

Detail Design

IU Physics Gradebook System

Yu Feng¹

Rohit Chandran²

Naga Rekha Malae³

¹Physics Department, Indiana University, Bloomington IN 47401

²Department of Computer Science, Indiana University, Bloomington IN 47401

³Department of Computer Science, Indiana University, Bloomington IN 47401

Executive Summary

This document covers certain technical aspects such as the architecture and database design of the IU Physics Gradebook system in significant detail.

The architecture of the proposed system is described by means of a structural system model, a control model and a subsystem decomposition model. In addition, modules to be used for the system and the interaction between the modules are explained. These modules are derived from the Data Flow Diagrams specified in the Requirements Specification Document. These include the template library, user interface generators, servlets, error handling libraries and the core libraries. Symfony framework is adopted to organize the project.

The ER diagram specified in the Requirements Specification Document is translated into an equivalent relational database schema. This schema identifies the fields for each of the tables along with the relationships amongst these tables. For the purpose of identifying each row in a table uniquely, a primary key is also specified for each table. The system will make use of advantages of the database platform for data consistency and automatic crash recovering.

The user interface of the proposed system is described in order to meet the client's functionality requirements. The operations performed on each screen and the navigation path between these screens is also explained. Certain common appearance and layout characteristics are mentioned.

The detailed test plan for the system has been proposed. The plan spans from testing operations to how these operations will be done.

Coding specifications for module decomposition rules, module header formats, in-line commenting and indentation are standardized. These coding standards are required by the QA plan to guarantee the quality of the information system.

Exceptions that may occur are mentioned and how they are handled by the system is also explained.

Additional and modified requirements of the client have been appended at the end of the document.

Contents

1	Introduction	5
2	Architecture	6
2.1	Architecture	6
2.1.1	Structural System Model	6
2.1.2	Control Flow Architecture	7
2.1.3	Component Architecture	8
3	Modules	10
3.1	Backend	10
3.1.1	Auth (Authentication)	10
3.1.2	Roster	11
3.1.3	Symfony generated modules	12
3.2	Frontend	13
3.2.1	Welcome	13
3.2.2	Auth (Authentication)	13
3.2.3	Gateway	14
3.2.4	Grade	15
3.2.5	Offering	17
3.2.6	Assignment	17
3.2.7	Final Grade	18
4	Data Design	20
4.1	Introduction	20
4.2	COURSE Table	20
4.3	COURSE_DEFAULT_COMPONENTS Table	20
4.4	PERSON Table	21
4.5	SEMESTER Table	22
4.6	OFFERING Table	22
4.7	COMPONENT Table	23
4.8	SECTION Table	24
4.9	ASSIGNMENT Table	24
4.10	GRADE Table	25
4.11	ENROLL_OFFERING Table	26
4.12	ENROLL_COMPONENT Table	27
4.13	TEACH_SECTION Table	28
4.14	JOIN_SECTION Table	28
4.15	LATEST_GRADE Table	29
4.16	COMPONENT_TYPE Table	30
4.17	ROLE_TYPE Table	30
4.18	ASSIGNMENT_TYPE Table	31

5	User Interface	32
5.1	Common Interfaces	34
5.1.1	Welcome	34
5.1.2	Non-CAS Login	35
5.1.3	CAS Login	36
5.2	Frontend	37
5.2.1	Add Assignment	37
5.2.2	Assignment List	38
5.2.3	Import Grades	39
5.2.4	Import Component Grades	39
5.2.5	Modify Grades	40
5.2.6	View Offering Grades	41
5.2.7	View Component Grades	41
5.2.8	View Section Grades	41
5.2.9	Frontend Gateway	42
5.3	Backend	43
6	Exception Handling	44
6.1	Platform Failures	44
6.2	Runtime Exceptions	44
7	Test Plan	46
7.1	Scope of Testing	46
7.2	Test Plan overview	46
7.3	Test Procedure	46
7.3.1	Unit Testing	46
7.3.2	Functional Testing	49
7.3.3	Usability Testing	50
7.3.4	Installation testing	51
7.3.5	Beta Testing	51
7.3.6	Performance Testing	51
	Index	57
	Appendices	61
A	List of Requirements Changes	62
B	Introduction to Symfony	63
C	Links to Requirements Specification	64
D	Coding Standards and Conventions	66
D.1	Introduction	66
D.2	Modules	66
D.3	Module Decomposition	66
D.4	Headers	66

D.4.1	PHP Headers	66
D.4.2	JavaScript Headers	66
D.4.3	CSS Headers	67
D.5	Commenting and Indentation	67
D.5.1	File Header	67
D.5.2	Member Function Header	67
D.5.3	In-line Commenting	67
D.5.4	Indentation Rules	68
D.5.5	Line Breaking Rules	68
D.5.6	Blank/Space Rules	68
D.6	Naming Conventions	69
D.7	Database Schema	69
D.8	File Naming Conventions	69
D.9	Local Variables	69
D.10	Global Variables	69
D.11	User Interface	70
D.12	Form Layout	70
D.13	Form Navigation	70
D.14	Informative Messages	70
D.15	Key Bindings	70

1 Introduction

The IU Physics Gradebook system is being developed in order to replace OnCourse for the purpose of storing and managing grades more efficiently.

As part of this effort, this document highlights and elaborates on certain design details of the proposed system.

Section 2 details the architecture design of the proposed system. This is done by means of a structural system model, which is explained in terms of the services provided by it, a control flow model and also a component architecture. The architecture design follows the guidelines of the Symfony framework. Details of these modules along with their interaction are specified.

The ER diagram specified in the requirements specification document is translated into an equivalent relational database schema in Section 4. This schema identifies the fields for each of the tables along with the relationships between these tables. For the purpose of identifying each row in a table uniquely, a primary key is also specified for each table. Foreign key constraints are also specified.

The user interface of the proposed system that meets the client's functionality requirements is brought to fore with the help of details of operations performed on each screen. In addition, navigation between these screens along with certain appearance and layout characteristics are mentioned in section 5.

Exceptions that may occur are mentioned and how they are handled by the system is also explained in detail with error handling modules in section 6.

Section 7 has the test plan for the system which mentions the different testing operations to be carried out and how these will be done.

Coding specifications for module decomposition rules, module header formats, in-line commenting and indentation are standardized in Appendix D. These coding standards, as required in the Quality Assurance plan in previous documents, provide a quality information system to the client.

Additional and modified requirements of the client have been added to the end of the document as the Appendix A. An introduction on the relation between the Symfony framework and IU Physics Gradebook system is given in Appendix B. Link to the Requirements Specification document has been added at the end of the document as Appendix C.

2 Architecture

2.1 Architecture

A structural system model, a control model and a sub-system decomposition model are designed to accomplish the overall architecture design of the system.

2.1.1 Structural System Model

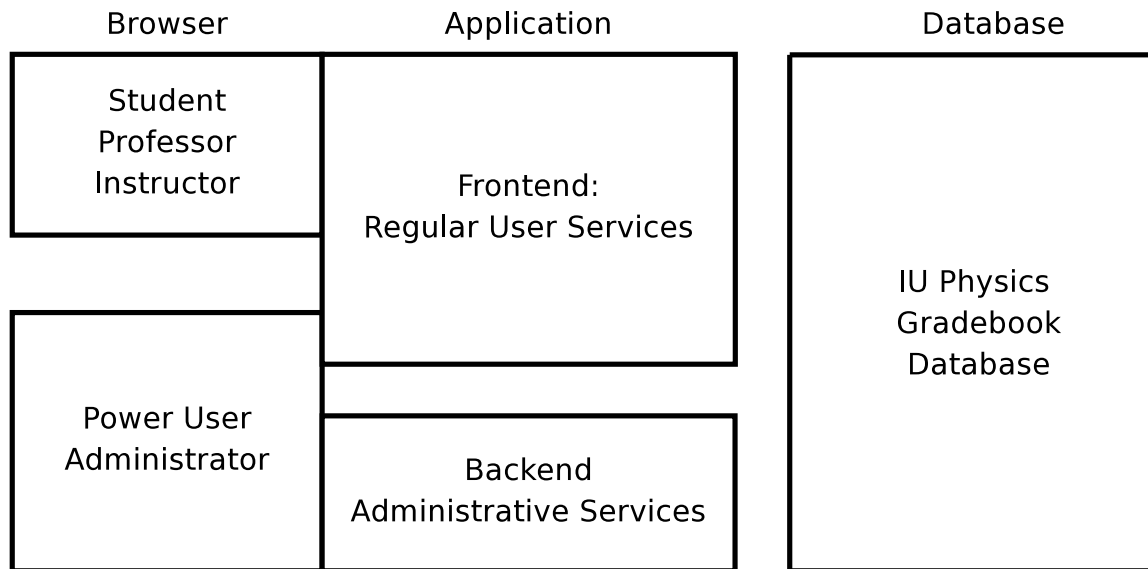


Figure 1: Architecture of the Services

In the above figure the common boundary stands for the interaction between users and the functionalities of the system.

From the user's perspective, the system is composed of two major services:

- Regular user services are focused towards the students, professors and instructors. The system provides services to enable
 - Viewing grades
 - Assigning grades
 - Manipulating assignments
- Administration services are focused towards the departmental staff. The system provides services to enable
 - Managing course hierarchy
 - Managing roster
 - Batch import/export of grades

2.1.2 Control Flow Architecture

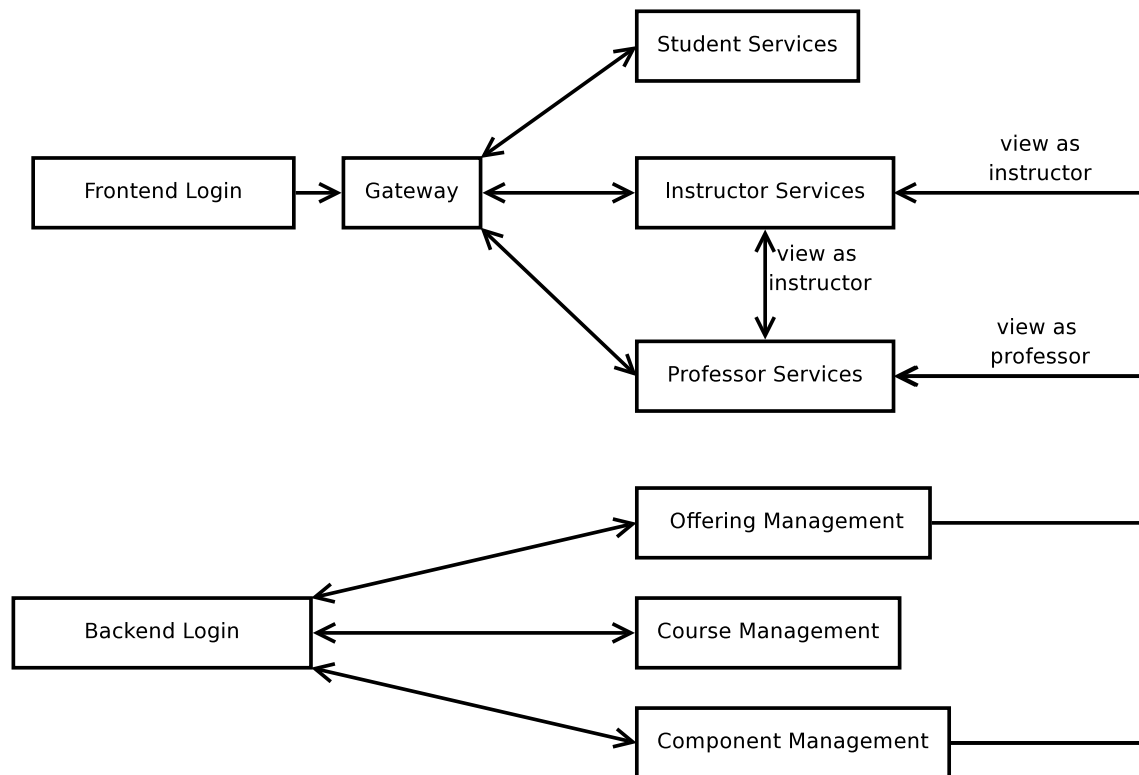


Figure 2: Architecture of the Control Flow

The control flow models the transition of the active service in the system for each interactive session.

