Second Iteration

Wei Luo (wl2671), Zhuangyu Ren (zr2209), Chi Zhang (cz2517), Danyang He (dh2914)

Repository Link

The link to our repository is: https://github.com/pow25/vergilplus

Use Cases

Name of Use Case	Search Professor
Actor	1. Potential Students 2. Students 3. Professors
Description	Every Vergilplus user can input the professor's name that he/she is interested in, and the system will return courses taught by that professor and some descriptions about that professor based on existing reviews.
Trigger	Users input name of the professor
Preconditions	1. User is connected to AWS service 2. AWS service can understand user's input, number of invalid inputs is less than 5
Basic flow	1. Vergilplus connects to AWS 2. Users types greeting such as "Hi" and "Good morning" to get started 3. Vergilplus connects to AWS Lex service and passes all the information that the user has input 4. AWS Lex asks the user whether he or she has some interested topics in mind 5. User types "Yes, I am interested in professor XXX XXX" (in the format of [first name last name]) 6. AWS Lex passes related information to Vergilplus 7. Vergilplus calls the function getWordsProfessor to return some descriptive words for that professor 8. AWS Lex asks if the user has taken any courses 9. User types their taken courses 10. AWS confirms users' taken courses 11. User will type "yes"

	12. AWS returns “searching results….. Please wait” 13. User will type “ok” to abort aws connection 14. Vergilplus calls searchKeywords function 15. Vergilplus outputs those courses taught by the professor to the console. 16. Vergilplus will exit
Alternative flow	9A1. User has not taken any courses 1. User types “No, I haven’t taken any course” 2. AWS will not confirm, instead it will tell Vergilplus to search directly 3. The use case continue (12) 9A2. User will type the course number, like COMS4771 1. The use case continue (10) 9A3. User will type the course name, like database 1. The AWS will match the “database” to correlated course name, like “introduction to database” 2. The AWS will then ask what’s the course number of this course 3. The user will input the course number 4. The use case continue (10)
Exception flow	EX1: User’s inputs cannot be understood by AWS service 1. User gives an invalid input which is not understandable to AWS service 2. User gives invalid inputs more than 5 times, this software will exit automatically EX2: User wrongly spells professor’s name 1. User spells the professor’s name wrong so system cannot find relative information 2. User only types the professor’s first name

Name of Use Case	Search Course Topics
Actor	1. Potential Students 2. Current Students 3. Professors
Description	A new student using vergilplus software can input the key word of his/her field of interest, then the system will return courses related to that key word based on all courses' descriptions and titles.
Trigger	The user input the key words
Preconditions	1. User is connected to AWS service 2. AWS service can understand user's input, number of invalid inputs is less than 5
Basic flow	1. Vergilplus connects to the AWS service. 2. Users types greeting, like "Hi", "Good morning" to get started 3. Vergilplus will connect to the AWS Lex service and pass all the information user inputted and AWS outputted. 4. AWS Lex will ask the user whether he or she has some interested topics in mind? 5. User will type "Yes, I am interested in XXXX" 6. AWS Lex will ask if the user have taken any courses or not 7. User will type their taken courses 8. AWS will confirm users' taken courses 9. User will type "yes" 10. AWS will return "searching results….. Please wait" 11. User will type "ok" to abort aws connection 12. Vergilplus will call searchKeywords function 13. Vergilplus will output those recommended courses to the console. 14. Vergilplus will exit
Alternative flow	7A1. User has not taken any course 1. User type "No, I haven't taken any course" 2. AWS will not confirm, instead it will tell the Vergilplus to search directly 3. The use case continue (10) 7A2. User will type the course number, like COMS4771 1. The use case continue (8) 7A3. User will type the course name, like "database" 1. The AWS will match the "database" to correlated course name, like "introduction to database"

	2. The AWS will then ask what's the course number of this course 3. The user will input the course number 4. The use case continue (8) 9A1. User thinks he previously typed courses are wrong or he hasn't inputted complete courses 1. User type "No" 2. AWS will ask him for re-typing his taken courses 3. The use case continue (8)
Exception flow	EX1: User's inputs cannot be understood by AWS service. 1. User gives an invalid input which is not understandable to AWS service. 2. User gives invalid inputs more than 5 times, this software will exit automatically.

