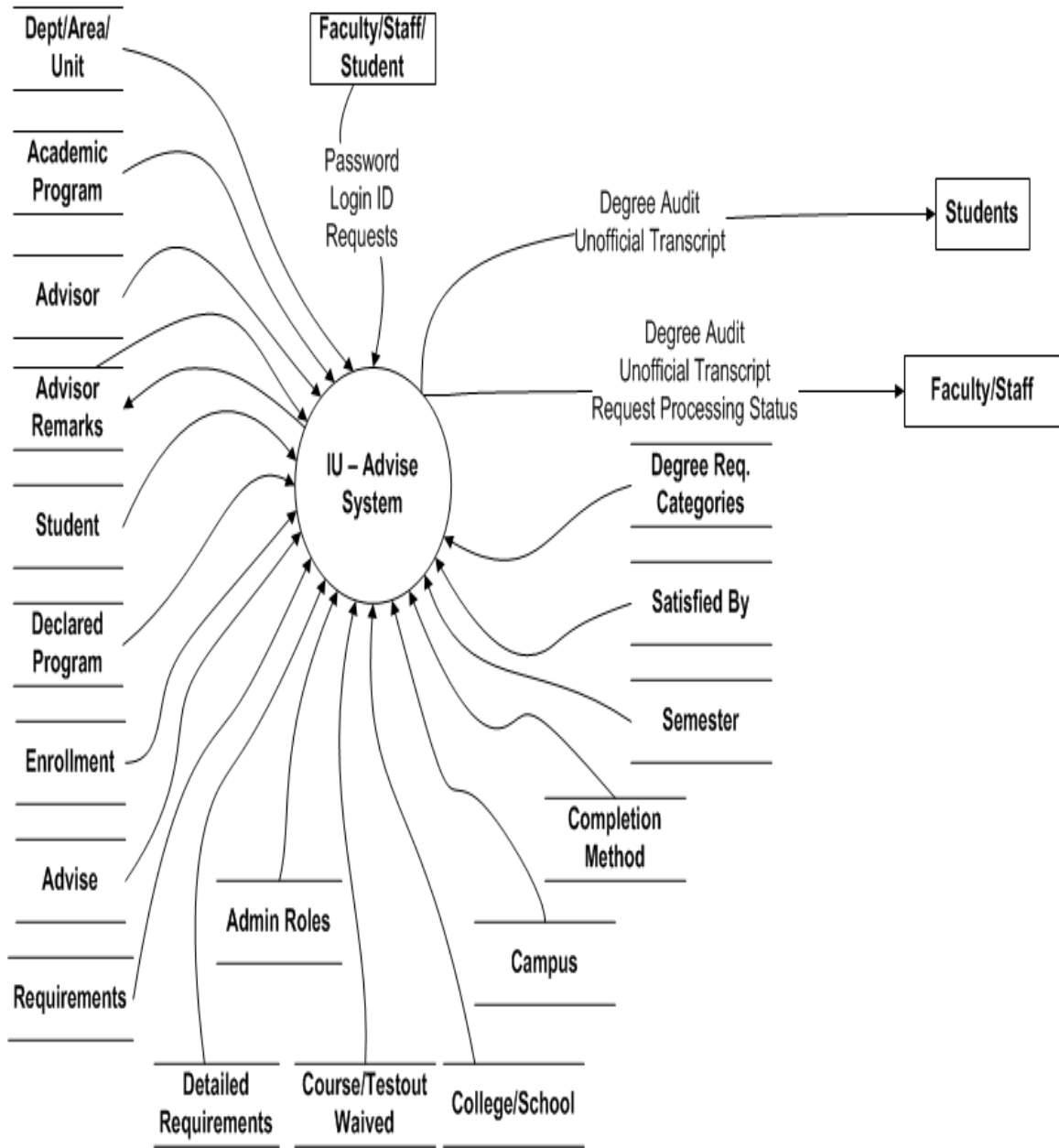


**Figure 3.2.** *Functional Decomposition Diagram*

### 3.4.2. Data Flow Diagrams

To model the functionalities of the system, we use data flow diagrams (DFD). In a data flow diagram, a rectangle represents an external agent (such a user) that interacts with system. The parallel lines represent the data stores (such as data tables in the database) from which a process retrieves and stores information. An oval represents a process (such as a code module). There are two types of diagrams: context and detailed DFD. A context diagram represents the entire system as one single abstract process. It also shows the interaction of the system with external agents, other processes and data stores. A detailed DFD decomposes the abstract process in the context level into subprocesses. At this level, the DFD diagram shows how subprocesses interrelate to each others and how data is modified while flowing through the internal processes. All the data stores in the detailed DFD are matched with those in the context DFD.

Figure 3.3 shows the overall context level data flow diagram of the IU-Advise system. Students, faculty and staff are the main constituents of the IU-Advise system. Figure 3.3 also shows which data tables are used for data processing. Once the overall interaction of the system with its environment is identified, we can decompose and discuss its constituent parts. In the following section (from page 37-58 and Figure 3.4 to Figure 3.25) we will systematically decompose, display and describe each subsystem using a context level and detailed level data flow diagram.

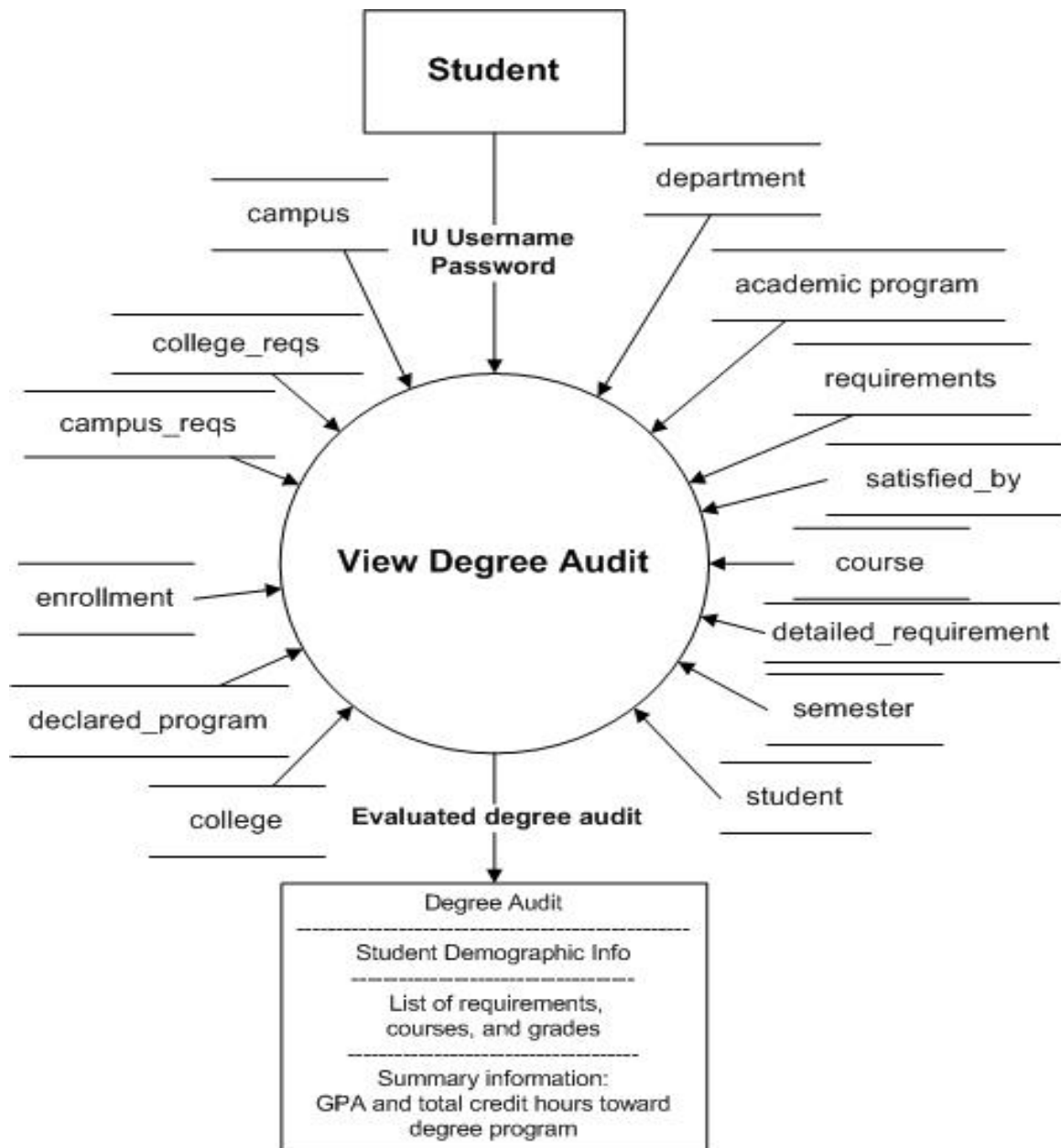


**Figure 3.3.** Context Data Flow Diagram of IU-Advise system



### 3.4.2.1. Student Functionality

An authenticated student can view his or her degree audit. Figure 3.4 shows data tables, the expected result and the input for producing a degree audit. The details for retrieving data and producing degree audit for a student are shown in Figure 3.5.



**Figure 3.4.** Student - View Degree Audit Context DFD

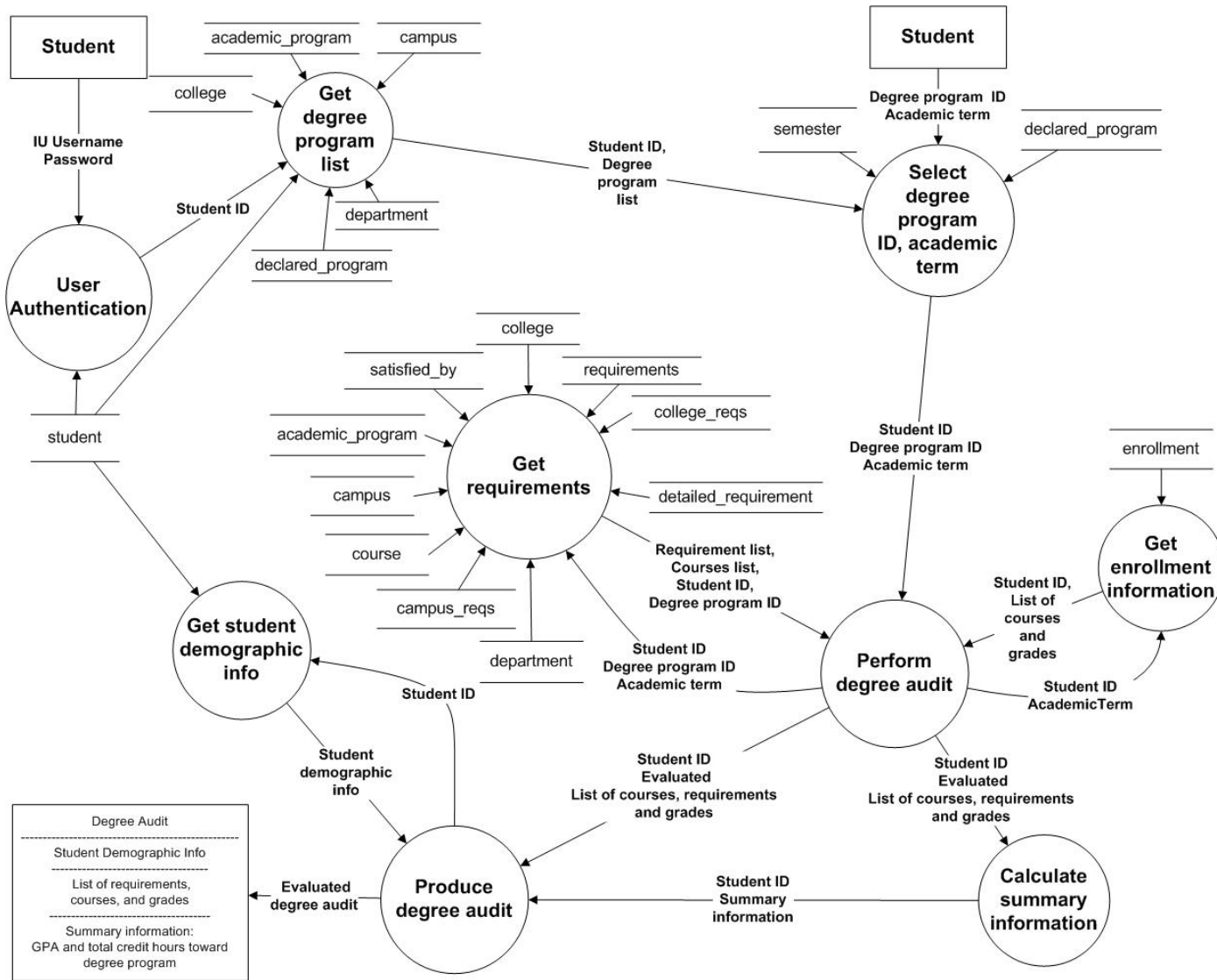
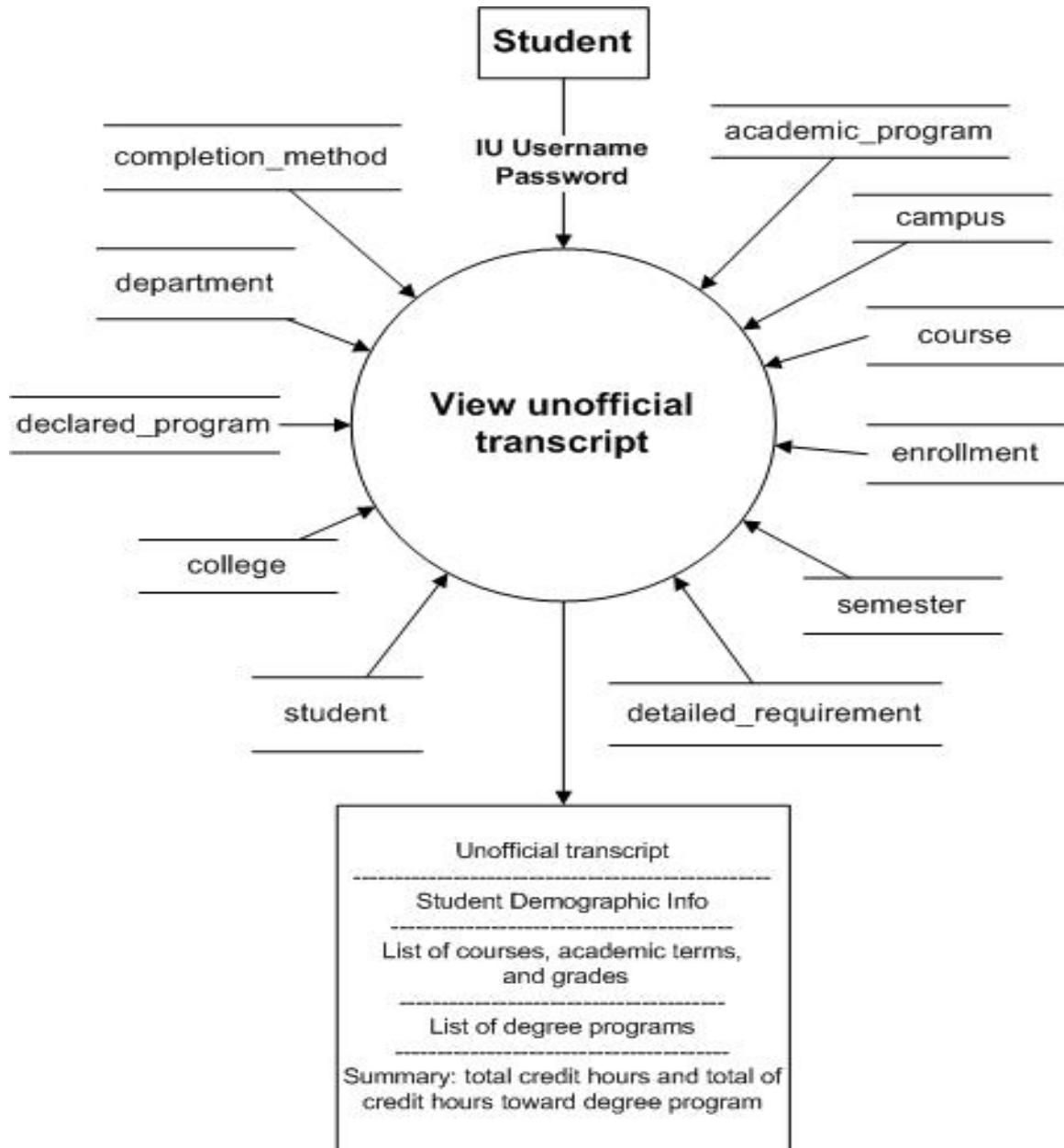


Figure 3.5. Student - View Degree Audit Detailed DFD

In addition to degree audit, another useful report is the unofficial transcript. Similar to degree audit, the context and the detailed DFDs are shown in Figures 3.6 and 3.7.



**Figure 3.6.** Student - View Unofficial Transcript Context DFD

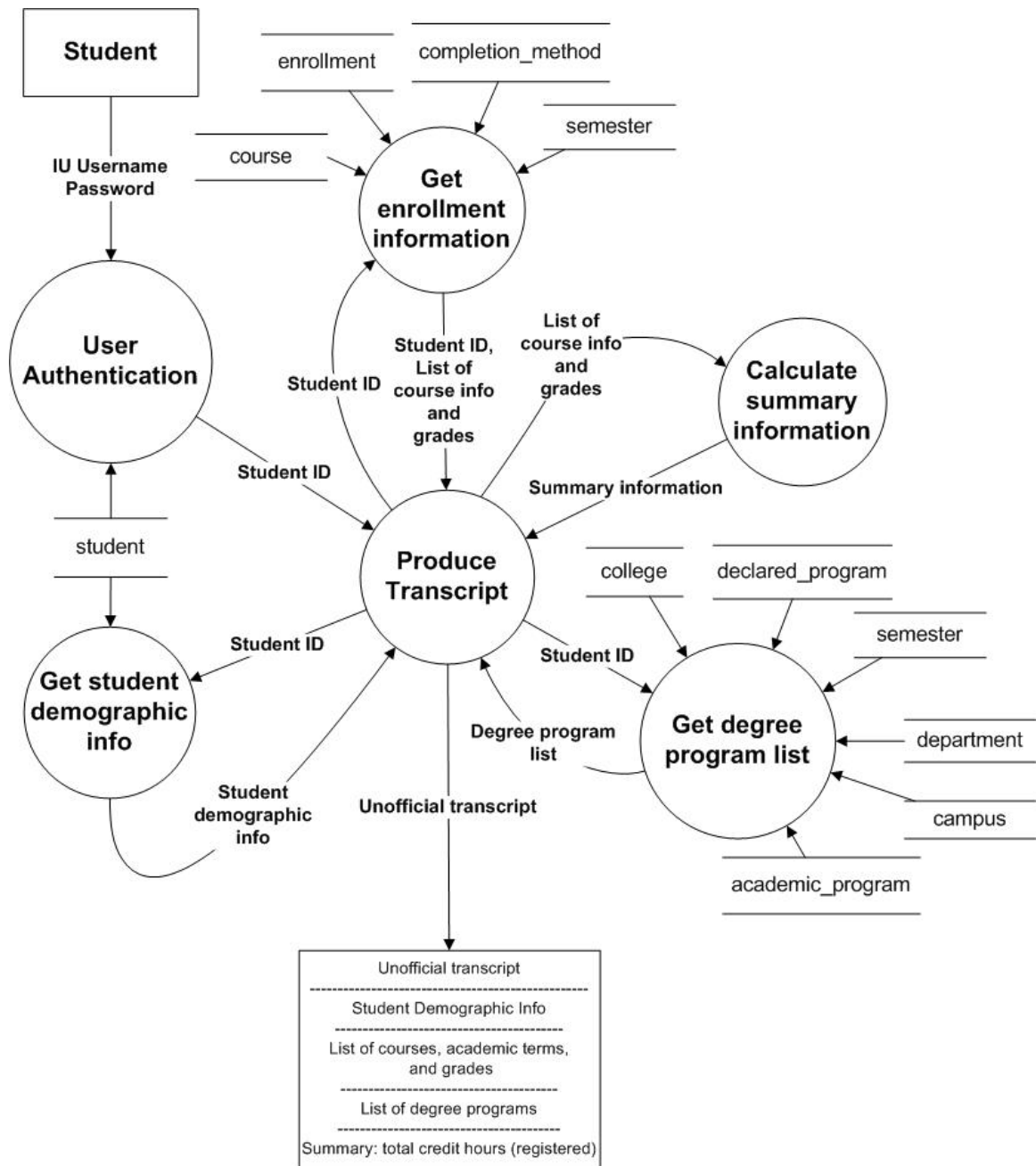
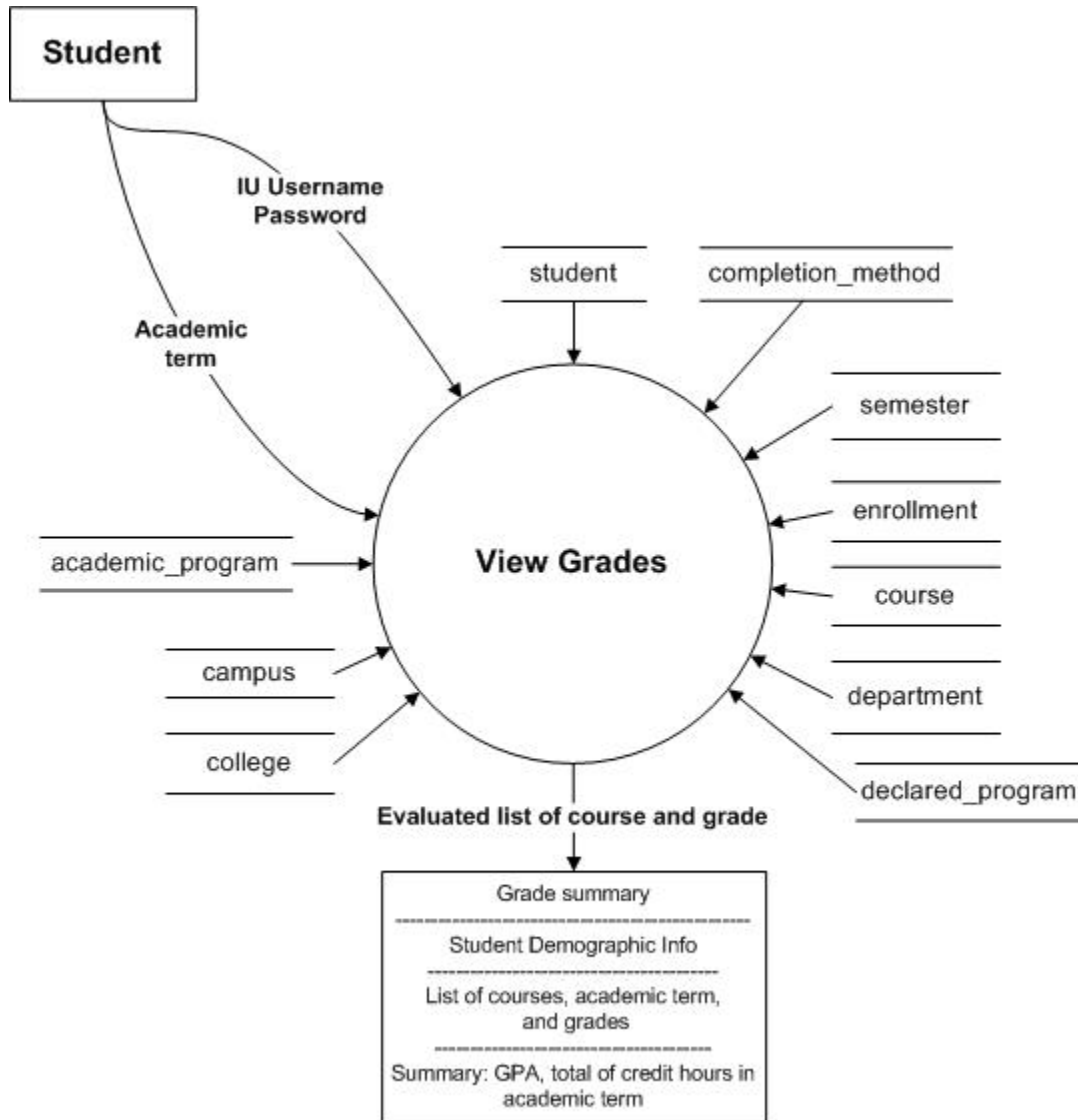


Figure 3.7. Student - View Unofficial Transcript Detailed DFD

Figures 3.8 and 3.9 shows what data are needed and how they are used for generating the grade report of a specific student. A student needs to provide the specific academic year and semester to produce an appropriate report.