2.1.3 Component Architecture

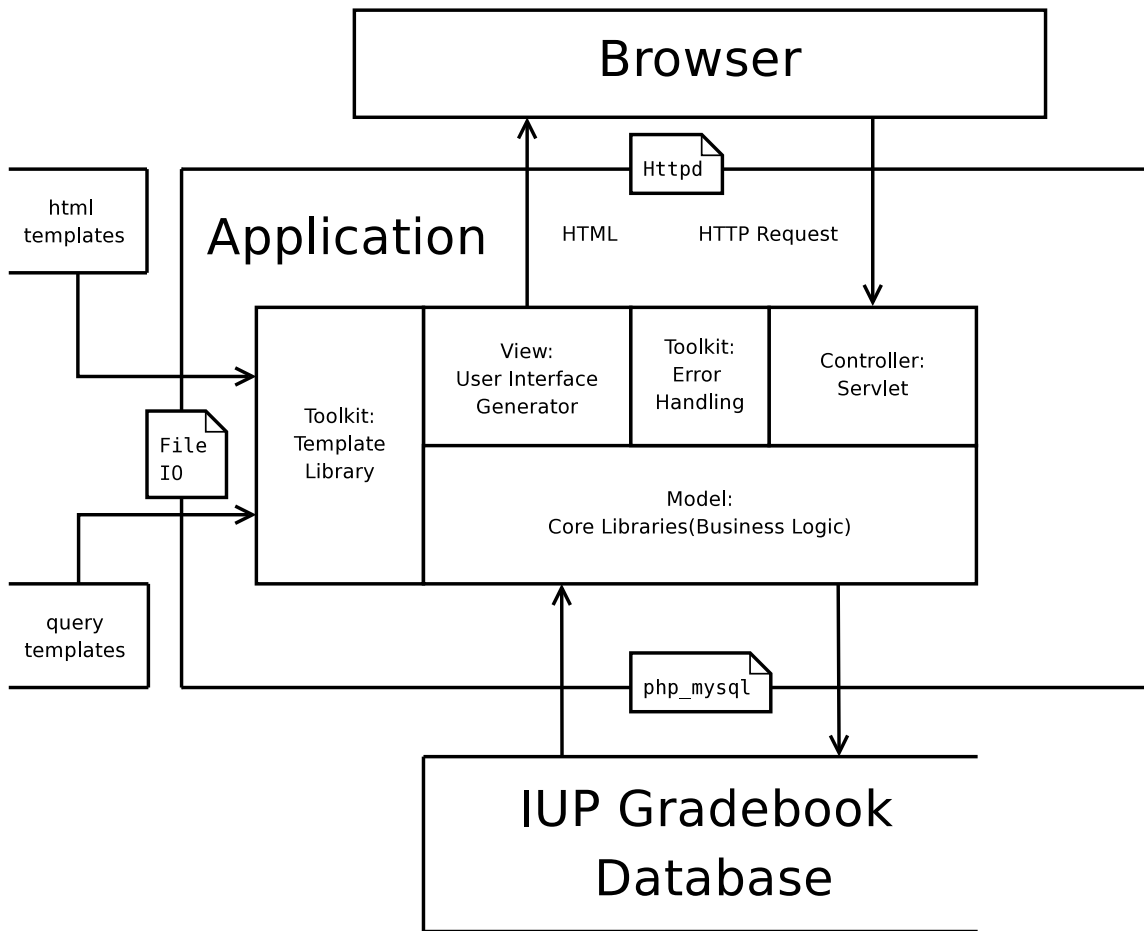


Figure 3: Architecture of the Components. The system is composed of three components which are shown in this figure.

Browser The web browser on the user's workstation directly interacts with the user. It must be noted that the development team has no control over the user's choice of browser. The team assumes the user of the system will use Mozilla Firefox(Gecko), Safari(WebKit), or Internet Explorer (Trident).

Database The DBMS system which provides fundamental storage services.

Applications The applications contain the business logic. They are responsible for resolving the requests from the browser, checking the validity of the requests, querying the database, and pushing the user interface to the browser.

There are two applications, the frontend and the backend, corresponding the Regular User Services and Administration Services, respectively. Five relatively independent modules:

- Template Library, a utility library that builds SQL queries or HTML pages from template files.

- User Interface Generators, server side scripts that generate the HTML files by merging the data obtained from the database and the page templates from the templates. (View)
- Servlets, server side scripts that respond to the HTTP requests from the client browser but do not directly push any content to the browser. (Controller)
- Error Handling library, the platform to support the error handling policies.
- Core Libraries (Model)
 - Session Management, an extension to the PHP session management library.
 - User and Permissions, a collection of functions which are used for user validation and user permission authorization.
 - Data Manipulation, a collection of data manipulation routines.

The architecture design follows the Model-View-Controller pattern and maps to the framework provided by the Symfony Project¹. In the following section the modules are designed according to the this mapping.

¹Refer to Appendix B: Symfony.

3 Modules

The system is implemented into two applications, the Backend application and the Frontend application.

The Backend application provides the administrative services. The user of the Backend application is primarily the Primary Client[2]. The Backend application is generated by the Object-Relation-Mapping (ORM) tool. The generated applications are also tweaked to be consistent with the general user interface rules of the entire system.

The Frontend application provides the regular user services. Most of other users interacts with the system via the Frontend application.

For each module, the requirement from which the module is derived is also listed. Please refer to the Functionality Requirement section in Requirement Specification[2].

3.1 Backend

3.1.1 Auth (Authentication)

NAME: auth

DERIVED FROM: None

DESCRIPTION: The purpose of this module is to authenticate users and assign privileges.

CALLS: None

CALLED BY: Symfony security framework

PRECONDITION: None

POSTCONDITION: None

MODEL(S) USED: Person

ASSOCIATED FORM(S): IUPLoginForm

ACTION(S): Login, Logout

Login

FORM: IUPLoginForm

BUTTON(S): Submit

FIELD(S): username, password

PSEUDOCODE DESCRIPTION:

```

    if !request.isPost()
        form = new IUPLoginForm();
        return;
    endif;
    if Person.select(username, password)
        user.setAuthenticated(true);
        forward('default');
    endif;
    error("Wrong username/password");

```

Logout

FORM: None

BUTTON(S): None

FIELD(S): None

PSEUDOCODE DESCRIPTION:

```
user.setAuthenticated(false);  
forward('default');
```

3.1.2 Roster

NAME: roster

DERIVED FROM: Req Spec 2.(d).xxii

DESCRIPTION: The purpose of this module is to support the importing and exporing of rosters.

CALLS: None

CALLED BY: Directly by the user

PRECONDITION: None

POSTCONDITION: None

MODEL(S) USED: Person, Semester

ASSOCIATED FORM(S): RosterImportForm, RosterExportForm

ACTION(S): Import, Export

Import

FORM: RosterImportForm

BUTTON(S): Submit

FIELD(S): None

FILES: roster

PSEUDOCODE DESCRIPTION:

```
if !request.isPost()  
    form = new RosterImportForm();  
    return;  
endif;  
foreach line in roster;  
    FIELD(S) = csv_split(line);  
    person = Person.select(cas_account = FIELD(S)[0])  
    if person is null  
        person = Person.create(FIELD(S)[0], FIELD(S)[1], FIELD(S)[2])  
    endif;
```

```

        foreach sn in FIELD(S) since 3
            section = Section.select(section_number = sn);
            section.enroll(person);
        endif;
    endif;

```

Export

FORM: RosterExportForm

BUTTON(S): Submit

FIELD(S): filename, semester

PSEUDOCODE DESCRIPTION:

```

    if !request.isPost()
        form = new RosterExportForm();
        return;
    endif;
    offering = Offering.select(semester = semester);
    foreach person in offering.students
        write(person.cas_account, person.first_name, person.last_name);
        foreach section in offering.sections
            if(section.students.has(person)) write section.section_number;
            else write ' ';
        endfor
    endfor

```

3.1.3 Symfony generated modules

NAMES: person, semester, course, offering, section, component, assignment, grade

DERIVED FROM: Req Spec 1, 2.(d)

DESCRIPTION: The module provides the interface for manipulating the information model.

CALLS: None

CALLED BY: Directly by the user

PRECONDITION: None

POSTCONDITION: None

MODEL(S) USED: Person, Semester, Course, Offering, Section, Component, Assignment, Grade

ASSOCIATED FORM(S): For each module, 3 forms for 'Edit', 'Filter', 'New' are generated.

ACTION(S): Index, Filter, New, Create, Edit, Update, Delete, Batch, BatchDelete

Index: List view action

Filter: Updates the filters

New: New view action

Create: Create a new entry
Edit: Edit view action
Update: Updates an entry
Delete: Deletes an entry
Batch: Executes a batch entry
BatchDelete: Executes a batch delete action

NOTES: These modules are generated by symfony based on the relation design. Refer to http://www.symfony-project.org/jobee/1_2/Doctrine/en/ for a more detailed description.

3.2 Frontend

3.2.1 Welcome

NAME: welcome

DERIVED FROM: None

DESCRIPTION: This module presents the home page of the frontend application to the user.

CALLS: None

CALLED BY: None

PRECONDITION: None

POSTCONDITION: None

MODEL(S) USED: None

ASSOCIATED SCREEN(S): Welcome

ASSOCIATED FORM(S): None

ACTION(S): Index

Index: Show the welcome page.

PSEUDOCODE DESCRIPTION: None

3.2.2 Auth (Authentication)

NAME: auth

DERIVED FROM: None

DESCRIPTION: This module implements the mechanisms for authenticating users in the frontend application.

CALLS: None

CALLED BY: None

PRECONDITION: None

POSTCONDITION: None

MODEL(S) USED: Person

ASSOCIATED SCREEN(S): IUP Login, CAS Login

ASSOCIATED FORM(S): IUPLoginForm

ACTION(S): Login, Logout, CASCallback

Login: The same as backend.

Logout: The same as backend.

CASCallback:

FORM: None

PSEUDOCODE DESCRIPTION:

```
result = verify_the_request
cas_account = result.account
if(cas_account is a Person)
    user.setAuthencated(true);
```

3.2.3 Gateway

NAME: gateway

DERIVED FROM: None

DESCRIPTION: This module implements the user interface to lead the user to their tasks.

CALLS: Grade, Assignment, Final Grade

CALLED BY: Directly by the user

PRECONDITION: User should be authenticated

POSTCONDITION: None

MODEL(S) USED: Grade, Assignment, Person

ASSOCIATED SCREEN(S): Gateway

ASSOCIATED FORM(S): None

ACTION(S): Index

Index

FORM(S): None

PSEUDOCODE DESCRIPTION:

```
person = Person.select(user.person_id);
offerings = person.offerings;
teachingOfferings = person.teachingOfferings;
teachingSections = person.teachingSections;
build_template(teachingOfferings, teachingSections, offerings);
```

3.2.4 Grade

NAME: grade

DERIVED FROM: Req Spec 2.(a).i and 2.(b).i

DESCRIPTION: This module is a collection of all functionalities to manipulate the grades.

CALLS: Assignment

CALLED BY: None

PRECONDITION: None

POSTCONDITION: None

MODEL(S) USED: Grade, Assignment, Person

ASSOCIATED SCREEN(S): Modify Grades

ASSOCIATED FORM(S): GradeEntryForm, AssignmentFilterForm

ACTION(S): Index, Filter, Export, Import

Index

FORM(S): AssignmentFilterForm, GradeEntryForm

SCREEN(S): GradeEntryScreen

FIELD(S): grades, assignments

BUTTON(S): Submit

PSEUDOCODE DESCRIPTION:

```

if !request.isPost() then
  foreach grade in grades
    Grade.save();
  endfor
else
  if request.type == 'offering' then
    students = request.offering.students;
    assignments = request.offering.assignments
  else
    students = request.section.students;
    assignments = request.section.component.assignments
  endif;
  form[grades] = new GradeEntryForm(students, null);
  form[filter] = new AssignmentFilterForm(assignments);
endif

```

Filter

FORM(S): AssignmentFilterForm, GradeEntryForm

SCREEN(S): GradeEntryScreen

FIELD(S): assignments

BUTTON(S): Filter

PSEUDOCODE DESCRIPTION:

```

if request.isPost() then
  if request.type == 'offering' then
    students = request.offering.students;
    assignments = request.offering.assignments
  else
    students = request.section.students;
    assignments = request.section.component.assignments
  endif;
  form[filter] = new AssignmentFilterForm(assignments);
  form[filter].bind(request);
  form[grades] = new GradeEntryForm(students,
                                     form[filter].assignments);
  form[grades].bind(request);
endif

```

Export

FORM(S): AssignmentFilterForm

SCREEN(S): GradeEntryScreen

FIELD(S): assignments

BUTTON(S): Export

PSEUDOCODE DESCRIPTION:

```

if request.isPost() then
  form = new AssignmentFilterForm();
  form.bind(request);
  if request.type == 'offering' then
    students = request.offering.students;
  else
    students = request.section.students;
  endif;
  foreach student in students
    write student.cas_account, student.last_name, student.first_name;
    foreach assignment in form.assignment
      write student.grades[assignment]
    endfor
  endfor
endif;

```

Import

FORM(S): FileChooserForm

SCREEN(S): GradeEntryScreen

FIELD(S): file

BUTTON(S): Import

PSEUDOCODE DESCRIPTION:

```

assignments = find_assignments(file.lines[0]);
foreach line in file.lines since 1
    student = Student.find(line.field[0]);
    foreach field in line.FIELD(S) since 3
        Grade.insert(student, assignments[field]);
    endfor
endfor

```

3.2.5 Offering

NAME: Offering

DERIVED FROM: None

DESCRIPTION: This module guides the Professor to Assignment module and Grade Module.

