CSCI398

Introduction to Enterprise Computing

Assignment 1 – 10 marks

"Distributed Systems Basics / Rich Client"

Spring Session 2013

You should complete Exercise 1 before attempting this assignment.

This assignment involves the implementation and correct deployment of a client-server system implemented using Java-RMI with the client supporting a rich graphical interface.

The application is in part a reprise of the "Nonsuch University" web-application created last session in CSCI399.

On the server side, the two applications are similar. There is a "server object". In the web application, the "server object" was a Zend application that used MySQL via the Zend PDO object-relational mapping system. The Zend application had multiple "Controllers" each handling a single aspect of the overall application. Now there will be a single multi-functional "server object" – an instance of a class extending the RMI class "UnicastRemoteObject" that uses the JPA object-relational mapping system to access the MySQL data base (you can use JDBC if you prefer). Actually, the server handles only persistence; data entry and editing actions are in the client.

The applications differ greatly on the client-side. The web-application created in CSCI399 utilized browser technology for data entry forms and report display. The CSCI398 application involves a "professional client" (or "rich client interface").

Generally, the use of web technology is preferable; but, sometimes, it is necessary to create more sophisticated clients and deploy applications using non-web technologies. Further, it is sometimes useful to have both – a web interface that handles customers, and a "rich client" interface for in house use. (Note, CSCI398 assignment 4 involves a further variation on this theme with the same application being recreated with EJB technology – *so keep all the code and data tables that you construct for this assignment*.)

Aims

The aim of this assignment is for you to become familiar with the basics of programming for distributed object systems and the construction of elaborate graphical "rich client" interfaces. You will learn about remote interfaces, server implementation, deployment, and support services such as a naming service used by clients seeking server instances. You will practice exploiting GUI build tools to create a client with an elaborate interface.

Objectives

On completion of this assignment, you should be able to:

- Define Java RMI remote interfaces;
- Create server side classes that implement these interfaces;
- Utilize JPA for data persistence;
- Implement client programs that utilize the operations supported by the server;
- Build sophisticated GUI interfaces for a client.

You will find that the code is very similar to supplied examples. This is really a training exercise in which you become familiar with distributed object computing and in which you start to master the tasks involved in deploying such moderately elaborate software systems.

Tasks

Nonsuch University Subject Management System

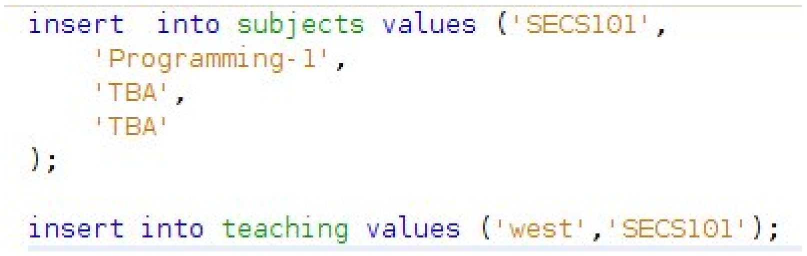
Students who completed CSCI399 in Autumn Session 2013 will remember Nonsuch University application and its requirement for a web-site that manages records for subjects offered at an imaginary university. That application was built in stages in CSCI399 assignments 4, 5, and 6. The web-site allowed visitors to view subjects on offer and allowed lecturers to edit subject descriptions, enrol students, add marks, and view marks. It is some of the "lecturer" aspects that are now to be implemented in "rich client" style while the server side is implemented as a "remote object".

If you did not take CSCI399 in autumn 2013, don't panic. You simply need to define tables for a number of record types – "staff", "student", "subject", "task", "teaching", "enrolment", and "mark". Possible MySQL-style definitions for some of these tables are in the CSCI399 assignment.

The new application is to work with the same database tables as already created (or recreated if you deleted them). The professional client style application is to provide the following functionality:

- Name and password controlled login
  - Creation of records for "lecturers" in the staff table is outside the scope of this application – that aspect will be handled using a standard database client.
  - The client application should work with the server to enforce login controls; the client will need to keep a record of the logged in lecturer
- Edit the subject description

    - Creation of “subject” records and “teaching” records is outside the scope of this application – that aspect will be handled using a standard database client:



```
insert  into subjects values ('SECS101',
     'Programming-1',
     'TBA',
     'TBA'
);

insert into teaching values ('west','SECS101');
```

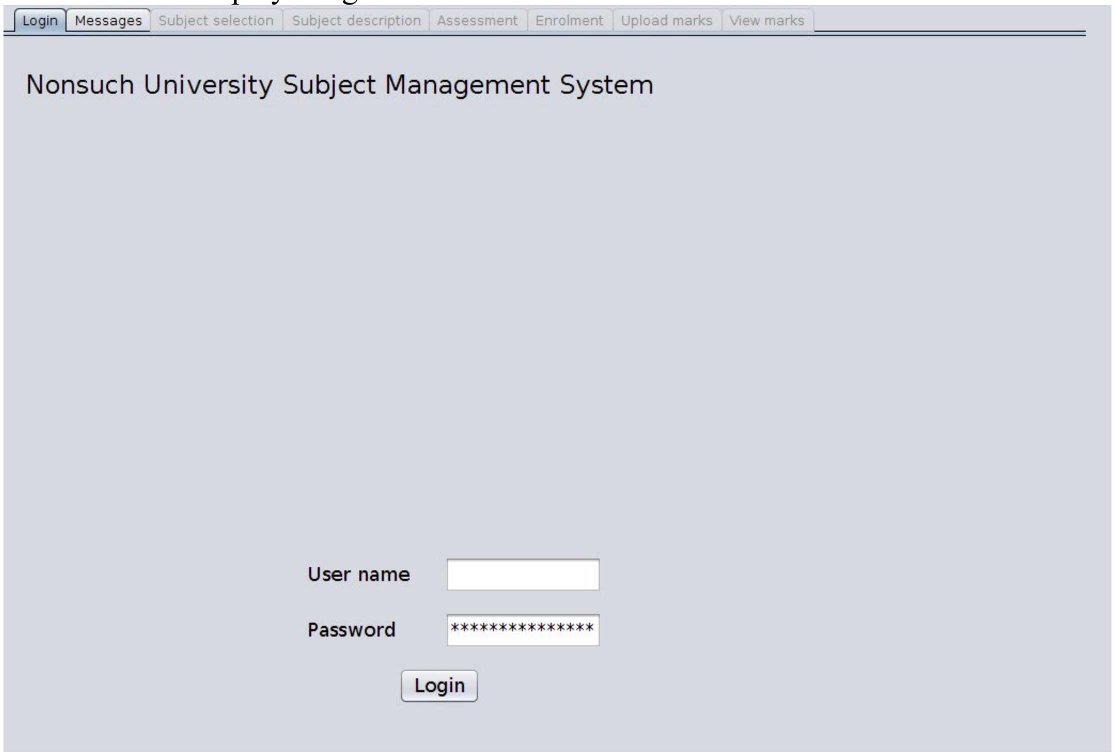
    - The “content” and “objectives” fields in a subjects record can be edited by the lecturer registered in the “teaching” table. (They can be edited at any time; Nonsuch university doesn’t object to lecturers redefining subjects after they have started.)
- Enrol students.
    - The application should have a “students” table with student ids (e.g. xyz123, qaz444, ead432) and names. Once again, this table is created and populated using a standard database client.
    - The application should allow the lecturer to enrol and withdraw students and view a list of those currently enrolled in any subject that he/she teaches. The lecturer only enters student ids (the application should check that these do correspond to entries in the students table maintained by Nonsuch University).
      If a student is withdrawn from a subject, all marks records for tasks already completed by that student are to be removed.
- Edit tasks.
    - The application should allow a logged in lecturer to edit tasks associated with any of the subjects that he/she teaches. Tasks have identifiers (that must be unique within a subject – must be Essay1, Essay2 etc), mark values (integer), and a short description.
    - Nonsuch university’s management policies are fairly lax – a staff member can change the mark value and description of a task at any time. If a task proved too hard and the students all got low marks, the lecturer can delete that task (all record of that task including any marks recorded for students will be removed).
- Record marks.
    - The application should allow a logged in lecturer to process files with marks awarded to students who have completed a task. Fractional marks are allowed. If a student resubmits work, the more recently allocated mark overwrites any existing record. The client application checks the data (student ids correspond to enrolled students, marks in the range allowed for the task), and if valid sends the data to the server for recording in the database.
- View marks.
    - The application should allow a logged in lecturer to view all marks awarded to students enrolled in a subject.