**Figure 3.8.** Student - View Grades Context DFD

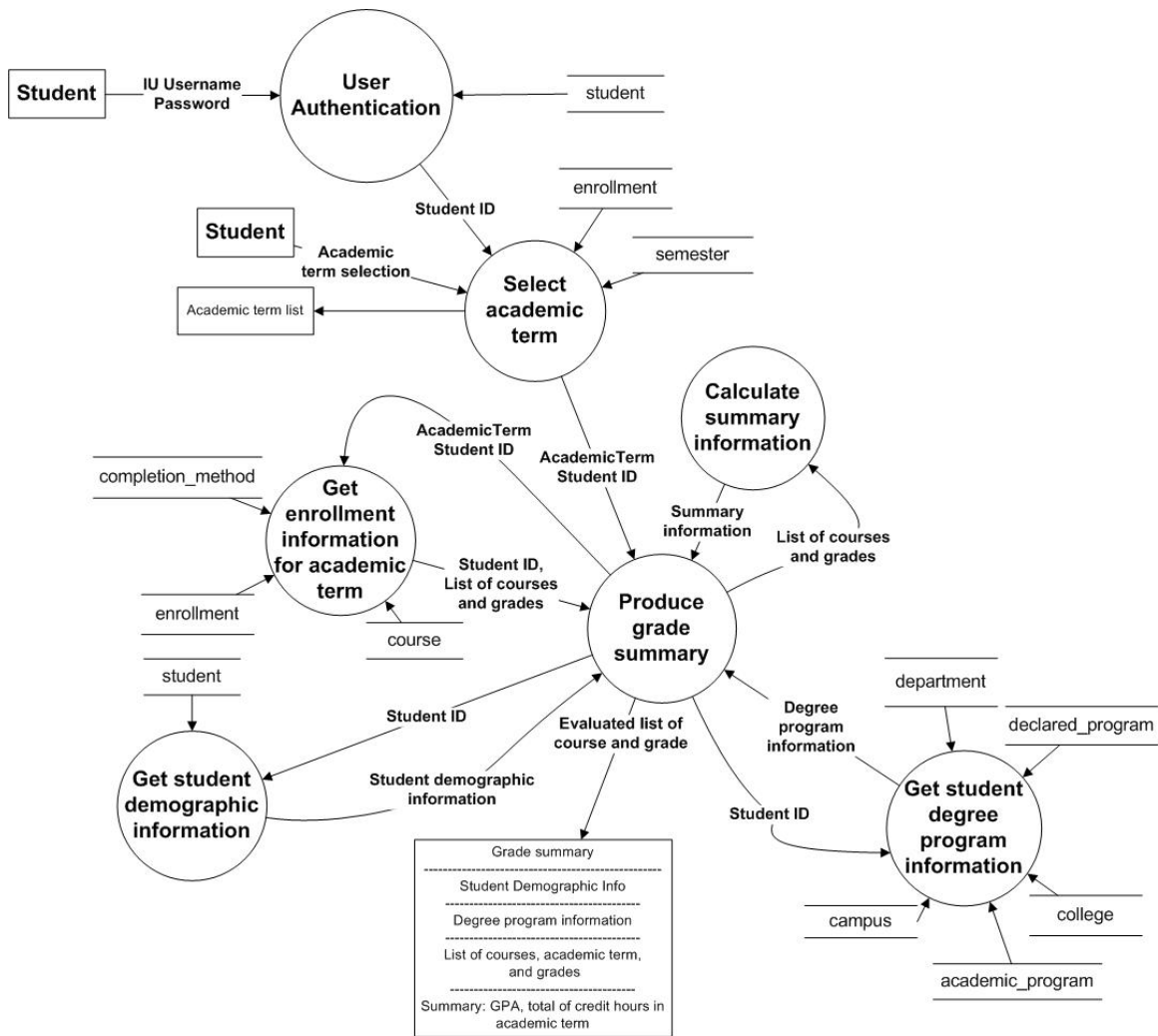
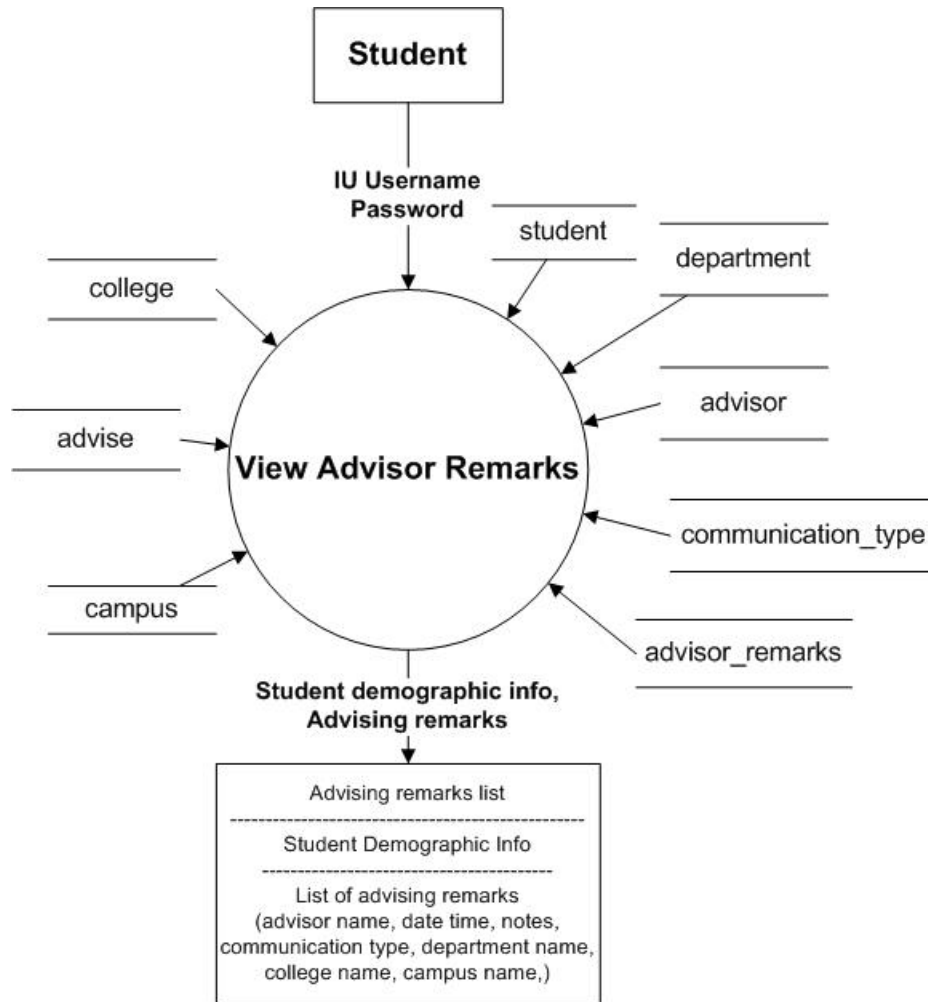


Figure 3.9. Student - View Grades Detailed DFD

A student can also review advisor remarks in public level from his or her advisors.

Figure 3.10 and 3.11 show how the system produces this list.



**Figure 3.10.** *Student - View Advisor Remarks Context DFD*

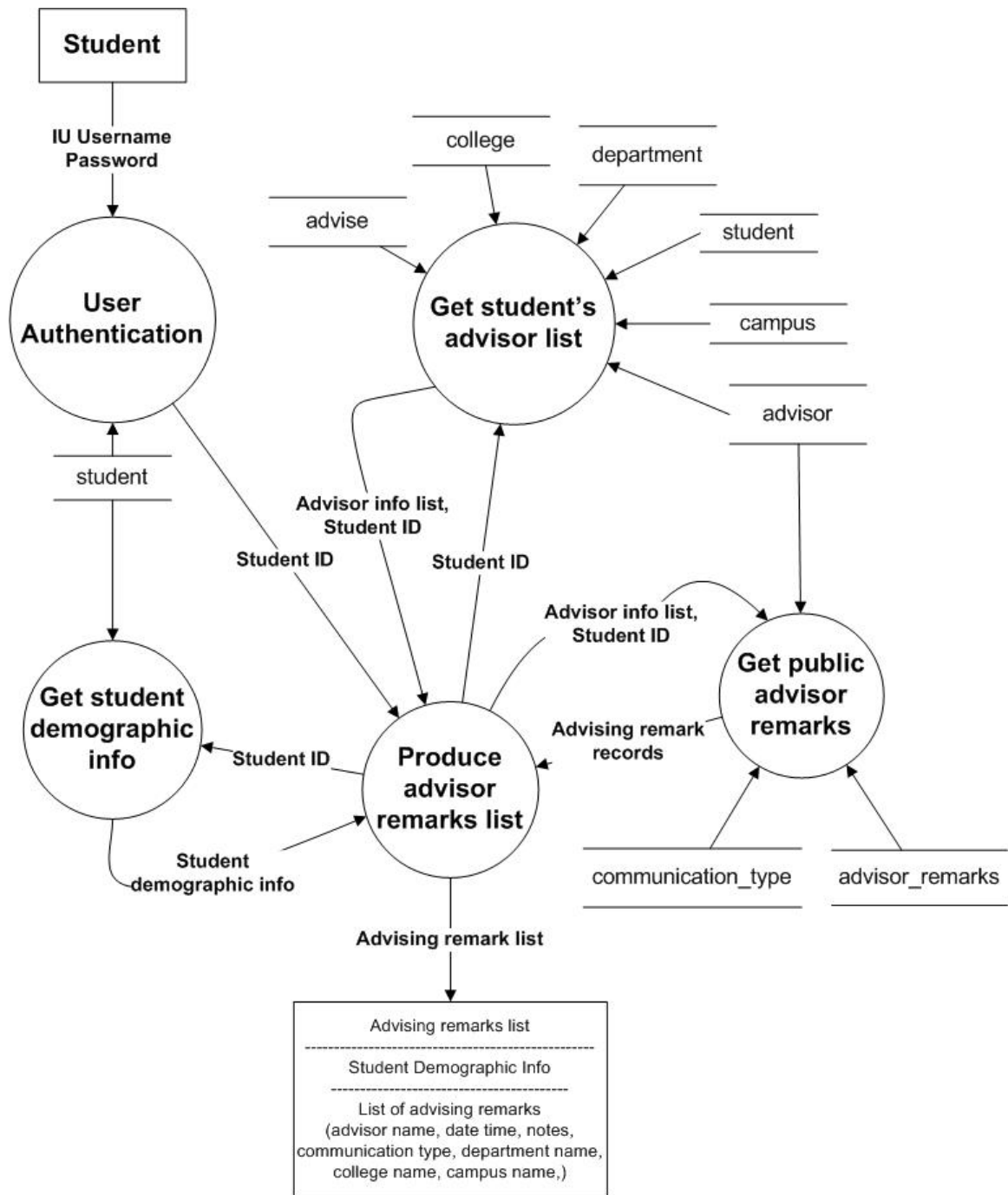
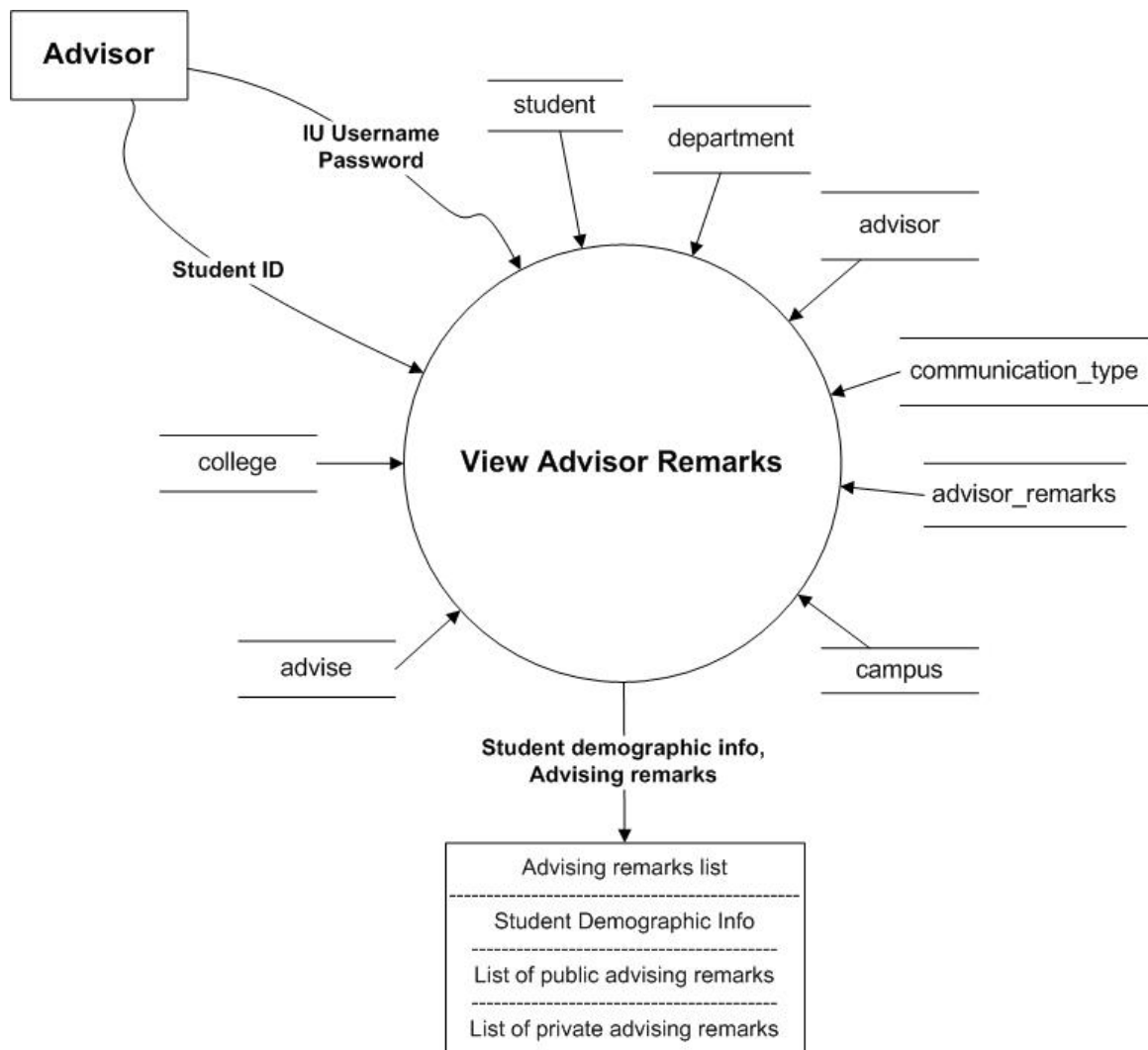


Figure 3.11. Student - View Advisor Remarks Detailed DFD

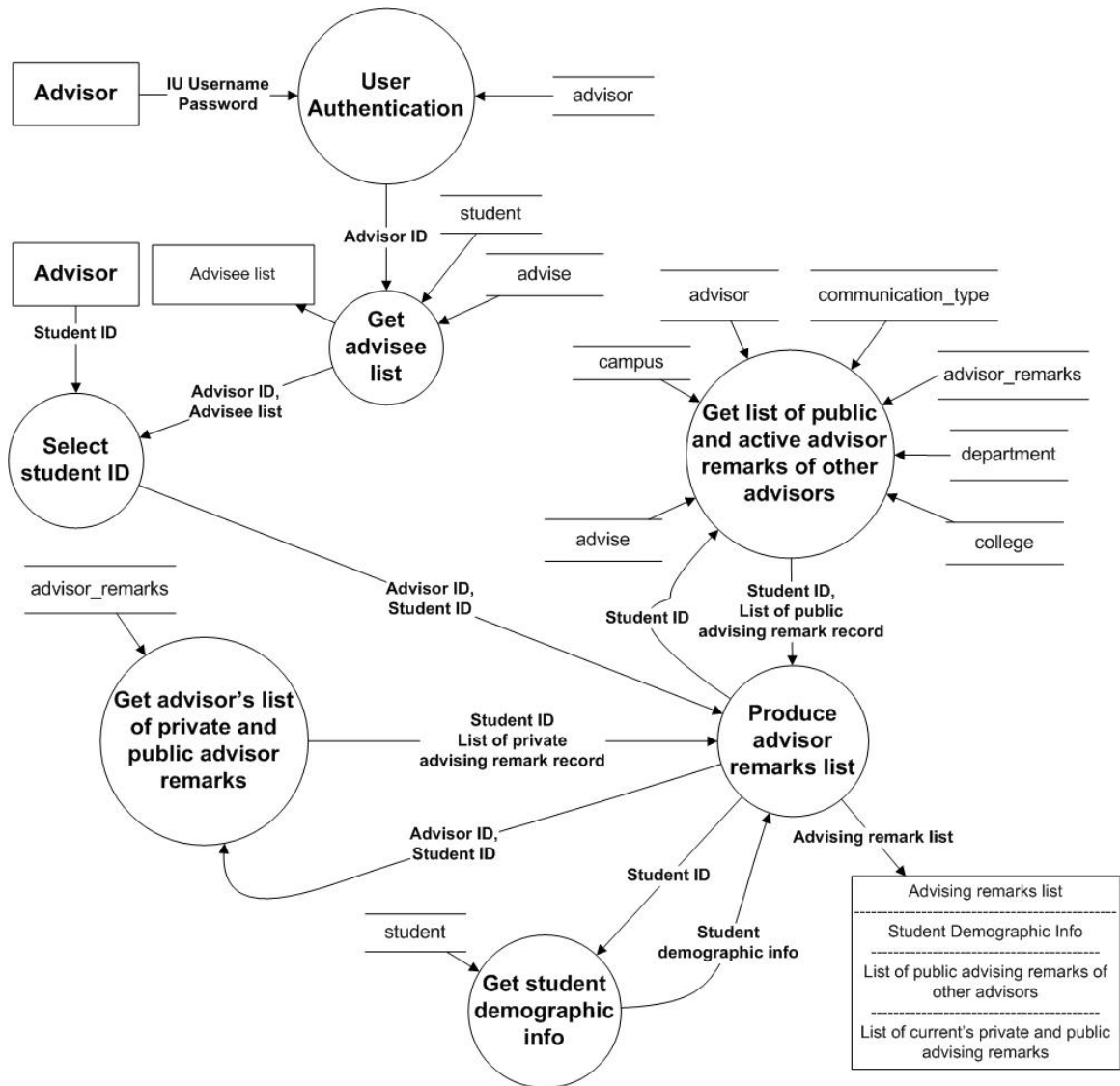


### 3.4.2.2. Advisor Functionality

Figure 3.12 and 3.13 present how an advisor views advisor remarks. An advisor can view all advisor remarks in public and advisor group level. Advisor remarks in advisor level can only be viewed by their owner.



**Figure 3.12.** Advisor - View Advisor Remarks Context DFD



**Figure 3.13.** *Advisor - View Advisor Remarks Detailed DFD*

The “View Degree Audit” function for advisors described in Figure 3.14 and 3.15 is essentially the same as “View Degree Audit” described under student functionalities.



**Figure 3.14.** *Advisor - View Degree Audit Context DFD*

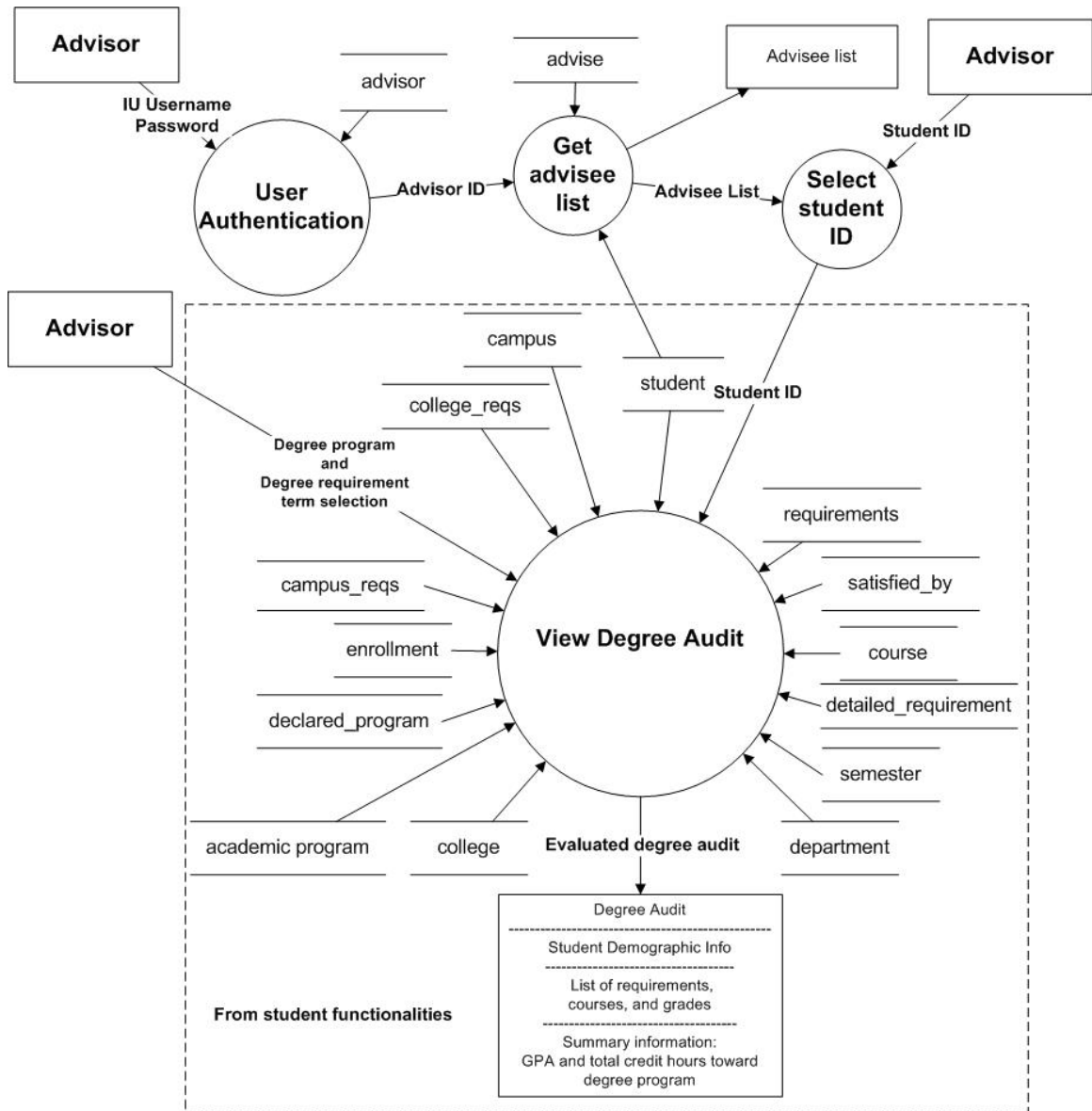
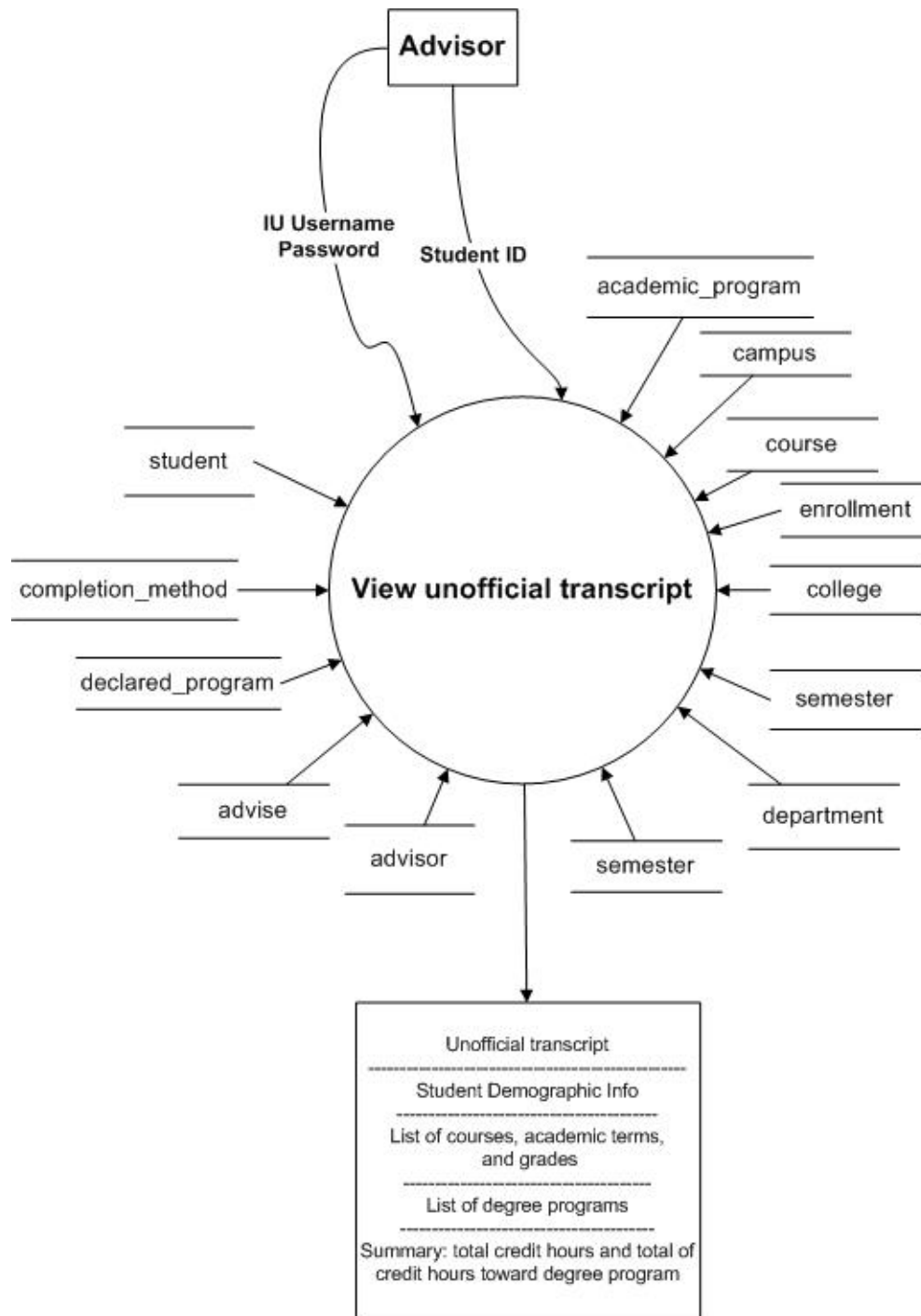


Figure 3.15. Advisor - View Degree Audit Detailed DFD

An advisor can view the “Unofficial Transcript” of a specified student. The process in Figure 3.16 and 3.17 makes use the same procedure described under student’s functionalities.



