

**Computer Science and Engineering**

**Cryptanalysis of Permutation Ciphers**

**Applied Cryptography**

Team Members: **Ajay Shenoy**

**George Lin**

**Shearyar Shamim Khan**

**Table of Contents**

**1 Introduction | 6**

1.1 Purpose | 6

1.2 Scope | 6

1.3 Identification | 6

1.4 Document Summary | 6

1.5 System Overview | 6

1.6 Document Overview | 7

**2 Reference Documents | 7**

**3 System Wide Design Designs | 7**

3.1 Software Component Architectural Design | 7

3.2 Software Architecture General Description | 8

3.3 Software Item Components | 8

3.4 Component Interface Identification | 10

3.5 Software Component Concept of Execution | 10

**4 Software Item Detailed Design | 10**

4.1 Structure | 10

*4.1.1 Software Unit Detailed Design | 10*

4.2 Static Relationship of Software Unit | 12

*4.2.1 Run-time Object Instances | 12*

4.3 Behavior | 12

*4.3.1 Sequence Diagrams | 12*

*4.3.2 Collaboration Diagrams | 16*

*4.3.3 Activity Diagrams | 17*

4.4 Concept of Execution | 18

4.5 Interface Design | 18

*4.5.1 Unique Identifier of Interface | 19*

*4.5.2 Interface Identification and Diagrams | 19*

**5 Implementation Architecture | 19**

5.1 All Active and Passive Classes Assigned to Components | 19

5.2 Diagrams of Physical Packaging of Logical Components | 19

**6 Deployment Architecture | 19**

6.1 Physical Deployment Architecture Diagram | 19

**7 Code (Pseudo) | 20**

**8 Dictionaries | 20**

**9 Software Item Computer Resource Utilization | 20**

**10 Requirements Traceability | 20**

10.1 Software Component-Level Requirements Traceability | 20

**11 System Design Testing | 20**

**12 Rationale | 21**

**Part 1:**

**Programming Report**

**1.1 Team Members and Specification Changes**

**Team Members:**

**Ajay Shenoy**

**George Lin**

**Shearyar Shamim Khan**

**No modifications were made** with respect to the following specifications

outlined in the project:

This cryptanalysis project consists of a software implementation of an algorithm that tries to decrypt an L-symbol challenge ciphertext computed using a permutation cipher. Informally speaking, your program's goal is to find the plaintext used to compute this ciphertext within a reasonable amount of time. Specifically, your program should print on screen something like "Enter the ciphertext:", obtain the ciphertext from stdin, apply some cryptanalysis strategy and output on screen something like "My plaintext guess is:" followed by the plaintext found by your strategy. In doing that, your program is allowed access to:

1. The ciphertext (to be taken as input from stdin)
2. A plaintext dictionary (to be posted on top of this web page), containing a number q of plaintexts, each one obtained as a sequence of space-separated words from the English dictionary
3. Partial knowledge of the encryption algorithm used (to be described below).

Your program is not allowed access to:

1. The key used by the permutation cipher.
2. Part of the encryption scheme (to be detailed below).

The plaintext is a space-separated sequence of words from the English dictionary (the sentence may not be meaningful). The key is a map from each English alphabet (lower-case) letter to a list of numbers randomly chosen between 0 and 102, where the length of this list is the (rounded) letter’s frequency in English text, as defined in the table below. The ciphertext is a space-separated sequence of encryptions of words, where each word is encrypted as a comma-separated list of numbers between 0 and 102, and these numbers are computed using the table below.

|  |  |  |
| --- | --- | --- |
| English letters | Average frequency | Key values (randomly chosen distinct numbers between 0 and 102) |
| a | 8 | k(a,1),…,k(a,8) |
| b | 1 | k(b,1) |
| c | 3 | k(c,1),…,k(c,3) |
| d | 4 | k(d,1),…,k(d,4) |
| e | 13 | k(e,1),…,k(e,13) |
| f | 2 | … |
| g | 2 |  |
| h | 6 |  |
| i | 7 |  |
| j | 1 |  |
| k | 1 |  |
| l | 4 |  |
| m | 2 |  |
| n | 7 |  |
| o | 8 |  |
| p | 2 |  |
| q | 1 |  |
| r | 6 |  |
| s | 6 |  |
| t | 9 |  |
| u | 3 |  |
| v | 1 |  |
| w | 2 |  |
| x | 1 |  |
| y | 2 |  |
| z | 1 |  |

The permutation cipher works as follows. It takes as input a plaintext from a message space and a key randomly chosen from a key space and returns a ciphertext.

* The message space is the set {<space>,a,..,z}^L. In other words the message m can be written as m[1]...m[L], where each m[i] is in {(space>,a,..,z}
* The ciphertext c can be written as c[1],...,c[L], where each c[i] is in {<space>,0,..,102}. To avoid ambiguities, cyphertext symbols are separated by a comma.
* The key space is the set of random maps from {0,..,26} to a permutation of all numbers in {0,…,102}, grouped in 26 lists, each list having length determined by column 2 of the table below.
* The encryption algorithm works as follows. A space in the plaintext is mapped to a space in the ciphertext. For each message character m[j], the algorithm finds m[j] in column 1 of the table below, and returns one of the keys in column 3 of the same row. The computation of which key is returned by the algorithm is based on a scheduling algorithm which is intentionally left unknown and is a deterministic algorithm (that is, it does not use new random bits) that may depend on j, L and the length of the list on that row.
* The decryption algorithm does the inverse process. It maps space to a space in the plaintext. On any ciphertext character different from a space, it finds the ciphertext character in column 3 of the table, and returns the column 1 plaintext letter that is on the same row.

For instance, assume k(b,1)=23, k(c,1)=11, k(c,2)=98, k(c,3)=5, k(g,1)=34, k(g,2)=56. Then the plaintext “cbcb gbgg gcb” may be encrypted as “98,23,5,23 34,23,56,34 34,11,23”.

We are currently choosing L=500, and a plaintext dictionary with q=5 plaintexts.

Your program will be scored based on two tests.

In the first test, your program will be run many times, each time on a new ciphertext, computed using the above encryption scheme and a plaintext randomly chosen from the plaintext dictionary, with a different scheduling algorithm. On the first execution, the scheduling algorithm will compute “j mod length(list)” and use this result to select the element of that position in the list. On the other executions, the scheduling algorithms will be more and more complex variations of this one.

In the second test, your program will be run a few times, each time on a new ciphertext, computed using a plaintext obtained as a space-separate sequence of words that are randomly chosen from the set of all English words (as in the attachment english\_words at the top of this page) and the above encryption scheme, with a different scheduling algorithm.

**1.2 Cryptanalysis Approach Used in the Program**

The Integrated University Departmental Information System is an organized approach intended to help universities regulate student records, laboratory administration, the ordering of goods and services, and the management of account receivable and payable along with a Bursar system. It should be linked to a wider University system responsible for staff salaries, student transcripts, etc. The IUDIS structure provides a new information system that is faster with a slicker user interface then the outdated and archaic systems, which makes it easier for students to access their records. Simultaneously, it will be effortless for staff and faculty to use the system for submitting grades and financial information. The new user interface allows for users to interact with the system seamlessly while being aesthetically pleasing, accommodating a broader spectrum of users. The structure of the information system will be rigorously secure to protect the confidentiality, integrity and availability of the individual user.

