Final Report: SchoolBook

Team 1: Omkar Rege, Bing Shi, Lam Tran

Table of Contents

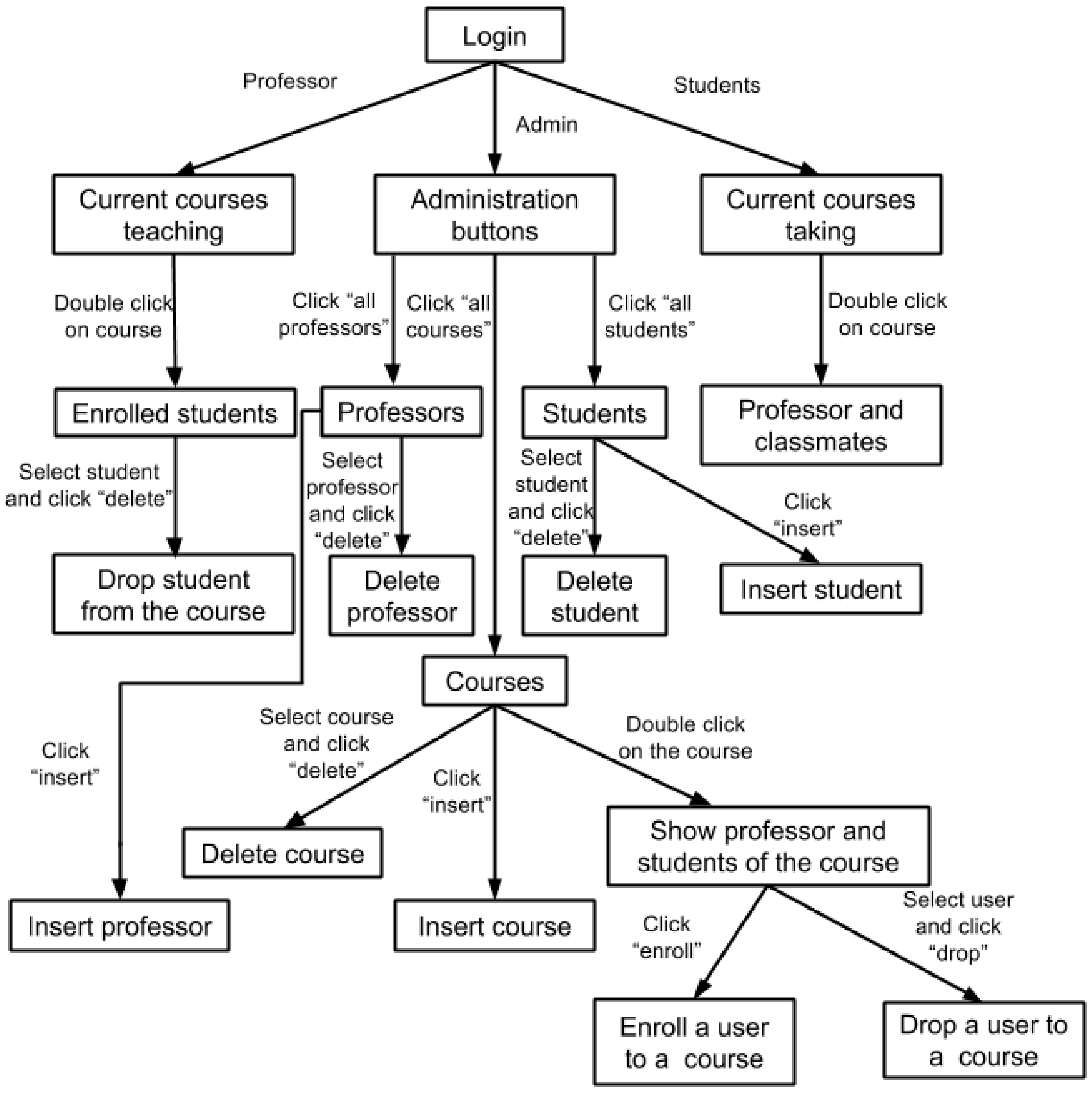
Features and functionalities
This application is designed for school administrative management. It supports user(professor and student) registration, course registration, enroll/ drop classes, view courses'/ user's' information, upload/ download files. It has three main access roles, the administrator, professors and students. Each user role will have different accesses and restrictions.

