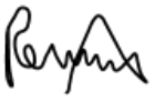
Declaration of Original Work for SC2002 Assignment

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 honored 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	Lab Group	Signature / Date
Xu Zixuan	SC2002	SCEA	
Isaac Wong Cheng Wei	SC2002	SCEA	
Lim En Sheng Royden	SC2002	SCEA	
Juan Sebastian Alexander	SC2002	SCEA	
Goh Sheow Ming	SC2002	SCEA	

Github: https://github.com/zacxu/SC2002Group6

1. Design Considerations

1.1. Overview of Our Approach

The Internship Placement Management System (IPMS) is designed using a modular, object-oriented approach while adhering to SOLID design principles. This ensures the application is extensible, maintainable, and easy to test. Each component is built to be independent yet cohesive, allowing seamless integration and future enhancements.

1.2. Assumptions Made

Firstly, we assumed that the data files containing the list of users and their information would be in the form of CSV files. As such, the IPMS will read the CSV files to extract the required data when it needs to initialise the registries during runtime. Secondly, we believe that any changes to the list of users and their information also need to be updated in the relevant CSV files during program execution. This would allow the changes to be saved even when the IPMS has stopped running, as the data is being stored in external CSV files.

2. System Architecture and Development

Our system is built using the Boundary-Control-Entity (BCE) architecture.

- Boundary: These are the classes that interact with the user, like StudentMenu and CompanyRepMenu, responsible for handling user input and displaying information
- Control: These classes, such as ApplicationController and InternshipController, contain the system's business logic. They process requests from Boundary classes and update the Entity classes.
- Entity: These are the core data objects of the system, such as Student, Application, and InternshipOpportunity

We will implement and test in order of dependency, namely: Entity, then Control, then Boundary. Testing each layer incrementally simplifies the debugging process and, as such, ensures the core logic is stable before building the user interface. Once the system is tested to be functional, then we will focus on further code refinement, such as improving readability and implementing robust exception handling.

3. Object-Oriented Programming Concepts

3.1. Abstraction

Our group used abstraction to reduce the complexity of the IPMS system by obscuring unessential information. To achieve this, we ensured that each subsystem would provide simple, intention-revealing methods while hiding its implementation details. For example, methods such as SessionService.getCurrentUser() (Interface) expose high-level behaviour without revealing how data is stored or processed, since the interface's implementation is hidden from the class that is using the getCurrentUser() method.

3.2. Encapsulation

Our group also practised encapsulation by restricting public access to the private attributes of entity objects. Should a class need to access an object's private attributes, it can only be done through the object's getter and setter methods. For instance, the InternshipRegistry's purpose is to store a list of internships, and access and modifications to its content are only possible through its public methods, such as getAllInternships(). This helps to prevent accidental access or modification of the registry's contents.

3.3. Inheritance

The IPMS system has to be able to handle different types of users. These users include Student, CareerCenterStaff and CompanyRepresentative. Moreover, these users require common attributes such as userID, name and password, as well as their accompanying getter and setter methods. As such, our group created an abstract User class which contains the common attributes and methods of the various users. By allowing the various types of users to inherit the common attributes and methods from the User superclass, we have reduced the complexity of the code in the concrete classes and enabled the system to be easily extended to more types of users.

3.4. Polymorphism

Given that the Student, CareerCenterStaff and CompanyRepresentative objects inherit from the User superclass, the User object can exhibit different types of behaviour depending on its actual subclass object that is instantiated at runtime.

3.5. SOLID Design Principles

In line with object-oriented (OO) programming practices, we designed our Internship Management System to be modular, extensible and maintainable. We applied a set of design principles to guide the structure and interaction of our classes, ensuring a clear separation of concerns and ease of future enhancements.

3.5.1. Single Responsibility Principle (SRP)

The system is split into several logical groups, with each class designed to tackle one specific task.

a. Entity Classes (Models): These classes are responsible only for holding data and defining core attributes. Examples include User, Student, InternshipOpportunity, Application and Withdrawal Request. They do not contain complex business logic.
b. Boundary Classes (Views): Classes that display information to the user and handle interactions for a specific role. For example, StudentMenu is responsible for the student's UI, CompanyRepMenu for the company representative's UI, and StaffMenu for the staff's UI.
c. Validator Classes: These classes encapsulate specific, complex business rules. For example, ApplicationValidator is solely responsible for checking if a student can apply for an internship (e.g. hasReachedApplicationLimit, validateYearRestriction). InternshipValidator handles rules for creating internships (e.g. isWithinSlotLimit, isDateRangeValid). This separates validation logic from controllers and models.
d. Controller Implementations: Classes that control the flow of the application, responding to user input and coordinating with registries and validators. Each controller handles a distinct feature set, such as AuthController for login/logout, ApplicationController for managing applications, and InternshipApprovalController for staff-side approvals.
e. Registries: These classes manage the persistence and retrieval of entity objects, acting as a collection or database layer.
f. Enums: Classes that define a set of constant values, such as InternshipLevel, InternshipStatus, and ApplicationStatus.

