Assessment 1 – Part 1: Requirements Analysis

Chun-Jie Hsieh – 25939707 Aditya Girishkumar Patel – 25723634   
Himel Mia - 25666468. Somit Nakarmi - 25670629

**1- Project Brief**

The task of this project Part 1 is to lay the groundwork for designing a complete student self-service course selection system at UTS through detailed requirements analysis, user story planning and UML modeling to ensure that the subsequent development phases can proceed smoothly.

**In this report, we will:**

1. Deconstruct requirements: Translate the specifications provided by UTS into specific User Stories and create a User-Story Backlog.
2. Design the System Architecture: Draw a UML Use Case Diagram to clarify the flow of interactions between students, administrators, and the system.
3. Planning Class Structure: Define core objects (e.g. Student, Enrolment, Subject) and their relationships through UML Class Diagram to ensure the clarity and extensibility of the code structure.

**2- User-Story Backlog**

**User-Story:**

**Student Actions**

***Story: Take a student to the student system if credentials are correct*** *→ 100*

***Story: Match username and password with the ones on file*** *→ 101*

***Story: Show an error message if credentials do not match*** *→ 102*

***Story: Enrol in a subject*** *→ 103*

***Story: Remove a subject from the enrolment list*** *→ 104*

***Story: View current enrolment list*** *→ 105*

***Story: Change the password*** *→ 106*

***Story: Show an error message if more than four subjects are enrolled*** *→ 107*

***Story: Validate the email format*** *→ 108*

***Story: Validate the password format*** *→ 109*

***Story: Display a unique six-digit student ID*** *→ 110*

**Admin Actions**

***Story: Log in as an admin*** *→ 201*

***Story: View all registered students*** *→ 202*

***Story: View students organized by grade*** *→ 203*

***Story: Categorize students into PASS/FAIL categories*** *→ 204*

***Story: Remove a student from the system*** *→ 205*

***Story: Clear all student data*** *→ 206*

**Admin - Subject Management**

***Story: Generate a unique 3-digit subject ID*** *→ 301*

***Story: Assign a mark for a subject*** *→ 302*

***Story: Calculate and assign a grade based on mark*** *→ 303*

***Story: Generate random grades and subject IDs upon enrolment*** *→ 304*

**Admin - Data Management**

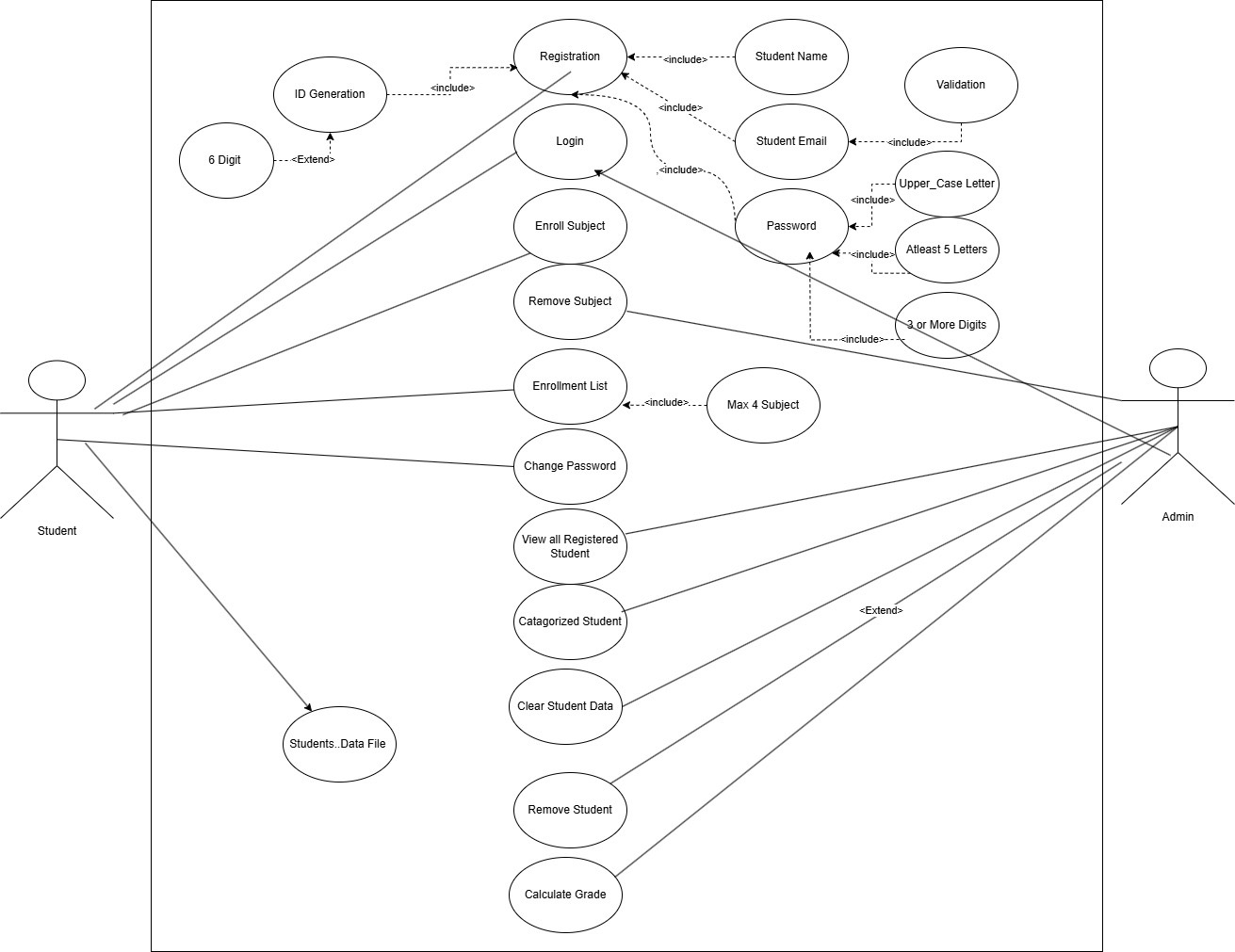
***Story: Export student data to a file*** *→ 305*