System administrator has full access of the user databases and course information. He/ she can view all users' and courses' information except user password. He/ she is also able to add/ remove users/ courses and can assign professor to specific course and can enroll/ drop certain student from certain courses. Also, he/ she would have access to upload/ download files.

Professors will have full access of course information they teach. They can view course they are teaching. They can view students' contact information who are enrolled to their course, as long as drop students from their courses. They can also upload/ download files for their courses.

Students can only view the courses they are enrolled to, and view their classmates and professors' contact information. They can only download files from the course they enrolled in.

Below is the operation flow based on different login role.



Technologies

For source control, we used private git from Atlassian Bitbucket.

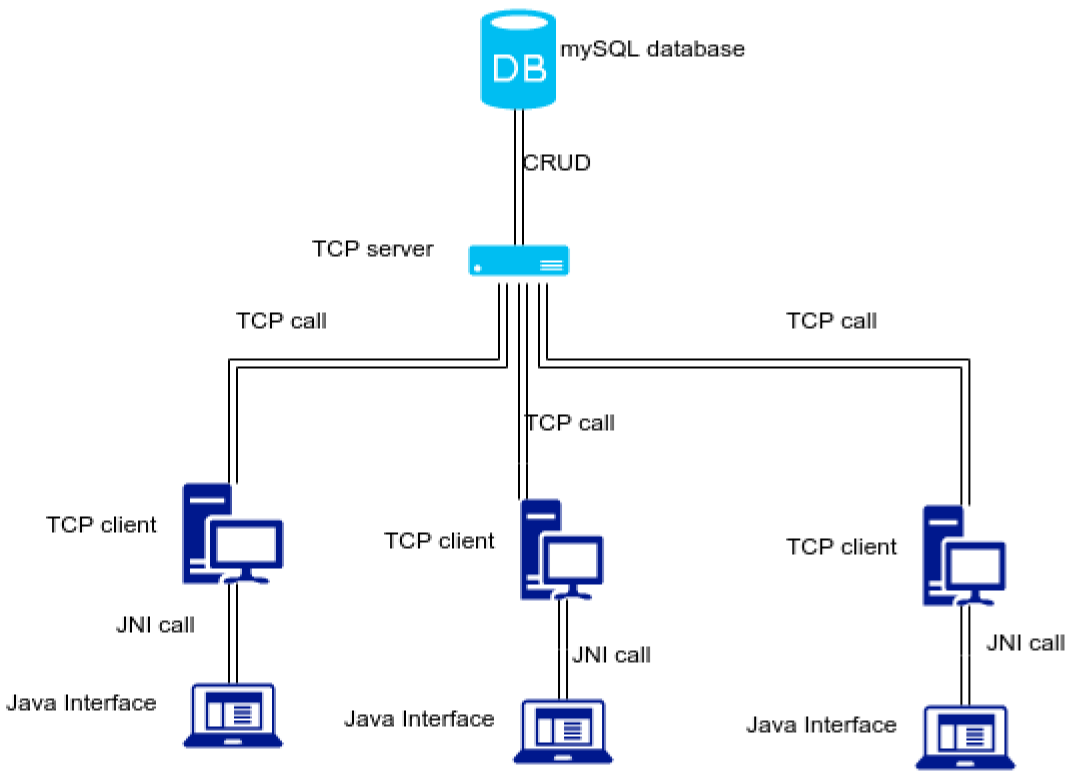
Server wise, we used PThreads, BSD Sockets, JSON-C, libmcrypt, and File handling.

For client UI side, we used Java Swing, and Java JNI.

Our database is MySQL.

