CSC2063: Group Coursework (Implementation Part)

Service-Oriented System for Student Information Management

Your brief is to deliver the Java implementation of a service-oriented software system that stores and manages information related to the students and the academic staff members of a university school. The suggested work plan for your project, the submission details, the due date of your project, and the technical description of the software system that you need to implement are specified in the sections of this document.

Suggested Work Plan

Please note that the implementation part of the coursework covers the material that is taught in the following weeks of the module: Weeks 4 - 8. A suggested work plan includes the following steps:

1. If needed, redesign the Web services of the system based on the feedback on your system in the first part of the project (**Week 9**)
2. Develop in a bottom-up way a separate Java Maven Dynamic Web Project in Eclipse EE for each SOAP Web API service that you have (re-)designed to include in your system. The services should implement the use cases that are described in the remainder of this document (**Weeks 9-10**)
3. Develop the JAXB serialization technique in each Web service for permanently storing in XML files the data managed by the service. More information about storing data are provided in the remainder of this document (**Week 11**)
4. Develop the clients of the SOAP Web APIs at the points of the system that are needed (**Week 12**).
5. Testing the functionality of the whole system based on the testing cases that are described in the remainder of this document (**Week 13**).

Q&A Sessions

- You can ask your questions/get help on your project by attending the in-person practical sessions on Tuesdays (location: CSB/0G/028), where you can ask for having 1-to-1 meetings during the practical sessions with the student support officer, Mr Leo McHugh, l.mchugh@qub.ac.uk (Tuesdays, 11-1pm).
- You can join the online meetings of my weekly office-hours to possibly ask your questions about the coursework (Thursdays, 3-5.30pm).

Canvas Submission and Due Date (Details/Restrictions)

Each group member should submit via Canvas at this page the following file(s):

- As many ZIP files as many the Maven Eclipse projects included in the implementation of your service-oriented system. In other words, you need to create one ZIP file for each Maven project of your system. To this end, you should right click on each Eclipse project that you have defined, you should export the project on your desktop, you should right click on the project and “send it to" a compressed (zipped) file. For instance, if your system includes four Maven Eclipse projects, then you should upload on Canvas four ZIP files.

- Each file must be named by applying the following pattern (tokens separated by underscore):
  teamNumber_StudentID_StudentName_EclipseProjectName.zip
  E.g., team12_4000000_TomLeeMcallen_RegistrationService.zip
- Each submitted file (i.e., Eclipse project) should not contain .jar files or .xml files.

Submission deadline: 10 April 2022
Feedback release: around 2-3 weeks after the submission deadline.

Note: The mark of the second/implementation part of the coursework is an individual mark for each student. To mark the implementation part of the coursework, we will compare your work against all the requirements that are specified in the rubric of the second part of the coursework that is available here. To this end, you should deliver a file that meets all the requirements that are specified in the rubric even if you have chosen to work individually or your group has ended up having less than three members.

System Description

Functional Requirements

The system manages information about academic staff members and students. In detail, the system manages the following information:

- *Role*: a role has one of the following predefined values: ACADEMIC_STAFF_MEMBER or STUDENT
- *Students*: an array of students is maintained (max array length: 1000)
- *Student*: the information that is maintained for each student is the following: an ID (an integer number), an array of module codes (max array length: 20) and an array of modules for students (max array length: 20)
- *Module*: a module is characterized by a code, an academic year, and a mark (the mark is declared only if the module is related to a student)
- *Module code*: a module code can equal one of the following predefined values: CSC1022, CSC1023, CSC1024, CSC1025, CSC1026, CSC1027, CSC1028, CSC1029, CSC1030, CSC1031
- *Academic year*: an academic year is a String value that follows this syntax: AY_2021_22
- *Mark*: a mark is a numerical (of double precision) value that belongs to the interval, [0, 100]
- *Academic staff members*: an array of staff members is maintained (max array length: 100)
- *Academic staff member*: an academic staff member is characterized by the following information: an ID, an array of module codes (max array length: 2), and an array of modules for academic staff members (max array length: 2).

