Software Requirements Specification

for

Student Residence Management System

Version 1.0 approved

Prepared by Brock Young, Alikhan Madeni, Ryan Cropley and Jose Contreras

Thompson Rivers University

April 2, 2024

Table of Contents

List of Figures

List of Tables

1. Product Description

The proposed Student Resident Management System (SRMS) will be developed to address a lack of efficient residency organization at the local university. The current system is outdated and done manually through paper and file keeping which is a tedious process and reduces effective employee time. Additionally, there have been issues of lost records and misplaced resident information. The new updated system will aim to address the shortcomings of the current one, with a digital interface to keep track of student and manager records and ease of use. The centralization of the system through a digital interface will increase efficiency while organizing the files in such a way for easy access.

1.1 Introduction

The modernization of administrative and management systems can be directly related to digitalization and increasing technological innovation. Prior to digital management systems, paper was the most used record keeping medium. The use of paper meant large management systems took up large physical space and had to be organized by hand in sophisticated ways to ensure efficiency. Physical storage also meant that documents could easily be lost or misplaced, leading to frequent data loss. Additionally, the documents would have to be manually created and fetched, leading to tedious work and inefficiency. Access to the information stored in the system was limited to those in a close proximity who could physically access the files, with no option for remote access. The creation of digital administration systems aims to tackle all these shortcomings and more.

The designed Student Resident Management System (SRMS) aims to replace the current paper-based approach of managing student residency records, aiming to increase areas such as efficiency, availability, scalability, and reusability. The modernization of the administration system is especially pertinent for the university residencies because of the frequent modification of records and fluctuation of students. The system will allow managers to access records remotely, rather than having to modify physical copies. Scalability of the SRMS will be significantly larger than a paper-based model, with dynamic servers and databases according to the demand.

While existing management software may exist to address the needs of the university, designing a proprietary system has a number of benefits. By facilitating the entire design of the system, specific functionality and complete customization are embedded into the software. The system isn't bound by the whim of external companies changing rates or costs at random, instead relying on internal pricing. Security of residential data and information is also able to be handled internally rather than being controlled by a third party.

The SRMS will be developed iteratively, with multiple prototypical developments and stages of design. Future work and outlined requirements may refer to the final delivery rather than the prototype designed in cohesion with this document. Details mentioned in the following sections are subject to change according to the development timeline.

The following sections will outline key information essential in the development and creation of the Student Resident Management System. The design problem to be addressed will be outlined, including the associated functions, objectives and requirements. The vision, business requirements, users and scope will be discussed to create a full product description. The functional requirements will be discussed next, including associated use cases of the system. Next the data requirements will be detailed, before the non-functional and interface requirements sections are established. Next the brainstormed solution alternatives and final solution are compared. The teamwork and project management sections outline how the team collaborated to accomplish the

designed solution. Finally, the conclusion and future work sections cement the key points discussed in the report and what may be improved upon in future iterations.

1.2 Design Problem

1.2.1 Problem Definition

The university requires a software solution for a Student Residence Management System to replace the current lacking system. The current paper-based approach has issues relating to limited scalability, loss of documents, large effort for basic tasks (view, or modify resident/residence data), reusability and more. The nature of having a physical system to handle large amounts of data means minimal automation; every record must be added manually which proves to be inefficient at a large scale. Additionally, documents can be easily lost or misplaced, with no trivial method of backup. The paper-based system also requires a sophisticated organization system to be able to find documents, and a tedious process to modify documents. The designed solution aims to address all the above issues with minimal cost overhead and satisfying all specified design requirements.

1.2.2 Design Requirements

1.2.2.1 Functions

The functional requirements are briefly outlined in this section and further expanded upon in Section 2: Functional Requirements.

The designed solution for the SRMS should accomplish the following functions:

- Management of users
  - The final deployed solution should facilitate the management of users, including adding and removing new residents, managers, modifying their data, and more
- Payment Services and Billing
  - The final deployed solution should enable the payment services and billing of students to enable new residents to purchase their rooms and view their billing information
- Management of Student Residences
  - The final deployed solution should record information regarding the student residences in the system, including available beds and assigned students
- Security and Privacy
  - The final deployed solution should maintain only necessary user data and in a secure way to ensure user privacy

1.2.2.2 Objectives

The objectives of the system are briefly discussed in this section and are related to and expanded by sections 1.2.2.3 Constraints and 4. Non-Functional Requirements.

The designed solution for the SRMS should aim to achieve the following key objectives:

- Availability
  - The system should be largely available and not suffer from large amounts of downtime or maintenance
- Reliability
  - The system should perform in a largely predictable way and achieve the same outputs for repeated inputs
- Modifiability
  - The SRMS should be designed in such a way to enable ease of modification

- Scalability
    - The SRMS should be able to scale according to the number of residents in the system
- Reusability
    - The system should be well documented and built in such a way to facilitate reuse in other systems for future use

1.3 Product Vision

The new SRMS will improve employee efficiency reducing tedious tasks and freeing time for less arduous work. Student records will be more easily accessible and general usability of the system will improve through a more streamlined process. The new system will serve as a foundation to be further iterated on and be updated as needed for future use.

For University residents, employees, managers and administrators
Who suffer from outdated managerial systems regarding residency at the university
The Student Resident Management System
Is an administrative management tool
That will reduce time employees spend on arduous tasks such as manually adding or removing residents, viewing room occupancy, assigning managers to residents or general resident file maintenance. The new system will serve as a baseline for further iterations and will be built in such a way to facilitate easy upgrades and updates for new residential buildings at the university or changes in resident policy.
Unlike the existing paper based Resident Management System
Our product will improve employee morale and serve as a centralized management system to track and update all things related to records of residents at the university. The system will increase employee efficiency, decrease resident query wait times, and reduce chances of misplaced records.

1.4 Business Requirements

The SRMS has the following business requirements:

- The SRMS will serve as a university wide central management system for administrative users to view or manipulate student residential records.
- The first version of the SRMS must be developed and usable within 6 months of project proposal and approval.
- The SRMS budget and projected maintenance and upkeep costs for the next 5 years should cost no more than 3% of the university's annual gross income in 2024.
- The SRMS must receive 80% employee approval by final implementation.
- The SRMS must not have downtime exceeding 5 minutes every six months (excluding downtime for upgrades scheduled every 3 months)
- The SRMS must reduce perceived employee effort regarding residential records by no less than 15%.
- The SRMS must not displace existing residential records and serve as iterative upgrade rather than complete overhaul and discarding of the old system.

1.5 Users and Other Stakeholders

Stakeholder	***Roles***	***Interests***	***Influence***	***Needs***	***Concerns***
Residential administrative employees	*System users, manipulation of records*	*Interacting with system on daily basis with minimal friction*	*High*	*Easy to learn and use interface, real-time system updates, organization of records*	*Learning new interface, migration to new system*
Student residents	*Recordees, indirect users of system*	*Proper record keeping, maintenance of records, knowledge of their manager*	*Medium*	*Accurate and consistent storage of records, view who their manager is*	*Misplaced or mismanaged records*
System developers	*Project developers*	*Functional system delivered on time and on budget, scalable system for future iteration*	*High*	*Clear and unambiguous project requirements, feedback from users, system testing, appropriate allocation of resources*	*Changes in scope or requirements, unforeseen bugs or implementation issues*
IT Department	*Maintenance and upkeep of system*	*Easily viewable and readable source code, sophisticated comments, interface with other university software*	*Low*	*Access to code for trouble shooting, interfacing of other university software, remote access and development server*	*Coding bad practice, difficulty debugging, hard to access code*
University Administration	*Stakeholder and regulatory body*	*Program represents university in good light, adheres to regulations*	*Medium*	*Documentation outlining adherence to regulations, proper usage of company logo, image*	*Breach of university and local regulations and guidelines, reducing image of university*

1.6 Project Scope

The SRMS will be delivered through various iterations to ensure a functional design with prioritized requirements to speed along deployment and increase exposure to early testers for feedback. The initial release will have a minimal set of features to provide basic functionality and be improved and upgraded on subsequent releases. The initial iteration will be released within 12 months of project proposal and acceptance, and subsequent releases will be tentatively scheduled for every 3 months following for at least 3 more releases.