CALLS: Grade, Assignment, Final Grade

CALLED BY: Directly by the user

PRECONDITION: User should be authenticated

POSTCONDITION: None

MODEL(S) USED: Assignment, Person, Offering

ASSOCIATED SCREEN(S): Offering

ASSOCIATED FORM(S): None

ACTION(S): Index

Index

FORM: None

PSUEDOCODE DESCRIPTION:

```

this->$offering = Offering.query(id);
this->$components = $offering.components;
this->$sections = $offerings.sections;
chain_up_to_parent();

```

3.2.6 Assignment

NAME: assignment

DERIVED FROM: Req Spec 2.(b).ii, 2.(b).iii, 2.(b).iv

DESCRIPTION: This module implements a user interface that manipulates the assignments.

CALLS: Grade, Assignment, Final Grade
 CALLED BY: Directly by the user
 PRECONDITION: User should be authenticated
 POSTCONDITION: None
 MODEL(S) USED: Assignment, Person
 ASSOCIATED SCREEN(S): Assignment
 ASSOCIATED FORM(S): AssignmentForm
 NOTES: This module is generated by symfony.

3.2.7 Final Grade

NAME: FinalGrade
 DERIVED FROM: Req Spec 2.(d).xxiv, 2.(d).xxv, 2.(d).xx, 2(d).xxi
 DESCRIPTION: This module provides the functionalities for final grade entrying.
 CALLS: Grade, Assignment, Final Grade
 CALLED BY: Directly by the user
 PRECONDITION: User should be authenticated
 POSTCONDITION: None
 MODEL(S) USED: Grade, Assignment, Person
 ASSOCIATED SCREEN(S): Final Grade
 ASSOCIATED FORM(S): FinalGradeEntryForm, FileChooserForm
 ACTION(S): Index, Import, Export

Index

FORM: FinalGradeEntryForm
 FIELD(S): grades
 BUTTON(S): submit
 PSUEDOCODE DESCRIPTION:

```

    if !request.isPost() then
        fetch grades;
        form = new FinalGradeEntryForm(grades);
        return chain_up();
    endif
    grades = request.grades;
    foreach grade in grades
        grade.save();
    endfor;
```

Import

FORM: FileChooserForm
FIELD(S): None
FILES: file
BUTTON(S): Submit
PSUEDOCODE DESCRIPTION:
 if !request.isPost() then
 fetch grades;
 form = new FileChooserForm(grades);
 return chain_up();
 endif
 break_file_into_grades;
 foreach grade in grades
 grade.save();
 endfor;

Export

FORM: None
FIELD(S): None
BUTTON(S): None
PSUEDOCODE DESCRIPTION:
 components = request.components;
 offering = request.offering;
 foreach student in request.students
 write(student.cas_account, student.first_name,
 student.last_name)
 foreach component in components do
 write(student.final_grade[component])
 endfor
 write(student.final_grade[offering])
 endfor;

4 Data Design

4.1 Introduction

A database schema is designed according to the ER model (Refer to Requirement Specification document[2]) captured in the requirement analysis stage.

4.2 COURSE Table

Introduction

The COURSE table contains information regarding the courses which are being offered by the professor.

Fields

Field	Datatype	Comment
course_name	varchar(50)	Name of the course being offered by the professor
course_id	int	Unique integer
course_comment	string	Course description
course_number	varchar(20)	Uniquely identifies each course

Table 1: COURSE Table Fields

Constraints

- course_id is the primary key of the COURSE table.

Keyname	Type	Field
Primary	PRIMARY	course_id

Table 2: COURSE Table Constraints

4.3 COURSE_DEFAULT_COMPONENTS Table

Introduction

The COURSE_DEFAULT_COMPONENTS table contains information regarding default component setup of a Course.

Fields

Field	Datatype	Comment
course_id	int	Unique integer for the course
component_type	string	Type of component

Table 3: COURSE_DEFAULT_COMPONENTS Table Fields

Constraints

- course_id and component_type together acts as a primary key for the COURSE_DEFAULT_COMPONENTS table.
- course_id in COURSE_DEFAULT_COMPONENTS table is a foreign key that references the course_id field in the COURSE table.
- component_type in COURSE_DEFAULT_COMPONENTS table is a foreign key that references the name field in the COMPONENT_TYPE table.

Keyname	Type	Field
primary	PRIMARY	course_id, component_type
course_id	FOREIGN_KEY	course_id
component_type	FOREIGN_KEY	component_type

Table 4: COURSE_DEFAULT_COMPONENTS Table Constraints

4.4 PERSON Table**Introduction**

The PERSON table contains all the details of the users and is used to validate the user during login. The default value for the role field is user. The person_id field is auto incremented by default.

Fields

Field	Datatype	Comment
itaccount	varchar(20)	IT account of the user
iupaccount	varchar(20)	IU Physics account of the user that does not have an IU account (non-IU user)
password	varchar(100)	Password to validate the user
first_name	varchar(20)	First name of the user
last_name	varchar(20)	Last name of the user
middle_name	varchar(20)	Middle name of the user
email	varchar(20)	Email ID of the user
person_id	int	ID internally generated by the database engine to uniquely identify each user
role_type_id	int	Unique integer

Table 5: PERSON Table Fields

Constraints

- person_id in the PERSON table is a primary key.
- role_type_id is a foreign key that references the role_type_id of the ROLE_TYPE table.

- itaccount in the PERSON table is a unique key because no two persons can have the same itaccount.
- iupaccount in the PERSON table is a unique key because no two persons can have the same iupaccount.

Keyname	Type	Field
person_id	PRIMARY	person_id
itaccount	UNIQUE	itaccount
iupaccount	UNIQUE	iupaccount
role_type_id	FOREIGN_KEY	role

Table 6: PERSON Table Constraints

4.5 SEMESTER Table

Introduction

The SEMESTER table contains information regarding the year and the semester in which the courses are being offered.

Fields

Field	Datatype	Comment
semester_id	int	Unique integer for the semester
year	int	Year the course is being offered
season	string	Season the course being offered

Table 7: SEMESTER Table Fields

Constraints

- semester_id is a primary key for SEMESTER table.

Keyname	Type	Field
primary	PRIMARY	semester_id

Table 8: SEMESTER Table Constraints

4.6 OFFERING Table

Introduction

The OFFERING table contains information about the offerings, such as the professor offering the course and the semester in which it is being offered.

Fields

Field	Datatype	Comment
offering_id	int	Unique integer
semester_id	int	Unique integer for the semester
course_id	int	Unique integer for the course
professor_id	int	Unique interger for the person

Table 9: OFFERING Table Fields

Constraints

- offering_id is a primary key for OFFERING table.
- course_id is a foreign key that references the course_id field in the COURSE table.
- semester_id is a foreign key that references the semester_id field in the SEMESTER table.
- professor_id is a foreign key that references the person_id field in the PERSON table.

Keyname	Type	Field
primary	PRIMARY	offering_id
course_id	FOREIGN_KEY	course_id
semester_id	FOREIGN_KEY	semester_id
professor_id	FOREIGN_KEY	person_id

Table 10: OFFERING Table Constraints

4.7 COMPONENT Table**Introduction**

The COMPONENT table contains information regarding the component such as component name, the corresponding course number, who is offering that course and when is it being offered.

Fields

Field	Datatype	Comment
component_id	int	Unique integer for the component
offering_id	int	Unique integer for the offering
component_type_id	string	Type of the component

Table 11: COMPONENT Table Fields

Constraints

- component_id is a primary key for the COMPONENT table.
- offering_id is a foreign key that references offering_id in the OFFERING table

- component_type_id is a foreign key that references the name field in the COMPONENT_TYPE table.

Keyname	Type	Field
primary	PRIMARY	component_id
offering_id	FOREIGN_KEY	offering_id
component_type_id	FOREIGN_KEY	component_type_id

Table 12: COMPONENT Table Constraints

4.8 SECTION Table

Introduction

The SECTION table contains information about a section such as section number, the component to which the section belongs, the course to which the component belongs, which professor is offering the course and in which semester the course is being offered.

Fields

Field	Datatype	Comment
section_id	int	Unique integer for the section
component_id	int	Unique integer for the component
section_number	string	String to name the section

Table 13: SECTION Table Fields

Constraints

- section_id is a primary key for the SECTION table.
- component_id is a foreign key for the SECTION table that references component_id of the COMPONENT table.

Keyname	Type	Field
primary	PRIMARY	section_id
component_id	FOREIGN_KEY	component_id
section_number	UNIQUE	section_number

Table 14: SECTION Table Constraints

4.9 ASSIGNMENT Table

Introduction

The ASSIGNMENT table contains information regarding the assignment such as name and type of assignment, the component to which the assignment belongs, the course to which the component belongs, which professor is offering the course and in which semester is it being offered.

Fields

Field	Datatype	Comment
assignment_id	int	Unique integer for the assignment
assignment_type_id	string	Unique integer
course_id	int	Unique integer for the course

Table 15: ASSIGNMENT Table Fields

Constraints

- assignment_id is a primary key for the ASSIGNMENT table.
- course_id is a foreign key that references the course_id of the COURSE table.
- assignment_type_id is a foreign key that references the name field in the ASSIGNMENT_TYPE table.

Keyname	Type	Field
primary	PRIMARY	assignment_id
course_id	FOREIGN_KEY	course_id
assignment_type_id	FOREIGN_KEY	assignment_type

Table 16: ASSIGNMENT Table Constraints

4.10 GRADE Table**Introduction**

The GRADE table contains information regarding grades assigned to students for different assignments in a section by instructors responsible to that section. The default value of the grade_timestamp field is CURRENT_TIMESTAMP.

Fields

Field	Datatype	Comment
grade_id	bigint	Unique integer for the grade and it is internally generated
teacher_id	int	Unique integer for the teacher
student_id	int	Unique integer for the student
assignment_id	int	Unique integer for the assignment
grade_timestamp	timestamp	Records timestamp of the submission of the grade
grade	string	Grade assigned to a given student for an assignment
grade_comments	text	Comments related to the grade for each student

Table 17: GRADE Table Fields

Constraints

- grade_id is a primary key for the GRADE table.
- assignment_id is a foreign key that references the assignment_id of the ASSIGNMENT table.
- teacher_id is a foreign key that references the person_id field in the PERSON table.
- student_id is a foreign key that references the person_id field in the PERSON table.

Keyname	Type	Field
primary	PRIMARY	grade_id
teacher_id	FOREIGN_KEY	teacher_id
student_id	FOREIGN_KEY	student_id
assignment_id	FOREIGN_KEY	assignment_id

Table 18: ASSIGN_GRADES Table Constraints

4.11 ENROLL_OFFERING Table

Introduction

The ENROLL_OFFERING table contains information regarding the students enrolled in the course along with the course information such as in which semester the course is being offered and which professor is offering that course. This table is also used to check the existence of the student in the course and also holds information regarding the student's final course grade.

Fields

Field	Datatype	Comment
offering_id	int	Unique integer for the offering
student_id	int	Unique interger for the student
offering_final_grade	string	Final component grade assigned to each student
active	boolean	Hides/shows a student from the course offering

Table 19: ENROLL_OFFERING Table Fields

Constraints

- offering_id, student_id fields together acts as a primary key for the ENROLL_OFFERING table.
- offering_id is a foreign key that references the offering_id in the OFFERING table.
- student_id field is a foreign key that references the person_id field of the PERSON table.

