**Department of Computer Science**

**School of Science**

**National Textile University**



**University Management System**

Submitted by:

Muhammad Kamran(18-NTU-CS-1130)

Abdul Qados(18-NTU-CS-1112)

Mian Mujtbah Asif (18-NTU-1122)

Muhammad Ahmed(18-NTU-CS-1133)

**[Date of submission]**

Submitted to:

**Full Name [Designation]**

***Course Title [Course code]- Fall-2001***

Table of Contents

[Abstract: 3](#_Toc78619354)

[Project Report: 3](#_Toc78619355)

[Chapter 01: Application of Principle of Abstraction 3](#_Toc78619356)

[Functional Requirements: 5](#_Toc78619357)

[Non-Functional Requirement: 6](#_Toc78619358)

[Chapter 02: Application of Principle of Refinement: 6](#_Toc78619359)

[Class Diagram: 8](#_Toc78619360)

[Use Case Diagram: 9](#_Toc78619361)

[Chapter 03: Recommendation and conclusion 9](#_Toc78619362)

[Reference and resources: 10](#_Toc78619363)

# Abstract:

The general function that the system can do is based on the transaction done by the university admin in managing the student, teacher, and user information. The admin can add, update, and delete the student, teacher, and user record. We have enhanced the security of the system. There are many bad smells in the code. Like code duplication. Bad variables name is used. Bad file structure is used. No template engine format is used for avoiding duplication code on front end. And the folder structure is also bad. We have made the UI more responsive. We converted the system into pure object-oriented. Created the template engine and required it where needed to avoid duplication. We enhanced the security of system and removed the bad smells. We have recoded the front end for the student and teacher module. Used the OPP concepts like inheritance, polymorphism, encapsulation, and abstraction.

# Project Report:

## Chapter 01: Application of Principle of Abstraction

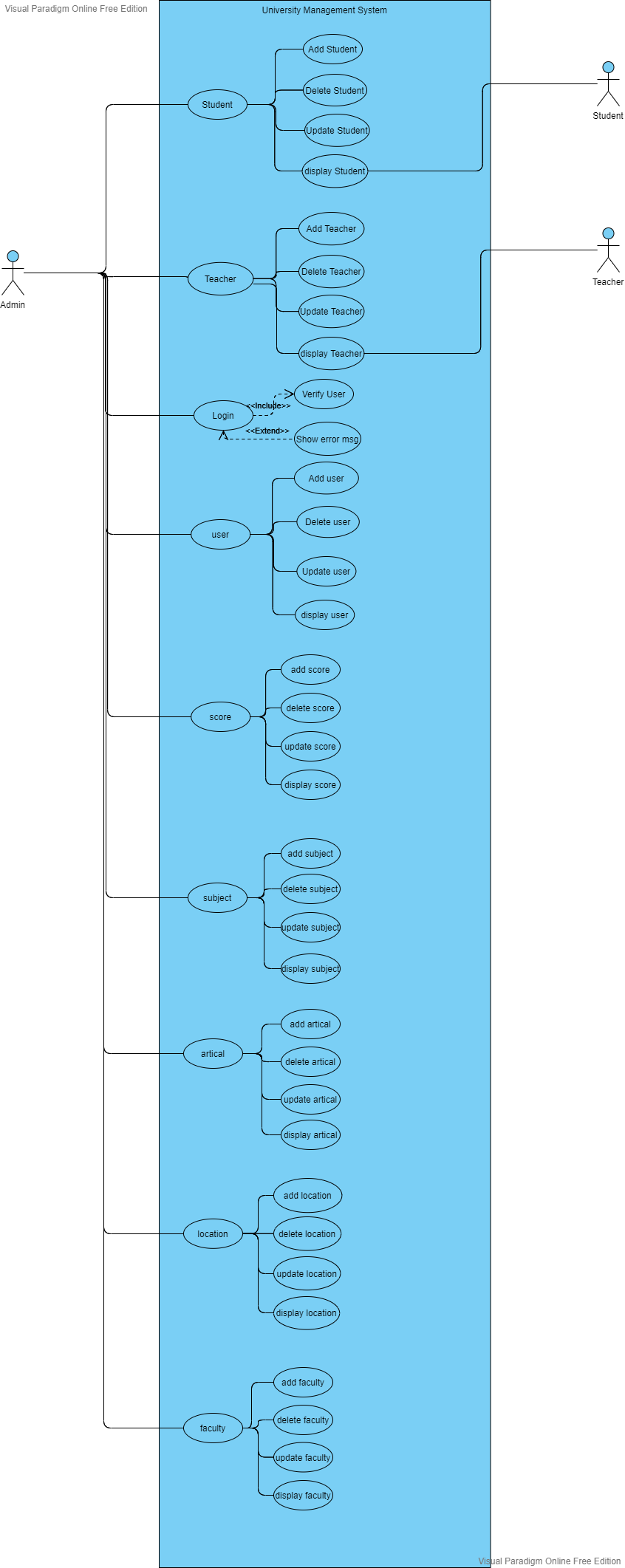
1. Code Comprehension:

The university management system is developed in php, and all the work is done with the help of function which is repeated in every section. The UI is not much user friendly. There is no template engine is used and the code is duplicated in every section. Static files, duplication file like JS, CSS, image folder was added. The folder structure is bad. The crud operations for student and teacher details adding are the same but used to save through different parameters. At every page the system check that the user is logged in or not. Long parameters are used. Long condition statements are used.

