Assessment 1 - Part 1: Requirements Analysis

CLIUniApp & GUIUniApp Development Project

Subject: 32555 Fundamentals of Software Development
Group: Coding Ninjas

Team Members:

- Vipin Kumar - 25742622 - Task 1: User Stories & Backlog
- Ved Mohith Chilimbi - 25534948 - Task 2: Use Case Diagram & Explanation
- Yuna Son - 26041590 - Task 3: Class Diagram & Explanation
- Fangfang Tang - 14645473 - Task 4: Final Report Editing & Submission

Table of Contents

1. Introduction

1.1 Project Overview

This document presents the requirements analysis for the development of CLIUniApp and GUIUniApp, an interactive student enrollment system for a local university. The program aims to provide a comprehensive platform that allows students to self-enroll in courses, as well as administrative tools for university staff. Since the system automates grade management and streamlines the enrolling process, both administrators and students should find it more effective and user-friendly.

1.2 Project Scope

The system encompasses two main components:

- **CLIUniApp**: A command-line interface application providing complete functionality for both students and administrators
- **GUIUniApp**: A graphical user interface component (challenge task) focused on core student operations

The project covers student registration, subject enrollment (maximum of four subjects per student), automated grade assignment, and administrative student management. Multi-semester enrollment and other advanced features are outside the scope of this assessment.

1.3 System Requirements Summary

The university requires a system that supports:

- Student registration with email and password validation
- Secure login functionality
- Subject enrollment management (maximum 4 subjects per student)
- Automated grade assignment and calculation
- Administrative student management capabilities

- File-based data persistence using students.data

1.4 Analysis Methodology

This requirements analysis follows a structured approach comprising:

1. User Story Development: Transforming requirements into actionable user stories with unique identification system
2. Use Case Modeling: Creating comprehensive UML use case diagrams illustrating system actors and interactions
3. Class Design: Developing detailed UML class diagrams with proper relationships and specifications

1.5 Document Structure

This document is organized according to the assessment requirements:

- Section 2 presents the complete user story backlog with requirement mapping
- Section 3 provides the UML use case diagram with detailed explanations
- Section 4 contains the UML class diagram with comprehensive class descriptions
- Section 5 summarizes the analysis and discusses implementation readiness

2. User-Story Backlog

The following table presents the comprehensive user story backlog for CLIUniApp and GUIUniApp. A distinct ID is given to each user narrative, which details the action taken by the user (or system), the anticipated result or outcome, and the related function. System Operations (300s), Admin Functions (400s), Non-Functional & Error Handling (500s), Registration & Authentication (100s), and Enrolment (200s) are the functional domains into which user stories are divided. This organised backlog is used as a guide for UML modelling, system design, and implementation.

2.1 Legend / Short Form Descriptions

- **CLI** – Command Line Interface: text-based program where users type commands.
- **GUI** – Graphical User Interface: window-based program where users click buttons/menus.
- **ID** – Unique identifier for each user story (grouped by feature: 100s = Registration/Login, 200s = Enrolment, etc.).
- **PASS/FAIL** – Student classification based on UTS grading rules (PASS ≥50, FAIL <50).
- **Grade Codes** – Standard UTS grade scale:

- **HD** = High Distinction (≥85)
- **D** = Distinction (75–84)
- **C** = Credit (65–74)
- **P** = Pass (50–64)
- **Z** = Fail (<50)

- **students.data** – File where all student details and enrolments are stored.
- **Backlog Status** – All stories are in *Backlog* stage for Part 1

2.2 Backlog

ID	User	Action	Result / Outcome	Function
Registration & Authentication (100s)				
101	Student	Register with name, email, and password	New account created with unique 6-digit student ID	Register
102	System	Validate email format	Only "@university.com" addresses accepted	Register
103	System	Validate password pattern	Password starts with uppercase, ≥5 letters, ≥3 digits	Register
104	System	Generate unique 6-digit ID	ID auto-padded with zeros if <6 digits	Register
105	Student	Attempt to register with duplicate email	System rejects; error shown	Register
106	Student	Login with credentials	Student subsystem access granted if credentials valid	Login
107	Student	Login with invalid credentials	Error message displayed; no access	Login
108	Student	Logout	Session ended; returned to login screen	Logout

