Software Design Document

Nefertari School Management System

Amira Galal, Farah Hisham, Menna Mohamed, Sara Hassan

May 15, 2018

1 Introduction

1.1 Purpose

This standard is intended for technical stakeholders (headmistress), developers and chef executive managers. It will guide a developer with the structure of the system and its components.

1.2 Scope

Upon meeting the stakeholders of Nefertari Schools, our team has decided on the purpose and functionality of the system to be developed and delivered. The Nefertari School Management System will allow its users to store any data concerned with the school in the database, and will also provide a website serving several functionalities. The system offers a better structured database for the data to be stored in, as well as an interactive website which will enhance the professional relationship between the parents and the school's administration and allows the contribution of the students and teachers.

1.3 Overview

This document is to be used by the developers to guide them through the development of the system. All the requirements of the system, user interfaces and diagrams will be included and the developers shall apply it.This document is mainly concerned with design and structure of the system.

1.4 Definitions and Acronyms

This section is optional. Provide definitions of all terms, acronyms, and abbreviations that might exist to properly interpret the SDD. These definitions should be items used in the SDD that are most likely not known to the audience.

Term	Definition
Model View Controller (MVC)	An architecture used to design the system.
Entity Attribute Value (EAV)	A system architecture used to design the database.

2 System Overview

Since every functionality in the system has several different views, we used the MVC architecture to implement in the system. The majority of the functions are sequential; each action depending on an another action before.

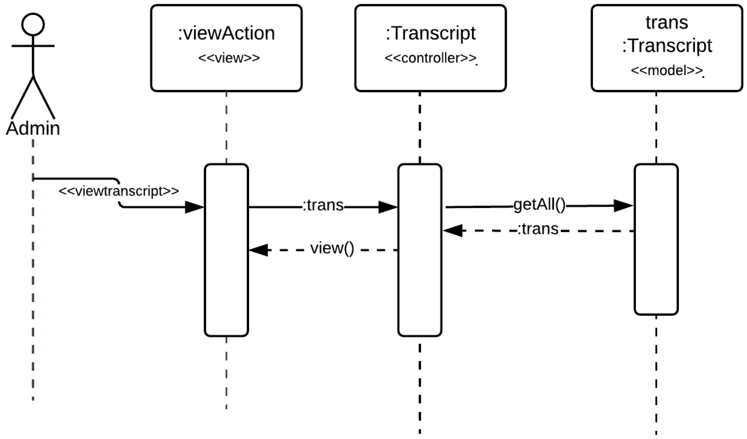


Figure 1: Transcript: View

3 System Architecture

3.1 Architectural Design

The architecture used for the entire system is the MVC. Each functionality in the system has multiple views and interacts with several models, so none of the functionalities look exactly the same. The MVC Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction and passes these interactions to the View and the Model.

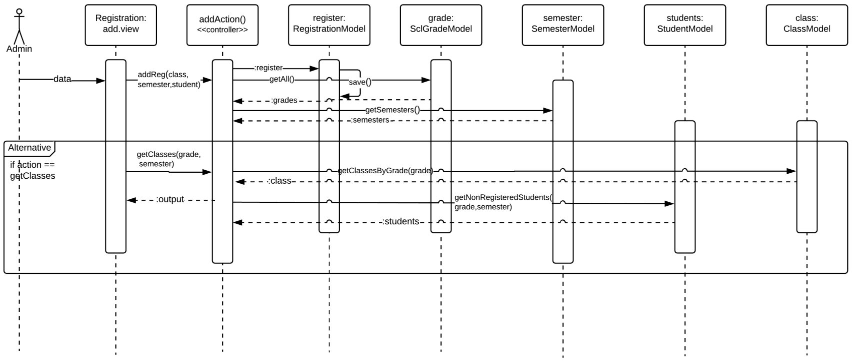


Figure 2: Registering a student

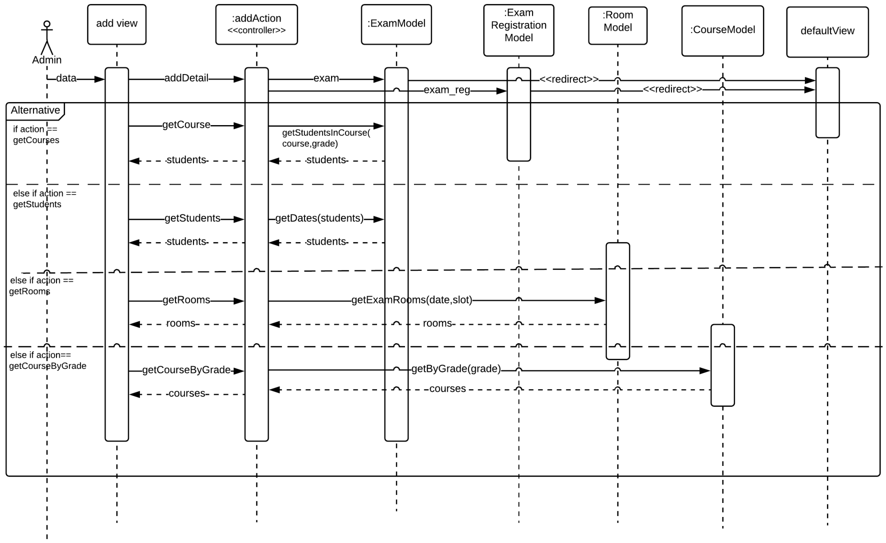


Figure 3: Adding a new exam

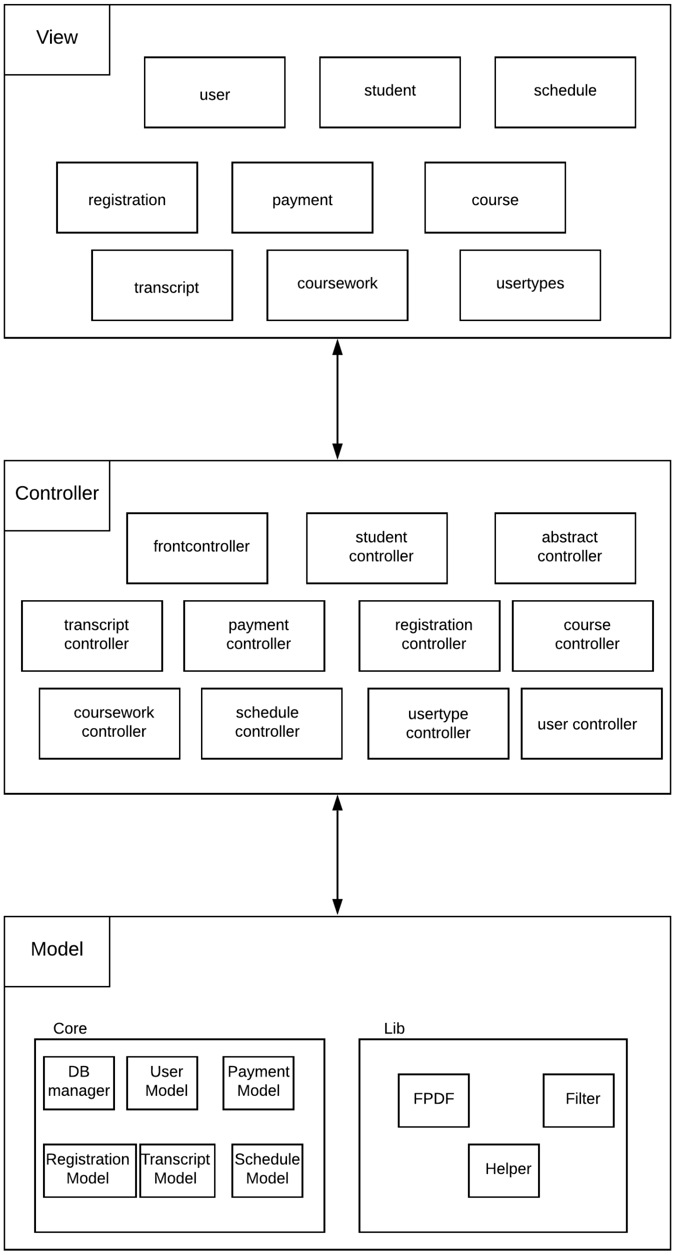

View
user
student
schedule
registration
payment
course
transcript
coursework
usertypes
Controller
frontcontroller
student controller
abstract controller
transcript controller
payment controller
registration controller
course controller
coursework controller
schedule controller
usertype controller
user controller
Model
Core
DB manager
User Model
Payment Model
Registration Model
Transcript Model
Schedule Model
Lib
FPDF
Filter
Helper

Figure 4: MVC

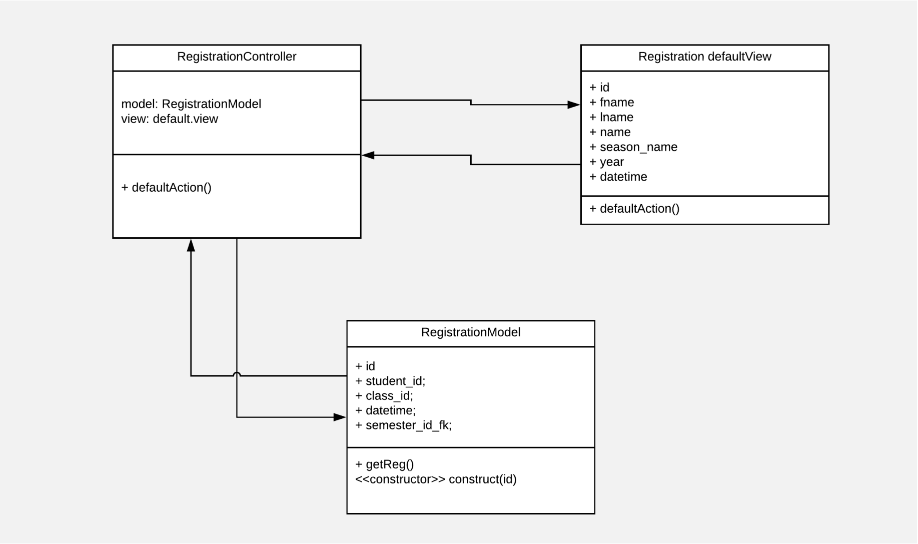


Figure 5: Registration: View

3.1.1 View

The registration's default view displays the student's ID and name, class name, semester and date/time. An object is created in the view, which is sent from the controller.