“Storyboard”/walk through

The following screen shots illustrate my version of the application. Yours must provide similar functionality – but you can change the appearance of the client, alter the interaction mechanisms, and slightly modify the processing if you so wish.

Server and then client applications can be started from within NetBeans.

The client will display a login form:



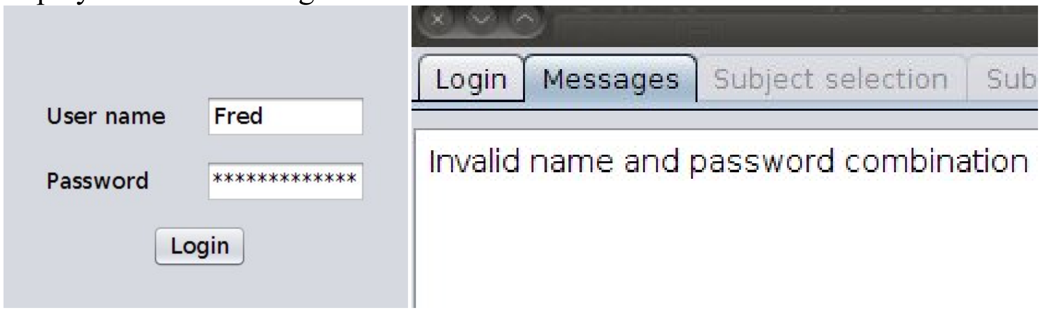
Nonsuch University Subject Management System

User name

Password ***************

(My implementation uses a “tabbed pane” as the basis of the interface. The client starts with the “login” tab displayed; other tabs are disabled so that they cannot be selected by the user.)

If a user enters an invalid name and password combination, the client application displays an error message:

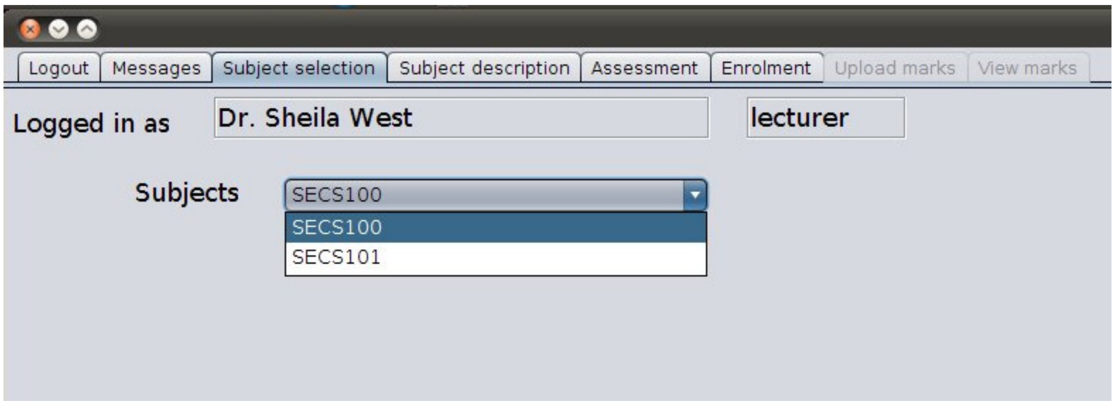


User name Fred

Password *************

Invalid name and password combination

Successful login will result in the display of the subjects taught by the logged in lecturer.

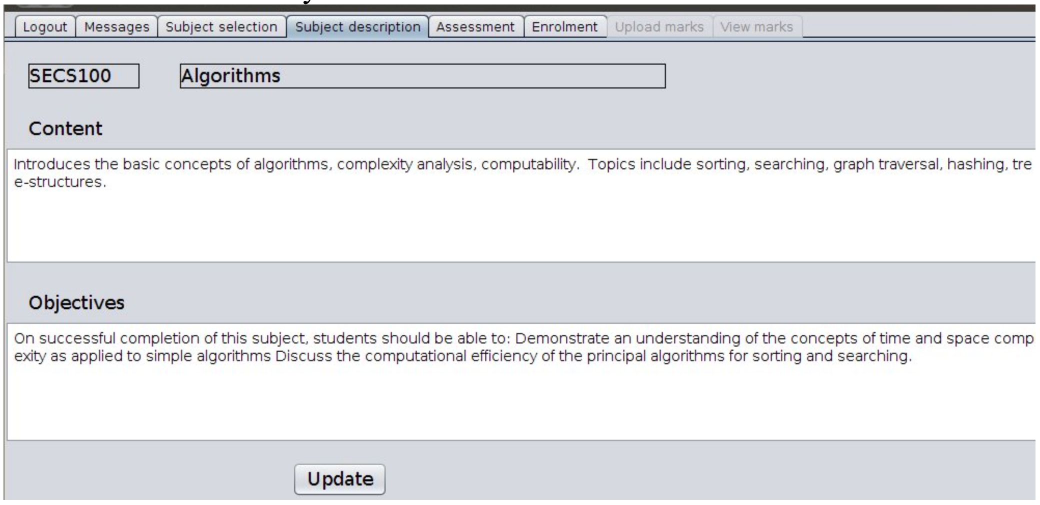


Logged in as Dr. Sheila West lecturer

Subjects SECS100

SECS100
SECS101

The lecturer can view the currently defined contents and objectives for their subjects, and can edit these if they wish:



SECS100 Algorithms

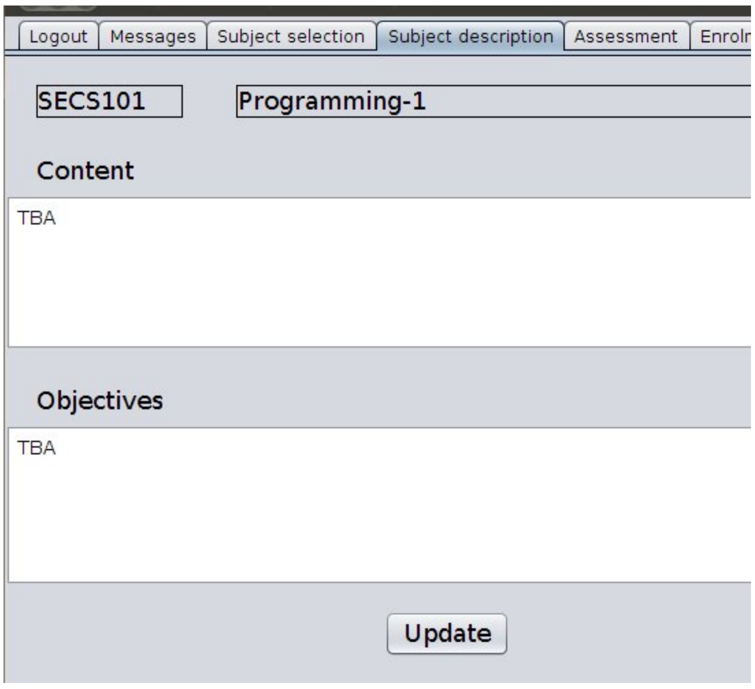
Content

Introduces the basic concepts of algorithms, complexity analysis, computability. Topics include sorting, searching, graph traversal, hashing, tre e-structures.