**1.3 Identification**

Integrated University Department Information System, B6, SDD Version 1.0, 2/3/2015

**1.4 Document Summary**

This document serves to describe the base level of the system, specifically this project most important components. Design details are provided along with feasibility of project. This document can generally be used to mass produce the software described in it.

**1.5 System Overview**

The Integrated University Departmental Information System is a smaller part of the larger university system that will interact with the budget system and the database system for the computer science department. The IUDIS is concerned with the users of the IUDIS, such as students, professors, advisors, other officials, and third party goods/service providers. The goal of the IUDIS is to please the user by having fast loading, easy to read, clearly arranged, and seamless navigation to all pages. Other factors that may affect the system is the load, the hard disk space, and the amount of down time the server faces. This product may be appear to be a combination of NYU’s Albert and Bursar systems.

**1.6 Document Overview**

Title Page

Review and Approvals

Table of Revisions

Table of Contents

Preface

List of Figures

This includes the Life Cycle and Methodology along with tools and techniques for documentations such as UML diagrams, Use Case diagrams, and Sequence Diagrams.

**2 Reference Documents**

Integrated University Department Information System, B6, Project Proposal Version 1.1

Integrated University Department Information System, B6, SRS Version 1.1

Integrated University Department Information System, B6, SPMP Version 1.0

Integrated University Department Information System, B6, SAS Version 1.0

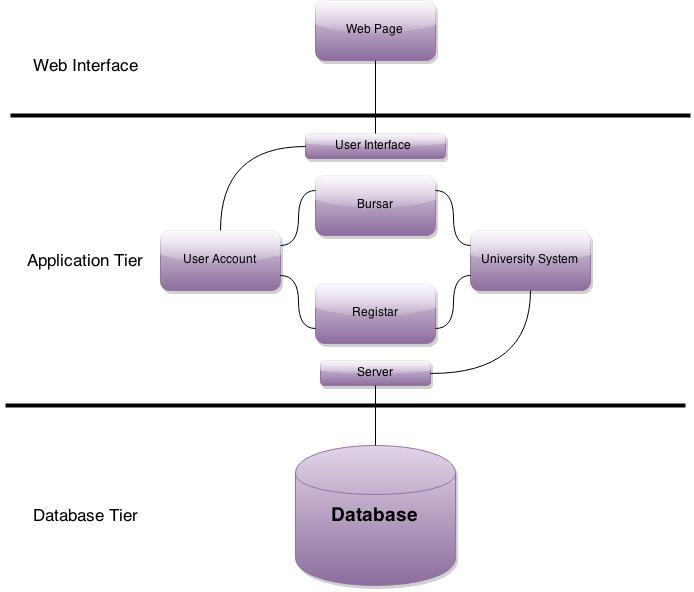
Integrated University Department Information System, B6, Project Proposal Version 1.2

Integrated University Department Information System, B6, RAS Version 1.0

Integrated University Department Information System, B6, SPMP Version 1.2

**3 System Wide Design Designs**

**3.1 Software Component Architectural Design**



**3.2 Software Architecture General Description**

The architecture has three tiers to it. They are the web interface, application server, and database. Such an architecture will enable a user to sign on from a location of their convenience. The application server will has accounts of users and user-specific information. For example, a student records will include registrar records and financial information along with billing and payment method. The student, professor, official, and administrator classes is inherited from user to enable code reuse.

**3.3 Software Item Components**

**Web Interface**

|  |  |
| --- | --- |
| **Component** | **Description** |
| Login Page | Composition that enables users to access their accounts |
| Student Page | Composition that allows a Student to access all of his/her information at once |
| Professor Page | Composition that allows a Professor to access all of his/her information at once |
| Official Page | Composition that allows an Official to access all of his/her information at once |
| Administrator Page | Composition that allows an Administrator to access all of his/her information at once |
| Manage Accounts Page | Composition that allows any user to manage their individual accounts |
| Registration Page | Composition that allows an |

**Application Tier**

|  |  |
| --- | --- |
| **Component** | **Description** |
| Student | Student’s account information |
| Professor | Professor’s account information |
| Official | Official account information |
| Administrator | Administrator account information |
| Register | Collective Registrar information |
| Bursar | Collective Bursar information |

**Database Tier**

|  |  |
| --- | --- |
| **Component** | **Description** |
| User table | Contains User IDs and Passwords |
| Classroom table | Contains Information about the classroom |
| Department table | Contains information about department |
| Course table | Contains information about courses |
| Professor table | Contains information about professor |
| Section table | Contains information about the section |
| Teaches table | Contains information about courses professors taught |
| Student table | Contains information about the students |
| Takes table | Contains information about what |
| Official table | Contains information about the official |
| Time\_slot table | Contains information about the time slot of the course |
| Prereq table | Contains information about the courses prerequisites |
| Third\_party table | Contains information about the third party company |
| Administrator table | Contains information about the administrator |

**3.4 Component Interface Identification**

User Interface - Web UI to Application Tier

Server - Application Tier to Database

**3.5 Software Component Concept of Execution**

User Interface executes when the following occurs:

Application starts a session

User interaction on page (for example a click to request transcript)

Information is sent to user interface from the application

Application executes when the following occurs:

User account is created or deleted

User logs in

Any of the actors call a method from their class for registrar or bursar

Database Server executes when the following occurs:

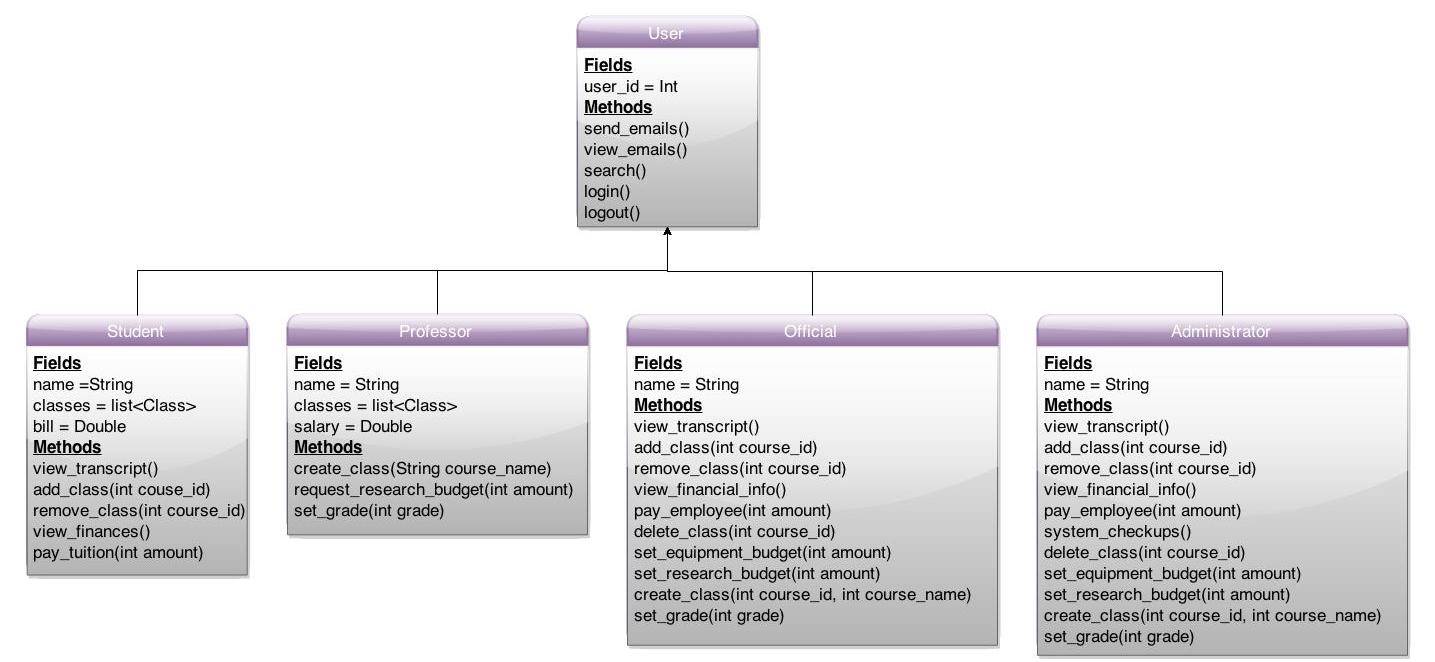
Application requests information

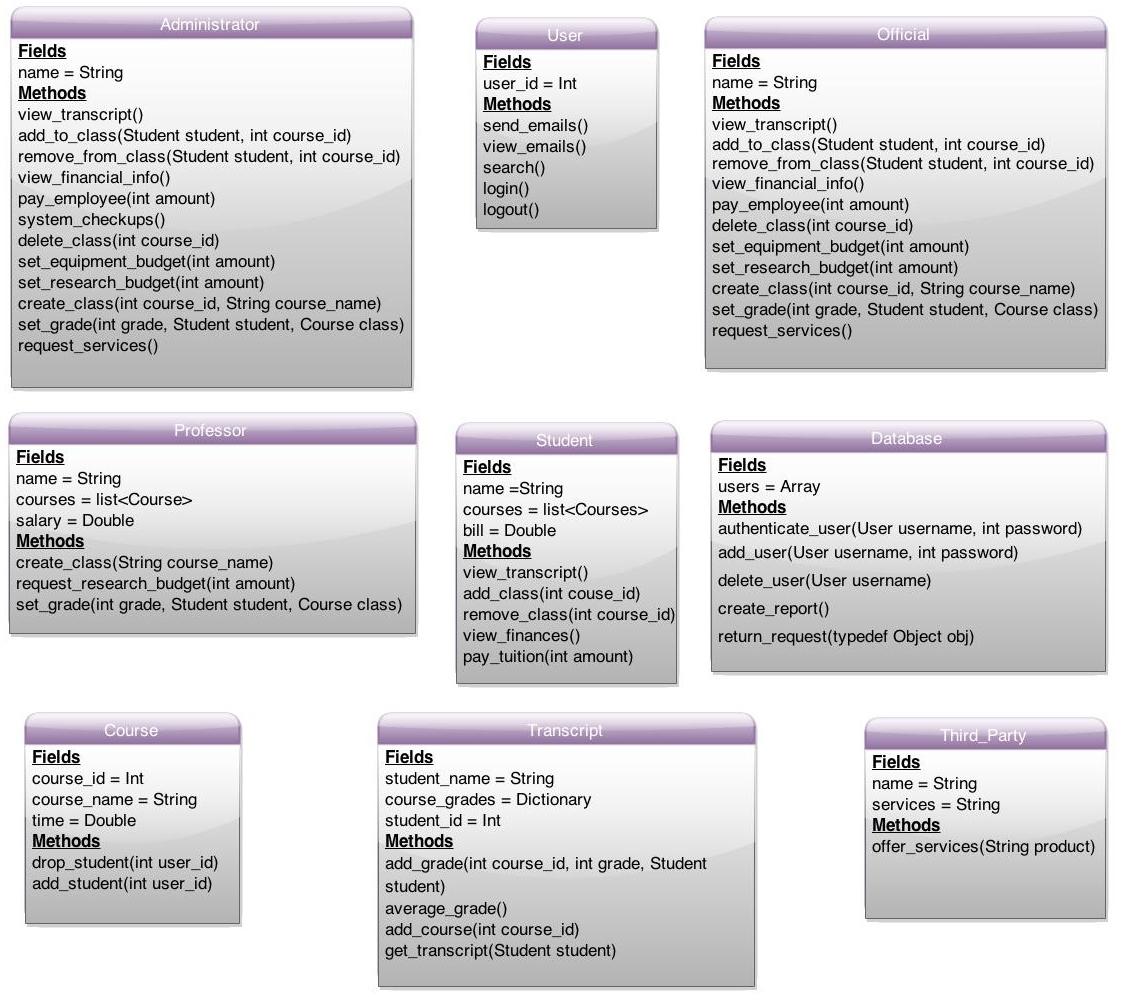
User account is created or deleted

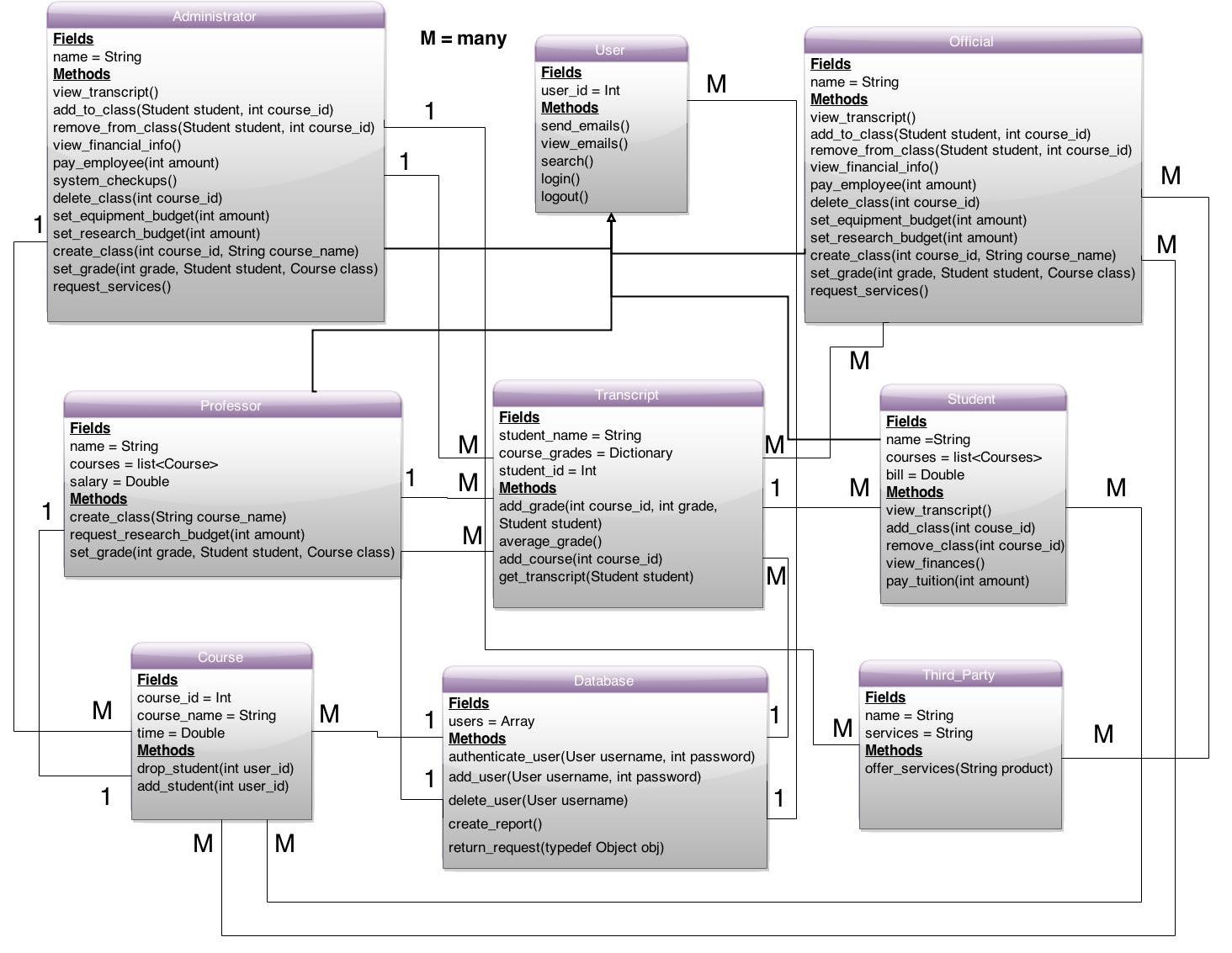
**4 Software Item Detailed Design**

**4.1 Structure**

**4.1.1 Software