Initial Release:
Key features to be included on initial release include:

- Interactive user interface
- Addition of student residents to rooms

- Removal of student residents from rooms
- Editing of student resident records
- Viewing and search of current residents and their rooms and managers
- Assignment of managers to student residents

Further Releases:
- Tutorial upon first time opening
- Login for enhanced security
- Encryption for records
- Addition of new residential buildings

1.7 Assumptions

In order for the stated requirements to be successful, certain assumptions have been considered. The success of the overall project relies on these assumptions to be valid. The assumptions are listed as below:

- AS-00: Requirements are elicited in a thorough and sophisticated manner to ensure all requirements are documented and known by stakeholders and developers
- AS-01: The information as stated in the SRS and related documents is regularly reviewed and updated with the most recent information
- AS-02: The developers and stakeholders involved in the process of creating the system are always updated with the most recent documentation and data relating to the requirements
- AS-03: Developers span a diverse background to ensure all technical needs to develop the system are met
- AS-04: Resources allocated to the project take into account potential delays or other errors which may otherwise push back project delivery
- AS-05: User data and usage of system is fixed and predictable to ensure server and related infrastructure is properly adequate for capacity and scalability.
- AS-06: The software is to be developed for the local university and can be scaled to accommodate new residencies but not outside the common locality

1.8 Constraints

Similar to assumptions, the project is limited to the constraints imposed on it by the stakeholders and clients. These constraints must all be satisfied in order to deem the project a success. Failure to satisfy any number of these constraints will be deemed a failure. The constraints are listed as below:

- The first iteration of the SRMS must be delivered within 12 months of project proposal being accepted
- The budget used in development of SRMS must not exceed 3% of the universities annual budget for 2024
    - Must not exceed 1.8% for initial deployment
    - Must not exceed 3% for all deployments and maintenance for first 5 years
- System must comply with university housing and residency policies
- System should allow capacity of at least 25,000 residents, with possibility to scale indefinitely in future iterations
- System should be accessible through web and application portal
- System data must be encrypted and adhere to university security constraints by third deployment

2. Functional Requirements

2.1 Use Case Model

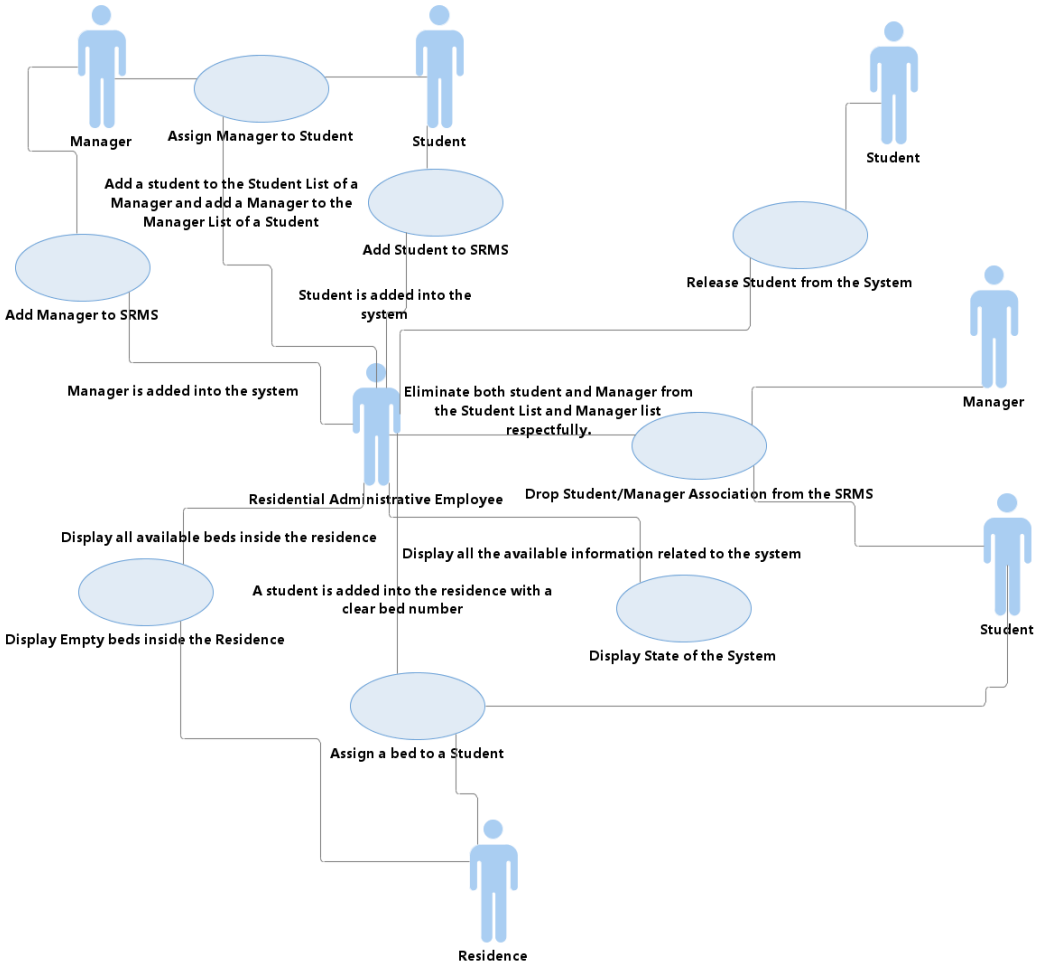


Figure #1: Student Resident Management System Use Case Diagram

Use Case 1: Add a Student to SRMS.

Actors: Residential Administrative Employees and Students

Stakeholders and Needs:

Residential Administrative Employees-To interact with the system in order to add new student into the system.

Student Residents- To have their data taken and stored into the system.

Preconditions:

The system is working correctly, and Residential Administrative Employee can see the menu.

There are available rooms vacant in the residence.

The new student is enrolled to the university the residence belongs to.

Postconditions:

The information saved in the record is only stored on the system if all fields are filled out with valid information

If any issues occur while adding a record the IT Department will be contacted.
Trigger: Residential Administrative Employee indicates they would like to add a new record through system interaction menu.
Basic Flow:
1. Residential Administrative Employee indicates they would like to add a new student record
2. SRMS validates that there are rooms available for new students
3. Residential Administrative Employee adds data for each required field including student identification
4. Residential Administrative Employee confirms submission of new record.
5. SRMS validates that the new record has been added to the system
Extensions:
5a There is some problem with the SRMS sending the information
5a1. The SRMS will redirect the Residence Administrative Employee into the main Menu.
5a2. The Administrative Employee should try again to enter the data.
5a3. If system keeps failing, the IT department will be contacted shortly.

Use Case 2: Add a Manager to SRMS.
Actors: Residential Administrative Employees and Manager
Stakeholders and Needs:
Residential Administrative Employees-To interact with the system in order to add new student into the system.
Manager- To have their data taken and stored into the system.
Preconditions:
The system is working correctly, and Residential Administrative Employee can see the menu
The Manager is part of the staff from the Institution where the resident currently is located.
Postconditions:
The information saved in the record is only stored on the system if all fields are filled out with valid information
If any issues occur while adding a record the IT Department will be contacted.
Trigger: Residential Administrative Employee indicates they would like to add a new Manager record through system interaction menu.
Basic Flow:
1. Residential Administrative Employee indicates they would like to add a new manager record
2. SRMS validates that there are rooms available for new students
3. Residential Administrative Employee adds data for each required field including Manager identification
4. Residential Administrative Employee confirms submission of new record.
5. SRMS validates that the new record has been added to the system
Extensions:
5a There is some problem with the SRMS sending the information
5a1. The SRMS will redirect the Residence Administrative Employee into the main Menu.
5a2. The Administrative Employee should try again to enter the data.
5a3. If system keeps failing, the IT department will be contacted shortly.

Use Case 3: Display empty beds of the System
Actors: Residential Administrative Employees and Manager
Stakeholders and Needs:
Residential Administrative Employees-To visualize all the available beds in a given residence.
Preconditions:
The system is working correctly, and Residential Administrative Employee can see the menu
Postconditions:
The Administrative Employee can visualize all the available beds.

Trigger: Residential Administrative Employee indicates they would like to get the list of all available beds record through system interaction menu.
Basic Flow:
1. Residential Administrative Employee indicates they would like to see all available beds inside the system.
2. The SRMS reads the data from the Residence.
3. The SRMS displays the available beds in the following format: "Beds with number: i", where i represents the number of a specific bed inside the system in a list layout.
Extensions:
2a. System cannot connect to residence data.
2a1. System prompts the Administrative Employee to try again and redirect them into the main menu.
2a2. Administrative employee tries again. If there is no connection, then enter step 2a3 but if there is connection resume basic flow at step 2.

Use Case 4: Assign bed to Student
Actors: Residential Administrative Employees and Students
Stakeholders and Needs:
Residential Administrative Employees –To interact with the system to assign a bed to a student on a given residence.
Student-To be able to be registered into a residence inside the institution.
Preconditions:
The system is working correctly, and Residential Administrative Employee can see the menu
The entered ID exists inside the system.
There are available rooms vacant in the residence.
Postconditions:
The student is registered into the residence and the attribute with the bed number is changed to the actual selected bed number inside the residence
Trigger: Administrative Employee selects the option to assign bed for a student inside the residence.
Basic Flow:
1. Administrative Employee selects for the assign bed option inside the interactive menu
2. SRMS prompts the Employee to enter the Student ID and bed label.
3. Administrative Employee enters the requested data into the system.
4. SRMS validates the data
5. SRMS assign the student a bed label and change all the related attributes in order to keep consistency across modules
Extensions:
4a. Entered ID does not match any other inside the system
4a1. The system prompts the Administrative Employee that the ID of student cannot was not found.
4a2. Redirects Administrative Employee into the Main menu.
4b. Entered bed label is not valid for the given residence.
4b1. The system prompts the user that the bed label is not valid.
4b2. The system redirects the user into the main menu.