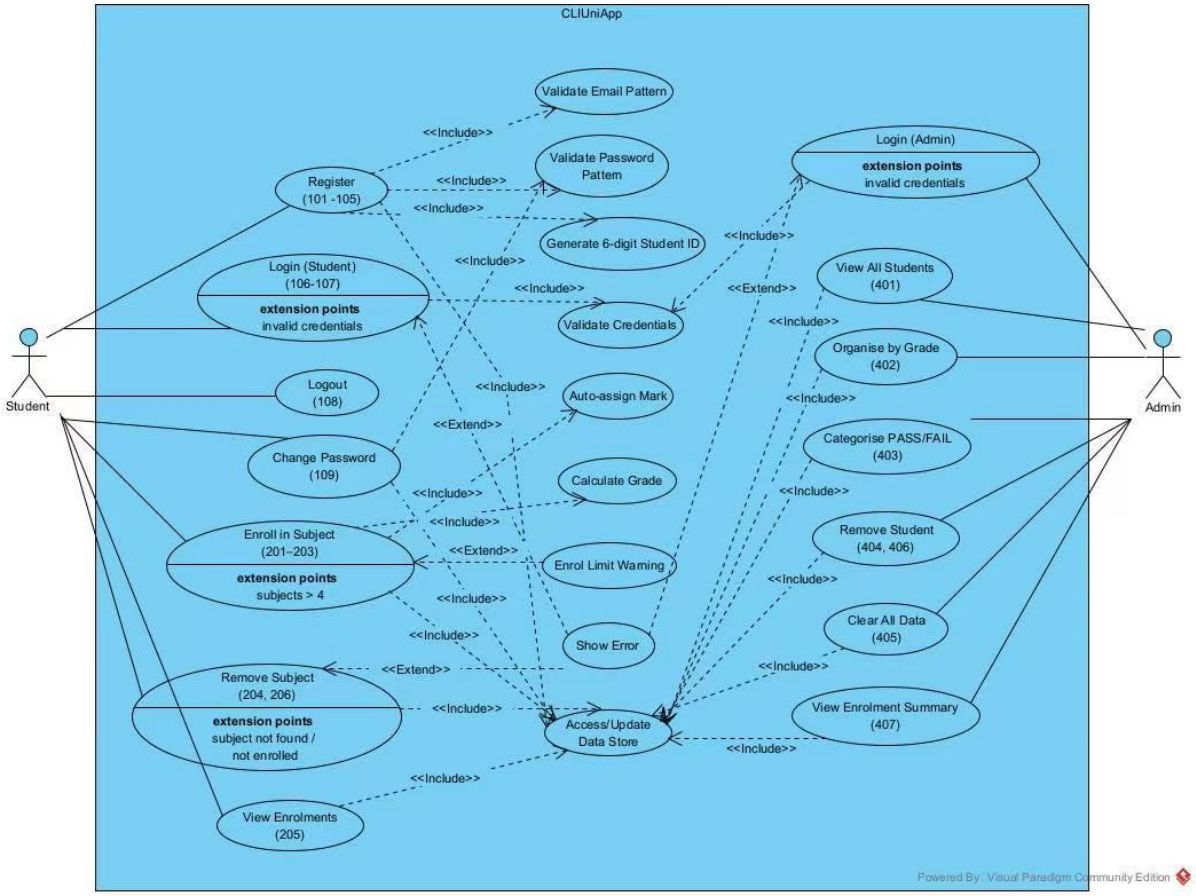
109	Student	Change password	Password updated after validation	Change Password
Enrolment (200s)				
201	Student	Enrol in subject	Subject added with unique 3-digit ID	Enrol Subject
202	System	Auto-generate subject mark (25–100)	Mark assigned; grade (Z, P, C, D, HD) calculated	Auto Grading
203	Student	Attempt to enrol in >4 subjects	System blocks enrolment; warning shown	Enrol Limit
204	Student	Remove a subject	Subject removed from enrolment list	Drop Subject
205	Student	View current enrolments	List of subjects with IDs, marks, and grades	View Enrolments
206	Student	Attempt to remove non-existent subject	Error message displayed	Drop Subject
207	Student	Attempt to enrol with invalid subject ID	System prevents action; error shown	Enrol Subject
System Operations (300s)				
301	System	Save student data to students.data	File updated with registration and enrolment details	File Storage
302	System	Retrieve data from students.data	Data loaded for CLI and Admin subsystems	File Storage
303	System	Handle empty file scenario	System informs user; no crash	File Storage
304	System	Backup students.data	Copy of data maintained	File Storage

