

# Preliminary Design

## IU Physics Gradebook System

Yu Feng<sup>1</sup>

Rohit Chandran<sup>2</sup>

Naga Rekha Malae<sup>3</sup>

<sup>1</sup>Physics Department, Indiana University, Bloomington IN 47401

<sup>2</sup>Department of Computer Science, Indiana University, Bloomington IN 47401

<sup>3</sup>Department of Computer Science, Indiana University, Bloomington IN 47401

## Executive Summary

This document covers certain technical aspects of the IU Physics Gradebook system.

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 library and the core libraries.

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 take the advantages of the database platform for data consistency and automatic crash recovering.

The user interface of the proposed system that meets the client's functionality requirements is brought to fore, along with the operations performed on each screen and the navigation path between these screens. Certain common appearance and layout characteristics are mentioned.

The preliminary test plan for the system has been proposed. The plan spans the from the 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 in detail.

Additional and modified requirements of the client have been added to the end of the document.



## Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Requirement Specifications</b>	<b>6</b>
2.1	Client Background . . . . .	6
2.2	Project Goals . . . . .	6
2.3	Environment . . . . .	6
2.4	Data Flow Diagrams . . . . .	6
2.5	Entity-Relationship Diagram . . . . .	6
2.6	Functionality Requirements . . . . .	6
<b>3</b>	<b>Architecture and Modules</b>	<b>7</b>
3.1	Architecture . . . . .	7
3.1.1	Structural System Model . . . . .	7
3.1.2	Control Flow Architecture . . . . .	8
3.1.3	Component Architecture . . . . .	9
3.2	Module Coupling . . . . .	10
3.2.1	Interaction between Modules . . . . .	10
3.2.2	Convention Type Names . . . . .	10
3.3	Template Library . . . . .	12
3.4	User Interface Generator . . . . .	12
3.5	Servlets . . . . .	14
3.5.1	Modify Grades Servlet . . . . .	14
3.5.2	Component Management Servlet . . . . .	14
3.5.3	Course Management Servlet . . . . .	14
3.5.4	Offering Management Servlet . . . . .	15
3.5.5	User Management Servlet . . . . .	15
3.6	Error Handling Module . . . . .	15
3.7	Core Libraries . . . . .	16
3.7.1	Session Management . . . . .	16
3.7.2	User and User Permission Authorization . . . . .	16
3.7.3	Data Manipulation . . . . .	17
<b>4</b>	<b>Data Design</b>	<b>20</b>
4.1	Introduction: . . . . .	20
4.2	Table Representation: . . . . .	20
4.3	PERSON Table . . . . .	20
4.3.1	COURSE Table . . . . .	21
4.3.2	OFFERING Table . . . . .	21
4.3.3	SEMESTER Table . . . . .	22
4.3.4	COMPONENT Table . . . . .	22
4.3.5	COMPONENT_TYPE Table . . . . .	23
4.3.6	COURSE_DEFAULT_COMPONENTS Table . . . . .	23
4.3.7	ENROLL_COMPONENT Table . . . . .	24
4.3.8	ENROLL_OFFERING Table . . . . .	25
4.3.9	JOIN_SECTION Table . . . . .	26

4.3.10	SECTION Table	27
4.3.11	TEACH_SECTION Table	28
4.3.12	SESSIONS Table	28
4.3.13	ASSIGN_GRADES Table	29
4.3.14	LATEST_ASSIGNED_GRADES Table	30
4.3.15	ROLE_TYPE Table	31
4.3.16	ASSIGNMENT_TYPE Table	31
4.3.17	ASSIGNMENT Table	32
<b>5</b>	<b>User Interface</b>	<b>34</b>
5.1	Administrator	35
5.1.1	Login	35
5.1.2	Administrator Gateway	36
5.1.3	Add Offering	37
5.1.4	Offering Management	37
5.1.5	Add Component to Course Offering	38
5.1.6	Component Management	38
5.1.7	Add Section to a Component	38
5.1.8	Add Student to a Course Offering	39
5.1.9	Add Assignment to a Component	40
5.1.10	Edit Assignment in a component	41
5.1.11	Edit Component	41
5.1.12	Edit Course	42
5.1.13	Export Grades	43
<b>6</b>	<b>Exception Handling</b>	<b>44</b>
6.1	Platform Failures	44
6.2	Runtime Exceptions	45
6.3	Error Handling Schema	45
<b>7</b>	<b>Test Plan</b>	<b>47</b>
7.1	Unit Testing	47
7.2	Security Testing	47
7.3	Functionality Testing	47
7.4	Integration Testing	47
7.5	User Acceptance Testing	47
7.6	Performance Testing	48
7.7	Regression Testing	48
7.8	Multi-User Testing	48
7.9	Usability Testing	48
7.10	Operation Acceptance Testing	48
7.11	Database testing	48
<b>A</b>	<b>List of Requirements Changes</b>	<b>51</b>

<b>B</b>	<b>Coding Standards and Conventions</b>	<b>52</b>
B.1	Introduction . . . . .	52
B.2	Modules . . . . .	52
B.3	Module Decomposition . . . . .	52
B.4	Headers . . . . .	52
B.4.1	PHP Headers . . . . .	52
B.4.2	Javascript Headers . . . . .	53
B.4.3	CSS Headers . . . . .	53
B.5	Commenting and Indentation . . . . .	53
B.5.1	File Header . . . . .	53
B.5.2	Member Function Header . . . . .	53
B.5.3	In-line Commenting . . . . .	54
B.5.4	Indentation Rules . . . . .	54
B.5.5	Line Breaking Rules . . . . .	55
B.5.6	Blank/Space Rules . . . . .	55
B.6	Naming Conventions . . . . .	55
B.7	Database Schema . . . . .	55
B.8	File Naming Conventions . . . . .	55
B.9	Local Variables . . . . .	56
B.10	Global Variables . . . . .	56
B.11	User Interface . . . . .	56
B.12	Form Layout . . . . .	56
B.13	Form Navigation . . . . .	56
B.14	Informative Messages . . . . .	57
B.15	Key Bindings . . . . .	57

## List of Tables

1	PERSON Table Fields . . . . .	20
2	PERSON Table Indexes . . . . .	21
3	COURSE Table Fields . . . . .	21
4	COURSE Table Indexes . . . . .	21
5	OFFERING Table Fields . . . . .	21
6	OFFERING Table Indexes . . . . .	22
7	SEMESTER Table Fields . . . . .	22
8	SEMESTER Table Indexes . . . . .	22
9	COMPONENT Table Fields . . . . .	22
10	COMPONENT Table Indexes . . . . .	23
11	COMPONENT_TYPE Table Fields . . . . .	23
12	COMPONENT_TYPE Table Indexes . . . . .	23
13	COURSE_DEFAULT_COMPONENTS Table Fields . . . . .	23
14	COURSE_DEFAULT_COMPONENTS Table Indexes . . . . .	24
15	ENROLL_COMPONENT Table Fields . . . . .	24
16	ENROLL_COMPONENT Table Indexes . . . . .	25
17	ENROLL_OFFERING Table Fields . . . . .	25

18	ENROLL_OFFERING Table Indexes . . . . .	26
19	JOIN_SECTION Table Fields . . . . .	26
20	JOIN_SECTION Table Indexes . . . . .	27
21	SECTION Table Fields . . . . .	27
22	SECTION Table Indexes . . . . .	27
23	TEACH_SECTION Table Fields . . . . .	28
24	TEACH_SECTION Table Indexes . . . . .	28
25	SESSIONS Table Fields . . . . .	29
26	SESSIONS Table Indexes . . . . .	29
27	ASSIGN_GRADES Table Fields . . . . .	29
28	ASSIGN_GRADES Table Indexes . . . . .	30
29	LATEST_ASSIGNED_GRADES Table Fields . . . . .	30
30	LATEST_ASSIGNED_GRADES Table Indexes . . . . .	31
31	ROLE_TYPE Table Fields . . . . .	31
32	ROLE_TYPE Table Indexes . . . . .	31
33	ASSIGNMENT_TYPE Table Fields . . . . .	32
34	ASSIGNMENT_TYPE Table Indexes . . . . .	32
35	ASSIGNMENT Table Fields . . . . .	32
36	ASSIGNMENT Table Indexes . . . . .	33
38	Offering Screen for Administrator . . . . .	37
39	List of Requirements Changes . . . . .	51

## List of Figures

1	Architecture of the Services . . . . .	7
2	Architecture of the Control Flow . . . . .	8
3	Architecture of the Components . . . . .	9
4	Menu Hierarchy at Login . . . . .	34
5	Menu Hierarchy for Administrator . . . . .	34
6	Menu Hierarchy for general users . . . . .	35
7	Menu Hierarchy for Professor . . . . .	35
8	Login screen . . . . .	36
9	Component Management screen . . . . .	38
10	Add Student screen . . . . .	39
11	Add Assignment screen . . . . .	40
12	Edit Assignment screen . . . . .	41
13	Edit Course screen . . . . .	42
14	Export Course Grades screen . . . . .	43
15	Error Handling Schema . . . . .	46

## 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 of the document is carried forward from the Requirements Specification and these bring to light details about the client background, project goals, environment along with Data Flow Diagrams and Entity-Relationship diagrams for both the current and proposed system.

Section 3 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 Controller within this component architecture consists of five modules, namely, the template library, user interface generators, servlets, error handling library and the core libraries. 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 tables are designed with the InnoDB structure that keeps the data consistent and also automatically recovers from a crash.

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.

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, inline commenting and indentation are standardized in Appendix B. These coding standards, as required in the QA plan in previous documents, provide a quality information system to the client.

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

Additional and modified requirements of the client have been added to the end of the document as the Appendix A.