Unit Detailed Design**







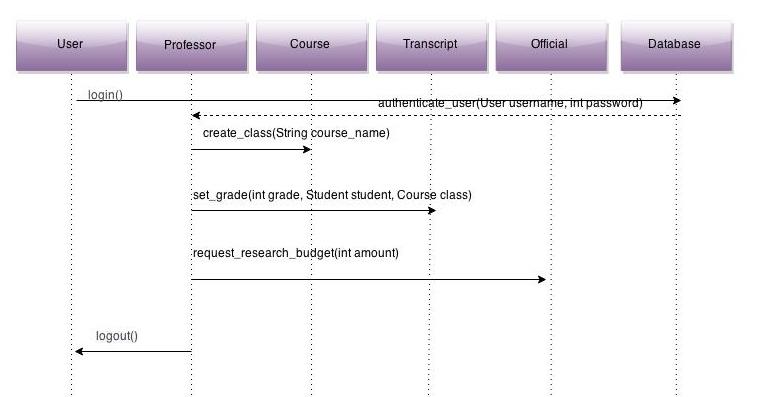
**4.2 Static Relationship of Software Unit**

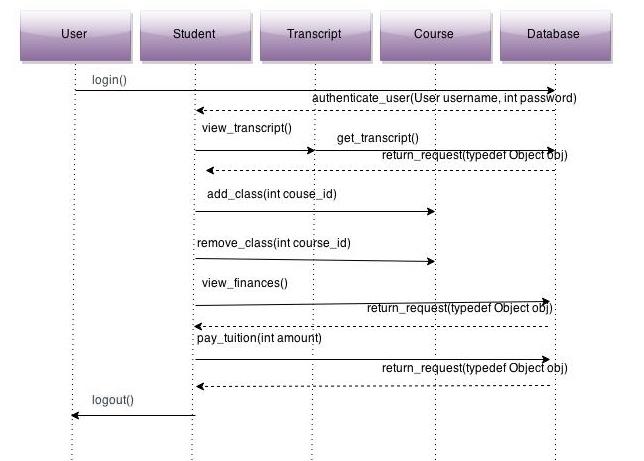
**4.2.1 Run-time Object Instances**

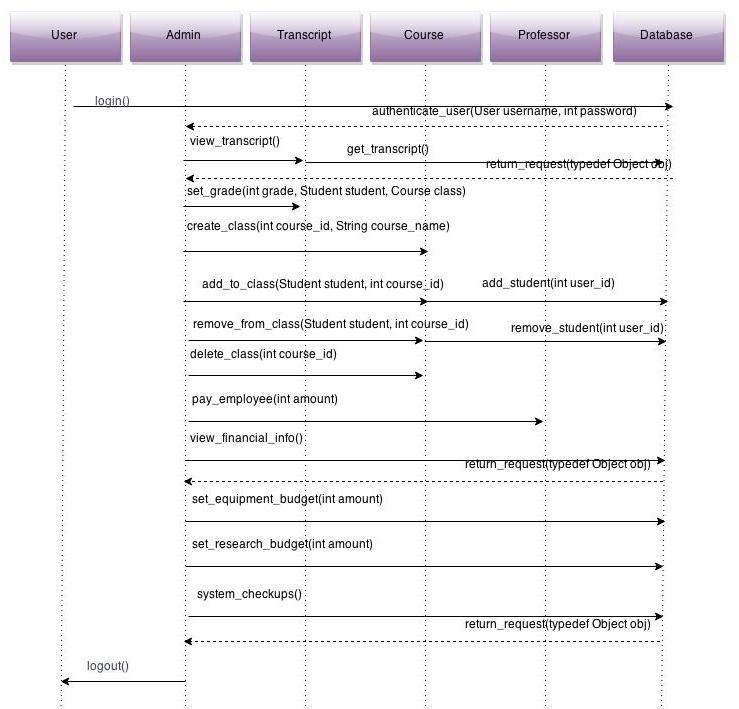
Refer to section 4.3.