A requirement for the information that is described above is that each bullet point should correspond to a separated object-oriented (Java) class. Moreover, the relationships between the classes are specified in the points above. You sound not include extra classes or relationships between the classes.

When the system starts up, the CLI of the system interacts with the Controller of the system to retrieve or insert information about students or academic staff members as described in the following use-cases. The Java functions that you will develop in your system should implement the description of these use-cases.

Use Cases

1. *Batch registration of end-users:* Whenever the system starts up, the CLI interacts with the Controller of the system to register a specific set of end-users (students and academic staff members) via using the suitable services of the system as follows. The set of the new end-users is hard-coded in the CLI. The information that is hard-coded in the CLI for each end-user is his/her ID and role. The CLI iteratively interacts with the Controller to add the new end-users one-by-one. For each end-user, the Controller first checks the role of the end-user. Based on the end-user's role, the Controller adds the new end-user in the corresponding array of the end-users using the ID of the new end-user. These two steps that are taken by the controller are included in a single programming function that is defined inside the controller. The Controller and all the services that participate in this use case assume that the information that is provided by the CLI is correct (no checks are required).
2. *Enrolling a student on a module:* An end-user who has the "ACADEMIC_STAFF_MEMBER" role can enroll a student on a module via using the CLI of the system. The CLI interacts with the Controller that takes the following steps. The Controller first checks if the end-user has the "ACADEMIC_STAFF_MEMBER" role. The Controller does this check via using the array of the academic staff members maintained by the system. If the end-user does not have the "ACADEMIC_STAFF_MEMBER" role, then the Controller returns back the value -1 to the CLI and the CLI prints out an error message. If the end-user has the "ACADEMIC_STAFF_MEMBER" role, then the Controller checks if the student has been previously registered in the system. The Controller does this check via using the array of the students maintained by the system. If the end-user has not been previously registered in the system, then the Controller returns back the value -1 to the CLI. In its turn, the CLI prints out an error message if the return value is -1. If the student has been previously registered in the system, then the Controller enrolls the student on the module using the ID of the student, the code of the module, and the academic year. Finally, when the student has been enrolled on the module, the Controller returns back the value 0 to the CLI and the CLI prints out the successful enrolment of the student. All the above steps that are taken by the Controller are included in a single programming function that is defined inside the controller.
3. *Inserting a new mark:* An end-user who has the "ACADEMIC_STAFF_MEMBER" role can insert the mark for a module of a student via using the CLI that interacts with the Controller as follows. The Controller first checks if the end-user has the "ACADEMIC_STAFF_MEMBER" role. The Controller does this check via using the array of the academic staff members maintained by the system. If the end-user does not have the "ACADEMIC_STAFF_MEMBER" role, then the Controller returns back the value -1 to the CLI and the CLI prints out an error message. If the end-user has the "ACADEMIC_STAFF_MEMBER" role, then the Controller checks if the student has been previously registered in the system. The Controller does this check via using the array of the students maintained by the system. If the end-user has not been previously registered in the system, then the Controller returns back the value -1 to the CLI and the CLI prints out an error message. If the student has been previously registered in the system, then the Controller checks if the student has been previously enrolled on the module. If the end-user has not been previously enrolled on the module, then the Controller returns back the value -1 to the CLI and the CLI prints

out an error message. If the student has been previously enrolled on the module, then the Controller inserts the mark for the module of the student. Finally, when the mark has been inserted for the module, the Controller returns back the value 0 to the CLI and the CLI prints out the successful insertion of the mark. All the above steps that are taken by the Controller are included in a single programming function that is defined inside the controller.