3.1.2 Controller

Each controller extends from the abstract controller. The registration controller has a function defaultView() which creates an object from the registration model, which fetches the information needed to display in the view using the getReg() function in the model. The controller is the connection between the view and the model.

3.1.3 Model

The registration model contains the attributes of the corresponding table in the database, and all the functions. The function getReg() executes a query and fetches the data from the database. An object from the model is created in the controller and the function is called.

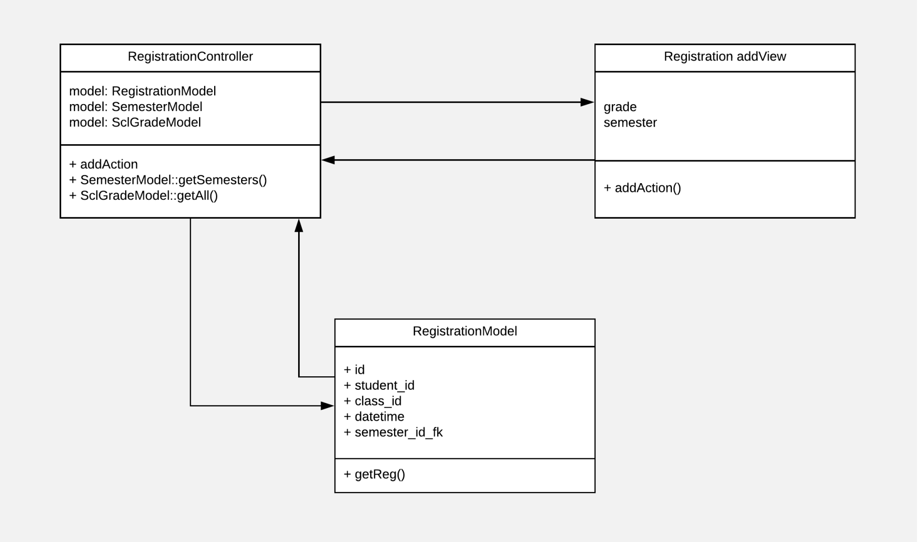


Figure 6: Registration: Add View

3.2 Decomposition Description

Several design patterns were used in this system, each used to implement a different functionality.

3.2.1 Singleton Pattern

Singleton design pattern was used in the database connection handler. The singleton pattern allows access to one and only one instance of a particular class. This is useful when exactly one object is needed.

3.3 Decorator Pattern

The decorator pattern provides a flexible alternative to subclassing for extending functionality. We used the decorator pattern for the payment function. In the very likely case of adding a new payment method, we would only add it as a new class open for extending but not open for modification.

3.4 Strategy Pattern

The strategy pattern allows to encapsulate behaviors inside a class and inherit them, which is better than rewriting the same function again in different classes. This pattern was used in adding forms.

3.4.1 Front Controller

The front controller pattern allowed us to have a single entrance point for our web application (e.g. index.php) that handles all of the requests. This code is responsible for loading all of the dependencies, processing the request and sending the response to the browser.

3.4.2 Class Diagram

Figure 7 shows the components that are used to make up the user.

Figure 8,9 and 10 shows how we implemented the decorator design pattern in the payment class The iPay is an interface, which the ExtraFeesModel inherits from, which is an abstract model. The DecoratorModel and DecoratorPriceModel define the type of decorator and its price.

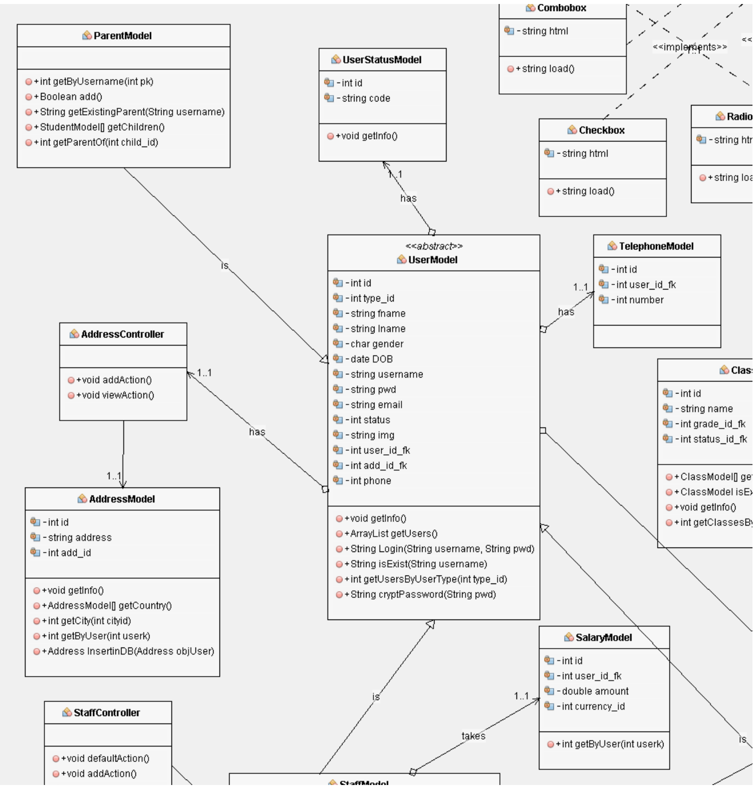


Figure 7: User

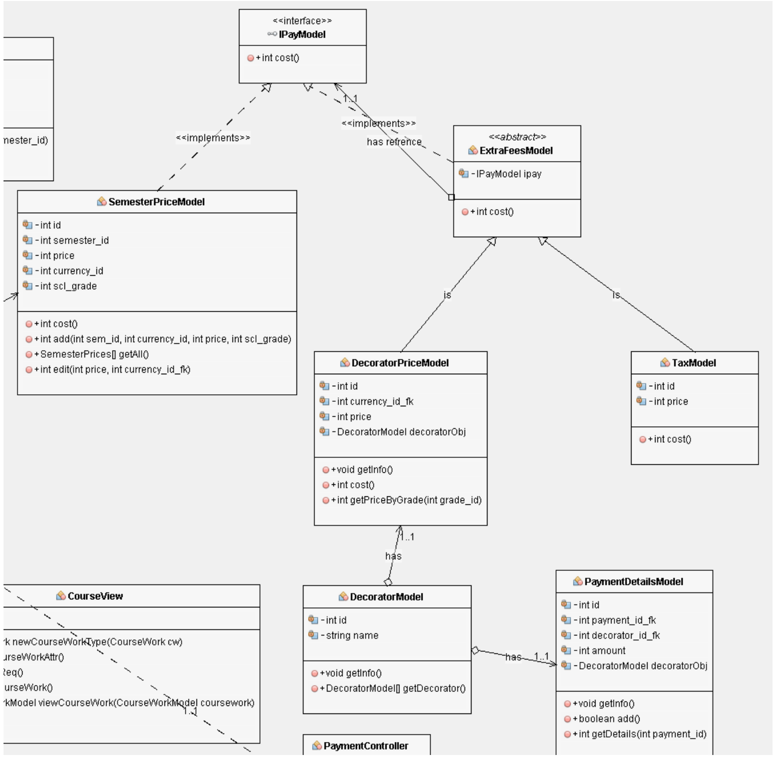
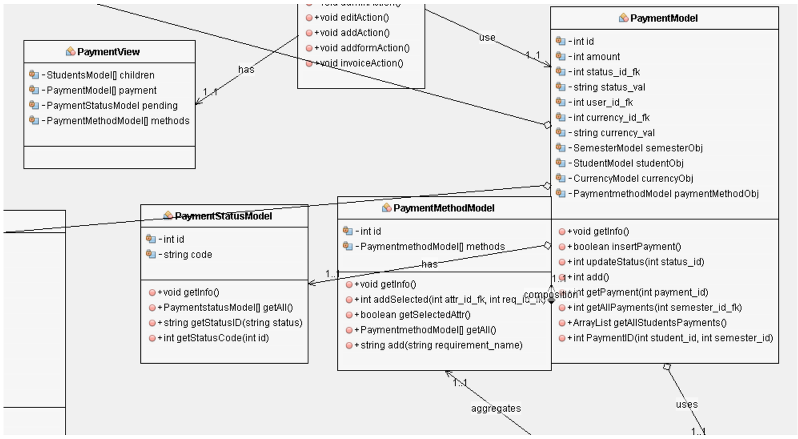


Figure 8: Decorator Pattern

Figure 11: The registration view shows what is viewed in the page, the con-



+void editAction()
+void addAction()
+void addformAction()
+void invoiceAction()
use
1..1
PaymentView
-StudentsModel[] children
-PaymentModel[] payment
-PaymentStatusModel pending
-PaymentMethodModel[] methods
has
1..1
PaymentModel
-int id
-int amount
-int status_id_fk
-string status_val
-int user_id_fk
-int currency_id_fk
-string currency_val
-SemesterModel semesterObj
-StudentModel studentObj
-CurrencyModel currencyObj
-PaymentmethodModel paymentMethodObj
+void getInfo()
+boolean insertPayment()
+int updateStatus(int status_id)
+int add()
+int getPayment(int payment_id)
+int getAllPayments(int semester_id_fk)
+ArrayList getAllStudentsPayments()
+int PaymentID(int student_id, int semester_id)
PaymentStatusModel
-int id
-string code
+void getInfo()
+PaymentstatusModel[] getAll()
+string getStatusID(string status)
+int getStatusCode(int id)
PaymentMethodModel
-int id
-PaymentmethodModel[] methods
+void getInfo()
+int addSelected(int attr_id_fk, int req_id_fk)
+boolean getSelectedAttr()
+PaymentmethodModel[] getAll()
+string add(string requirement_name)
has
1..1
1..1
composition
1..1
aggregates
uses
1..1