## **2 Requirement Specifications**

### **2.1 Client Background**

The client background is described in [2]. The preliminary design does not include a copy of it because there is no specific change since requirement analysis.

### **2.2 Project Goals**

The project goals is described in [2]. The preliminary design does not include a copy of it because there is no specific change since requirement analysis.

### **2.3 Environment**

The environment is described in [2]. The preliminary design does not include a copy of it because there is no specific change since requirement analysis.

### **2.4 Data Flow Diagrams**

The Data Flow diagrams are described in [2]. The preliminary design does not include a copy of it because there is no specific change since requirement analysis.

### **2.5 Entity-Relationship Diagram**

The ER diagram are described in [2]. The preliminary design does not include a copy of it because there is no specific change since requirement analysis.

### **2.6 Functionality Requirements**

The functionality requirements are described in [2]. The list of changes to the requirements are listed in the appendix of this document (Appendix A)

### 3 Architecture and Modules

#### 3.1 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.

##### 3.1.1 Structural System Model

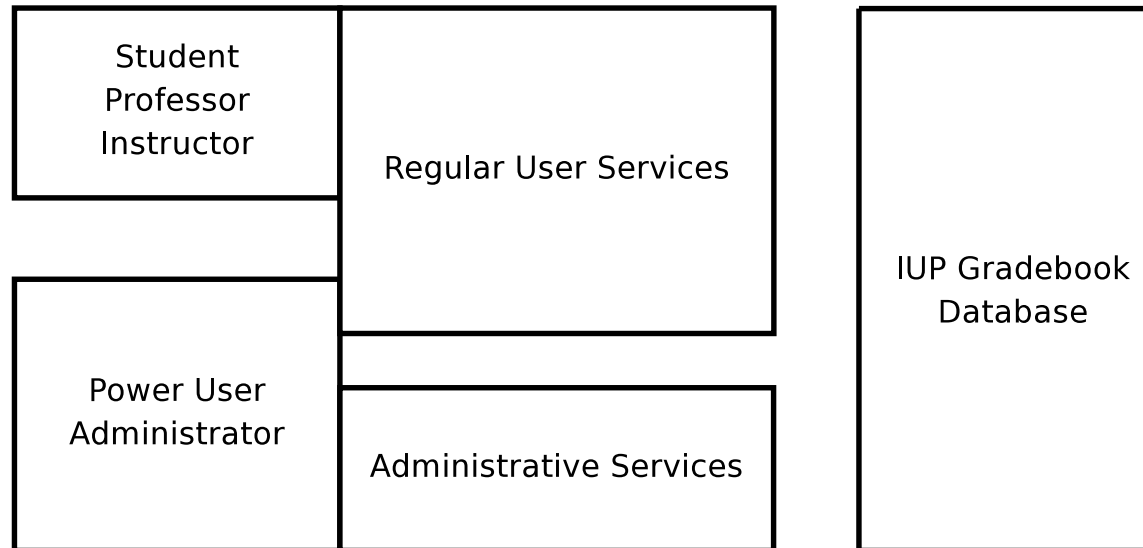


Figure 1: Architecture of the Services

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
- Administrative services are focused towards the departmental staff. The system provides services to enable
  - Managing course hierarchy
  - Managing roster
  - Batch import/export of grades

### 3.1.2 Control Flow Architecture

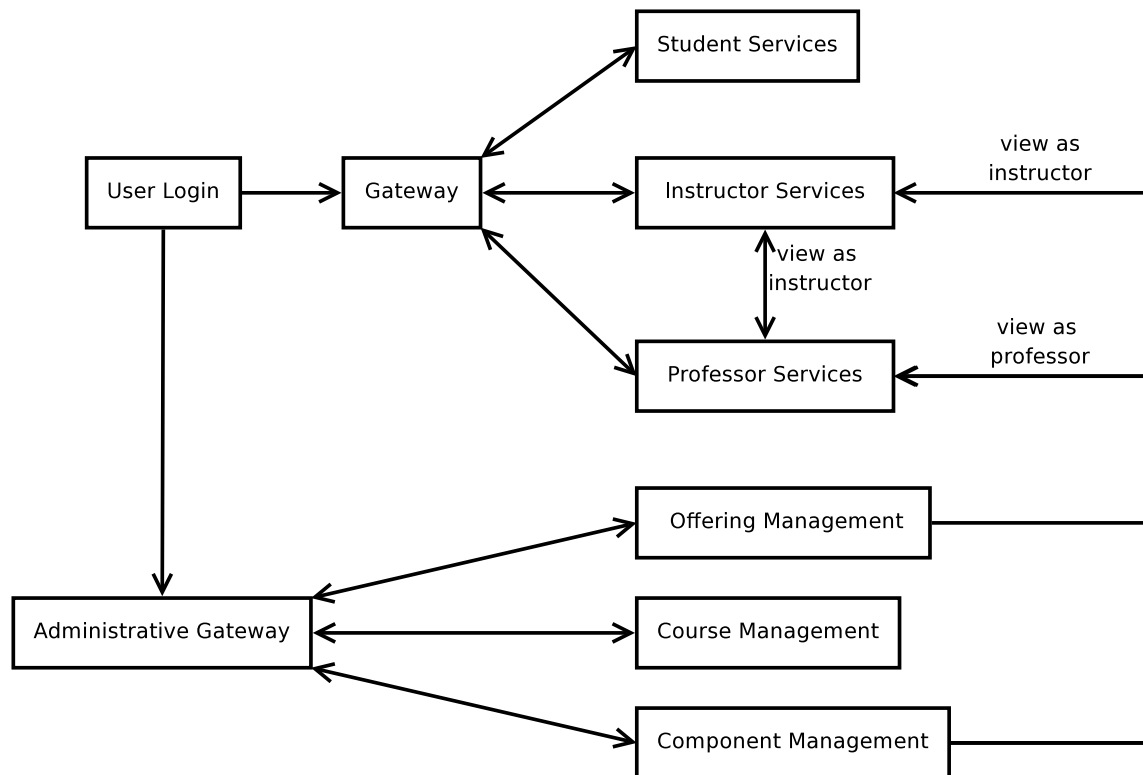


Figure 2: Architecture of the Control Flow

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

## 3.1.3 Component Architecture

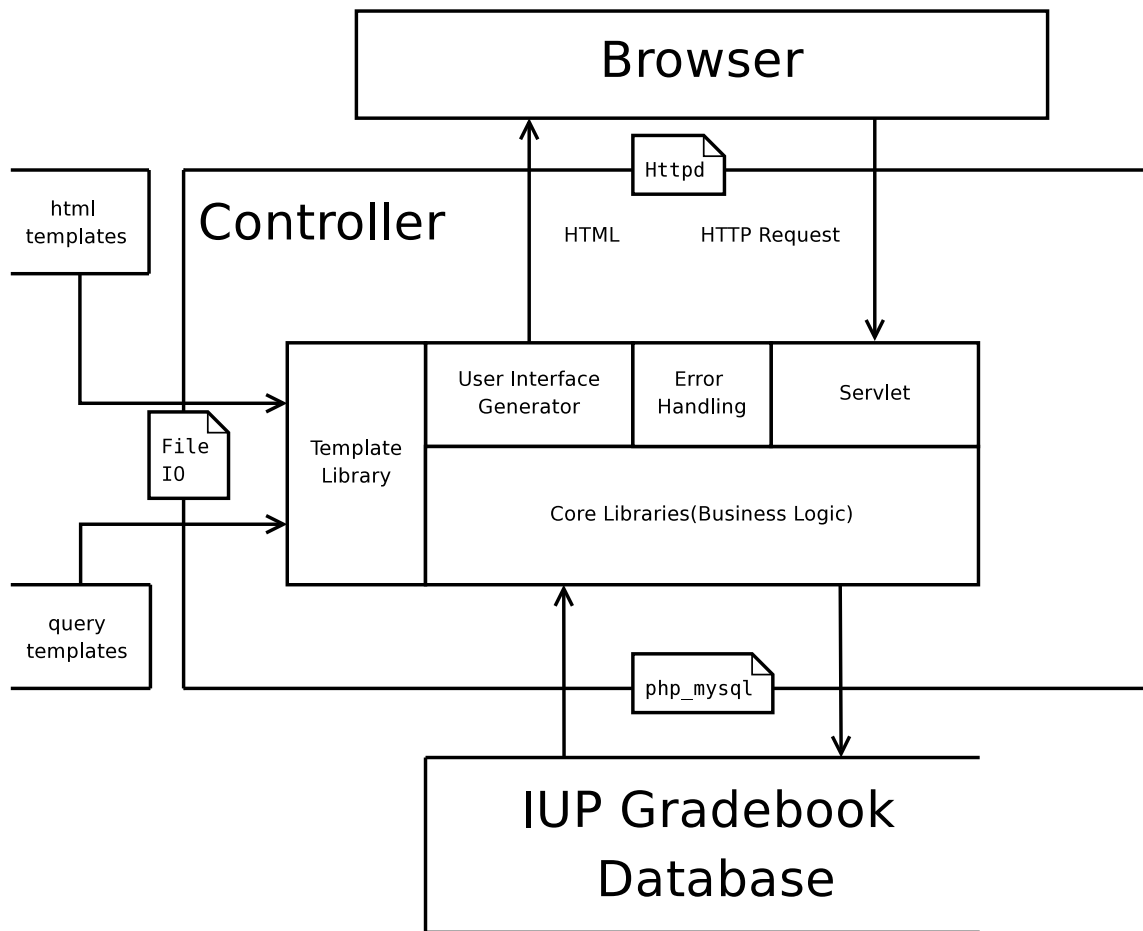


Figure 3: Architecture of the Components

The system is composed of three components, as separated by the big boxes in Figure 3 on page 9:

**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 Gecko, WebKit, or Trident.

**Database** The DBMS system which provides fundamental storage services.

**Controller** The controller contains the business. It is 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.??

The Controller contains five relatively independent modules:

- Template Library
- User Interface Generators
- Servlets
- Error Handling library
- Core Libraries
  - Session Management
  - User and Permissions
  - Data Manipulation

The detailed hierarchy of the module design in the Controller is described in the following sections. ?????????????????????????????????????