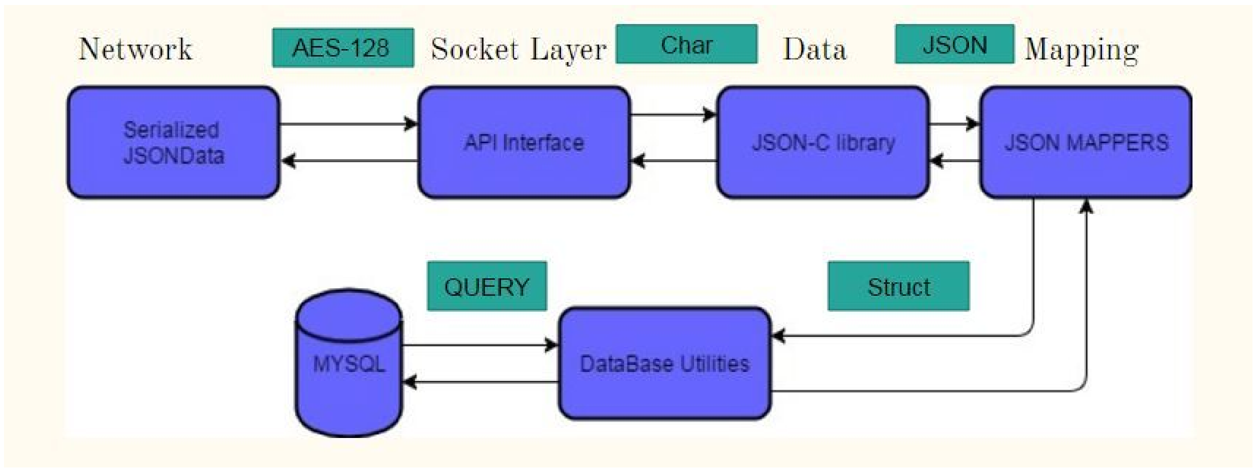
Architecture and High level design

Overview



There are four main components in the application which contains MySQL database and the TCP server in the server side as well as TCP client and Java interface in the client side. The main application logic is in the TCP server which will be communicate to the client through TCP message in a json format. Then the message will be passed to the Java interface through a JNI layer which will be described in the section below.

TCP Server



The server architectural block diagram is shown as above. The server implements following components in order to receive data,

1. API Interface :

The API Interface consists of TCP sockets in order to send and receive data. Like any other typical socket program server binds itself to an address and listens to a port which user can configure. The server receives commands in the form of strings from the client and validates the command. If client sends invalid command it will give back json showing error message like,
{error : 'Error Message'} so client will get appropriate error messages. Once the command has been validated it will give an internal call to JSON-C library in order to deserialize the JSON data that has been received.

2. The JSON-C library provides the convenient way of transforming c structures to the JSON representation and vice versa. It has its own set of functions which returns json_object. This set of functions returns us different json_objects based on which datatype in c we want to convert to. For example, `json_object *jint = json_object_new_int(10);` will gives us a handle/pointer to the json object and will contain an integer with value 10. This component also has json-glib library for serializing and deserializing json_object and json data that we receive at API interface level.

3. JSON Mappers : This block is set of user defined functions which maps data containers such as c structs to the appropriate json_object defined in JSON-C library.

This json_objects are then serialized/deserialized by the JSON-C library as described above.

4. Database utilities : This is the set of utility functions which communicates directly with the database. It has capability to dynamically form SQL queries based on the filters and entities provided to this layer. It supports basic SQL operations like Insert, update , delete and fires appropriate queries based on API invoked. It also dynamically allocates the user defined structs and map sql records to appropriate struct objects so as to return result. The functions regarding this layer can be found in dbutil.h/dbutil.c files.

5. User Defined structures :
To map the database entities which are described below we use user defined structures which are defined in schema.h header file. These structures are useful in mapping the tuple data into arguments which can be passed to appropriate procedures.Some of these structures are,
PERSON, COURSE_DATA, COURSE_STUDENTS_RL, etc.