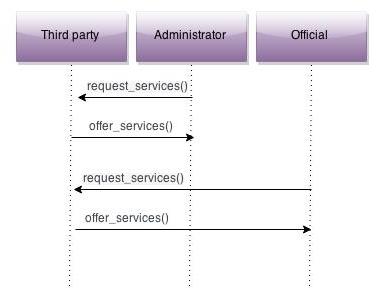
**4.3 Behavior**

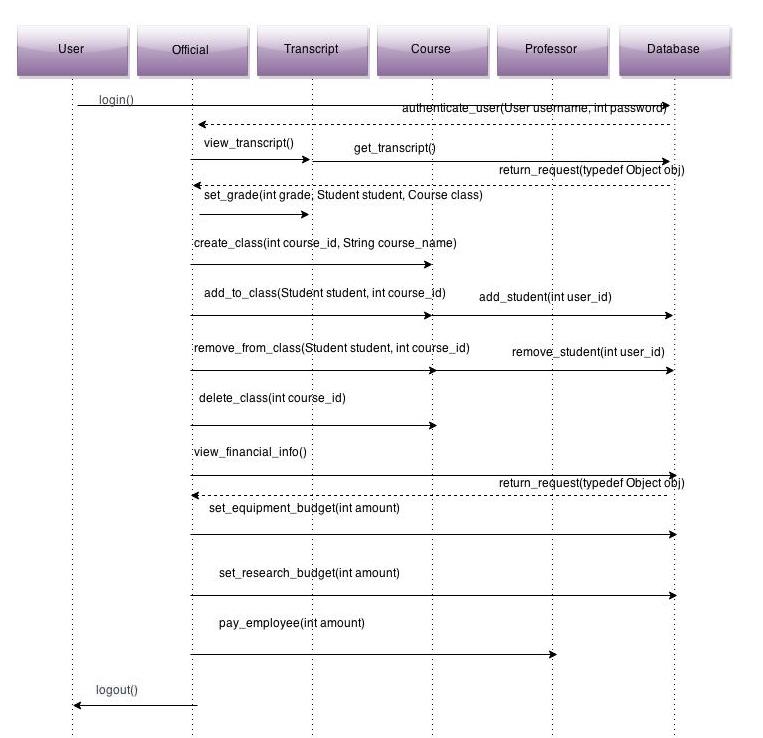
**4.3.1 Sequence Diagrams**



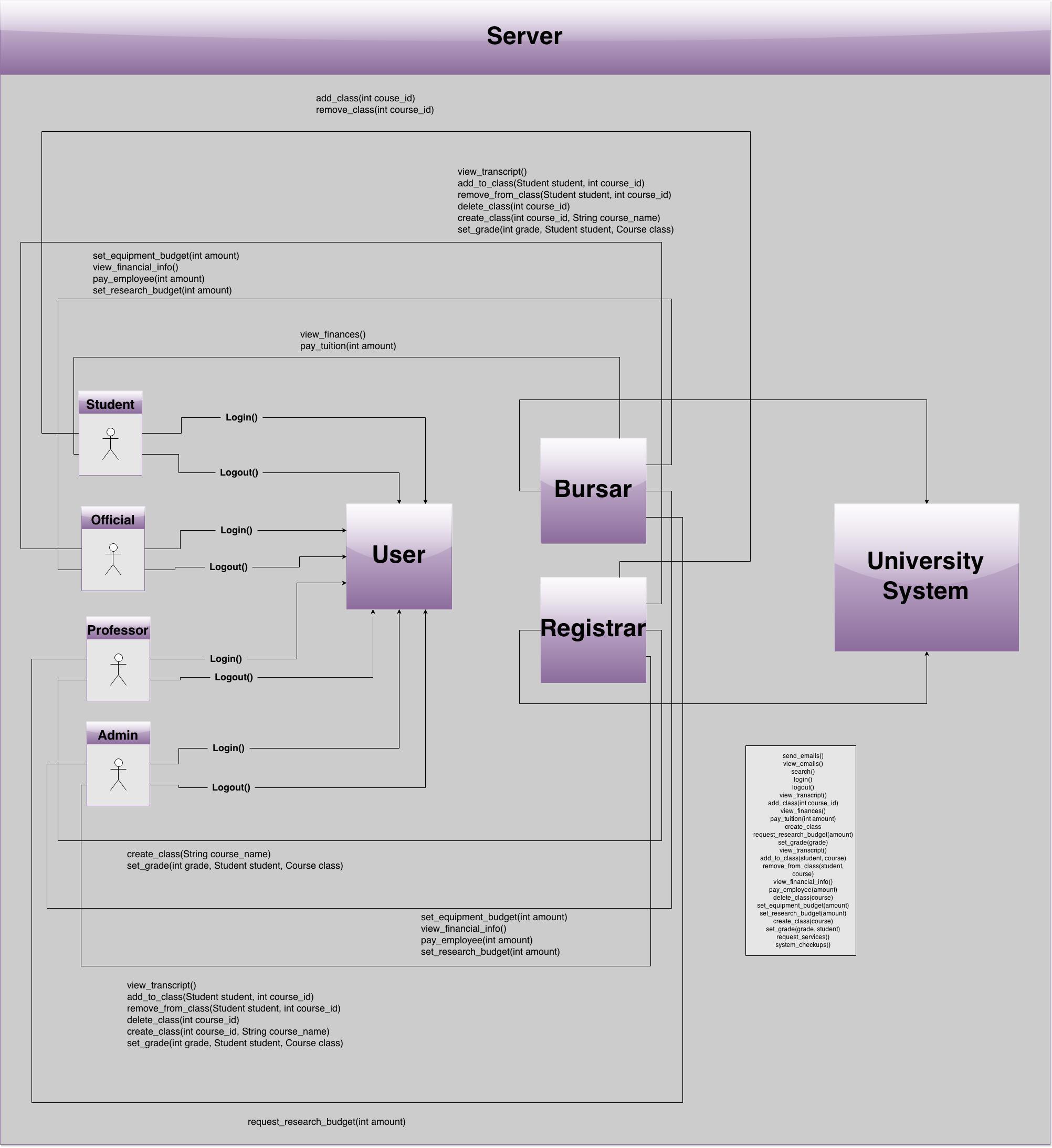




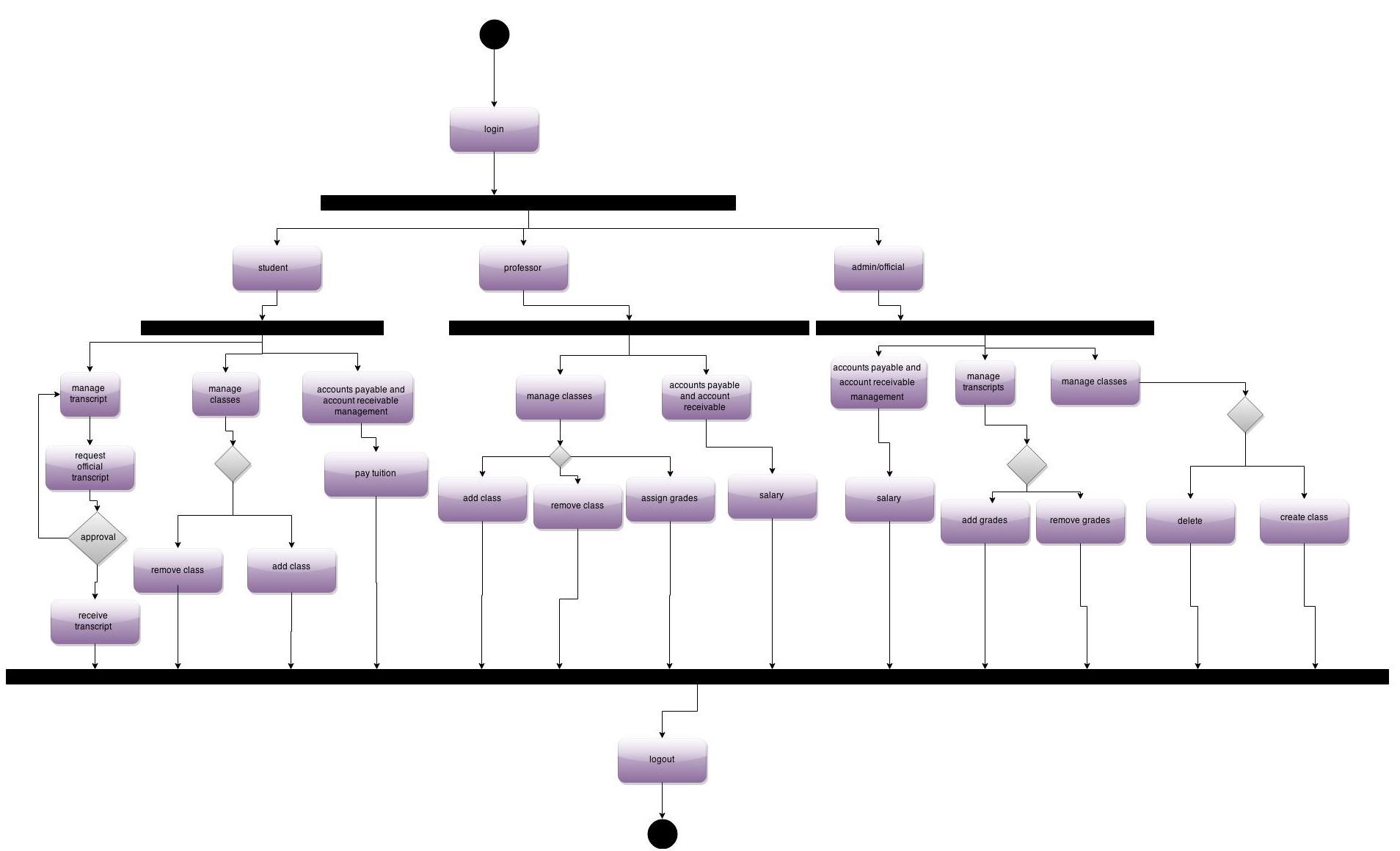




**4.3.2 Collaboration Diagrams**



**4.3.3 Activity Diagrams**

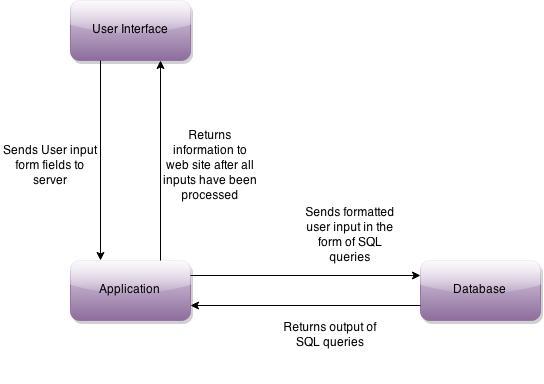


**4.4 Concept of Execution**

The IUDIS is reactive to user inputs. Once a user makes an input from the interface, the IUDIS becomes active. When an account is created for a user, the information (like username and password) that is entered will be validated and sent to appropriate database tables. When a user logs in, s/he is authenticated through the use