Use Case 5: Display State of the System
Actors: Administrative Employee
Stakeholders and Needs:
Administrative Employee-To visualize a snapshot of all the relevant data inside the system on a given instance.
Preconditions:
The system is working correctly, and Administrative Employee can visualize the menu

Postconditions:
A snapshot of the state of the system (relevant data) is shown to the Administrative Employee in a user-friendly display.
Trigger: Administrative Employee selects the option to see the status of the system inside the menu
Basic Flow:
1. Administrative Employee selects the system state option inside the interactive menu.
2. SRMS displays all the relevant data for the Administrative Employee to see.
Extensions:
2a. SRMS cannot get the data from the database.
2a1. SRMS calls for IT department for maintenance.

Use Case 6: Drop Student/Manager association from SMRS.
Actors: Administrative Employee, Student and Manager.
Stakeholders and Needs:
Administrative Employee-To control the lists of students associated with Managers
Student-To know which manager they are under supervision.
Manager-To know which students they supervise.
Preconditions:
The system is working correctly, and Administrative Employee can visualize the menu
The student and Manager are registered into the system.
Postconditions:
The list of Managers associated to a student will have the specific Manager deleted from it. As well the list of Students associate to a manager will have the specific Student removed from it.
Trigger: Administrative Employee selects the option of Dropping Association inside the menu.
Basic Flow:
1. The Administrative Employee selects the drop association option.
2. SRMS request for IDs of the student and Manager to be dropped association.
3. SRMS deletes the student and Manager from each other list.
4. SRMS confirms that the deletions were successful.
Extensions:
3a. SRMS cannot get remove student from the Manager's List.
3a1. SRMS calls for IT department for maintenance.
3b. SRMS cannot get remove manager from the Student's List.
3b1. SRMS calls for IT department for maintenance.

Use Case 7: Release Student from the SRMS.
Actors: Residential Administrative Employees and Student
Stakeholders and Needs:
Residential Administrative Employees – to interact with the system to delete old student records
Student Residents – To have their records removed permanently from the system database
Preconditions:
The system is working correctly, and Residential Administrative Employee can see the menu.
There are non-zero amounts of students occupying the residencies
The system is available and not down for maintenance
The student to be deleted was previously inside the residence the
Postconditions:
The information saved in the record is only deleted from the system if the administrator has sufficient access control and confirms reason for deletion
If any issues occur while deleting a record the IT Department is automatically contacted
Student is alerted of their record deletion from the system
Trigger: Residential Administrative Employee indicates they would like to delete an old student record through system interaction menu.

Basic Flow:
1. Residential Administrative Employee indicates they would like to delete existing student record
2. SRMS validates that student record exists and is currently at a residence of university
3. SRMS performs deletion (clearing the bed used by that student and also deleting their data from SRMS) and Residential Administrative Employee confirms student ID and other information of record inside the SRMS

Extensions:
2a. Student ID not found,
2a1. SRMS prompts the Administrative Employee to enter a valid Student I.D. Finally, enters the basic flow at step 3.

Use Case 8: Assign a Manager to a Student.

Actors: Residential Administrative Employees, Student and Manager.

Stakeholders and Needs:
Residential Administrative Employees – to interact with the system to view existing student records
Student Residents – To be assigned to a Residential Manager
Residential Managers – To have students assigned to

Preconditions:
The system is working correctly, and Residential Administrative Employee can see the menu.
There are non-zero amounts of students occupying the residencies
There are available managers with the capacity to be assigned extra students
The system is available and not down for maintenance

Postconditions:
The information saved in the record is only stored on the system if there are no system conflicts
If any issues occur while adding a record the IT Department is automatically contacted
The system contacts the Residential Manager to notify them of their new assigned student

Trigger: Residential Administrative Employee indicates inside the interactive menu that they would like to assign a manager to a student

Basic Flow:
1.Residential Administrative Employee indicates they would like to assign students to Residential Managers
2. SRMS prompts Residential Administrative Employee to select target Residential Manager and student(s) to be assigned based on their IDs
3. Residential Administrative Employee selects all involved people
4. Residential Administrative Employee confirms changes to Residential Manager assigned students
5. SRMS saves information and changes to the system database

Extensions:
2a. One of the IDs is empty
2a1. The system prompts the Administrative Employee that the ID of an employee or a student cannot be empty.
2a2. Redirects Administrative Employee into the Main menu.

Finally, in order to explain with more details, the behavior of the system, two sequences diagrams were created based on Use cases 1 and 8, being as follows:

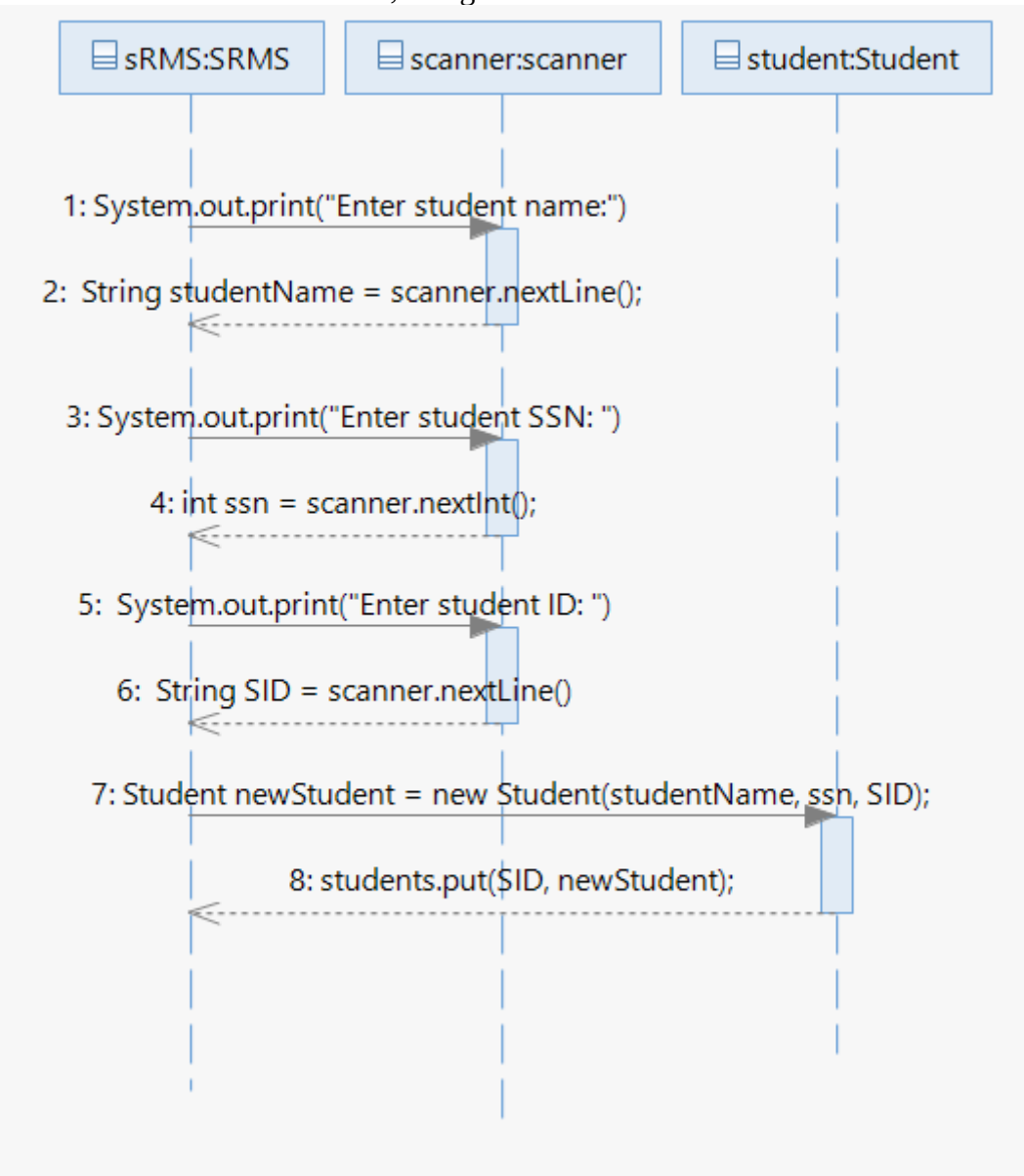


Figure #2: Sequence Diagram for Use Case 1: Add a Student to SRMS.

