Declaration of Original Work for SC2002/CE2002/CZ2002 Assignment

We hereby declare that the attached group assignment has been researched, undertaken, completed, and submitted as a collective effort by the group members listed below.

We have honoured the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

Name	Course (SC2002/CE2002 CZ2002)	Lab Group	Signature /Date
Pang Lu Hiang Victor	SC2002	SACD	 20/11/2025
Lim Kwan Yit, Calvin	SC2002	SACD	 20/11/2025
Low Hong Wei	SC2002	SACD	 20/11/2025
Teo Kok Wei, Oscar	SC2002	SACD	 20/11/2025
Kannaiyan Ishwaryaa	SC2002	SACD	 20/11/2025

Important notes:

1. Name must EXACTLY MATCH the one printed on your Matriculation Card.

2. Student Code of Academic Conduct includes the latest guidelines on usage of Generative AI and any other guidelines as released by NTU.

Link to GitHub Project:

https://github.com/OTEO001/SC2002-Project-Internship-Management-System-

1. Introduction

As part of SC2002, our group designed an Internship Placement Management System as required to show the application of OO design principles. The main objective of this system is to streamline the internship placement workflow for Students, Company representatives and Career Centre Staff. Through CMI, users are able to log in and manage user-specific tasks. In designing this system, we prioritised achieving clear separation of responsibilities which emphasised loose coupling between components and maximised extensibility. To that end, the system takes into consideration core OO principles such as encapsulation, abstraction, inheritance and polymorphism alongside relevant design patterns and a modular class structure. Services classes handle business logic, controller classes act as the intermediaries between the user interface and the operations, repositories manage retrieving and storage of data and lastly entity classes contain structured data.

2. Design Considerations

2.1 Object Oriented Principles (OOP)

Encapsulation

Our group achieved encapsulation by structuring the system according to the Boundary-Controller-Entity (BCE) structure. Each component has a specific responsibility. The boundary classes handle user interaction through Command-Line Interface (CLI) menus. The controller classifies the coordinate user requests and calls the appropriate methods. Lastly, the entity classes represent the core data models and store domain-specific attributes. This separation ensures that internal data is only accessed through well-defined methods, which reduces coupling and prevents unintended modification across layers.

Inheritance

Inheritance is applied within the User hierarchy. The base User abstract class defines attributes that are shared between the other user classes, such as user ID, name and password. Subclasses such as Student, CompanyRepresentative and CareerCenterStaff extend this abstract class by adding role-specific attributes and methods. This avoids duplication while promoting consistency across all the user types.

Polymorphism

Polymorphism is enabled through the use of controller and service interfaces. Components such as IStudentService, ICompanyRepService, and IStaffService allow multiple implementations. Controllers depend only on the interface, which makes the system flexible and extensible. For example, when logging in, controllers can handle the User object without needing to know whether it is specifically a Student, CompanyRepresentative or CareerCentreStaff. The correct behaviour is automatically determined based on the user's actual subclass.

Abstraction

We implemented abstraction through interfaces and services. For instance, repositories like IUserRepository abstract user data storage, allowing different underlying persistence mechanisms (e.g. CSV, in-memory, or future database integration) without affecting the rest of the system. This level of abstraction makes the system adaptable and easier to extend.

2.2 SOLID design principles

2.2.1 Single Responsibility Principle (SRP)

We ensured that each class is designed with a single responsibility. Boundary classes were designed such that they manage user inputs and displays. They do not contain any logic relating to authentication, validation or processing of workflow. Controller classes mainly coordinate the actions and requests made by the user. They read the user's chosen action and call the appropriate service methods then return the results back to the boundary classes. The service interfaces and the implementations of them contain business logic. This includes validating the user's actions, handling the approval workflows and applying the rules of the system. Repositories manage retrieval of data and data persistence. Lastly, entities represent the data models and do not contain any business logic or system operations.

2.2.2 Open/Closed Principle (OCP)

We took into consideration the Open/Closed Principle by ensuring that the components of the system are open for extension but closed for modification. This is mainly achieved through the interfaces such as IAuthenticationService, IUserRepository, etc, allowing for new implementations to be added without changing existing code. For example, adding a new storage format would only require a new repository implementation without impacting the rest of the system

2.2.3 Liskov Substitution Principle (LSP)

The Liskov Substitution Principle was applied in our design too. All the subclasses of User can be used wherever the base type is expected without altering system behaviour. For example, the authentication flow returns a generic User object, which is safely passed into the appropriate menu regardless of whether the user is a Student, Company Representative or Staff member. This ensures consistent and predictable behaviour through polymorphic substitution.