Objectives

On successful completion of this subject, students should be able to: Demonstrate an understanding of the concepts of time and space comp exity as applied to simple algorithms Discuss the computational efficiency of the principal algorithms for sorting and searching.

SECS101 doesn't have any real content and objectives defined – it's a new record in the database.



SECS101 Programming-1

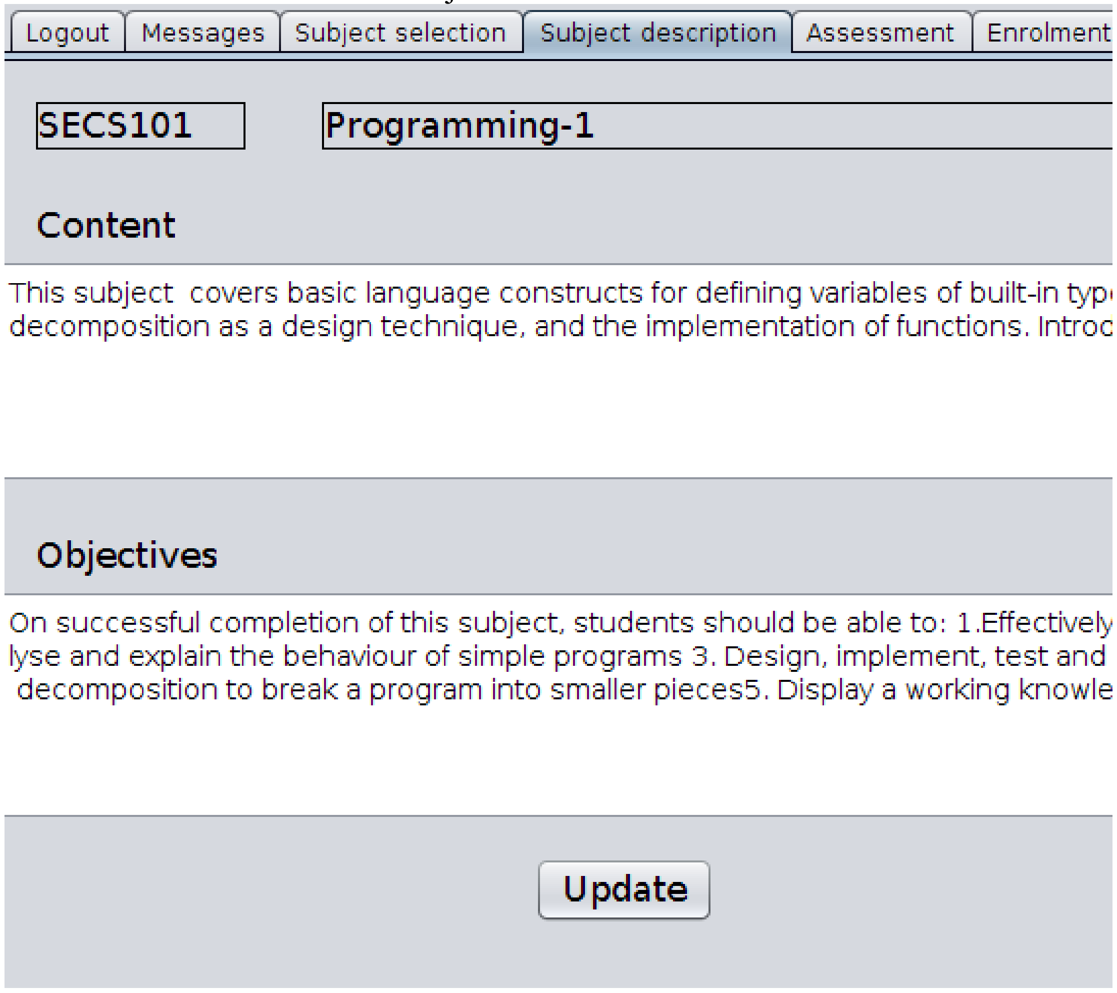
Content

TBA

Objectives

TBA

Lecturer adds some content and objectives:



SECS101 Programming-1

Content

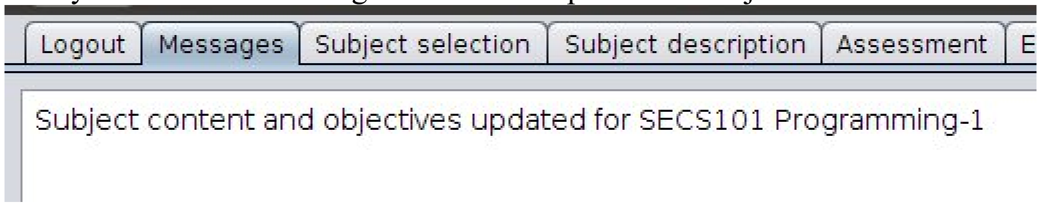
This subject covers basic language constructs for defining variables of built-in typ
decomposition as a design technique, and the implementation of functions. Introc

Objectives

On successful completion of this subject, students should be able to: 1.Effectively
lyse and explain the behaviour of simple programs 3. Design, implement, test and
decomposition to break a program into smaller pieces5. Display a working knowle

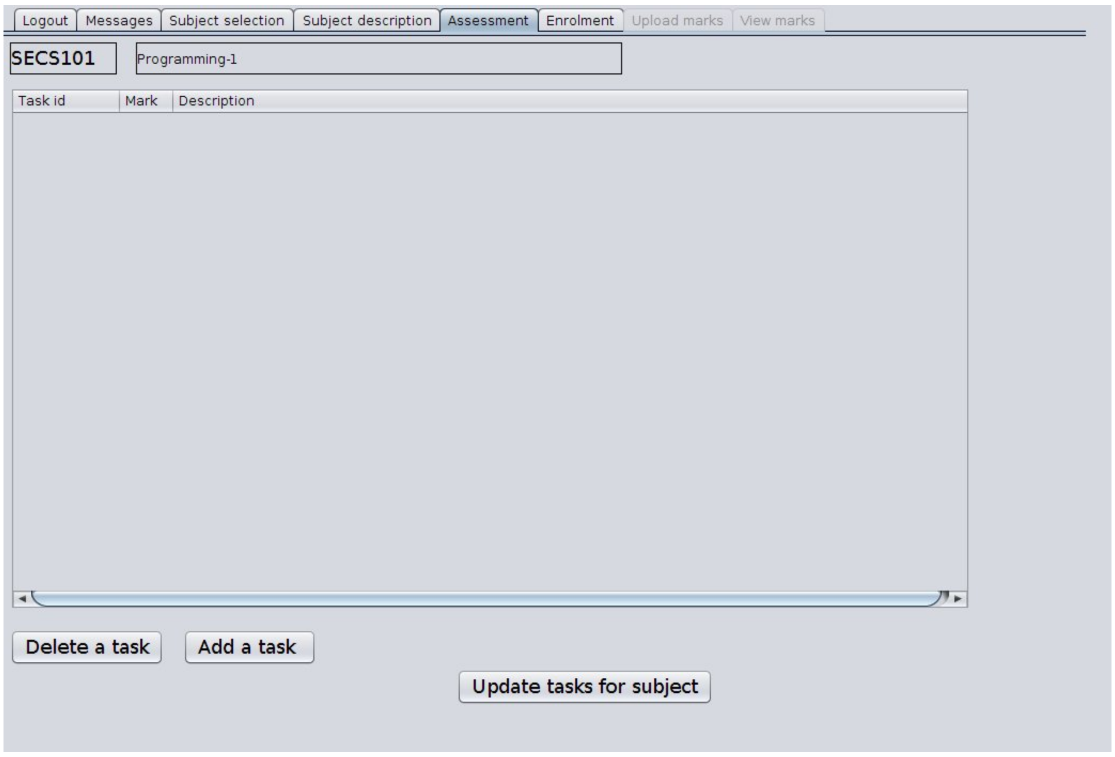
Lecturer updates the subject.