## 3.2 Module Coupling

### 3.2.1 Interaction between Modules

The adjacent boundaries of the modules indicate inter-module interactions.

The Template Library is used by the Core Libraries and User Interface Generators to generate the SQL queries and HTML pages respectively. The User Interface Generators and Servlets, where the client business is implemented, are supported by the functionalities of the Core Libraries. The Error Handling library provides a mechanism for error throwing and forwarding between the three libraries.

### 3.2.2 Convention Type Names

PHP Objects are used for communication between modules. These objects are built on-the-fly by associating the property name with property values. Classes are not implemented. Instead, a convention for the property names associated with named objects is set in this section.

The objects postfixed with 'key' should to be sufficient to identify the corresponding entity in the database. A reference to the corresponding table structure in the data design is given when it is appropriate.

**course\_key** An object representing the primary key of the COURSE table in the database design.

Field	Explanation
course_number	Refer to the Data Design

**offering\_key** An object representing the primary key of the OFFERING table in the database design.

Field	Explanation
course_key	The foreign key to the course
professor_id	Refer to the Data Design
semester	Refer to the Data Design

**component\_key** An object representing the primary key of the COMPONENT table in the database design.

**section\_key** An object representing the primary key of the SECTION table in the database design.

Field	Explanation
component_key	The foreign key to the component
section_number	Refer to the Data Design

**person** An object to represent a person. Related to, but not limited to an entry in the PERSON Table.

Field	Explanation
id	The person_id of the person
last_name	Refer to the Data Design
first_name	Refer to the Data Design
email	Refer to the Data Design
cas_account	Refer to the Data Design
iup_account	Refer to the Data Design
active	Refer to the Data Design; can be undefined
grades	An array of grades; See below ????? for the definition of grade; can be undefined

**grade** An object representing an entry in the ASSIGN\_GRADES table.

Field	Explanation
grade	Refer to the Data Design
comment	Refer to the Data Design
timestamp	Refer to the Data Design; can be undefined if it is for the latest grade
assigner	A person object describing the person that assigns the grade; grades and active should be undefined
assignment	An assignment object describing the assignment for which the grade is assigned

**assignment** An object representing an entry in ASSIGNMENT Table.

Field	Explanation
component_key	The foreign key to a component
title	Refer to the Data Design. Also referred as assignment_title
type	Refer to the Data Design

**template** An object representing a template. Created by the Template Library.

Field	Explanation
template_name	The name of the template
result	The result after the template substitution; private member and should not be accessed

**error** An object to encode a error. Created by the Error Handling module.

Field	Explanation
code	The code of the error; a string describing the main feature of the error, in capital letters, and separated by the underscore character
message	The formatted error message
level	The severity of the error; integer; lower the integer, greater is the severity

**result** A data type provided by php\_mysql module to represent the result of an SQL query.

### 3.3 Template Library

**Introduction** The template library is a utility library that build SQL queries or HTML pages from template files.

**Initialization** Find the location for storing template files in the file system.

**Finalization** N/A

#### Interface Design

Name Description IN OUT	template_load Load the template given by template_name into memory and return the template object template_name, type template
Name Description IN OUT	template_get_child Obtain a child template in a template template, child_template_name child_template
Name Description IN OUT	template_reset The template to an unparsed state, invalidate the result field template N/A
Name Description IN OUT	template_assign_prepend Parse the template, prepend template variables with their values; therefore, the template variables are preserved template, array (variable => value) N/A
Name Description IN OUT	template_assign Assign values to the template variables template, array (variable => value) N/A
Name Description IN OUT	template_get_result Obtain the result of the template substitution template string

### 3.4 User Interface Generator

User Interface Generators are server side scripts that generate the HTML files by merging the data obtained from the database and the page templates from the templates. The persistence of sessions and user permissions checking are performed at this level. For a generator, the IN parameters are stored in the HTTP Request header, and the only OUT parameter is the generated HTML page.

Generator Description IN OUT ERROR	Trivial Generator The trivial generator generates pages without accessing the database template_name, (other parameters depending on the template) N/A ACCESS_DENIED, RESOURCE_NOT_FOUND if the template is not found
Generator Description IN OUT ERROR	Gateway Generator It generates the Gateway page. Refer to the User Interface Section N/A N/A N/A
Generator Description IN OUT ERROR	Administrative Gateway Generator It generates the Gateway page for the Administrators. Refer to the User Interface Design N/A N/A ACCESS_DENIED if the use is not a power user or administrator
Generator Description IN OUT ERROR	Modify Grades Generator It generates the Modify Grades pages and View Grade pages. Refer to the User Interface Design offering_key, component_key, section_key, readonly N/A ACCESS_DENIED, RESOURCE_NOT_FOUND
Generator Description IN OUT ERROR	Modify Roster Generator It generates the View Roster page. Refer to the User Interface Design readonly, offering_key, component_key, section_key N/A ACCESS_DENIED, RESOURCE_NOT_FOUND
Generator Description IN OUT ERROR	Manage Course Generator It generates the Course Management page. Refer to the User Interface Design course_key N/A ACCESS_DENIED, RESOURCE_NOT_FOUND
Generator Description IN OUT ERROR	Manage Offering Generator It generates the Offering Management page. Refer to the User Interface Design offering_key N/A ACCESS_DENIED, RESOURCE_NOT_FOUND
Generator Description IN OUT ERROR	Manage Component Generator It generates the Component Management page. Refer to the User Interface Design component_key N/A ACCESS_DENIED, RESOURCE_NOT_FOUND



### 3.5 Servlets

Servlets are server side scripts that respond to the HTTP requests from the client browser but do not directly push any content to the browser. For servlets, the IN parameters are stored in the HTTP request header. They do not have OUT parameters. Rather, they redirect the browser to send requests to other Servlets or User Interface Generators. The persistence of sessions and user permission checking are performed at this level.

Servlet Description	<b>Login Servlet</b> Perform the login actions, validating the user name and password, establish the session
IN	username, password
OUT	N/A
ERROR	INVALID_USERNAME, INVALID_PASSWORD
Servlet Description	<b>CAS Callback Servlet</b> Finish the CAS login process, establish the session
IN	CAS ticket
OUT	N/A
ERROR	

#### 3.5.1 Modify Grades Servlet

Servlet Description	<b>Modify Grades Servlet</b> Modify the grades
IN	array(student with grades)
OUT	N/A
ERROR	ACCESS_DENIED, INVALID_INPUT

#### 3.5.2 Component Management Servlet

Servlet Description	<b>Component Management Servlet</b> Modify the component, adding assignments, etc
IN	component_key, operation, operation_parameters (depending on the operation)
OUT	N/A
ERROR	ACCESS_DENIED, INVALID_INPUT

#### 3.5.3 Course Management Servlet

Servlet Description	<b>Course Management Servlet</b> Modify the course
IN	course_key, operation, operation_parameters (depending on the operation)
OUT	N/A
ERROR	ACCESS_DENIED, INVALID_INPUT

**3.5.4 Offering Management Servlet**

Servlet Description	Offering Management Servlet
IN	Modify the offering, add components, etc offering_key, operation, operation_parameters (depending on the operation)
OUT	N/A
ERROR	ACCESS_DENIED, INVALID_INPUT

**3.5.5 User Management Servlet**

Servlet Description	User Management Servlet
IN	Add/Remove users user_id, operation, operation_parameters (depending on the operation)
OUT	N/A
ERROR	ACCESS_DENIED, INVALID_INPUT

**3.6 Error Handling Module**

**Introduction** The error handling module is the platform to support the error handling policies described in Section . An error typically contains an error code and an error message.

**Initialization** Initialize the error table with error codes and the format of the error messages.

**Finalization** N/A

**Interface Design**

Name	error_new_from_http_request
Description	Create a new error by decoding variables in the HTTP Request
IN	N/A
OUT	error
Name	error_new_from_code
Description	Create a new error by specifying the error code and error parameters
IN	error_code, array(parameters)
OUT	error
Name	error_to_url_postfix
Description	Encode an error to a string that can be postfixed to the URL
IN	error
OUT	string
Name	panic
Description	Stop the execution of current UI Generator or Servlet with the given error message
IN	error_code, array(parameters)
OUT	N/A

### 3.7 Core Libraries

#### 3.7.1 Session Management

**Introduction** The session management module is an extension to the PHP session management library. It overrides the default handlers in the library for starting and ending sessions, storing and removing session variables, and garbage collecting so that these session data are stored into the a database table described in Section 4.3.12.