2.2.4 Interface Segregation Principle (ISP)

The interfaces are made such that the classes that implement these interfaces use all the methods defined in the interfaces. In our system, IStudentService contains only business logic that is only relevant to the students. Similarly, ICompanyRepService handles all the operations that are relevant to company representatives and IStaffService grants approval to the company representatives upon registration. This keeps all the interfaces simple and prevents bloated interfaces with too many purposes.

2.2.5 Dependency Inversion Principle (DIP)

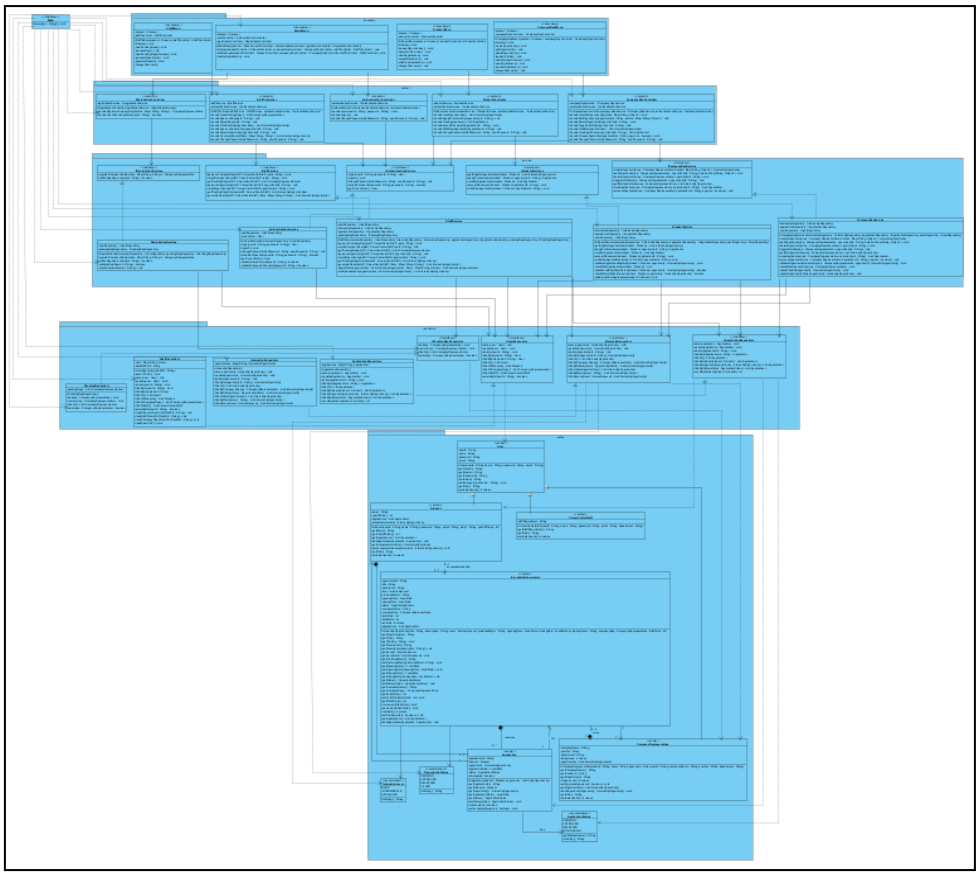
According to the course material, dependency Inversion Principle ensures that high-level modules depend on abstractions rather than concrete implementations. We stuck to this principle by letting high-level modules such as our controller classes depend on abstractions such as service interfaces. This allows any of the services to be replaced or extended without affecting the logic of the controller. Similarly, services interact with repositories through interfaces rather than direct implementations which ensures loose coupling and enhances testability.

2.3 Design Trade-offs

When designing this system, we had to make a few trade-offs to find the right balance between complexity and maintainability. Firstly, the implementation of multiple interfaces across the service and the repository layers added more files and made the design more complicated to implement. However, this provided significant benefits as the system was easier to work with as changes could be introduced without worrying about breaking unrelated parts of the code. This trade-off was considered to be worthwhile as this interface based design allowed the system to be scaled further with additional modifications. Another decision our group made was to introduce a service layer between the controllers and repositories. This layer consolidates all the business logic in one place and ensures that the system follows the same workflow rules everywhere and prevents the controllers from directly accessing or modifying the data. While this extra layer increased the total number of classes and required more coordination between the components, it provided clearer separation of concerns and made the system more maintainable and scalable.

