CE/CZ2002: Object-Oriented Design & Programming

Assignment: Building an OO Application

Done by: Group 6

Declaration of Original Work for CE/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 honored the principles of academic integrity and have upheld the 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 (CE2002 or CZ2002)**	**Lab Group**	**Signature/Date**
Chua Wen Qing	CZ2002	SS6	25/11/20
Jessica Halim	CZ2002	SS6	25/11/20
Fung Kai Xiang Daniel	CZ2002	SS6	25/11/20
Lee Yih Jie	CZ2002	SS6	25/11/20

Important notes:

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

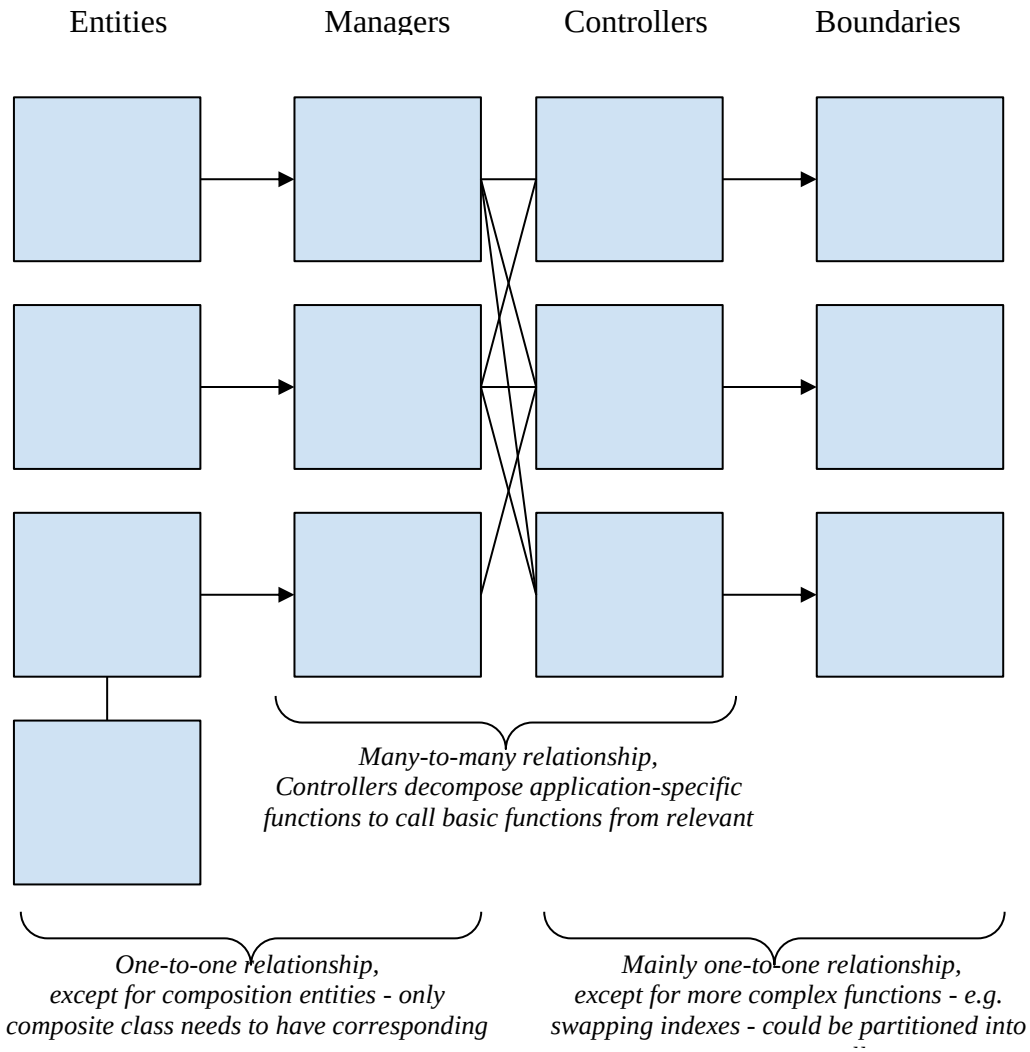
1. *Design Considerations*

<u>Overarching Approach</u>

We use four main categories of classes in our STARS application:

- Entity Classes: User.*java, Admin.java, Student.java, Courses.java, Index.java, Lesson.java, PendingSwop.java, TimeTable.java*
- Manager Classes
    - Interface Classes: *BasicManager.java, RegistrationManager.java*
    - Concrete Classes: *adminManager.java, studentManager.java, courseManager.java, indexManager.java, accessDateTimeManager.java, swopManager.java*
- Controller Classes: *adminController.java, studentController.java, loginController, notrificationController.java , swopController.java*
- Boundary Classes: *adminBoundary.java, studentBoundary.java, loginPage.java*

Flow of program (with regards to main class categories)



a) Entities represent the base object classes which contain data that persist throughout the use of the application (e.g. Course, Index, Student, Lesson).

b) Managers are classes which have the main responsibility of updating, saving and retrieving the entity objects. All managers are based on a foundational interface — 'BasicManager'. 'BasicManager' outlines a few main functions to modify the entity classes. 'RegistrationManager' extends 'BasicManager' with extra functions, and is implemented by only some relevant managers, while all other managers implement 'BasicManager'.

c) Thereafter, the controller classes combine basic functions of the managers to satisfy more complex functionalities required by the application.

d) Finally, the boundary classes collect the necessary inputs from the user to then invoke and return the appropriate outputs generated by controller classes.

Assumptions

a) Student Functions
   i) If a student attempts to register into a course with zero vacancies, they will be automatically added to the back of the index-specific waitlist.
   ii) If a student is in the waitlist of an index of some course A, they will be prevented from registering into the index of some course B with a clashing timetable with the initial course A index.
   iii) A student cannot be waitlisted and registered in the same course at the same time.
b) Admin Functions
   i) Multiple admin accounts are present in the application database.
   ii) When creating a Course, admins will have the relevant information queried for by the application at hand (i.e. Course Code, Number of AUs, School, all Index details and Lesson details within each Index).
   iii) Once entered, admins cannot change the lesson details of any index of a course (doing so could lead to unexpected clashes after students have already registered some courses).

Object-Oriented Design Principles

1. **Open-Closed Principle (OCP)**

The open-closed principle denotes the ability of the program to allow for extensions in its functions, without the changing of source code. An easy way to implement this is via the use of interfaces. The interface is closed to modification, but the overarching program easily allows modification via the creation of new classes which inherit from the interface.

A key element of our application design is the interface 'BasicManager' provides a formal template of functions the managers use to retrieve, modify and store entity classes (i.e. find(Object), create(Object), delete(Object), fileRead(), fileWrite()). As a result, the program can be extended to handle more entity classes, such as 'Teaching Assistants' who may 'register' to assist professors

in one or a few Courses, as long as an appropriate corresponding manager is created (e.g. 'TAManager') to manage the relevant entity class.

Dynamic Polymorphism

The usage of interfaces in our application allows for dynamic binding. Firstly, during compilation of the program, 'BasicManager' is instantiated. When the application is run, 'BasicManager' is assigned a 'role' to be able to modify the corresponding entity in order to serve the purpose of the application-specific function, according to the user's request at runtime. For example, when an Admin user requests to update the details of the access datetimes of a group of students, the 'BasicManager' is instantiated to 'accessDateTimeManager' and is thus given the ability to modify the Access Date Time entries to incorporate the updates requested by the Admin user.

2. **Interface Segregation Principle (ISP)**

ISP maintains that implementation of classes should be specific and not depend on interfaces that they do not use. In our application, another interface implemented is 'RegistrationManager' which extends 'BasicManager' to include the functions register(Object, Matriculation Number) and deregister(Object, Matriculation Number). The registration and deregistration of students in and from an index only concern two entity classes: Index and Student. Resultantly, only 'indexManager' and 'studentManager' implement this interface.