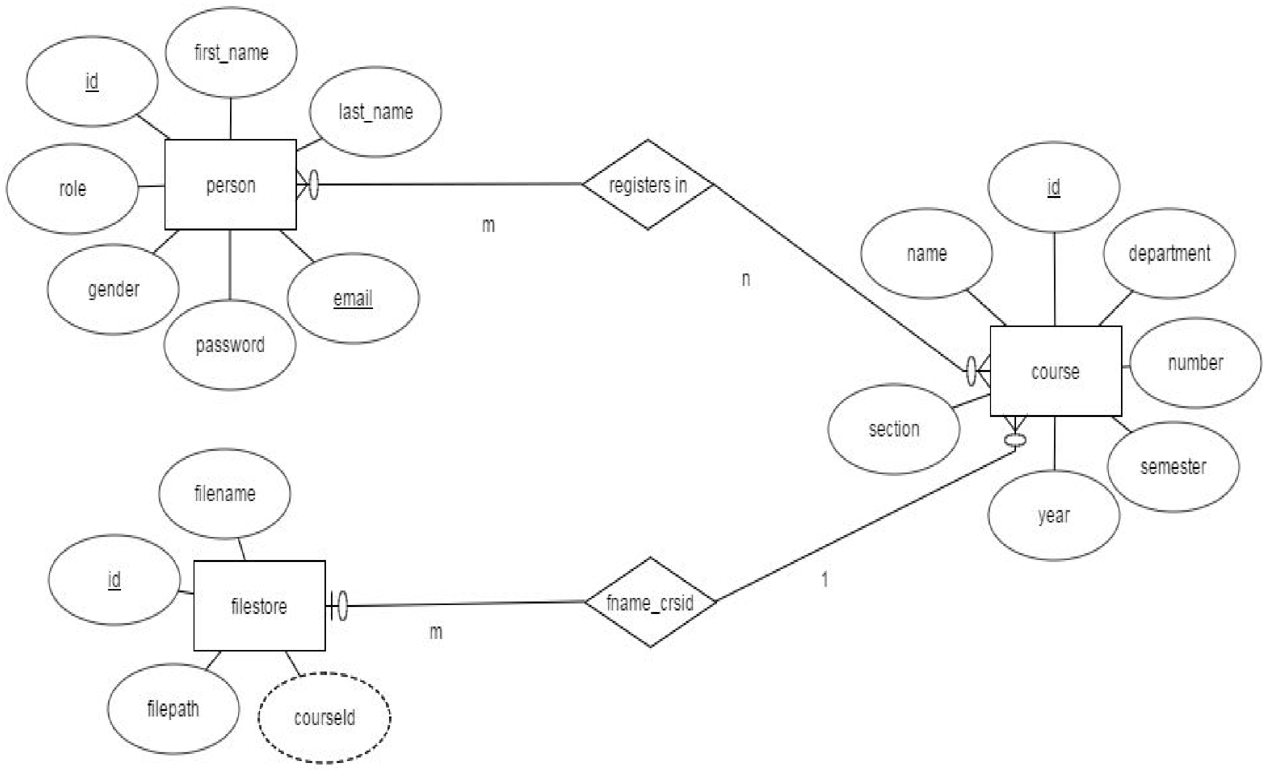
6. Security :
The data exchanged between our client and server is encrypted with AES-128 block cipher using CBC scheme. The libmcrypt library encrypts the outbound message along with decrypting inbound message to client as well as to the server. The code to encrypt and decrypt is part of encryption.c and encryption.h files which uses libmcrypt functions internally. For now the initial vector and key has been hardcoded in the client as well as the server but later we can implement any key exchanges algorithm such as DH algorithm or RSA to exchange data more securely.