**Initialization** Setup the PHP session management library to use database for session storage.

**Finalization** N/A

##### Interface Design

Name	session_set_current_person
Description	Store the person id to a session variable
IN	person_id
OUT	N/A
Name	session_get_current_person
Description	Read the person id in current session
IN	N/A
OUT	person_id

#### 3.7.2 User and User Permission Authorization

**Introduction** This module is a collection of functions for user validation and user permission authorization.

**Initialization** N/A

**Finalization** N/A

##### Interface Design

Name	person_is_offering_student
Description	Test if a person is a student in an offering
IN	person_id, offering_key
OUT	boolean
Name	person_is_section_instructor
Description	Test if a person is an instructor of a section
IN	person_id, section_key
OUT	boolean
Name	person_is_offering_professor
Description	Test if a person is a professor for the given offering
IN	person_id, offering_key
OUT	boolean

Name Description IN OUT	person_is_power_user Test if a person is a power user person_id boolean
Name Description IN OUT	person_is_administrator Test if a person is an administrator person_id boolean
Name Description IN OUT	person_authorize_by_iup Authorize a person with its iup account and password iup_account, password(plaintext) person_id or FALSE(if failure)
Name Description IN OUT	person_authorize_by_cas Authorize a person with CAS system; executed asynchronously and a callback servlet has to be specified cas_account, password, url of CAS callback servlet N/A

### 3.7.3 Data Manipulation

**Introduction** This module collects the data manipulation routines. These routines do not validate if the caller has sufficient permissions to perform the given operations. Most of the functions are convenient wrappers over `database_query_from_template` or `database_query_from_string`. Notice that data entry routines are not encapsulated in the module. Data entry is directly resolved in the Servlets via `database_query_from_template`.

**Initialization** Establish a connection to the SQL database and select the database schema.

**Finalization** N/A

#### Interface Design

Name Description IN OUT	database_query_from_template Construct a query by template substitution and execute the query template_name, array(variable => value) result
Name Description IN OUT	database_query_from_string Query the database with an SQL query string string result
Name Description IN OUT	database_start_transaction Start a database transaction N/A N/A

Name Description IN OUT	database_commit_transaction Commit a database transaction N/A N/A
Name Description IN OUT	database_select_students_from_offering Return an array of students enrolled in the given offering offering_key array( person_id => person)
Name Description IN OUT	database_select_students_from_component Return an array of students enrolled in the given component component_key array(person_id => person)
Name Description IN OUT	database_select_students_from_section Return an array of students enrolled in the given section section_key array(person_id => person)
Name Description IN OUT	database_select_assignments_from_component Return an array of assignments in the given component component_key array(assignment_title => assignment)
Name Description IN OUT	database_select_components_from_offering Return an array of components in the given offering offering_key array(component_type => component)
Name Description IN OUT	database_select_sections_from_component Return an array of sections in the given component component_key array(section_number => section)
Name Description IN OUT	database_select_courses Return an array of all courses N/A array(course_number => course)
Name Description IN OUT	database_select_semesters Return an array of all semesters N/A array(semester)
Name Description IN OUT	database_select_offerings_from_course Return an array of all offerings of an course course_key array(semester => offering)
Name Description IN OUT	database_select_offerings_from_professor Return an array of all offerings taught by an professor person_id array(offering)

Name Description IN OUT	database_select_sections_from_instructor Return an array of all sections taught by an instructor person_id array(section_number => section)
Name Description IN OUT	database_select_offerings_from_student Return an array of all offerings in which a student is enrolled. person_id array(offering)
Name Description IN OUT	database_select_grades_from_students_assignments An array of grades associated with the direct product of the students and the assignments array(person), array(assignment) array(person)
Name Description IN OUT	database_select_grade_history_from_student_assignment An array of history of grades associated with a given student for a given assignment person_id, assignment_key array(grade)

## 4 Data Design

### 4.1 Introduction:

A database schema is designed according to the ER model (Refer to Section 2.5) captured in the requirement analysis stage.

The module for data manipulation is defined in 3.7.3.

### 4.2 Table Representation:

### 4.3 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	bigint(20)	ID internally generated by the database engine to uniquely identify each user
role	varchar(20)	Kind of user

Table 1: PERSON Table Fields

#### Indexes

- role in the PERSON table is a foreign key that references the name field 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.
- person\_id in the PERSON table is a primary key that is internally generated by the database engine and it is used to uniquely identify a person in the system.

Keyname	Type	Field
itaccount	UNIQUE	itaccount
iupaccount	UNIQUE	iupaccount
Primary	PRIMARY	person_id
role	INDEX	role

Table 2: PERSON Table Indexes

#### 4.3.1 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_comment	longtext	Course description
course_number	varchar(20)	Uniquely identifies each course

Table 3: COURSE Table Fields

##### Index

- course\_number is the primary key of the COURSE table.

Keyname	Type	Field
Primary	PRIMARY	course_number

Table 4: COURSE Table Indexes

#### 4.3.2 OFFERING Table

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

##### Fields

Field	Datatype	Comment
offering_semester	varchar(20)	Year and semester in which the course is being offered
course_number	varchar(20)	Uniquely identifies a course
professor_id	bigint(20)	ID internally generated by the database engine to identify each professor uniquely

Table 5: OFFERING Table Fields



**Indexes**

- course\_number is a foreign key that references the course\_number field in the COURSE table.
- semester is a foreign key that references the name field in the SEMESTER table.

Keyname	Type	Field
Primary	PRIMARY	offering_semester, course_number, professor_id
course_number	INDEX	course_number
semester	INDEX	offering_semester

Table 6: OFFERING Table Indexes

**4.3.3 SEMESTER Table**

**Introduction** The SEMESTER table contains information regarding the year and the semester in which the course is being offered.

**Fields**

Field	Datatype	Comment
name	varchar(20)	Year and semester in which the course is being offered

Table 7: SEMESTER Table Fields

**Indexes**

Keyname	Type	Field
Primary	PRIMARY	name

Table 8: SEMESTER Table Indexes

**4.3.4 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 it being offered.

**Fields**

Field	Datatype	Comment
course_number	varchar(20)	Uniquely identifies a course
offering_semester	varchar(20)	Year and semester in which the course is being offered
professor_id	bigint(20)	ID internally generated by the database engine to identify each professor uniquely
component_type	varchar(20)	Name of the component in a course

Table 9: COMPONENT Table Fields

**Indexes**

- course\_number, offering\_semester, professor\_id in the COMPONENT table together form a foreign key that references course\_number, offering\_semester, professor\_id fields in the OFFERING table and which together acts as a primary key for the COMPONENT table.
- component\_type in COMPONENT table is a foreign key that references the name field in the COMPONENT\_TYPE table.

Keyname	Type	Field
Primary	PRIMARY	course_number, offering_semester, professor_id
course_number	INDEX	course_number, offering_semester, professor_id
component_type	INDEX	component_type

Table 10: COMPONENT Table Indexes

**4.3.5 COMPONENT\_TYPE Table**

**Introduction** The COMPONENT\_TYPE table contains information regarding the type of component.

**Fields**

Field	Datatype	Comment
name	varchar(45)	Type of component

Table 11: COMPONENT\_TYPE Table Fields

**Indexes**

Keyname	Type	Field
Primary	PRIMARY	name

Table 12: COMPONENT\_TYPE Table Indexes

**4.3.6 COURSE\_DEFAULT\_COMPONENTS Table**

**Introduction** The COURSE\_DEFAULT\_COMPONENTS table contains information regarding default component setup of a course.

**Fields**

Field	Datatype	Comment
course_number	varchar(20)	Uniquely identify a course
component_type	varchar(20)	Type of component

Table 13: COURSE\_DEFAULT\_COMPONENTS Table Fields

### Indexes

- course\_number in COURSE\_DEFAULT\_COMPONENTS table is a foreign key that references the course\_number 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.
- course\_number, component\_type together act as a PRIMARY key for the COURSE\_DEFAULT\_COMPONENTS table.

Keyname	Type	Field
Primary	PRIMARY	course_number, component_type
course_number	INDEX	course_number
component_type	INDEX	component_type

Table 14: COURSE\_DEFAULT\_COMPONENTS Table Indexes

### 4.3.7 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 component final grade.

### Fields

Field	Datatype	Comment
course_number	varchar(20)	Uniquely identifies a course
offering_semester	varchar(20)	Year and semester in which the course being offered
professor_id	bigint(20)	ID internally generated by the database engine to identify each professor uniquely
component_type	varchar(20)	Type of component
student_id	bigint(20)	ID internally generated by the database engine to identify each student uniquely
component_final_grade	decimal(10)	Final component grade assigned to each student
active	tinyint(1)	Hides/unhides a student from the component

Table 15: ENROLL\_COMPONENT Table Fields

### Indexes

- student\_id field in the ENROLL\_COMPONENT is a foreign key that references the person\_id field in the PERSON table.

Keyname	Type	Field
component	INDEX	course_number, offering_semester, professor_id, component_type
student	INDEX	student_id
Primary	PRIMARY	course_number, offering_semester, professor_id, component_type

Table 16: ENROLL\_COMPONENT Table Indexes

### 4.3.8 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_semester	varchar(20)	Year and semester in which the course being offered
course_number	varchar(20)	Uniquely identifies a course
professor_id	bigint(20)	ID internally generated by the database engine to identify each professor uniquely
student_id	bigint(20)	Id internally generated by the database engine to identify each student uniquely
offering_final_grade	varchar(10)	Final component grade assigned to each student
active	tinyint(1)	Hides/unhides a student from the course offering

Table 17: ENROLL\_OFFERING Table Fields

#### Indexes

- offering\_semester, course\_number, professor\_id, student\_id fields together acts as a primary key for the ENROLL\_OFFERING table.
- offering\_semester, course\_number, professor\_id fields together acts as a foreign key for the ENROLL\_OFFERING table that references the offering\_semester, course\_number, professor\_id fields of the OFFERING table.
- student\_id field in the ENROLL\_OFFERING table is a foreign key that references the person\_id field of the PERSON table.

Keyname	Type	Field
Primary	PRIMARY	offering_semester, course_number, professor_id, student_id
offering	INDEX	offering_semester, course_number, professor_id
student	INDEX	student_id

Table 18: ENROLL\_OFFERING Table Indexes

### 4.3.9 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 belongs, the course to which the component belongs, which professor is offering the course and in which semester the course is being offered. It also holds information regarding a student quitting the section. The default value for the start\_timestamp field is CURRENT\_TIMESTAMP.