4. *Printing out the mark for a module of a student:* An end-user who has the "ACADEMIC_STAFF_MEMBER" role can print out the mark for a module of a student via using the CLI that interacts with the Controller as follows. The Controller first checks if the end-user has the "ACADEMIC_STAFF_MEMBER" role. The Controller does this check via using the array of the academic staff members maintained by the system. If the end-user does not have the "ACADEMIC_STAFF_MEMBER" role, then the Controller returns an empty String (not null value) to the CLI and the CLI prints out an error message. If the end-user has the "ACADEMIC_STAFF_MEMBER" role, then the Controller checks whether the student has been previously registered on the system. The Controller does this check via using the array of the students maintained by the system. If the student has not previously registered on the system, then the Controller returns to the CLI an empty String (not null value) and the CLI prints out an error message. If the student has been previously registered in the system, then the Controller checks if the student has been previously enrolled on the module. If the end-user has not been previously enrolled on the module, then the Controller returns to the CLI an empty String (not null value) and the CLI prints out an error message. If the student has been previously enrolled on the module, then the Controller retrieves the mark for the module of the student. The information is returned in the following format: module code, academic year, and mark. Finally, the Controller returns to the CLI this information and the CLI prints out to the screen the information. All the above steps that are taken by the Controller are included in a single programming function that is defined inside the controller.
5. *Assigning a module to an academic staff member:* An end-user who has the "ACADEMIC_STAFF_MEMBER" role can enroll a module to an academic staff member via using the CLI of the system that interacts with the Controller as follows. The Controller first checks if the end-user has the "ACADEMIC_STAFF_MEMBER" role. The Controller does this check via using the array of the academic staff members maintained by the system. If the end-user does not have the "ACADEMIC_STAFF_MEMBER" role, then the Controller returns to the CLI the value -1 and the CLI prints out an error message. If the end-user has the "ACADEMIC_STAFF_MEMBER" role, then the Controller assigns the module to the academic staff member. The new module is added in the list of the modules that have been assigned to the academic staff member. Finally, when the module has been assigned to the academic staff member, the Controller returns back the value 0 to the CLI and the CLI prints out the successful assignment of the module to the academic staff member. All the above steps that are taken by the Controller are included in a single programming function that is defined inside the controller.

Test Cases/Execution Traces of the System

You should hard-code proper data and include Java code in the CLI of your system (that contains the "main" function) so that the following lines are printed in the console during the execution of the system:

```
1. Batch registration of end-users

Enrolling the student with ID 0
Enrolling the student with ID 1

Enrolling the student with ID 2

Enrolling the student with ID 3

Enrolling the student with ID 4

Enrolling the academic staff member with ID 5

Enrolling the academic staff member with ID 6


2. Enrolling a student on a module

Error enrolling student with ID 1 by the academic staff member with ID 0

Error enrolling student with ID 7 by the academic staff member with ID 5

Successful  enrolment  of  the  student  with  ID  1  on  the  module  with  code  CSC_1022,
AY_2021_22 by the academic staff member with ID 5


3. Inserting a new mark

Error  inserting  the  mark  100  for  the  student  with  ID  1  on  the  module  with  code
CSC_1022 by the academic staff member with ID 0

Error  inserting  the  mark  100  for  the  student  with  ID  1  on  the  module  with  code
CSC_1023 by the academic staff member with ID 5

Error  inserting  the  mark  100  for  the  student  with  ID  7  on  the  module  with  code
CSC_1022 by the academic staff member with ID 5

Successful insertion of the mark 100 for the student with ID 1 on the module with code
CSC_1022 by the academic staff member with ID 5


4. Reporting the mark of a student

Error reporting the module for the student with ID 1 by the academic staff member with
ID 2

Reporting the modules for the student with ID 1 by the academic staff member with ID 5

CSC1022 AY_2021_22 100.0


5. Assigning a module to an academic staff member
```

```
Error assignment of the module CSC_1022, AY_2021_22 to the academic staff member with
ID 2

Successful assignment of the module CSC_1022, AY_2021_22 to the academic staff member
with ID 5
```