3.5.2. Open-Closed Principle (OCP)

To ensure extensibility, OCP is followed by utilising interfaces and abstract classes to promote loose coupling, allowing the system to be extended without modifying existing code.

a. Abstract Classes: The User class is an <<abstract>> class. It is extended by Student, CompanyRepresentative, and CareerCenterStaff. The system can be extended to include new types of users (e.g. an Admin user) simply by creating a new class that extends User. Existing code, like the SessionController, which operates on User objects, will not need to be changed.
b. Interfaces: Controller classes depend on service interfaces, not concrete implementations. For example, AuthController depends on the SessionService (interface).

3.5.3. Liskov Substitution Principle (LSP)

This design ensures that derived classes are fully substitutable for their base classes. This is primarily seen in the User inheritance hierarchy. Any part of the system that works with a base User object (e.g. SessionService.setCurrentUser(User) or AuthService.login(String, String)) can transparently operate on instances of Student, CompanyRepresentative, or CareerCenterStaff. These subclasses honour the contract of the User class, and can be used in its place without causing errors.

3.5.4. Interface Segregation Principle (ISP)

The system's interfaces are specific and segregated by client role, ensuring that classes do not depend on methods they do not use.

Instead of one large "ManageSystem" interface, functionality is split into many small, role-specific interfaces. For example:

a. A Student client, via the StudentMenu, interacts with StudentInternshipQueryService and StudentApplicationService
b. A CompanyRepresentative client, via the CompanyRepMenu, interacts with CompanyInternshipManagementService and CompanyApplicationApprovalService.
c. A CareerCenterStaff client, via the StaffMenu, interacts with StaffInternshipApprovalService, StaffWithdrawalApprovalService, and various reporting services

This ensures that the Student client, for instance, is not forced to depend on methods for approving internships which it does not need.

3.5.5. Dependency Injection Principle (DIP)

The system is designed so that high-level modules depend on abstraction (interfaces) rather than low-level concrete implementations.

a. Boundary Classes and Services: The highest-level boundary classes (the menus) depend on service interfaces, not concrete controllers. For example, the StudentMenu class depends on the StudentApplicationService interface, not the concrete ApplicationController that implements it.