3. **Dependency Inversion Principle (DIP)**

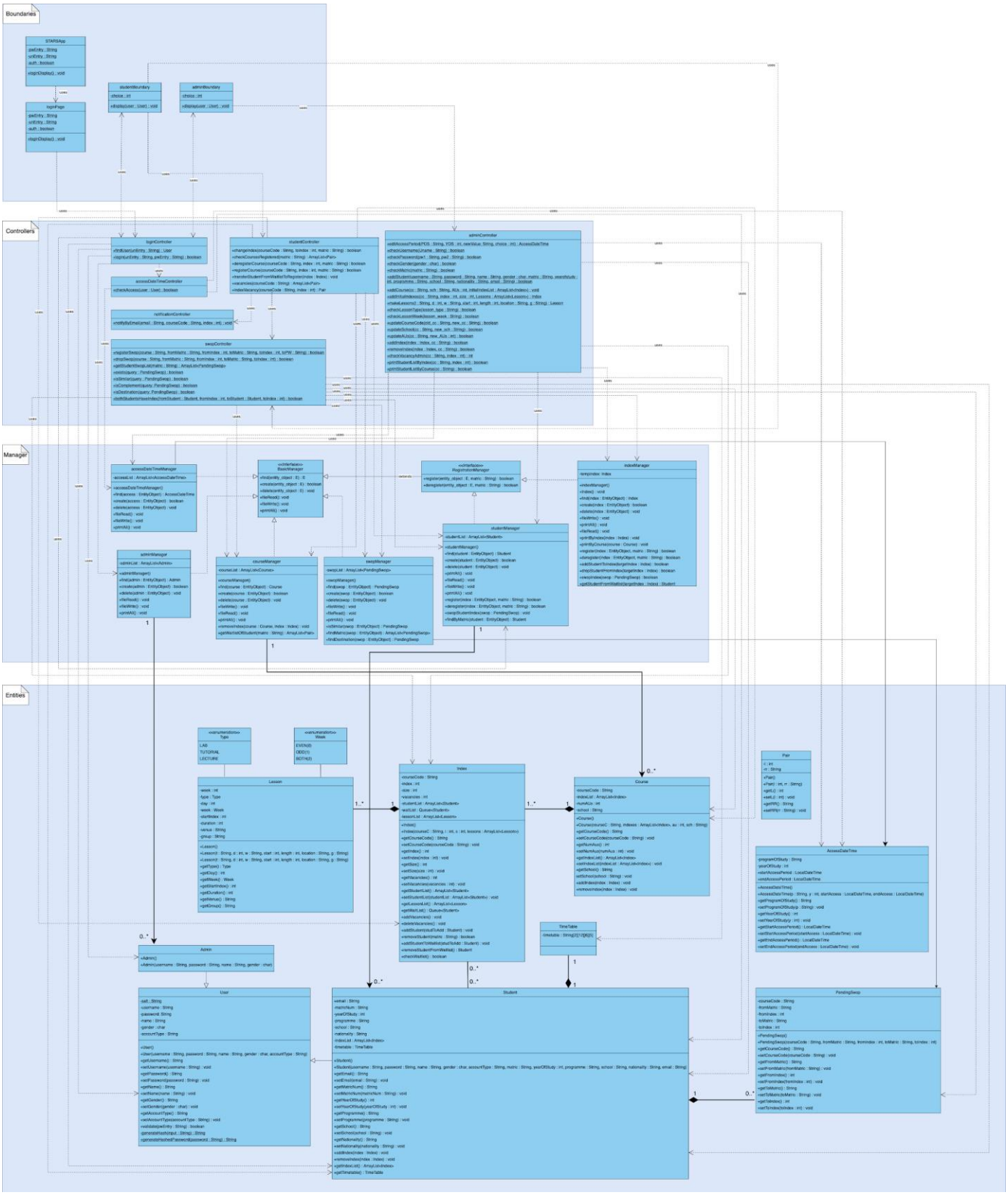
DIP promotes the program's dependence on abstraction rather than concrete classes. This is to mitigate the application's fragility when it is based on volatile concrete classes which frequently require changes in source code.

Our application-specific functions are implemented in the Controller classes, which decompose these functions into simpler basic functions (outlined by the interface manager classes) employed by the manager classes. As a result, the entire application centers on the abstraction provided by the manager class interfaces (i.e. BasicManager, RegistrationManager).