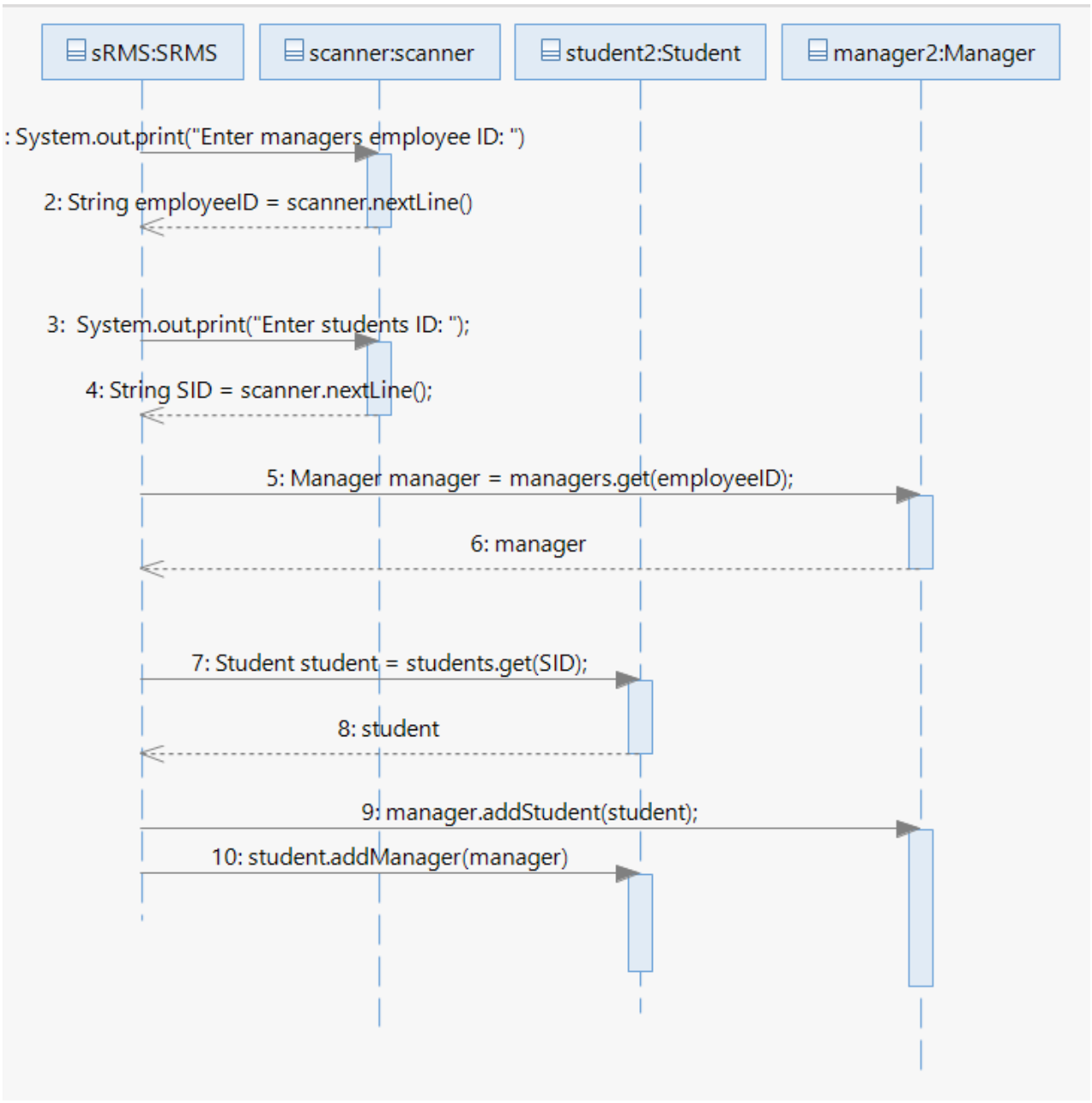


Figure #3: Sequence Diagram for Use Case 8: Assign a Manager to a Student.

2.2 FR-01: Management of Users

2.2.1 The system shall be able to receive user input.
2.2.2 The system should store information from users.
2.2.3 System should authenticate the user input.
2.2.4 System should display a warning message if user input is not authenticated.
2.2.5 Admin should able to modify the information of students and staff if required.

2.2.6 Admin should be able to add a student record.
2.2.7 Admin should be able to modify the records related to beds inside all residences.

2.3 FR-02: Payment Services and Billing

2.3.1 System should allow online payment through online wire transfer or Interact e-transfers.
2.3.2 System should notify user is payment was received.
2.3.3 System should allow communication with third parties such as Bank Services and Authentication Services

2.4 FR-03: Management of Student Residences

2.4.1 System should keep a record of all the available beds.
2.4.2 System should keep a record of all the booked beds.
2.4.3 System should keep the information related to a booked bed, details such as: Start date, end date.
2.4.4 Manager should have access to the booking information about their assigned students.

2.5 FR-04: Security and Privacy.

2.5.1 System should ensure that students personal data such as their name, contact information, and assigned manager is only accessible to authorized users.
2.5.2 System shall encrypt sensitive user information such as passwords.
2.5.3 The system should automatically log users out after a certain period of inactivity

3. Data Requirements

3.1 Logical Data Model

After some discussion with the client done by the BA, the SRMS's environment will have the following entities and associations:

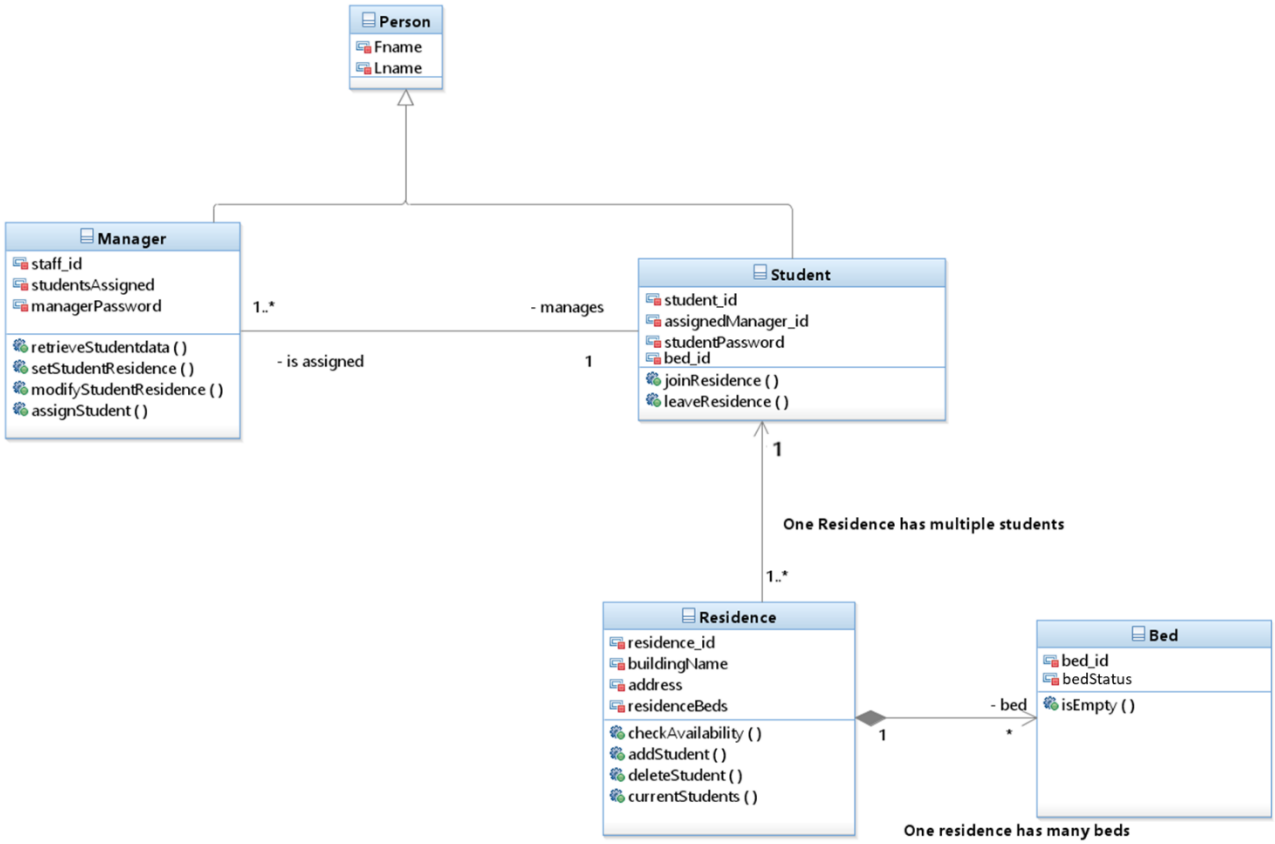


Figure #4: Class Diagram for Student Residence Management System

Inside Figure #4, the Student Residence Management System is divided into 5 classes, being as follows:

1. Person: Superclass to represent people entities. This class has the following attributes: Fname (First Name) and Lname (Last Name).
2. Manager: It is a child class from Person Class. It contains the previous mentioned attributes, with new ones related to the role given, being one of a manager. The specialized attributes are as follows: staff_id, studentAssigned and managerPassword. Finally, it contains the following methods: retrieveStudentData(), setStudentResidence(), modifyStudentResidence(), assignStudent(). The Manager class has a relationship with a student, being the one that one manager can be in charge of multiple students or just one.
3. Student: This class describes the behavior of the student entities inside the system and it is a child class from Person class. The specific attributes to this class are: student_id, assignedManager_id, bed_id ,studentPassword. In addition, it includes the following methods: joinResidence() and leaveResidence(). It contains a direct association with residence, where one student can stay in one residence. As well, contain an association with manager, being one student is assigned to one manager.
4. Residence: The class representing each residence/building inside the university. It contains the following attributes: residence_id, buildingName, address, residenceBeds. Also, it currently contains these methods: checkAvailability(), addStudent(), deleteStudent(), currentStudents(). Finally, this class has two association, the first one with the student class, where one residence has multiple

students in their rooms and the last association is related to the bed class, where one residence has multiple beds across its spaces.