#### Fields

Field	Datatype	Comment
start_timestamp	timestamp	Records the date and time that the student is enrolled into the section
end_timestamp	timestamp	Records the date and time that student leaves the section. NULL if student is currently part of the section
course_number	varchar(20)	Uniquely identifies a course
offering_semester	varchar(20)	Year and Semester in which the course is being offered
professor_id	bigint(20)	ID internally generated by the database engine to identify each professor uniquely
component_type	varchar(20)	Type of component
section_number	varchar(20)	Section number of the section in a component
student_id	bigint(20)	ID internally generated by the database engine to identify each student uniquely

Table 19: JOIN\_SECTION Table Fields

#### Indexes

- start\_timestamp, course\_number, offering\_semester, professor\_id, component\_type, section\_number, student\_id fields together act as a primary key for the JOIN\_SECTION table.
- start\_timestamp, course\_number, offering\_semester, professor\_id, component\_type, section\_number fields together acts as a foreign key for the JOIN\_SECTION table that references the start\_timestamp, course\_number, offering\_semester, professor\_id, component\_type, section\_number fields of the SECTION table.
- student\_id field is a foreign key of the JOIN\_SECTION table that references the person\_id field of the PERSON table.

Keyname	Type	Field
Primary	PRIMARY	start_timestamp,course_number, offering_semester, professor_id, component_type, section_number, student_id
section	INDEX	start_timestamp,course_number, offering_semester, professor_id, component_type, section_number
student	INDEX	student_id

Table 20: JOIN\_SECTION Table Indexes

#### 4.3.10 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_number	varchar(20)	Section number of the section in a component
component_type	varchar(20)	Type of component
course_number	varchar(20)	Uniquely identifies a course
offering_semester	varchar(20)	Year and semester in which the course being offered
professor_id	bigint(20)	ID internally generated by the database engine to identify each professor uniquely

Table 21: SECTION Table Fields

##### Indexes

- section\_number, component\_type, course\_number, offering\_semester, professor\_id fields together act a primary key for the SECTION table.
- component\_type, course\_number, offering\_semester, professor\_id fields together acts as a foreign key for the SECTION table that references component\_type, course\_number, offering\_semester, professor\_id fields of the COMPONENT table.
- section\_number is also made unique because a course can have a component without sections in it.

Keyname	Type	Field
Primary	PRIMARY	section_number, component_type, course_number, offering_semester, professor_id
component	INDEX	component_type, course_number, offering_semester, professor_id
section_number	UNIQUE	section_number

Table 22: SECTION Table Indexes

#### 4.3.11 TEACH\_SECTION Table

**Introduction** The TEACH\_SECTION table contains information regarding the instructor such as the section to which the instructor is assigned, the component to which the section belongs, the course to which the component belongs, which professor is offering the course and in which semester it is being offered.

##### Fields

Field	Datatype	Comment
course_number	varchar(20)	Uniquely identifies a course
offering_semester	varchar(20)	Year and semester in which the course being offered
professor_id	bigint(20)	ID internally generated by the database engine to identify each professor uniquely
component_type	varchar(20)	Type of component
section_number	varchar(20)	Section number of the section in a component
instructor_id	bigint(20)	ID internally generated by the database engine to identify each instructor uniquely

Table 23: TEACH\_SECTION Table Fields

##### Indexes

- course\_number, offering\_semester, professor\_id, component\_type, section\_number, instructor\_id fields together acts as a primary key for the table TEACH\_SECTION.
- course\_number, offering\_semester, professor\_id, component\_type, section\_number fields together is a foreign key for the table TEACH\_SECTION which references the course\_number, offering\_semester, professor\_id, component\_type, section\_number fields of the SECTION table.
- instructor\_id field is also a foreign key for the table TEACH\_SECTION which references the person\_id field of the PERRSON table.

Keyname	Type	Field
primary	PRIMARY	course_number, offering_semester, professor_id, component_type, section_number, instructor_id
section_number	INDEX	course_number, offering_semester, professor_id, component_type, section_number,
instructor_id	INDEX	instructor_id

Table 24: TEACH\_SECTION Table Indexes

#### 4.3.12 SESSIONS Table

**Introduction** The SESSION table stores information regarding each session.

**Fields**

Field	Datatype	Comment
session	varchar(255)	name of the session
session_expires	int(10)	the time when the session expires
session_data	text	data what session would contain

Table 25: SESSIONS Table Fields

**Indexes** session field in the SESSION table is a primary key.

Keyname	Type	Field
Primary	PRIMARY	session

Table 26: SESSIONS Table Indexes

**4.3.13 ASSIGN\_GRADES Table**

**Introduction** The ASSIGN\_GRADES table contains information regarding grades assigned to students for different assignments in a section by instructors belonging to that section. The default value of the grade\_timestamp field is CURRENT\_TIMESTAMP.

**Fields**

Field	Datatype	Comment
teacher_id	bigint(20)	ID internally generated by the database engine
student_id	bigint(20)	ID internally generated by the database engine to uniquely identify each student
course_number	varchar(20)	Uniquely identifies a course
offering_semester	varchar(20)	Year and semester in which the course being offered
professor_id	bigint(20)	ID internally generated by the database engine to uniquely identify each professor
component_type	varchar(20)	Type of component
assignment_title	varchar(100)	Title of the assignment
grade_timestamp	timestamp	Records timestamp of the submission of the grade
grade	varchar(10)	Grade assigned to a given student for an assignment
grade_comments	text	Comments related to the grade for each student

Table 27: ASSIGN\_GRADES Table Fields

**Indexes**

- teacher\_id , student\_id, course\_number, offering\_semester, professor\_id , component\_type, assignment\_title, grade\_stamp fields together act as a primary key for the ASSIGN\_GRADES table.



- course\_number, offering\_semester, professor\_id, component\_type, assignment\_title fields together act as a foreign key of the ASSIGN\_GRADES table that references the offering\_semester, professor\_id, component\_type, assignment\_title fields of the ASSIGNMENT table.
- teacher\_id is a foreign key for the ASSIGN\_GRADES table that references the person\_id field in the PERSON table.
- student\_id is a foreign key for the table ASSIGN\_GRADES that references the person\_id field in the PERSON table.

Keyname	Type	Field
Primary	PRIMARY	teacher_id, student_id, course_number, offering_semester, professor_id, component_type, assignment_title, grade_timestamp
teacher_id	INDEX	teacher_id
student_id	INDEX	student_id
assignment	INDEX	course_number, offering_semester, professor_id , component_type, assignment_title

Table 28: ASSIGN\_GRADES Table Indexes

#### 4.3.14 LATEST\_ASSIGNED\_GRADES Table

**Introduction** The LATEST\_ASSIGN\_GRADES 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	bigint(20)	ID internally generated by the database engine
student_id	bigint(20)	ID internally generated by the database engine
course_number	varchar(20)	Uniquely identifies a course
offering_semester	varchar(20)	Year and semester in which the course is being offered
professor_id	bigint(20)	ID internally generated by the database engine
component_type	varchar(20)	Type of component
assignment_title	varchar(100)	Name of the assignment
grade	varchar(10)	Latest grade assigned to a student for an assignment

Table 29: LATEST\_ASSIGNED\_GRADES Table Fields

##### Indexes

- teacher\_id , student\_id, course\_number, offering\_semester, professor\_id , component\_type, assignment\_title fields together act as a primary key for the LATEST\_ASSIGN\_GRADES table.

- course\_number, offering\_semester, professor\_id, component\_type, assignment\_title fields together act as a foreign key in the LATEST\_ASSIGN\_GRADES table that references offering\_semester, professor\_id, component\_type, assignment\_title fields of the ASSIGNMENT table.
- teacher\_id is a foreign key for the LATEST\_ASSIGN\_GRADES table that references the person\_id field of the PERSON table.
- student\_id is a foreign key for the LATEST\_ASSIGN\_GRADES table that references the person\_id field of the PERSON table.

Keyname	Type	Field
Primary	PRIMARY	teacher_id, student_id, course_number, offering_semester, professor_id, component_type, assignment_title
teacher_id	INDEX	teacher_id
student_id	INDEX	student_id

Table 30: LATEST\_ASSIGNED\_GRADES Table Indexes

#### 4.3.15 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
name	varchar(20)	Type of user

Table 31: ROLE\_TYPE Table Fields

##### Indexes

- name field is a primary key for the ROLE\_TYPE table.

Keyname	Type	Field
Primary	PRIMARY	name

Table 32: ROLE\_TYPE Table Indexes

#### 4.3.16 ASSIGNMENT\_TYPE Table

**Introduction** The ASSIGNMENT\_TYPE table contains information regarding the type of assignment such as clicker questions and home work.

Field	Datatype	Comment
name	varchar(20)	Type of assignment

Table 33: ASSIGNMENT\_TYPE Table Fields

**Indexes**

- name field is a primary key for the table ASSIGNMENT\_TYPE table.

Keyname	Type	Field
Primary	PRIMARY	name

Table 34: ASSIGNMENT\_TYPE Table Indexes

**4.3.17 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
course_number	varchar(20)	Uniquely identifies a course
offering_semester	varchar(20)	Year and semester in which the course being offered
professor_id	bigint(20)	ID internally generated by the database engine to uniquely identify each professor
component_type	varchar(20)	Type of component
assignment_title	varchar(100)	Name of the assignment
assignment_type	varchar(20)	Type of assignment

Table 35: ASSIGNMENT Table Fields

**Indexes**

- course\_number, offering\_semester, professor\_id, component\_type, assignment\_title fields together act as a primary key for the ASSIGNMENT table.
- course\_number, offering\_semester, professor\_id, component\_type fields together act a foreign key of the ASSIGNMENT table that references the course\_number, offering\_semester, professor\_id, component\_type fields of the COURSE table.
- assignment\_type field is a foreign key of the ASSIGNMENT table that references the name field in the ASSIGNMENT\_TYPE table.