Below is the Pseudo code how request is handled by the server for each thread it spawns,
Note: You can find original code in server.c in zip file attached

```
void * clientHandler(void * arg) {
      //args needed ssock,
      int ssock = *((int *)arg);
       while (1) {
          /*
           GET THE COMMAND ENTERED BY CLIENT
           */
          if (valid command) {

             if (strcmp(inputs[0], "current") == 0) {
                    /*
```

```
                GET THE CURRENT SEMESTER
                */
            }
            else if (strcmp(inputs[0], "select") == 0) {

                if ((strcmp(inputs[1], "person") == 0 || strcmp(inputs[1], "course") == 0) &&
                    ((i - 2) % 3 == 0 || ((i - 3) % 3 == 0 && strcmp(inputs[i - 1], "current") == 0))) {
                        /*
                         SELECT PERSON OR COURSE CALL TO DBUTIL.c
                        */

                } else if(strcmp(inputs[1], "filestore")==0) {
//select filestore courseId is 1
                    /*
                    SELECT FILESTORE to RETRIEVE a FILE
                    */
                } else {
                    commandNotFound(ssock, "Wrong table name or wrong number of commands");
                }
            }
            else if (strcmp(inputs[0], "insert") == 0) {
                if (i == 8)  {
                    if (strcmp(inputs[1], "person") == 0) {
//insert person Lam Tran lamtran@gmail.com 910209 2 1
                        /*

                         CREATE A PERSON
                         */
                    } else if (strcmp(inputs[1], "course") == 0) {
//insert course CMPE 280 Fall 2016 UIDesign 01
                        /*

                          CREATE A COURSE
                         */
                    } else {
                        printf("\n\n Wrong table name! \n\n");
                        fflush(stdout);
                    }
                } else {
                    commandNotFound(ssock, "Wrong number of commands");
                }
            }

        else if (strcmp(inputs[0], "delete") == 0) {
//delete person email bingshi0112@gmail.com
//delete course id 9
                if (strcmp(inputs[1], "person") == 0 || strcmp(inputs[1], "course") == 0) {
                    /*
                    DELETE PERSON OR COURSE
                    */
                } else {
                    printf("Wrong table name!\n\n");fflush(stdout);
                    commandNotFound(ssock, "Wrong table name");
                }
            }
        else if (strcmp(inputs[0], "check") == 0) {
                if (i == 3) {
//check bingshi011@gmail.com 900112
                     /*

                    CHECK IF COURSE OR PERSON EXIST
                    */
                } else {
                    commandNotFound(ssock, "Wrong number of commands");
                }
            }
        else if (strcmp(inputs[0], "get") == 0) {
                    /*

                    GET THE COURSE OR PERSON RELATIONSHIP
                    */
         }

        else if (strcmp(inputs[0], "enroll") == 0) {
//enroll person email bingshi011@gmail.com course id 1
                    /*
                    ENROLL A STUDENT INTO COURSE IF EXIST
                    */
       }
        else if (strcmp(inputs[0], "drop") == 0) {
//drop person email bingshi011@gmail.com course id 1
                    /*
                    DROP A PERSON FROM THE COURSE
                    */
         }
           else if(strcmp(inputs[0], "upload")==0 ){
//upload file_name course_id
                /*
                  UPLOAD A FILE TO THE COURSE IF EXIST
                 */
            } else if(strcmp(inputs[0], "download")==0 && i==3) {
//download file_name courseId
            //1. check if file exists
                /*
                    DOWNLOAD A FILE IF EXISTS FOR A COURSEID
                 */

                }
         }
        else {
                /*
                  SEND ERROR JSON TO THE CLIENT SINCE COMMAND IS NOT FOUND
                */
         }
     }// end of while
}
```

__Database__



The database has following entities,

1. Person:

This entity has attributes like id (unique), first_name, last_name, email (unique), password, gender and role. The role defines whether the person will be a student, instructor or admin. A student can register in multiple courses as shown in ER diagram above.

2. Course:

A course is subject in which student can enroll into. Course has attributes like id(unique), department, number, semester, year, name, section. A course can have multiple students enrolled into it. Also there can be different sections of the same course under different instructors.