Keyname	Type	Field
primary	PRIMARY	offering_id, student_id
offering	FOREIGN_KEY	offering_id
student	FOREIGN_KEY	student_id

Table 20: ENROLL_OFFERING Table Constraints

4.12 ENROLL_COMPONENT Table

Introduction

The ENROLL_COMPONENT table contains information regarding the students enrolled in a component along with the component information such as the course to which the component belongs, in which semester the course is being offered and which professor is offering that course. This table is also used to check the existence of the student in the component and also holds information regarding the student's final grade for the component.

Fields

Field	Datatype	Comment
component_id	int	Unique integer for the component
student_id	int	ID internally generated by the database engine to identify each student uniquely
component_grade	string	Final component grade assigned to each student
active	boolean	Hides/unhides a student from the component

Table 21: ENROLL_COMPONENT Table Fields

Constraints

- component_id, student_id fields together acts as a primary key for the ENROLL_COMPONENT table.
- student_id field is a foreign key that references the person_id field in the PERSON table.
- component_id is a foreign key that references component_id field in the COMPONENT table.

Keyname	Type	Field
primary	PRIMARY	component_id, student_id
component	FOREIGN_KEY	component_id
student	FOREIGN_KEY	student_id

Table 22: ENROLL_COMPONENT Table Constraints

4.13 TEACH_SECTION Table

Introduction

The TEACH_SECTION table contains information regarding the instructor .

Fields

Field	Datatype	Comment
section_id	int	Unique integer for the section
instructor_id	int	Unique integer for the person

Table 23: TEACH_SECTION Table Fields

Constraints

- section_id, instructor_id fields together acts as a primary key for the table TEACH_SECTION.
- section_id is a foreign key which references the section_id in the SECTION table.
- instructor_id field is also a foreign key which references the person_id field of the PERSON table.

Keyname	Type	Field
primary	PRIMARY	section_id, instructor_id
section_id	FOREIGN_KEY	section_id
instructor_id	FOREIGN_KEY	instructor_id

Table 24: TEACH_SECTION Table Constraints

4.14 JOIN_SECTION Table

Introduction

The JOIN_SECTION table contains information regarding the student such as when the student joined the section, the component to which the section belong, the course to which the component belong, which professor is offering the course and in which semester the course is being offered. It also holds information regarding the student withdrawing from the section. The default value for the start_timestamp field is CURRENT_TIMESTAMP.

Fields

Field	Datatype	Comment
section_id	int	Unique integer for the section
student_id	int	Unique integer for the student
end_timestamp	timestamp	Records the date and time that student leaves the section. NULL if student is currently part of the section

Table 25: JOIN_SECTION Table Fields

Constraints

- section_id, student_id fields together act as a primary key for the JOIN_SECTION table.
- section_id is a foreign key that references the section_id in the SECTION table.
- student_id field is a foreign key that references the person_id in the PERSON table.

Keyname	Type	Field
primary	PRIMARY	section_id, student_id
section	FOREIGN_KEY	section_id
student	FOREIGN_KEY	student_id

Table 26: JOIN_SECTION Table Constraints

4.15 LATEST_GRADE Table**Introduction**

The LATEST_GRADE table contains information regarding the latest grades assigned to students for different assignments in different sections for which different instructors are responsible.

Fields

Field	Datatype	Comment
teacher_id	int	Unique integer for the teacher
student_id	int	Unique integer for the student
grade_id	int	Unique integer for the grade
assignment_id	int	Unique integer for the assignment
grade	string	Latest grade assigned to a student for an assignment

Table 27: LATEST_GRADE Table Fields

Constraints

- grade_id is a primary key for the LATEST_GRADE table.
- assignment_id is a foreign key in the LATEST_GRADE table that references assignment_id of the ASSIGNMENT table.
- teacher_id is a foreign key for the LATEST_GRADE table that references the person_id field of the PERSON table.
- student_id is a foreign key for the LATEST_GRADE table that references the person_id field of the PERSON table.

Keyname	Type	Field
primary	PRIMARY	grade_id
teacher_id	FOREIGN_KEY	teacher_id
student_id	FOREIGN_KEY	student_id
assignment_id	FOREIGN_KEY	assignment_id

Table 28: LATEST_GRADE Table Constraints

4.16 COMPONENT_TYPE Table

Introduction

The COMPONENT_TYPE table contains information regarding the type of component such as recitation component, lab component, lecture component etc

Fields

Field	Datatype	Comment
component_type_id	int	Unique integer
name	string	Type of component

Table 29: COMPONENT_TYPE Table Fields

Constraints

Keyname	Type	Field
primary	PRIMARY	component_type_id

Table 30: COMPONENT_TYPE Table Constraints

4.17 ROLE_TYPE Table

Introduction

The ROLE_TYPE table contains information regarding the roles the user can have. The user of the system can only take the roles which are specified in this table.

Fields

Field	Datatype	Comment
role_type_id	int	Unique integer
name	string	Type of user

Table 31: ROLE_TYPE Table Fields

Constraints

- role_type_id is a primary key for the ROLE_TYPE table

Keyname	Type	Field
primary	PRIMARY	role_type_id

Table 32: ROLE_TYPE Table Constraints

4.18 ASSIGNMENT_TYPE Table**Introduction**

The ASSIGNMENT_TYPE table contains information regarding the type of assignment such as clicker questions, home work etc

Fields

Field	Datatype	Comment
assign_type_id	int	Unique integer
name	string	Type of assignment

Table 33: ASSIGNMENT_TYPE Table Fields

Constraints

- assign_type_id is a primary key for the table ASSIGNMENT_TYPE table.

Keyname	Type	Field
primary	PRIMARY	assign_type_id

Table 34: ASSIGNMENT_TYPE Table Constraints

5 User Interface

The user interface of the proposed system is described in terms of what the user must do to use the system.

Figure 4 on page 32, Figure 6 on page 33 and Figure 5 on page 32 show the menu hierarchy for different users in the system. Clearly, we pick an entity first and then an operation for that entity.

The User Interfaces used in the system are of two types:

1. Regular page which is in the form of a regular browser window.
2. Dialog box which is in the form of a pop-up window. When an operation is performed in a dialog box, the user returns to the regular browser window.

The following is the standard layout for all screens:

- The banner of the website is shown at the top of all pages.
- The page heading is always at the top of the screen just below the banner.
- The links for Help and Logout are always on the left hand side of the screen, except in the Welcome and Login pages where only the link for Help is shown.
- Any error message is displayed just below the field for which the error occurs in the page.
- All Submit Buttons are displayed below the text fields.

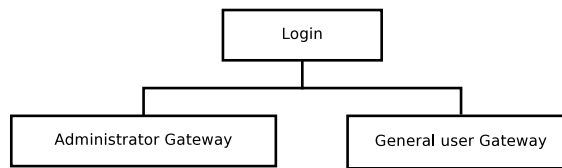


Figure 4: Menu Hierarchy at Login

As shown in this figure, the user is directed to either of the two gateways after login.

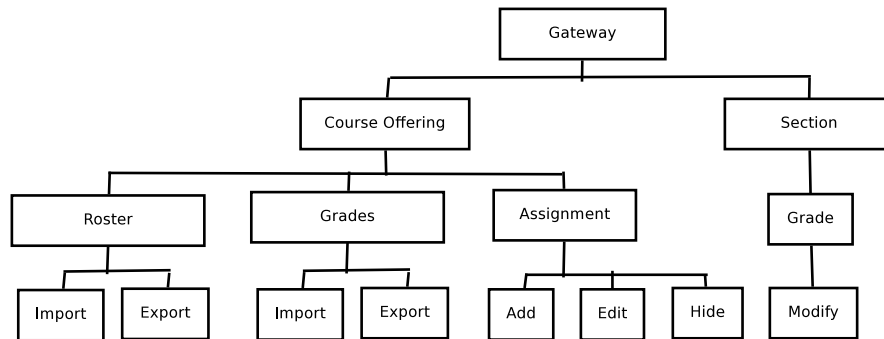


Figure 5: Menu Hierarchy for General Users

As shown in this figure, the forms for General Users are organized first by entity and then by operation.

Here, the entities are Offering, Component and Section. The operations are Edit, Add and Hide.

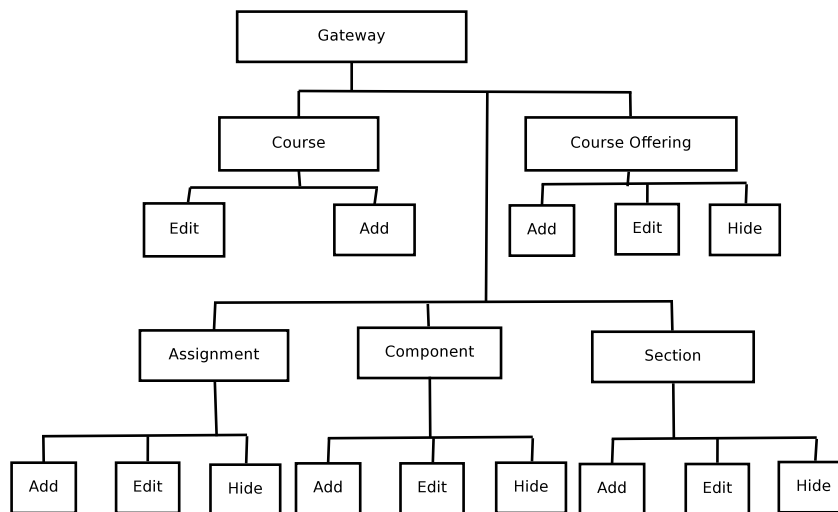


Figure 6: Menu Hierarchy for Administrator

As shown in this figure, the forms for backend are organized first by entity and then by operation. Here, the entities are Course, Course Offering, Component and Section and Assignment. The operations are Add, Edit and Hide.

5.1 Common Interfaces

The following user interfaces are common to all users of the system:

5.1.1 Welcome

This screen is merely a page that asks the user to go the appropriate page for login. The user may go to either the CAS login or the non-CAS login page. This page is generated by the 'Index' action of Welcome module.

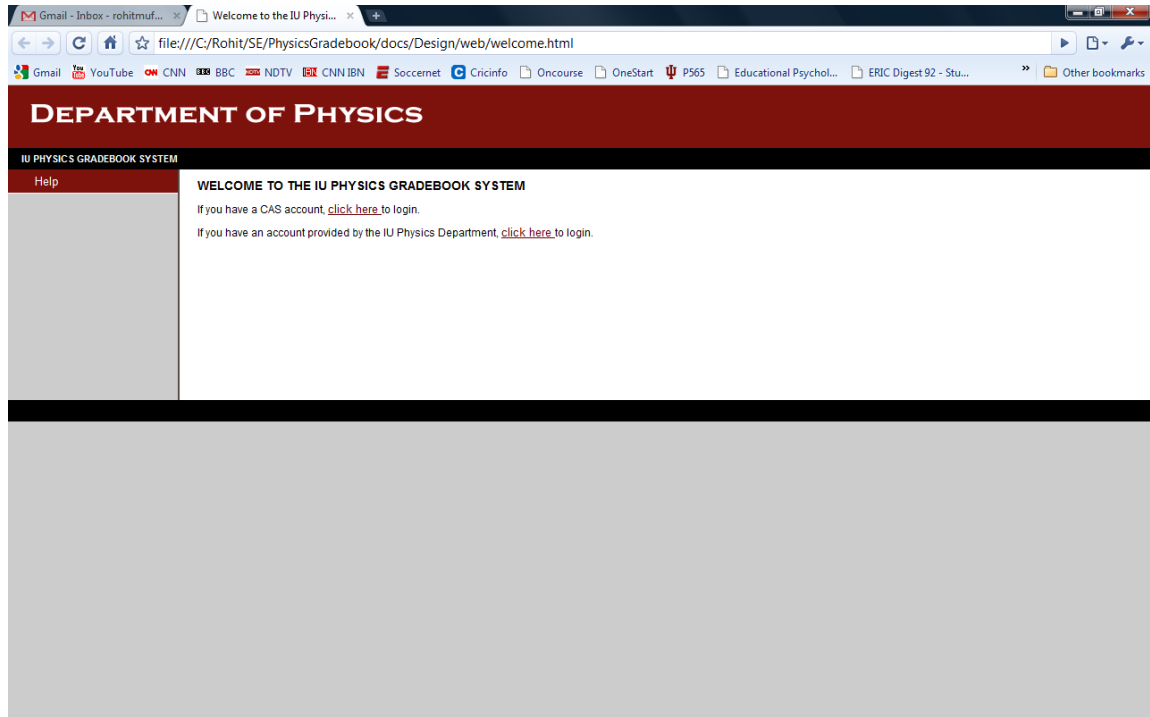


Figure 7: Welcome screen

Operation

A user with a valid IU account clicks on the “click here” link that takes the user to the CAS login screen. If not, the user chooses to go to the non-CAS login page.