***Story: Show an error message if admin credentials are incorrect*** *→ 306*

**User-Story Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **User** | **Action** | **Result** | **Function** |
| **100** | **Student** | **Log in with valid number** | **Student is directed to the student system** | **login\_Student()** |
| **101** | **Student** | **Enter username and password** | **System matches the entered username and password with stored data** | **match\_student()** |
| **102** | **Student** | **Enter incorrect username or password** | **Show an error message indicating the mismatch** | **Error\_Message()** |
| **103** | **Student** | **Enrol in a subject** | **Add a new subject to student’s enrolment list** | **enrolIn\_Subject()** |
| **104** | **Student** | **Remove a subject from the enrolment list** | **Remove the subject from the enrolment list** | **remove\_Subject()** |
| **105** | **Student** | **View current enrolment list** | **Display the list of enrolled subjects** | **viewEnrolmentList()** |
| **106** | **Student** | **Change the password** | **Update the password for the student** | **changePassword()** |
| **107** | **Student** | **Enrol in more than four subjects** | **Show an error message if more than four subjects are enrolled** | **SubjectLimit()** |
| **108** | **Student** | **Enter a valid email address** | **System valid the email format (**[**firstname.lastname@university.com**](mailto:firstname.lastname@university.com)**)** | **Email()** |
| **109** | **Student** | **Enter a valid password** | **System validates the password format** | **Password()** |
| **110** | **Student** | **View student unique ID** | **Display a unique six-digit student ID** | **StudentID()** |
| **201** | **Admin** | **Log in as an admin** | **Admin is directed to the admin subsystem** | **loginAdmin()** |
| **202** | **Admin** | **View all registered students** | **Display the list of all students and their details** | **viewAllStudents()** |
| **203** | **Admin** | **View students organized by grade** | **Display students sorted by their grades** | **viewStudentsByGrade()** |
| **204** | **Admin** | **Categorize students into PASS/FAIL categories** | **Display a list of students categorized by PASS or FAIL status** | **categorizeStudents()** |
| **205** | **Admin** | **Remove a student from the system** | **Remove a student’s details from the system** | **removeStudent()** |
| **206** | **Admin** | **Clear the entire student data** | **Remove all data from the students file** | **clearAllStudentsData()** |
| **301** | **Admin** | **Generate unique 3-digit subject ID** | **Assign a unique ID between 001 and 999 for each subject** | **SubjectID()** |
| **302** | **Admin** | **Assign a mark for a subject** | **Randomly assign a mark between 25 and 100 for each subject** | **SubjectMark()** |
| **303** | **Admin** | **Calculate and assign a grade based on mark** | **Assign grade based on the mark: Z, P, C, D, HD** | **calculateGrade()** |
| **304** | **Admin** | **Generate random grades and subject IDs** | **Assign random grades to subjects upon enrolment** | **generateGrades()** |
| **305** | **Admin** | **Export student data to a file** | **Save all student data to students.data file** | **saveDataToFile()** |
| **306** | **Admin** | **Handle login error (invalid admin credentials)** | **Show error message if admin credentials are incorrect** | **showAdminErrorMessage()** |

**3- UML Use Case Diagram**



This Use Case Diagram illustrates how users (actors) interact with a system using the Unified Modeling Language (UML). There are two main actors in the diagram: Student and Admin.

**Use Cases and Relationships:**

Registration of Students (Includes Validation):

Email validation requires the following format: [firstname.lastname@university.com](mailto:firstname.lastname@university.com). Student, name, email, and password are required.

**Rules for Passwords:**

It begins with a capital letter. At least five letters are present.

Three or more digits after it.

**Student ID generation**

* Range: 1–999,999 o Format in six digits: Prefix ID with zeros if it contains fewer than six digits (002340, for example).
* Shown as "ID Generation" expanding "Registration" in the diagram.