**Figure 3.16.** *Advisor - View Unofficial Transcript Context DFD*

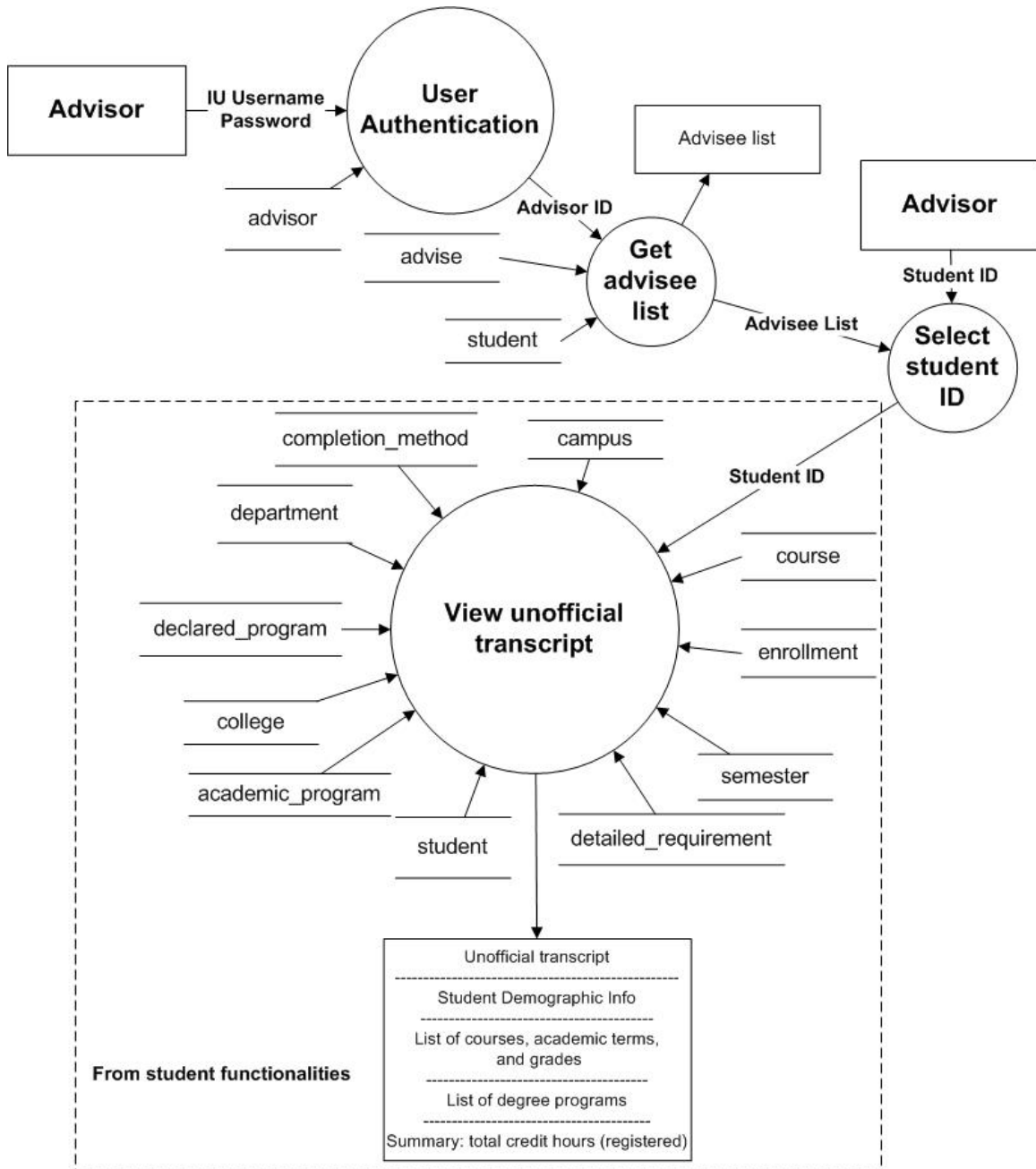
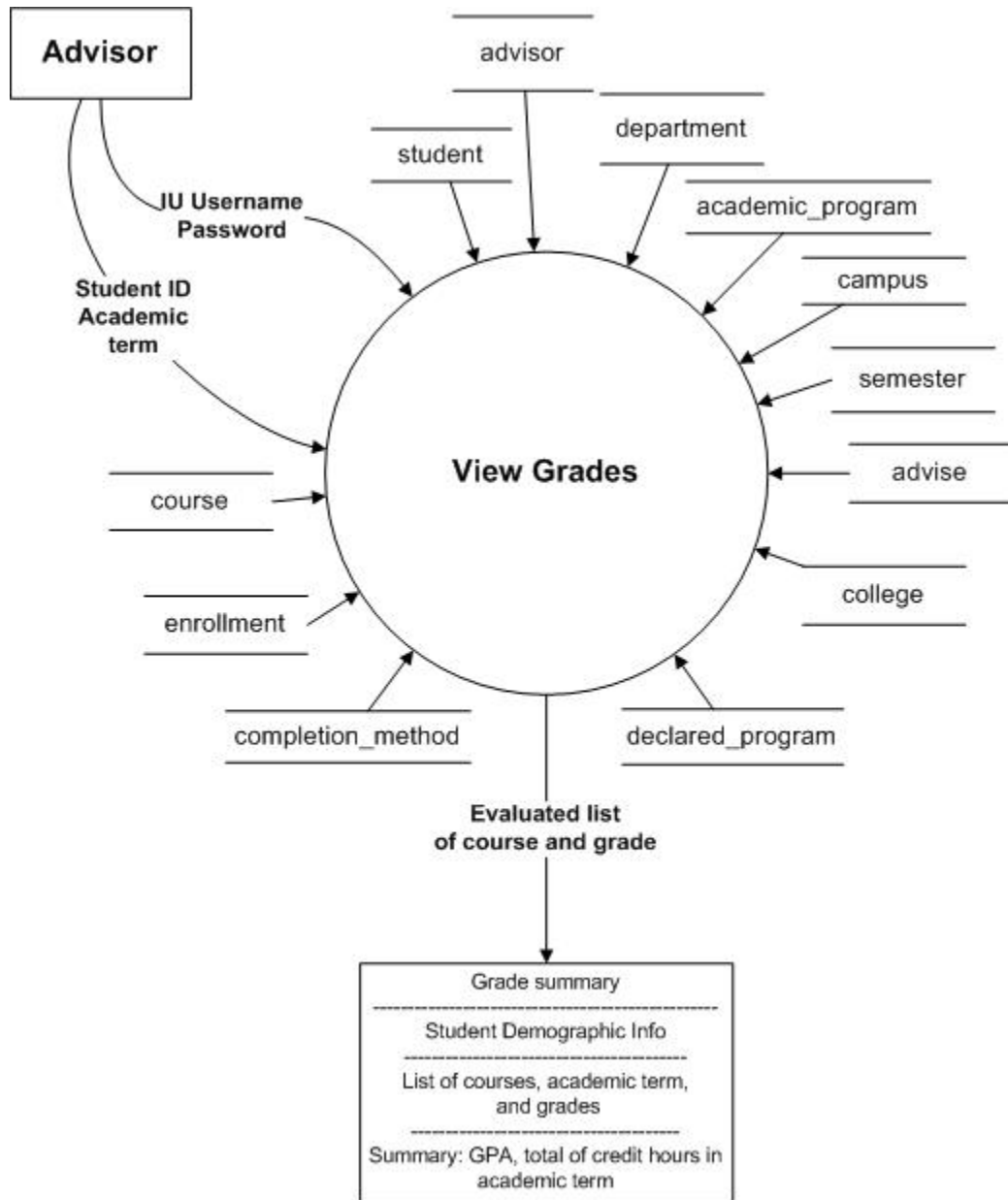


Figure 3.17. Advisor - View Unofficial Transcript Detailed DFD

Figure 3.18 and 3.19 display the view grade report generating steps for an advisor's request.



**Figure 3.18.** Advisor - View Grades Context DFD



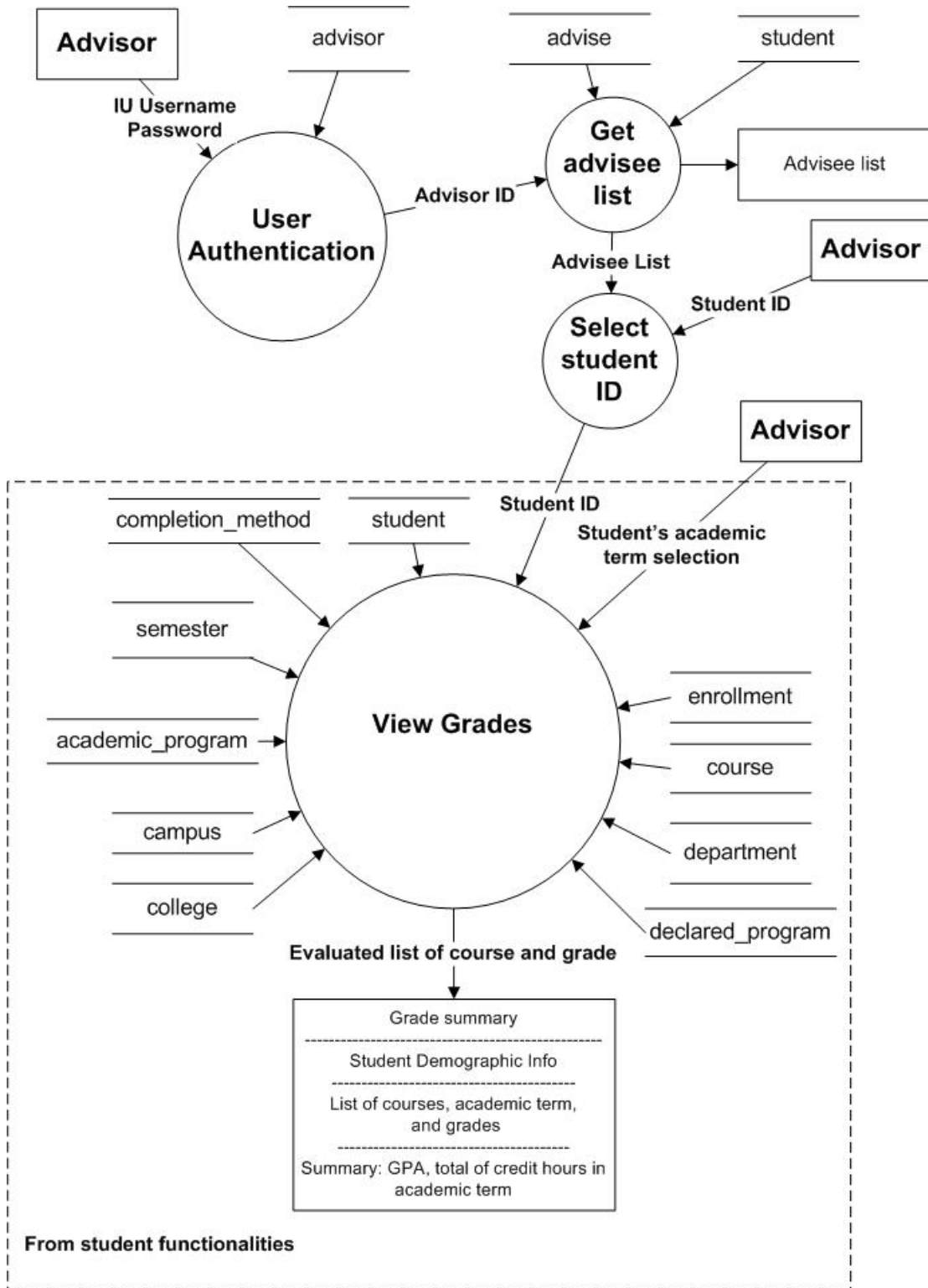
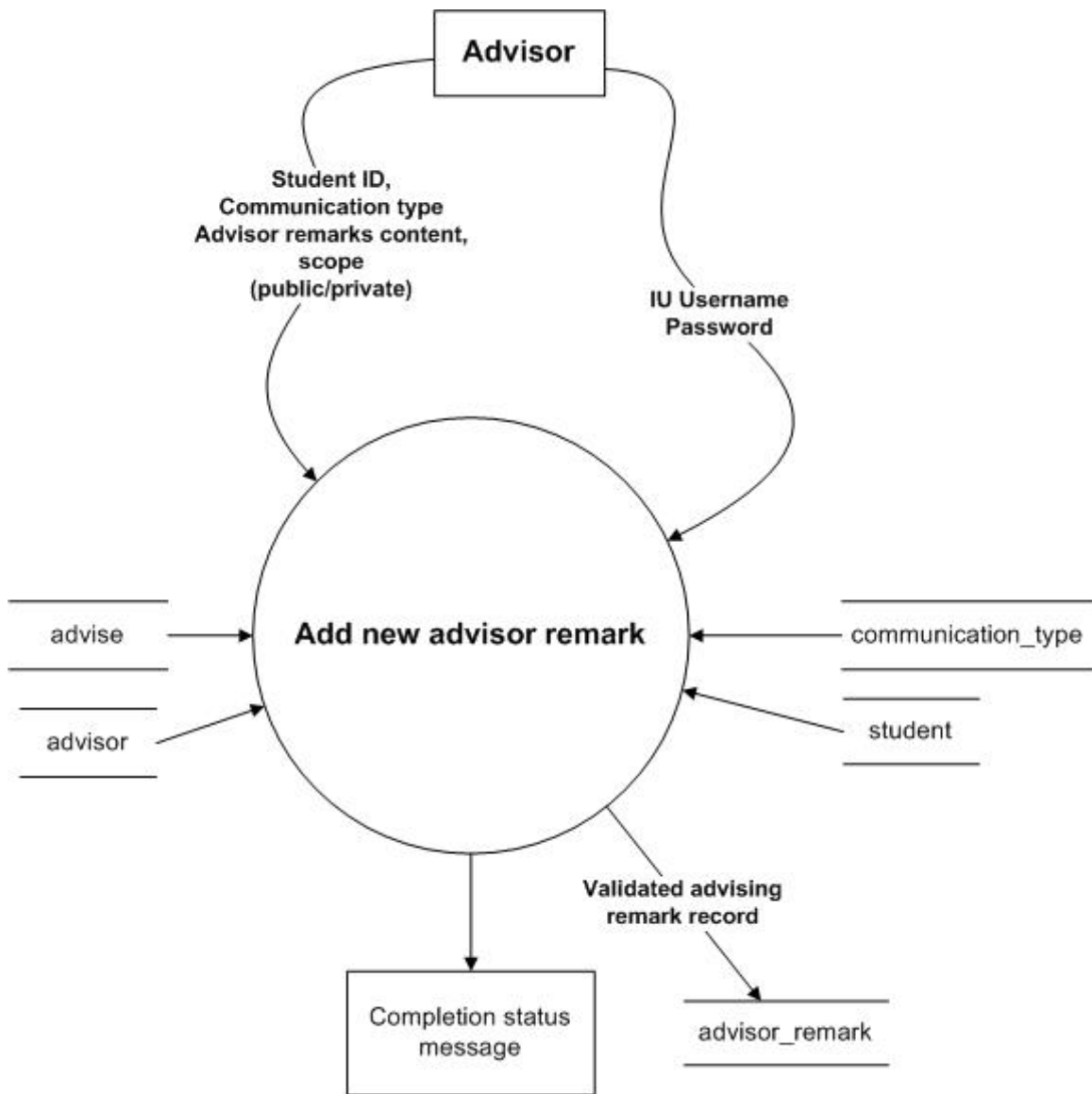


Figure 3.19. Advisor - View Grades Detailed DFD

An advisor can create advisor remarks in all three levels (Figure 3.20 and 3.21).



**Figure 3.20.** *Advisor - Create Advisor Remark Context DFD*

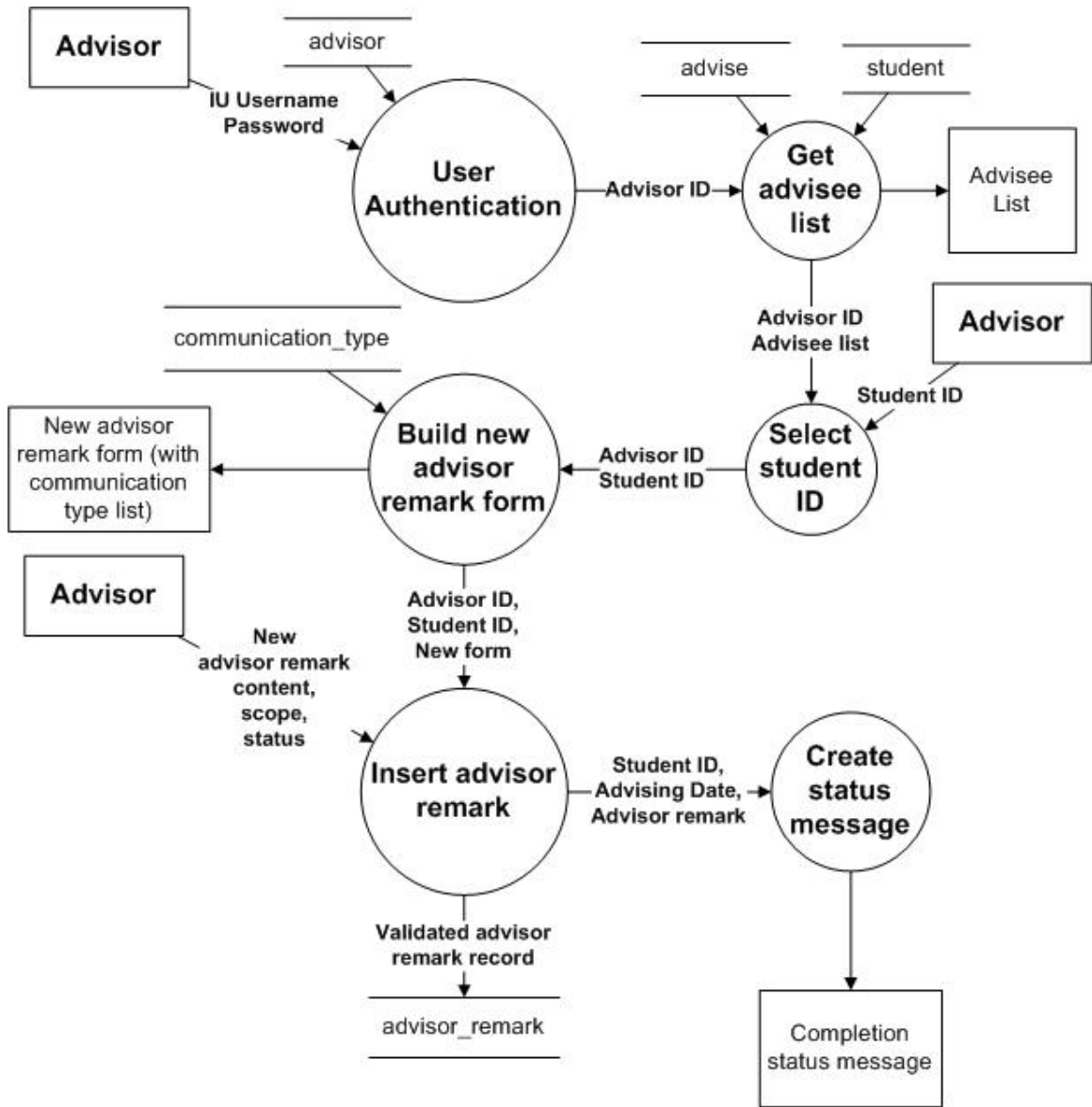
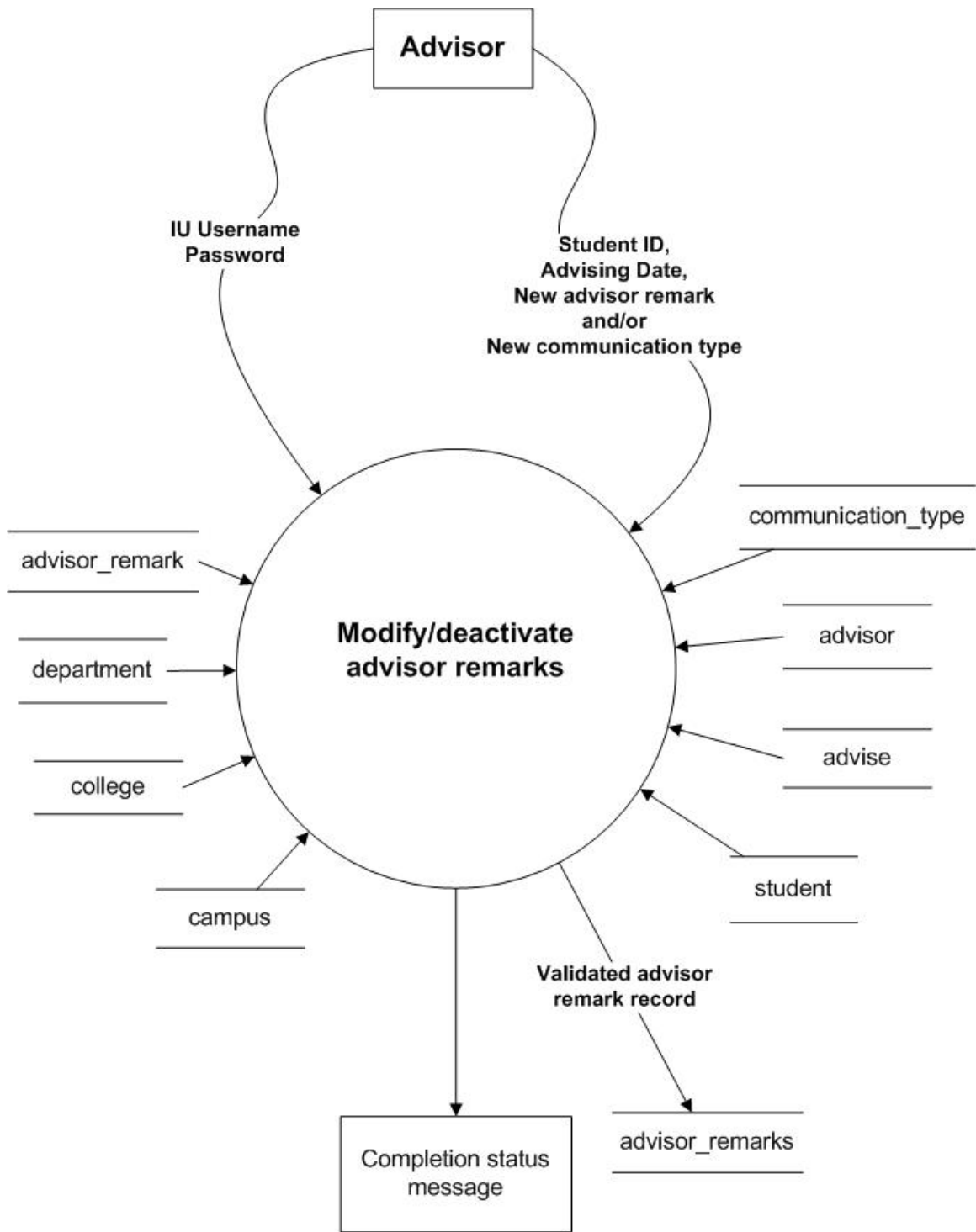


Figure 3.21. Advisor - Create Advisor Remark Detailed DFD

An advisor can only edit the content, change the access level or communication type of a remark, if he or she is its owner (Figure [3.22](#) and [3.23](#)). The advisor can only make such modifications to non-public remarks.