Navigation

Hyperlink	Destination
click here	Non-CAS login page/CAS login page

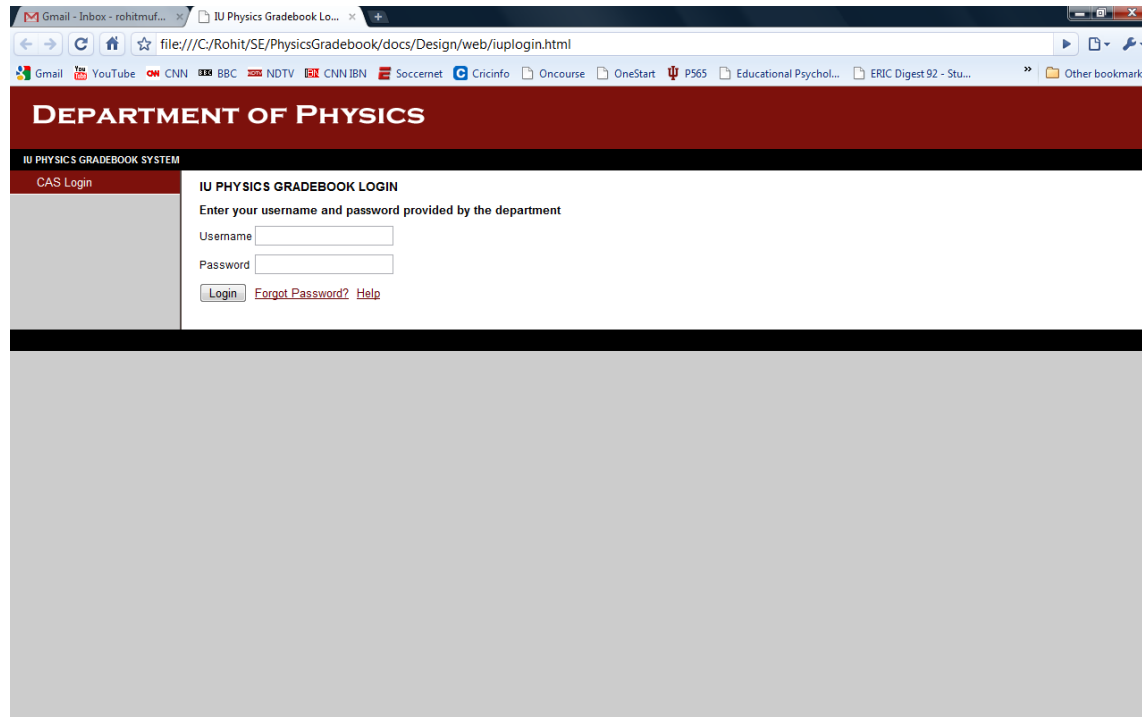
Table 35: Navigation from Welcome screen

Appearance

A Welcome picture is placed on the page immediately below the page heading. But, we do not explicitly show this in this document.

5.1.2 Non-CAS Login

This screen enables a non-CAS user to log in to the system. This page is generated by the 'Index' action of Auth module.



The screenshot shows a web browser window with the address bar displaying `file:///C:/Rohit/SE/PhysicsGradebook/docs/Design/web/iuplogin.html`. The browser's bookmark bar includes links to Gmail, YouTube, CNN, BBC, NDTV, CNN IBN, Socccernet, Cricinfo, Oncourse, OneStart, P565, Educational Psychol..., ERIC Digest 92 - Stu..., and Other bookmarks. The web page has a dark red header with the text "DEPARTMENT OF PHYSICS" in white. Below the header, a black bar contains the text "IU PHYSICS GRADEBOOK SYSTEM" in white. The main content area is white and features a login form. On the left side of the form, there is a vertical bar with the text "CAS Login" in red. The form itself is titled "IU PHYSICS GRADEBOOK LOGIN" and contains the instruction "Enter your username and password provided by the department". It includes two input fields: "Username" and "Password". Below these fields are two buttons: "Login" and "Forgot Password? Help". The "Forgot Password? Help" link is in red text.

Figure 8: Non-CAS Login screen

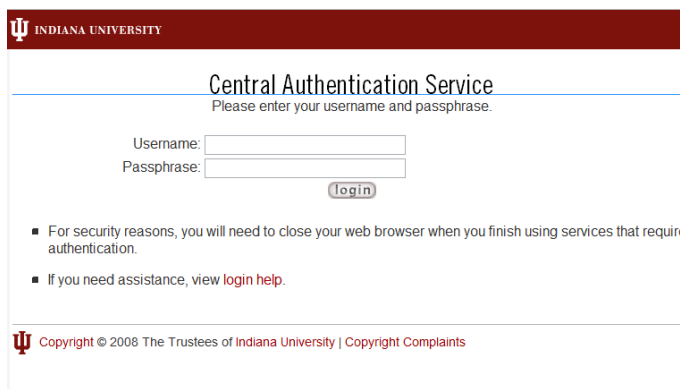
Operation

The user types the username and password and then clicks on the Login button. This invokes 'Login' action of the Auth Module.

A correct username and password takes the user to either to the Gateway page.. An incorrect username and/or password entered by the user means that the user is shown the login screen again with a message indicating this invalid entry.

5.1.3 CAS Login

This page allows CAS users to log in to the system.



INDIANA UNIVERSITY

Central Authentication Service

Please enter your username and passphrase.

Username:

Passphrase:

[login](#)

- For security reasons, you will need to close your web browser when you finish using services that require authentication.
- If you need assistance, view [login help](#).

INDIANA UNIVERSITY Copyright © 2008 The Trustees of Indiana University | Copyright Complaints

Figure 9: CAS Login screen

Operation

The user types the IU username and password and then clicks on the Login button. CAS System then sends a callback to the module.

5.2 Frontend

5.2.1 Add Assignment

This page is handled by the Assignment module generated by Symfony.

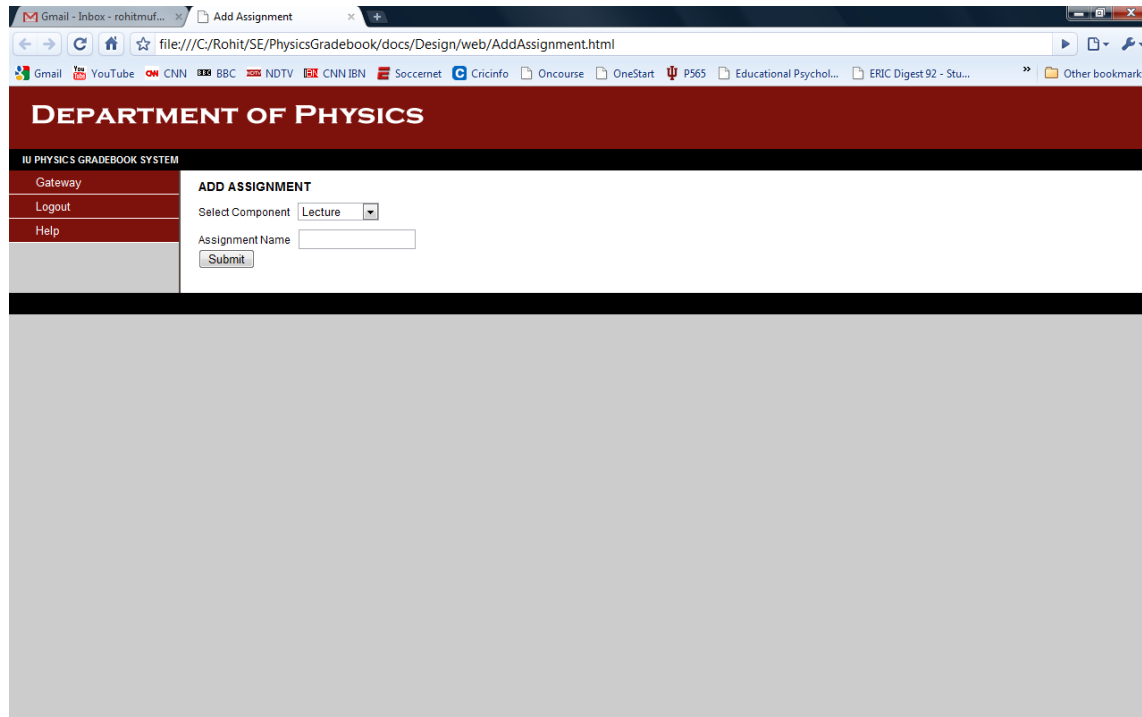


Figure 10: Add Assignment screen

Operation

The administrator enters the assignment title in the text box provided and also selects the component to which the assignment is to be added. On clicking the “Submit” button, the 'Insert' action is invoked.

5.2.2 Assignment List

This page is generated by the Assignment module.

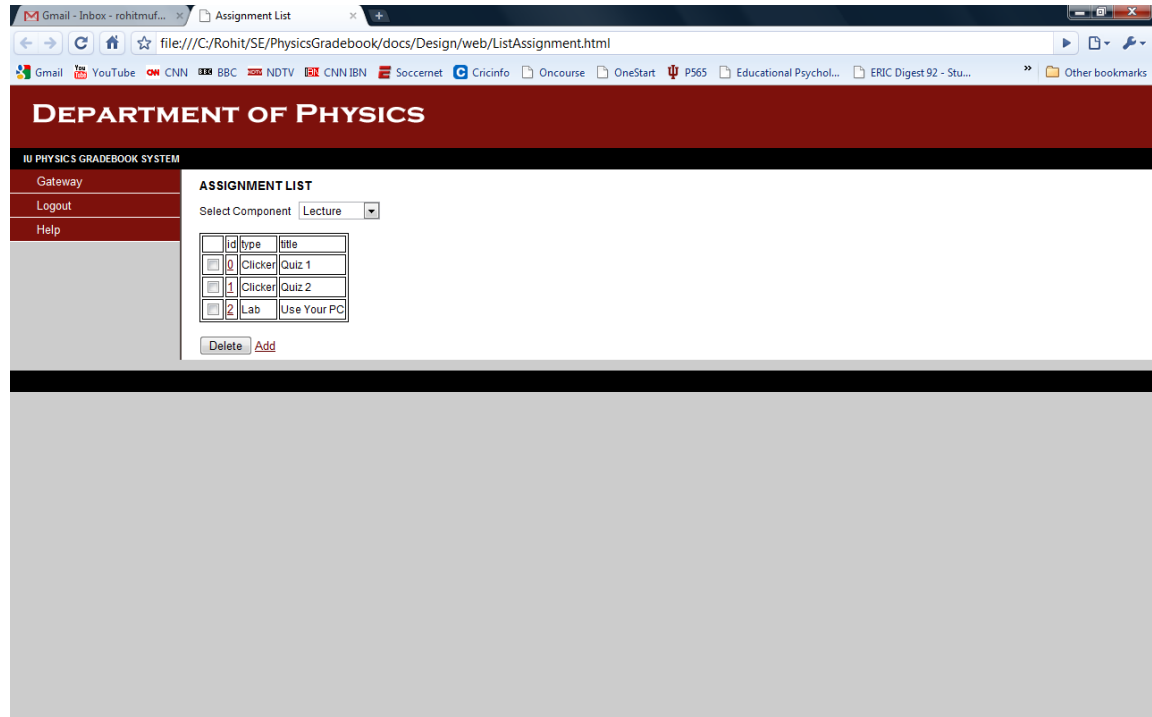


Figure 11: Modify Assignment screen

Operation

The administrator chooses the assignment to be edited or deleted. On clicking the “Submit” button, the underline module is invoked.