Figure 9: Payment

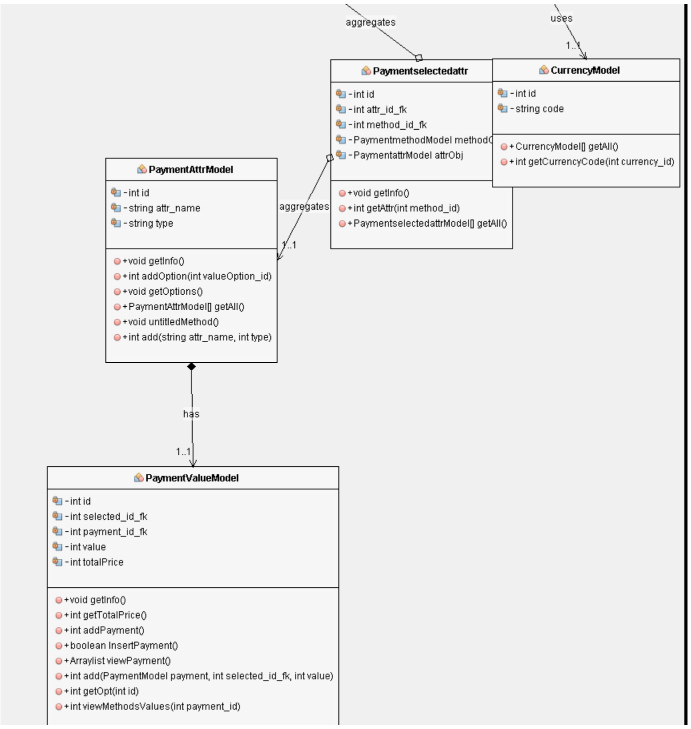


Figure 10: Payment

troller sends the info to the view taken from the model.

Figure 12: The student controller communicates with the transcript in order

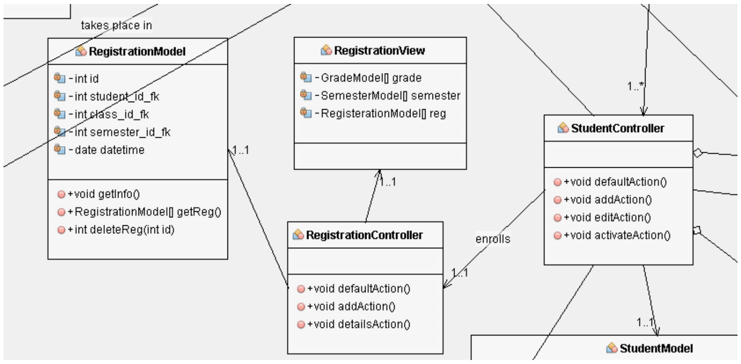


Figure 11: Registration

to view its details and access it.

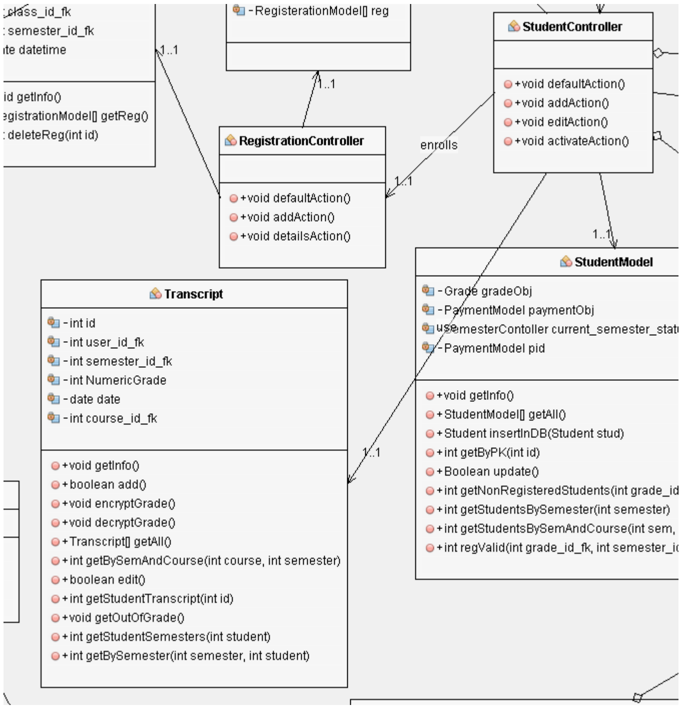


Figure 12: Transcript

Figure 13: We used the strategy design pattern for the forms. iElement is an interface, and several classes inherit from it, which are used when creating a new form.

Figure 14: The schedule controller has a function that takes the data from the model and passes it on to the view.

Figure 15 and 16: The course controller fetches data from the model and is viewed in the course view. The coursework diagram shows the different attributes it has.

3.5 Design Rationale

Since the system has many different screens, we used the MVC architecture in order to have several different views and interact with models through controllers. No other architecture was as suitable since we are developing a web application. For durability when it comes to change in requirements, the EAv architecture would give room for change without the need to rewrite code.

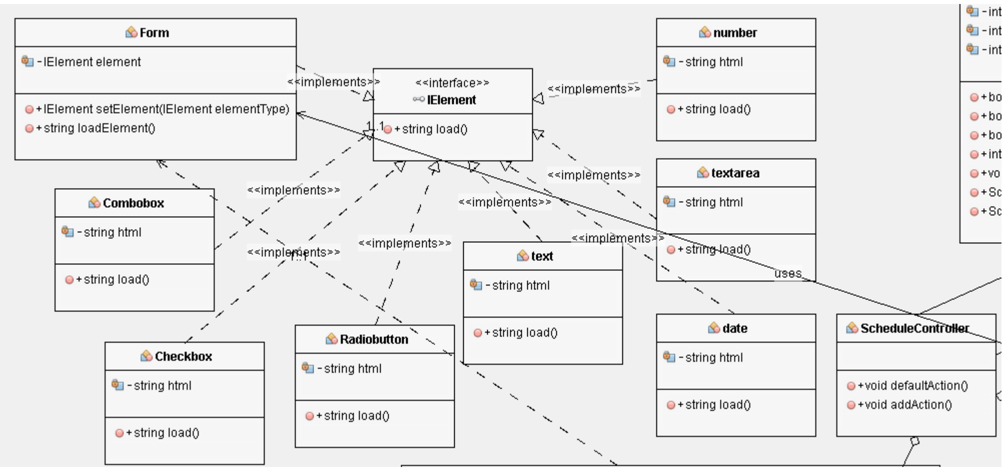


Figure 13: Strategy Pattern

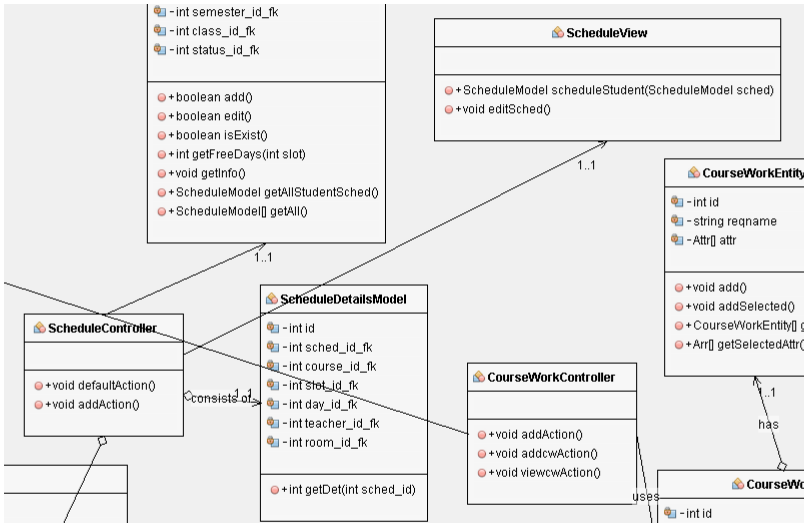


Figure 14: Schedule

4 Data Design

4.1 Data Description

A few non-functional requirements were met when designing the database. Maintainability was me by implementing the EAV. Security is met since we hashed the users' passwords, and encrypted the students' grades.

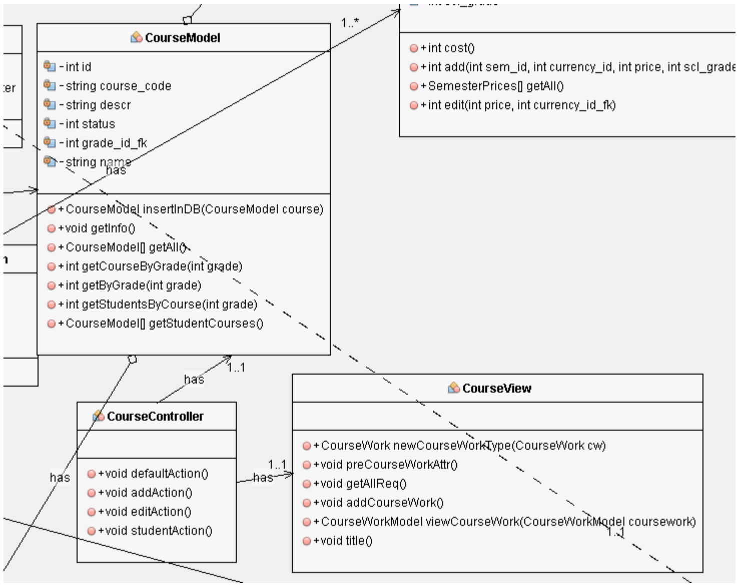

CourseModel
-int id
-string course_code
-string descr
-int status
-int grade_id_fk
-string name
+CourseModel insertInDB(CourseModel course)
+void getInfo()
+CourseModel[] getAll()
+int getCourseByGrade(int grade)
+int getByGrade(int grade)
+int getStudentsByCourse(int grade)
+CourseModel[] getStudentCourses()
+int cost()
+int add(int sem_id, int currency_id, int price, int scl_grade
+SemesterPrices[] getAll()
+int edit(int price, int currency_id_fk)
1..*
has
1..1
has
CourseController
+void defaultAction()
+void addAction()
+void editAction()
+void studentAction()
has
1..1
has
CourseView
+CourseWork newCourseWorkType(CourseWork cw)
+void preCourseWorkAttr()
+void getAllReq()
+void addCourseWork()
+CourseWorkModel viewCourseWork(CourseWorkModel coursework)
+void title()
1..1