The system will acknowledge a successful update of a subject:



Subject content and objectives updated for SECS101 Programming-1

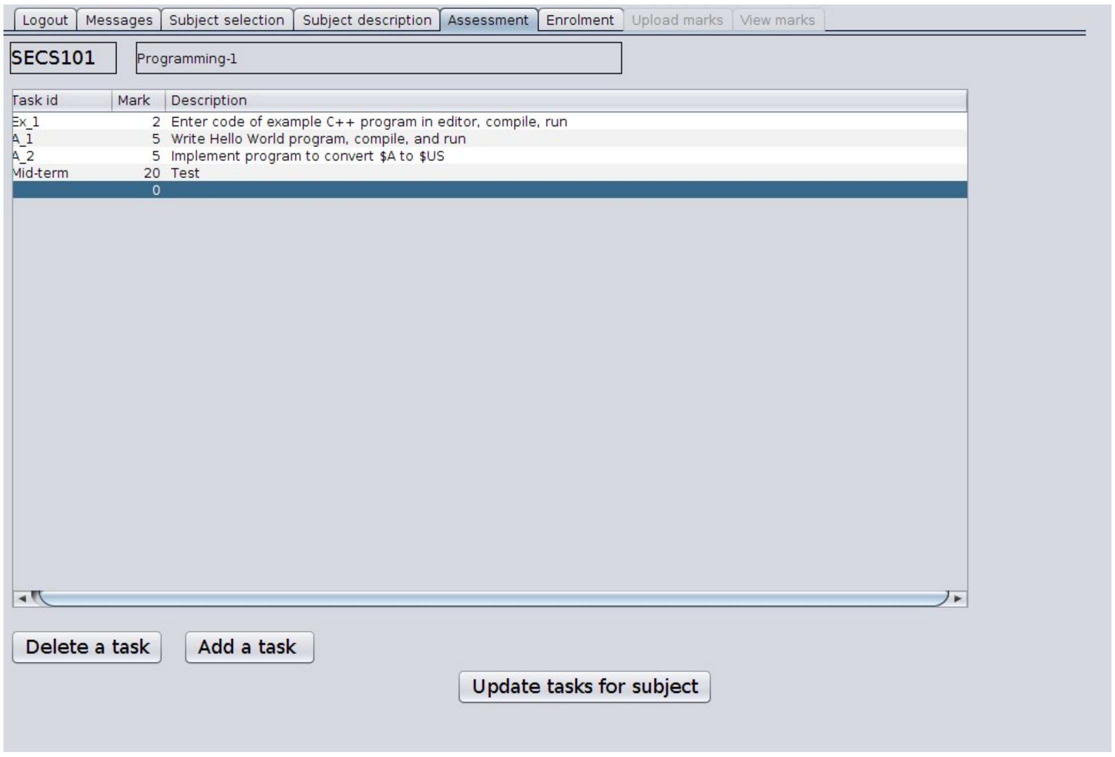
The lecturer can define assessment tasks. A task will have an identifier (unique within the subject – the system generates an overall unique id), a mark (0<mark<=100) and a description. Task details should be displayed in a table. There will be an "Add Task" option. An existing task can be selected and deleted. Changes made on the client are only finalised in the server side database when the "Update tasks for subject" action button is used.



SECS101 Programming-1

Task id	Mark	Description

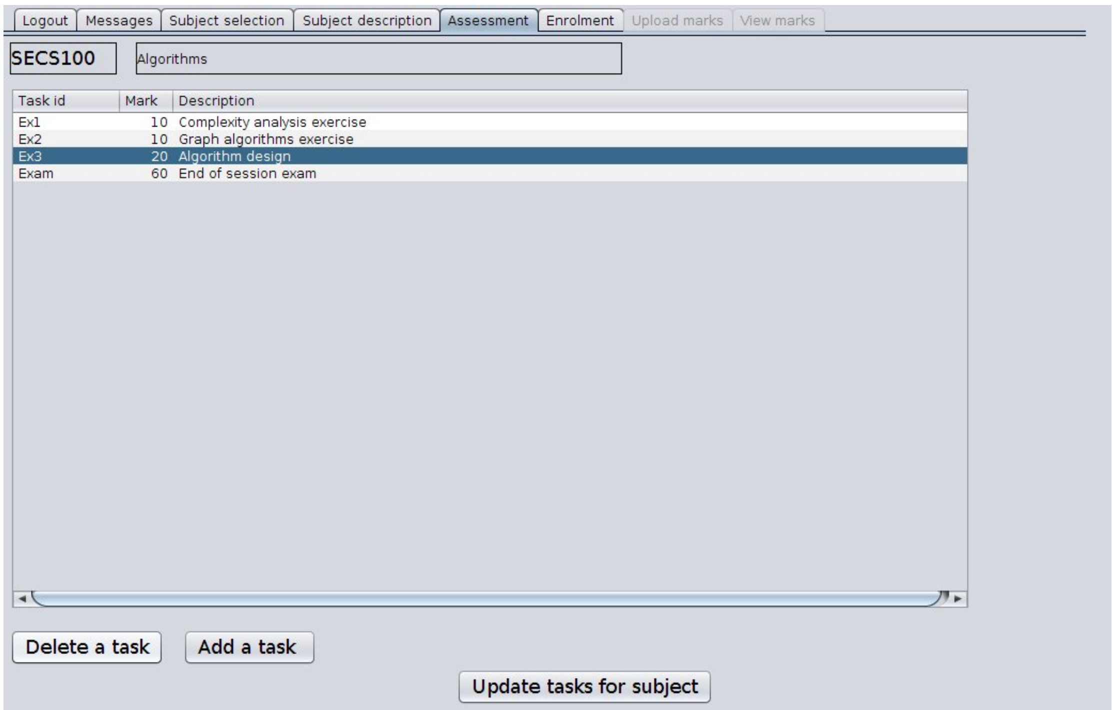
Added a few tasks:



SECS101 Programming-1

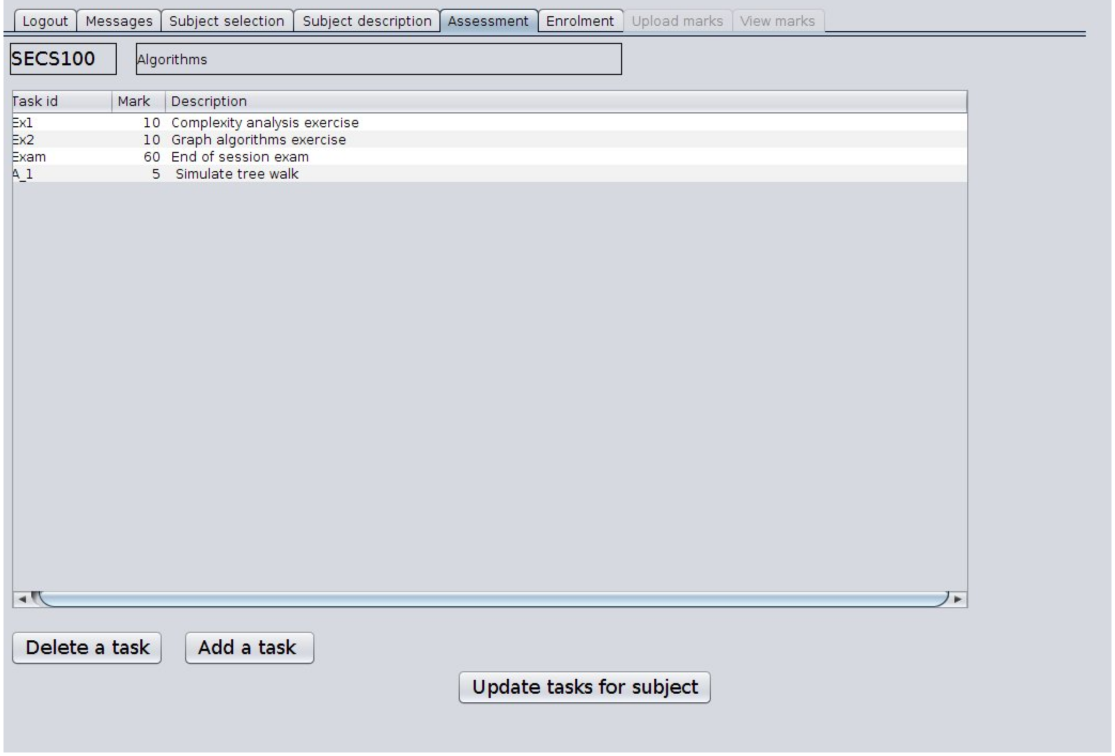
Task id	Mark	Description
Ex_1	2	Enter code of example C++ program in editor, compile, run
A_1	5	Write Hello World program, compile, and run
A_2	5	Implement program to convert $A to $US
Mid-term	20	Test
	0	

Existing records can be edited or removed, and additional records can be added at a later date.



SECS100 Algorithms

Task id	Mark	Description
Ex1	10	Complexity analysis exercise
Ex2	10	Graph algorithms exercise
Ex3	20	Algorithm design
Exam	60	End of session exam

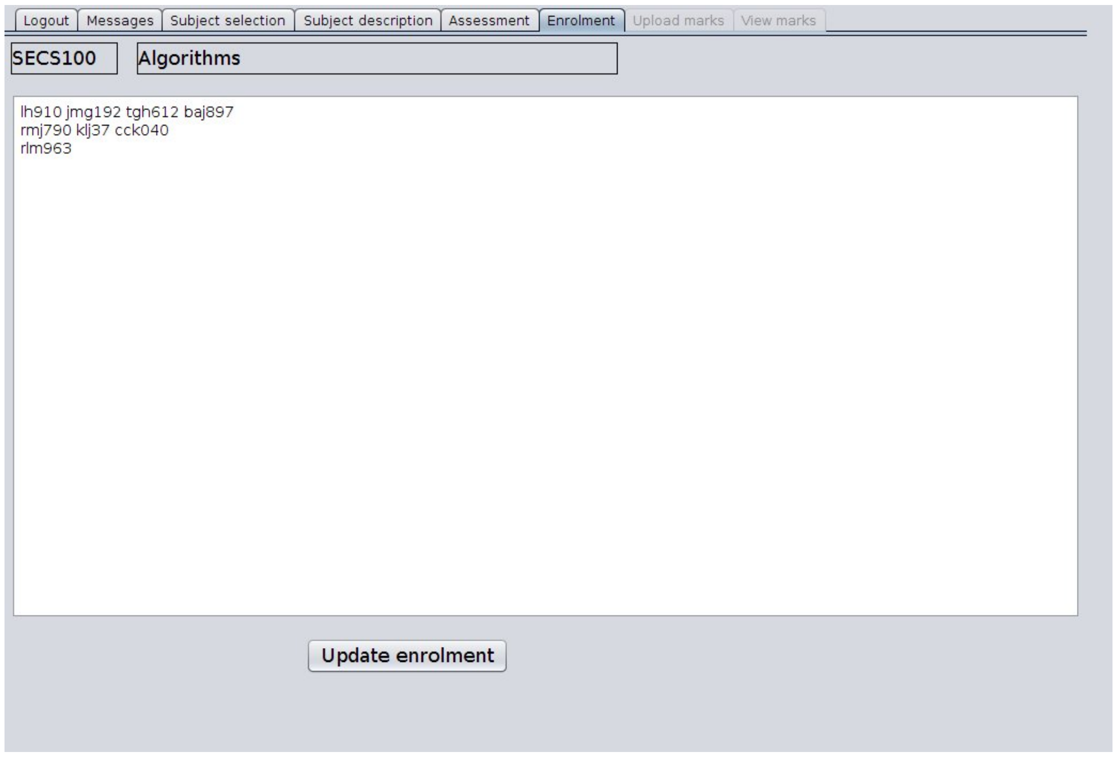


SECS100 Algorithms

Task id	Mark	Description
Ex1	10	Complexity analysis exercise
Ex2	10	Graph algorithms exercise
Exam	60	End of session exam
A_1	5	Simulate tree walk

Nothing really changed until “update tasks for subject” used.

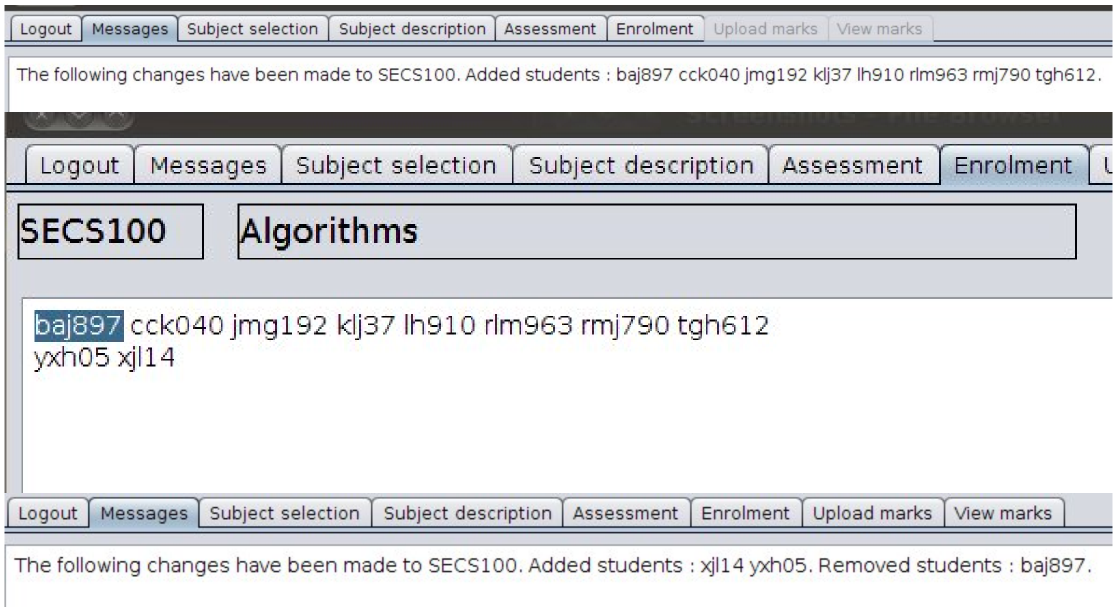
Students can be enrolled or withdrawn:



SECS100 Algorithms

```
lh910 jmg192 tgh612 baj897
rmj790 klj37 cck040
rlm963
```

An update of the enrolment will result in a report identifying those students actually added and those withdrawn:



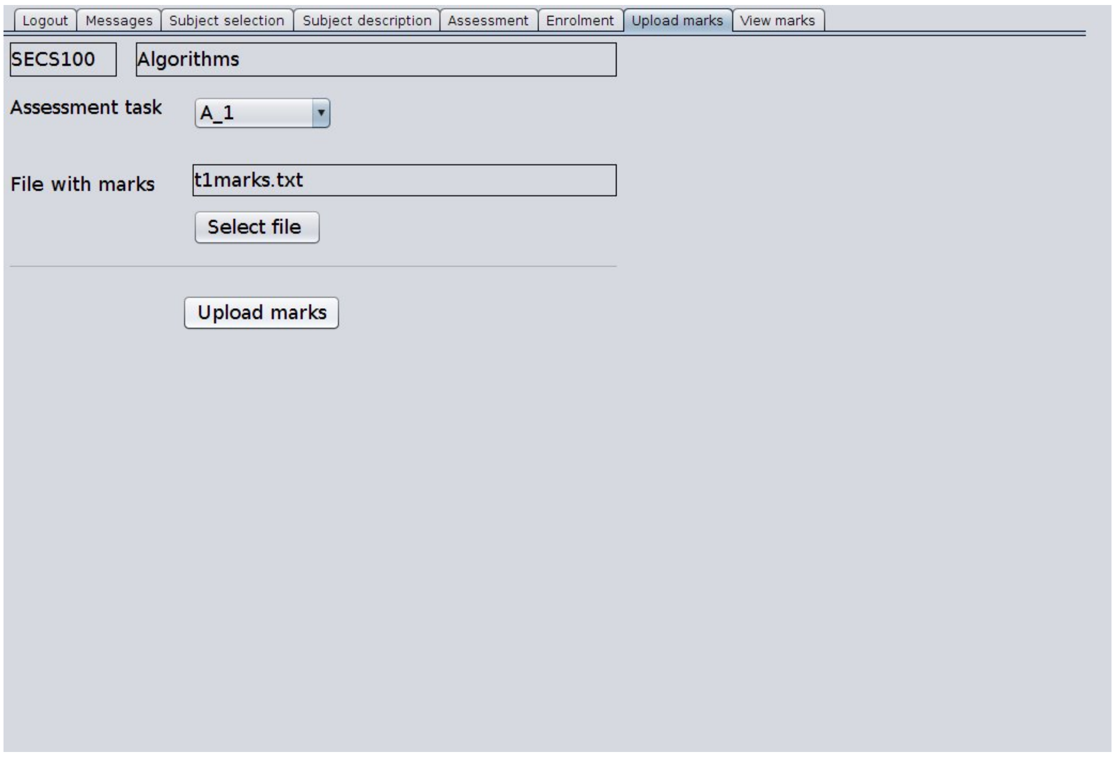
The following changes have been made to SECS100. Added students : baj897 cck040 jmg192 klj37 lh910 rlm963 rmj790 tgh612.

SECS100 Algorithms

```
baj897 cck040 jmg192 klj37 lh910 rlm963 rmj790 tgh612
yxh05 xjl14
```

The following changes have been made to SECS100. Added students : xjl14 yxh05. Removed students : baj897.

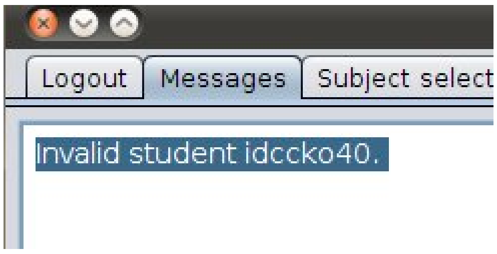
Marks are read from a text file that contains student identifiers and marks (fractional marks are permitted). The client application validates the student identifier (it must be an enrolled student), and checks that the mark is within the range of marks allowed for the task:



SECS100 Algorithms

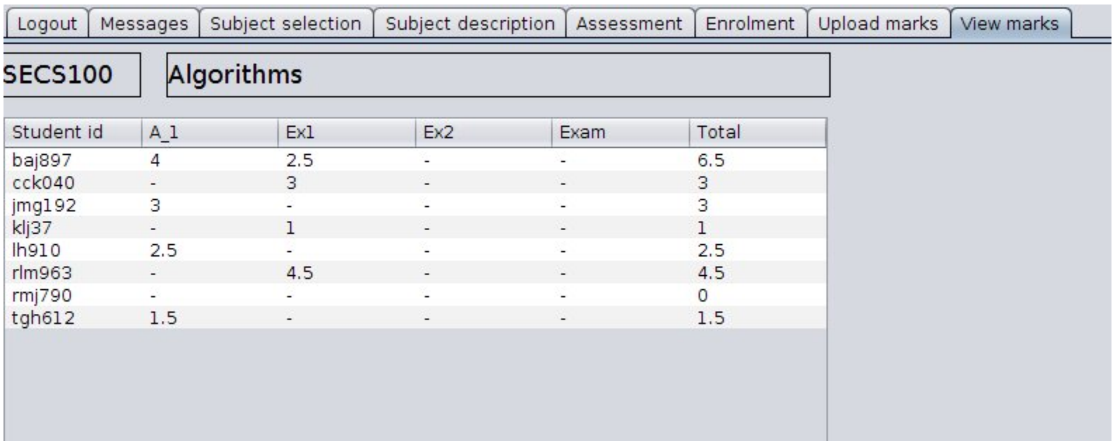
Assessment task: A_1

File with marks: t1marks.txt



Invalid student idccko40.

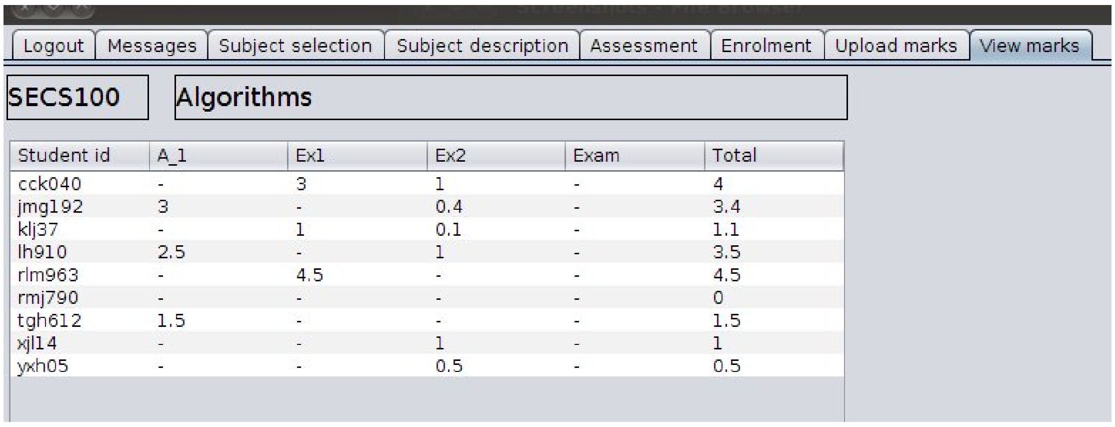
The marks can be viewed:



SECS100 Algorithms

Student id	A_1	Ex1	Ex2	Exam	Total
baj897	4	2.5	-	-	6.5
cck040	-	3	-	-	3
jmg192	3	-	-	-	3
klj37	-	1	-	-	1
lh910	2.5	-	-	-	2.5
rlm963	-	4.5	-	-	4.5
rmj790	-	-	-	-	0
tgh612	1.5	-	-	-	1.5