Keyname	Type	Field
Primary	PRIMARY	course_number, offering_semester, professor_id, component_type, assignment_title
COURSE_FK_INDEX	INDEX	course_number, offering_semester, professor_id, component_type
ASSIGNMENT_TYPE_INDEX	INDEX	assignment_type

Table 36: ASSIGNMENT Table Indexes

## 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 34, Figure 5 on page 34, Figure 6 on page 35 and Figure 7 on page 35 show the menu hierarchy for different users in the system. Clearly, we pick an entity first and then an operation for that entity.

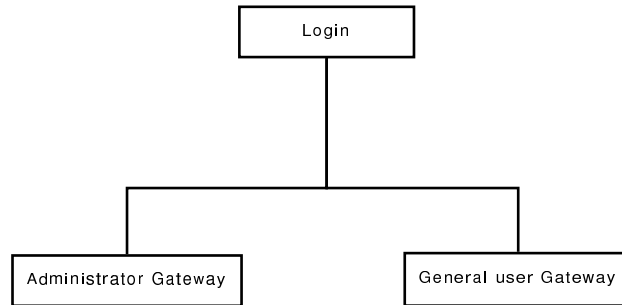


Figure 4: Menu Hierarchy at Login

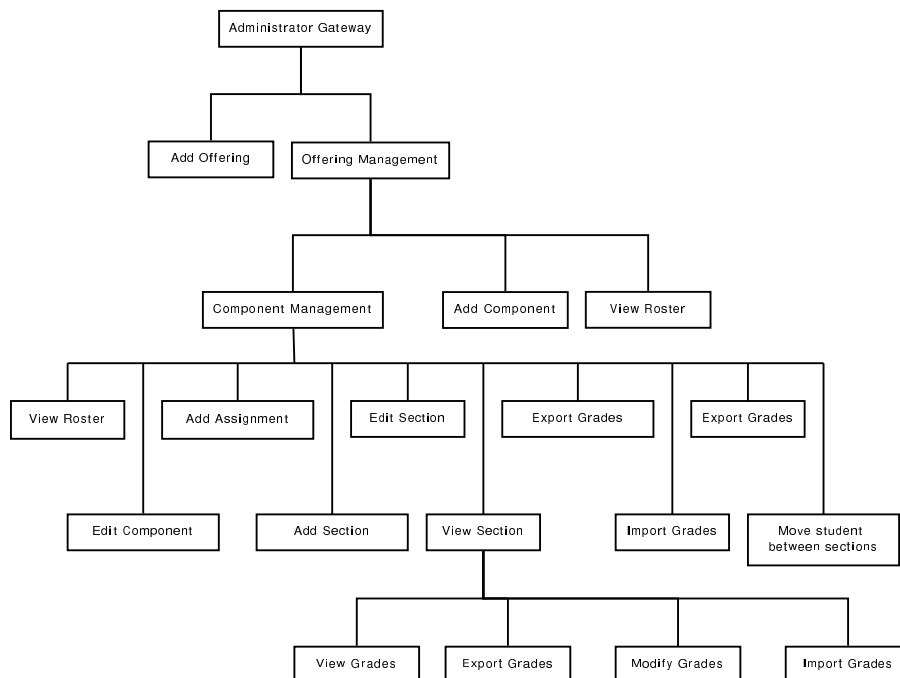


Figure 5: Menu Hierarchy for Administrator

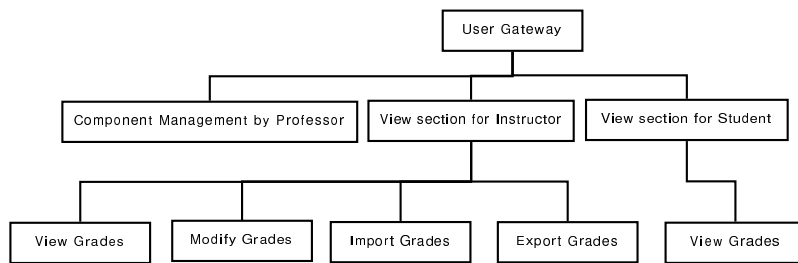


Figure 6: Menu Hierarchy for general users

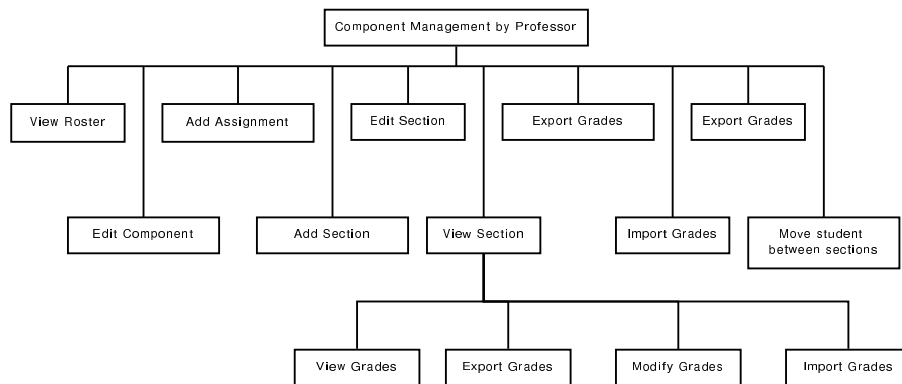


Figure 7: Menu Hierarchy for Professor

## 5.1 Administrator

### 5.1.1 Login

This screen enables a user to log in to the system.

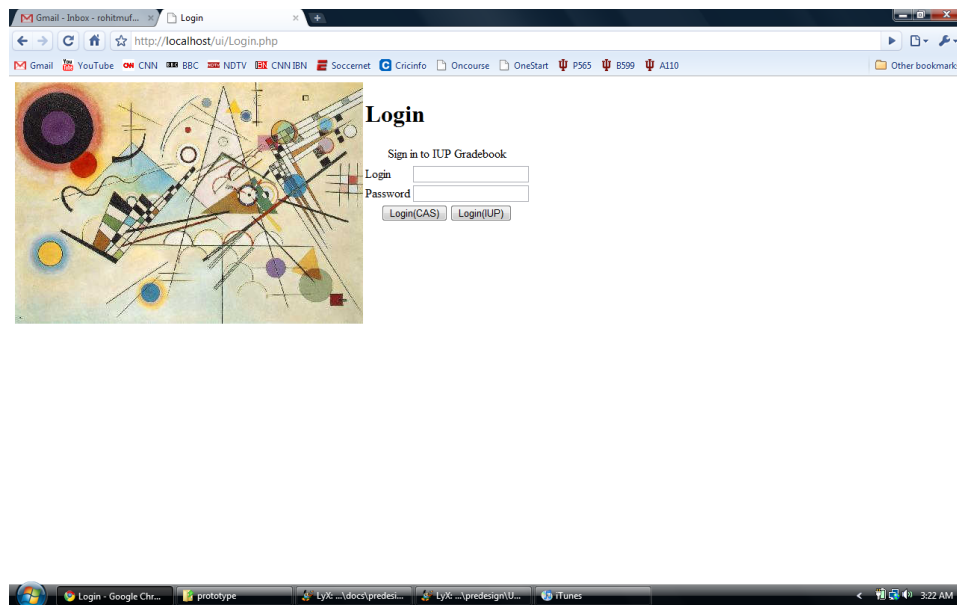


Figure 8: Login screen

**Operation:**

The user types in the username and password and then hits the appropriate login button. A user with an IU username and password hits the CAS login button and all other users hit the IUP login button.

**Navigation:**

A correct username and password takes the user to either the 5.1.2 or the ???. 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.

**Appearance and Layout:****5.1.2 Administrator Gateway****Operations:**

- Add Offering
- View Offering

**Navigation:**

- The administrator clicks on the Add Offering hyperlink to go to the Add Offering screen.
- The administrator clicks on a course offering title hyperlink to go to the Offering Management screen.

### 5.1.3 Add Offering

#### Operation:

The administrator enters the course offering title in the text box provided.

#### Navigation:

The administrator hits the Submit button and returns to the Administrator Gateway screen.

### 5.1.4 Offering Management

#### Operations:

- Edit Offering
- Add Component
- Component Management
- View Component as Professor
- Add student
- Move student between sections
- Export Grades
- Import Grades

#### Navigation:

Event	Action
Click Export Grades	Go to Export Grades screen
Click Import Grades	Go to Import Grades screen
Click Hidden checkbox in Edit Offering	N/A
Click Submit button in Edit Offering	Modify existing offering
Click Component Hyperlink	Go to ???
Click Add Hyperlink in Components	Go to ???
Click View as Professor	Go to ???
Click Add Hyperlink in Students	Go to ???
Click checkbox under Hidden column in Students	N/A
Click Submit button in Students	Hide student from roster of the offering

Table 38: Offering Screen for Administrator



### 5.1.5 Add Component to Course Offering

#### Operation:

The administrator enters the component type in the text box provided.

#### Navigation:

The administrator hits the Submit button and returns to the Offering Management screen.

### 5.1.6 Component Management



Figure 9: Component Management screen

#### Operations:

???

#### Navigation:

???

### 5.1.7 Add Section to a Component

#### Operation:

The administrator enters the section name in the text box provided.

**Navigation:**

The administrator hits the Submit button and returns to the Component Management screen.