Figure 15: Course

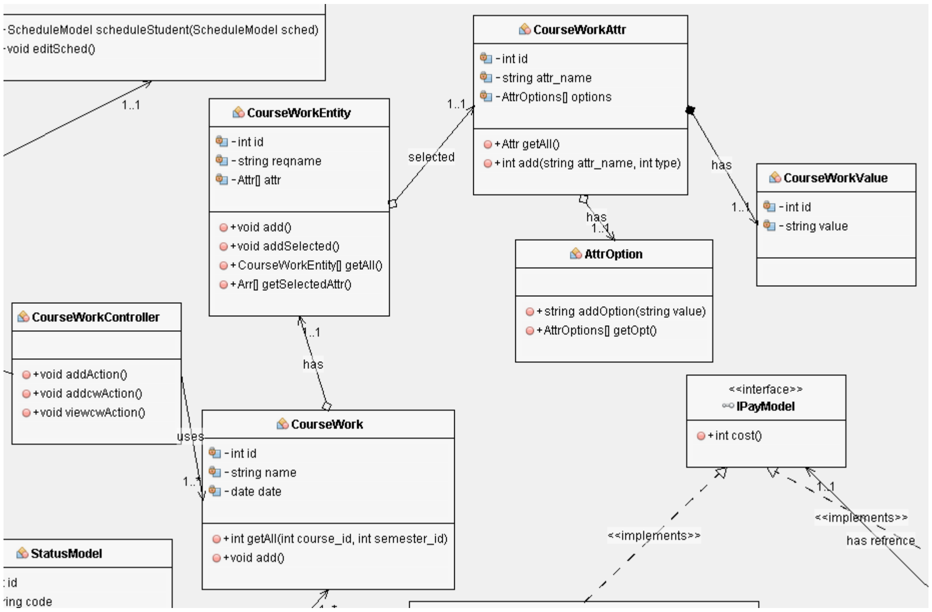
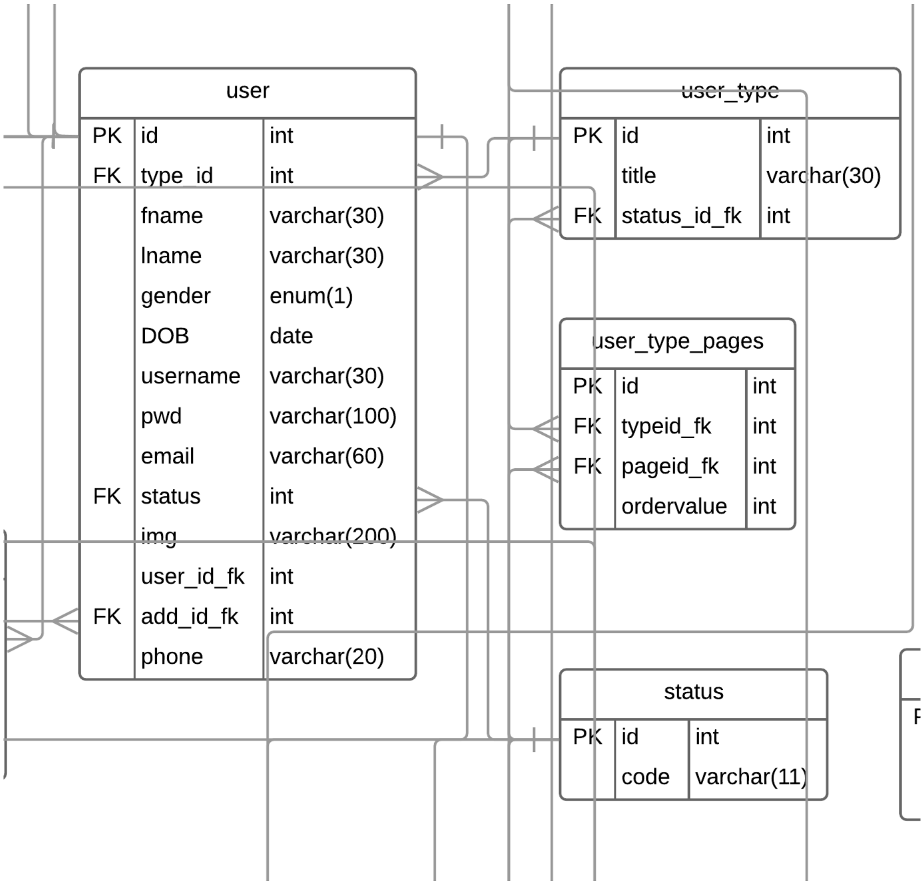


Figure 16: Coursework

4.1.1 User Table

The users are the people who will use the system or are registered into the system. Table ”user” stores all of the information of the people who are registered into the system. The users are assigned a type.



user

PK	id	int
FK	type_id	int
	fname	varchar(30)
	lname	varchar(30)
	gender	enum(1)
	DOB	date
	username	varchar(30)
	pwd	varchar(100)
	email	varchar(60)
FK	status	int
	img	varchar(200)
	user_id_fk	int
FK	add_id_fk	int
	phone	varchar(20)

user_type

PK	id	int
	title	varchar(30)
FK	status_id_fk	int

user_type_pages

PK	id	int
FK	typeid_fk	int
FK	pageid_fk	int
	ordervalue	int

status

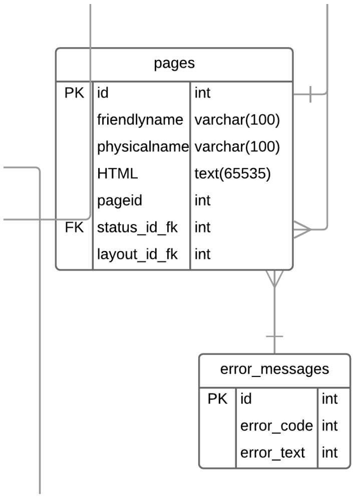
PK	id	int
	code	varchar(11)

Figure 17: User

- user: includes every person in the school
- user_type: every user has a type (e.g. student, teacher,..)
- user_type_pages: defines which usertype has access to which page
- status: every user is either active or inactive

4.1.2 Pages

- pages: includes all the names of every page in the system
- error_messages: pages have error messages



pages		
PK	id	int
	friendlyname	varchar(100)
	physicalname	varchar(100)
	HTML	text(65535)
	pageid	int
FK	status_id_fk	int
	layout_id_fk	int

error_messages		
PK	id	int
	error_code	int
	error_text	int

Figure 18: User

4.1.3 Coursework

One of the non-functional requirements is satisfied by using the EAV: maintainability. In the very likely case of having the desire to add a new coursework, a new row would be inserted in the database and the source code would remain untouched. This makes the change in requirements easy to handle.

- course: includes all the courses
- coursework; each course can have several types of coursework
- coursework_requir: entity, defines the type of coursework
- coursework_attr: coursework has attributes

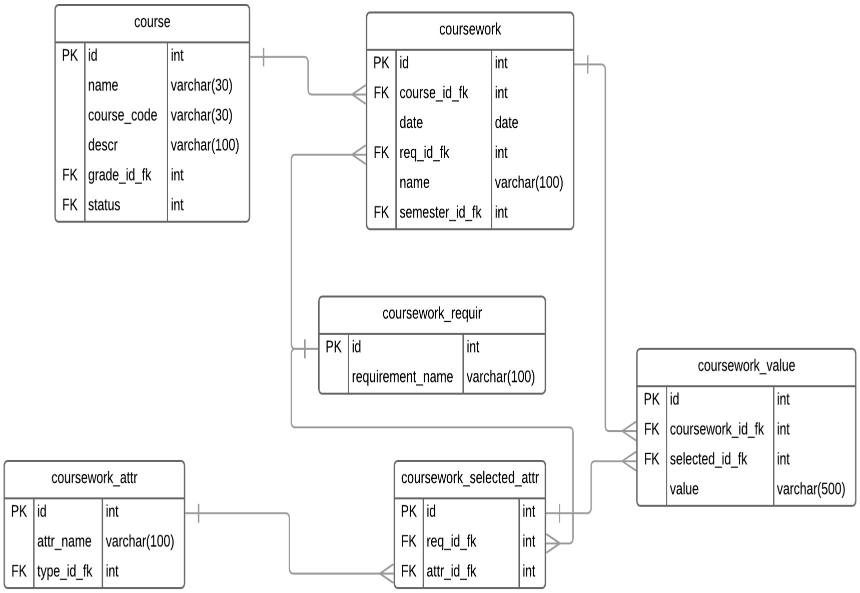


Figure 19: Coursework EAV

- coursework_selected_attr: each requirement and its corresponding one or more attribute
- coursework_value: each attribute has a value/name

4.1.4 Payment

Again, we used the EAV for the payment. The payment concerns everything the student pays for in the school. A design pattern was also used to make this possible; the decorator pattern. The students' fees are distributed upon several attributes, from bus fees to the semester fees. The decorator includes what services can be paid for.