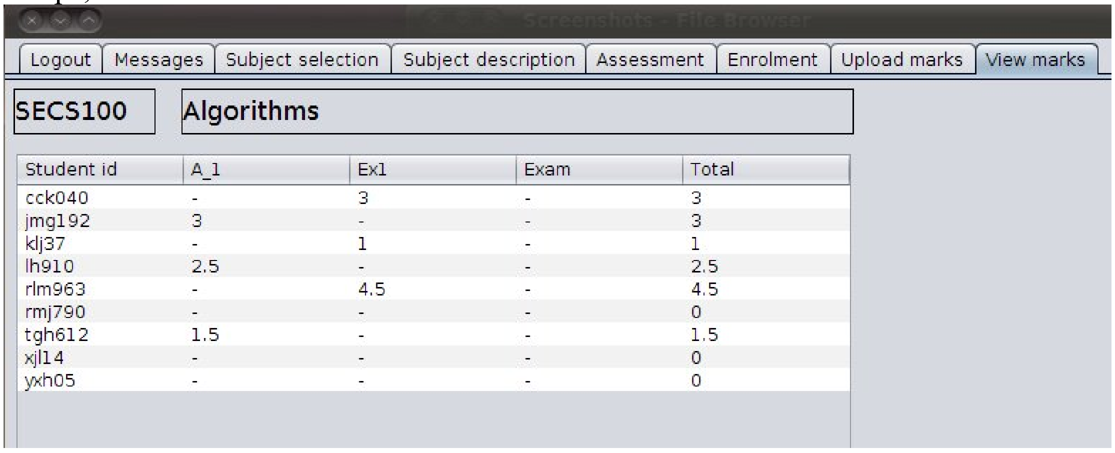
More marks added



SECS100 Algorithms

Student id	A_1	Ex1	Ex2	Exam	Total
cck040	-	3	1	-	4
jmg192	3	-	0.4	-	3.4
klj37	-	1	0.1	-	1.1
lh910	2.5	-	1	-	3.5
rlm963	-	4.5	-	-	4.5
rmj790	-	-	-	-	0
tgh612	1.5	-	-	-	1.5
xjl14	-	-	1	-	1
yxh05	-	-	0.5	-	0.5

Ooops, Ex2 was too hard – delete it

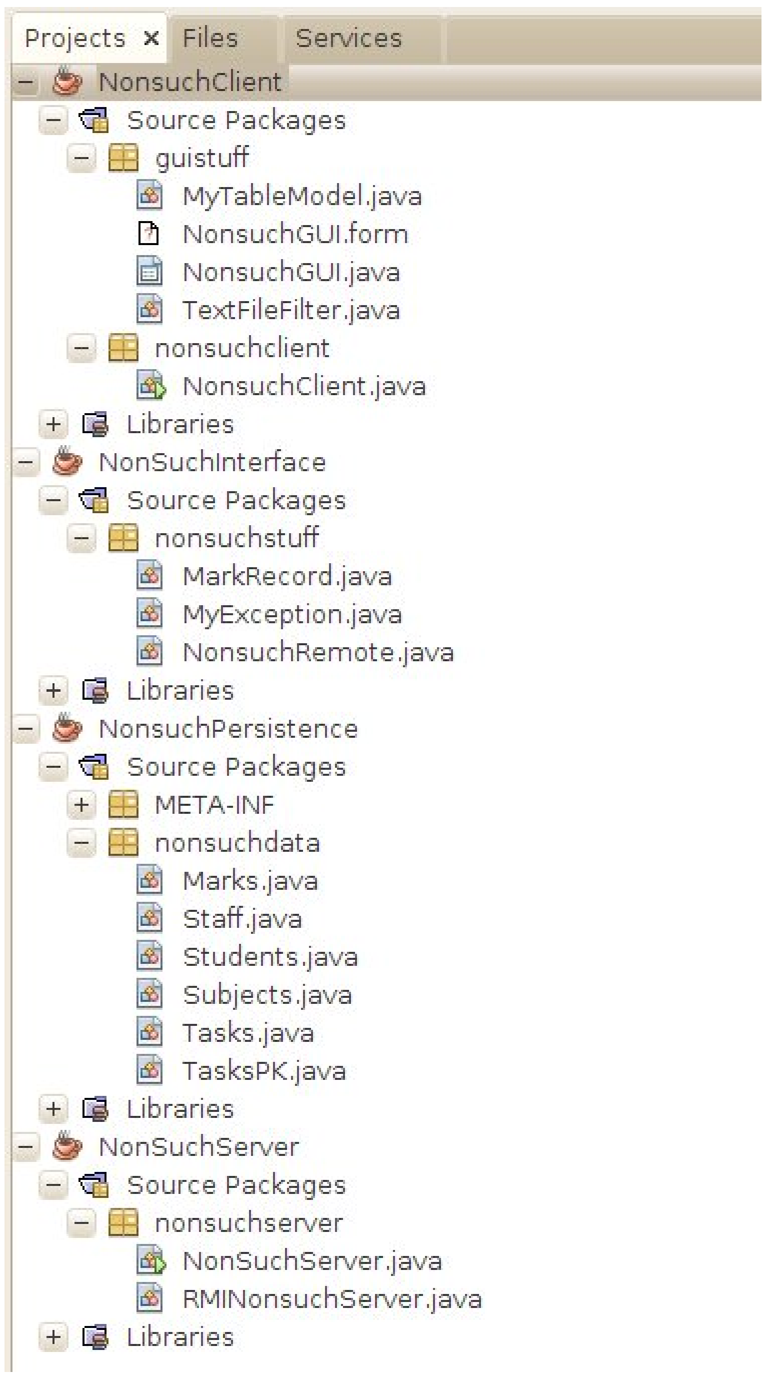


SECS100 Algorithms

Student id	A_1	Ex1	Exam	Total
cck040	-	3	-	3
jmg192	3	-	-	3
klj37	-	1	-	1
lh910	2.5	-	-	2.5
rlm963	-	4.5	-	4.5
rmj790	-	-	-	0
tgh612	1.5	-	-	1.5
xjl14	-	-	-	0
yxh05	-	-	-	0

Application

The application is to be implemented using Java RMI. It will be composed from four Netbeans projects:



```
Projects ×   Files   Services
− NonsuchClient
  − Source Packages
    − guistuff
        MyTableModel.java
        NonsuchGUI.form
        NonsuchGUI.java
        TextFileFilter.java
    − nonsuchclient
        NonsuchClient.java
  + Libraries
− NonSuchInterface
  − Source Packages
    − nonsuchstuff
        MarkRecord.java
        MyException.java
        NonsuchRemote.java
  + Libraries
− NonsuchPersistence
  − Source Packages
    + META-INF
    − nonsuchdata
        Marks.java
        Staff.java
        Students.java
        Subjects.java
        Tasks.java
        TasksPK.java
  + Libraries
− NonSuchServer
  − Source Packages
    − nonsuchserver
        NonSuchServer.java
        RMINonsuchServer.java
  + Libraries
```

- NonsuchPersistence
  - This project is a Java *library* project. It contains the entity classes auto-generated from the database tables. (Minor edits, e.g. provision of a useful toString() member, can be done after auto-generation.)
- NonSuchInterface

    - This project is also a Java *library* project. It contains the definition of the "remote interface" that defines the service (along with any auxiliary exception classes and "structs" that might be required).
- NonSuchServer
    - This project is a Java *application* project. It contains the pretty much standardized Java-RMI server class, and the class that extends UnicastRemoteObject and implements the interface defined in the NonSuchInterface project.
      The server will use JPA for data persistence. JPA generated entity classes are "serializable" and can be used as arguments to and results from RMI remote system calls. (You can use JDBC if you really want – you will have to define more serializable structs for data transfer; put those struct definitions in the NonSuchInterface library.)
- NonSuchClient
    - This project is a Java *application* project. It contains the pretty much standardized Java-RMI client class that performs the lookup and connection to the server, and which then creates and starts the graphical interface. The project also contains the classes that make up this user interface.