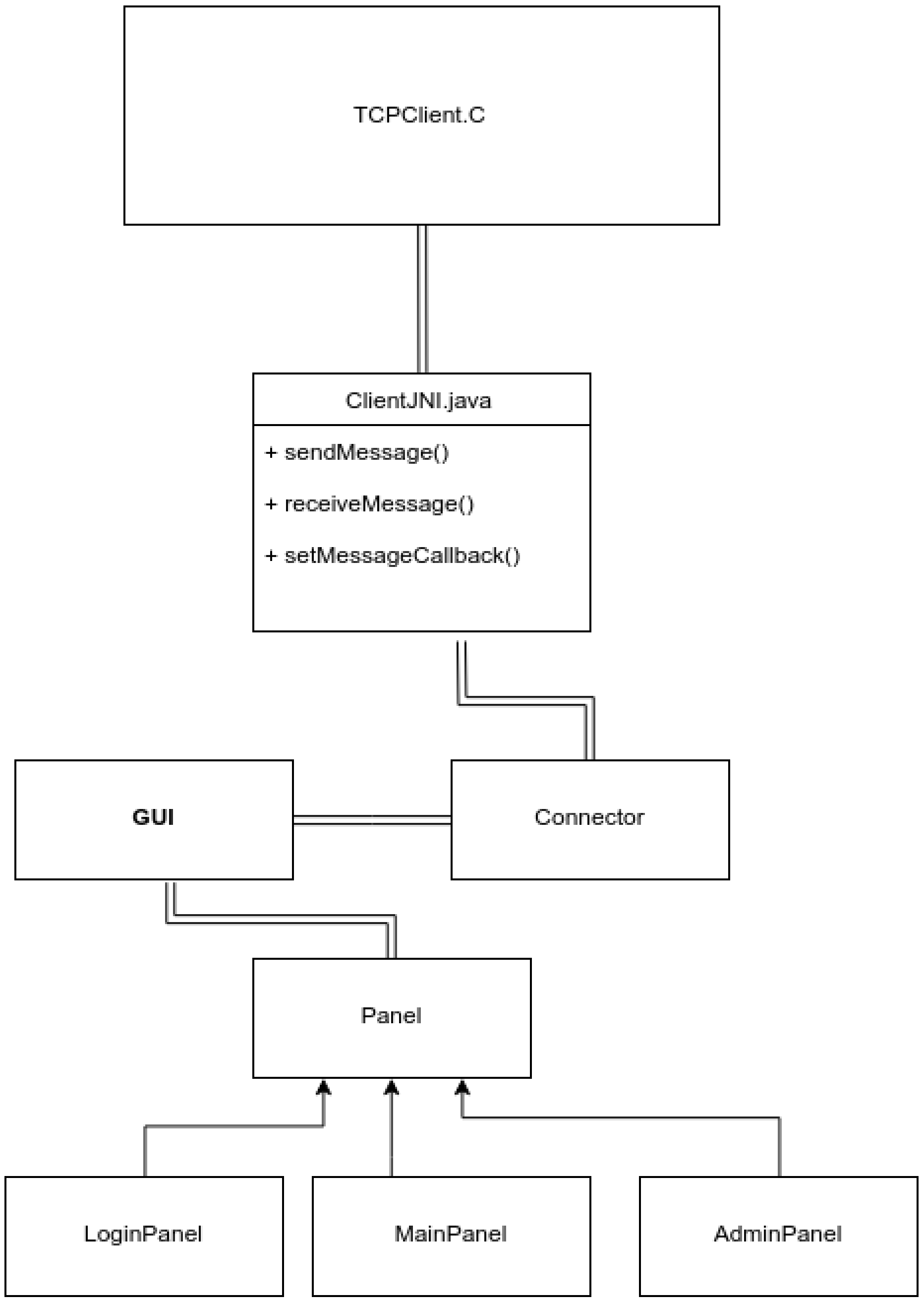
3. course_students_rltn :

This is the relationship table which maintains a relation between person entity and course entity. As described earlier this relation can be many to many therefore it has been maintained using different table.

4. filestore:

This table was added later to the schema as a part of file sharing feature. It stores the data regarding file or the course material that has been uploaded to the course. While downloading a file server check for available file_name and courseId relation in order to provide appropriate result to the client.