Admin Functions (400s)				
401	Admin	View all registered students	Full list displayed with IDs and enrolments	Admin Management
402	Admin	Organise students by grade	Students grouped by Z, P, C, D, HD	Admin Management
403	Admin	Categorise students as PASS/FAIL	PASS (≥50) / FAIL (<50) classification shown	Admin Management
404	Admin	Remove individual student	Selected student deleted from students.data	Admin Management
405	Admin	Clear all student data	Entire students.data file emptied	Admin Management
406	Admin	Handle invalid student ID during removal	Error message shown; no deletion	Admin Management
407	Admin	View enrolment summary report	Displays number of students per grade category	Admin Reporting
Non-Functional & Error Handling (500s)				
501	System	Display clear error messages	User-friendly feedback; no raw technical errors shown	Error Handling
502	System	Prevent unauthorised admin access	Only valid admin credentials grant access	Security
503	System	Prevent empty login fields in GUI	Error message shown; login blocked	GUI Error Handling

504	System	Prevent enrol >4 subjects in GUI	Error shown; no enrolment allowed	GUI Error Handling

3. UML Use Case Diagram

3.1 UML Diagram



3.2 Explanation

System boundary

CLIUniApp lets students self-enrol in subjects (max 4), and lets admins manage student records. All data is stored in students.data so both subsystems can read/write it.

Actors & goals

Student - register; log in/out; change password; enrol (auto 3-digit subject ID, auto mark 25-100, auto grade Z/P/C/D/HD); remove subject; view enrolments (cap = 4 subjects, warn if exceeded).

Admin - authenticate and manage students: view all, organise by grade, categorise PASS/FAIL (50 cut-off), remove student, clear all data, view enrolment summary. Uses the same students.data file.

Use cases

Student Use Cases

- **Register (101-105)** - Create student with a validated email/password and auto 6-digit ID.
- **Login (Student) (106-107)** - Grant access on valid credentials; show error on invalid.
- **Logout (108)** - end session and return to login.
- **Change Password (109)** - Update password after validation.
- **Enrol in Subject (201-203)** - Add subject (auto 3-digit ID), assign random mark 25-100, calculate grade, warn if >4.
- **Remove Subject (204, 206)** - Drop a subject; error if subject not found.
- **View Enrolments (205)** - List enrolled subjects with IDs/marks/grades.

Admin Use Cases

- **Login (Admin)** - Authenticate admin (no admin registration in scope).
- **View All Students (401)** - List all students with IDs and enrolments.
- **Organise by Grade (402) -** Group students by Z/P/C/D/HD.
- **Categorise PASS/FAIL (403)** - Classify PASS (≥50) / FAIL (<50).
- **Remove Student (404, 406)** - Delete a student (error if invalid ID).
- **Clear All Data (405)** - Empty the students.data file.
- **View Enrolment Summary (407)** - Show counts per grade category.

Common behaviours

- **Validate Email Pattern** - Only “@university.com” addresses.
- **Validate Password Pattern** - Starts uppercase, ≥5 letters, ≥3 digits.
- **Generate 6-digit Student ID** - Pad with zeros if needed.
- **Validate Credentials** - Match username/password to stored data.
- **Auto-assign Mark** - Random 25-100 on enrol.
- **Calculate Grade** - Apply UTS grade scale (Z/P/C/D/HD).

- **Show Error** - Standardised error display.
- **Enrol Limit Warning** - Trigger when enrolment exceeds 4.
- **Access/Update Data Store** - Read/Write students.data (save, load, handle empty, backup).

Relationships

- Actor ↔ Use case associations.

    - Student initiates Register, Login/Logout, Change Password, Enrol/Remove/View Enrolments.
    - Admin initiates Login and all management/report UCs.