4. Additional Features

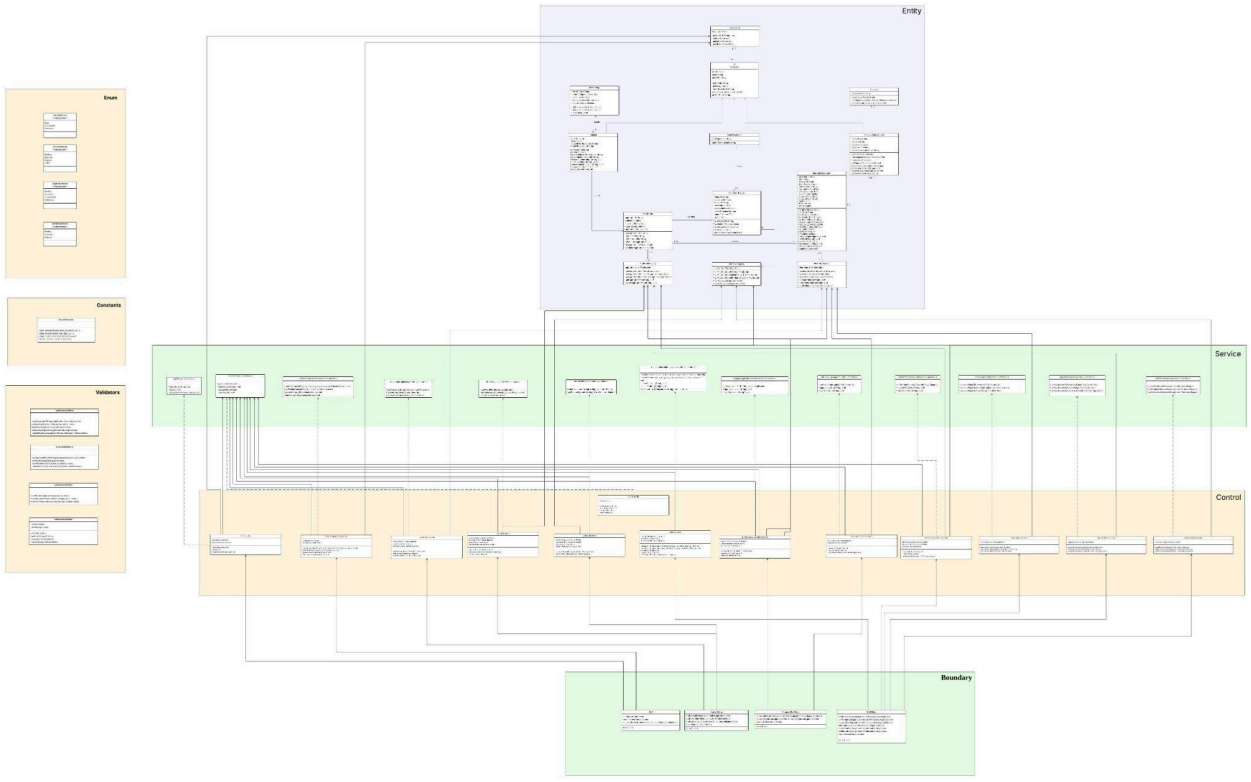
4.1.1. Edit Internship

Company reps can edit existing internships with partial updates (keep existing values if not provided).

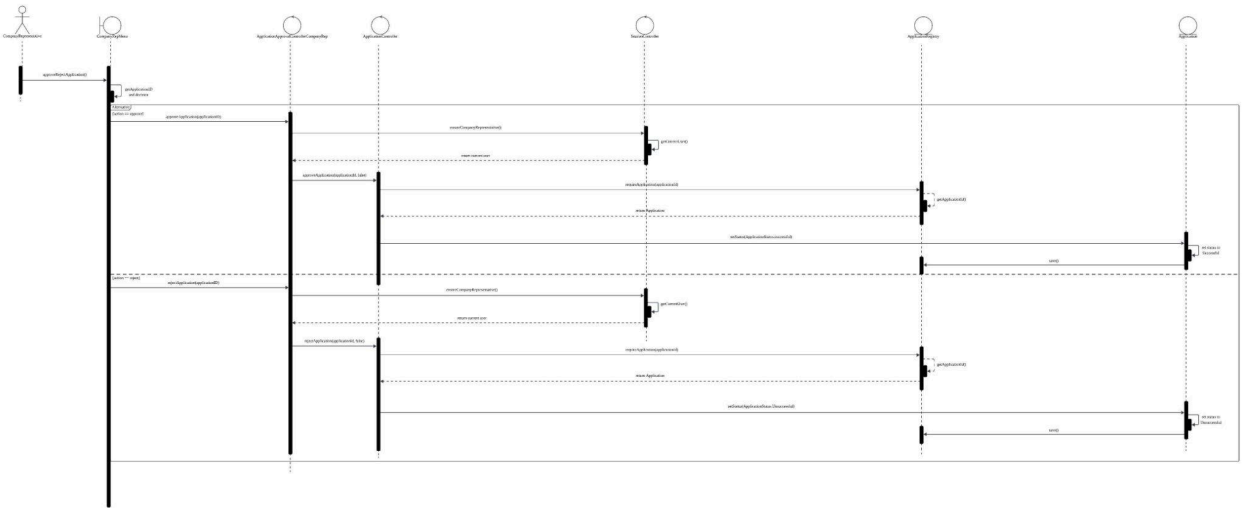
4.1.2. Delete Internship

Company reps can delete internships when prompted.

5. UML Class Diagram



6. Sequence Diagram for Company Representative Approve/Reject Applications



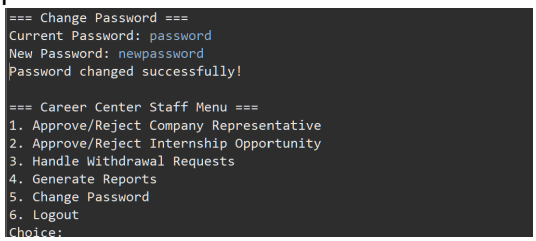
7. Functional Tests and Results

7.1. Authentication

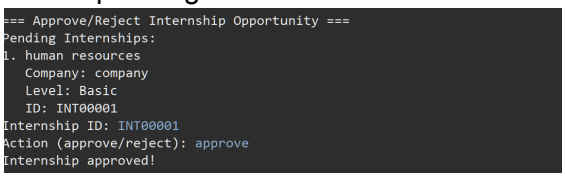
Test case	Test Description	Expected Result	Pass/Fail