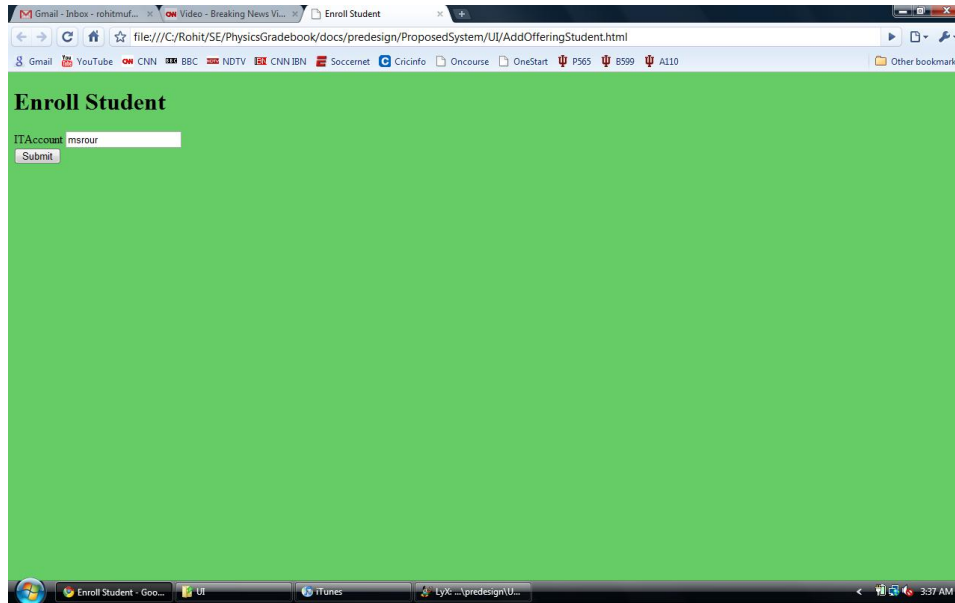
**5.1.8 Add Student to a Course Offering**

Figure 10: Add Student screen

**Operation:**

The administrator enters the section name in the text box provided.

**Navigation:**

The administrator hits the Submit button and returns to the Component Management screen.

### 5.1.9 Add Assignment to a Component

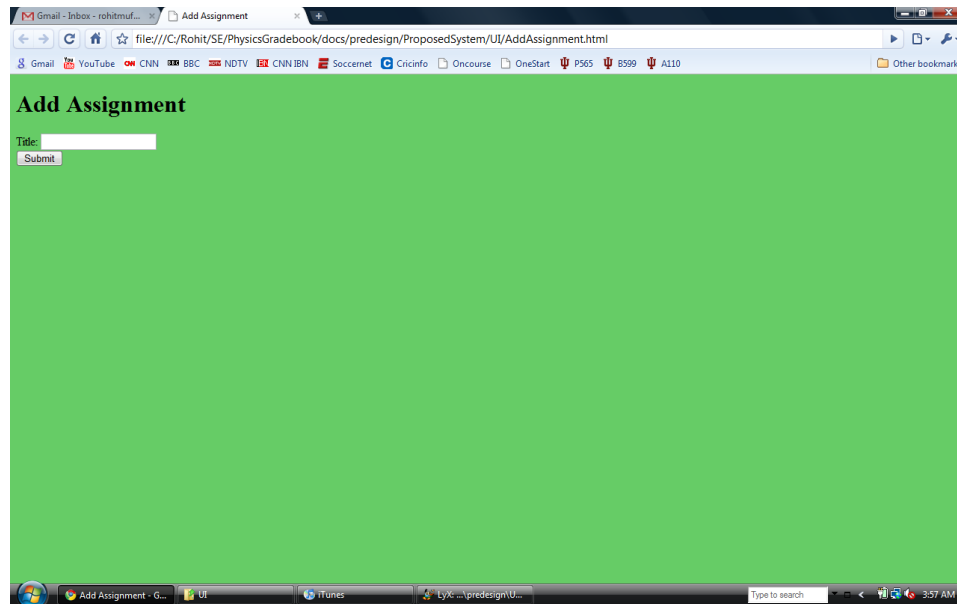


Figure 11: Add Assignment screen

**Operation:**

The administrator enters the assignment title in the text box provided.

**Navigation:**

The administrator hits the Submit button and returns to the Component Management screen.

### 5.1.10 Edit Assignment in a component

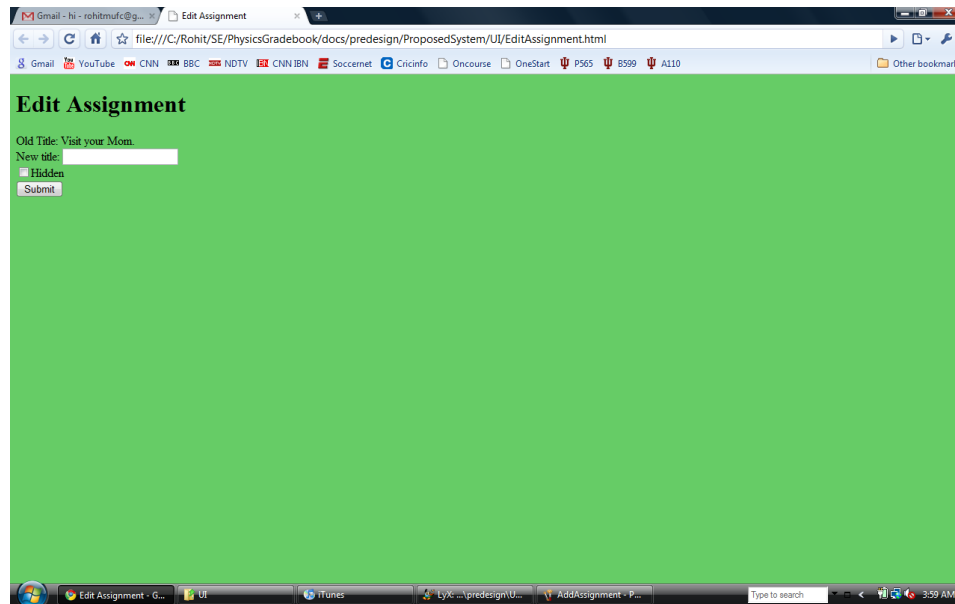


Figure 12: Edit Assignment screen

**Operation:**

The administrator enters the new assignment title in the text box provided.

**Navigation:**

The administrator hits the Submit button and returns to the Component Management screen.

### 5.1.11 Edit Component

**Operation:**

The administrator enters the new component type in the text box provided.

**Navigation:**

The administrator hits the Submit button and returns to the Component Management screen.

### 5.1.12 Edit Course

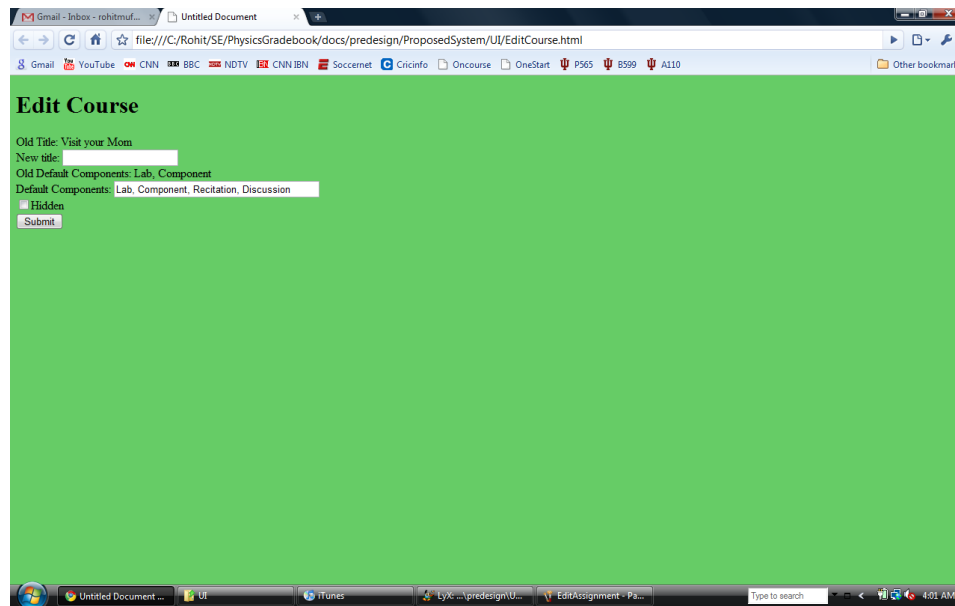


Figure 13: Edit Course screen

**Operation:**

The administrator enters the new course title in the New Title text box and also provides the new default components of the course in the Default Components text box.

**Navigation:**

The administrator hits the Submit button and returns to the Component Management screen.

### 5.1.13 Export Grades

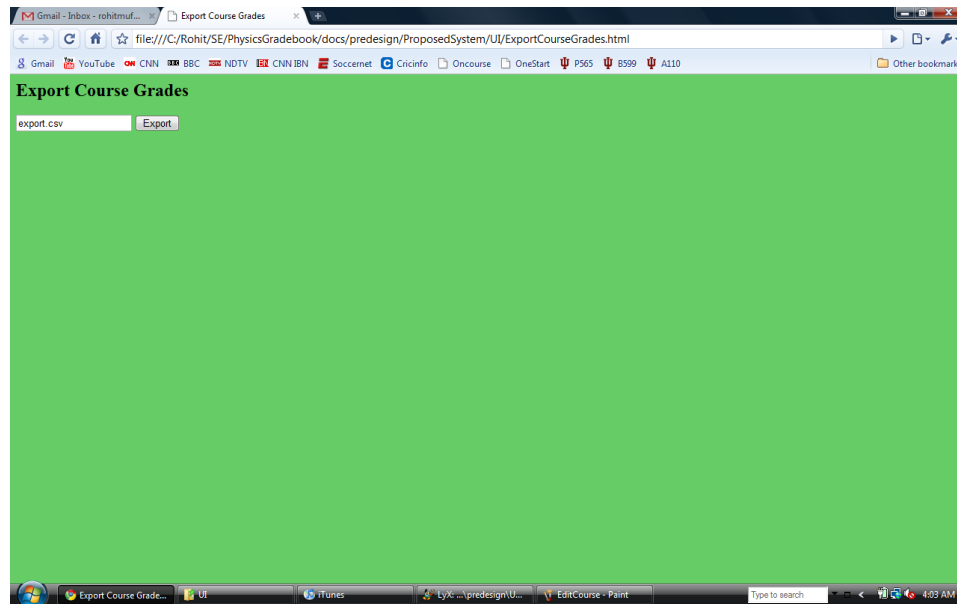


Figure 14: Export Course Grades screen

**Operation:**

The administrator hits on the browse button to chose the file that need to be exported.