- payment_method: entity, defines choice of payment
- payment_attr: defines attributes of a payment method

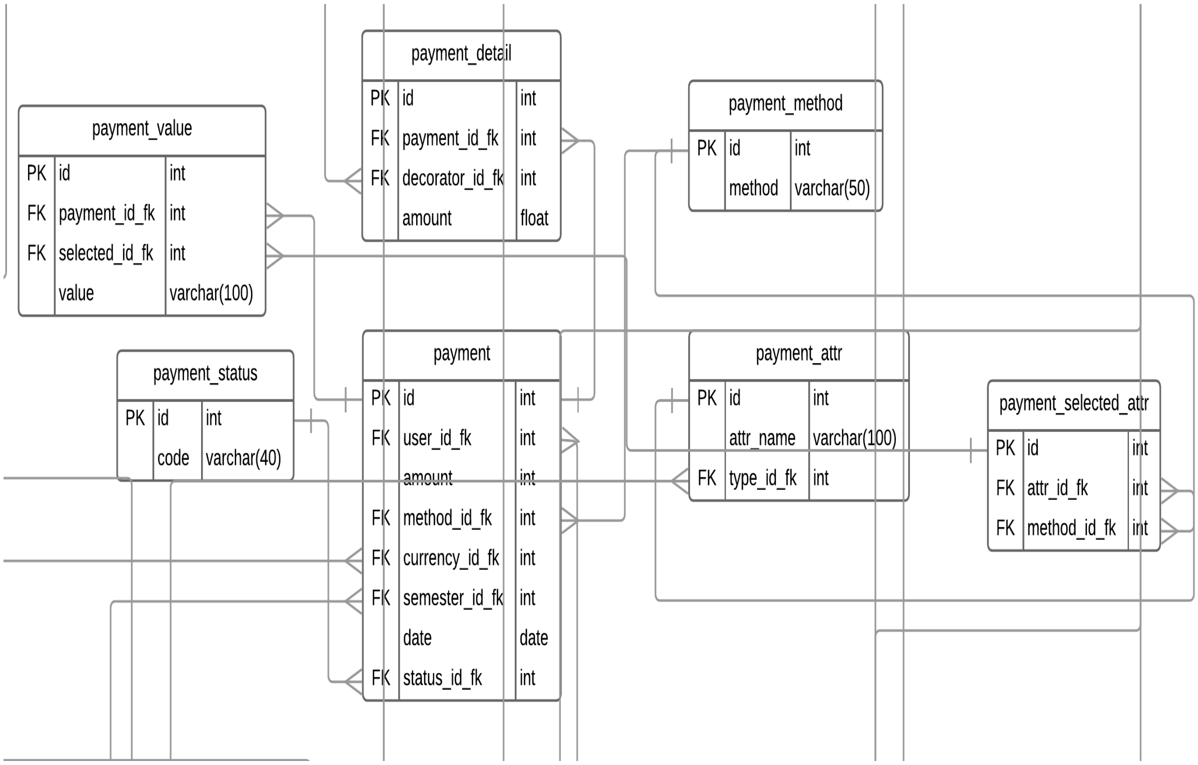
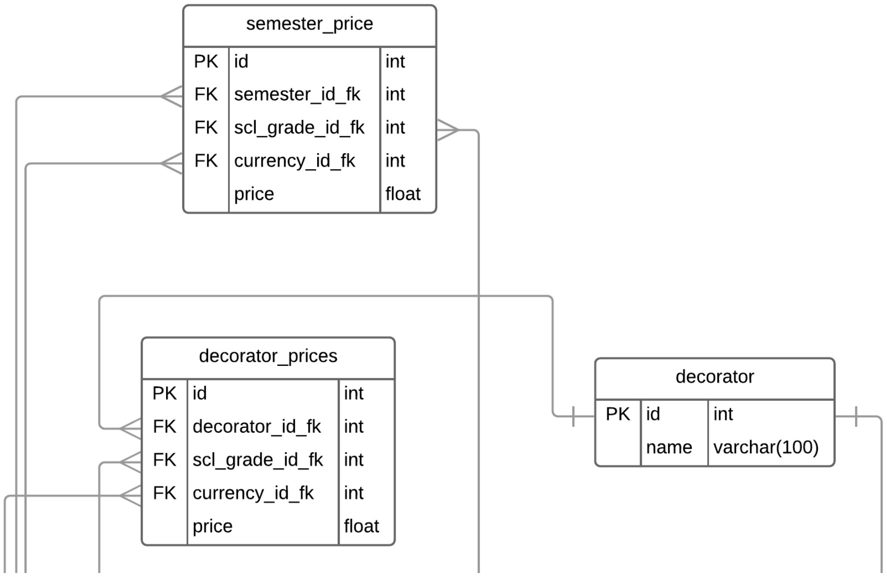


Figure 20: Payment EAV

- payment_detail: defines which user is paying for what from the decorator
- payment_selected_attr: every method and its attribute
- payment_status: defines whether the payment is done or pending
- payment_value: defines which user chose which method and its attributes and defines its value

- semester_price: base, defines the price of each semester according to its grade
- decorator: defines type (e.g. semester, bus), or what's the user paying for
- decorator_price: defines price according to decorator type



semester_price

PK	id	int
FK	semester_id_fk	int
FK	scl_grade_id_fk	int
FK	currency_id_fk	int
	price	float

decorator_prices

PK	id	int
FK	decorator_id_fk	int
FK	scl_grade_id_fk	int
FK	currency_id_fk	int
	price	float

decorator

PK	id	int
	name	varchar(100)

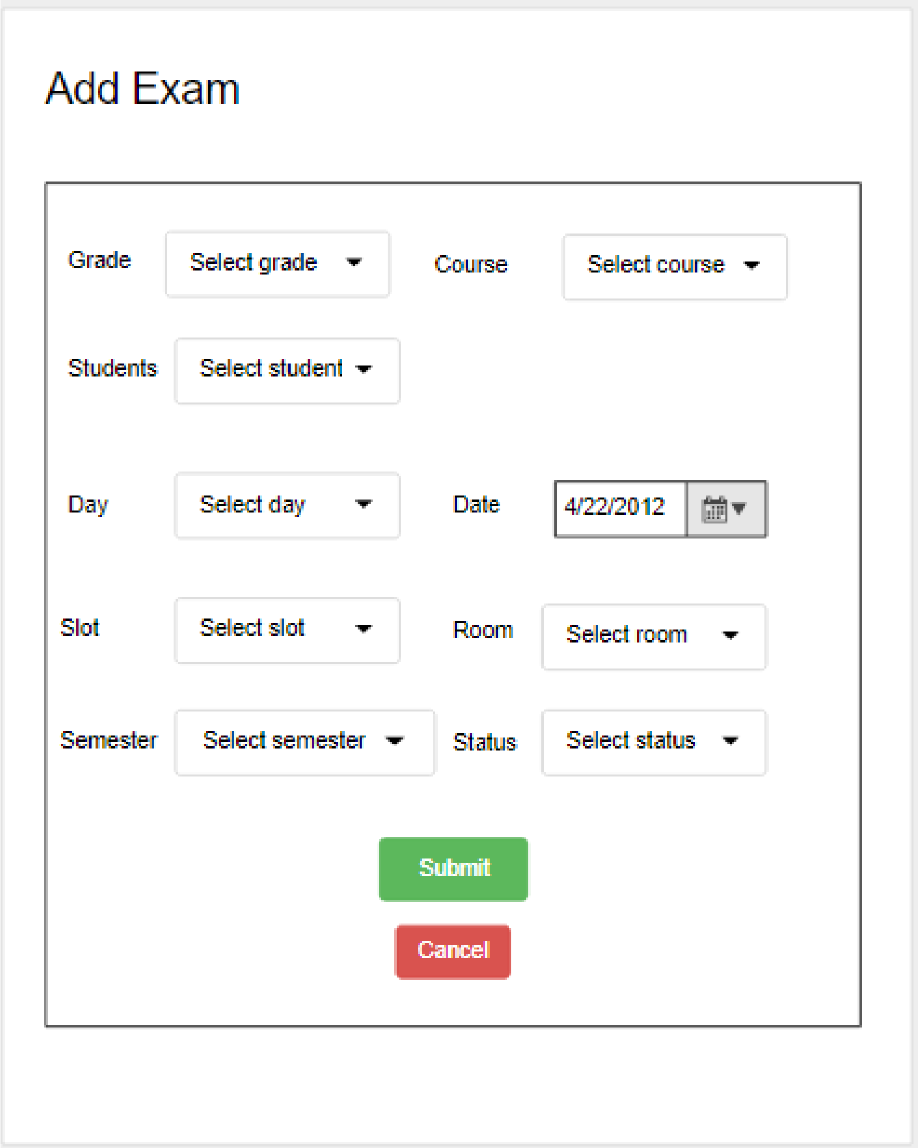
Figure 21: Decorator

5 Human Interface Design

5.1 Overview of User Interface

We have two main users who will be using the system: the administrators and students/parents. From the admin side, the administrator will have access to everything in the system.