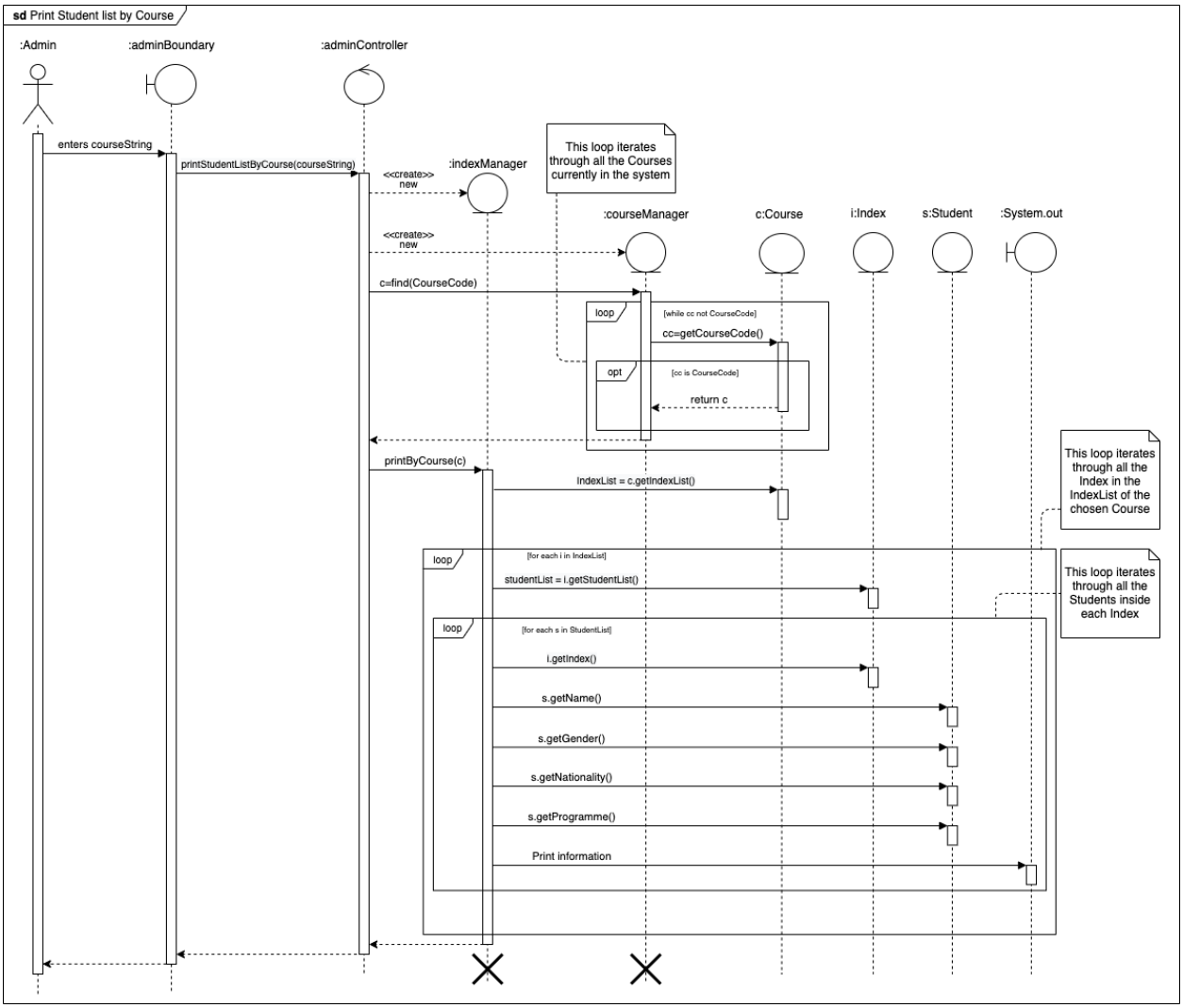
Video Demonstration

https://youtu.be/m6_9Z-4hD9E

2. Detailed UML Class Diagram



3. Detailed UML Sequence Diagram of stated function.



4. Test Cases and Results

1. Student Login

	Test Case	Expected Outcome
a	Login before allowed period (dates)	You are not allowed to access the system at this time. Please try again.
b	Login after allowed period (dates)	You are not allowed to access the system at this time. Please try again.
c	Wrong password	Invalid username or password. Please try again.