Name of Use Case	Recommend courses
Actor	Students
Description	Vergilplus software returns courses based on reviews' scores. If user inputs courses he\she has taken, then the system will return courses by filtering the courses user has taken and the courses that user can take after meeting some prerequisites.
Trigger	The user choose recommend function or input courses that he/she has taken
Preconditions	1. User is connected to AWS service 2. AWS service can understand user's input, number of invalid inputs is less than 5
Basic flow	1. Vergilplus connects to the AWS service 2. Users types greeting, like "Hi", "Good morning" to get started 3. Vergilplus will connect to the AWS Lex service and pass all the information user inputted and AWS outputted 4. AWS Lex will ask the user whether he or she has some interested topics in mind? 5. User will type "No. Just recommend some popular ones" 6. AWS Lex will ask if the user have taken any courses or not 7. User will type their taken courses 8. AWS will confirm users' taken courses 9. User will type yes 10. AWS will return "searching results….. Please wait" 11. User will type "ok" to abort aws connection 12. Vergilplus will call recommendCourses function 13. User will type "ok" to abort aws connection 14. Vergilplus will output those recommended courses to the console 15. Vergilplus will exit
Alternative flow	7A1. user have not taken any course 1. User type "No, I haven't taken any course" 2. AWS will not confirm, instead it will tell the Vergilplus to search directly 3. The use case continue (10) 7A2. User will type the course number, like COMS4771 1. The use case continue (8) 7A3. User will type the course name, like database 1. The AWS will match the "database" to correlated course name,

	like “introduction to database” 2. The AWS will then ask what’s the course number of this course 3. the user will input the course number 4. the use case continues(8) 9A1. User thinks he previously typed courses are wrong 1.User type “No” 2.AWS will ask him for re-typing his taken courses 3.The use case continue (8)
Exception flow	EX1: User’s inputs cannot be understood by AWS service 1. User gives an invalid input which is not understandable to AWS service 2. User gives invalid inputs more than 5 times, this software will exit automatically EX2: User inputs the wrong course numbers 1. User inputs wrong course numbers that the software cannot find such a course in database

Test Suite

All of the test suite resides in src/test/java/com/vplus/maven/vplus/AppTest.java. The equivalence partitions that we go into details here pertains to the major subroutines, which are located in MasterController.java.

1. recommendCourses
Valid Equivalence Class (including boundary cases):

- `recommendCoursesValidEq`
    - Valid taken courses input of size 2
- `recommendCoursesValidEqTestBoundaryMin`
    - Valid taken course of size 0
- `recommendCoursesValidEqTestBoundaryMax`
    - Max valid taken courses of size 12
- `recommendCoursesValidEqTestBoundaryBelowMax:`
    - Valid taken courses length just below max (size 11)

Invalid Equivalence Class:

- `recommendCoursesInValidEqTestBoundaryAboveMax`
    - Invalid taken courses length just above max (size 13)
- `recommendCoursesInValidEqTestBoundaryNull`
    - Invalid case when the taken courses list is null

2. processTakenCourses
Valid Equivalence Class (including boundary cases):

- `processTakenCoursesValidEq`
    - Valid taken courses input of size 2
- `processTakenCoursesValidEqTestBoundaryMin`
    - Valid taken course of size 0

Invalid Equivalence Class:

- `processTakenCoursesInvalidEqTestBoundaryNull`
    - Taken courses list is null

3. filterByPrerequisites
Valid Equivalence Class:

- `filterByPrerequisitesValidEqHasPreq`
    - Input courses list has one course, and that course has a prerequisite
- `filterByPrerequisitesValidEqNoPreq`
    - Input courses list has one course, and that course has no prerequisite

- `filterByPrerequisitesValidEqTestBoundaryMin`
    - Input courses list is empty

Invalid Equivalence Class:

- `filterByPrerequisitesInValidEqTestBoundaryNull`
    - Input courses list is null

4. getWordsProfessor

Valid Equivalence Class:

- `getWordsProfessorValidEqHasRecord`
    - Professor name is valid and in our db
- `getWordsProfessorValidEqNoRecord`
    - Professor name is valid but not in our db

Invalid Equivalence Class:

- `getWordsProfessorInvalidEqEmptyString`
    - Empty professor name
- `getWordsProfessorInvalidEqNullString`
    - Professor name string is null
- `getWordsProfessorInvalidEqInvalidNameLengthOne`
    - Invalid professor name of length one - name must be two words
- `getWordsProfessorInvalidEqInvalidNameLengthThree`
    - Invalid professor name of length three - name must be two words

Code Coverage

For the branch coverage testing, our project used Jacoco to generate the coverage report. The report is in the html format that stores in the the directory report/jacoco-ut. The whole process is integrated into Maven, so travis-CI will generate the coverage report after running Test cases and upload it to the github repository. In addition, we also integrated CODECOV to the travis, so that the report can be viewed online (by clicking on the badge of README.md).

Our code coverage reaches 93%.