**Figure 3.22.** *Advisor - Modify Advisor Remark Context DFD*

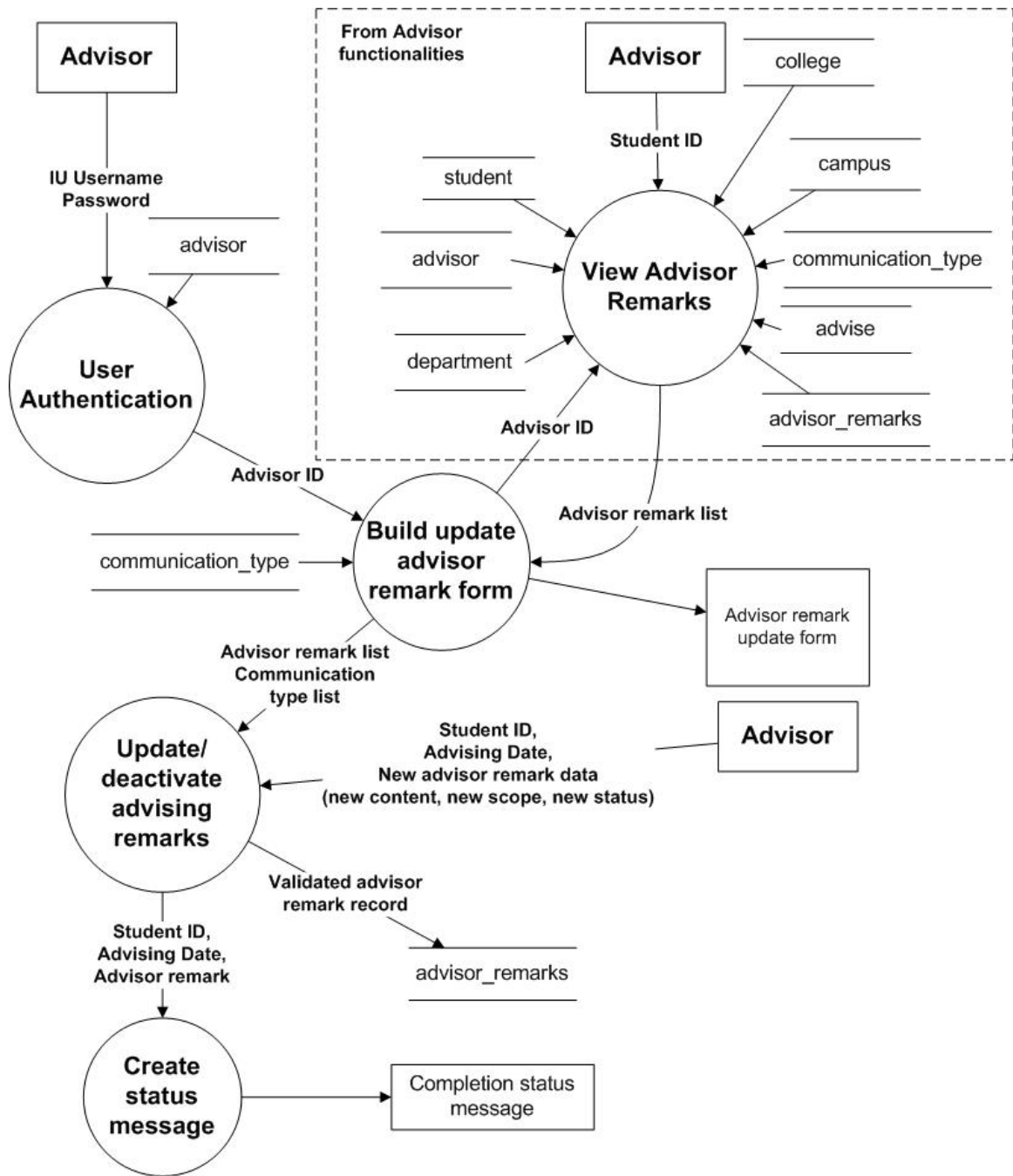
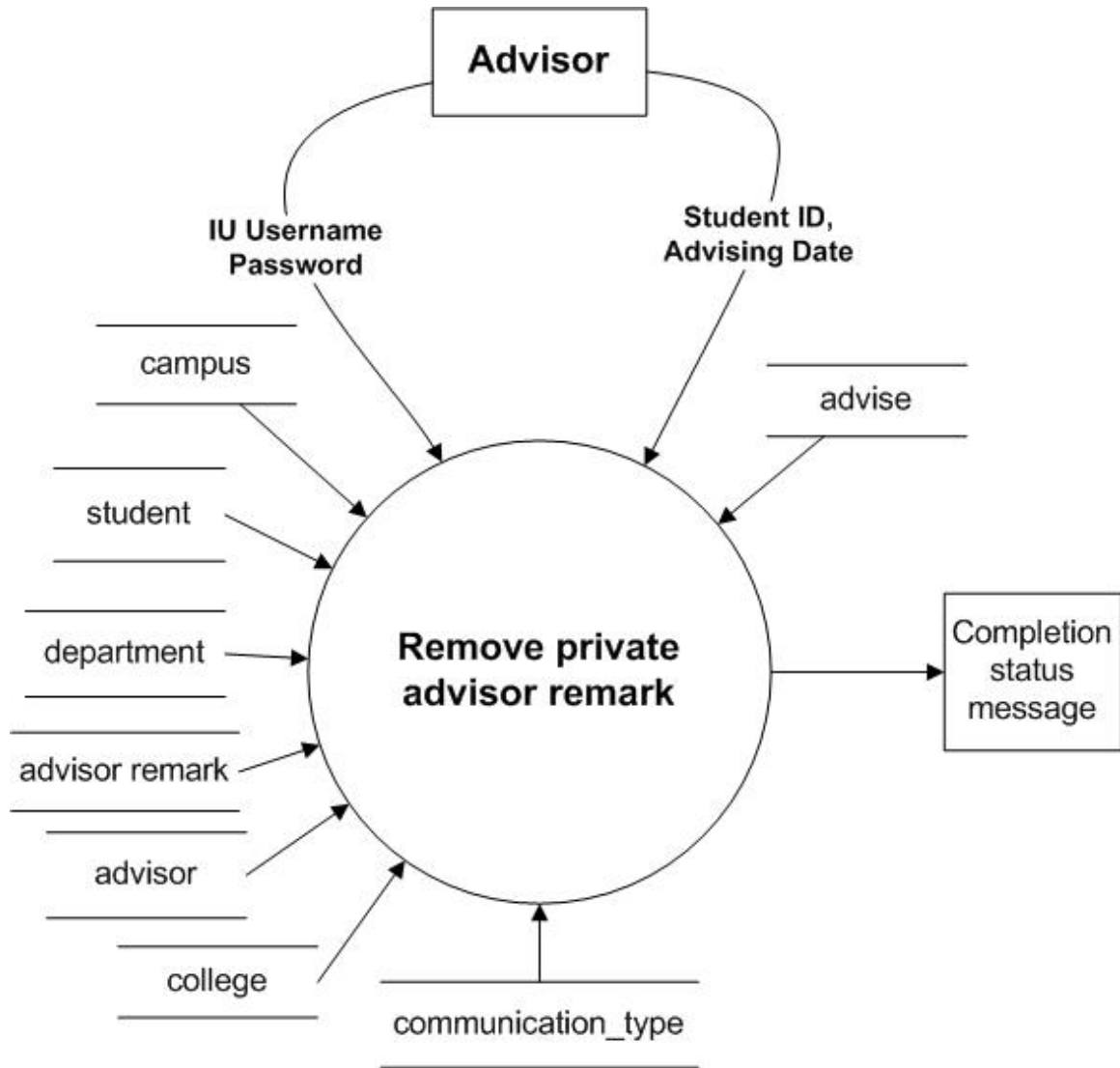


Figure 3.23. Advisor - Modify Advisor Remark Detailed DFD

The system only allows an advisor to remove his or her own advisor remarks in advisor level (Figure 3.24 and 3.25).



**Figure 3.24.** Advisor - Remove Advisor Remark Context DFD

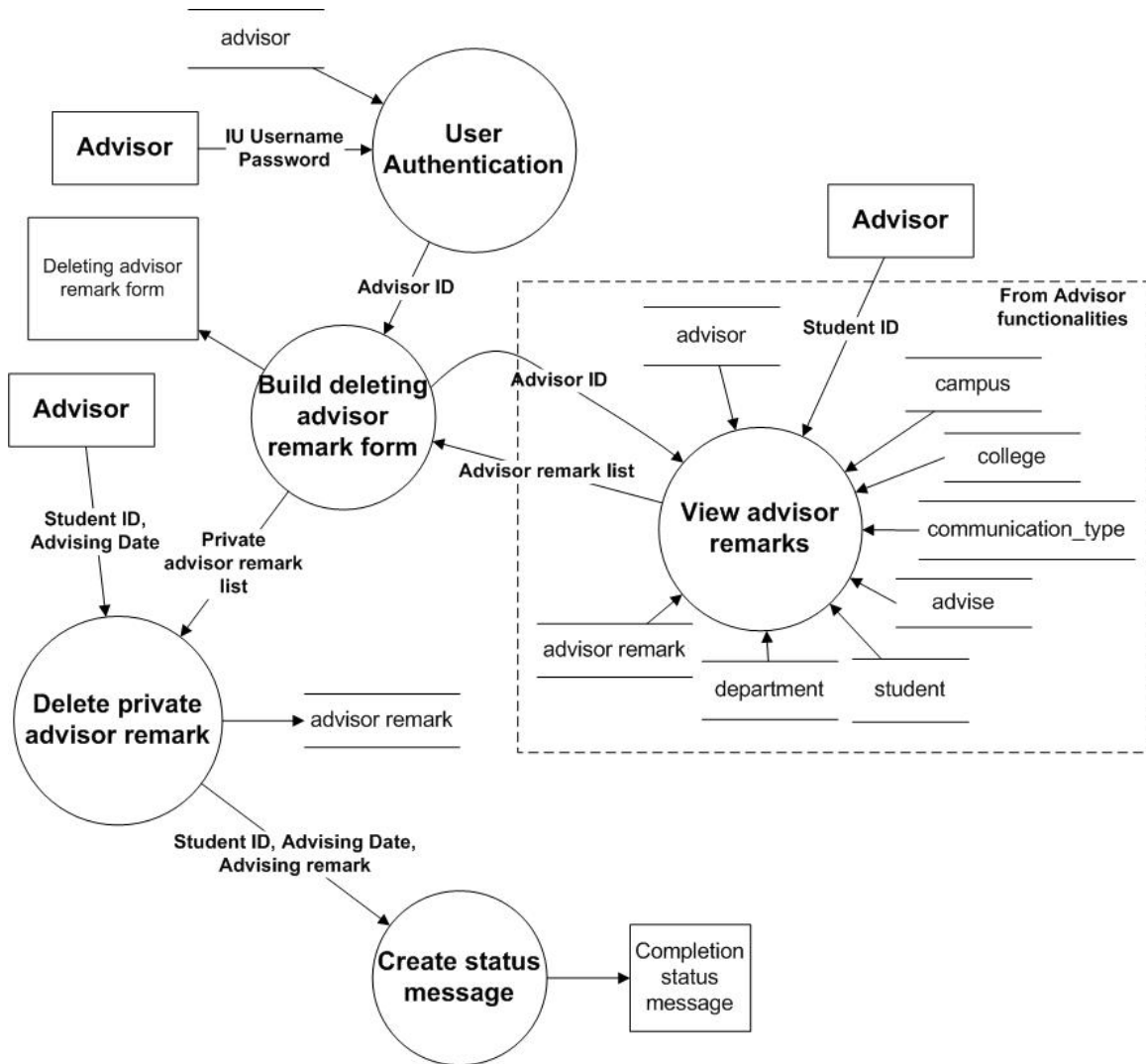


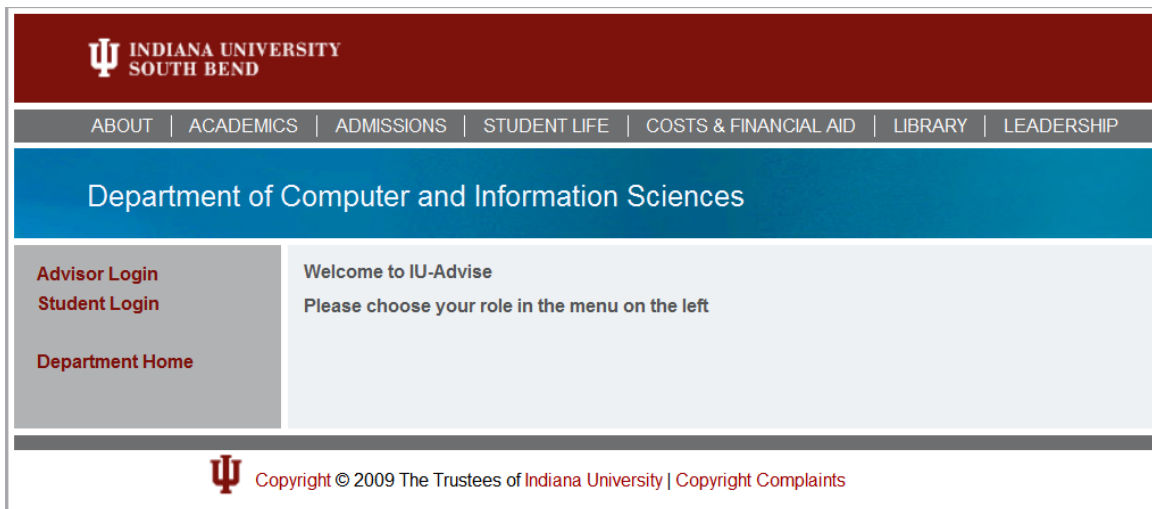
Figure 3.25. Advisor - Remove Advisor Remark Detailed DFD



## 3.5. User Interface

### 3.5.1. Home Page

Upon visiting the advising website, all users will be presented the following web page (Figure 3.26) and must explicitly choose a role (advisor or student) in order to proceed to login page. In the following sections, we will first describe the advisor functionalities (section 3.5.2) and later the student functionalities (section 3.5.3).



**Figure 3.26.** *Chosen role screen for IU-Advise*

### 3.5.2. Advisor Functionality

By clicking on ‘Advisor Login’ link, the advisor is presented with the login page (Figure 3.27). Upon validation, the user will be directed to Advisor page (Figure 3.30). To increase the security of the website, when the provided credentials are not correct, users are allowed only two more tries before having to wait 1 minute for new login attempts (Figure 3.28). This login page also provides input validation such as invalid characters or if any required field is left empty. If the characters are invalid, a message is displayed (Figure 3.29).

The screenshot shows the top navigation bar with the Indiana University South Bend logo and menu items: ABOUT | ACADEMICS | ADMISSIONS | STUDENT LIFE | COSTS & FINANCIAL AID | LIB. Below this is a blue header for the Department of Computer and Information Sciences. On the left is a sidebar with links: Advisor Login, Student Login, and Department Home. The main content area is titled 'ADVISOR LOGIN' and contains the instruction 'Please type your user name and password'. It features two input fields: 'Username' and 'Password', followed by a 'Login' button.

**Figure 3.27.** *Login form for an advisor*

This screenshot shows the same website layout as Figure 3.27, but the main content area displays a message: 'ADVISOR LOGIN Please wait for 1 minute to login again'. The sidebar and navigation bar remain the same.

**Figure 3.28.** *Server validation message*

This screenshot shows the login form with validation messages. The 'ADVISOR LOGIN' section contains the instruction 'Please type your user name and password'. The 'Username' input field has a red error message: 'This field is required.' The 'Password' input field also has a red error message: 'This field is required.' The 'Login' button is still present.

**Figure 3.29.** *Client validation message*

### 3.5.2.1. Advisor Page

After the user is authenticated, the system displays the advisor screen (Figure 3.30). The system displays the left navigation menu specifically designed for an advisor role user. On the right panel, there are two tabs which allow an advisor to view advisee list with students' pictures (Figure 3.30) and review his or her own personal information (Figure 3.31).

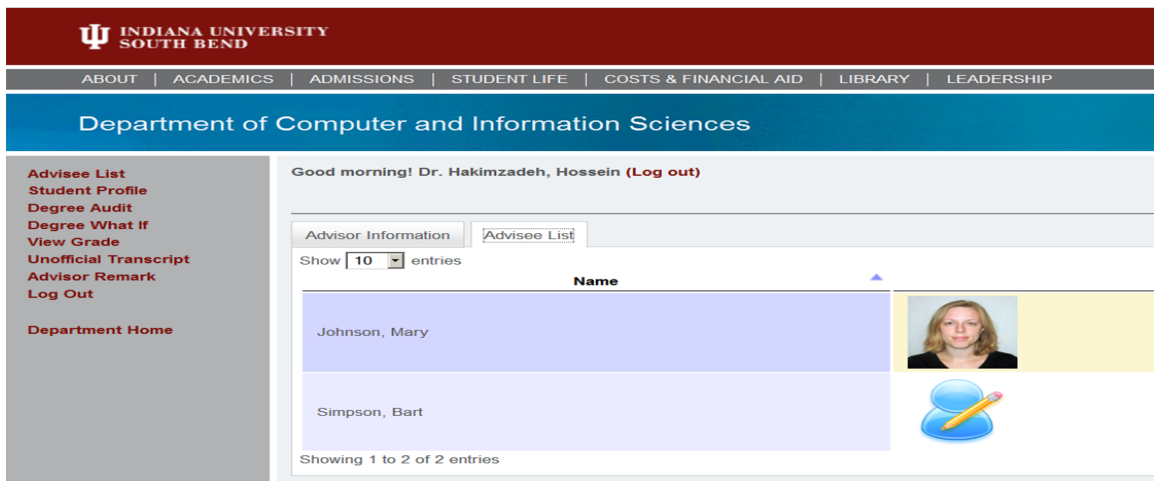


Figure 3.30. Advisor index page

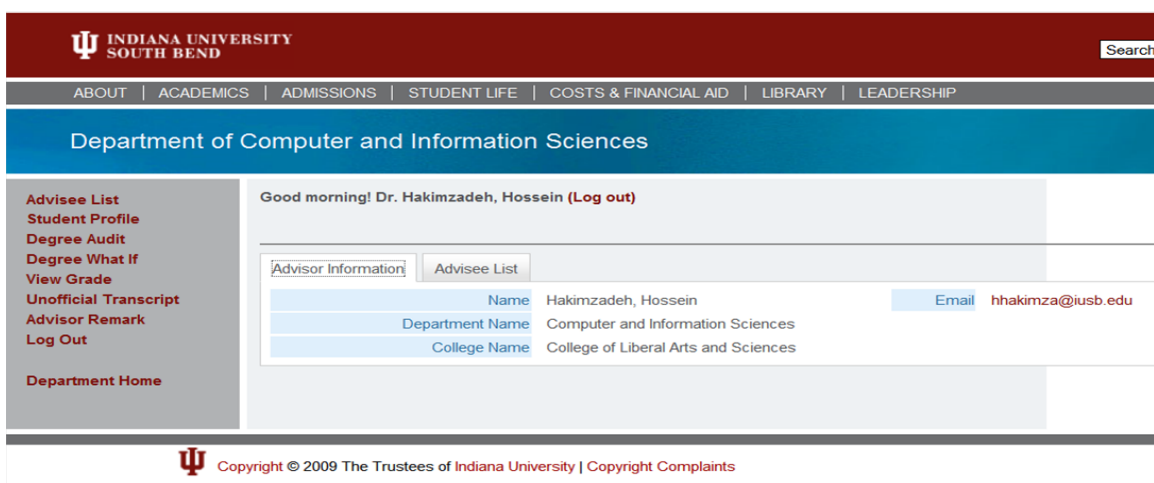


Figure 3.31. Advisor information tab

### 3.5.2.2. Student Profile Page

Once a particular student is selected, the student profile page displays student related information in three tabs. The first tab displays a student's demographic information (Figure 3.32). The second tab gives the latest admission information (Figure 3.33). The third tab lists the test out or placement exams' result (Figure 3.34). To activate a tab, an advisor can click on the title to show the corresponding information. The advisor can also choose a different advisee from the combo box on the top of the tab panels. The data for the new student will be loaded automatically. The advisee combo box automatically becomes a list box if an advisor has more than 8 advisees.

The screenshot shows the student profile page for Mary Johnson. The page is part of the Indiana University South Bend website, specifically the Department of Computer and Information Sciences. The user is logged in as Dr. Hakimzadeh, Hossein. The page displays a dropdown menu for selecting an advisee, currently set to 'Johnson, Mary'. Below this, there are three tabs: 'Student Demographic', 'Latest Admission Information', and 'Test Out Information'. The 'Student Demographic' tab is active, showing a student photo and a table of demographic information.

Name	Johnson, Mary	Birth Year	1970
Student ID	1000	AdmissionStatus	AFQL
Email	student@retain.iusb.edu	Alternate Email	
Home Phone	574-555-1000	Mobile Phone	574-386-0546
Address	1000 Main St. , South Bend IN 46615		

**Figure 3.32.** Student demographic information tab

The screenshot shows the 'Latest Admission Information' tab selected. The page header includes the Indiana University South Bend logo and a search bar. A navigation menu lists: ABOUT | ACADEMICS | ADMISSIONS | STUDENT LIFE | COSTS & FINANCIAL AID | LIBRARY | LEADERSHIP. The main header is 'Department of Computer and Information Sciences'. A sidebar on the left contains links: Advisee List, Student Profile, Degree Audit, Degree What If, View Grade, Unofficial Transcript, Advisor Remark, Log Out, and Department Home. The main content area displays a greeting: 'Good morning! Dr. Hakimzadeh, Hossein (Log out)'. Below this is a dropdown menu for 'Please choose an advisee to view profile' with 'Johnson, Mary' selected. There are three tabs: 'Student Demographic', 'Latest Admission Information' (active), and 'Test Out Information'. A table below shows the following data:

Academic Year	Degree Program	Department	College	Campus
Spring 2007	Computer Science BS	Computer and Information Sciences	College of Liberal Arts and Sciences	Indiana University South Bend

The footer contains the IU logo, copyright information: 'Copyright © 2009 The Trustees of Indiana University | Copyright Complaints', and a 'Text' link.