3. UML Class Diagram & Sequence Diagrams

3.1 Class Diagram

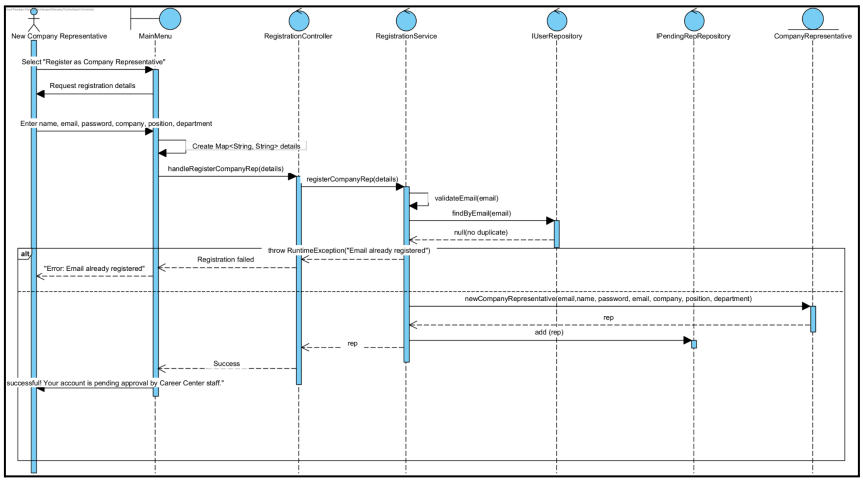


This is the Class Diagram for our Internship Placement Management System, this is a screenshot preview. The visual paradigm file for drawing the Class Diagram for our system implementation is attached along with this report.

3.1.1 Explanation

Our project is split into 5 packages: Boundary, Controller, Entity, Repository and Service. By organising code into distinct, purpose-driven packages, each contributing to a clear, modular and maintainable design. This aligns with the OOP practice of **encapsulation** through clear boundaries and modularity, **abstraction** by separating interface definitions and utility functions, **inheritance** by sharing common attributes and methods across related classes. and **polymorphism** by allowing interfaces to define interchangeable implementations.

3.2 Sequence Diagrams

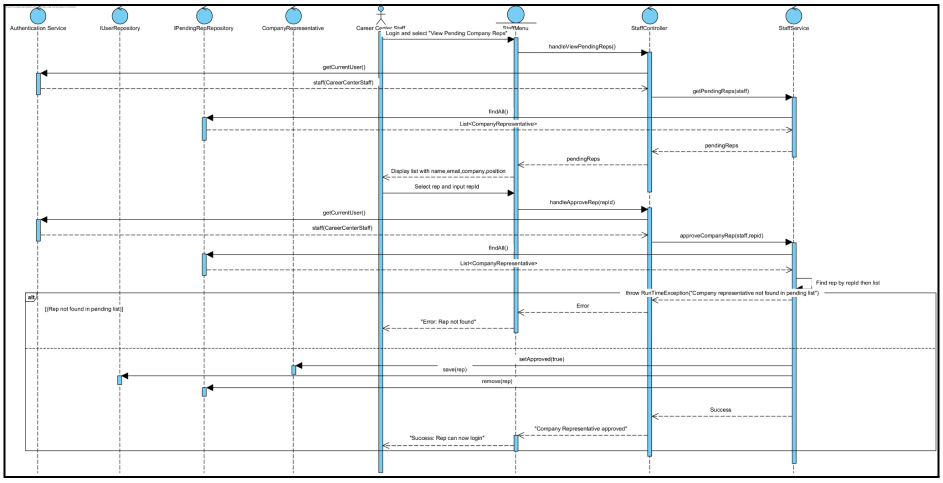


3.2.1 Company Representative Registering for a New Account

Prior to accessing the portal, New Company Representatives must register their accounts and can only log in to their accounts after approval from Career Center Staff. The series of actions are illustrated in the sequence diagrams and can be broken down into 2 phases.