1. Design and architecture recovery:

The software provides the facility to admin of adding, deleting, updating, viewing individual details, viewing records, setting, and updating them as well. The overall work in the system is done without object-oriented and with much complication.

Use Case Diagram:



1. Development of specification of document:

## Functional Requirements:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Requirement #** | **Title** | **Requirement** | **Rationale** | **Dependency** |
| 1F | Login | Login for the security of website | For the security of the information the login system is important so the everyone should login to so that all the information remain secure | The user should be authenticated. |
| 2F | Adding data | The admin and authentic users should be able to add all the data. | admin or user must be logged in. | The admin or user must be logged in with valid id and password. |
| 3F | Deleting data | The admin and authentic users should be able to delete all the data. | admin or user must be logged in. | The admin or user must be logged in with valid id and password |
| 4F | Updating data | The admin and authentic users should be able to update all the data. | admin or user must be logged in. | The admin or user must be logged in with valid id and password |
| 5F | View all record | The admin and authentic user should be able to view all record | admin or user must be logged in. | The admin or user must be logged in with valid id and password |
| 6F | View record for student | The students should only be able to view their details. They should not be able to modify it | student must be logged in. | The student must be logged in with valid id and password. |
| 7F | Viewing record for teacher | The teacher should be able to view there record only. | teacher must be logged in using valid login details. | The teacher must be logged in with valid id and password. |

## Non-Functional Requirement:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Requirement #** | **Title** | **Requirement** | **Rationale** | **Dependency** |
| 1NF | Security | The system should be secured | For the security of information | None |
| 2NF | Reliability | The system must have least downtime | To ensure the proper working of system any time | System which is being used. |
| 3NF | Portable | The system should be able to run on any platform. | The system should be able to run on another platform | Running platform |
| 4NF | Speed | The system should perform actions rapidly | Operator can do his tasks quickly | The system is being used |
| 5NF | Responsiveness | The system should be easily access on every device | Operator can access the system at any place | None |

## Chapter 02: Application of Principle of Refinement:

1. Generation of test case, modified specification document:

The admin should be able to add, delete, update and view record of the student and teacher and the teacher and student should only be able to view their details.

1. Modification of design and architecture:

The previous design and architecture of the system is in procedural form, function based and consist of bad smalls there are also code duplication. We removed the bad smells. There is no template engine to avoid the duplication of the code. We created a template engine and used them where we needed. There is static file in the system and the file structure in bad. We put the static files in a include folder and put them in one file and included in the section to avoid duplication of static file. We have used the OOP paradigm We used the OOP concepts like inheritance, polymorphism, abstraction, and encapsulation. The long parameters have been changed to arrays

. We modified the long condition statements into small statements. We modified the login session so that the user must be verified one to use every page instead of verifying that the user is logged in at every page.

In **Databases refactoring** phase we just used three tables teacher, student, and user for our project.

We upgraded the MySQL server from 5.3.9 to latest version of 8. Php version was 5..3.9 which is also have security leaks we upgraded it to php 8.