5.2.3 Import Grades

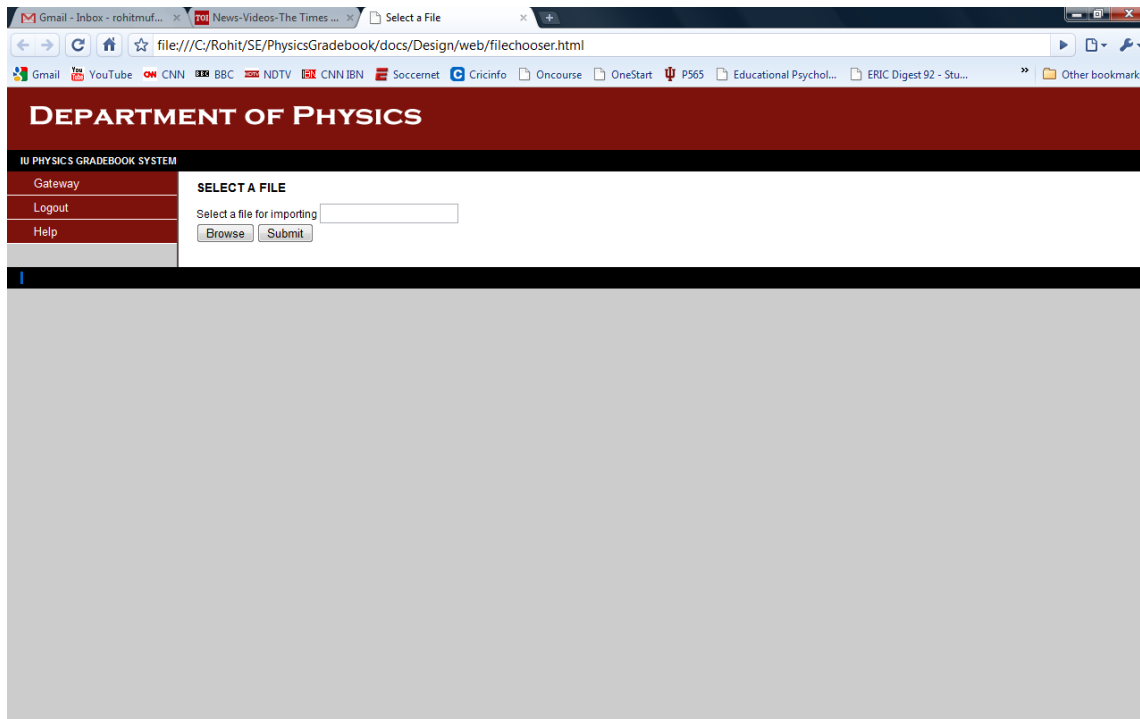


Figure 12: Import Grades screen

Operation

The Professor clicks the “Browse” button to choose the file that needs to be imported. Clicking on the “Import” button invokes the ‘Import’ action.

5.2.4 Import Component Grades

Refer to 5.2.3.

5.2.5 Modify Grades

This page is generated by the 'Index' action of the Grade module.

Figure 13: Modify Grades screen

Operation

The user assigns/modifies grades for any student for any assignment by editing the appropriate text boxes. The user can use the Student or Assignment filter by typing something to search for a specific subset of students.

Navigation

Hyperlink	Destination
Import Grades	Import Offering Grades screen
Export Grades	Export Offering Grades screen

Table 36: Navigation from Modify Offering Grades screen

Appearance and Layout

- During filtering, the results are highlighted in yellow color.

- A valid grade entry in a text box is shown in green color.
- An invalid grade in a text box is highlighted in red color.

5.2.6 View Offering Grades

This page is generated by the 'Index' action of Grade Module. This is the same as the Modify Grades screen but in read-only mode.

Operation

The user can use the Student or Assignment filter to search for a specific subset of students.

Appearance and Layout

- During filtering, the results are highlighted in yellow color.

5.2.7 View Component Grades

Refer to 5.2.6.

5.2.8 View Section Grades

Refer to 5.2.6.

5.2.9 Frontend Gateway

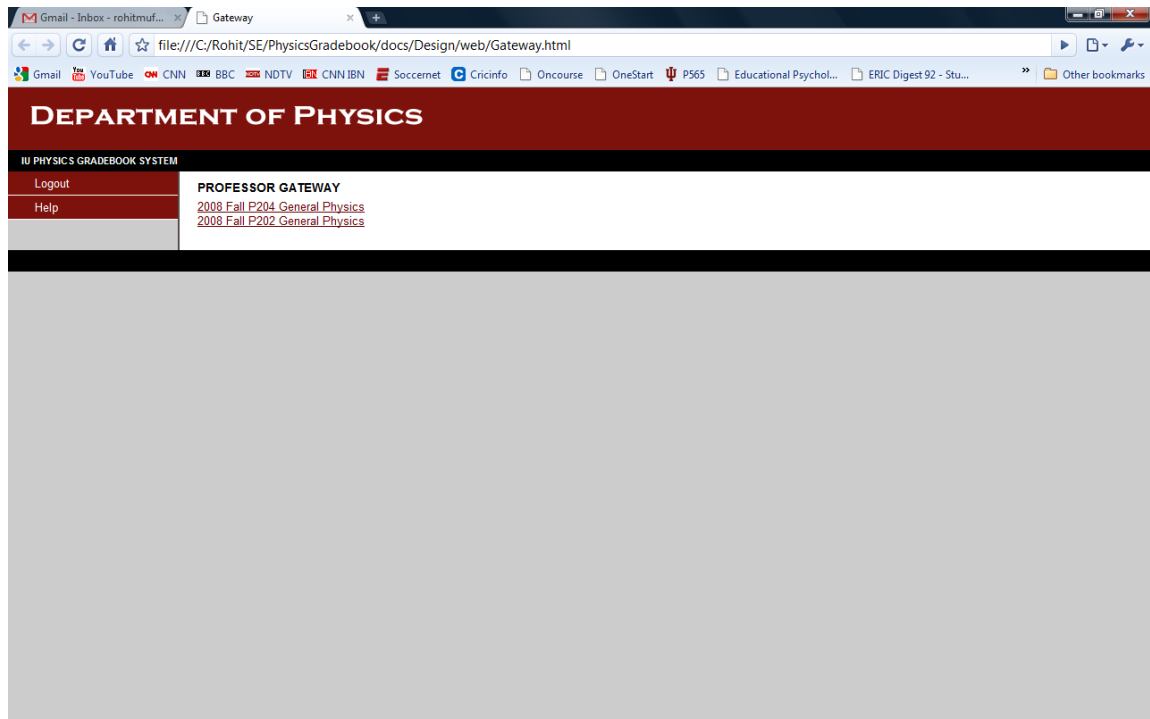


Figure 14: Frontend Gateway screen

Operation

The user views the list of course offerings associated with that user for that semester.

- If the user is a Professor, then only the Professor Gateway is shown to the user.
- If the user is a Student, then only the Student Gateway is shown to the user.
- If the user is an Instructor, then only the Instructor Gateway is shown to the user.
- A user may be both a Student and an Instructor, in which case both the Student and Instructor Gateways are shown to the user.

Navigation

Hyperlink	Destination
links to Offering (Professor)	Offering screen
links to Offering (Student)	Report Grades screen
links to Section (Instructor)	Modify Grades screen

Table 37: Navigation from Frontend Gateway screen

5.3 Backend

The Backend application is entirely generated by Symfony. The general feeling of the user interfaces is described in this section.

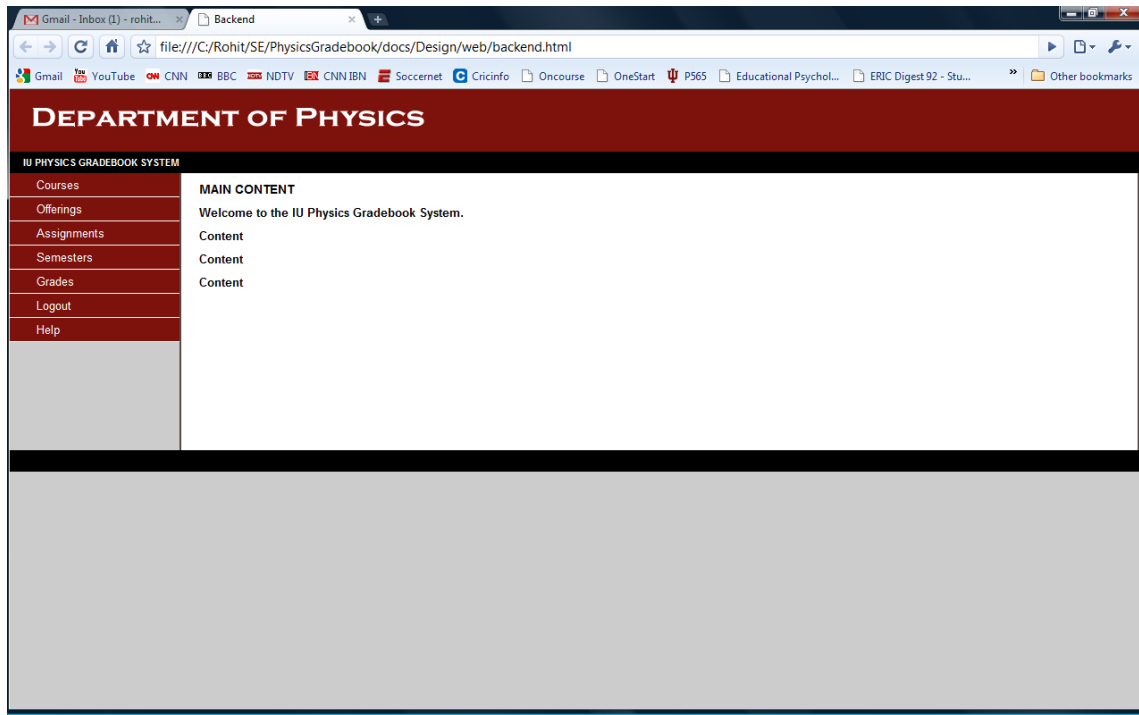


Figure 15: Backend screen layout

Operation

The user clicks on one of the links from the left and will be directed to the pages for managing the corresponding items. The general features of the generated are also described in [11].

6 Exception Handling

This section covers the policy and schema for handling exceptions.

6.1 Platform Failures

A platform failure occurs either because of wrong configuration of the system or malfunction of the platform. The platform failures are resolved at the platform levels. A typical list of possible platform failures and how they will be handled is provided in the following text. Usually, a platform failure should stop the system from working, and until the cause of the failure is resolved in the platform, the system cannot come back online.

Power Failure Power Failure is handled at the operating system and database system level.

Database Failure Temporary database failures are handled at the database system level. MySQL is a mature database management system and the client can rely on its failure recovery mechanism. In rare cases, the IT staff in the client organization has to manually fix the database.

A special case is that the exceptions on the foreign key constraints are not database failures. Instead, they are handled in the Core Libraries and translated into runtime exceptions.

File System Failure File system failures are handled at the operating system level. A corrupt file system does not always put the system offline immediately, but if the corrupt blocks span the files required to support the software system, it will fail. The recovery usually also requires the intervention of the IT staff in the client organization.

Web Server Failure Web server failures are handled at the web server level. A web server failure is also not self-recoverable.

- Internal Server Error
The request was not completed; the server met an unexpected condition
- Not Implemented
The request was not completed; the server did not support the functionality required
- Bad Gateway
The request was not completed; the server received an invalid response from the upstream server
- Service Unavailable
The request was not completed; the server is temporarily overloaded or down
- Gateway Timeout
The gateway has timed out
- HTTP Version Not Supported
The server does not support the "HTTP protocol" version

6.2 Runtime Exceptions

Runtime exceptions occur during the online period of the system when the users are interacting with the system in sessions. Usually the runtime exceptions are generated (thrown) in the Servlets, and handled in

the User Interface Generators. Runtime exceptions do not put the system offline. The Error Handling module provides the mechanisms for runtime exception handling.

- **ACCESS_DENIED**
The user is not allowed to access the requested information
- **INVALID_USERNAME**
The provided user name does not exist
- **INVALID_PASSWORD**
The provided password does not match the user name
- **INVALID_INPUT**
The input is invalid; the definition of invalidity depends on the context
- **RESOURCE_NOT_FOUND**
The template or the requested database entry is not found; the user will be forwarded to an HTTP 404 error page

7 Test Plan

7.1 Scope of Testing

The IU Physics Gradebook project will perform the following tests to ensure the quality of the software:

- Unit Testing
- Functional Testing
- Usability Testing
- Integration Testing
 - The Symfony framework has already provided a reliable integration. To reduce the cost of the development, this test is not repeated by the team.
- Installation Testing
- Beta Testing
- Stress Testing
- Performane Testing
 - Stress Testing on this system is not required, and therefore omitted because the usage of the system is expected to be low.

Regression Test is performed periodically on the Unit Test cases when there is a major change in the code after the system passes all functional tests. This is done at every stage of testing.

7.2 Test Plan overview

Phase	Schedule	Software	Resources
Unit Testing	February 5 to March 5	Symfony	Dreamhost site
Functional Testing	February 15 to March 5	Symfony	Dreamhost site; the interaction with CAS need an on-campus site.
Usability Testing	March 5 to March 15	N/A	Primary client. Dreamhost site and an on-campus site.
Installation Testing	March 15	N/A	Dreamhost site.
Beta Testing	Mar. 15	N/A	Users. An on-campus site.