of the database tables which will have the user’s username and password. Base on who the user is, a set of permissions will be given to the user upon logging in. Any changes invoked during the user’s session will require the user password or user’s session to be live. The change will then be invoked in the database.

**4.5 Interface Design**

**4.5.1 Unique Identifier of Interface**



**4.5.2 Interface Identification and Diagrams**

User Interface ← → Application Server

Application Server ← → Database

**5 Implementation Architecture**

**5.1 All Active and Passive Classes Assigned to Components**

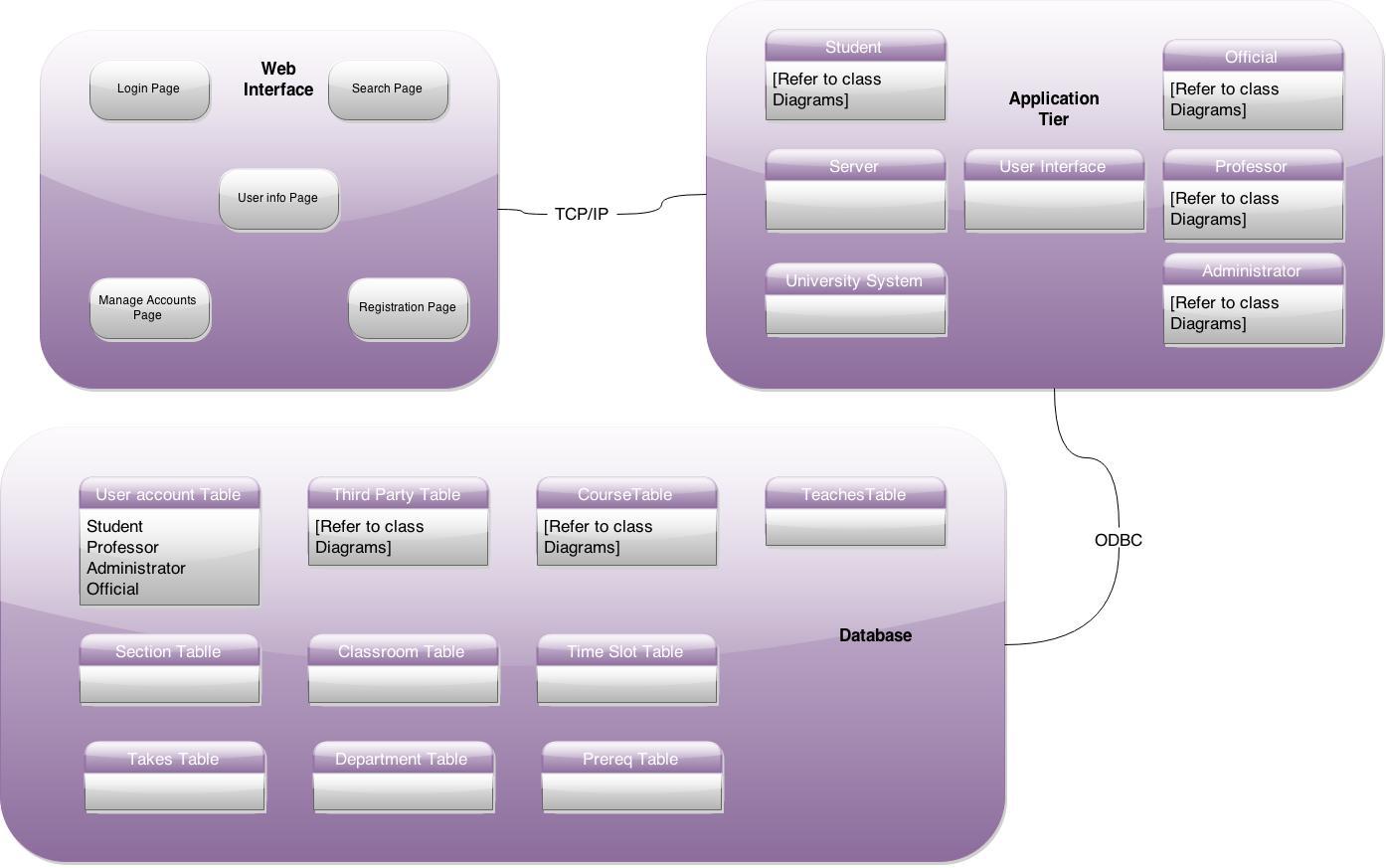
This section will not be discussed.