5. Bed: It describes the bed's entity inside the Residence Management System, where it has only two attributes, the unique id distributed for each bed and the status of the bed. The only method isEmpty() checks the status of the bed, if available (free) or already being used by a student (used).

Nevertheless, the previous shown Data Model is not static. As the SRMS project has been undertaken under the iterative framework, the Logical Data Model might present some changes but keeping the essence of the first iterations, depending on the change of requirements or the constraints set by the client. To prevent any confusion, the developer team decided to implement class diagrams for each individual solution undertaken inside this project, in order to demonstrate key differences between the "base" Logical Data Model and the "polished" Logical Data Model after the Engineering Design Process.

3.2 Data Privacy and Security:

The SRMS will take sensitive data from the users, students and staff; the former will enter their data to apply for an opportunity to stay inside the residence and the latter will enter the system with their credential and then see the information from the students that they oversee. Therefore, the SRMS needs to protect the data that is inside its database, ensuring that the university has a strong cybersecurity in their housing administration.

Students should not have access to sensitive information from other students or staff, they should be able to visualize their own data but anything else is out of boundary. Students should be able to change their data whenever they want.

Managers should be able to see previously agreed data from their assigned students, such as the student's name, so that managers can carry their task at ease. "Outside Managers", a manager that is not assigned to a certain student, should not be able to see the information to unassigned students.

4. Non-Functional Requirements:

Some Non-functional Requirements:

NFR_01: Availability & reliability:

- The SRMS should be available with a maximum downtime of 15 minutes per year.
- The system should handle the peak of user's traffic.
- The system should be able to operate with at least 1000 users at a time.
- The system should be able to be open inside different devices, such as personal computers and mobiles phones.

NFR_02: Modifiability:

- The SRMS design should facilitate ease of integrating to the new source of reporting cases (e.g., enabling schools to report new cases).
- The admin should be able to perform maintenance with ease.
- The source code of the system should follow modularity principle and practices.

NFR_03: Scalability:

- The SRMS should support the reported cases' potential growth from a few hundred to several million.

NFR_04: Reusability:

- The SRMS design should enable at least 20% of the system's architecture to be reused in the other online monitoring platforms.
- The system should be well-documented.
- The system should come with a manual for the users.

5. Interface Requirements

This section will describe interfaces through which SRMS will interact with users, hardware and software.

5.1 User Interfaces

The system will have an intuitive and functional GUI for different user classes including residents/students, administrators, residence advisors and managers.
The features of UI will include:

- Web Interface: the system will be accessible through web applications such as Chrome, Safari, Edge, Opera, etc.
- Mobile and Computer Applications: the interface will have an adaptable UI in applications depending on the device allowing interacting with the system, getting notifications from both PC's and tablets/phones.
- Role-Based Access: The interface will provide different functionalities based on the user's role. Residents, administrators, managers, etc. Will have different levels of access.
- Accessibility: The interface will comply with WCAG 2.2 to ensure usability for individuals with disabilities. It will also be updated further every time new WCAG is published.

5.2 Hardware Interfaces

The SRMS will interact with various hardware components required for system operation, including:

- University Servers: The system will be hosted on university-managed servers to ensure data security and compliance with institutional policies.

- Biometric security: The system may support authentication through biometric scanning for student access verification.
- Printers and Scanners: The system will support document printing and scanning functionality for administrative use.

5.3 Software Interfaces

` The SRMS will integrate with various external software components and services to ensure smooth operation and interoperability. These integrations include:

- University Student Database: The system will connect to the university's existing student database to verify enrollment status before adding a new student record.
- HTTP/HTTPS: Secure web communication using TLS encryption.
- Email & Notification Services: The system will send automated email and SMS notifications for important updates such as successful registration, room assignment, and pending actions.
- Passkey technology: Passkey implementation will allow users to login securely using the API.
- Data Encryption Services: To ensure secure storage and transmission of sensitive data, the SRMS will use encryption libraries and protocols.

- Payment Processing System: The SRMS will integrate with third-party payment systems for processing deposit and fee payments.

6. Solution

6.1 Solution 1

The first solution was a "robust" design of the entire system, simple and easy to follow for the developers in order to understand the different relationships that were being taken into account inside the SRMS. It can be described as the first iteration/Logical Data Model of the design of the SRMS, where Managers oversee multiple students, but a student has only one manager. In addition, each student is assigned into a resident and inside each resident there are multiple beds. This solution can be easily shown in the following figure:

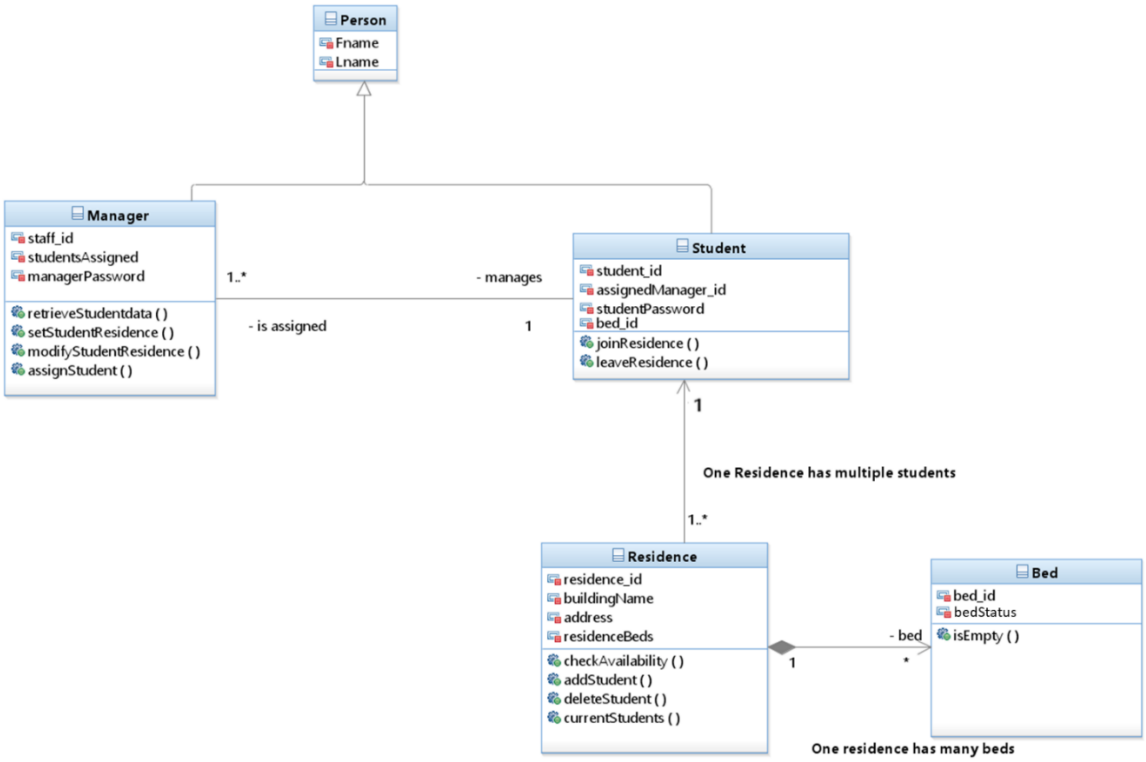


Figure #5: Class Diagram for Solution 1

Due to this solution being the first "iteration" of the design is missing key features such as a "controller" class to implement the actual features/functions from each class in an ordered mannered. Instead, inside this design, the developer would need to understand very clearly the way that each class works to obtain the major benefit from this architecture. This was the main reason why the developer team did not decide to implement this solution as the final one but instead of complete ignoring the entire solution, some key concepts/areas were reused in other solutions.

6.2 Solution 2

The second solution was a re-design of the first solution. It can be described as a second iterations based on the first solution, where some ideas were eliminated in order to bring more cohesion and strong relationships between the multiple classes that the program will use. The main difference with Solution 1 is that there is a controller class that is the one performing the actual operations depending on the user input, nevertheless, ideas such as inheritance, were kept as fundamentals of the design but improved and applied to implement major complexity throughout modules. The following class diagram was prepared to understand the key difference between Solution 1 and 2.