7.3 Test Procedure

7.3.1 Unit Testing

Unit Testing is the testing done to individual modules after they have been implemented. In Symfony framework, the unit tests are performed on library routines. Because the Core Library (Models) is generated by the framework and little code is written by the team, there are very few Unit test cases, and they are almost identical.

Model Person

Preconditions

```
person = new Person();
```

Tests

```
person.last_name = "Last Name"
person.first_name = "First Name"
assert(person.__toString() == "Last Name, First Name");
```

Model Semester**Preconditions**

```
semester = new Semester();
```

Tests

```
semester.year = "2009";
semester.season = "Fall";
assert(semester.__toString() == "2009 Fall");
```

Model Course**Preconditions**

```
course = new Course();
```

Tests

```
course.course_number = "P201";
course.name = "Physics";
assert(course.__toString() == "P201 Physics");
```

Model Offering**Preconditions**

```
offering = new Offering();
course = new Course();
person = new Person();
semester = new Semester();
offering.course = course;
offering.semester = semester;
offering.professor = person;
```

Tests

```
person.last_name = "L";
person.first_name = "F";
course.course_number = "P201";
course.name = "Physics";
semester.year = "2009";
semester.season = "Summer I";
assert(offering.__toString() == "L F P2001 Physics 2009 Summer I");
```

Model Component**Preconditions**

```
offering = new Offering();
course = new Course();
person = new Person();
semester = new Semester();
component = new Component();
offering.course = course;
offering.semester = semester;
offering.professor = person;
component.offering = offering;
```

Tests

```
person.last_name = "L";
person.first_name = "F";
course.course_number = "P201";
course.name = "Physics";
semester.year = "2009";
semester.season = "Summer I";
component.type = "Lab";
assert(component.__toString() == "L F P2001 Physics 2009 Summer I Lab");
```

Model Section**Preconditions**

```
section = new Section();
```

Tests

```
section.section_number = "28981";
assert(section.__toString() == "28981");
```

Model Assignment**Preconditions**

```
ass = new Assignment();
```

Tests

```
ass.type = "Clicker";
ass.name = "1.1";
assert(ass.__toString() == "Clicker 1.1");
```

Model Grade**Preconditions**

```
grade = new Grade();
```

Tests

```
grade.grade = "1238";
grade.timestamp = now();
assert(grade.__toString() == "1238");
```

7.3.2 Functional Testing

Functional Test focuses on the functionality of the system. This test is performed on the modules and their actions. A database connection is required and the database has to be populated with predefined fixtures. The plan given in this document serves only as a general guide line for the Functionaal Test.

Tests on Symfony generated modules are not performed.

Module F/Welcome

Precondition None

Tests No test is planned.

Module F/auth and B/auth

Precondition Fixture 00-person

Tests

```
request('auth/login');
assert(response.has('Forgot Password?'));
request('auth/login?username=user&password=pass', 'POST');
assert_ok();
request('auth/login?username=user&password=wrong', 'POST');
assert_failure();
request('auth/logout');
assert_ok();
```

Module B/Roster

Precondition Fixture 00-person, 10-course-hier

Tests

```
file = <<EOF
cas1, L, F, 00000, 00001, 99999
cas2, L, F, 00000, 00001, 99999
cas3, L, F, 00000, 00001,
EOF
request('roster/import/2009-fall/dan/p201-physics', file);
assert_failure();
file = <<EOF
cas1, L, F, 00000, 00001, 00002
cas2, L, F, 00000, 00001, 00002
cas3, L, F, 00000, 00001,
EOF
request('roster/import/2009-fall/dan/p201-physics', file);
assert_ok();
request('roster/export/2009-fall/dan/p201-physics');
assert(response.has(file));
```

Module F/Gateway

Precondition Fixture 00-person, 10-course-hier, F/auth

Tests

```
request('login?username=user&password=pass');
request('gateway/index');
assert(response.has('P201'));
```

Module F/Grade

Precondition Fixture 00-person, 10-course-hier

Tests Test cases on filter and import/export are to be decided.

```
request('login?username=user&password=pass');
request('grade/index/offering/0');
assert(response.has('P201'));
```

Module F/Assignment

Precondition None

Tests Test is not planned.

Module F/FinalGrade

Precondition Fixture 00-person, 10-course-hier

Tests Test cases for import/export are to be determined.

```
request('login?username=user&password=pass');
request('final/offering/0');
assert_ok();
```

7.3.3 Usability Testing

This step involves the client in testing, evaluating whether the system is useful and suitable for the client's operation and meets all the functional requirements. In this plan the team is going to test how the system is going to meet the functional requirements mentioned by the client and the user friendliness of the screens. The test cases are formatted in a Screen(Action) pattern.

Frontend

Precondition Welcome(IUP Login or CAS login) -> iuplogin or caslogin(submit) -> gateway for all the functional requirements.

View Student Grades gateway(offering)-> offering(view grades) -> gradereport

Instructor Assign Section Grades gateway(section) -> gradeentry

Professor Assign Section Grades gateway(offering) -> offering(section) -> gradeentry

Add Assignments gateway(offering) -> offering(assignment) -> assignment(add assignment)

Modify Assignments gateway(offering) -> offering(assignment) -> assignment(modify assignment)

Hide Assignments gateway(offering) -> offering(assignment) -> assignment(hide assignment)

Import/Export Course/Component Grades gateway(offering) -> offering(export grade) -> grade

Import/Export Course/Component Roster gateway(offering) -> offering(export roster)

Backend

Precondition Welcome(IUP Login) -> iuplogin(submit) -> gateway

Add Courses gateway(course) -> course(add)

Modify Courses gateway(course) -> course(modify)

Add Course Offerings gateway(offering) -> offering(add)

Modify Course Offerings gateway(offering) -> offering(modify) -> modify

Assign Professors to Offerings gateway(offering) -> offering(field : professor) -> modify
Add Students to Offerings gateway(offering) -> offering(field : student list) -> modify
Hide Students from Offering Rosters gateway(offering) -> offering(field : student list) -> modify
Add Components gateway(component) -> component(add component)
Hide Components gateway(component) -> component(hide component)
Modify Components gateway(component) -> component(modify component)
Add Assignments gateway(assignment) -> assignment(add assignment)
Modify Assignments gateway(assignment) -> assignment(modify assignment)
Hide Assignments gateway(assignment) -> assignment(hide assignment)
Move Students between Sections gateway(person) -> person (modify)
Add Sections gateway(section) -> section(add)
Modify Sections gateway(section) -> section(modify)
Hide Sections gateway(section) -> section(hide)
Assign Instructors to Sections gateway(section) -> section(modify)

7.3.4 Installation testing

After the schedule date, the team performs the Installation Test at on-campus site.

The steps for the Installation test are:

1. Clean up the on-campus site;
2. Make a snapshot of the repository;
3. Copy the snapshot to the on-campus site;
4. Change the configurations;
5. Perform the Usability Test on the installed system.

7.3.5 Beta Testing

This is performed after the installation testing.

1. Code will be freezed;
2. Deliver the system to the client;
3. Get it installed;
4. Get feed back from the users of the system;
5. Ressolve the issues if existed

7.3.6 Performance Testing

This step examines how well the system performs, that is, the manner in which it does its computations. Performance testing on this system is not required, and therefore omitted because the usage of the system is expected to be low.

References

- [1] Feasibility Study for Physics Gradebook System.
- [2] Requirement Specification for physics Gradebook System.
- [3] P465-6 & P565-6 Software Engineering for Information Systems I & II: Information Packet, Computer Science Department, Indiana University, 2008.
- [4] IEEE Standard for Software Quality Assurance Plans (STD 730-1984), Inst. of Electrical and Electronics Engineers, New York, 1984.
- [5] Requirement Specifications of Sakai Gradebook. <https://source.sakaiproject.org/svn/gradebook/trunk/xdocs/specs23/index.htm>
- [6] Information about Integrating CAS with a website. <http://kb.iu.edu/data/atfc.html>
- [7] PHP-Unit Project, <http://www.phpunit.de/manual/3.3/en/index.html>
- [8] Selenium Project, <http://selenium.seleniumhq.org/>
- [9] OpenWebLoad Project, <http://openwebload.sourceforge.net/>
- [10] NIKTO Project, <http://www.cirt.net/nikto2>
- [11] Practical Symphony (Doctrine ORM), http://www.symfony-project.org/doc/1_2/

List of Tables

1	COURSE Table Fields	20
2	COURSE Table Constraints	20
3	COURSE_DEFAULT_COMPONENTS Table Fields	20
4	COURSE_DEFAULT_COMPONENTS Table Constraints	21
5	PERSON Table Fields	21
6	PERSON Table Constraints	22
7	SEMESTER Table Fields	22
8	SEMESTER Table Constraints	22
9	OFFERING Table Fields	23
10	OFFERING Table Constraints	23
11	COMPONENT Table Fields	23
12	COMPONENT Table Constraints	24
13	SECTION Table Fields	24
14	SECTION Table Constraints	24
15	ASSIGNMENT Table Fields	25
16	ASSIGNMENT Table Constraints	25
17	GRADE Table Fields	25
18	ASSIGN_GRADES Table Constraints	26
19	ENROLL_OFFERING Table Fields	26
20	ENROLL_OFFERING Table Constraints	27
21	ENROLL_COMPONENT Table Fields	27
22	ENROLL_COMPONENT Table Constraints	27
23	TEACH_SECTION Table Fields	28
24	TEACH_SECTION Table Constraints	28
25	JOIN_SECTION Table Fields	29
26	JOIN_SECTION Table Constraints	29
27	LATEST_GRADE Table Fields	29
28	LATEST_GRADE Table Constraints	30
29	COMPONENT_TYPE Table Fields	30
30	COMPONENT_TYPE Table Constraints	30
31	ROLE_TYPE Table Fields	31
32	ROLE_TYPE Table Constraints	31
33	ASSIGNMENT_TYPE Table Fields	31
34	ASSIGNMENT_TYPE Table Constraints	31
35	Navigation from Welcome screen	34
36	Navigation from Modify Offering Grades screen	40
37	Navigation from Frontend Gateway screen	43
39	List of Requirements Changes	62

List of Figures

1	Architecture of the Services	6
2	Architecture of the Control Flow	7

3	Architecture of the Components. The system is composed of three components which are shown in this figure.	8
4	Menu Hierarchy at Login	32
5	Menu Hierarchy for General Users	32
6	Menu Hierarchy for Administrator	33
7	Welcome screen	34
8	Non-CAS Login screen	35
9	CAS Login screen	36
10	Add Assignment screen	37
11	Modify Assignment screen	38
12	Import Grades screen	39
13	Modify Grades screen	40
14	Frontend Gateway screen	42
15	Backend screen layout	43

Index

Administration services, 6
Assignment, 24, 33, 41

banner, 32
browser, 8

CAS, 34
Component, 23, 32, 33
control model, 6
controller, 8
Core Libraries, 9
Course, 20, 22, 23, 26, 33
course hierarchy, 6
Course Offering, 33

Data Manipulation, 9
DBMS, 8
Dialog box, 32

ER model, 20
Error Handling library, 9
error message, 32
Exception Handling, 44

File system failure, 44
filtering, 40, 41

gateways, 32
General Users, 32

heading, 32

Instructor, 25, 28, 29, 42
Internet Explorer, 8

Offering, 22, 32
offerings, 42
ozilla Firefox, 8

password, 36
PHP session, 9
picture, 35
Power Failure, 44
Professor, 20, 26, 42

Regular page, 32

Regular user services, 6
Runtime exception, 44

Safari, 8
Section, 24, 28, 32, 33
section number, 24
Semester, 22
semester, 42
Servlets, 9
Session Management, 9
structural system model, 6
Student, 27, 41, 42
sub-system decomposition model, 6
Submit, 32

Template Library, 8
Temporary database failure, 44

User and Permissions, 9
user interface, 32
User Interface Generators, 9
username, 36

Web server failure, 44

Glossary

Term	Definition
Apache	Computer program that is responsible for accepting HTTP requests from web clients, which are known as web browsers, and serving them HTTP responses along with optional data contents, which usually are web pages such as HTML documents and linked objects (images, etc.).
Assignment	Homework given to students enrolled in the components. It could also be a test. In our project, an assignment means just the name of the assignment.
Course	Offered to students as part of the curriculum or program.
Component	Division of a course.
Central Authentication Service	A login service that allows access to multiple password-protected web systems after logging in once on a central authentication server; this is often referred to as single sign-on.
Data Flow Diagram	Graphical representation of the "flow" of data through an information system.
Entity-Relationship Model	Relational schema database modeling method used in software engineering to produce a type of conceptual data model (or semantic data model) of a system, often a relational database.
MySQL	Relational database management system (RDBMS). The program runs as a server providing multi-user access to a number of databases.
OnCourse	Indiana University's online collaboration and learning environment, powered by the Sakai community, supports teaching and learning, committees, projects, research, and portfolios for Indiana University's community of students, faculty, and staff.
PHP	Scripting language, originally designed for producing dynamic web pages.
Section	Division of a component.
Web Hosting Service	Type of Internet hosting service that allows individuals and organizations to make their own website accessible via the World Wide Web.