**5.2 Diagrams of Physical Packaging of Logical Components**

This section will not be discussed.

**6 Deployment Architecture**

**6.1 Physical Deployment Architecture Diagram**



**7 Code (Pseudo)**

Actual code is provided in the coding document.

**8 Dictionaries**

Refer to section 14.1.

**9 Software Item Computer Resource Utilization**

At worst case, the IUDIS will have 2TB of storage and 64GB of RAM. The number of users is expected to be around 60,000. At any given time, about 15,000 users are expected to be using the IUDIS. A quad-core processor is expected to work in this situation. The amount of users previously described, at least 50Mbps of bandwidth will be needed.

**10 Requirements Traceability**

**10.1 Software Component-Level Requirements Traceability**

All components are traceable to the Requirements and Analysis Specification (RAS) Document.

**11 System Design Testing**

Multiple tests must be performed. Such tests must target different components along with processes of the entire system. Furthermore, integration with the larger university system must be done likewise. The result of these tests will either approve or disapprove of this system based on its requirements as specified in the RAS and this document.

**12 Rationale**

The rationale behind the Integrated University Department Information System is to allow a seamless integration between huge university systems and the department system. The computer science department may have many small and independent functions from the main university system and so it should prove to be easy if the department handles its budget by itself while the larger university system tallies the department’s budget into its own budget. Furthermore, students’ grades and registration should be handled likewise.

**13 Notes**

**14 Appendices**

**14.1 Dictionaries**

Class Dictionaries

|  |  |  |  |
| --- | --- | --- | --- |
| User | Serves as a base for four classes | user\_id = int | send\_emails()  view\_emails()  search()  login()  logout() |

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Gives functionality specific to students | name = String  courses = list<Courses>  bill = double | view\_transcript()  add\_class(int course\_id)  remove\_class(int course\_id)  view\_finances()  pay\_tuiton(int amount) |

|  |  |  |  |
| --- | --- | --- | --- |
| Professor | Gives functionality specific to professors | name = String  courses = list<Courses>  salary = double | create\_class(String class\_name)  request\_research\_budget(int amount)  set\_grade(int grade, Student student, Course class) |

|  |  |  |  |
| --- | --- | --- | --- |
| Official | Gives functionality specific to officials | name = String | view\_transcript()  add\_to\_class(Student student, int course\_id)  remove\_from\_class(Student student, int course\_id)  view\_financial\_info()  pay\_employee(int amount)  delete\_class(int course\_id)  set\_equipment\_budget(int amount)  set\_research\_budget(int amount)  create\_class(int course\_id, String course\_name)  set\_grade(int grade, Student student, Course class)  request\_services() |

|  |  |  |  |
| --- | --- | --- | --- |
| Administrator | Gives functionality specific to administrators | name = String | view\_transcript()  add\_to\_class(Student student, int course\_id)  remove\_from\_class(Student student, int course\_id)  view\_financial\_info()  pay\_employee(int amount)  system\_checkups()  delete\_class(int course\_id)  set\_equipment\_budget(int amount)  set\_research\_budget(int amount)  create\_class(int course\_id, String course\_name)  set\_grade(int grade, Student student, Course class)  request\_services() |

|  |  |  |  |
| --- | --- | --- | --- |
| Third\_Party | Gives functionality specific to third party | name = String  services = String | offer\_services() |

|  |  |  |  |
| --- | --- | --- | --- |
| Transcript | Offers accessing and editing abilities of student transcripts for certain classes | student\_name = String  course\_grades = Dictionary  student\_id = int | add\_grade(int course, int grade, Student student)  average\_grade()  add\_course(int course\_id, Student student)  get\_transcript(Student student) |

|  |  |  |  |
| --- | --- | --- | --- |
| Course | Provides functionality to edit courses and associate courses with classes | course\_id = int  course\_name = String  time = double | add\_student(int user\_id)  drop\_student(int user\_id) |

|  |  |  |  |
| --- | --- | --- | --- |
| Database | Class that hands all storage abilities for other classes | users = Array | authenticate\_user(User username, int password)  add\_user(User username, int password)  delete\_user(User username)  create\_report()  return\_request(typedef Object obj) |

Method Dictionaries

|  |  |  |  |
| --- | --- | --- | --- |
| send\_email | Allows communication between other users | None | User |

|  |  |  |  |
| --- | --- | --- | --- |
| view\_email | Views messages | None | User |

|  |  |  |  |
| --- | --- | --- | --- |
| search | Allows for the exploration through the universities databases | None | User |

|  |  |  |  |
| --- | --- | --- | --- |
| login | Allows access to the university database depending on permissions | None | User |

|  |  |  |  |
| --- | --- | --- | --- |
| view\_transcript | Views grades on transcript | None | Student, Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| add\_class | Allows for the addition of courses | int course\_id | Student |

|  |  |  |  |
| --- | --- | --- | --- |
| remove\_class | Allows for the removal of courses | int course\_id | Student |

|  |  |  |  |
| --- | --- | --- | --- |
| view\_finances | View payables to university | None | Student, |

|  |  |  |  |
| --- | --- | --- | --- |
| pay\_tuition | Allows student to pay tuition | int amount | Student |

|  |  |  |  |
| --- | --- | --- | --- |
| create\_class | Allows professors to create a class | String course\_name | Professor |

|  |  |  |  |
| --- | --- | --- | --- |
| request\_research\_budget | Allows for a professor to request a research budget | int amount | Professor |

|  |  |  |  |
| --- | --- | --- | --- |
| set\_grade | Gives a grade to a student | int grade, Student student, Course class | Professor, Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| add\_to\_class | Adds a student to a course | Student student, int course\_id | Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| remove\_from\_class | Removes a student from a course | Student student, int course\_id | Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| view\_financial\_info | Allows for the viewing of the whole university’s finances | None | Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| pay\_employee | pays employee | int amount | Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| system\_checkups | Reports systems failures | None | Administrator |

|  |  |  |  |
| --- | --- | --- | --- |
| delete\_class | Removes a class from the Database | int course\_id | Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| set\_equiptment\_budget | Sets the equipment budget for professors that need to buy material for labs and such | int amount | Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| set\_research\_budget | Sets the research budget for professors that need to buy material for student’s pay and materials | int amount | Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| create\_class | creates a class on the database | int course\_id, String course\_name | Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| set\_grade | Sets the grade on the database for the student | int grade, Student student, Course class | Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| request\_services | Allows for the request of services from third party entities | None | Administrator, Official |