Figure 3.33. Latest admission information tab

The screenshot shows the 'Test Out Information' tab selected. The page header and navigation menu are identical to Figure 3.33. The sidebar is also the same. The main content area displays the same greeting: 'Good morning! Dr. Hakimzadeh, Hossein (Log out)'. Below this is a dropdown menu for 'Please choose an advisee to view degree audit' with 'Johnson, Mary' selected. There is a section for 'Choose a degree program' with a dropdown menu set to 'Computer Science BS' and a 'View degree audit' button. Below this are two tabs: 'Degree Audit Summary' and 'Degree Audit' (active). A table displays the following data:

Cummulative GPA Hours	35
Total Required Credit Hours	113
Cummulative GPA	3.5774193548387

Figure 3.34. Test out information tab

### 3.5.2.3. *Advisor Remark Page*

After a typical advising session, an advisor may record some remarks about the discussion. This can be done by clicking on the ‘Advisor Remark’ link. The system then presents the default page for the advisor, from which the advisee can select an advisee and click view advisor remarks. Once the advisee is selected, in the first tab, an advisor can view advisor remarks in public and advisor group level made by other advisors about the current student (Figure 3.35). The second tab (Figure 3.36) allows advisors to:

- Create an advisor remark;
- Delete, update and edit an advisor remark in advisor level;
- Deactivate or activate an advisor remark in public and advisor group level;

RSITY Search  GO

ADMISSIONS | STUDENT LIFE | COSTS & FINANCIAL AID | LIBRARY | LEADERSHIP

Computer and Information Sciences

Good morning! Dr. Hakimzadeh, Hossein (Log out)

Johnson, Mary View advisor remarks

Public Advisor Remarks Personal Advisor Remarks

Show 10 entries Search:

Advising Date	Advisor Remark Content	Advisor Name	Department
May 11, 2009 10:00:37 PM	This is a public advisor remark	Knight, William	Computer and Information Sciences

Showing 1 to 1 of 1 entries

**Figure 3.35.** *Advisor remark list*

To add a new advisor remark, an advisor click on the ‘Add’ button to display the New Advisor Remark form (Figure 3.37). The advisor then chooses student name, communication type (Email, Walk-in, or Telephone), access level of the advisor remark (public, advisor group or advisor) and provides the content of the remark. If the advisor leaves the content empty, a message box will appear to alert. In case the advisor choose to create new

Johnson, Mary View advisor remarks

Public Advisor Remarks Personal Advisor Remarks

Show 10 entries Search:

<input type="checkbox"/>	Advising Date	Advisor Remark Content	Communication Type	Access Level	Visible
<input type="checkbox"/>	Apr 27, 2009 7:00:09 PM	This is a public advisor remark	Email	Public	Yes
<input type="checkbox"/>	May 7, 2009 4:00:26 AM	This is a public advisor remark	Phone call	Public	Yes
<input type="checkbox"/>	May 11, 2009 7:00:01 PM	This is an advisor level notes	Email	Advisor Group	No
<input type="checkbox"/>	May 11, 2009 10:00:45 PM	Advise student to take this course	Email	Public	No
<input type="checkbox"/>	May 11, 2009 10:00:48 PM	Advise student to do research on database topics	Email	Advisor	

Showing 1 to 5 of 5 entries

**Figure 3.36.** Personal advisor remark list

public or advisor group remark, he or she can check the ‘Make visible’ check box to make the remark visible to other users. After providing information, the advisor clicks the ‘Save’ button to submit the new advisor remark.

Computer and Information Sciences

Good morning! Dr. Hakimzadeh, Hossein (Log out)

New Advisor Remark

Current date time: May 11, 2009 10:45:53 P  
 Student: Johnson, Mary  
 Communication Type: Email  
 Access Level: PUBLIC-Both student and advisor roles can access the remark  
 Make Visible:   
 Advisor remark content: Advise student to take this course

**Figure 3.37.** Add new advisor remark form

To edit an advisor remark, an advisor first has to select the “remark” which to be modified by checking the check box in front of that remark (Figure 3.38). Then, the advisor

clicks the ‘Edit’ button to load the chosen record. If the record is not valid for editing, the advisor is directed back to the ‘Advisor Remark’ page. If it is valid, data is loaded into a form (Figure 3.39). At this point, the advisor can edit the contents of the remark, choose different communication type, or change the access level of the remark. If the access level is changed from advisor level to public or advisor group level, the system will alert the advisor to confirm the change of access level. This is critical since public or advisor group level advisor remarks cannot be edited or deleted, they can only be activated or deactivated. After clicking the ‘OK’ button, the advisor is shown modifications and asked to confirm the operation.

Good morning! Dr. Hakimzadeh, Hossein (Log out)

Johnson, Mary

Public Advisor Remarks Personal Advisor Remarks

Show  entries Search:

<input type="checkbox"/>	Advising Date	Advisor Remark Content	Communication Type	Access Level	Visible
<input type="checkbox"/>	Apr 27, 2009 7:00:09 PM	This is a public advisor remark	Email	Public	Yes
<input type="checkbox"/>	May 7, 2009 4:00:26 AM	This is a public advisor remark	Phone call	Public	Yes
<input type="checkbox"/>	May 11, 2009 7:00:01 PM	This is an advisor level notes	Email	Advisor Group	No
<input type="checkbox"/>	May 11, 2009 10:00:45 PM	Advise student to take this course	Email	Public	No
<input checked="" type="checkbox"/>	May 11, 2009 10:00:48 PM	Advise student to do research on database topics	Email	Advisor	

Showing 1 to 5 of 5 entries

**Figure 3.38.** Choose a user advisor remark

To delete an advisor remark, the advisor first select advisor remarks need to be deleted (Figure 3.40). If there are invalid records for deletion (e.g improper ownership), the system will automatically filter out and only show valid records for confirmation (Figure 3.41). There will be a message to inform the advisor if there is no record to be deleted. Otherwise, the advisor clicks the ‘OK’ to confirm the deletion.



Good morning! Dr. Hakimzadeh, Hossein ([Log out](#))

---

**Edit Advisor Remark**

Current date time	May 11, 2009 10:48:55 PM
Student	1000
Communication Type	Email
Access Level	USER-Only advisor who created the remarks can access them
Make Active	<input type="checkbox"/>
Advisor remark content	Advise student to do research on database topics

**Figure 3.39.** Load data into form

Good morning! Dr. Hakimzadeh, Hossein ([Log out](#))

---

Johnson, Mary

---

Public Advisor Remarks | Personal Advisor Remarks

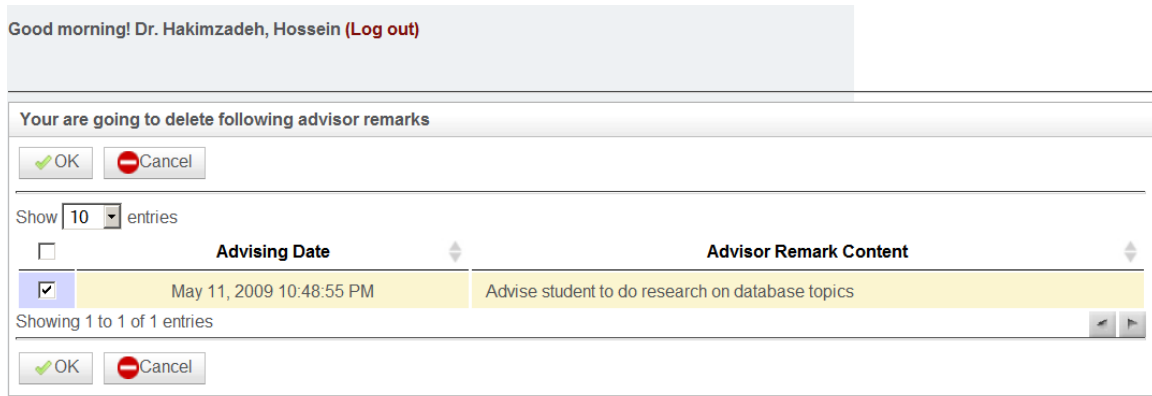
Show 10 entries Search:

<input type="checkbox"/>	Advising Date	Advisor Remark Content	Communication Type	Access Level	Visible
<input checked="" type="checkbox"/>	Apr 27, 2009 7:00:09 PM	This is a public advisor remark	Email	Public	Yes
<input checked="" type="checkbox"/>	May 7, 2009 4:00:26 AM	This is a public advisor remark	Phone call	Public	Yes
<input checked="" type="checkbox"/>	May 11, 2009 7:00:01 PM	This is an advisor level notes	Email	Advisor Group	No
<input checked="" type="checkbox"/>	May 11, 2009 10:00:45 PM	Advise student to take this course	Email	Public	No
<input checked="" type="checkbox"/>	May 11, 2009 10:00:48 PM	Advise student to do research on database topics	Email	Advisor	

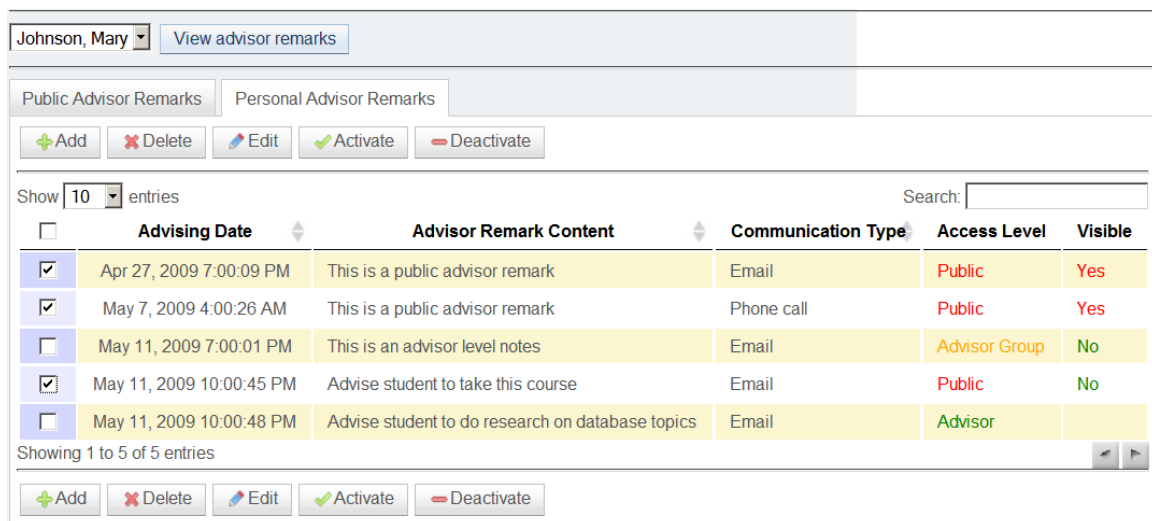
Showing 1 to 5 of 5 entries

**Figure 3.40.** Choose multiple user advisor remark

With public and advisor group level remarks, advisors can only either activate or deactivate them. To activate or deactivate advisor remarks, the advisor chooses the remarks (Figure 3.42) then either click 'Activate' or 'Deactivate' button.



**Figure 3.41.** Only list valid records



**Figure 3.42.** Chosen records for activating or deactivating

### 3.5.2.4. Degree Audit

To prepare for each advising session and help a student plan his or her upcoming schedule, an advisor often needs to view the degree audit for a given student. The degree audit provides a convenient method for identifying which degree requirements are which requirements are still unmet, helping focus the discussion between the student and his or her advisor. This report can be accessed by clicking on the ‘Degree Audit’ link. The advisor then chooses an advisee from the advisee list and click ‘View degree audit’ button. The

degree audit for the chosen student is displayed as in Figure 3.43. A degree audit report contains two tabs. The first tab is the summary information. The second tab is the detail information which lists the degree requirements, detailed requirements and courses that satisfy the detailed requirements (Figure 3.44). A green line in detail part of the degree audit indicates a student has successfully completed the given detailed requirement. On the contrary, a red line indicates that a student has attempted to complete the detailed requirement, but has not yet satisfied it. In addition to pass and fail indicator, if no attempt is made to satisfy a given detailed requirement, the degree audit feature presents a list of courses that can potentially satisfy that detailed requirement.

INDIANA UNIVERSITY SOUTH BEND

ABOUT | ACADEMICS | ADMISSIONS | STUDENT LIFE | COSTS & FINANCIAL AID | LIBRARY | LEADERSHIP

Department of Computer and Information Sciences

Advisee List  
Student Profile  
Degree Audit  
Degree What If  
View Grade  
Unofficial Transcript  
Advisor Remark  
Log Out

Department Home

Good morning! Dr. Hakimzadeh, Hossein (Log out)

Please choose an advisee to view degree audit  
Johnson, Mary

Choose a degree program  
Computer Science BS

View degree audit

Degree Audit Summary | Degree Audit

Cummulative GPA Hours	35
Total Required Credit Hours	113
Cummulative GPA	3.5774193548387

Figure 3.43. Degree audit result

Fundamental Literacies (13-18 Credits)									
Course No	Title	Credit	Grade	Semester	Year	Notes	Hrs Passed	GPA Hrs	GPA Units
ENG-W 131	English Composition	3	A	Spring	2007		3	3	12
PHIL-P 150	Critical Thinking	3							
SPCH-S 121	Oral Communication	3	A	Spring	2007		3	3	12
JOUR-J 210	Visual Literacy	3							
ADMIN-A 999	Quantitative Reasoning	0							
COAS-Q 110	Information Literacy	1	A	Spring	2007		1	1	4
ADMIN-A 999	Computer Literacy	0							
						Credits Passed	7		
							GPA Hrs Total	7	
							GPA		4

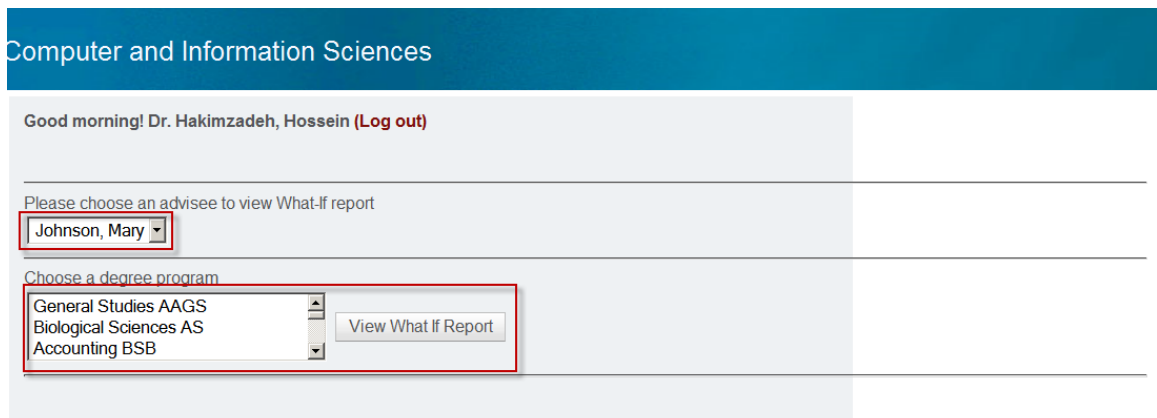
  

Common Core (12 Credits)									
Course No	Title	Credit	Grade	Semester	Year	Notes	Hrs Passed	GPA Hrs	GPA Units
AST-N 190	The Natural World	3							
SOC-B 190	Human Behavior & Social Institutions	3							

Figure 3.44. Degree audit detail tab

### 3.5.2.5. Degree What-if Report

Many students would like to know how their existing courses apply to the degree requirements for another major. The “What-If” report is designed to answer such questions. To generate a what-if report, an advisor clicks on ‘Degree What If’ link. Then, the advisor selects the advisee (from the first combo box) and the new degree program (from the second combo box) before clicking on the ‘View What If Report’ button (Figure 3.45). Figure 3.46 shows the details of a What-If report. The What-If report also supports matching the equivalent courses to satisfy for a detail requirement. Along with the detail report, the system also produces a list of “elective” courses which do not fit in any particular degree requirements (Figure 3.47).



Computer and Information Sciences

Good morning! Dr. Hakimzadeh, Hossein (Log out)

Please choose an advisee to view What-If report

Johnson, Mary

Choose a degree program

General Studies AAGS  
Biological Sciences AS  
Accounting BSB

View What If Report

**Figure 3.45.** Input for producing degree what-if report

Course No	Title	Credit	Grade	Semester	Year	Notes	Hrs Passed	GPA Hrs	GPA Units
MATH-M 118	Finite Mathematics	3							
ADMIN-A 999	Statistics	0							
Credits Passed									
GPA Hrs Total									
GPA									
Informatics Core (34 Credits)									
Course No	Title	Credit	Grade	Semester	Year	Notes	Hrs Passed	GPA Hrs	GPA Units
INFO-I 101	Introduction to Informatics	4							
INFO-I 201	Math Foundations of Informatics	4							
INFO-I 202	Social Informatics	3							
INFO-I 210	Information Infrastructure I	4	T	Spring	2005		4	4	0
INFO-I 211	Information Infrastructure II	4	B-	Spring	2005		4	4	10.8
INFO-I 308	Information Representation	3							
INFO-I 310	UpperCore Course 1	3							
INFO-I 310	UpperCore Course 2	3							
INFO-I 450	Capstone Course 1	3							
INFO-I 420	Capstone Course 2	3							
Credits Passed							8		

Figure 3.46. Degree what-if report details

Please choose an advisee to view What-If report  
 Johnson, Mary

Choose a degree program  
 Criminal Justice BS  
 Computer Programming CRT  
 Computer Science AS  
 Computer Science BS

View What If Report

What If Result: Excluded Courses

Show 10 entries

Course ID	Course No.	Title	Credit	Grade	Year	Semester	Explanations
011231	CSCI-C 151	MULTIUSER OPERATING SYSTEMS	2	A	2005	Spring	No Comment
011246	CSCI-C 243	INTRO TO DATA STRUCTURES	4	A	2005	Summer	No Comment
011248	CSCI-C 251	FOUNDATIONS OF DIGITAL COMPUTG	3	A	2005	Summer	No Comment
011260	CSCI-C 308	SYSTEM ANALYSIS AND DESIGN	4	B	2006	Spring	No Comment
011262	CSCI-C 311	PROGRAMMING LANGUAGES	3	B-	2007	Spring	No Comment
011266	CSCI-C 335	COMPUTER STRUCTURES	4	A	2007	Spring	No Comment

Showing 1 to 6 of 6 entries

Figure 3.47. Excluded courses report

### 3.5.2.6. View Grade

The ‘View Grade’ link allows advisors to view the semester grade of a given advisee. To do so, an advisor clicks the ‘View Grade’ link, select an advisee from the list, then select the academic term (semester and year), and the grade report will be produced (Figure 3.48). The result report is displayed below the academic term combo box.

Good morning! Dr. Hakimzadeh, Hossein (Log out)

Please choose an advisee to view grade

Johnson, Mary

Spring 2007  
Spring 2006  
Summer 2005

View grades

Grades

Search:

Course	Course Title	Credit Hours	Grade
CSCI-C 101	COMPUTER PROGRAMMING I	4	T
CSCI-C 151	MULTIUSER OPERATING SYSTEMS	2	A
CSCI-C 201	COMPUTER PROGRAMMING II	4	B-
Semester Credit Hours: 6			
Semester GPA: 3.13333333333333			

**Figure 3.48.** Advisor view grades

### 3.5.2.7. View Unofficial Transcript

To display a chronological view of the student’s grades an advisor can consult the unofficial transcript. To show the report, an advisor clicks on the ‘Unofficial Transcript’ link, selects an advisee and clicks ‘View unofficial transcript’ button to display the report as in Figure 3.49.

## 3.5.3. Student Functionality

When a user clicks on ‘Student Login’ link, a login form similar to the one shown to advisors will appear. The authentication process is similar, however, this form uses the

Unofficial Transcript			
Spring 2007			
Course	Course Title	Credit Hours	Grade
COAS-Q 110	INTRO TO INFORMATION LITERACY	1	A
CSCI-C 311	PROGRAMMING LANGUAGES	4	B-
CSCI-C 335	COMPUTER STRUCTURES	4	A
ENG-W 131	ELEMENTARY COMPOSITION 1	3	A
MUS-A 190	ARTS	0	A
SPCH-S 121	PUBLIC SPEAKING	3	A
Semester Credit Hours: 15			
Semester GPA: 3.65333333333333			
Spring 2006			
Course	Course Title	Credit Hours	Grade
CSCI-C 308	SYSTEM ANALYSIS AND DESIGN	4	B
Semester Credit Hours: 4			
Semester GPA: 3			

**Figure 3.49.** *Unofficial transcript detail*

*student* data table for authenticating the user. As an authenticated and authorized student can access the welcome screen with the left navigation menu specifically designed for students (Figure 3.50). The right panel of the screen displays the demographic information, latest admission information and test out or placement results in three tabs.

**Profile**

[Degree Audit](#)

[Degree What If](#)

[View Grades](#)

[Unofficial Transcript](#)

[Advisor Remark](#)

[Log Out](#)

[Department Home](#)


Good morning! Johnson, Mary ([Log out](#))

---

Student Demographic

Latest Admission Information

Test Out Information



Name	Johnson, Mary	Birth Year	1970
Student ID	1000	Alternate Email	
Email	student@retain.iusb.edu	Mobile Phone	574-386-0546
Home Phone	574-555-1000		
Address	1000 Main St., South Bend IN 46615		

**Figure 3.50.** *Student index page*

In addition to viewing profile, the student can also access other reports by clicking on the corresponding links on the left navigation menu of the welcome screen. The produced reports have similar style and layout with those for advisors. By Clicking on 'Advisor

Remark' link, a student can view all public level advisor remarks in a table (Figure 3.51). This table can be sorted by clicking on the column headers.

Good morning! Johnson, Mary ([Log out](#))

Advisor Remark List

Show  entries Search:

Advising Date	Advisor Remark Content	Advisor Name	Department
Apr 27, 2009 7:00:09 PM	This is a public advisor remark	Hakimzadeh, Hossein	Computer and Information Sciences
May 7, 2009 4:00:26 AM	This is a public advisor remark	Hakimzadeh, Hossein	Computer and Information Sciences
May 11, 2009 10:00:37 PM	This is a public advisor remark	Knight, William	Computer and Information Sciences

Showing 1 to 3 of 3 entries ◀ ▶

**Figure 3.51.** Student advisor remark list

Clicking on the 'Degree Audit' link and choosing a degree program, the student displays the degree audit summary as in Figure 3.52 and the details in Figure 3.53.

Department of Computer and Information Sciences

Good morning! Johnson, Mary ([Log out](#))

Choose a degree program

Cumulative GPA Hours	35
Total Required Credit Hours	113
Cummulative GPA	3.5774193548387

Copyright © 2009 The Trustees of Indiana University | Copyright Complaints Text Only

**Figure 3.52.** Student degree audit

The degree what-if report is activated by the 'Degree What If' link. Then, the student chooses a target degree program and clicks 'View what if report' to show the result page (Figure 3.54).