5.2 Screen Images



Add Exam

Grade: Select grade

Course: Select course

Students: Select student

Day: Select day

Date: 4/22/2012

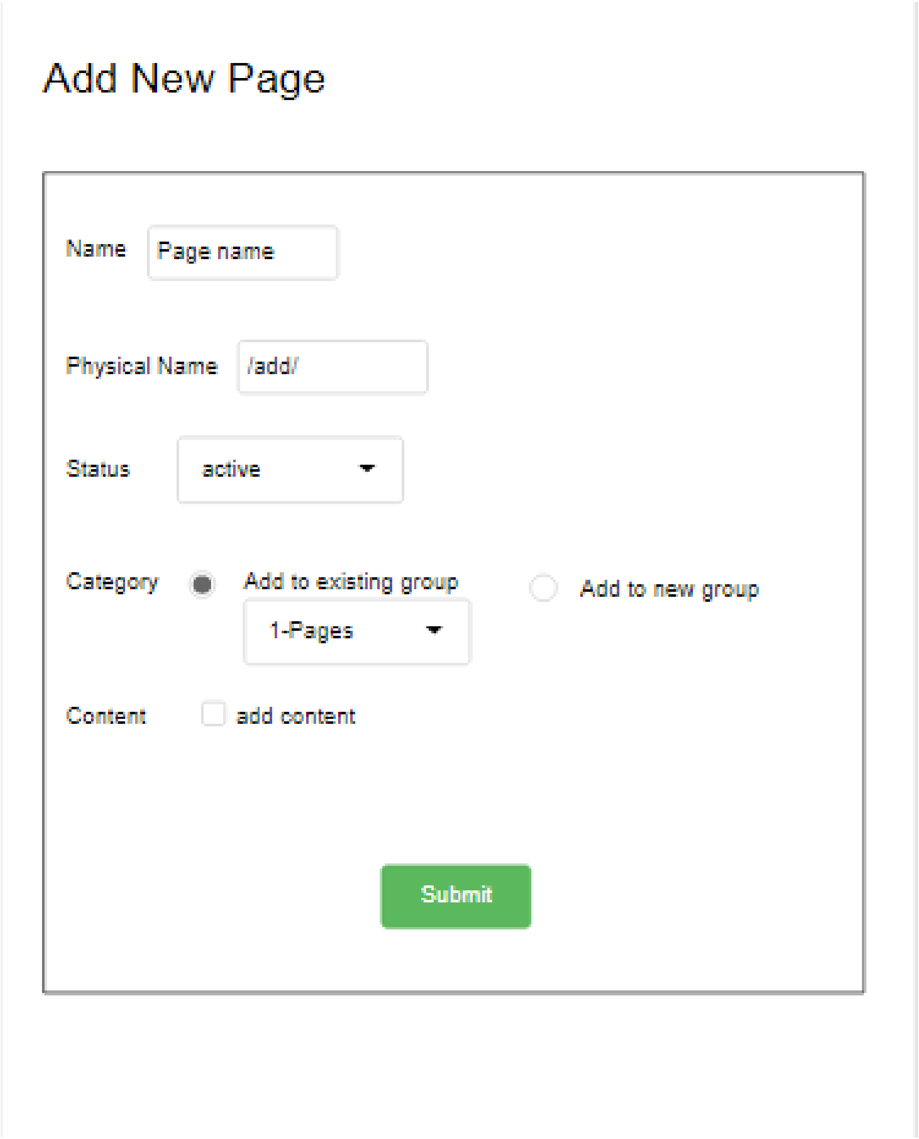
Slot: Select slot

Room: Select room

Semester: Select semester

Status: Select status

Figure 22: Adding a new exam



Add New Page

Name: Page name

Physical Name: /add/

Status: active

Category: ● Add to existing group: 1-Pages ○ Add to new group

Content: □ add content

Figure 23: Adding a new page

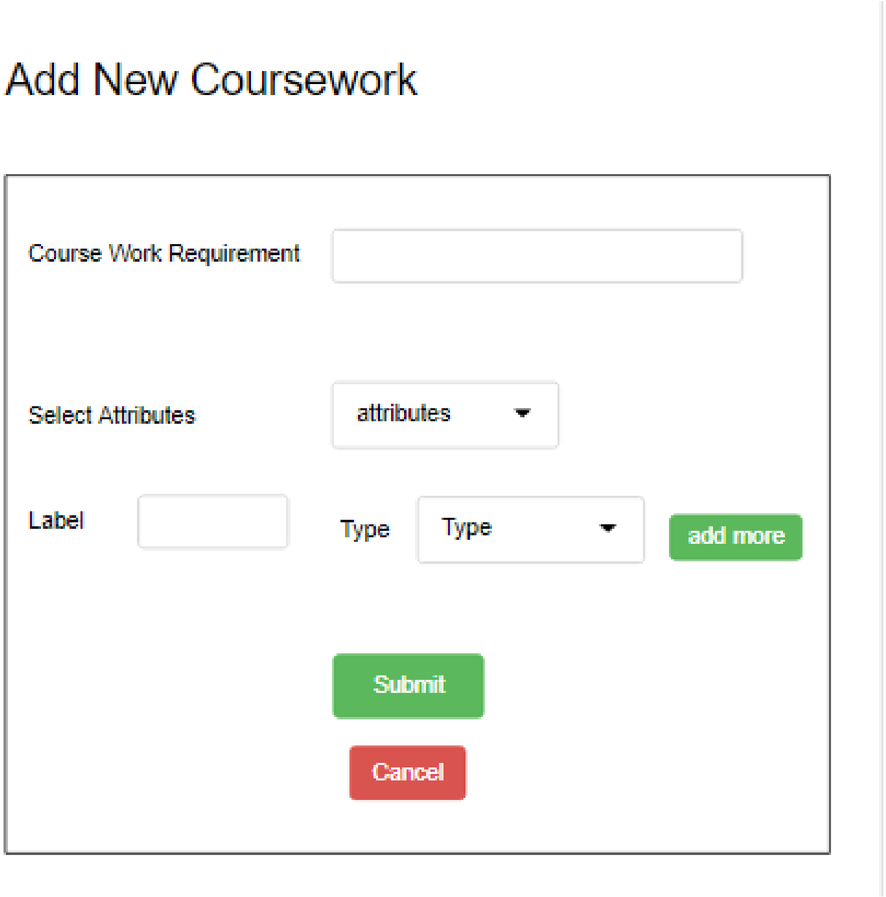
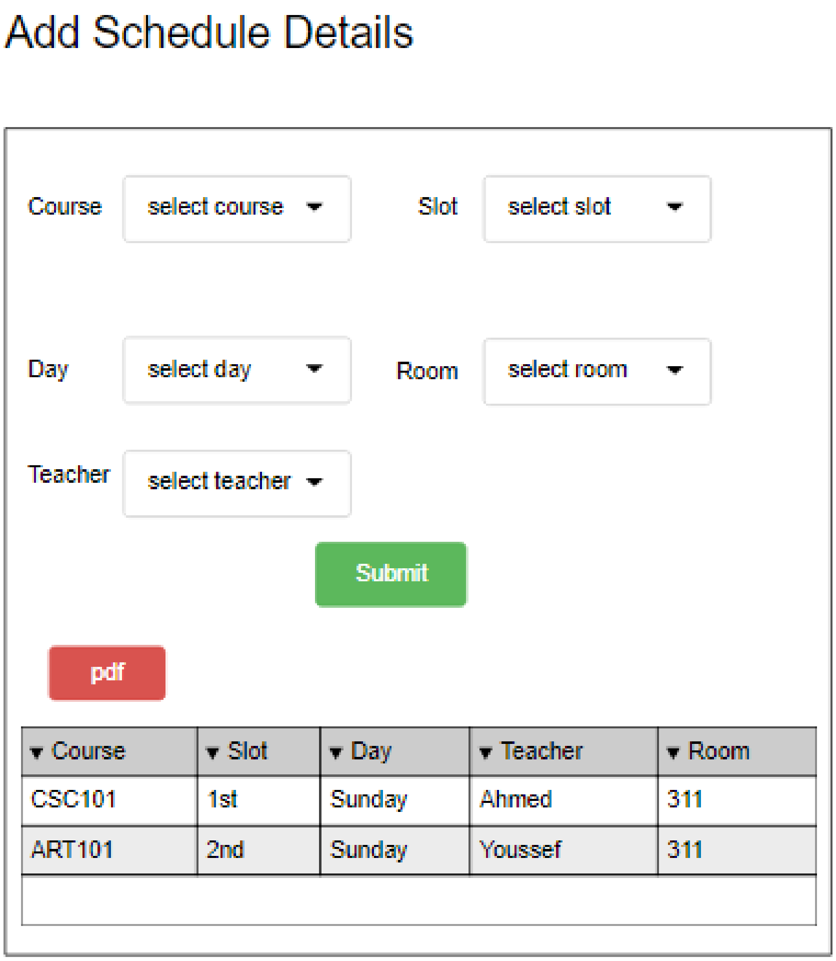