- <<include>> (required reused steps)

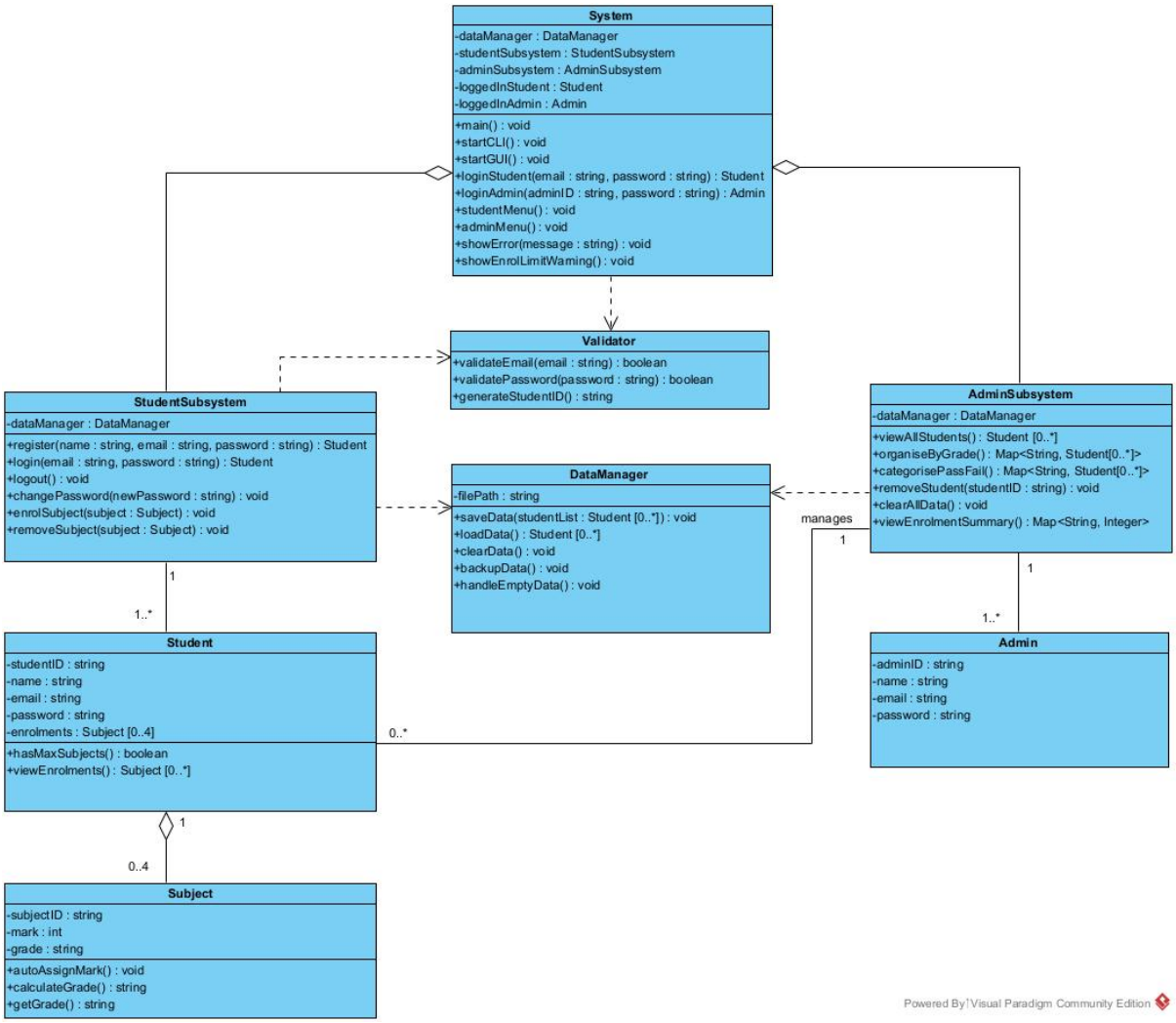
    - Register <<include>> Validate Email Pattern, Validate Password Pattern, Generate 6-digit Student ID, Access/Update Data Store.
    - Login (Student/Admin) <<include>> Validate Credentials.
    - Change Password <<include>> Validate Password Pattern, Access/Update Data Store.
    - Enrol in Subject <<include>> Auto-assign Mark, Calculate Grade, Access/Update Data Store.
    - Remove Subject <<include>> Access/Update Data Store; View Enrolments <<include>> Access/Update Data Store.
    - Admin UCs <<include>> Access/Update Data Store to read (view/organise/categorise/summary) or write (remove/clear).

- <<extend>> (conditional/exception paths)

    - Show Error <<extend>> Login (Student/Admin) and Remove Subject with extension condition as ‘invalid credentials’ and ‘subject not found/not enrolled’
    - Enrol Limit Warning <<extend>> Enrol in Subject when subjects > 4.