Non-Functional Requirements

1. *User interface of the system*: An end-user cannot directly interact with the system. In other words, the system should not use the standard input (e.g., Scanner Java class) to provide information to the system. All the input information to the system should be hard-coded inside the "main" function of the system. The system includes a single "main" function for all the end-user's roles. The system does not provide a graphical user interface.
2. *Service interfaces and implementation classes*: You should define for each Web service a Java interface and an implementation class with JAX-WS annotations. Each Web service should be exposed as a SOAP Web API. Apart from the Java classes that have been described in the "*Functional Requirements*" section of this document, the interface and the implementation class of each Web service are the extra classes that should be defined. The implementation class of the Java interface of each service should implement all the programming logic/steps of the service and should further implement the JAXB serialization technique. You do not need to apply the interface-identification algorithm to split the Java interface of a service into multiple interfaces.
3. *Packages of services*: You can assume that each service contains only one package.
4. *Runtime Java exceptions:* The functions of the implementation class of the interface of each service should throw exceptions. In other words, these functions should not include "try-catch" blocks. The functions of the Controller should not throw exceptions, but the Controller should include "try-catch" exception blocks.
5. *Permanent storing of data*: Each service that manages data should permanently store the data in one XML file that is automatically generated by using the JAXB serialization technique. If a service uses multiple classes to manage data, then you need to define/choose a root Java class that is the entry point of the employment of the serialization technique. If a service does not manage data but it just uses/retrieves data provided by other parts of the system for transferring them to another part of the system, then the service should store them in an XML file. Also, you should not specify in your services the directory where the XML files will bestorted. In other words, if a service of your system creates the XML file, "XX.xml", then your program should use the file name, "XX.xml", without including/declaring in your program the absolute/relative path on your hard disk where the file has been stored.

Technical Requirements for the Implementation of Web APIs

You must use the following (versions of) platforms, technologies, tools, and libraries that have been used in the lectures and the practicals for developing your system:

- Eclipse IDE JEE 2020-12
- jdk1.7.0_80/jdk-8.0.292.10-hotspot
- Dynamic Web Module v2.5 (for Dynamic Web Projects within Eclipse)
- Tomcat v8.0 server
- JAX-WS v2.2.6 (it should be specified in the dependencies of the POM files)
- JAXB
- XML RPC v1.1.2 (it should be specified in the dependencies of POM files)

- XML resolver v20050927 (it should be specified in the dependencies of POM files)
- Axis v1.4 (it should be specified in the dependencies of POM files)
- Maven war plugin v3.2.3 (it should be specified in the plugins of POM files)
- wsimport tool of jdk1.7.0_80/jdk-8.0.292.10-hotspot.

Troubleshooting on (Re-)Deploying Services on Server

- When you develop a new service in Eclipse, you need to follow the steps that are presented at these slides one-by-one (without omitting any step). You *should not import/add* an existing project into Eclipse and to make changes on it for developing your service. On the contrary, you should develop a new Eclipse project from scratch that corresponds to a service by following the steps described above.
- After deploying a service/running on the server, if a deployment error is caught by the Tomcat server, it's better to undeploy the service (remove the service from the Tomcat) by right clicking on the server and choosing the option "Add and remove…". Then, it's better to right click on the server again, choose the option "clean", and then stop the execution of the server. Finally, try to run your service on the server from scratch.
- If you make changes in the interface of a service or in the signature/first line of the functions of the implementation class of the service, you need to undeploy the service (remove the service from the Tomcat) by right clicking on the server and choosing the option "Add and remove…". Then, it's better to right click on the server again, choose the option "clean", and then stop the execution of the server. Finally, try to run your service on the server from scratch. If you do not make changes in the interface of a service or in the signature/first line of the functions of the implementation class of the service, but you make changes inside the functions of the implementation class of the service (or any other class of the service), then you do not need to undeploy the service. Moreover, the Tomcat is by default setup to automatically get the updates in the implementation class without needing you to do anything.
- When you develop a client of a service in Eclipse, you need to follow the steps that are presented at these slides one-by-one (without omitting any step). After developing the client, if you have made changes in the interface of the service, then you need to delete the files that were generated by the "wsimport" tool in the client side and to repeat the steps for the development of the client from scratch (executing the "wsimport" tool again). If you have made changes inside the functions of the implementation class of the service (or in any other class of the service), without changing the signature/first line of the functions of the implementation class of the service or the interface of the service, then you do not need to delete the generated files in the client side and to execute the "wsimport" tool from scratch.