Phase 1: Company Reprsentative Registering for a New Account

New Company Representatives begin the registration for a new account by selecting “Register as Company Representative” in the Main Menu user interface. They would be prompted to enter their registration details. Thereafter, the Registration Service controller would run through checks to ensure that the email format is valid, and that it doesn’t already exist in the User Repository. In the event the email is already registered, a RuntimeException error would be returned to the Registration Controller, and the user would be notified that the email is already in use. Otherwise, a new instance of Company Representative would be initiated, and the details will be mapped into the Pending Rep Repository for the Career Centre Staff to approve the account. Users would be notified that their registration is successful and their account is awaiting approval from the Career Center Staff.

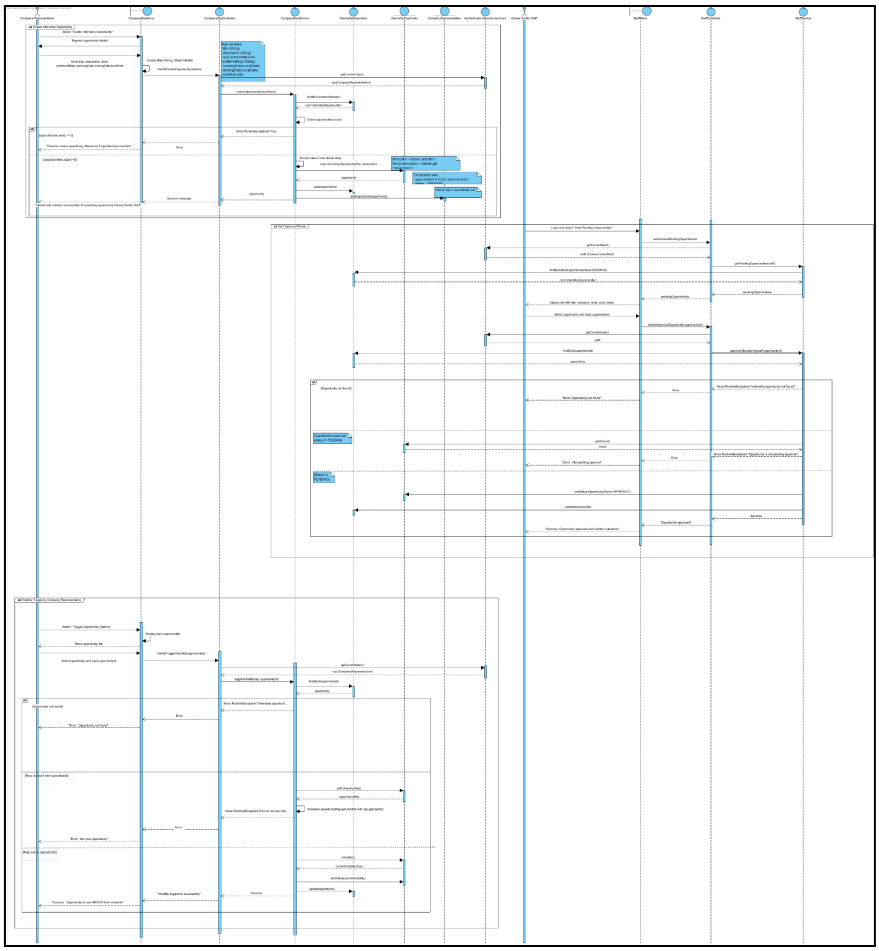


C

3.2.2 Career Center Staff approves New Company Representative's account

Phase 2: Staff Approval Phase for New Company Representative's Account

Career Center Staff proceeds to login to their account and is able to view the list of pending company representatives after their account is authenticated. They are then able to approve the new Company Representatives according to their respective User ID. If the wrong User ID is being inputted, a RunTimeException error will be thrown to the Career Center Staff. Otherwise, the new Company Representative's account will be approved by the Staff Service controller, and the new Company Representative's account details will be saved in the User Repository. At the same time, the new Company Representative's account details will be removed from the Pending Rep Repository - this repository serves as a temporary placeholder for the unapproved pending Company Representative accounts; After approval from the Career Center Staff, they are transferred to the User Repository. Lastly, the Career Center Staff receives a success notification as displayed by the Staff Menu. The new Company Representatives may then proceed to log in with their account.



3.2.3 Creating Internship Opportunity & Staff Approval with Visibility Toggle

The sequence diagram illustrates the process by which a Company Representative creates internship opportunities, including the subsequent Staff Approval phase. It further details the repositories and entity classes involved in these interactions. The diagram ends by showing how the Company Representative utilises the Visibility Toggle to control whether students can view internship programs.