4. UML Class Diagram

4.1 UML Diagram



4.2 Class Diagram Explanation Document

Class

1. System (CLIUniApp)

- Purpose: Entry point of the application. Provides interaction between students, admins, and subsystems.
- Fields:
  - dataManager: DataManager
  - studentSubsystem: StudentSubsystem
  - adminSubsystem: AdminSubsystem
  - loggedInStudent: Student
  - loggedInAdmin: Admin
- Methods:
  - main(): void → Starts the application and invokes the CLI or GUI entry point.

- startCLI(): void → Launches the command-line interface for user interaction.
- startGUI(): void → Launches the graphical user interface (if supported).
- loginStudent(email: String, password: String): Student → Authenticates a student and returns the logged-in student object.
- loginAdmin(adminID: String, password: String): Admin → Authenticates an admin and returns the logged-in admin object.
- studentMenu(): void → Displays and navigates options available to students (delegates to StudentSubsystem).
- adminMenu(): void → Displays and navigates options available to admins (delegates to AdminSubsystem).
- showError(message: String): void → Displays an error message to the user.
- showEnrolLimitWarning(): void → Displays a warning when a student attempts to enrol in more than the maximum allowed subjects.

2. Student

- Purpose: Represents a registered student with enrolment capabilities.
- Fields:
  - studentID: String
  - name: String
  - email: String
  - password: String
  - enrolments: List<Subject>
- Methods:
  - hasMaxSubjects(): boolean → Checks if the student has reached the maximum subject enrolment limit (4).
  - viewEnrolments(): List<Subject> → Returns a list of all subjects currently enrolled by the student.

3. Subject

- Purpose: Represents a subject enrolled by a student.
- Fields:
  - subjectID: String
  - mark: int
  - grade: String
- Methods:
  - autoAssignMark(): void → Automatically assigns a random or system-defined mark to the subject.
  - calculateGrade(): String → Computes the grade based on the assigned mark (e.g., HD, D, C, P, F).
  - getGrade(): String → Returns the grade of the subject.

4. Admin

- Purpose: Represents an authenticated admin user.
- Fields:

- adminID: String
- name: String
- email: String
- password: String

5. AdminSubsystem

- Purpose: Provides management operations for admin users.
- Fields:
  - dataManager: DataManager

- Methods:
  - viewAllStudents(): List<Student> → Retrieves a list of all registered students.
  - organiseByGrade(): Map<String, List<Student>> → Groups students based on their grades.
  - categorisePassFail(): Map<String, List<Student>> → Divides students into pass/fail categories.
  - removeStudent(studentID: String): void → Deletes a student record by ID.
  - clearAllData(): void → Removes all student data by calling DataManager.
  - viewEnrolmentSummary(): Map<String, Integer> → Returns a summary of enrolments per subject.

6. StudentSubsystem

- Purpose: Provides interaction functions for logged-in students.
- Fields:
  - dataManager: DataManager

- Methods:
  - register(name: String, email: String, password: String): Student → Creates a new student record and assigns a generated ID.
  - login(email: String, password: String): Student → Authenticates a student using email and password.
  - logout(): void → Logs out the currently active student session.
  - changePassword(newPassword: String): void → Updates the student's password in the system.
  - enrolSubject(subject: Subject): void → Adds a subject to a student's enrolment list if under the limit.
  - removeSubject(subject: Subject): void → Removes a subject from a student's enrolment list.

7. DataManager

- Purpose: Manages persistence of student and enrolment data in students.data.
- Fields:
  - filePath: String
- Methods:

- saveData(studentList: List<Student>): void → Saves the list of students and enrolments to persistent storage.
- loadData(): List<Student> → Loads all student records from persistent storage.
- clearData(): void → Deletes all student records.
- backupData(): void → Creates a backup of the current student data file.
- handleEmptyData(): void → Handles cases when no data exists.

8. Validator (Utility Class)

- Purpose: Ensures correct input patterns and ID generation.
- Methods:
  - validateEmail(email: String): boolean → Verifies that the email follows the correct format.
  - validatePassword(password: String): boolean → Checks if a password meets security rules (e.g., min length).
  - generateStudentID(): String → Creates a unique student ID for new students.