|  |  |  |  |
| --- | --- | --- | --- |
| offer\_services | Allows third party entities to offer services to the admin or officials within a university | None | Third\_party |

|  |  |  |  |
| --- | --- | --- | --- |
| add\_grade | Adds a grade for a course to a student’s transcript | int course\_id, int grade, Student student | Transcript |

|  |  |  |  |
| --- | --- | --- | --- |
| avrage\_grade | gives the averages of the student’s grades | None | Transcript |

|  |  |  |  |
| --- | --- | --- | --- |
| add\_course | Adds a course to a student's transcript | int course\_id | Transcript |

|  |  |  |  |
| --- | --- | --- | --- |
| get\_transcript | Returns the students transcript from the database | Student student | Transcript |

|  |  |  |  |
| --- | --- | --- | --- |
| drop\_student | Removes student from a course | int user\_id | Courses |

|  |  |  |  |
| --- | --- | --- | --- |
| add\_student | Adds a student to the course | int user\_id | Courses |

|  |  |  |  |
| --- | --- | --- | --- |
| authenticate\_user | Makes sure that the user that is using the system has permissions necessary to access the Database | User username, int Password | Database |

|  |  |  |  |
| --- | --- | --- | --- |
| add\_user | Adds a user to the Database | User username, int Password | Database |

|  |  |  |  |
| --- | --- | --- | --- |
| delete\_user | Removes a user from the database | User username | Database |

|  |  |  |  |
| --- | --- | --- | --- |
| create\_report | Displays system messages | None | Database |

|  |  |  |  |
| --- | --- | --- | --- |
| return\_request | Returns any data off the database to user with correct permissions | typedef Object obj | Database |

Attribute Dictionaries

|  |  |  |  |
| --- | --- | --- | --- |
| user\_id = int | Unique ID to access users | Simple | User |

|  |  |  |  |
| --- | --- | --- | --- |
| name = String | String for user’s name | Simple | Student, Professor, Official, Administrator |

|  |  |  |  |
| --- | --- | --- | --- |
| courses = list<Course> | Stores courses student takes or professor teaches | Simple | Student, Professor |

|  |  |  |  |
| --- | --- | --- | --- |
| bill = double | Student’s tuition amount | Simple | Student |

|  |  |  |  |
| --- | --- | --- | --- |
| salary = double | Professor’s pay | Simple | Professor |

|  |  |  |  |
| --- | --- | --- | --- |
| services = String | Description/Name of product/services | Simple | Third\_Party |

|  |  |  |  |
| --- | --- | --- | --- |
| course\_grades = Dictionary | Holds student’s grades | Simple | Transcript |

|  |  |  |  |
| --- | --- | --- | --- |
| student\_id = int | Unique student ID to get appropriate transcript | Simple | Transcript |

|  |  |  |  |
| --- | --- | --- | --- |
| course\_id = int | This defines a unique class ID | Simple | Course |

|  |  |  |  |
| --- | --- | --- | --- |
| course\_name = String | This defines the class name | Simple | Course |

|  |  |  |  |
| --- | --- | --- | --- |
| time = double | This defines the class time | Simple | Course |

|  |  |  |  |
| --- | --- | --- | --- |
| users = Array | This holds elements or users which this system works on | Simple | Database |

Event Dictionaries

|  |  |  |  |
| --- | --- | --- | --- |
| Authenticate | Determine who the user is depending on their username. From there, a user will see student, professor, official, or administrator page | Login Page | Student Page, Professor Page, Official Page, or Administrator Page |

|  |  |  |  |
| --- | --- | --- | --- |
| Enter Wrong Credentials | Return’s user back to the login page | Login Page | Login Page |

|  |  |  |  |
| --- | --- | --- | --- |
| Pick Menu Option | Displays a variety of options that a user can choose | Login Page | Student Transcript Page, Student Course Page, Student Bursar, Professor Transcript Page, Professor Course Page, Professor Salary Page, Debugging Page |

|  |  |  |  |
| --- | --- | --- | --- |
| Exit/Return | After a user is finished with whatever page s/he is currently on, the user will then go back to their original page. | Student Transcript Page, Student Course Page, Student Bursar, Professor Transcript Page, Professor Course Page, Professor Salary Page, Debugging Page | Student Page, Professor Page, Official Page, or Administrator Page |

**14.2 UML Diagrams**

**14.3 Schedule Tracking**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Artifact or Deliverable** | **Who** | **Estimated** | **Actual** | **Difference** |
| RAS | Munieshwar Ramdass | 10 | 13 | 3 |
| RAS | Ajay Shenoy | 12 | 13 | 1 |
| RAS | August Tan | 7 | 13 | 6 |
| RAS | Summary | 29 | 39 | 10 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Artifact or Deliverable** | **Who** | **Estimated** | **Actual** | **Difference** |
| SPMP | Munieshwar Ramdass | 2 | 2 | 0 |
| SPMP | Ajay Shenoy | 2 | 2 | 0 |
| SPMP | August Tan | 2 | 2 | 0 |
| SPMP | Summary | 6 | 6 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Artifact or Deliverable** | **Who** | **Estimated** | **Actual** | **Difference** |
| SDD | Munieshwar Ramdass | 20 | 18 | 2 |
| SDD | Ajay Shenoy | 18 | 18 | 0 |
| SDD | August Tan | 16 | 18 | 2 |
| SDD | Summary | 54 | 54 | 0 |

**Cumulative**

|  |  |  |  |
| --- | --- | --- | --- |
| **Who** | **Estimated** | **Actual** | **Difference** |
| Munieshwar Ramdass | 32 | 33 | 1 |
| Ajay Shenoy | 32 | 33 | 2 |
| August Tan | 25 | 33 | 8 |
| Summary | 89 | 99 | 10 |

**14.4 Defect Tracking**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Artifact or Deliverable** | **Who** | **Estimated** | **Actual** | **Difference** |
| RAS | Munieshwar Ramdass | 5 | 1 | 4 |
| RAS | Ajay Shenoy | 4 | 1 | 3 |
| RAS | August Tan | 6 | 1 | 5 |
| RAS | Summary | 15 | 3 | 12 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Artifact or Deliverable** | **Who** | **Estimated** | **Actual** | **Difference** |
| SPMP | Munieshwar Ramdass | 0 |  |  |
| SPMP | Ajay Shenoy | 0 |  |  |
| SPMP | August Tan | 0 |  |  |
| SPMP | Summary | 0 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Artifact or Deliverable** | **Who** | **Estimated** | **Actual** | **Difference** |
| SDD | Munieshwar Ramdass | 9 |  |  |
| SDD | Ajay Shenoy | 8 |  |  |
| SDD | August Tan | 7 |  |  |
| SDD | Summary | 24 |  |  |

**Cumulative**

|  |  |  |  |
| --- | --- | --- | --- |
| **Who** | **Estimated** | **Actual** | **Difference** |
| Munieshwar Ramdass | 14 |  |  |
| Ajay Shenoy | 12 |  |  |
| August Tan | 13 |  |  |
| Summary | 39 |  |  |