Phase 1: The Company representative creates the internship opportunity

The Company Representative creates the internship by interacting with the Company Rep Menu Interface. They input the required details for a new internship opportunity. The program then retrieves the representative's profile to check the total number of opportunities they have previously created. If this count exceeds five, the code will return an error and notify the user that they have hit the max limit. If the count is five or fewer, the program proceeds to save the newly created internship details in the Internship Repository. The flow concludes with a confirmation message sent back to the Company Representative, acknowledging the successful creation.

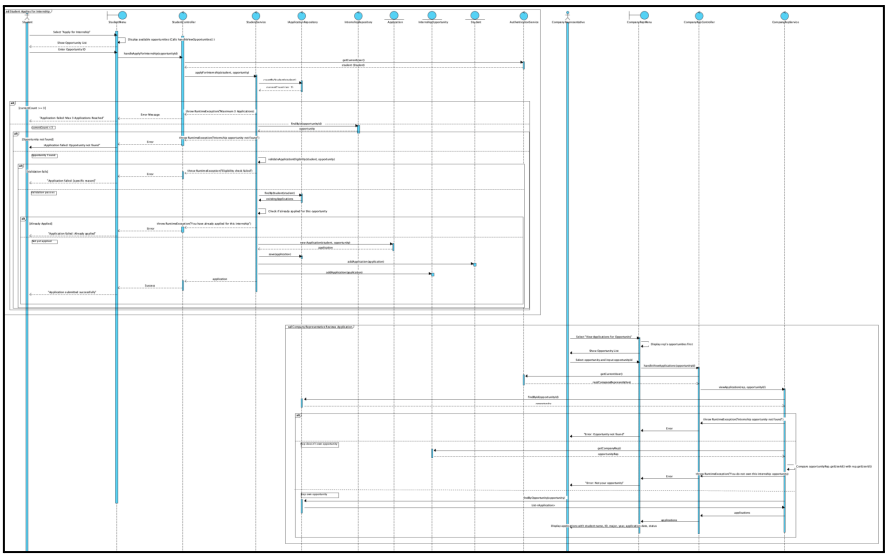
Phase 2: Staff Approval Phase for Internship

The Career Centre staff can view and retrieve submitted internship opportunities created by the Company Representative. By accessing the Internship Repository, they can find and review the specific details of a

selected opportunity from a list. The subsequent flow proceeds through an alt fragment to handle different scenarios: an exception error is raised if no matching opportunity is found; if an opportunity is found but its status is not 'pending approval', the staff is simply notified that the opportunity is not pending for approval. Finally, if the opportunity's status is confirmed as 'pending approval', the staff then proceeds to approve the opportunity.

Phase 3: Visibility Toggle by Company Representative

This process illustrates how the Company Representative utilizes the Visibility Toggle to manage student access to view the internship program. After interacting with the Company Rep Menu Interface, the system retrieves the current visibility status from the Internship Repository. The process then enters an alt fragment. If the opportunity is not found, the system notifies the Company Representative of the error. Next, it checks whether the representative owns the specific opportunity. Only once ownership is confirmed is the representative authorised to update the visibility.



3.2.4 Student Applies for Internship & Company Representative View Application

This sequence diagram illustrates how students submit their applications and how company representatives will review and process these applications. It showcases the interaction between our 5 packages as well as the participating objects within these 5 packages.

Phase 1: Student Applies for Internship

The student selects “Apply for Internship” and enters an opportunity ID. The StudentController will then retrieve the current user via the AuthenticationService and delegate it to StudentService. The service will then perform the following validation checks:

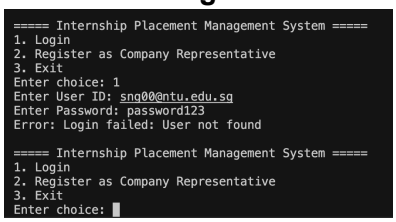
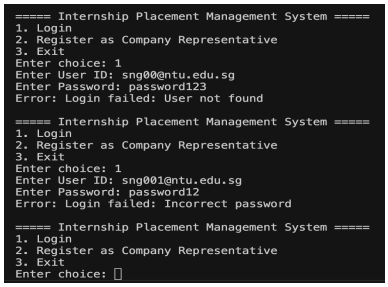
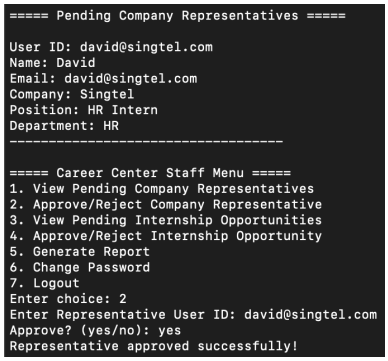
1. **Maximum Applications:** Verify if the student has 3 or lesser applications
2. **Valid Opportunity**: Retrieve the opportunity from InternshipRepository and check if it exists, or if its approved and whether its visible
3. **Eligible Year**: Validate the student’s year and check if it matches the opportunity level (Eg: Year 1-2 → BASIC only, Year 3 - 4 → All Levels)
4. **Available Slots**: Ensures that there are available slots (> 0)

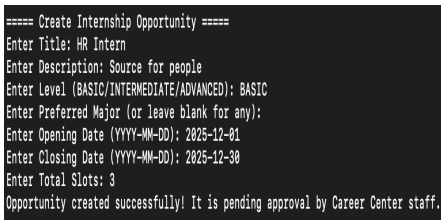
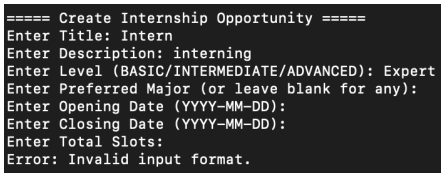
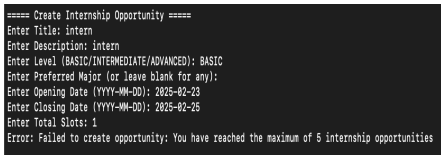
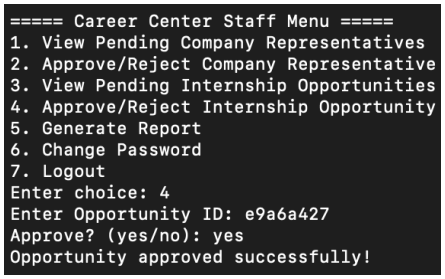
5. **No Duplicate Applications:** Checks if the student already applied for this internship

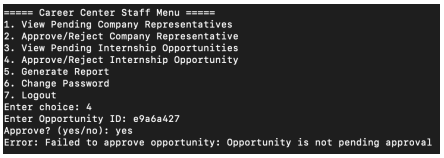
Phase 2: Company Representative Reviews Applications
The company representative will select "View Applications for My Opportunities." The CompanyRepController will then retrieve the current representative and find all opportunities owned by this representative and then retrieve the corresponding applications for each opportunity. The representative will be able to view the list and select whether to approve or reject the application.

4. Test Case & Results

For our test cases, we chose to include a few key test results to showcase the functionality of our system.

Test Case	Expected Outcome	Result
1. Valid User Login (Staff)	Successful login and System displays the Staff Menu **[Exception Handling]** If user enters an incorrect userId, an error message "User not found is displayed If the user enters an incorrect password, an error message "Incorrect Password" will be displayed instead Afterwards, it will then show the user the original menu screen	**Successful Login:**  **Invalid Password or UserId:** 
2. Staff Approve Company Representative	Staff successfully logs in and views the pending company representatives list which they can then choose to approve or reject. Upon successful approval, it will send a message "Representative approved successfully". Rep status changes from PENDING to APPROVED and they can login and create opportunities **[Exception Handling]** If the company representative is not found in the pending list, or	**Success**  **Invalid Company Rep UserID**

	the staff types in an incorrect UserID which is not in the pending list, it will display an error message	
3. Company Representative Creates Internship Opportunity	Company Representative successfully logs in and the system will prompt them for the required details. Upon successful creation, the system will send a message saying "Opportunity created successfully!. It is pending approval by Career Center Staff." **[Exception Handling]** If the input is incorrect in either the date format, number format or level, an Error Message "Error: Invalid Input Format" will be displayed If the Company Representative tries to create a 6th opportunity when they already have 5, an Error Message will be shown	**Success**  **Invalid Input Format**  **More than 5 Opportunities** 
4. Staff Approves Internship Opportunity	Staff logs in and selects the option to approve or reject the internship opportunity. Once approved, the Opportunity status will change from PENDING to APPROVEd and become visible to eligible students where they can now browse and apply. **[Exception Handling]** If the staff enters an opportunity ID that is incorrect, an Error message will be shown. If the staff tried to approve an opportunity that has either been APPROVED or REJECTED, an Error message will be shown.	**Success**  **Internship Opportunity not Found**  **Opportunity not Pending Approval**