Relationships

- System ↔ StudentSubsystem / AdminSubsystem: Aggregation
- The System aggregates the subsystems and manages them. This represents a "has-a" relationship where the subsystems can exist independently of the main system.
- StudentSubsystem ↔ Student / Validator / DataManager: Dependency / Association
- The StudentSubsystem manages Student data and depends on Validator and DataManager to perform its tasks.
- AdminSubsystem ↔ Admin / Student / DataManager: Dependency / Association
- The AdminSubsystem uses Admin data for authentication, manages Student records, and depends on the DataManager for file operations.
- Student ↔ Subject: Aggregation
- A Student aggregates a list of Subject objects, but each Subject can exist on its own.

Multiplicity

- Student → Subject: 0..4 (A student can enrol in between 0 and 4 subjects.)
- StudentSubsystem → Student: 1..* (The Student Subsystem manages one or more students.)
- AdminSubsystem → Student: 0..* (The Admin Subsystem can manage zero or more students.)
- System → DataManager / StudentSubsystem / AdminSubsystem: 1 (The System has exactly one of each subsystem and one Data Manager).

5. Conclusion

5.1 Analysis Summary

This report provides a comprehensive requirements analysis and preliminary design for the University's student self-enrolment system (CLIUniApp & GUIUniApp). We meticulously followed the project brief, transforming the client's needs into structured user stories and mapping them to a requirements table. This process ensured full functional coverage and traceability. A UML use case diagram was employed to illustrate the relationships between the three main actors—the student, the administrator, and the system—and the features of the system. To further define the fundamental entities, like Student and Subject, we developed a UML class diagram that included information on their attributes and functions.

5.2 Requirements Coverage

All of the important specifications listed in the project brief are entirely met by our design. By use of user stories (e.g., IDs 101-109 for authentication and registration, and ID 201 for enrolment), we have made certain that all essential student functions, like login, enrolment, registration, changing passwords, and accessing enrolment lists—are fulfilled. We've also incorporated the administrator's needs by designing a dedicated subsystem for student management and data maintenance. All non-functional requirements, including email and password validation rules and the auto-generation of student and subject IDs, have also been carefully considered.

5.3 Design Consistency

All design elements within this report maintain a high degree of consistency. The UML class diagram's classes, properties, and methods are used to realise the functionality that our user stories immediately correlate to the use cases outlined in the UML use case diagram. Using the Student class, for instance, the "Register Student" use case will be implemented, with precise actions such as "Validate Password" matching to a specific method within that class. This end-to-end coherence guarantees that our design makes sense and is prepared for a smooth transition from the analysis stage to the development stage.

5.4 Technical Considerations

In terms of technical considerations, we chose to use a students.data file as the primary data storage medium to support all CRUD (Create, Read, Update, Delete) operations. This design simplifies data management and allows for data sharing between both the CLI and GUI applications. Furthermore, we've considered the robustness of data validation by designing the system to automatically verify email domains and password formats, which enhances system stability and security. The mechanism for auto-generating and padding student and subject IDs with zeros demonstrates our attention to data formatting and integrity.

5.5 Implementation Roadmap

Looking ahead to the Part 2 development phase, we'll first implement the core functionalities of the CLIUniApp, including both the student and admin subsystems. This will involve file I/O operations to handle the students.data file. Following that, we'll develop the GUIUniApp as the challenge task, focusing on its login and enrolment features and handling related exceptions, such as empty fields or exceeding the enrolment limit. To ensure efficient development, we'll adhere to the class structure defined in the UML class diagram, systematically implementing its attributes and methods to ensure the code's structure aligns perfectly with our design documentation.

5.6 Quality Assurance

Our plan is to ensure the software works correctly and reliably. During development, we'll focus on testing every requirement from our user stories. This includes checking if the login, registration, and enrolment features work as they should. We'll also test for common errors, like what happens when a user enters a wrong password or tries to enroll in too many subjects. Finally, we will verify that all data is correctly saved to and loaded from the students.data file to ensure data integrity.

5.7 Project Readiness

This concludes our analysis and design phase. We have successfully created all required documents, including the user story backlog, UML use case diagram, and UML class diagram. We now have a clear **blueprint for development**. Our team understands the project goals and our individual tasks for Part 2. We are confident and ready to begin the implementation phase and build the application as designed.