Term	Definition
Dream Host	A web hosting provider and domain name registrar.
Doctrine/phpDoctrine	An object-relational mapper that sits on top of a database abstraction layer. It allows easy access to all types of databases, such as MySQL, through the use of PHP objects.
Object-Relational Mapping	Object-relational mapping is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages.
Model-View-Controller	A design pattern which decouples data access, business logic, and data presentation and user interaction.
Symfony	A web application framework written in PHP which follows the model-view-controller (MVC) paradigm.

A List of Requirements Changes

New Requirement	Old Requirement	Reason for Change
Assignments are classified by types	N/A	In a lecture component, assignments can be either homework or “clicker questions”
Roles: users, power users and administrators	Single Administrator	Secretaries also do the management tasks, but they do not manage user privileges; client requirement
IUP users and CAS users	All CAS users and the administrator authorized internally	Some students do not have CAS account; specified by client
Basic validation for the grades	N/A	Some grades are obviously ill-formed and should be excluded, e.g., “A B”, “3.B”, and extra spaces
A report for number of students in each section	N/A	New client requirement
Grades can be as long as 20 characters	N/A	To allow for verbose grades like “Excellent”.
Calculate the average grade in a component (per student)	N/A	The simple average grade helps in calculating the final grades; new client requirement
The system must be made secure from regular external hackers	N/A	New client requirement

Table 39: List of Requirements Changes

B Introduction to Symfony

Symfony provides an architecture, components and tools for developers to build complex web applications faster. The framework of Symfony follows the Model-View-Controller (MVC) pattern, which separates the business logic (model) and the presentation (view). The result is a high maintainability product.

Fortunately, the architecture design of IU Physics Gradebook system perfectly fits into the MVC pattern. In the Symfony framework:

1. The core library is implemented. The database manipulation module is automatically generated by from the relational database design by Doctrine Object-Relation-Model (ORM) tool.
2. The template library is implemented. Symfony framework provides three levels of abstraction in UI implementations: layouts, content templates, and forms.
3. The Regular User Services and Administration Services are separately supported by two web applications: the frontend and the backend.
4. The fundamental data manipulation backend is automatically generated by Doctrine ORM tool.
5. Validations are integrated into the Symfony Form/Widget framework.
6. The modules in Symfony map to the Servlet and UI Generators in the IU Physics Gradebook design.

C Links to Requirements Specification

Requirement	Tables	Module	User Interface
2(a).i View Student Grades	Person, Offering, Component, Assignment, Grade	F/Grade	GradeReportScreen
2(b).i Assign Section Grades	Grade	F/Grade	GradeEntryScreen
2(c).i Assign Section Grades	Grade	F/Grade	GradeEntryScreen
2(c).ii Add Assignments	Assignment	F/Assignment	AssignmentScreen
2(c).iii Modify Assignment	Assignment	F/Assignment	AssignmentScreen
2(c).iv Hide Assignments	Assignment	F/Assignment	AssignmentScreen
2(c).v View/Export Overall Course Grade	Grade, Assignment, Person, Offering	F/Final Grade	GradeScreen
2(c).vi View/Export Overall Component Grade	Grade, Assignment, Person, Component	F/Final Grade	GradeScreen
2(c).vii Import/Export Course Roster	Removed from the requirements		
2(c).viii Import/Export Component Roster	Removed from the requirements		
2(d).i Add Courses	Course	B/Course	CourseScreen
2(d).ii Modify Courses	Course	B/Course	CourseScreen
2(d).iii Add Course Offerings	Offering	B/Offering	OfferingScreen
2(d).iv Modify Course Offerings	Offering	B/Offering	OfferingScreen
2(d).v Assign Professors to Offerings	Offering	B/Offering	OfferingScreen
2(d).vi Add Students to Offerings	Person, Offering	B/Person and Offering	PersonScreen
2(d).vii Hide Students from Offering Rosters	Person, Offering	B/Person and Offering	PersonScreen
2(d).viii Add Components	Component	B/Component	ComponentScreen
2(d).ix Hide Components	Component	B/Component	ComponentScreen
2(d).x Modify Components	Component	B/Component	ComponentScreen
2(d).xi Add Assignments	Assignment	B/Assignment	AssignmentScreen
2(d).xii Modify Assignment	Assignment	B/Assignment	AssignmentScreen
2(d).xiii Hide Assignments	Assignment	B/Assignment	AssignmentScreen
2(d).xiv Move Students between Sections	Section, Component	B/Section, Component	SectionScreen
2(d).xv Add sections	Section	B/Section	SectionScreen
2(d).xvi Modify Sections	Section	B/Section	SectionScreen
2(d).xvii Hide Sections	Section	B/Section	SectionScreen

Requirement	Tables	Module	User Interface
2(d).xviii Assign Instructors to Sections	Section, Person	B/Section, Person	SectionScreen
2(d).xix Assign Grades	Grade, Person, Assignment	B/Grade, Person, Assignment	GradeScreen
2(d).xx View/Export Overall Course Grades	Offering, Grade	F/Offering, Grade	GradeScreen
2(d).xxi View/Export Overall Component Grades	Component, Grade	F/Component, Grade	GradeScreen
2(d).xxii View/Export Course Roster	Offering, Person	F/Offering, Person	OfferingScreen
2(d).xxiii View/Export Component Roster	Component, Person	F/Component, Person	ComponentScreen
2(d).xxiv Import final Course Grades	Offering, Person	F/Offering, Person	OfferingScreen
2(d).xxv Import final Component Grades	Component, Person	F/Component, Person	ComponentScreen

D Coding Standards and Conventions

D.1 Introduction

The goal of these guidelines is to facilitate the programmers to create uniform code that is easier to understand and maintain. In this document, the guidelines on PHP modules, database naming and user interface design are listed.

D.2 Modules

D.3 Module Decomposition

All source code will be grouped into modules. The team is not going to adopt Object Oriented features of PHP; rather, each module corresponds to a name-space or class that deals with a single, unique domain.

- One source code file corresponds to one module in the system.
- If the module deals with the objects, the handler of the object is passed in function calls instead of global variables.

D.4 Headers

Headers in PHP and JavaScript are different from other languages like C or C++. In PHP and JavaScript, the modules themselves are contained in the headers.

D.4.1 PHP Headers

- All the headers are included by the same entrance file "include.php" which resolves the location of other headers and includes them. This is to avoid troubles for source code in different sub-directories to locate the headers.
- Use "include" statement to include the "include.php" file.
- Site configuration files are also provided as PHP headers. Missing site configurations should not cause the system to panic in a runtime-error manner.
- All global variables should be declared in a single file; the initialization is in each module.
- Each file begins with the includes, followed by context lines. Then the declarations of functions follows.
- Public member functions are prefixed with the module name.
- Private member functions are prefixed with an underscore ("_").

D.4.2 JavaScript Headers

- HTML pages include JavaScript headers with relative path names.
- Dependencies are resolved in the HTML pages. JavaScript headers never resolve the header dependencies.
- No global variables are preserved between JavaScript headers.
- Public members are prefixed with module name.
- Private members are prefixed with an underscore ("_").

D.4.3 CSS Headers

- CSS headers are included statically in the HTML pages.

D.5 Commenting and Indentation

D.5.1 File Header

Each source file begins with a header section consisting of comments. The purpose of the header is to describe the general characteristics of the file and the module.

```

/*****
 * Reference to the Copyright Licenses (GPL)
 *
 * FILE: sample.php
 * MODULE NAME: sample_module
 *
 * DESCRIPTION: This is a sample module (DEPRECATED).
 *
 * NOTES: If the file is included, the application dies.
 *****/

```

Fortunately, PHP, JavaScript and CSS share the same syntax of commenting.

D.5.2 Member Function Header

Each member function in a module should have a comment header to describe the behavior of the member function. The gtk-doc² convention is used.

```

/*****
 * sample_function:
 *   @self: the handler of sample object.
 *   @param2(out): the second parameter.
 *
 * sample_function do nothing. It should do something.
 *   Refer to @sample_function2.
 *
 * returns: TRUE if success, FALSE if not.
 *
 * Changes: added in rev 212.
 *****/

```

D.5.3 In-line Commenting

In complicated member functions, appropriate in-line comments should be written to describe the process. These in-line comments begins with "/*", follows by a line to describe the comment and stops with "*/". The in-line comments should always follow the same indenting rules of the code it is explaining.

²<http://www.gtk.org/gtk-doc/>

- “FIXME” stands for known rare wrong behavior.
- “NOTE” stands for special notes about the code.
- “TODO” stands for unfinished implementations.

```

.....
....
/* FIXME: divided by zero is not handled */
$i = $i / $number;
....
.....

```

D.5.4 Indentation Rules

Indentation are by Tabs (ASCII code 08). Every programmer's editor should be configured to visualize Tabs with 4 spaces.

Context elements that increase the indentation level

- Context level
- Member functions
- Code blocks

```

/*****
 * FILEHEADER
 *****/
$GLOBAL_VAR = "something";
/*****
 * HEADER
 *****/
function sample_function() {
    .....

    /* TODO: write stuff */
    .....

    .....
}

```

D.5.5 Line Breaking Rules

- Between comment header and member functions there should be no blank lines.
- Between in-line comments and the code there should be no blank lines.
- Add a line break after a curly brace.
- Add a line break after each code block.
- Use common sense to add extra line breaks when semantic groups of code finish.

D.5.6 Blank/Space Rules

- Operators and operands are generally separated by a space (ASCII 32).

- Commas should be followed by a space.
- Avoid continuous spaces.
- In the function definition, a space should be inserted between the function name and the formal parameter list.
- In a function invocation, no space should be inserted between the function name and the argument list.
- A space should always be inserted before an opening curve brace (“{”).

D.6 Naming Conventions

The general guidelines for naming is to use descriptive names. Therefore the names tend to be long and verbose.

D.7 Database Schema

- Table names should be in upper case. Multiple words are separated by underscores (“_”).
- Field names should be in lower case. Multiple words are separated by underscores (“_”). In a field name, the name of the table should be prefixed in lower case if possible. E.g., `table_field` is a field for table `TABLE`.

D.8 File Naming Conventions

- File names for internal modules are in lowercase. If it contains multiple words, separate them by underscores (“_”). No spaces are allowed. For example, `sql_template_manager.php` is a valid file name.
- File names for external modules (UI and Servlets) are in Camel Case.
- File names for JavaScript modules are in lowercase.
- File names for SQL query templates are in lower case.
- File names for HTML templates are the same as the UI module names, with an extension `.tpl`.

D.9 Local Variables

- Local variables are in lowercase, separated by underscores (“_”).
- Module static(private) variables are in uppercase, separated by underscores (“_”) and prefixed with underscores (“_”).

```
$_SAMPLE_LOCAL_VARIABLE = "local";
function sample_function () {
    $local_variable = "value";
}
```

D.10 Global Variables

- Global variables are in uppercase, separated by underscores (“_”).

```
$DB_USERNAME = "dbadmin";
```

D.11 User Interface

User interface modules require special standardization.

D.12 Form Layout

- The banner³ should always be at the top of the form.
- Color highlighting should not be abused; and special considerations for visually handicapped people should be taken.
- The submit button should always be on the left bottom of the form.
- Graphics should have alternate text.
- Input fields should have labels.
- Tables should have captions.
- Large tables should be scrollable and limited to the browser area of the user.

D.13 Form Navigation

- Forms are linked with plain hyperlinks.
- JavaScript should be avoided for submitting forms and form navigation.

D.14 Informative Messages

- Informative messages should be present just below the title of the form.
- A box or similar structure should be used to separate the informative messages from the other elements of the form.
- Informative messages should not be highlighted.

D.15 Key Bindings

- Default key bindings of Web Browser should not be altered.
- Tabs sequence should be redefined for modifying grades.

³The picture to indicate the existence of the system.