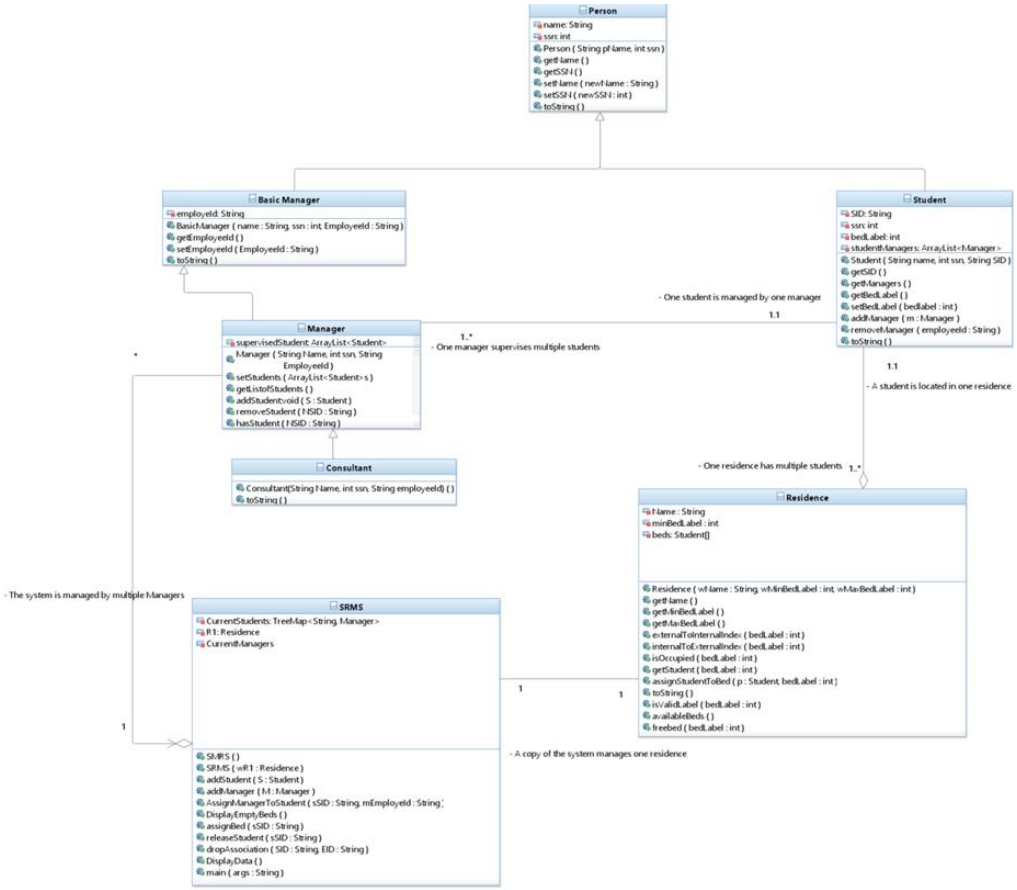


Figure #6: Class Diagram for Solution 2

Comparing the class diagram from solution 1 and 2, the beds table was eliminated and implemented with just an array inside the Residence Class. As well of the creation of new classes such as Person, SRMS (Student Residence Management System) in order to improve quality and control throughout the system. Nevertheless, when the actual code was presented based-on this class diagram, there was not a clear implementation for exception handling, meaning that the reliability of the system based on user-input would decrease if it was not exactly as under "specific/controlled" environment. As well, some relationship between classes were not as detailed or precise as expected from this system. These two main circumstances are what make the developer team to choose a more reliable solution.

6.3 Solution 3

The third solution aimed to serve as a proof-of-concept prototype that described all the main functionalities of the final deployed system for minimal budget and resources needed for development. It served as a foundation to build off while proving the feasibility of the system and how the different modules of the system may be organized. The basic class structure of all key elements was defined, along with some key functionality. The solution incorporated exception handling, with potential faults in the system defined and handled accordingly. In-code documentation was also created to enable ease of comprehension and easier modifiability and comprehension.

The solution was split into numerous class modules to represent the different actors in the system, along with a main execution class to run the main system. Seven classes in total were created, with six representing the actors in the system. The six actor classes were Residence, Person, Student, BasicManager, Manager, and Consultant. Student and BasicManager inherited from the Person class, while Consultant inherited from Manager, and Manager inherited from BasicManager. The full relationship of the system is described below in the class diagram of Figure #7.

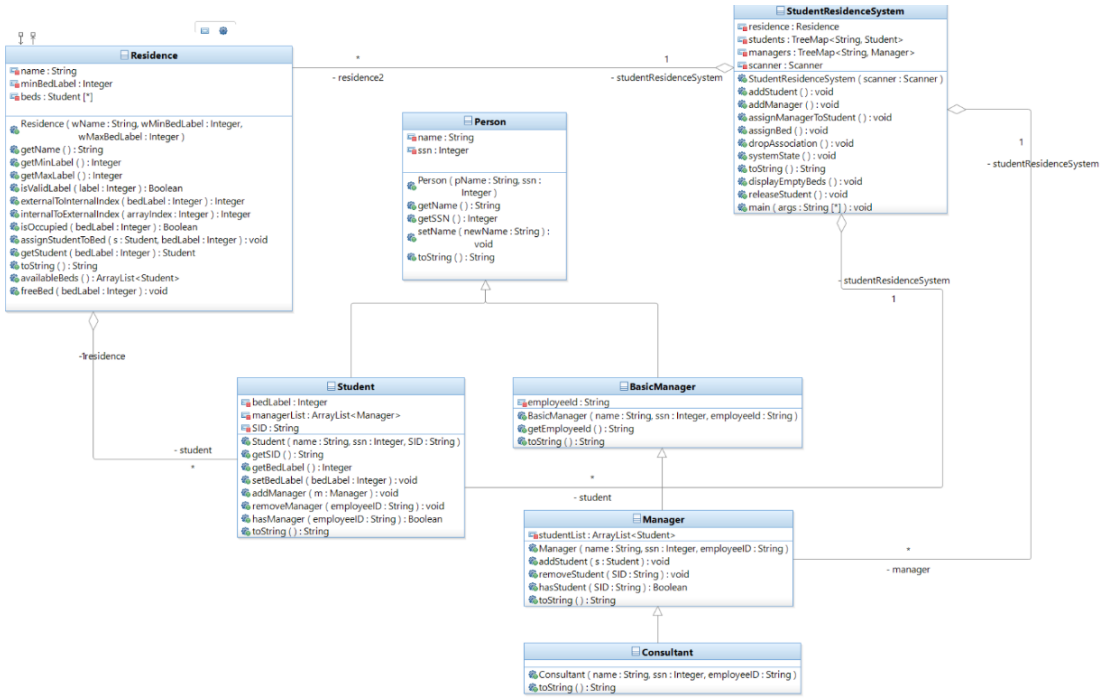


Figure #7: Class Diagram of Solution 3

It can be seen from the figure that each class had a variety of attributes and methods according to the type of class being described. Typically each actor inheriting from person had an identifier, with Students having a student ID (SID), and employees having an employee ID. It can also be seen that the main execution class StudentResidenceSystem incorporates the Residence, Student, and Manager/Consultant classes directly. The StudentResidenceSystem enabled a user interface which took user input and allowed for management over the residences and students stored inside the system.

Solution 3 described the initial class structures necessary for building the SRMS, along with the key functionality to prove feasibility. Users would be able to interact with the system to view the current state, including all current students and managers in the system, along with the managers associated with each student, and students associated with each manager. The information of each actor in the system was additionally able to be modified and removed.

6.4 Final Solution

Ultimately Solution 3 was chosen to serve as the Final Solution for a variety of reasons. While the other solutions proposed differing but still viable methods of implementing the SRMS, Solution 3 was the most complete. Other solutions included varying capabilities, but typically were missing one or more elements making them less complete prototypes. Solution 3 included a full UML class diagram describing all relationships, key functionality of managing and viewing resident

information. Exception handling and detailed documentation of the system was also implemented to enable easy modification of the system and enhanced reliability.

While Solution 3 seems to be the best solution in abstract discussion, a decision matrix chart was also used to evaluate all solutions numerically for objectivity. The decision matrix compares several key criterion amongst all three solutions to compute a total score on a scale of 0 to 1 to determine a ranking of the solutions. The decision matrix can be seen below in Table I.

Table #1: Decision matrix chart for the considered alternatives

		Solutions					
		Solution 1		**Solution 2**		**Solution 3 (Final Solution)**	
Criteria	**Weight**	**Score**	**Partial Score**	**Score**	**Partial Score**	**Score**	**Partial Score**
Availability	0.25	2/10	0.05	4/10	0.1	7/10	0.175
Reliability	0.25	4/10	0.1	3/10	0.0075	8/10	0.200
Modifiability	0.15	6/10	0.09	5/10	0.075	7.5/10	0.113
Scalability	0.20	5/10	0.1	5/10	0.1	8/10	0.16
Reusability	0.15	9/10	0.135	6/10	0.009	7.5/10	0.113
Sum	1.00		0.475		0.2915		0.761

The key criterion on which the solutions were evaluated were the availability, reliability, modifiability, scalability, and reusability of the system. Collectively, the success of the solution can be determined based on how well each of these areas are satisfied. The system must be as available and reliable as possible to facilitate maximum serviceability to all individuals who interact with the system. The system must be able to be easily modified to address required changes and upgrades as they are needed. The system must also be scalable to handle growth in university residences for the indefinite future. Finally, the system should also be reusable for future system to build off the foundation created by this design to reduce future work needed.

6.4.1 Data Dictionary

The final solution provided a set of classes, where each class has different attributes and they contain a set of rules in order to define proper identities inside the system. For each class, a data dictionary was created being as follows:

Table #2: Person Class

Field Name	Data Type	Size	Definition	Constraints	Example
Name	String	Variable Size	Full name of a person	It cannot be null	John Doe
SSN	integer	Maximum of a 9-digit number	Social Security Number	It cannot be null	986756023

It is important to mention that the student class inherits the methods and attributes of the Person Class.