Figure 24: Adding coursework

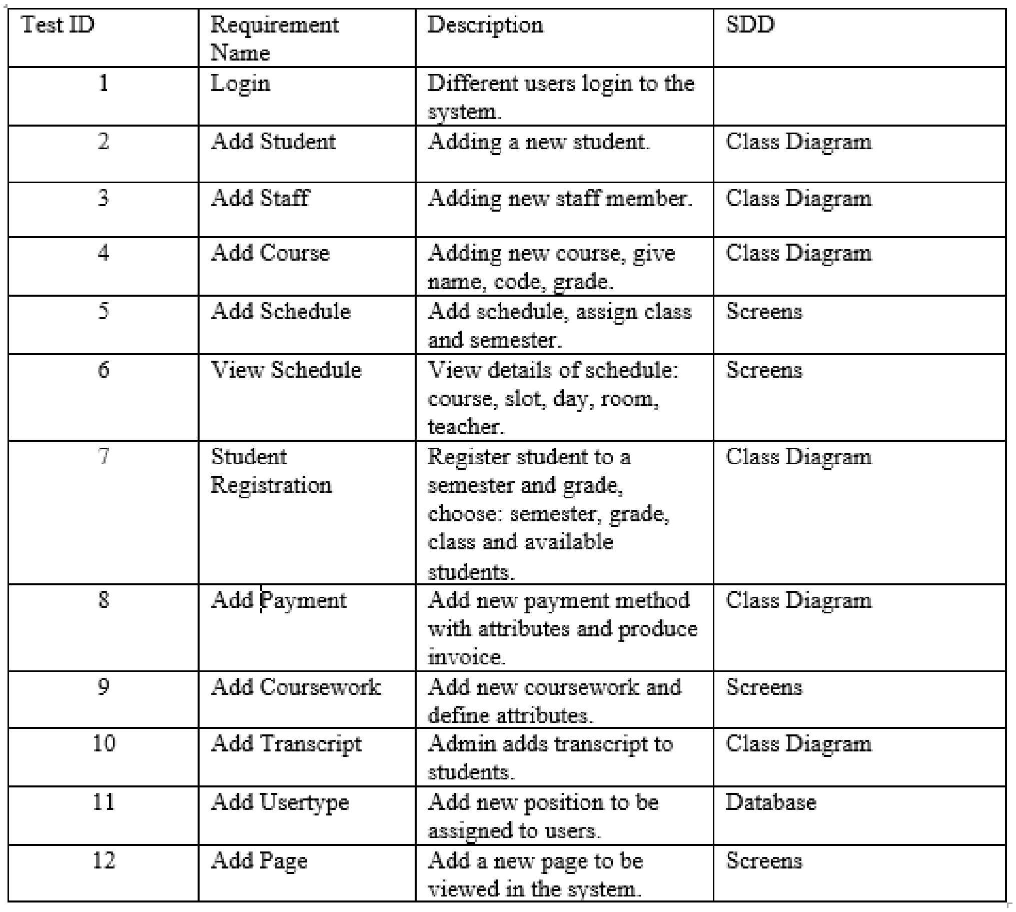


Add Schedule Details

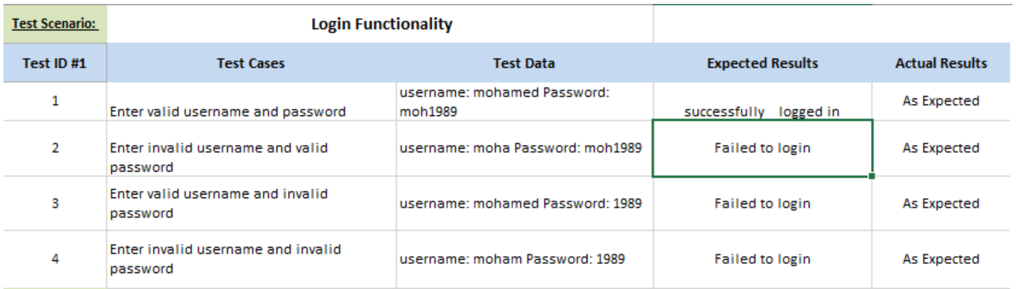
▾ Course	▾ Slot	▾ Day	▾ Teacher	▾ Room
CSC101	1st	Sunday	Ahmed	311
ART101	2nd	Sunday	Youssef	311

Figure 25: Adding coursework

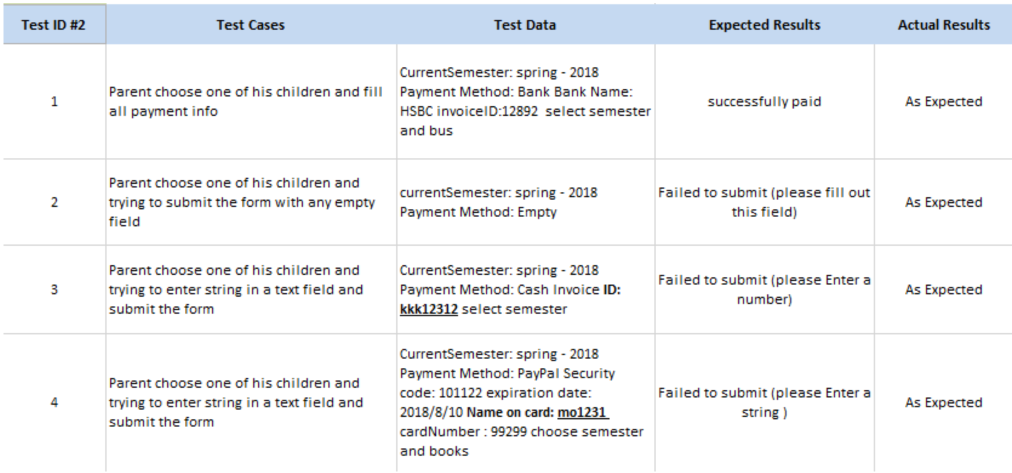
6 Requirements Matrix



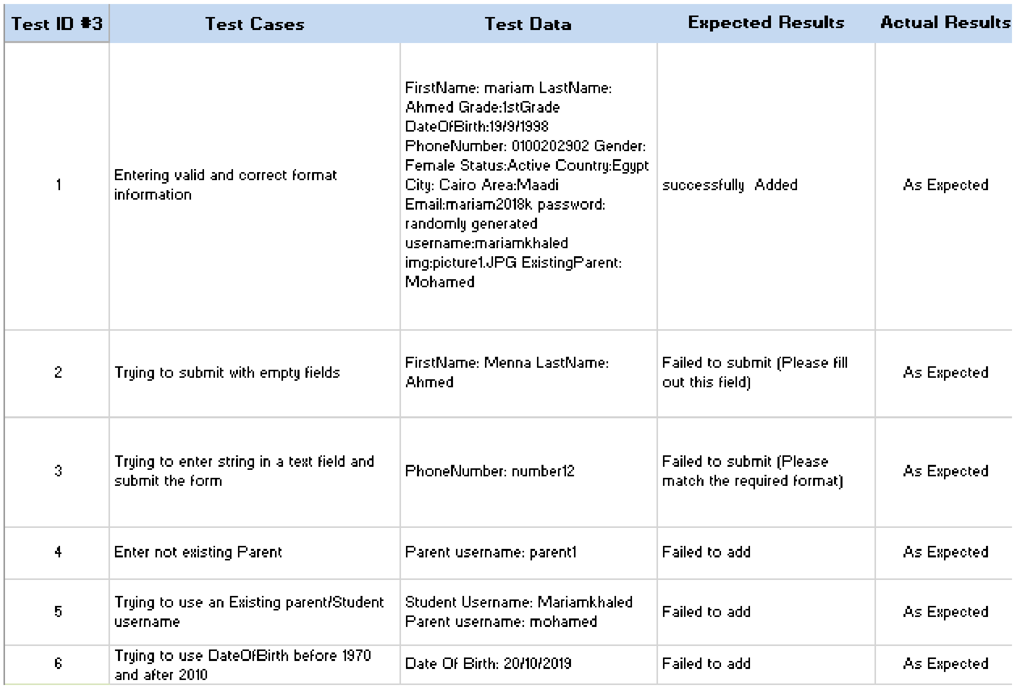
Test ID	Requirement Name	Description	SDD
1	Login	Different users login to the system.	
2	Add Student	Adding a new student.	Class Diagram
3	Add Staff	Adding new staff member.	Class Diagram
4	Add Course	Adding new course, give name, code, grade.	Class Diagram
5	Add Schedule	Add schedule, assign class and semester.	Screens
6	View Schedule	View details of schedule: course, slot, day, room, teacher.	Screens
7	Student Registration	Register student to a semester and grade, choose: semester, grade, class and available students.	Class Diagram
8	Add Payment	Add new payment method with attributes and produce invoice.	Class Diagram
9	Add Coursework	Add new coursework and define attributes.	Screens
10	Add Transcript	Admin adds transcript to students.	Class Diagram
11	Add Usertype	Add new position to be assigned to users.	Database
12	Add Page	Add a new page to be viewed in the system.	Screens



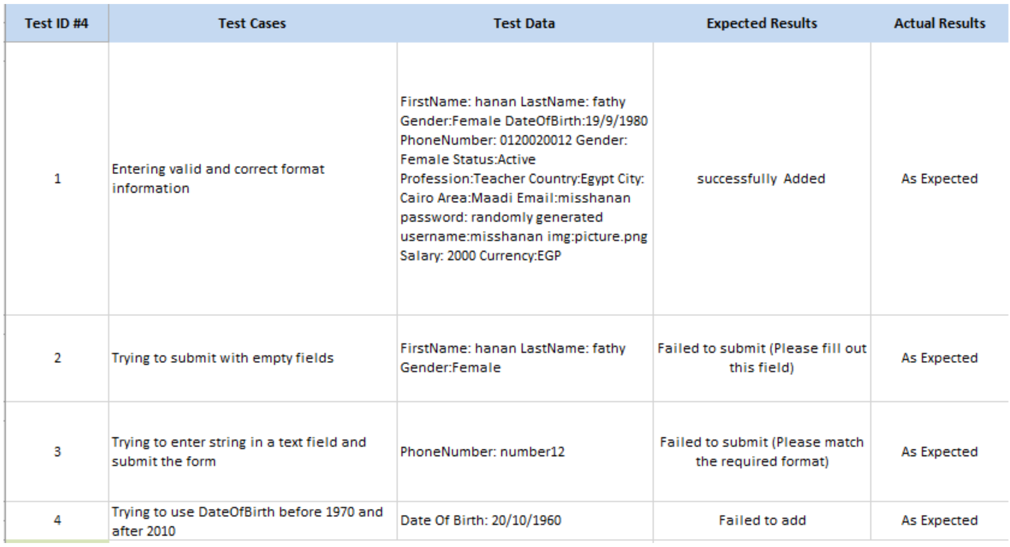
Test Scenario:	Login Functionality			
Test ID #1	**Test Cases**	**Test Data**	**Expected Results**	**Actual Results**
1	Enter valid username and password	username: mohamed Password: moh1989	successfully logged in	As Expected
2	Enter invalid username and valid password	username: moha Password: moh1989	Failed to login	As Expected
3	Enter valid username and invalid password	username: mohamed Password: 1989	Failed to login	As Expected
4	Enter invalid username and invalid password	username: moham Password: 1989	Failed to login	As Expected



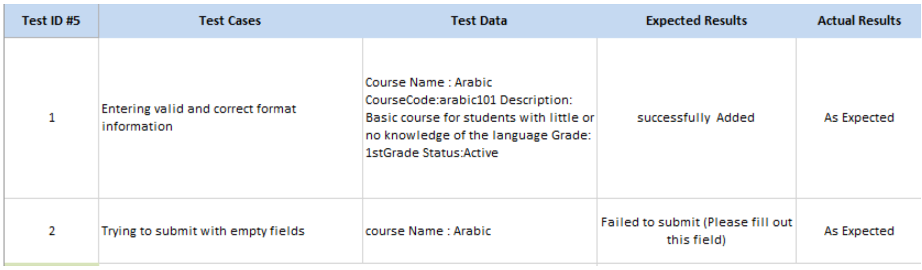
Test ID #2	Test Cases	Test Data	Expected Results	Actual Results
1	Parent choose one of his children and fill all payment info	CurrentSemester: spring - 2018 Payment Method: Bank Bank Name: HSBC invoiceID:12892 select semester and bus	successfully paid	As Expected
2	Parent choose one of his children and trying to submit the form with any empty field	currentSemester: spring - 2018 Payment Method: Empty	Failed to submit (please fill out this field)	As Expected
3	Parent choose one of his children and trying to enter string in a text field and submit the form	CurrentSemester: spring - 2018 Payment Method: Cash Invoice **ID:** **kkk12312** select semester	Failed to submit (please Enter a number)	As Expected
4	Parent choose one of his children and trying to enter string in a text field and submit the form	CurrentSemester: spring - 2018 Payment Method: PayPal Security code: 101122 expiration date: 2018/8/10 **Name on card:** **mo1231** cardNumber : 99299 choose semester and books	Failed to submit (please Enter a string)	As Expected