Fundamental Literacies (13-18 Credits)									
Course No	Title	Credit	Grade	Semester	Year	Notes	Hrs Passed	GPA Hrs	GPA Units
ENG-W 131	English Composition	3	A	Spring	2007		3	3	12
PHIL-P 105	Critical Thinking	3							
SPCH-S 121	Oral Communication	3	A	Spring	2007		3	3	12
FINA-A 100	Visual Literacy	3							
ADMIN-A 999	Quantitative Reasoning	0							
COAS-Q 110	Information Literacy	1	A	Spring	2007		1	1	4
ADMIN-A 999	Computer Literacy	0							
Credits Passed							7		
GPA Hrs Total								7	
GPA									4

Common Core (12 Credits)									
Course No	Title	Credit	Grade	Semester	Year	Notes	Hrs Passed	GPA Hrs	GPA Units
CHEM-N 190	The Natural World	3							

Figure 3.53. Student degree audit details

Informatics Core (34 Credits)									
Course No	Title	Credit	Grade	Semester	Year	Notes	Hrs Passed	GPA Hrs	GPA Units
INFO-I 101	Introduction to Informatics	4							
INFO-I 201	Math Foundations of Informatics	4							
INFO-I 202	Social Informatics	3							
INFO-I 210	Information Infrastructure I	4	T	Spring	2005		4	4	0
INFO-I 211	Information Infrastructure II	4	B-	Spring	2005		4	4	10.8
INFO-I 308	Information Representation	3							
INFO-I 300	UpperCore Course 1	3							
INFO-I 300	UpperCore Course 2	3							
INFO-I 400	Capstone Course 1	3							
INFO-I 451	Capstone Course 2	3							
Credits Passed							8		
GPA Hrs Total								4	
GPA									2.7

Informatics Elective (6 Credits)									
Course No	Title	Credit	Grade	Semester	Year	Notes	Hrs Passed	GPA Hrs	GPA Units
ADMIN-A 999	Informatics Elective I	0							

Figure 3.54. Student degree what-if report

Similar to an authenticated advisor, an authenticated student can also view his or her grades from a specific academic term by clicking on the ‘View Grade’ link, choosing the desired semester and year and clicking on the ‘View grades’ button. The resulting page is displayed as in Figure 3.55.

ABOUT | ACADEMICS | ADMISSIONS | STUDENT LIFE | COSTS & FINANCIAL AID | LIBRARY | LEADERSHIP

Department of Computer and Information Sciences

Good morning! Johnson, Mary (Log out)

Spring 2007  
Spring 2006  
Summer 2005

View grades

Grades

Course	Course Title	Credit Hours	Grade
CSCI-C 101	COMPUTER PROGRAMMING I	4	T
CSCI-C 151	MULTIUSER OPERATING SYSTEMS	2	A
CSCI-C 201	COMPUTER PROGRAMMING II	4	B-
Semester Credit Hours: 6			
Semester GPA: 3.13333333333333			

Figure 3.55. Student view grade report

A student can review his or her own academic progress by clicking on the ‘Unofficial Transcript’ link. A report is produced and shown as in Figure 3.56.

Course	Course Title	Credit Hours	Grade
COAS-Q 110	INTRO TO INFORMATION LITERACY	1	A
CSCI-C 311	PROGRAMMING LANGUAGES	4	B-
CSCI-C 335	COMPUTER STRUCTURES	4	A
ENG-W 131	ELEMENTARY COMPOSITION 1	3	A
MUS-A 190	ARTS	0	A
SPCH-S 121	PUBLIC SPEAKING	3	A
Semester Credit Hours: 15			
Semester GPA: 3.65333333333333			

Course	Course Title	Credit Hours	Grade
CSCI-C 308	SYSTEM ANALYSIS AND DESIGN	4	B
Semester Credit Hours: 4			
Semester GPA: 3			

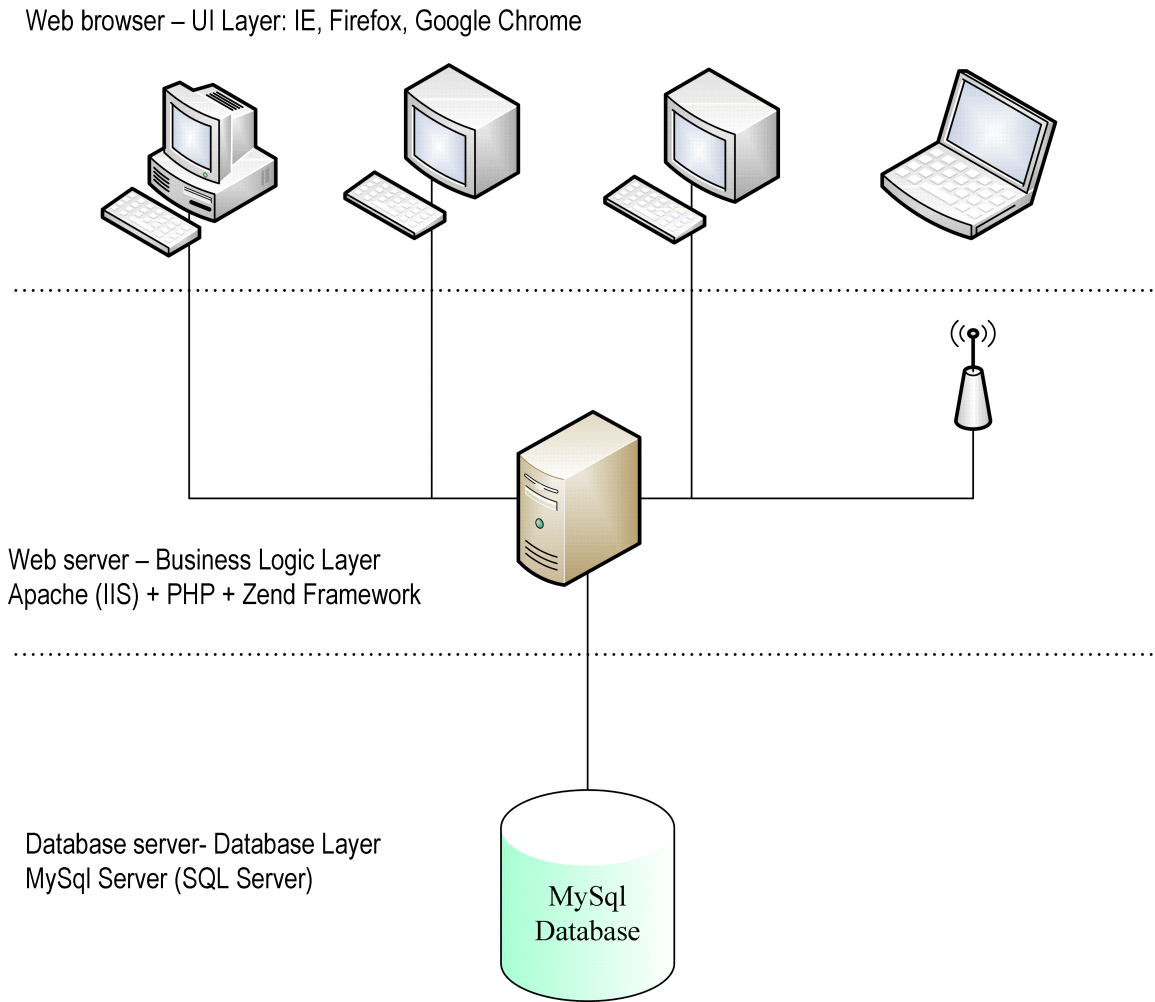
Course	Course Title	Credit Hours	Grade
CSCI-C 243	INTRO TO DATA STRUCTURES	4	A
CSCI-C 251	FOUNDATIONS OF DIGITAL COMPUTG	4	A
Semester Credit Hours: 8			

Figure 3.56. Student view unofficial transcript

## 4. Implementation Technologies

### 4.1. Application Architecture

The project is implemented by using a three tier software model which includes a user interface, a business logic and a database layer. A common web browser such as Firefox or Internet Explorer can be used as user interface. These web browsers support most of the new specifications of HTML and CSS to layout and present input forms as well as output result. In this project, Internet Explorer and Firefox is mainly used for testing purposes. The business layer is responsible for validating user input, as well as processing and interaction with the database and maintaining security and privacy of data. This layer is implemented using a combination of Apache, Zend Framework, and PHP. Apache is a free and open source web server. It is widely used and easy to install on both the Windows and Linux platforms. PHP is a C++ like open source server-side scripting language. A PHP engine can be installed on both Apache and IIS. In addition, we also have utilized the Zend Framework for interfacing the business logic layer with the user interface layer. Moreover, Zend Framework has a built-in templating engine and useful abstract classes suited for developing a web application. Finally, the database layer is implemented using MySQL Server Community version. MySQL Server is a popular open source database engine that supports most features of commercial database engines such as Microsoft SQL Server. The use of PHP and Zend Framework helps to completely separate the three layers of the application and also increases the flexibility and maintainability of the project. In addition, the prototype was developed in Microsoft Windows Vista and ported to Linux in order to demonstrate the cross platform utility of this system. Figure 4.1 shows the architecture of the IU-Advise system.



**Figure 4.1.** Architecture of the proposed system

## 4.2. Ajax

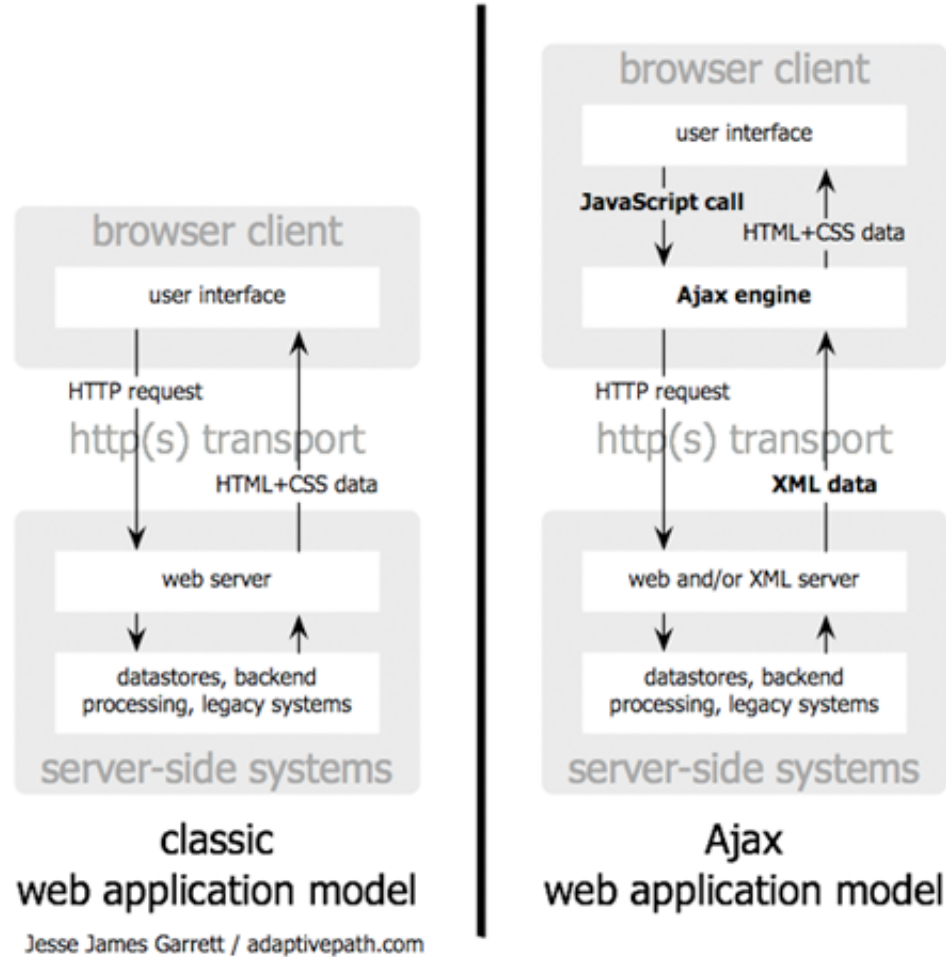
In addition to above tools, Ajax, which is a new web programming model, is also used in IU-Advise to build a friendlier and dynamic user interface. This section describes our research and use of this model within IU-Advise.

Ajax stands for Asynchronous JavaScript and XML [16]. This phrase was defined by Jesses James Garrett in [16] and is not considered to be an acronym. Ajax is not composed by one single technology. In fact, it includes:

- XHTML and/or CSS-for controlling appearance of a HTML page;
- Document Object Model (DOM)-for interacting and manipulating elements in a HTML page;
- XML or JSON-for transmitting data between server and client;
- JavaScript-for developers to work with above technologies;

These technologies are available in most of modern web browsers and can be used to provide better functionality, more dynamic and rich user interface to web applications

Before presenting the ideas behind Ajax, we will examine the traditional web application development model. In traditional model, a web browser submits a HTTP request in form of HTTP packets to a web server. The server processes this request, produces and sends an HTML page along with CSS and JavaScript files back to client's web browser (the left column of Figure 4.2). The web browser loads or reloads all the files from the server to display a new page. This model is called synchronous because each user interaction always corresponds with a server processing (top portion of Figure 4.3) [16]. The traditional model perfectly works in the point of view of developers and matches the nature of hypertext medium. It, however, does not provide a dynamic and friendly user interaction since the users have to wait for the new results while the server is processing and resending the information back to the browser. This model requires large network bandwidth for transmitting data since each transmission usually includes unnecessary duplicated data that

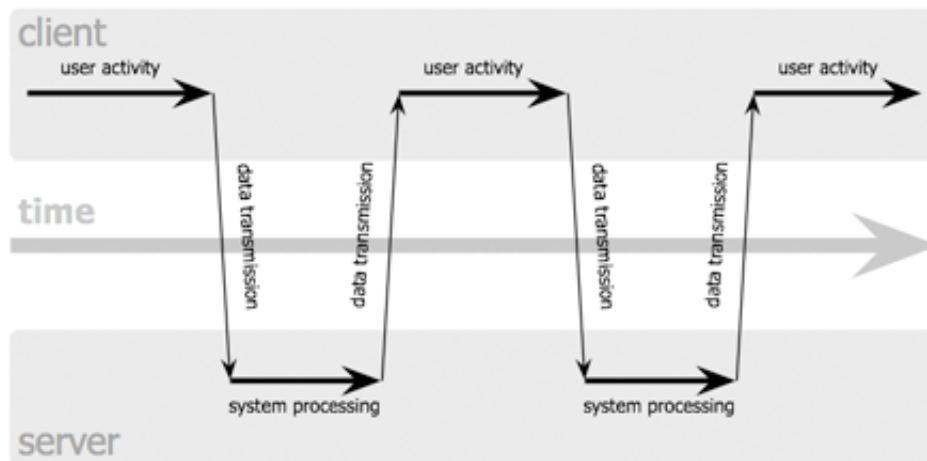


**Figure 4.2.** AJAX (right) and traditional model (left) [16]

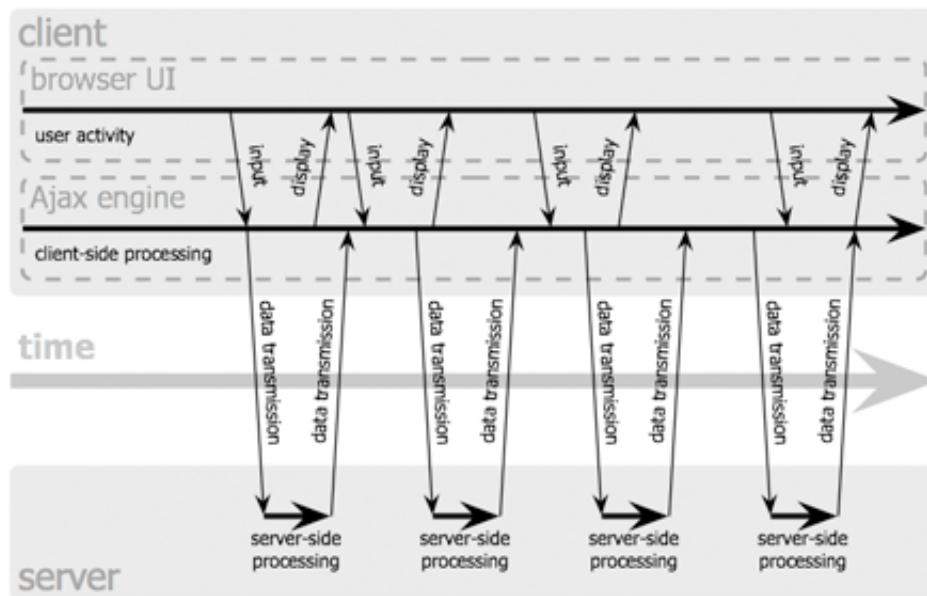
has not been changed. These limitations make web applications less favorable than desktop applications.

In contrast to the traditional web model, the Ajax model introduces an Ajax engine. This component is written in JavaScript, loaded when the web browser starts up and typically hidden from users. A user interaction then generates a JavaScript call to this Ajax engine (the right column of Figure 4.2). The engine then handles the request on its own if it is simple such as checking to see if a required field is not empty. For more complex requests that require server-side processing, it will generate a HTTP request to the server (e.g. user interaction requires inserting new record into database.) The server will respond

### classic web application model (synchronous)



### Ajax web application model (asynchronous)



Jesse James Garrett / adaptivepath.com

**Figure 4.3.** AJAX (bottom) and traditional model (top) [16]

to the request by sending a message in XML or JSON format. The browser then decodes data via the Ajax engine which updates components in the user interface. Since each user interaction does not need to be matched by a server processing (bottom portion of Figure 4.3) to update the user interface, this model is called asynchronous [16]. By using Ajax, users

do not have to wait for HTTP packets to be sent to the server, and for the result to come back in order to indicate that user input misses required fields or has included an invalid character. This feature makes web applications responsive and friendlier to users by the instant response from the Ajax engine. Ajax model also offers effective bandwidth usage since only needed data is transmitted between client and server. This flexibility enables developers to create richer web applications that can compete with desktop applications. Although providing foundation for richer web application, Ajax has one important disadvantage. Ajax requires users to enable JavaScript in their browser. However, not all the users want to turn on this feature because of the potential risk of bad and dangerous JavaScript execution. In addition to this, some devices such as mobile phones, PDAs, etc may not completely support JavaScript.

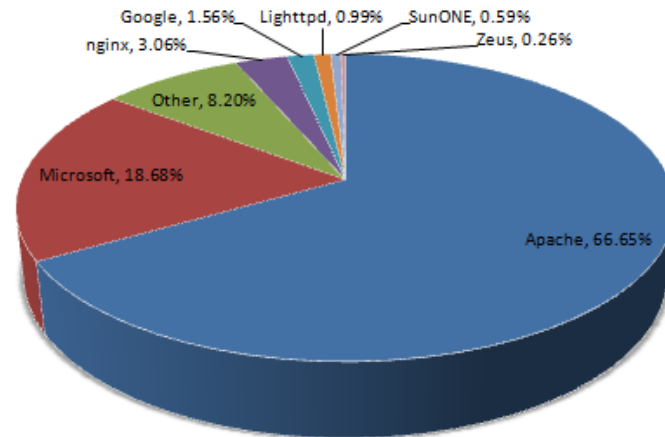
Generally, the combination of mature Ajax model, cross platform operation and rich graphical user interface make web applications a more popular choice for developers. It, however, requires a careful attention to security issues in order to make sure that using Ajax does not make the application or the client system becomes vulnerable.

### 4.3. Apache Web Server

Apache HTTP Server Project is a part of Apache Software Foundation whose objective is to create full-feature and open source implementation of an HTTP (web) server [17]. The Apache web server was initially created by Robert McCool and started as a patch to an old server known as NCSA HTTPd [18]. Today, the project is maintained by contributions from programmers and developers around the world.

The latest version of Apache is 2.2. Apache is quite popular and provides configurable, modular design, supporting multiple scripting languages (Perl, PHP, etc) and operating cross-platform from UNIX-based to Microsoft Window servers [19]. Figure 4.4 compares the number of servers running Apache to other web servers.





**Figure 4.4.** *Server Share among the Million Busiest Sites, March 2009* [20]

In the latest version, Apache provides web masters and web developers a comprehensive environment and tool for managing and building web application which includes:

- Supporting new features of latest standards
  - HTTP 1.1 and also backward compatible with HTTP 1.0
  - Common Gateway Interface (CGI), FastCGI for non-Perl CGI
  - Secure Socket Layer (SSL)
  - Java Servlet
  - Server Side Include (SSI)
- Simple and efficient file-based configuration: configuring and changing Apache behavior with the primary configuration file *httpd.conf*. We can also tell Apache to handle more than one domain name by using only simple text editor like Notepad. Apache can determine virtual hosts either by IP (IP-based) or name (name-based) monitoring.
- Supporting PHP, Perl and many other scripting language with modular design. This allows Apache to host most open source software packages of web development and makes it favorable to businesses for saving expenses. Modular design makes Apache extensible to accommodate new technology.

- Flexible and customizable logs and server status. These feature allows web master to easily monitor websites with their own specifications. They can quickly track down problems to source when a failure occurs.
- Implementing message-digest-based authentication: basic HTTP authentication for web servers.

## **4.4. Server-Side Scripting with PHP**

### **4.4.1. Server-side Scripting**

Server-side scripting is a web technology in which a web page is dynamically and programmatically generated by executing a script on a web server based on requests from users. These scripts often submit queries to one or many backend databases on server side, assemble the results into HTML documents and send them back to clients. Security for client and server is the main advantage of server-side scripting language. A server-side script file is always executed by the server and generates a HTML response that is sent back to clients. Since no code needs to be executed at the client side, the developers have minimized the risk of illegally running dangerous codes. The client program (web browser) cannot view the source code which produces the web page. Therefore, it cannot retrieve any information about the underlying resources or about the server such as database structure, application configuration, etc. It is also convenient for web browsers which only receives and renders HTML. Browsers consequently do not need to support the different server-side scripting languages. In conclusion, server-side scripting is an good way for delivering web services as well as protecting web applications.

### 4.4.2. PHP

PHP stands for Hypertext Preprocessor. PHP is a widely-used open source scripting language. PHP gains its popularity by offering wide variety of features and compatibility with many platforms [21]:

- A server-side scripting language: PHP collects form data, generates HTML web pages, sets and retrieves cookies, and communicates with a web server. It also can produce PDF file or stream data such as Flash video on the fly;
- A useful command line scripting: PHP is packed with useful text processing functions which include regular expression processing and XML document parsing;
- A cross-platform programming language: PHP independently operates on major platforms such as Microsoft Windows, Linux, Mac OS X, Solaris, etc. PHP is also supported by most popular web servers. Microsoft Information Internet Services and Apache are two well-known web servers that provide PHP support;
- Comprehensive database abstraction extensions: PHP provides object-oriented classes that encapsulate the complexity of interacting with MySQL, Microsoft SQL Server, Microsoft Access, Oracles, and many databases, making PHP a popular tool for application development;
- PHP can also communicate with mail services by using POP3, LDAP, IMAP, SNMP and other protocols;
- With the use of PHP-GTK, PHP can be used to develop desktop applications;

Despite of the emergence of new server-side scripting languages such as Ruby, Groovy and the commercial competitors such as ASP.NET, PHP is remains a popular tool for web developers. According to Tiobe Programming Community Index of May 2009 [22], it is the fourth popular programming language after Java, C and C++.

### 4.4.3. Standard PHP Library (SPL) and Auto Loading

#### 4.4.3.1. Standard PHP Library

The Standard PHP Library is a set of object-oriented facilities, standard data structures for resolving standard and general problems of handling exceptions, collection iteration, automatically loading classes and physical file accessing jobs. SPL tries to bring the advantages of object-oriented principals to increase the productivity of PHP coding. The following list presents some important features of SPL [23]:

- Standard data structures: *SplDoublyLinkedList*, *SplStack*, *SplQueue*, etc
- For collection iteration: *ArrayIterator*, *CachingIterator*, *DirectoryIterator*, *SimpleXMLIterator*, etc
- Exception handling: *BadFunctionCallException*, *BadMethodCallException*, *OutOfBoundsException*, *OutOfRangeException*, *OverflowException*, etc
- Auto loading: *class\_implements*, *spl\_autoload*, *spl\_autoload\_register*, etc
- File handling: *SplFileInfo* class

In the above list, the `spl_autoload` and `spl_autoload_register` are the key components that allow an application in order to load classes on the fly to handle requests and process data.

#### 4.4.3.2. Auto Loading

In PHP language, auto loading is an advanced mechanism for dynamically instantiating objects of classes without explicitly using *include*, *include\_once*, *require*, *require\_once*. *spl\_autoload* is called whenever a class is called in a PHP script. When a class is called for the first time, `spl_autoload` loads the source code file of the class based on the include path and the extensions registered with PHP. PHP allows developers to define and register their own auto load function for user-defined classes by using *spl\_autoload\_register*. Besides freeing our code from large sections of include statements, this dynamic loading allows us to load a requested class, instantiate objects and execute their methods on demand rather

than loading all the source files at start up. This mechanism has given use to the engine of web applications which is developed using a dynamic design pattern. Such applications may have many code modules which are organized in a large number of source code files. Without dynamic loading of files, loading all of these source code files at once consumes server resources and impacts the server performance significantly. Dynamic loading prevents wasteful heavy loads on the server because not all the functions are needed in most common scenarios.