2. Add a student

	Test Case	Expected Outcome
a	Add a new student	|Username |Matric Number |Programme of Study |School | +----------+ |WCHAN999 |U2042323A |CSC |SCSE
JIEEL210	U1838484D	CE	SCSE
SHTAN66	U1921316F	REP	SCSE
DHIDE024	U1721316F	MAE	MAE
TQYING013	U1824834F	BCG	SCSE
JYEO11	U1324832D	CS	WKWSCI
ZIQI221	U1724832D	ECON	SSS
WKAIJ221	U2024132D	DA	ADM
YGOH921	U1923122D	ACC	NBS
SAMS001	U1819102D	BUS	NBS
GREGL980 |U4444444H |CSC |SCSE | +----------+ Please select one of the options below: 1. Edit Student Access Period 2. Add a Student 3. Add a course 4. Update a course 5. Check available slot for an index number 6. Print student list by index number 7. Print student list by course 0. Log out
b	Add an existing student	Student is already added!
c (Invalid data input)	Enter unmatching password during password confirmation	Error. The passwords you entered were not the same.

	Enter invalid gender input	Invalid Gender Input.
	Enter invalid Year of Study	Year of Study must be between 1 to 5. Please try again.

3. Add a course

	Test Case	Expected Outcome
a	Add a new course (with combination of (ii) from above)	+----------+ |Course |School |AUs |Indexes/Vacancies | +----------+ |CZ2002 |SCSE |3 |10205 (30/30) | | | | | |10206 (30/30) | | | | | |10207 (30/30) | |CZ2003 |SCSE |3 |10208 (30/30) | | | | | |10209 (30/30) | | | | | |10210 (30/30) | |CZ2004 |SCSE |3 |10211 (30/30) | | | | | |10212 (30/30) | | | | | |10213 (1/1) | |CZ2012 |SCSE |3 |123 (20/20) | +----------+
b	Add an existing course	The course *course_code* already exists.
c	Invalid data entries	Test case [Lesson type]: Invalid lesson type. Please try again. Test case [Lesson week]: Invalid lesson week. Please try again. Test case [for the rest of lesson/index details]: Invalid lesson/index input entered. Please enter all lesson/index details of this index again. Error [InputMismatchException | IllegalArgumentException]: Invalid input entered. Please enter course details again.

4. Register student for a course

	Test Case	Expected Outcome
a	Add a student to a course index with available vacancies.	Registration for *course_code*, index *index_number* successful.
b	Add a student to a course index with 0 vacancies in Tut / Lab.	There are currently no vacancies for this index. You will be added to the waitlist of *index_number*. Registration for *course_code*, index *index_number* was not successful.
c	Add a student to the same course again	You are already registered in this course. If you wish to change index, please go to [5. Change index of existing registered index].
d	Add a student to a course index that student is waitlist-ed under	You are already in the waitlist of this index.
	Add a student to a course index where the student is waitlist-ed under a different index of the same course.	You are already in the waitlist of index *index_number* in this course. Please go to [9. Drop waitlisted courses] to drop your waitlisted index if you wish to register to this index
e	Add a student to a course which will cause student to pass maximum AU limit	You will exceed your total AUs. Please choose another course
f	Invalid data entries	Test case [Invalid course code]: No Course of Course Code *invalid_course_code* Found. Registration for *invalid_course_code*, index *index_number* was not successful. Test case [Valid course code, Invalid index]: Index *invalid_index_number* is not found!

		Try again. Registration for *course_code*, index *invalid_index_number* was not successful.

g	List courses registered	`>Total AUs: 6`
		` Registered courses:`
		`+----------+`
		`| Course Code | Index |`
		`| CZ2004 | 10213 |`
		`| CZ2002 | 10206 |`
		`+----------+`

5. Check available slot in a class (vacancy in a class)

	Test Case	Expected Outcome
a	Check for vacancy in course index	Number of vacancies for index *index_number* of *course_code* : [*vacancy/total_size*]
b	Check for vacancy of all indexes in a course	Number of vacancies for index *index_number_1*: [*vacancy/total_size*] Number of vacancies for index *index_number_2*: [*vacancy/total_size*] Number of vacancies for index *index_number_3*: [*vacancy/total_size*]
c	Invalid data entries (eg course code, class code etc)	Test case a) The course code you entered does not exist. Test case b) The course code/ index you entered does not exist.