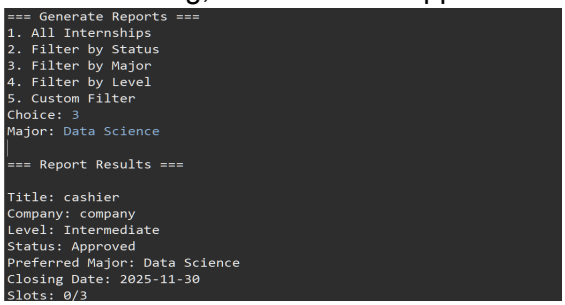
1	[Cannot Login]
	a. Entering wrong UserID for the selected role.
	b. Entering correct UserID but wrong password for the selected role.
		User fails to log in and returns to the login menu.
		`=== Login ===`
		`1. Login`
		`2. Register as Company Representative`
		`Choice: 1`
		`User ID: wrong`
		`Password: wrong`
		
		`Error: User ID not found`
		`=== Login ===`
		`1. Login`
		`2. Register as Company Representative`
		`Choice:`
			Pass

2	[Successful Login]
	User enters correct UserID and password for the selected role.
		User logs in successfully and a different menu is displayed for both Staff and Student, with different functions available.
		`=== Login ===`
		`1. Login`
		`2. Register as Company Representative`
		`Choice: 1`
		`User ID: sng001`
		`Password: password`
		
		`Login successful! Welcome, Dr. Sng Hui Lin (CareerCenterStaff)`
		
		`=== Career Center Staff Menu ===`
		`1. Approve/Reject Company Representative`
		`2. Approve/Reject Internship Opportunity`
		`3. Handle Withdrawal Requests`
		`4. Generate Reports`
		`5. Change Password`
		`6. Logout`
		`Choice:`
			Pass

3	[Change Password]	User is prompted to enter old password for verification and confirmation, then directed to enter new password.Once successful, User returns to the previous menu. 	Pass

7.2. Career Center Staff Functionality

Test case	Test Description	Expected Result	Pass/Fail
1	[Approve/Reject Company Representative] a. Approve b. Reject	A list of pending registrations is displayed, and the Staff is prompted to enter the user ID to approve/reject. a. The Staff member enters “approve”, and the user is registered as a company representative and removed from the pending list. b. The Staff member enters “reject”, and the user is rejected and removed from the pending list. 	Pass
2	[Approve/Reject Internship Opportunity] a. Approve b. Reject	A list of pending internships is displayed, and the Staff is prompted to enter the internship ID of the internship to approve/reject. a. The Staff member enters “approve”, and the internship is approved and removed from the pending list. b. The Staff member enters “reject”, and the internship is rejected and removed from the pending list. 	Pass
3	[Handle Withdrawal	A list of pending withdrawal requests is displayed, and the	Pass

	Requests] a. Approve b. Reject	Staff is prompted to enter the requestID to approve/reject. If the requestID is valid: a. The Staff enters “approve”, and the application is withdrawn, and the withdrawal request is removed from the pending list. b. The Staff enters “reject”, and the withdrawal request is rejected and removed from the pending list. 	
4	[Generate Reports] a. All Internships b. Filter by Status c. Filter by Major d. Filter by Level e. Custom Filter	a. The Staff chooses to show all internships, and a list of all internships is generated. b. The Staff chooses to filter by status, then is prompted to enter the status (“Pending”, “Approved”, “Rejected”, “Filled”) to filter by. After entering, the list of filter opportunities is generated. c. The Staff chooses to filter by major, then is prompted to enter the Major to filter by. After that, the list of filter opportunities is generated. d. The Staff chooses to filter by level, then is prompted to enter the level (“Basic”, “Intermediate”, “Advanced”) to filter by. After entering, the list of filter opportunities is generated. 	Pass
5	[Change Password]	Refer to 5.1.3	Pass

7.3. Student Functionality

Test case	Test Description	Expected Result	Pass/Fail
1	[View Available Internships]	A list of available internships is displayed.	Pass