The server project contains the more or less standard RMI server code (this simple demonstration can use naïve deployment with an integral nameservice component rather than a full deployment with separate nameservice process). The implemenation class should support multiple concurrent clients – opening and closing EntityManager connections for each operation.

You will need to allow for JPA's habit of caching data. If new tasks are added to a subject, or deleted from a subject, the changes may not show in a cached record. EntityManager refresh operations (that guarantee synchronization of cached and persisted data) should be employed where appropriate.

If you use JPA and tables with foreign key constraints you could end up loading the entire contents of the database into the server and then try serializing these data and sending them to the client. Don't do it. Use "lazy loading" of collections. Only send data that the client really needs.

Passwords in the staff data table are supposed to have been encrypted using MySQL's MD5 function. There is no Java function that simply returns an MD5 string. But you can use an appropriate MessageDigest object to create an MD5 hash code as a byte[]; and then you can hex-encode the resulting bytes. (I'm sure Google can help if you have problems.)

Report

Your assessment will be based on a report (A1.pdf) that you compose. This report should be submitted as a PDF file.

The report should have an index, an overview, a walkthrough demonstration of all aspects of the application, and then a technical section presenting the code.

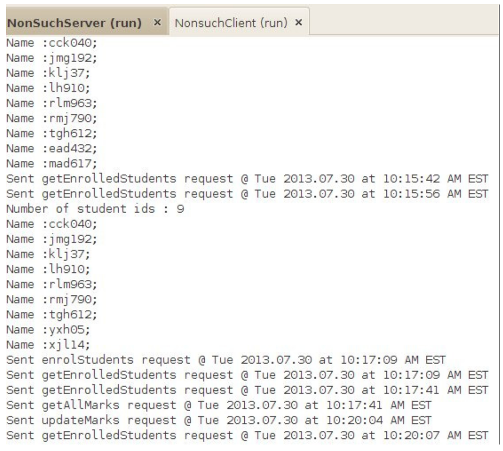
The technical section should include embedded code with additional annotations and comments. The comments should explain details of the code where you are handling major tasks such as object registration.

The report:

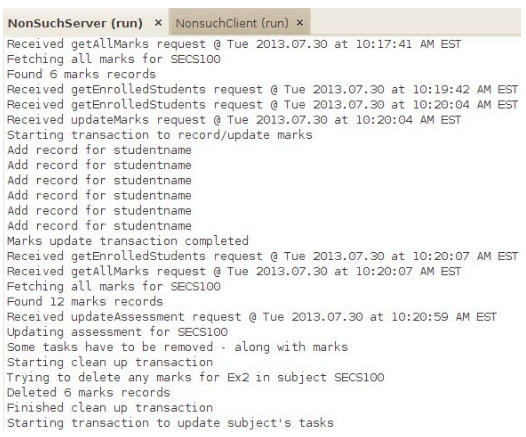
1. *2 marks*
   Evidence that all aspects have been successfully implemented.

   Screenshots. Data table contents.

   Use tracer logs on client and server as part of this evidence:



```
Name :cck040;
Name :jmg192;
Name :klj37;
Name :lh910;
Name :rlm963;
Name :rmj790;
Name :tgh612;
Name :ead432;
Name :mad617;
Sent getEnrolledStudents request @ Tue 2013.07.30 at 10:15:42 AM EST
Sent getEnrolledStudents request @ Tue 2013.07.30 at 10:15:56 AM EST
Number of student ids : 9
Name :cck040;
Name :jmg192;
Name :klj37;
Name :lh910;
Name :rlm963;
Name :rmj790;
Name :tgh612;
Name :yxh05;
Name :xjl14;
Sent enrolStudents request @ Tue 2013.07.30 at 10:17:09 AM EST
Sent getEnrolledStudents request @ Tue 2013.07.30 at 10:17:09 AM EST
Sent getEnrolledStudents request @ Tue 2013.07.30 at 10:17:41 AM EST
Sent getAllMarks request @ Tue 2013.07.30 at 10:17:41 AM EST
Sent updateMarks request @ Tue 2013.07.30 at 10:20:04 AM EST
Sent getEnrolledStudents request @ Tue 2013.07.30 at 10:20:07 AM EST
```



```
Received getAllMarks request @ Tue 2013.07.30 at 10:17:41 AM EST
Fetching all marks for SECS100
Found 6 marks records
Received getEnrolledStudents request @ Tue 2013.07.30 at 10:19:42 AM EST
Received getEnrolledStudents request @ Tue 2013.07.30 at 10:20:04 AM EST
Received updateMarks request @ Tue 2013.07.30 at 10:20:04 AM EST
Starting transaction to record/update marks
Add record for studentname
Add record for studentname
Add record for studentname
Add record for studentname
Add record for studentname
Add record for studentname
Marks update transaction completed
Received getEnrolledStudents request @ Tue 2013.07.30 at 10:20:07 AM EST
Received getAllMarks request @ Tue 2013.07.30 at 10:20:07 AM EST
Fetching all marks for SECS100
Found 12 marks records
Received updateAssessment request @ Tue 2013.07.30 at 10:20:59 AM EST
Updating assessment for SECS100
Some tasks have to be removed - along with marks
Starting clean up transaction
Trying to delete any marks for Ex2 in subject SECS100
Deleted 6 marks records
Finished clean up transaction
Starting transaction to update subject's tasks
```

2. *2 marks*
   Appearance and functionality of the "rich client".

3. *5 marks*
   Coding details for server and client applications

   This section of the report should present the implementation of your server and client applications. You will need commented listings of your code (remember that NetBeans can give you formatted code listing via the "Print as HTML" option).

   The code presenting the client should be supplemented with comments explaining each section. Code involving remote invocations of server operations should be highlighted.

   Do not include code of functions and classes taken from the Internet. Do NOT include auto-generated code that lays out the contents of JPanel classes and other GUI elements. The code needed is simply that which you wrote to handle events and the business data processing.

4. *1 mark* Presentation.
   One mark covers "presentation" - this includes appearance and organization of the report, correct grammar and spelling.

Marks will also be deducted for bad programming style.

Submission

The due date for submission will be announced in lectures; provisionally it is Friday August 30th (end of week 5).

The assignment is to be submitted using `turnin`; you must be logged in on *banshee* to use turnin, turnin does not send you email and gives only a terse acknowledgment.

```
turnin -c csci398 -a 1 A1.pdf
```

or, for late submission,

```
turnin -c csci398 -a 1late A1.pdf
```

You do not need to apply for late submission, you simply use the facility. Typically, if an assignment is due on a Friday night, the late submission facility will close at midnight on the following Tuesday. Submissions are not accepted after this time; nor are they accepted via email. Penalties apply to late submissions. Apply via SOLS if you have a medical condition justifying a late submission; if you application is approved, the late penalty will not apply (you cannot extend the submission date beyond that allowed for late submission).

Penalties apply to submission involving corrupted, unreadable files.

Notes

Don’t download the entire database into the client!

You might transfer a “Subject” record with its code, title, content, and objectives but not the collection of students enrolled. (Actually, don’t ever need Student records in the client – simply a Collection<String> with ids of enrolled students.)

Your database is going to have “many-to-many” relations. There are many subjects; there are many students. Each individual student is (or could be) enrolled in several subjects. Each subject has many students enrolled. You need to be careful when using JPA objects. If you remove a student from a subject (using objects) you should remove the student from the subject’s “collection of students” and remove the subject from the student’s “collection of subjects”.

If you have many foreign key constraints, the code generator that creates the JPA classes from the table definitions may generate invalid JPA annotations.

Sometimes, surrogate primary keys are acceptable if they simplify the JPA mappings!