## 4.5. MySQL

The IU-Advise system uses MySQL Community version 5.1.34 as the backend database engine. MySQL is an open source relational database management system (RDBMS) which is used for managing and manipulating relational databases. MySQL is owned and financed by MySQL AB, a subsidiary of Sun Microsystems and Oracle Corp. The SQL-92 standard compatibility, high availability for wide range of operating systems on different computer architectures and low total cost of ownership are the key features that spread MySQL all over the software industry world. Currently, there are two types of MySQL versions provided [24, 25]:

- Community: a free version that is intended for users or organizations that are comfortable with installing, configuring, managing and securing databases with standard documentations and support;
- Enterprise: is a paid version bundled with many first-class support services which include:
  - Automated notification for updates and new versions
  - Technical advice for installing, configuring, etc
  - Fast response time resolution for technical problems

Both versions of MySQL provide a programming API for Java, .NET, C++, PHP, Ruby, and Python. This capability provides flexibility to developers to select their programming language and framework for building applications. As a RDBMS, MySQL provides developers facilities for organizing, querying and manipulating data. If referential integrity is desired, the developers can use the INNODB engine for data tables. Views and stored routines are offered for optimizing query performance. Constraints and triggers are included for automating data consistency checking. In addition to offering tools to developers, MySQL also has a built-in tool to secure database objects, control and authorize data access through Access Privilege System and User Account Management. Administrators also are equipped with wide variety backup and recovery strategies.

### 4.5.1. Views

Views (prewritten queries) are composed of multiple data tables. Data columns or fields in a view can be aliased in a more meaningful way. This feature makes the data in view more convenient to be used by the developers. By using views, developers can perform complex data manipulation without needing to be database or SQL experts. Starting with MySQL version 5.0, a view under certain conditions can update, insert and delete records in underlying data tables which define it. Views also are good tools for hiding the structure of a database from unauthorized developers. Below is an example of a view which combines data from *enrollment*, *completion\_method*, and *student* into a single and convenient view named *student\_testout\_view*.

#### Listing 4.1. A view definition

```
CREATE OR REPLACE
ALGORITHM=UNDEFINED
DEFINER='root'@'localhost'
SQL SECURITY DEFINER VIEW `student_testout_view` AS
SELECT
    `completion_method`.`Description` AS `Description`,
```

```

    'student`.`StudentID` AS `StudentID`,
    'enrollment`.`Explanation` AS `Explanation`,
    'enrollment`.`Grade` AS `Grade`
FROM (('enrollment` JOIN `completion_method` ON
      (('enrollment`.`CompletionMethodID` = `completion_method`.`CompletionMethodID`)))
JOIN `student` ON
      (('enrollment`.`StudentID` = `student`.`StudentID`))
WHERE (('completion_method`.`CompletionMethodID` = 'TESTOUT') AND
      ((NOT(('enrollment`.`Grade` LIKE '%A%'))
      OR (NOT(('enrollment`.`Grade` LIKE '%B%'))
      OR (NOT(('enrollment`.`Grade` LIKE '%C%'))
      OR (NOT(('enrollment`.`Grade` LIKE '%D%'))
      OR (NOT(('enrollment`.`Grade` LIKE '%F%')))))

```

### 4.5.2. Stored Routines

A stored routine is a set of SQL statements that are stored in a database server to query or manipulate database objects. MySQL categorizes stored routines into two types: stored procedures and functions. A stored procedure has parameter list and return result either as set of data rows or a single value. It also accepts output parameters which can be used for returning value. A stored procedure can return a set of data rows and multiple single values by output parameters at the same time. Similarly, a function also have parameter list but its parameters are always input parameters. Another restriction on a function is that it can only returns a single value.

Stored routines offer some advantages in data accessing performance and cross platform execution. First, a stored routine is only compiled the first time it is called. No compilation is necessary in the subsequent calls to the procedure unless its content or properties are changed. This characteristic provides improved performance for recurring queries. Submitting a SQL statement to MySQL has lower performance because a SQL statement has to be parsed, optimized and compiled each time it is submitted to a MySQL server. Second, a stored routine can be invoked by using its name and passing parameters. This syntax

usually is shorter than the content of SQL statements that actually performs the processing. Therefore, bandwidth requirement is reduced. Third, as stored routines are stored in database server and written in SQL syntax, they can be executed by different programming languages and platforms. Platform independence makes stored routines reusable and promotes consistency. Finally, stored routines can be used as security mechanism to insure that developers can only access to authorized information. Using stored routines, however, increases the load on server since most of the processing is done on the server. As the result, a busy server can be flooded with multiple long runtime stored routines.

Listing 4.2 is a definition of a stored procedure. This stored procedure accepts two VARCHAR input parameters and uses them in the WHERE clause to retrieve all the advisor remarks of an advisor (pAdvisorID) about the given student (pStudentNetworkId).

**Listing 4.2.** *A stored procedure definition*

```

CREATE DEFINER= 'root'@'localhost '
PROCEDURE 'get_all_advisor_remarks_by_advisor_student_sp'(
IN pStudentNetworkID VARCHAR(10),
IN pAdvisorId VARCHAR(8))
BEGIN
SELECT
    communication_type.CommTypeName, advisor_remark.AdvisorRemark,
    advisor_remark.AdvisingDate, campus.Name AS CampusName,
    college.Name AS CollegeName, department.Name AS DeptName,
    advisor.LastName AS AdvisorLastName, advisor.FirstName AS AdvisorFirstName,
    advisor_remark.Active, advisor_remark.Level,
    advisor_remark.StudentID, advisor_remark.AdvisorID,
    advisor_remark.CollegeID, advisor_remark.CampusID,
    advisor_remark.DeptID
FROM
    communication_type
INNER JOIN advisor_remark ON
    (communication_type.CommTypeID = advisor_remark.CommunicationType)
INNER JOIN student ON (advisor_remark.StudentID = student.StudentID)

```



```
INNER JOIN advisor ON (advisor_remark.AdvisorID = advisor.AdvisorID)
AND (advisor_remark.DeptID = advisor.DeptID)
AND (advisor_remark.CollegeID = advisor.CollegeID)
AND (advisor_remark.CampusID = advisor.CampusID)
INNER JOIN department ON (advisor.DeptID = department.DeptID)
AND (advisor.CollegeID = department.CollegeID)
AND (advisor.CampusID = department.CampusID)
INNER JOIN college ON (department.CollegeID = college.CollegeID)
AND (department.CampusID = college.CampusID)
INNER JOIN campus ON (college.CampusID = campus.CampusID)
WHERE student.NetworkId = pStudentNetworkID
AND advisor_remark.AdvisorID = pAdvisorID;
END
```

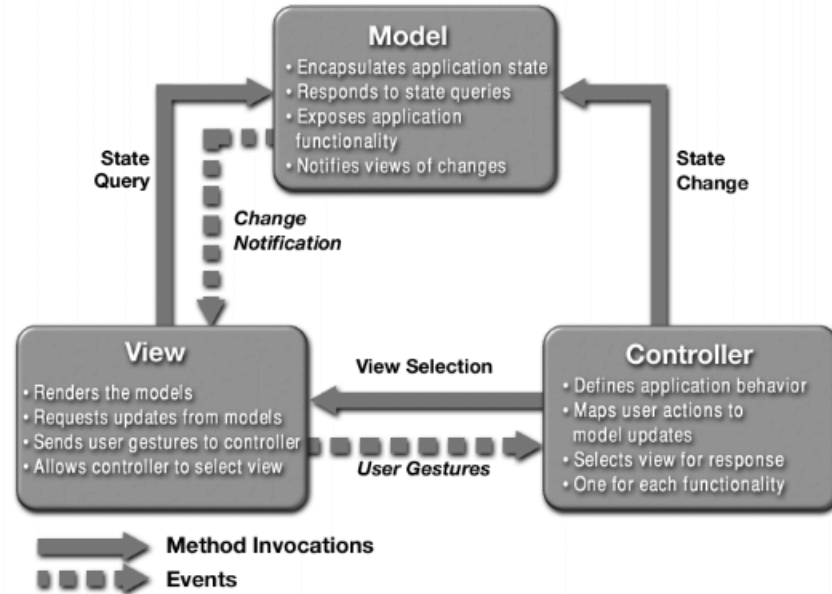
## 4.6. Design Patterns

### 4.6.1. Model-View-Controller (MVC)

#### 4.6.1.1. Introduction

MVC or "Model-View-Controller" [26] is a software design pattern which was first implemented in user interface framework of Smalltalk-80 v2.0 library of Xerox PARC. MVC in Smalltalk-80 was used to solve the problem of generating multiple views based one computer model. This pattern decomposes an application into three main components [27]:

- A model that queries and manipulates data of an application;
- A view receives the data and the model's states to render corresponding presentation and update user interface;
- A controller which catches user input such as keystrokes, mouse activities, etc. It then translates them into command to alert the model for processing data and choose the appropriate view for rendering;



**Figure 4.5.** *Structure of MVC model*

#### 4.6.1.2. *Basic Concepts*

Models [28, 27] are data structures that present data. Models usually include business rules/logics and information which is used for making decisions about changing data and changing the state of the models themselves. In procedural programming paradigm, a set of procedures or sub-routines can be used to form a model. For object-oriented programming paradigm, classes are usually applied to building models. A class Student, for instance, like in Figure 4.6 can be a model that governs data about a student.

<b>Student</b>
-Student ID : String
-First Name : String
-Last Name : String
-Available Date : Date
+Set Available() : Boolean

**Figure 4.6.** *Structure of Student class*

The components produce a form presentation to visualize the states of models based on data retrieved from models are called views. In a web application, views generate HTML files or any requested type of files to respond to users.

The controllers directly handle user inputs such as keystrokes, HTTP requests, etc and translate them into commands to manipulate the model and view components. Users only interact with the application through the controllers.

#### 4.6.1.3. *Communication between MVC components in a web application*

The control flow of an web application designed by applying MVC pattern starts with the controller receiving a HTTP request. The controller has to parse the HTTP request into an understandable form to the web application which includes named variable. The controller then retrieves certain parameters to determine the model (application logic) that needs to be notified for data processing and the view (user interface) that need to be rendered. After getting the necessary information, the controller hands the request to the chosen model. The chosen model queries and manipulates data from the back-end database. The model also changes its state if required. Finally, the processing result is passed to the view for rendering appropriate presentation or updating user interface.

### **4.6.2. Centralized controller - Front Controller Pattern**

A complex web application may contain services that are used by all of components. Authentication and authorization, for example, are common processes need to be performed before allowing the users to access a service. With a large application, it is required to have a mechanism to manage these components so that programmers can maintain and extend the functionalities of these kind of services without worrying of forgetting to update new changes in all segments of codes that use them. This is also a problem of an MVC web application since it has many controllers to accept user input. Therefore, the common services described above need to be applied in each controller by including the same file.

In a large web application correctly managing this replication is a non trivial task. Moreover, a web application is less secure with multiple entry points. Therefore, having one central gateway that provides uniform access mechanism to all services is a good solution to overcome those disadvantages. This approach also more secure and maintainable since we redirect all user requests to one centralized controller for preprocessing requirements such as authentication and authorization. The centralized front controller helps to secure applications, reduce redundancy and maintenance cost.

## 4.7. MVC in Zend Framework Implementation

The concept of MVC is applied in many frameworks for developing web applications. Table 4.1 lists major and well-known frameworks that utilize MVC design pattern.

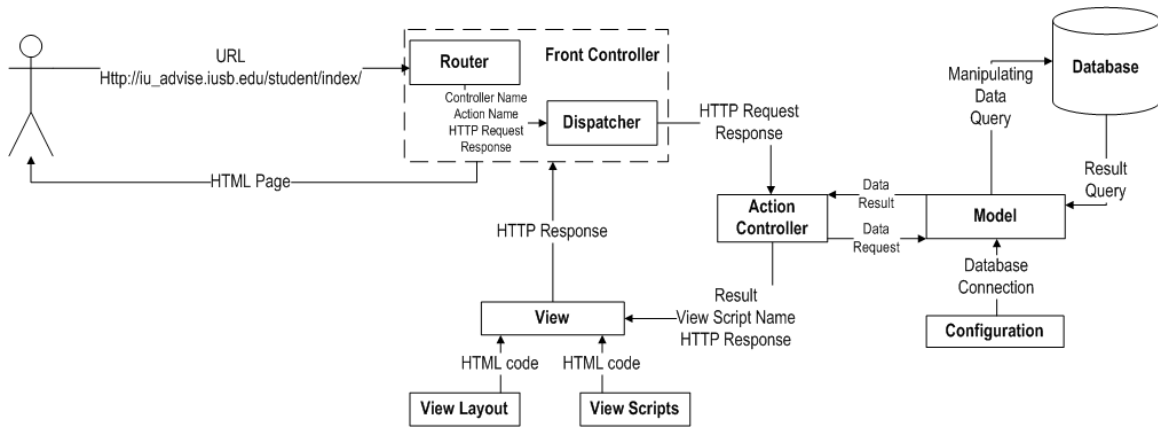
Language	Frameworks
PHP	Zend Framework, CakePHP, CodeIgniter, Drupal, Symfony Framework, PureMVC
Ruby	Ruby on Rails
C#.NET and VB.NET	ASP.NET MVC Framework

**Table 4.1.** MVC implementations for different languages

Zend Framework implements the MVC through a set of abstract classes which includes:

- The Controller component of MVC design pattern is modeled by `Zend_Controller_Action`. In addition to this class, `Zend_Controller_Dispatcher_Abstract`, `Zend_Controller_Plugin_Abstract`, `Zend_Controller_Request_Abstract`, `Zend_Controller_Response_Abstract`, and `Zend_Controller_Router_Abstract` automate the routing and information interchange among controllers;
- The View Helper, derived from `Zend_Controller_Action_Helper_Abstract` class, corresponds to the View component of MVC design pattern;

- The Model component of MVC design pattern does not have an explicit correspondence class in Zend Framework. Zend Framework, however, provides set of Zend\_Db prefixed class to meet all the requirements of developers in implementing Model components.



**Figure 4.7.** *Zend implementation of MVC design pattern*

Figure 4.7 presents the implementation of Zend Framework for modeling MVC design pattern. As a HTTP request comes to a web application, it first is processed by the Front Controller. At the Front Controller, the URL is parsed to determine the name of the specialized controller and action to generate responds. For example, the Student controller has DegreeAudit action to produce the degree audit report. The Front Controller also packed all the parameters into the HTTP Request. Having the specialized controller and action's names, the Front Controller dispatches the HTTP Request and Response objects to the specified Action Controller. The Action Controller then retrieves parameters from the HTTP Request. It then requests data from one or more Models which communicate and manipulate with the back-end database. When all the inputs are ready, the Action Controller executes the processing steps, specify the view script name and send processing result to the View component. View component uses the view script name to invoke the code for generating the output for sending back to the users.

In addition these abstract classes, Zend Framework also offers a wide variety of components to facilitate web developers manage application configuration, implement security solutions, and meet the complex requirements of web applications. These classes are listed with brief overviews in the Appendix. Developers can also communicate with popular web services of Yahoo, Google, Amazon, Delicious, Twitter, etc. to provide rich featured applications using Zend Framework.

Zend Framework is a really useful tool for server-side web developers. Advantages of this powerful and professional object-oriented programming library can be realized regardless of the whether MVC pattern is used. Classes used in this thesis are just a subset of the whole framework. Further potential use and benefits from using Zend Framework need more careful investigation. In general, Zend Framework brings PHP applications to a professional level that competes with commercial web applications. The use of PHP and Zend Framework allow the current system to be customized, maintained and expanded to accommodate changes and new requirements in the future.

## 5. Conclusion

Academic advising plays an important role in creating a friendly and relevant educational environment for college students. At the same time, advising can be a complex and time consuming process for academic advisors especially with the dynamic nature of the degree programs and degree requirements within educational institutions. Despite these challenges, academic advisors always try to do their best to offer accurate, up-to-date and consistent advising information to their students. The goal of this thesis is to explore the design and implementation of a computerized tool to facilitate this process. IU-Advise was designed to assist advisors in their efforts to provide quality academic advising services to students.

Upon proper authentication, students are able to view their own grades for a specified semester, unofficial transcript, advising history and degree audit which shows a student's progress toward a degree. Similarly, advisors can select a particular advisee from a list and view their progress, conduct advising information and record notes to themselves and other advisors or to the students. Special attention has been given to managing the privacy of the advising history and advisor comments. The system is implemented using PHP, Zend Framework, Apache, and MySQL and has been tested with Microsoft Windows Vista and Fedora 10. These open source software packages not only cut down the expenses but also offer a set of professional tools for developing a customizable, flexible and reliable application. The use of MVC design pattern and Zend Framework make the system more adaptable to new changes and requirements. These tools also increase the reusability of components in IU-Advise system. MySQL database objects such as stored routines and views enable the reusability of backend database. To manage security threats on server side, Zend Framework provides useful classes for automating common security procedures such as validating user inputs, removing unexpected tags, authenticating with a data table

in a database and authorizing users with a flexible but powerful data structures. The Ajax model is applied for improving the user interaction with web applications.

This project helps in understanding the implementation of Zend Framework, principals of MVC design pattern and PHP. This project exposes the author to the challenges of designing and developing reusable software components. The implementation of this project also equipped the author with deeper understandings of developing a web application at the professional level. The thesis provides a foundation for additional exploration of applying design patterns and using frameworks in the development of web applications. Additional area which could be explored but was outside the scope of this thesis includes the implementation of administrative subsystem. This subsystem provides a friendly user interface to the university administrators as they attempt to add new information to the system. Such new information includes new programs, courses, student information, prerequisite requirements, advisor assignments and degree requirements. Another feature which can be helpful would be to store tentative advising information such as course selections for future semesters in database. This functionality allows an advisor to check if his or her advisee is following previously agreed upon schedule.



**Appendix A.**  
**Zend Framework**

## A.1. MVC Implementation Classes

### A.1.1. Zend\_Controller\_Front and The Dispatching Loop

Figure A.1 shows how a request is received by the Zend Framework, how it is dispatched and handled by the proper Action Controller. `Zend_Controller_Front` is the main component that controls the whole process of processing request from parsing HTTP from user to sending responses to the clients. The `Zend_Controller_Front` also instantiates a `Zend_Controller_Response_Http` object for sending response to clients.

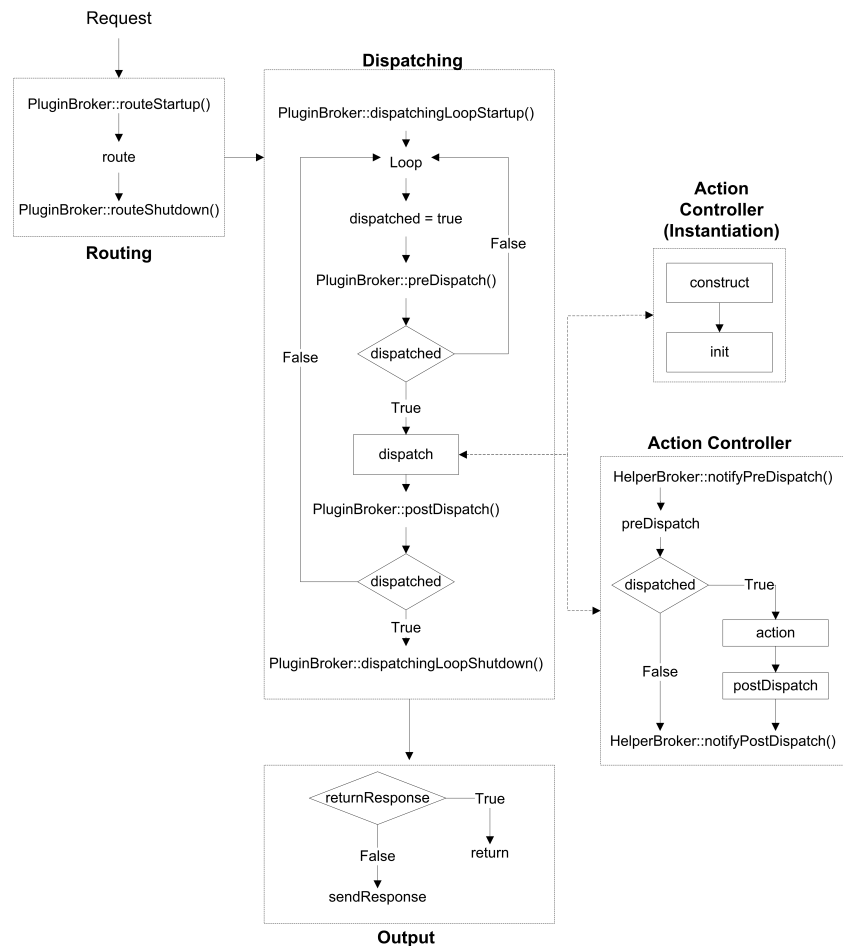


Figure A.1. Dispatching cycle of Zend Framework

To be more specific, the dispatching process needs to determine the controller and the action needed to process requests. This sub process is called "Routing". To resolve the controller and action name from the URL, the `Zend_Front_Controller` instantiate an object of `Zend_Controller_Router_Rewrite`. This router contains all the declared routes to controllers and actions in a web application. We can load these routes by using a configuration file and `Zend_Config` class. This router, however, always adds the default route automatically to `IndexController` and `IndexAction` which are required for MVC application. At the end of this Routing process, the router calls private method `_setRequestParams()` on the current `Request` object to set the module, controller, and action name. The actual controller is then instantiate and the action is called in the dispatching loop by an object of `Zend_Controller_Dispatcher_Standard` class. In figure [A.1](#), we have two internal steps in the `dispatch()` call. The first step is the dynamic instantiation of the controller we want to work with. This step is completed in the `dispatch()` method of the `Zend_Controller_Dispatcher_Standard` object. In second step, the control is transferred to the requested controller's code in order to execute the action.

Finally, the controller checks to see if it needs to return the `Response` object to the caller or directly send the response to clients.

### **A.1.2. Zend\_Controller\_Plugin\_Abstract Class**

Plugins are user code segments that are executed when a certain event is raised in the scope of `Zend_Front_Controller` execution (figure [A.1](#)). They are useful complements to `Zend_Front_Controller` for implementing common services that needs to be applied for all controllers. To implement a plugin, we extend (inherit) `Zend_Controller_Plugin_Abstract` class. We have to provide code for one or more of the following event method in our class:

- `routeStartup()` is raised before `Zend_Front_Controller` calls the router to determine the controller and action to process the request;
- `routeShutdown()` is raised after the router finishes routing the request;

- `dispatchLoopStartup()` is raised before `Zend_Controller_Front` get into the loop of processing requests;
- `preDispatch()` is raised before the dispatcher dispatches the request. At this point, we can alter the request or reset the its dispatched flag to replace the controller and action or skip the current controller and action;
- `postDispatch()` is raised after an action is dispatched. We can change manipulate the request and its dispatched flag if we need to apply new actions from other methods;
- `dispatchLoopShutdown()` is raised after `Zend_Controller_Front` get out of the dispatch loop;

Listing A.1 defines a plugin to check the existence of a username in the session data. If there is no username, the request is rerouted to the default controller. A sample code segment used to register this plugin is shown in Listing A.2. After this registration, the plugin is automatically triggered in the `dispatchLoopStartup` event.

**Listing A.1.** *Plugin declaration*

```
class IU_Plugin_Authenticate extends Zend_Controller_Plugin_Abstract
{
    public function dispatchLoopStartup(Zend_Controller_Request_Abstract $request)
    {
        if(!isset(Zend_Registry::get('sessionDb')->username) ||
            empty(Zend_Registry::get('sessionDb')->username)){
            $request->setModuleName('default');
            $request->setControllerName('index');
            $request->setActionName('index');
        }
    }
}
```

In order to let the `Zend_Front_Controller` know of existence of a plugin, classes must be registered by calling its `registerPlugin()` (Listing A.2). This method in turn calls the `registerPlugin()` method of `Zend_Controller_Plugin_Broker` class. The object of the `Zend_Controller_Plugin_Broker` then informs our registered plugins when an event is raised.