6. Day/Time clash with other course

	Test Case	Expected Outcome

a	Add a student to a course index with available vacancies.	There is a clash! You will not be registered to this course. Registration for *course_code* , index *index_number* was not successful.

7. Waitlist notification

	Test Case	Expected Outcome
a	Add studentA to a course index with 0 vacancies	There are currently no vacancies for this index. You will be added to the waitlist of *index_number* .
b	Drop studentB from the same course index	Course number, index “” is dropped. *Email successfully sent to ‘studentA’*
c	Display studentA timetable (with waitlist-ed courses)	*Refer to Appendix A for the timetable image.*
d	Display student own list of waitlist-ed courses (Under “Check Courses Registered” function)	>Total AUs: 6 Registered courses: +----------+ | Course Code | Index | | CZ2002 | 10206 | +----------+ Waitlisted courses: +----------+ | Course Code | Index | | CZ2004 | 10213 | +----------+

8. Print student list by index number, course

	Test Case	Expected Outcome

<table>
<tr><td>a</td><td>Print list by
(i) Course
(ii) index</td><td>i)By Course

```
|Name                        |Gender |Nationality   |Programme          |Index |
+-------------------------------------------------------------------------+
|Chan Wei Chang              |M      |Singaporean   |CSC                |10205 |
|Shawn Tan                   |M      |Singaporean   |REP                |10205 |
|Yvette Goh Qian Wei         |F      |Malaysian     |ACC                |10205 |
|Zi Qi                       |F      |Chinese       |ECON               |10206 |
+-------------------------------------------------------------------------+
```

ii)By Index

```
|Name                        |Gender |Nationality   |Programme          |
+-------------------------------------------------------------------------+
|Chan Wei Chang              |M      |Singaporean   |CSC                |
|Shawn Tan                   |M      |Singaporean   |REP                |
|Yvette Goh Qian Wei         |F      |Malaysian     |ACC                |
+-------------------------------------------------------------------------+
```

</td></tr>
<tr><td>b</td><td>Invalid data entries (eg course code, index code etc)</td><td>Test case [Invalid course_code]:
No Course of Course Code invalid_course_code Found.

Test case [Invalid index]:
No Index of Number invalid_index_number Found.</td></tr>
</table>

9. Edit Student Access Period

	Test Case	Expected Outcome
a	Changing the access start time/end time	|Programme |Year |Access Start |Access End | --- |CSC |1 |2020-11-25T00:00 |2021-12-01T14:00 | --- (displays the updated access date-time entry)
b	Invalid Access Date-time entry (e.g. access date-time for the course programme does not exist in the database)	Access Datetime Entry does not exist.

Appendix A

Even Timetable:

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
08:30-09:30						
09:30-10:30						
10:30-11:30		<registered> CZ2002:LECTURE LT2A [G1]				
11:30-12:30					<waitlisted> CZ2004:TUTORIAL TR15 [SS3]	
12:30-13:30			<registered> CZ2002:TUTORIAL TR17 [SS2]		<waitlisted> CZ2004:LECTURE LT2A [G3]	
13:30-14:30					<registered> CZ2002:LAB SWL1 [SS1]	
14:30-15:30					<registered> CZ2002:LAB SWL1 [SS1]	
15:30-16:30						
16:30-17:30						
17:30-18:30						
18:30-19:30						
19:30-20:30						

Odd Timetable:

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
08:30-09:30						
09:30-10:30						
10:30-11:30		<registered> CZ2002:LECTURE LT2A [G1]				
11:30-12:30					<waitlisted> CZ2004:TUTORIAL TR15 [SS3]	
12:30-13:30			<registered> CZ2002:TUTORIAL TR17 [SS2]		<waitlisted> CZ2004:LECTURE LT2A [G3]	
13:30-14:30						
14:30-15:30						
15:30-16:30			<waitlisted> CZ2004:LAB SWL1 [SS1]			
16:30-17:30			<waitlisted> CZ2004:LAB SWL1 [SS1]			
17:30-18:30						
18:30-19:30						
19:30-20:30						