Table #3: Student Class

Field Name	Data Type	Size	Definition	Constraints	Example

Name	String	Variable Size	Full name of a person	It cannot be null	John Doe
SSN	integer	Maximum of a 9-digit number	Social Security Number	It cannot be null	986756023
bedLabel	integer	From minimum number of beds to the maximum allowed inside the residence	The unique number for a bed inside a residence	It cannot be null. It is set to –1 when a student is not associated with a bed.	10
managerList	ArrayList<Manager>	Depends on the number of managers associated with a student.	The list of Managers associated with a student	It can be empty.	ManagerList = John Doe
SID	String	Depends on the policies	The unique ID of a Student	It cannot be null. It should be unique for every student More rules are associated with the policies of a given institution.	T007523472

It is important to mention that the BasicManager class inherits the methods and attributes of the Person Class.

Table #4: BasicManager Class

Field Name	Data Type	Size	Definition	Constraints	Example
SSN	integer	Maximum of a 9-digit number	Social Security Number	It cannot be null	986756023
employeeId	String	Depends on the policies of the institution	Unique identifier given to each Manager when they enter the system	It cannot be null. More rules such as the starting word might change based on institution policies	E000126345

It is important to mention that the Manager class inherits the methods and attributes of the BasicManager Class.

Table #5: Manager Class

Field Name	Data Type	Size	Definition	Constraints	Example
SSN	integer	Maximum of a 9-digit number	Social Security Number	It cannot be null	986756023
employeeId	String	Depends on the policies of the institution	Unique identifier given to each Manager when they enter the system	It cannot be null. More rules such as the starting word might change based on institution policies	E000126345
studentList	ArrayList<Student>	Depends on the number of students that a manager supervises on a given time	List of students that are supervised by a given Manager	It cannot be null. It can be empty.	StudentList = John Doe.
Name	String	Depends on the actual Name of the Manager	Full Name of the Manager	It cannot be null or empty.	John Doe

It is important to mention that the Consultant class inherits the methods and attributes of the Manager Class.

Table #6: Consultant Class

Field Name	Data Type	Size	Definition	Constraints	Example
SSN	integer	Maximum of a 9-digit number	Social Security Number	It cannot be null	986756023
employeeId	String	Depends on the policies of the institution	Unique identifier given to each Manager when they enter the system	It cannot be null. More rules such as the starting word might change based on institution policies	E000126345
studentList	ArrayList<Student>	Depends on the number of students that a	List of students that are supervised by a given Manager	It cannot be null. It can be empty.	StudentList = John Doe.

		manager supervises on a given time			
Name	String	Depends on the actual Name of the Manager	Full Name of the Manager	It cannot be null or empty.	John Doe

Table #7: Residence Class

Field Name	Data Type	Size	Definition	Constraints	Example
name	String	Depends on the name of the Residence/Building	The official name of the residence	It cannot be null	McGill Residence
minBedLabel	integer	From 0 to the limit capacity of the residence	The starting number of the available beds inside the residence	It cannot be null. It cannot be greater than MaxBedLabel	10
beds	Array of Students	The size is decribed by the MaxbedLabel – minBedLabel +1	An array to represent the beds of the residence	Fixed size. For each bed to be empty, means that they are null. When the element of the array is not null, then a student is associated with that bed.	Beds= [John Doe].

Table #8: Student Resident Management System Class

Field Name	Data Type	Size	Definition	Constraints	Example
residence	Residence		The Residence where the system has been installed	It cannot be null	Residence R1 = new Residence(wName, wMinBedLabel, wMaxBedLabel)
students	TreeMap<String, Student>	Depends on the number of	All the students registered into the	It cannot be null. It can be empty at	

		students inside the system	residence	beginning	
managers	TreeMap<String, Manager>	Depends on the number of managers inside the system	All the managers that oversee the given residence	It cannot be null. It can be empty at beginning	
scanner	Scanner		Used to get user input throughout the methods	Used just for input readings	Scanner scanner

6.4.2 Features

The final solution includes the following features:

- Quit:
  - The system allows the user to exit out of the system
- Student Addition:
  - The system allows the user to add a new student using addStudent()
- Manager Addition:
  - The system allows the user to add a new manager using addManager()
- Manager Assignment:
  - The system allows the user to assign a manager to a student using assignManagerToStudent()
- Empty Bed Display:
  - The system can display a list of all empty beds in the residence using displayEmptyBeds()
- Bed Assignment:
  - The system can assign a student to a bed using assignBed()
- Student Release
  - The system can remove a student from the residence using releaseStudent()
- Remove Assigned Manager
  - The system can remove a manager from a student using dropAssociation()
- Info Display
  - The system can display all current system info using systemState()

6.4.3 Environmental, Societal, Safety, and Economic Considerations

6.4.3.1 Environmental Considerations

The designed Student Residence Management System was created in such a way to take into account environmental factors and to minimize negative impact through the following ways:

Table #9: Environmental Considerations

Minimize the usage of paper	The designed solution migrates the storage of documents and information from a large paper based solution to a digital solution. This reduces the amount of natural resources directly needed for the system to operate. Previously each record would require an individual piece of paper, and every small change or modification would require a completely new piece of paper. On a large scale this can create a large negative impact through the cutting down of trees.
Employ good coding practices	The software was designed to make use of good coding practices, saving energy, required memory, computational power, and more. Inefficient software can lead to wastage of resources through unnecessary computation. Code was split into various classes depending on the actors relation to the overall system. Modules were designed to be weakly coupled and highly cohesive to facilitate a scalable system with efficient code.
Proprietary software	The software was created specifically for the SRMS, meaning it was able to be tailored to be as efficient as possible. External software usage designed for general tasks would mean less optimization and further unnecessary consumption of energy.
Efficient scalability	The designed system was created in such a way to be easily scaled to large amounts of data efficiently. If the system was inefficient when handling large amounts of data, large unnecessary amounts of resources may be consumed.

6.4.3.2 Societal Considerations

The Student Residence Management System aims to facilitate easier access to on-campus residence for prospective or returning students. Higher education is a cornerstone to a well performing and functional society, and the SRMS aims to improve accessibility to the university for students who live far away from campus. An efficient SRMS means a less tedious process for attaining student residence. The university residences are able to more effectively scale due to the sophisticated management system, leading to increased effective capacity.

The modernization of the existing system showcases the benefits of technology and software. The success of the system in terms of key aspects such as efficiency, reliability, and

availability would inspire other outdated systems to make the shift. The modernization of such systems would create a more functional society with increased automation, safety, efficiency, and less tediousness.

New or prospective students are often overwhelmed or stressed from the big life transition that comes with university. By having a streamlined and simple to use SRMS, some of the stress can be alleviated leading to a more positive experience for students. Having an easy and intuitive solution to housing allows them to redirect their focus to other avenues, such as their academics.

The SRMS additionally contributes to the university image as being cutting edge in terms of technology and modernization. The university may seem more appealing to prospective students or investors with modern and efficient systems in place. Additionally, having an efficient management system reduces the tedious labour required by resident management employees and staff. It will allow them to redirect their efforts into more important work directly interacting with the students and people who require assistance, rather than spending time wrestling with the system.

6.4.3.3 Safety Considerations

Safety is paramount when designing the SRMS due to the nature of affecting potentially thousands of students and their data. The system was designed to take into account the impact the system might have on the stored information and greater influence on safety of users. The digitalization of the system ensures that the information is saved in such a way to prevent data loss. Users are able to access the information stored in the system remotely, at anytime excluding exceptional cases.

The system employs role based access, meaning the only ones who have access to all information are the administrators of the system. This prevents students from accessing sensitive information, or modifying other student information. Each student also has a list of their managers to ensure a point of contact in case of inquiries or emergency. Similarly, each manager has a list of their students for in case of contact.

The program was designed to handle exceptional cases; in case of unexpected execution, an appropriate error message will occur. The use of exception handling infers easier debugging and troubleshooting in the rare case of unexpected system execution. This leads to lower amount of system downtime and higher quality code.

The software was developed using version control software (Git and Github) to facilitate backups of the code and functional software. Github allows for branching to add experimental features, and only merge to main upon verification of the additions. It also allows a history of past commits to facilitate documentation and how the program was built over time. This reduces chance of catastrophic system failure, and reduces troubleshooting efforts.

6.4.3.4 Economic Considerations

The transition to a digital solution aims to achieve improved performance, while being projected to save finances in the future. The nature of the system being developed iteratively through various deployment and prototypical stages aims to maximize the budget at each step. The designed solution will maximize the initial investment costs of development by maximizing productivity gains, reducing operation costs, and returning long term value.

The digital system will require an initial financial investment, but aims to return the cost in the span of less than three years. The development cost is capped at 1.8% of 2024 gross income for the initial deployment, and 3% for the first five years of maintenance and all deployments. This initial cost will not create significant financial damage to the university, but aims to modernize one of the core administrative systems of the facility.

Costs are expected to be saved from the migration to the digital solution. Previously each record was stored on an individual sheet of paper. Each time a modification occurred to data, a new copy would need to be printed, costing both electricity and ink from the printer. The cost of labour was also higher, since the time per transaction was significantly higher, along with the amount of tedious work. With the modern solution, higher capacities of students can be easily managed, meaning more residents and more economic growth.