**Listing A.2. Register a plugin**

```
// Register plugin
$pluginIU = new IU_Plugin_Authenticate();
$frontController->registerPlugin($pluginIU);
```

**A.1.3. Zend\_Controller\_Action**

`Zend_Controller_Action` is the parent class for implementing the real controller which corresponds to the controller part of MVC design pattern. By inheriting this class, developers have access to the Request object for retrieving parameter's values, view helper for controlling how to render result, FrontController object for interacting with the dispatching process, etc. To have a full feature parent class, a developer usually has to build his or her own parent class depends on the particular requirements of each application. Listing A.3 is an example of a user-defined controller action class. In this custom controller class, a protected method was added to validate username and role (line 6 to 11) for the child class to reuse it.

**Listing A.3. IU\_Controller\_Action class definition**

```
1 abstract class IU_Controller_Action extends Zend_Controller_Action
2 {
3     /**
4      * This function checks valid username and role
5      */
6     protected function _isValidRoleAndUserName()
7     {
8         $roleValidator = new Zend_Validate_InArray($this->_validRoles);
9         return (!empty($this->_currentRole) and !empty($this->_currentUserName)
10             and $roleValidator->isValid($this->_currentRole));
11     }
12 }
```

**A.1.4. Zend\_Controller\_Action\_Helper\_Abstract Class**

The main aim of `Zend_Controller_Action_Helper_Abstract` is to supply `Zend_Controller_Action` with runtime or on-demand functionalities. The `Zend_Controller_Action_Helper_Abstract`'s child classes helps developers to automate the necessary services that need to be applied after each controller action finishes processing. Helpers are triggered when

a `Zend_Controller_Action` calls its `notifyPreDispatch()` and `notifyPostDispatch()` method (figure A.1). For example, `Zend_Controller_Action_Helper_ViewRenderer`, which covers the view part of MVC design pattern, is the default `Zend_Controller_Action_Helper` for most normal HTTP requests. It is invoked in the `notifyPostDispatch()` method of each action controller for producing HTML page. Another good example is `ContextSwitchHelper`. This powerful helper allows one action controller to return more than formats on a request. It means that an action controller can respond to HTTP requests which need a HTML page as well as Ajax requests which prefer JSON or XML format. To automatically trigger and manage helpers, `Zend_Controller_Action_HelperBroker` object uses its `Zend_Controller_Action_Helper` stack. When an event in `Zend_Controller_Action` is raised, `Zend_Controller_Action_HelperBroker` checks the stack and invokes the appropriate method of all the helpers in the stack.

### **A.1.5. Zend Framework Class for Model Component**

As we mentioned above, Zend Framework do not have explicit abstract classes for implementing Model components. However, it implicitly facilitates developers with a set of class which are prefixed by `Zend_Db`:

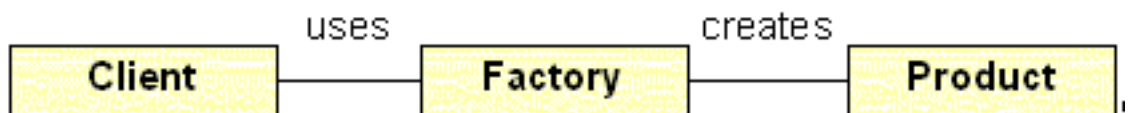
- `Zend_Db_Adapter` uses factory design pattern to handle creation of database connection;
- `Zend_Db_Table`, `Zend_Db_Table_Row`, `Zend_Db_Table_Row_Set`, and `Zend_Db_Table_Relationships` abstract data tables, relationships between them and most data manipulation operations using object-oriented programming;
- `Zend_Db_Statement` offers developers an object-oriented class to executing SQL statements.

In the current implementation, we did not use all the provided `Zend_Db` classes. We make use `Zend_Db_Adapter`, `Zend_Db_Table` and `Zend_Db_Statement` for developing Model components.

### A.1.5.1. *Zend\_Db\_Adapter and Factory design pattern*

Nowadays, many relational database management system (RDBMS) are being offered to developers, vendors and businesses. Developers can choose between the commercial products like Oracle, Microsoft SQL Server and MySQL (Enterprise version) or from open source solution like SQLite, PostgreSQL or compact version of MS SQL Server. In addition to these popular RDBMS, Microsoft Access of Microsoft Office Suite and Base of Open Office Suite also contribute to richness of RDBMS market. With wide variety of options, developers need to produce an application that can interact with more than one. To meet this crucial need, PHP offers vendor-specific extensions for working most of popular RDBMS. Although these extensions satisfy the developers' desires, they make applications less flexible because the differences in programming interfaces. Therefore, consolidating the programming interfaces of different extensions is critical to increasing the flexibility of developing and deploying web applications.

To overcome this challenge, we have to build an abstract class that does not only accommodate the differences between extensions' programming interfaces but also produces standardize objects based on the configuration set up by developers. This type of polymorphic development can be achieved by applying Factory design pattern [29]. Figure A.2 presents the logical model of Factory design pattern [30]. In this design pattern, the client factor does not create directly create an instance of the requested product. Instead it setups and provides information to the factory factor which uses this to instantiate the appropriate object and return it to the client.



**Figure A.2.** *Logical model of Factory design pattern*

To better understand this pattern, let take a look at Listing A.4 and A.5. In two listings, we only see two differences:

- The content of *\$params* array is specifically setup for each type of database
- The name of database driver: *PDO\_SQLITE* and *PDO\_MYSQL*

Both listings return *\$db* as a *Zend\_Db\_Adapter*. In both cases, a developer can use code as shown in Listing A.6 to create server table on the database.

**Listing A.4.** *Create MySQL database connection with Zend\_Db factory*

```
require_once 'Zend.php';
Zend::loadClass("Zend_Db");

$params = array (
    'host' => '127.0.0.1',
    'username' => 'user',
    'password' => 'password',
    'dbname' => 'example'
);

$db = Zend_Db::factory('PDO_MYSQL', $params);
```

**Listing A.5.** *Create SQLite database connection with Zend\_Db factory*

```
include "Zend.php";
Zend::loadClass("Zend_Db");

$params = array ('dbname' => 'example');

$db = Zend_Db::factory('PDO_SQLITE', $params);
```

**Listing A.6.** *Execute query by using an object of Zend\_Db factory*

```
$db->query('CREATE TABLE server (key, value)');
```

### A.1.5.2. *Zend\_Db\_Table*

*Zend\_Db\_Table* is an object-oriented programming interface that supports many methods for executing common data manipulation operations. This abstract class hides the detail of building and submitting SQL statements to the server by using the public method concept of object-oriented programming. *Zend\_Db\_Table* internally uses *Zend\_Db\_Adapter* for connecting to databases and executing queries when we call its *insert()*, *update()*, and *delete()* methods. It is more convenient to have a default database adapter available in the



Zend\_Db\_Table\_Abstract class since all our classes extends Zend\_Db\_Table and inherits all protected members from it. To set a default Zend\_Db\_Adapter, we can use listing [A.7](#) with *\$dbAdapter* created by using one of above code segment

**Listing A.7.** *Set default database adapter*

```
Zend_Db_Table_Abstract::setDefaultAdapter($dbAdapter);
```

After setting up the database adapter, the Zend\_Db\_Table class can be extended to declare a customized class. In simple case, we only need to provide the table name that matches with the one in database (Listing [A.8](#)).

**Listing A.8.** *Zend\_Db\_Table definition*

```
class Model_DbTable_AdvisorRemarks extends Zend_Db_Table
{
    protected $_name = 'advisor_remark';
}
```

By using the declaration from Listing [A.8](#), we can insert a record to *advisor\_remark* table with following code segment:

**Listing A.9.** *Zend\_Db\_Table insert() method*

```
1 public function save(array $data)
2 {
3     $table = $this->_table;
4     $fields = $table->info(Zend_Db_Table_Abstract::COLS);
5
6     foreach ($data as $field => $value) {
7         if (!in_array($field, $fields)) {
8             unset($data[$field]);
9         }
10    }
11    return $table->insert($data);
12 }
```

Listing [A.9](#) first retrieves the field list of current table by using the static property *COLS* of Zend\_Db\_Table\_Abstract class. The foreach loop from line 6 to 10 removes any field that is not in the field list. Finally, it inserts the new record to database.

The procedure of deleting a record with Zend\_Db\_Table object includes building the WHERE clause to determine the record need to be removed and a call to the *delete()* method with this WHERE clause. The WHERE is usually built based on the primary key of a table. In the constructing process of the WHERE clause (line 8 to 18), it is recommended to use

the *quoteInto()* method of the *Zend\_Db\_Adapter* in order to make sure that we always have correct quoted string and date. The database adapter retrieves this quoting information based on the vendor-specific extension being used. This WHERE clause is actually an array composed of many single statements. These statements are combined by AND operator by default in the final SQL statement.

**Listing A.10.** *Zend\_Db\_Table delete() method*

```

1 public function deleteRows($pDataRows)
2 {
3     $currentAdapter = $this->_table->getAdapter();
4     $deletedCount = 0;
5
6     foreach($pDataRows as $currentRow){
7         // Build where clause
8         $whereStatement = array();
9         $whereStatement[] = $currentAdapter->quoteInto('AdvisingDate = ?',
10             $currentRow['AdvisingDate']);
11         $whereStatement[] = $currentAdapter->quoteInto('AdvisorID = ?',
12             $currentRow['AdvisorID']);
13         $whereStatement[] = $currentAdapter->quoteInto('CollegeID = ?',
14             $currentRow['CollegeID']);
15         $whereStatement[] = $currentAdapter->quoteInto('DeptID = ?',
16             $currentRow['DeptID']);
17         $whereStatement[] = $currentAdapter->quoteInto('CampusID = ?',
18             $currentRow['CampusID']);
19         $deletedCount += $this->_table->delete($whereStatement);
20     }
21     return $deletedCount;
22 }

```

Similar to the deleting procedure, the updating function in Listing A.11 has the equivalent structure with WHERE clause constructing followed by a call to update method. However, there is a difference in parameter list of *update()* method. This method requires two parameters. The first parameter is the new values in form an array of key  $\Rightarrow$  value elements. The key is the field or column name and value is the new value for the field. The WHERE clause is used as the second parameter.

**Listing A.11.** *Zend\_Db\_Table update() method*

```

1 public function updateRows($pDataRows)
2 {
3     $currentAdapter = $this->_table->getAdapter();
4     $updatedCount = 0;
5     foreach($pDataRows as $currentRow){
6         // Build data
7         $newData = $currentRow['NewData'];

```

```

8         $whereStatement = array();
9         // Build where clause
10        $whereStatement[] = $currentAdapter->quoteInto('AdvisingDate = ?',
11            $currentRow['AdvisingDate']);
12        $whereStatement[] = $currentAdapter->quoteInto('AdvisorID = ?',
13            $currentRow['AdvisorID']);
14        $whereStatement[] = $currentAdapter->quoteInto('CollegeID = ?',
15            $currentRow['CollegeID']);
16        $whereStatement[] = $currentAdapter->quoteInto('DeptID = ?',
17            $currentRow['DeptID']);
18        $whereStatement[] = $currentAdapter->quoteInto('CampusID = ?',
19            $currentRow['CampusID']);
20        $updatedCount += $this->_table->update($newData, $whereStatement);
21    }
22    return $updatedCount;
23 }

```

### A.1.5.3. *Zend\_Db\_Statement*

*Zend\_Db\_Statement* brings the convenience for developers with intensive knowledge of SQL programming. It allows programmers to execute user-generated SQL statements, call stored routines or complex queries that cannot be captured by *Zend\_Db* class. By providing this class, Zend Framework free developers from the time consuming process of declaring *Zend\_Db\_Table* classes and defining relationships among them using *Zend\_Db\_Table\_Relationship* in order to make use of all the functionalities of *Zend\_Db*. *Zend\_Db\_Statement* is usually not created directly. In most scenarios, we use the `query()` method of *Zend\_Db\_Adapter* to execute a statement for returning a *Zend\_Db\_Statement* object. Listing A.12 makes a call to *degree\_audit* stored procedure and use the *Zend\_Db\_Statement* to fetch the result of the processing.

#### **Listing A.12.** *Example of Zend\_Db\_Statement object*

```

// Get current database adapter
$currentDbAdapter = Zend_Registry::get('dbAdapter');
$degreeAuditStatement = $currentDbAdapter->query('CALL degree_audit');
$this->_degreeAuditDbStatement = $degreeAuditStatement->fetchAll();

```

## A.2. Useful Classes from Zend Framework

Although MVC implementation is the key feature of Zend Framework, it is not the only gem in this functional programming framework. Zend Framework is not only shipped with MVC idea but also packed in it all the best practices of object-oriented programming and developing web programming of the web developer community.

### A.2.1. Application Configuration with Zend\_Config

Zend\_Config provides an easy-to-use programming interface for accessing and managing configuration data within an application. It helps developers deal with the troubles caused by the dynamic nature of an application's operation environment. A web application is opened to all users. Therefore, its working environment is not always the same and can be affected by a simple user interaction. Moreover, database configuration is not the only data that a web application needs. A web application also has to provide the rich and friendly graphical user interface for any user that interacts with it. This requirement requires the application to support wide variety of languages and change the user interface to match users' cultural characteristics. These internationalization requirements require a flexible mechanism to keep the application operating properly and developers from dealing with unnecessary details of graphical presentation in developing process. Zend\_Config make sure that applications are flexible to change database configuration in case of a server failure with least amount of coding effort or to have a new theme for web pages without having to modify each single page. It can also boost the application deployment by using a staging hierarchy. This hierarchy allows developers to switch from developing to production environment with the same configuration set.

Zend\_Config supports three configuration file formats. We can use a native PHP INI file, a XML or a simple PHP array for providing configuration data. This is an example of PHP INI configuration file for an application. This configuration file includes three sections correspond to three operational stage of the application. All three stages share the same

database driver and database name. The application, however, uses different username and password as a developer change the operational stage.

```
[production]
database.adapter          = "PDO_MYSQL"
database.params.dbname   = "advise-4"
database.params.username = "root"
database.params.hostname = "localhost"

[development : production]
;; Do not use "root" username for connecting to database
database.params.username = "developer"
database.params.password = "develop"

[testing : production]
database.params.username = "tester"
database.params.password = "test"
```

Listing A.13 shows the example of using `Zend_Config` object to retrieve configuration data. Line 2 to 3 defines a global variable which sets the operational stage to *'development'*. Then, we use this global variable to tell the `Zend_Config` object to read the development section of the above configuration file in line 6 to 8. Finally, line 11 to 14 demonstrates how we access the value of each configuration key in the above configuration file.

**Listing A.13.** *Example of using Zend\_Config object*

```
1 // Specify the operational stage
2 defined('APPLICATION_ENVIRONMENT')
3     or define('APPLICATION_ENVIRONMENT', 'development');
4 // Instantiate Zend_Config object
5 $configuration = new Zend_Config_Ini(
6     APPLICATION_PATH . '/config/app.ini',
7     APPLICATION_ENVIRONMENT
8 );
9 // Retrieve configuration data
10 $dbName = $configuration->database->params->dbname;
11 $dbLoginName = $configuration->database->params->username;
12 $dbPassword = $configuration->database->params->password;
13 $dbHostName = $configuration->database->params->hostname;
14 $dbAdapterName = $configuration->database->adapter;
```

## A.2.2. Application Security with Zend Framework

Since assisting web developers to rapid deliver consistent, friendly, and rich feature web applications is the ultimate goal of Zend Framework, Zend Framework also provides tools for programmers on server-side to secure applications from common type web attacks. One basic step to protect applications from common web attacks is properly validating and filtering users input before using them. For this purpose, `Zend_Validate` and `Zend_Filter` are good developers' companions. `Zend_Db_Adapter`'s `quoteInto` is a recommended method to prevent SQL injection. Zend Framework also equips programmers with `Zend_Acl` and `Zend_Auth` for implementing security using software mechanism.

### *Zend\_Validate*

`Zend_Validate` is stocked with a set of common needed validation functions. Developers can use single validator or building a chain of multiple validators to apply on single user input on the order specified by users. Listing A.14 illustrates the use of `Zend_Validate` class as a single validator. Listing A.15 is the validator chain definition which includes multiple rules.

**Listing A.14.** *Example of using Zend\_Validate object*

```
protected function _isValidRoleAndUserName()
{
    $roleValidator = new Zend_Validate_InArray($this->_validRoles);
    return (!empty($this->_currentRole) and !empty($this->_currentUserName)
        and $roleValidator->isValid($this->_currentRole));
}
```

**Listing A.15.** *Example of using Zend\_Filter object*

```
private function getValidators($minLength, $maxLength)
{
    $validatorArray = array(array('Alnum', true), array('NotEmpty', true),
        array('StringLength', true, $minLength, $maxLength));
    return $validatorArray;
}
```

## *Zend\_Filter*

`Zend_Filter` removes unwanted characters from untrusted user input. Similar to `Zend_Validate`, `Zend_Filter` supports both single use and chaining modes. The filter array in Listing A.16 removes whitespaces, tags, new line characters and HTML entities from the input.

**Listing A.16.** *Example of using `Zend_Config` object*

```
private function getFilters()
{
    $filterArray = array('StringTrim', 'StripTags', 'HTMLEntities',
                        'StripNewlines');
    return $filterArray;
}
```

## *Zend\_Auth*

`Zend_Auth` is used to establish identity based on a set of predefined credentials for common scenarios. Currently, `Zend_Auth` supports database table, digest, HTTP, LDAP, and Open ID authentication. Listing A.17 is an illustration of database table authentication.

**Listing A.17.** *Example of using `Zend_Auth` object*

```
// Create authentication adapter
$authAdapter = new Zend_Auth_Adapter_DbTable($dbAdapter);

// Set authentication table information
$authAdapter->setTableName($arrayTable[$pRoleName][0])
->setIdentityColumn($arrayTable[$pRoleName][1])
->setCredentialColumn($arrayTable[$pRoleName][2]);

// Set authentication information to check
$authAdapter->setIdentity($pUsername);
$authAdapter->setCredential($pPassword);

// Authenticate user information
$authResult = $authAdapter->authenticate();
if ($authResult->isValid()){
    return TRUE;
}else{
    return FALSE;
}
```

## *Zend\_Acl*

`Zend_Acl` is a lightweight and flexible access control list to deploy a privileges management system. Developers has full control on defining roles, resources and authorization with the use of classes of `Zend_Acl`. To use this control access list, users first defines roles and resources. Then, this information needs to be register with a `Zend_Acl` object. After everything is setup, programmers call the `isValid()` method of `Zend_Acl` to test the authorization. `Zend_Acl` also supports inheritance between roles. Developers inform `Zend_Acl` about this by providing the parent role for a new role in `addRole()` method. Listing A.18 defines the *'advisor'* role, *'Advisee List'* resource and allows the *'advisor'* role to view *'Advisee List'*.

**Listing A.18.** *Example of using Zend\_Acl*

```
defined('RESOURCE_ADVISEE_LIST')
    or define('RESOURCE_ADVISEE_LIST', 'Advisee List');

// Access control list definition
$zendAcl = new Zend_Acl();

// Setup role definition
$advisorRole = new Zend_Acl_Role('advisor');
$zendAcl->addRole($advisorRole)

// Setup resource definition
$zendAcl->add(new Zend_Acl_Resource(RESOURCE_ADVISEE_LIST))

// Allow an advisor to view advisee list
$zendAcl->allow('advisor', RESOURCE_ADVISEE_LIST, RESOURCE_CONTROL_VIEW);
```



# Bibliography

- [1] Joan F. Marques. Hitting and Missing the Jackpot: The NACADA 2005 National Conference. *The Mentor: An Academic Advising Journal*, March 2006.
- [2] O. Marques, Xundong Ding, and S. Hsu. Design and development of a web-based academic advising system. *Frontiers in Education Conference, 2001. 31st Annual*, 3:S3C-6-10 vol.3, 2001.
- [3] R. J. Multari. Integrating Technology into Advisement Services. *The Mentor: An Academic Advising Journal*, May 2004.
- [4] redLANTERN. History. redLANTERN, 2009. online at <http://www.redlanternu.com/content.jsp?articleId=80>.
- [5] Ohio University. DARS Online. Ohio University, 2006. Online at <http://www.ohio.edu/registrar/darsonline.cfm>.
- [6] University of San Diego. USD: DARS. University of San Diego, 2006. Online at <http://www.sandiego.edu/dars/>.
- [7] University of Missouri St. Louis. UMSL DARS - Degree Audit Reporting System. University of Missouri - St. Louis. Online at <http://www.umsl.edu/services/dars/>.
- [8] RedRock Software Corporation. AdvisorTrac. RedRock Software Corporation. Online at <http://www.advisortrac.net/>.
- [9] University of Louisville. Undergraduate Academic Advising. University of Louisville. Online at <http://louisville.edu/advising/advisortrac/schedule-an-appointment-with-advisortrac.html>.
- [10] Western Kentucky University. Academic Advising and Retention Center. University of Louisville, 2008. Online at <http://www.wku.edu/advising/index.php?page=advisortrac>.
- [11] Indiana University-Purdue University Fort Wayne. Advising and Academics. Indiana University-Purdue University Fort Wayne. Online at <http://www.ipfw.edu/as/advising/advisortrac.shtml>.
- [12] redLANTERN. About Us. redLANTERN, 2003. Online at <http://www.redlanternu.com/about>.
- [13] Rosemary Pleva Flynn. OneStart/EDEN- A Description of IU's Transaction Processing/Recordkeeping Environment. *Electronic Records Project Archivist*, September 2001.

- [14] Office of the Registrar Bloomington Campus. Exploring Innovative Methods of Student Service Delivery, 1998. [http://www.wiche.edu/telecom/Resources/publications/guide1003/guide/{82%C44D64-CA7A-11D3-9309-005004AD2ACC}\\_2200\\_1784.htm](http://www.wiche.edu/telecom/Resources/publications/guide1003/guide/{82%C44D64-CA7A-11D3-9309-005004AD2ACC}_2200_1784.htm).
- [15] CSCI C442/A510 Database System, 2007. Online at <http://mypage.iusb.edu/~hhakimza/IU-ADVISE/>.
- [16] Jesse James Garrett. Ajax: A New Approach to Web Applications. Online at <http://www.adaptivepath.com/ideas/essays/archives/000385.php>.
- [17] Apache Software Foundation. Apache HTTP Server Project. Online at [http://httpd.apache.org/ABOUT\\_APACHE.html](http://httpd.apache.org/ABOUT_APACHE.html).
- [18] Wikipedia. Robert McCool. Online at [http://en.wikipedia.org/wiki/Robert\\_McCool](http://en.wikipedia.org/wiki/Robert_McCool).
- [19] Mohammed J. Kabir. *Apache Server 2 Bible*. Hungry Minds, Inc, 909 Third Avenue, New York, NY 10022, 2002.
- [20] Netcraft. April 2009 Web Server Survey. Online at [http://news.netcraft.com/archives/web\\_server\\_survey.html](http://news.netcraft.com/archives/web_server_survey.html).
- [21] PHP.net. What Can PHP Do. Online at <http://us2.php.net/manual/en/intro-whatcando.php>.
- [22] Tiobe Software The Coding Standard Company. TIOBE Programming Community Index for May 2009. Online at <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>.
- [23] PHP.net. SPL Introduction. Online at <http://us2.php.net/manual/en/ref.spl.php>.
- [24] MySQL. MySQL Downloads. Online at <http://dev.mysql.com/downloads/>.
- [25] W. Jason Gilmore. *Beginning PHP and MySQL: From Novice to Professional, Third Edition*. Apress, 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705, third edition, 2008.
- [26] Steve Burbeck. Applications Programming in Smalltalk-80(TM):How to use Model-View-Controller (MVC). Online at <http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html>.
- [27] Sun Microsystems. Java BluePrints Model-View-Controller. Java BluePrints. Online at <http://java.sun.com/blueprints/patterns/MVC-detailed.html>.
- [28] Microsoft. Model-View-Controller. MSDN. Online at <http://msdn.microsoft.com/en-us/library/ms978748.aspx>.

- [29] Kevin McArthur. *Pro PHP: Patterns, Frameworks, Testing and more*. Apress, 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705, 2008.
- [30] Doug Purdy. Exploring the Factory Design Pattern. Online at [http://msdn.microsoft.com/en-us/library/ms954600.aspx#factopattern\\_topic1](http://msdn.microsoft.com/en-us/library/ms954600.aspx#factopattern_topic1).