Additional Test Cases: https://github.com/OTEO001/SC2002-Project-Internship-Management-System-

5. Additional Features

5.1 Password Strength Validation

To enhance security and protect user accounts against unauthorised access, we attempted to incorporate password strength validation into the system design. The “validatePasswordStrength()” method implemented in the "AuthenticationService” class in the “service” package enforces a minimum password length of 6 characters. This validation prevents users from setting weak passwords with small password lengths such as “123” or “abc”. It is applied whenever a user initiates a password change, ensuring that credentials adhere to our basic security requirements.

5.2 Comprehensive Input Validation

By implementing a comprehensive input validation, it ensures that all data processed by the system is accurate, consistent and aligned with the predefined criterias. This reduces errors arising from wrong inputs and safeguards the system against improper or invalid data entries.

How it was implemented:

The abstract “User” class in the “entity” package defines a base “validateUserId()” method which is overridden by the subclasses “Student”, “CareerCenterStaff” and “CompanyRepresentative” to make it more specialised. For CareerCenterStaff, the overridden method checks that the staff email is non-null and contains the domain @ntu.edu.sg, ensuring that only legitimate NTU staff accounts can be registered. For CompanyRepresentative, the method verifies that the email address is non-null and contains the company’s domain (e.g. @companyname.com. This ensures proper identification of company users. Lastly, student accounts must provide a valid NTU matriculation number. This validation ensures the ID is non-null and matches the required format U\d{7}[A-Z] (e.g. U2423987G) reinforcing data integrity across student profiles.

5.3 Duplicate Prevention System

Duplicate Application Check: In the “StudentService” class in the “service” package, in the “applyForInternship()” method, students are not allowed to apply multiple times to the same opportunity.
If the same match is found, the method throws a RunTimeException to prevent the student from applying again.

Duplicate Email Check: In the “RegistrationService” class in the “service” package, in the “checkDuplicateEmail()” method, company representatives are prevented from making multiple registrations under the same email address. If the same match is found, the method throws a RunTimeException error, preventing the creation of a duplicate account.

6. Reflection

6.1 Difficulties encountered

During the course of this project, we faced some difficulties. One difficulty faced was determining the interactions between the controller and service classes. Initially, we did not clearly distinguish what belonged in the controller classes and services classes which resulted in repeated logic and difficulty maintaining the code. To resolve this, we revisited the BCE model and reorganised the system so that boundary classes focused solely on user interaction, controllers handled request coordination and services contained all business rules. Another difficulty we faced was to maintain consistency across the different workflows especially when updates in one part of the system affected the other parts. For example, when a student accepted an internship, all their other applications needed to be withdrawn automatically and the respective internship slots had to be updated. Implementing these overlapping updates in a clean and non-coupled way took several tries.

6.2 Knowledge gained from this course

Through this project, designing this Internship Placement System helped us gain a lot of hands-on experience on how an application is designed. Working with the BCE model taught us how starting the project off by designing ensures clarity and maintainability when implemented correctly. Using a BCE model taught us that our project can be constructed in a more clearer way if used correctly. We also gained a stronger understanding of OO concepts such as encapsulation, abstraction, inheritance and polymorphism. This served to be useful especially in designing reusable services and controller classes that operate on the same entities. The SOLID principles taught in this course further cultivated good design habits. Another learning point was the importance of planning early. Initially, we rushed into coding without a complete view of the system. It was only later that we realised that implementing the app without a proper class diagram caused unnecessary readjustments. We learnt that creating UML diagrams, identifying the different relationships and structuring the system was essential for avoiding structural issues and ensuring consistency across user workflows.

6.3 Possible further improvements

Although our system functions well, there are some areas we would improve with more time. First, on the user experience side, developing a simple GUI to replace the command-line interface would make the system much easier to navigate through and use. We would also like to implement automated testing to help catch errors earlier on and ensure the system remains stable if more features are added. Overall, these improvements would make the system polished and bring it closer to what a real internship management platform might offer.