**Student Operations (After Login)**

Once registered, students can:

• Enrol in a Subject *(Auto-assigns subject with a unique 3-digit ID and a random mark between 25-100)*

**Grade Calculation:**

▪ < 50 → Z

▪ 50-64 → P

▪ 65-74 → C

▪ 75-84 → D

▪ ≥ 85 → HD

• Remove a Subject *(From enrolment list)*

• View Enrolment List *(Shows enrolled subjects and marks/grades)*

• Change Password *(Allows students to update their credentials.)*

• Enrolment Limit *(System prevents enrolling in more than 4 subjects.)*

• Data Storage *(Student enrolment details are stored in students' data files.)*

**Admin Operations**

• View All Registered Students *(Access all student records from students’ data.)*

• Categorize Students and Calculate Grade

*(Pass/Fail based on marks → PASS ≥ 50, FAIL < 50.)*

• Remove a Student *(Deletes an individual student from the system.)*

• Clear Student Data *(Deletes all student records from students’ data.)*

**Relationships:**

• Includes (<<include>>): Represents a mandatory dependency, meaning that one use case always includes another.

• Extends (<<extend>>): Represents an optional dependency, meaning that the extended use case occurs only under certain conditions.

**Summary:**

• Student can register, log in, enrol in subjects (with a max of 4), remove subjects, and manage their passwords.

• Admin has broader control, such as viewing and categorizing students, clearing data, removing students, and calculating grades.

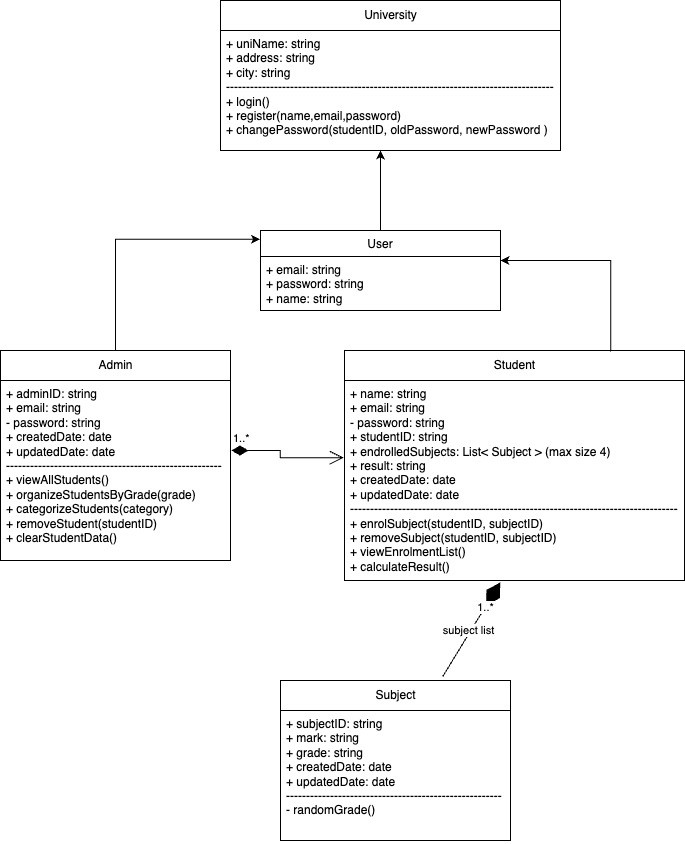
• The Registration process includes multiple validation steps to ensure correct user

Information.

• ID Generation extends to include a six-digit ID system.

• UML Use Case Diagram provides an overview of the system's functionalities from both the student's and admin's perspectives.

**4- UML Class Diagram**



This UML diagram represents a university management system where different roles interact with each other. The main entity is the University, which has basic attributes like name, address, and city. The university provides functions such as login, register, and change Password to manage user access. The User class is a parent class for both Admin and Student, inheriting attributes like email, password, and name. This means that both admins and students share common login credentials but have different functionalities. The system enables stud. The main entity is the **University**, which has basic attributes like name, address, and city. The university provides functions such as **login, register, and change Password** to manage user access. The **User** class is a parent class for both **Admin** and **Student**, inheriting attributes like **email, password, and name**. This means that both admins and students share common login credentials but have different functionalities. The system enables students to enrol in subjects while allowing admins to organize and manage students based on grades or categories.

**Admin** class includes functionalities like **view All Students, organize Students By Grade, categorize Students, remove Student, and clear Student Data**. This indicates that admins have full control over student records, including categorization and removal. The **Student** class contains attributes like **student ID, enrolled Subjects (up to 4), and results, along with actions like enrol Subject, remove Subject, view Enrolment List, and calculate Result**. This means that students can enrol in courses, drop courses, and view their enrolment details. The relationship bet details. The relationship between **many** relationship, meaning that an admin manages multiple students.

**Subject** class stores information about individual courses, including **subject ID, mark, grade, and timestamps for creation and updates**. The subject class is linked to the **Student** class through an association, meaning each student can have multiple subjects. The function **random Grade()** suggests that the system can generate random grades for subjects, likely for testing or evaluation purposes. The overall system allows smooth management of university operations by defining a structured way to handle student records, subject enrolments, and administrative controls. It ensures that students and admins have distinct roles while interacting efficiently within the system.