**Navigation:**

The administrator hits the Export button to start exporting and returns back to Referer

## 6 Exception Handling

### 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 sections. Usually, a platform failure should stop the system from working, and until the cause of the failure is solved 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 solution 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

## 6.3 Error Handling Schema

The above diagram explains that the errors are thrown by User Interface Generator and servlets and they are handled by the functions defined in the Error Handling Module in Section 3.6.

Refer to Error Handling Module in the ??



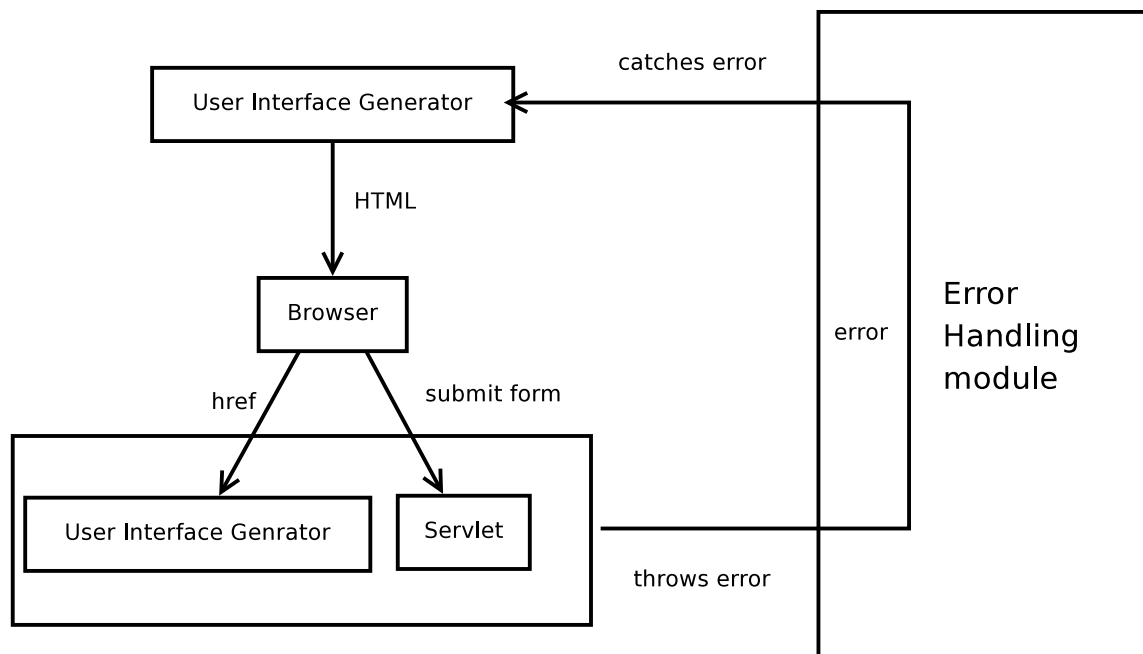


Figure 15: Error Handling Schema

## 7 Test Plan

Developed from the requirement specification document[2], the testing scope outlined below describes the types of testing that will be performed on the proposed information system. For each testing type, a policy for performing the test is given.

### 7.1 Unit Testing

The objective of unit testing is to verify that the individual units of source code are working properly. In this project, a unit is equivalent to a function in a module. Considered as a white box testing technique, unit testing is generally done before integration testing.

**Policy** This testing will be carried out before the code enters the library. PHPUnit will be used to perform the unit testing.

### 7.2 Security Testing

The objective of security testing is to test factors such as confidentiality, integrity, authentication, authorization, availability and non-repudiation of the system. The client specifically mentioned about the safety of the system from external hackers; to ensure this safety, the team will perform regular security testings with standard tools.

**Policy** This testing will be carried out after the implementation phase. Nikto is used to perform the security testing.

### 7.3 Functionality Testing

The objective of the functionality testing is to verify whether the system meets the functionality requirements specified in the requirement specification document[2].

**Policy** Functionality testing is performed after the entire system is implemented. Open source tool Selenium is used to carry out the functionality testing.

### 7.4 Integration Testing

This test proves that all components of the system interface with each other correctly without gaps in the data flows. The final integration testing will prove that system works as an integrated unit when all the bugs are fixed.

**Policy** Integration testing is performed after the libraries are frozen, when no more unit testing is required.

### 7.5 User Acceptance Testing

This test ensures that the system operates in the expected manner, and any supporting materials such as procedures, forms are accurate and suitable for the intended purpose. This testing also ensures that there are no gaps between the functionality of the entire system and the user's conceptions.

**Policy** User acceptance testing is performed after the integration testing.

## 7.6 Performance Testing

These tests ensure that the response time of the system is acceptable. The proposed system will have multiple users using the system simultaneously. The system should respond within acceptable time to all the users under an average load.

**Policy** Performance testing is performed after the integration testing. Open source tool OpenWebLoad is used to carry out the Performance testing.

## 7.7 Regression Testing

The objective of regression testing is to ensure that new functionality and improved stability of the software does not compromise the existing functionalities.

**Policy** Regression testing is not carried out because there is only one release cycle.

## 7.8 Multi-User Testing

Multi-user testing will attempt to prove that it is possible for an acceptable number of users to work with the system at the same time. The objective of the test is to break the system.

**Policy** Multi-user testing is performed after the integration testing.

## 7.9 Usability Testing

The objective of this testing is to ensure the user friendliness of the screens. As there are several roles for the users of the system, the testing should be performed with respect to the user interfaces for various roles in order to ensure the quality of the system.

**Policy** This testing is performed through all stages of the development.

## 7.10 Operation Acceptance Testing

The objective of this testing is to ensure that processes and procedures are in place to allow the entire system to be used and maintained.

**Policy** This testing phase is to be performed prior to deploying the system to a live site.

## 7.11 Database testing

The objective of this testing is to ensure whether the queries are retrieving data from the database as expected.

**Policy** This testing is performed after the database and all the queries that the project requires are created. This is done manually.

## 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>

## Nomenclature

**banner** The picture to indicate the existence of the system.

**Camel Case** A mixture of upper and lower case letter, e.g GObjectClass.

**course hierarchy** Each year, a course is offered in an Offering. An Offering consists of Component(s), and students in each component were divided into Section.

**Gecko** Mozilla web browser engine.

**Nikto** Nikto is a tool for finding default web files and examining web server and CGI security.  
Reference link:<http://www.cirt.net/nikto2>

**OpenWebLoad** OpenWebLoad is a tool for load testing web applications. It aims to be easy to use and providing near real-time performance measurements of the application under test.  
Reference link:<http://openwebload.sourceforge.net/>

**PHPUnit** It is a member of the xUnit family of testing frameworks and provides both a framework that makes the writing of tests easy as well as the functionality to easily run the tests and analyse their results.  
reference link:<http://www.phpunit.de/manual/3.3/en/index.html>

**Selenium** Selenium is a suite of tools to automate web app testing across many platforms.  
Reference link:<http://selenium.seleniumhq.org/>

**Servlets** Servlets are server side scripts that respond to the HTTP requests from the client browser but do not directly push any content to the browser.

**Session** A session is a semi-permanent interactive information exchange, between two or more communicating devices, or between a computer and user.

**Trident** known as MSHTML, the html layout engine from Microsoft.

**WebKit** Google Chrome and Safari are based on WebKit browser engine.

**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. eg '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 Coding Standards and Conventions

### B.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.

### B.2 Modules

### B.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 namespace 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.

### B.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.

#### B.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 ("\_").

### B.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 (“\_”).

### B.4.3 CSS Headers

- CSS headers are included statically in the HTML pages.

## B.5 Commenting and Indentation

### B.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.

### B.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-DOC 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.
*
*****/

```

### B.5.3 In-line Commenting

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

- '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;
....
....

```

### B.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 */
    .....
}

```

```

        . . . . .
    }

```

### B.5.5 Line Breaking Rules

- Between comment header and member functions there should be no blank lines.
- Between inline 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.

### B.5.6 Blank/Space Rules

- Operators and operands are generally separated by a space (ASCII 32).
- Commas should be followed by a space.
- Avoid continues 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 (“{”).

## B.6 Naming Conventions

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

### B.7 Database Schema

- Table names should be in upper case. Multiple words are separated by underscores (“\_”).
- Table column names should be in lower case. Multiple words are separated by underscores (“\_”). Table name should be repeated if possible.

### B.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, e.g., ‘sql\_template\_manager.php’.
- 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'.

### B.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";  
}
```

### B.10 Global Variables

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

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

### B.11 User Interface

User interface modules require special standardization.

### B.12 Form Layout

- The banner 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.

### B.13 Form Navigation

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

### **B.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.

### **B.15 Key Bindings**

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