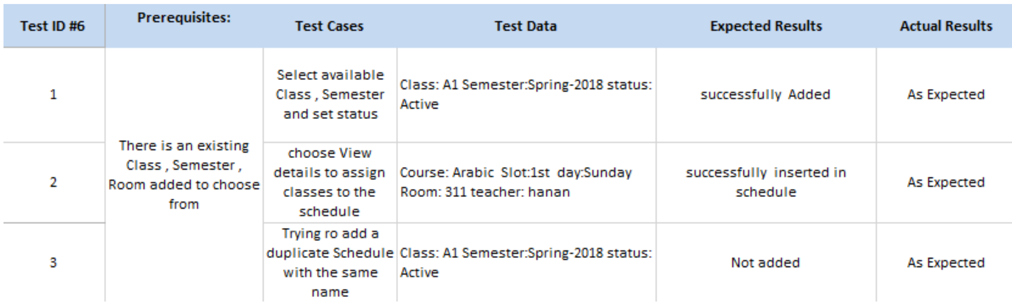
Test ID #3	Test Cases	Test Data	Expected Results	Actual Results
1	Entering valid and correct format information	FirstName: mariam LastName: Ahmed Grade:1stGrade DateOfBirth:19/9/1998 PhoneNumber: 0100202902 Gender: Female Status:Active Country:Egypt City: Cairo Area:Maadi Email:mariam2018k password: randomly generated username:mariamkhaled img:picture1.JPG ExistingParent: Mohamed	successfully Added	As Expected
2	Trying to submit with empty fields	FirstName: Menna LastName: Ahmed	Failed to submit (Please fill out this field)	As Expected
3	Trying to enter string in a text field and submit the form	PhoneNumber: number12	Failed to submit (Please match the required format)	As Expected
4	Enter not existing Parent	Parent username: parent1	Failed to add	As Expected
5	Trying to use an Existing parent/Student username	Student Username: Mariamkhaled Parent username: mohamed	Failed to add	As Expected
6	Trying to use DateOfBirth before 1970 and after 2010	Date Of Birth: 20/10/2019	Failed to add	As Expected



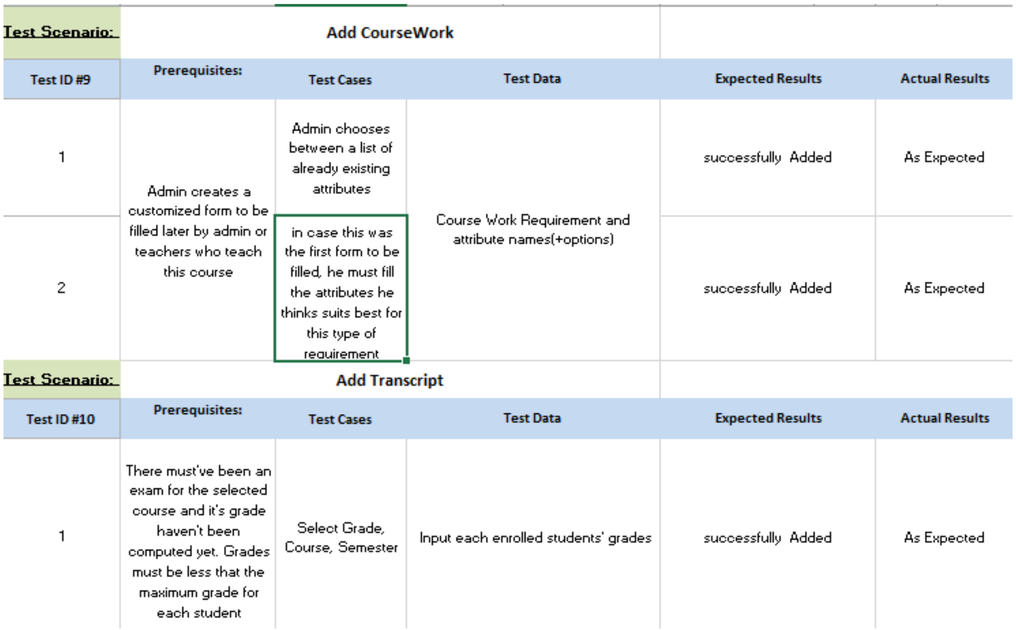
Test ID #4	Test Cases	Test Data	Expected Results	Actual Results
1	Entering valid and correct format information	FirstName: hanan LastName: fathy Gender:Female DateOfBirth:19/9/1980 PhoneNumber: 0120020012 Gender: Female Status:Active Profession:Teacher Country:Egypt City: Cairo Area:Maadi Email:misshanan password: randomly generated username:misshanan img:picture.png Salary: 2000 Currency:EGP	successfully Added	As Expected
2	Trying to submit with empty fields	FirstName: hanan LastName: fathy Gender:Female	Failed to submit (Please fill out this field)	As Expected
3	Trying to enter string in a text field and submit the form	PhoneNumber: number12	Failed to submit (Please match the required format)	As Expected
4	Trying to use DateOfBirth before 1970 and after 2010	Date Of Birth: 20/10/1960	Failed to add	As Expected



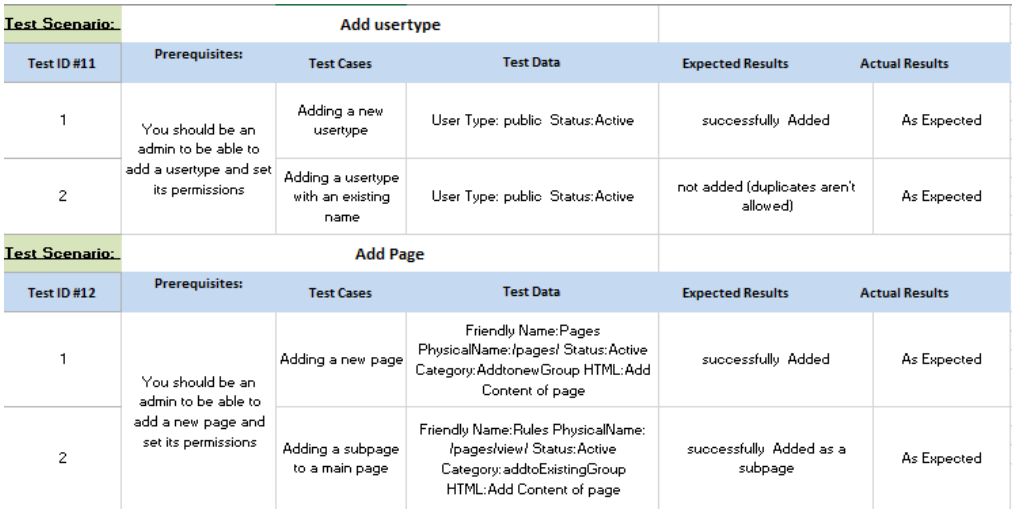
Test ID #5	Test Cases	Test Data	Expected Results	Actual Results
1	Entering valid and correct format information	Course Name : Arabic CourseCode:arabic101 Description: Basic course for students with little or no knowledge of the language Grade: 1stGrade Status:Active	successfully Added	As Expected
2	Trying to submit with empty fields	course Name : Arabic	Failed to submit (Please fill out this field)	As Expected



Test ID #6	Prerequisites:	Test Cases	Test Data	Expected Results	Actual Results
1	There is an existing Class , Semester , Room added to choose from	Select available Class , Semester and set status	Class: A1 Semester:Spring-2018 status: Active	successfully Added	As Expected
2		choose View details to assign classes to the schedule	Course: Arabic Slot:1st day:Sunday Room: 311 teacher: hanan	successfully inserted in schedule	As Expected
3		Trying ro add a duplicate Schedule with the same name	Class: A1 Semester:Spring-2018 status: Active	Not added	As Expected



Test Scenario:	Add CourseWork				
Test ID #9	Prerequisites:	Test Cases	Test Data	Expected Results	Actual Results
1	Admin creates a customized form to be filled later by admin or teachers who teach this course	Admin chooses between a list of already existing attributes	Course Work Requirement and attribute names(+options)	successfully Added	As Expected
2		in case this was the first form to be filled, he must fill the attributes he thinks suits best for this type of requirement		successfully Added	As Expected
Test Scenario:	Add Transcript				
Test ID #10	Prerequisites:	Test Cases	Test Data	Expected Results	Actual Results
1	There must've been an exam for the selected course and it's grade haven't been computed yet. Grades must be less that the maximum grade for each student	Select Grade, Course, Semester	Input each enrolled students' grades	successfully Added	As Expected



Test Scenario:		**Add usertype**			
Test ID #11	**Prerequisites:**	**Test Cases**	**Test Data**	**Expected Results**	**Actual Results**
1	You should be an admin to be able to add a usertype and set its permissions	Adding a new usertype	User Type: public Status:Active	successfully Added	As Expected
2		Adding a usertype with an existing name	User Type: public Status:Active	not added (duplicates aren't allowed)	As Expected
Test Scenario:		**Add Page**			
Test ID #12	**Prerequisites:**	**Test Cases**	**Test Data**	**Expected Results**	**Actual Results**
1	You should be an admin to be able to add a new page and set its permissions	Adding a new page	Friendly Name:Pages PhysicalName:/pages/ Status:Active Category:AddtonewGroup HTML:Add Content of page	successfully Added	As Expected
2		Adding a subpage to a main page	Friendly Name:Rules PhysicalName: /pages/view/ Status:Active Category:addtoExistingGroup HTML:Add Content of page	successfully Added as a subpage	As Expected