2	[Apply for Internship]	The Student is prompted to enter the internshipID. If the internshipID is valid, an application will be submitted.	Pass
3	[View My Applications]	A list of applications will be displayed.	Pass
4	[Accept Placement]	After a list of successful applications is displayed, the Student will be prompted to enter the applicationID of the successful application he/she wants to accept. If the applicationID is valid, the placement will be accepted.	Pass
5	[Request Withdrawal]	After a list of applications is displayed, the Student will be prompted to enter the applicationID of the application he/she wants to withdraw. If the applicationID is valid, the Student will be prompted to enter the reason for withdrawal(optional), and the withdrawal request will be submitted.	Pass
6	[Change Password]	Refer to 5.1.3	Pass
7	[Filter/Sort Internships] a. Filter by Status b. Filter by Level c. Sort Options d. Reset Filters	a. The Student chooses to filter by status. The Student is then prompted to select the status (“1” for Approved, “2” for Pending, “3” for Rejected, “4” for Filled, “5” for Clear). If the Student inputs a valid option, the internships are filtered. b. The Student chooses to filter by level. The Student is then prompted to select the level (“1” for Basic, “2” for Intermediate, “3” for Advanced). If the Student inputs a valid option, the internships are filtered. c. The Student chooses to filter by sort options. The Student is then prompted to select the sort option (“1” for Alphabetical, “2” for Closing Date, “3” for Level, “4” for Filled, “5” for Clear). If the Student inputs a valid option, the internships are filtered. d. The Student chooses to reset filters, and the filters are reset.	Pass

7.4. Company Representative Functionality

Test case	Test Description	Expected Result	Pass/Fail
1	[Create Internship Opportunity]	The Representative is asked to key in details of the internship (e.g Title, Level, Opening Date, etc). If all the details entered are valid, the internship is created and will be pending approval from the Staff.	Pass
2	[View My Internships]	A list of internships created by the Representative is displayed.	Pass
3	[Edit Internship Opportunity]	The Representative is asked to key in the internshipID of the internship he/she wishes to edit. If the internshipID is valid, the Representative will be asked to key in the details of the internship. If the details are valid, the internship will be updated.	Pass
4	[Delete Internship Opportunity]	The Representative is asked to key in the internshipID of the internship he/she wishes to delete. If the internshipID is valid and the Representative confirms the deletion, the internship will be deleted.	Pass
5	[View Applications for Internship]	A list of applications will be displayed.	Pass
6	[Approve/Reject Application] a. Approve b. Reject	The Representative is asked to key in the applicationID to approve/reject. If the applicationID is valid: a. The Representative enters “approve”, and the application is approved. b. The Representative enters “reject”, and the application is rejected.	Pass
7	[Toggle Internship Visibility]	The Representative is asked to key in the internshipID of the internship to toggle the visibility of. If the internshipID is valid, the visibility is toggled.	Pass
8	[Change Password]	Refer to 5.1.3	Pass

8. Reflection

8.1. Introduction

Initially, our team approached development with a strong focus on building a working system, but we overlooked the importance of architectural patterns and SOLID principles. This resulted in tightly coupled classes, unclear responsibilities, and a structure that would not scale well as the project grew and more additions and modifications were to be made. We realised that proper architectural discipline was crucial for building a maintainable and extensible system, given the complexity of our domain. We transitioned into an architecture with 4 separate layers (Entity, Control, Boundary and Service layer). This helps us reorganise the system into clearer layers and significantly improve the overall quality of our design.

8.2. Difficulties Encountered

One of our biggest challenges that we faced was the improper distribution of responsibilities across the system. Controllers were overloaded with business logic and data persistence. Entities handled their own validation, violating the Single Responsibility Principle (SRP). Our initial design also struggled with extensibility; adding new features often required modifying existing classes, conflicting with the Open/Closed Principle.

We also faced difficulties managing complex class relationships, constraints and the interactions between multiple user roles (students, company representatives and staff). We had to ensure that each role had appropriate access, behaviours, and constraints required careful coordination across controllers, service interfaces and registries. Implementing validation logic was another challenge, especially before we introduced dedicated validator classes.

8.3. Knowledge Learnt

Through the redesign, we gained a deeper understanding of object-oriented programming and the practical application of SOLID principles. Some of the key lessons learned were the importance of SRP, Open/Closed Principle, 4-layered architecture, Design patterns, Abstraction and Inheritance, Validation and Data Integrity, Session and Authorisation Logic. These insights secure authentication and ownership checks.

8.4. Further Improvements

We can potentially use hashing functions to manage the passwords to have a quicker response time. We can also expand documentation for class responsibilities, workflows and sequence diagrams to aid future developers. We can also aim for better scalability by exploring persistence layers beyond in-memory registries, such as database-backed repositories.