The system was also designed to be both scalable and reusable, leading to reduced cost in future improvements. The efforts needed to maintain the system and scale it according to growth in student residents will be minimized due to the nature of the solution being designed at a dynamic scale. Additionally, the code was structured in such a way to be reusable, meaning future projects are projected to have a lower cost of development.

6.4.4 Limitations

The designed solution aims to showcase the essential features of the system as an early prototype model and act as a proof of feasibility but lacks features that may be found in the final design. The limitations of this solution are detailed below:

- Lack of data persistence (database)
  - The delivered prototype does not handle persistent data between sessions
- No remote access
  - The current version of the system runs off an executable and is not available through the web or other remote interfaces
- No login portal
  - The current system assumes the user interacting is an administrator and does not check for login credentials
- No tutorial for navigation
  - Upon entering the software for the first time there is no tutorial or guidance for the user to navigate or interact with the system
- Text based navigation rather than GUI
  - The current system is displayed through a terminal output window and
- Lack of encryption of data
  - Data stored in the system is not yet encrypted due to nature of not being stored. Ideally future prototypes would implement hashing system or filter data through third party software for security purposes
- Validation of input data
  - Current prototype only has elementary input validation for when adding or modifying current system records (i.e. throwing exceptions)
- Search for users or managers
  - Current system displays all records at once rather than having a filtering or search filter to narrow down specific records
- Payment services and billing
  - Payment and billing services are not yet implemented in current prototype

7. Team Work

7.1 Meeting 1

Time: 1hr
Agenda: Start of the project

Team Member	Previous Task	Completion State	Next Task
Brock	N/A	N/A	Sections 1-1.5
Jose	N/A	N/A	Sections 2 and 3
Alikhan	N/A	N/A	Section 4 and 5

7.2 Meeting 2

Time: 1hr
Agenda: Assignment 1

Team Member	Previous Task	Completion State	Next Task
Brock	Sections 1-1.5	80%	Section 6
Jose	Sections 2 and 3	90%	Section 6
Alikhan	Section 4 and 5	100%	Section 6
Ryan	Additions to Section 2 and 4	100%	Section 6

7.3 Meeting 3

Time: 1hr
Agenda: Assignment 2

Team Member	Previous Task	Completion State	Next Task
Brock	Section 6	100%	Section 1.5-1.8
Jose	Section 6	100%	Section 7
Alikhan	Section 6	100%	Section 8
Ryan	Section 6	100%	Sections 9, Reformat Document

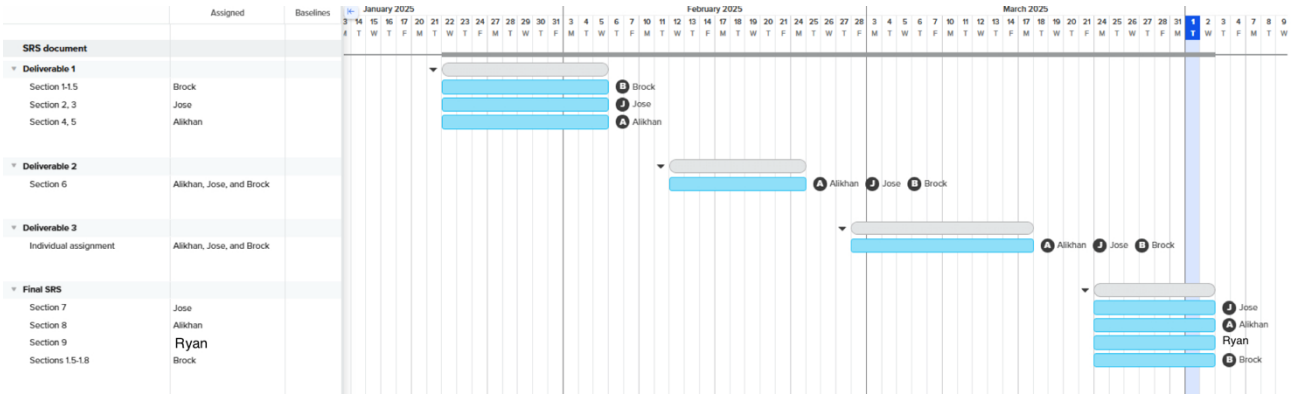
7.4 Meeting 4

Time: 1hr
Agenda: Final Stretch

Team Member	Previous Task	Completion State	Next Task
Brock	Section 1.5-1.8	100%	Presentation
Jose	Section 7	100%	Presentation
Alikhan	Section 8	100%	Presentation
Ryan	Sections 9, Reformat Document	100%	Presentation

8. Project Management

Figure #8: Gantt Chart



9. Conclusion

The development and documentation of this Student Residence Management System served as a critical step in advancing the universities technological infrastructure and management capabilities. The system successfully showcased the need for the software based SRMS to replace the older and incapable paper based model. The problem definition was first explored, before the project criteria was established, including the functions, objectives, and necessary constraints.

The project accomplished key criteria to various degrees, including functions such as the management of users, management of student residences, and security. In terms of objectives, the designed solution was chosen based on having the highest ratings against criteria such as availability, reliability, modifiability, scalability, and reusability. Additionally, the necessary constraints, as outlined in section 1.8 were satisfied for the developed prototype, including project delivery timing, indefinite scaling, and budgetary constraints. Future iterations are on track to satisfy additional expectations and requirements if following the expected trajectory.

The design document outlined all necessary functionality that the system would need to be considered successful but lacked a concrete model to represent the early design when actually implemented. Three different solutions were brainstormed from the elicited requirements, but ultimately one was chosen to best meet the needs of the system. Ultimately the final solution was chosen to be solution number 3 due to best meeting the requirements outlined in the document, and scoring the highest against a decision matrix comparing the how well the solutions achieved the main objectives.

Solution three showcased key functionality that will be present in the final administration system, such as the management of students, managers, and the residences through an interactive console interface. The prototype targeted the most paramount functionality that will prove the feasibility of the product, such as adding and deleting students and managers, viewing vacant beds, and assignment of students to managers or beds. The scope was limited to focus on a key set of criteria so as to not exceed budget and be easy to evaluate performance.

10. Future Work

While the final selected solution successfully showcased the current need for the Student Residence Management System and the feasibility of the design, future work is needed for a final system deployment. The current system was limited in design and scope to only accomplish key criteria and leave out extra functionality that would be expected to be present in a sophisticated and commercially viable management system. The areas of future work have been detailed below:

Table #10: Future Work

Feature	Context	Future Work
Data Persistence	The current prototype does not have data persistence between sessions, restarting back to the beginning every time it is launched. This makes it easy to demonstrate functionality and test the system comprehensively, but is not viable for final deployment.	The final deployed solution will include an external database to store user information via services such as Firebase or an SQL alternative
User Interface	The current prototype does not have an interactive user interface, rather prioritizing functionality over aesthetics. Currently all interaction is done through a terminal window	The final deployed solution will include an interactive graphical user interface designed for accessibility and user ease of use
Security Enhancements	The current prototype does not have any enhanced security features, instead assuming the user is already logged in as an administrator, and having no need for encryption of passwords or user data since no data persists between sessions	The final deployed solution will incorporate encrypted data storage, authenticated user sessions and a login portal, with different access levels to the system depending on the user logged in

Payment Services and Billing	The current prototype does not currently have any option for payment services and billing due to the prototypes focus on the administration aspects of the system	The final deployed solution will allow an option in the student interface to facilitate payment and billing services through third party payment processors
Advanced Search and Filtering	The current prototype does not currently have any options for searching or filtering the displayed data, instead just displaying all the available data	The final deployed solution will include a sophisticated search and filtering option to further categorize and organize the displayed data
Input Validation and Error Handling	The current prototype only has simple error handling and input validation, throwing exceptions if receiving unexpected results and terminating the program	The final deployed system will handle the unexpected inputs and errors with more elegance, prompting the user to rectify the mistakes and continuing execution
System Remote Access	The current prototype runs as an executable and must be downloaded and run locally with no current alternative to access the solution remotely	The final deployed system will allow access through a web based interface to allow system interaction from the serviceable area with a stable internet connection

11. Self Reflection

For the development of the Student Resident Management System, there were several options and paths that were considered in order to bring the main “idea” into the realm of the reality. It cannot be said that the selected option was the best from the point of view of outsiders, but at time, the development team approach the given project based-on the experiences and knowledge of the members. Being more experienced in other languages such as C++, this project proposed a real challenge to the development team where every member had to actually spend a certain time of the project just to check the actual syntax of language and compared with the knowledge known to understand and implement the requirements and deliver an actual working prototype.

Nevertheless, the development team agrees that the most interesting stage of the SRMS project was the actual development of UML diagrams and understand the actual meaning of each component from the diagrams and finally translate it into code, keeping the UML models as the base of the code. Inside this project, the start point for coding was related to the gathering of requirements and then translate them into something more “visual” so that the system can be developed. Following this behavior and discipline, the action of coding become a more structured process instead of actual “heads-on” into the coding realm. In the other hand, the most challenging

stage throughout the project was the actual organization of the team to actual “develop” the Project. All the members have really tight schedules, therefore, meetings were done and carried out when the schedules permitted.

For future Projects, the developer team agreed that more of a formal organization should be taken into consideration in order to provide a more suitable environment for the creation and development of products with a high-level of quality.