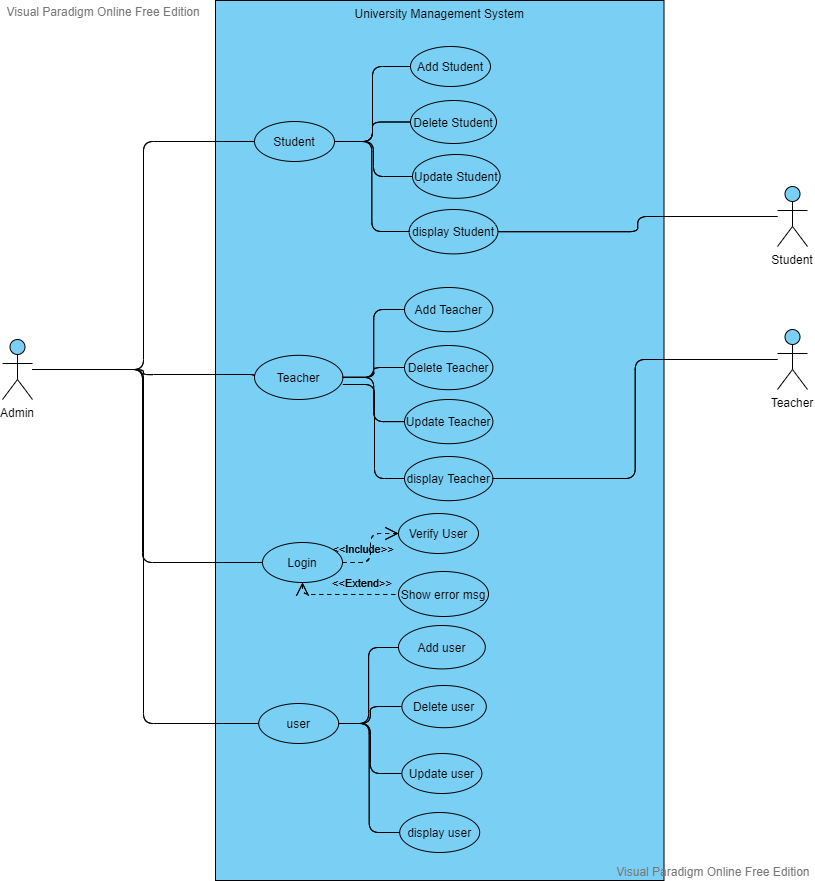
*Also MySQL was previously deprecated In the PHP5 and in PHP 7 MSQL support was removed because of lack of security of the MySQL now converted it to MSQLI in PHP 8 and higher version of php 5*

There were bad database names. Bad tables name. we have refactored them to good naming conventions so that they are understood easily.

The UI of the website is old and included some errors in it we have recoded the front end of teacher and student and make it more responsive and attractive.

## Class Diagram:

## Use Case Diagram:



1. Refactoring of code:

We enhanced the performance of the system and increases the readability and understandability of the system. So that the system is easy to understand. The system was in procedural way before we changed it to object-oriented and class base. Which help making the code more efficient and maintainable.

Here are some code snippets for the demo. that we have refactored.

# Chapter 03: Recommendation and conclusion

We have used the three modules of the university management system. There are also other modules available we can also inherit them using the object-oriented programing concepts which is inheritance, polymorphism, and encapsulation. To make the system more understandable the one can use comments for increasing the understandability of the code in the future. The other modules like faculties, subjects, location, and article sections have also bad smells in them which can be removed. The Whole system can also be converted into complete OOP based.

The **session login module** updated for these three modules can also be used in another module. If other module we attach to these three modules we do not need to go and add session login in every module we just need to add the login route in the Navbar session is will automatically will be applied to new modules as well. We have project them through routes.

The general function that the system can do is based on the transaction done by the university admin in managing the student, teacher, and user information. The admin and the user can add, update, and delete the student, teacher, and user record. The security of the system is enhanced so that only authentic user can access the information stored and can modify them.

There are many bad smells in the code. Like the attribute names are not understandable and there is also code duplication. Bad variables names are used. Bad file structure is used like all the files are directly added into the file, but we have changed the file structure and included it into one file and then added that into one file and required it at places where we needed.

No template engine format is used for avoiding duplication code. We make the UI more responsive. Reduced the complexity of the front end make it easier to understand. the system is converted from the procedural to pure object-oriented. Created the template engine and required it where needed to avoid duplication.

Data base connection was deprecated and prone to security risks PHP 5 was used which have performance issues and security as well. MySQL queries were used which are deprecated in the php 5 and removed in the php 7 we refactored code in the newer version of the php that is 8 stable and secure and used PDO (php data base object) queries and connection with exception handling as it offers abstraction to database connection and encapsulated data base object as protected and inherited it to child classes.

We have recoded the front end for the student and teacher module user and admin. Using template engine format like every modern framework dose. On Back end Used the object-oriented concepts like inheritance, polymorphism, encapsulation, and abstraction.

As database using in every class so we made database class as parent and inherited People class which further have child classes Student and Teachers. These two classes performing crud, but little number of parameters were different we inherited signature of crud operations from people class and implement in the child classes by overriding concept. In this way make use of polymorphism. And inherited the user class which provide the login facility to admin from the data base class. Other modules of this project can also be converted in OOP based in the same way we have done these three.

# Reference and resources:

Codegrip

# Title of the post : ****Everything you need to know about Code Smells****

url

<https://www.codegrip.tech/productivity/everything-you-need-to-know-about-code-smells/>

PHP official documentation site

url

<https://www.php.net/manual/en/function.mysql-connect.php>

Centuples site Post title: MSQLI VS PDO

<https://code.tutsplus.com/tutorials/pdo-vs-mysqli-which-should-you-use--net-24059>

Chapter 3 Software Re-Factoring Slides

Book: Software Reengineering