TCP Client and Java Interface

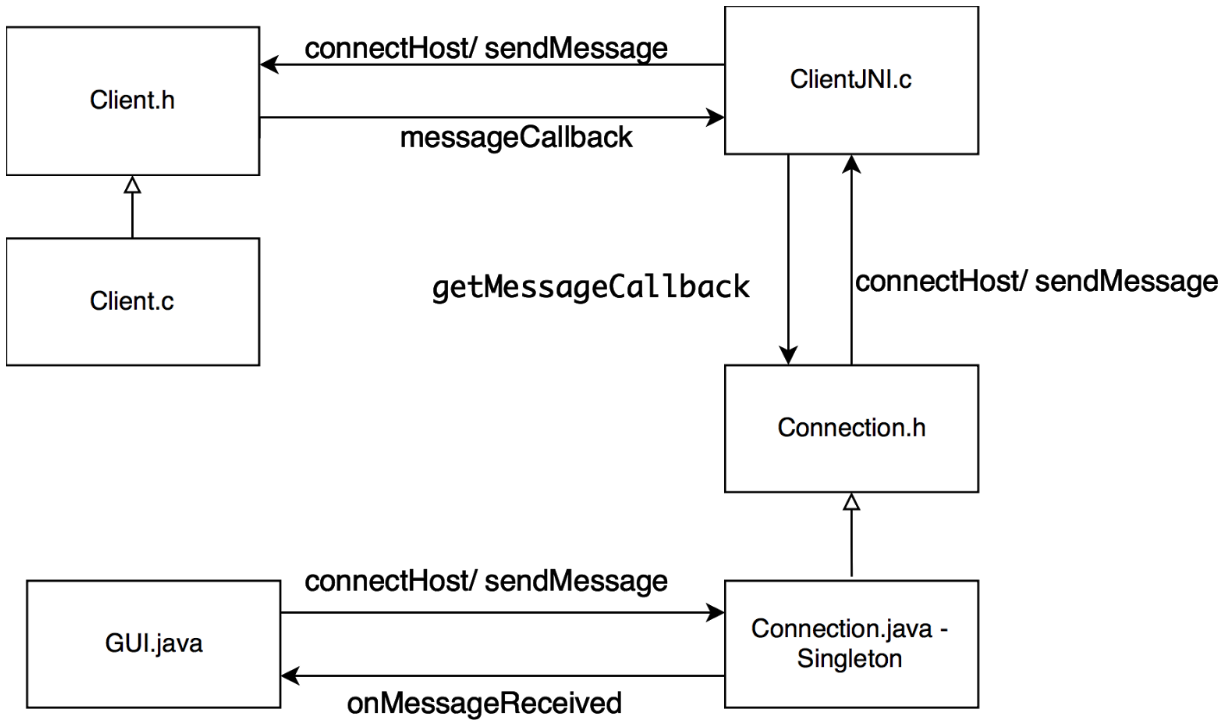


All of our logic and client-server communications are happened through C server and client. Java application is only used to display the message received from C client.

TCP Client
Main responsibility of the client is to send and received TCP message with our special convention to prevent loss of message we described in the Server section. When the

client started, it will try to establish the connection to the server then wait for further command from the user input.

TCP Client also need to provide an interface for Java JNI connect to through function call and callback. Whenever a command is sent to the client from user interface, a function pointer as extra parameter. That function will be trigger if there is any response from the server. The whole response string will be sent back using the callback function pointer.



Client.c and java application communicate

Client.c

```
+    void connectHost(const char *host, const char *service, const void
     (*onMessageReceive)(char const *));

+    int sendMessage(int s, char *message);
```

Java Application

ClientJNI.c

Java Application includes ClientJNI as an Java Native Interface layer to transfer data between java interface and the native C code. ClientJNI has connectToHost(), sendMessage() and receiveMessageCallback() which call directly into Client.c I mentioned above.

```
static void getMessageCallback(JNIEnv * env, jobject o, char * message) {
        midStr = (*env)->GetMethodID(env, (*env)->GetObjectClass(env, o),
"getMessageCallback", sigStr);
         jstring string = (*env)->NewStringUTF(env, message);
         (*env)->CallVoidMethod(env, o, midStr, string);
}
static void getMessageFromClient(const char * mess) {
    getMessageCallback(smEnv, smObject, mess);
}
JNIEXPORT void JNICALL Java_Connection_connectHost  (JNIEnv * env, jobject jobj) {
    smEnv = env;
    smObject = jobj;
    void (*mess) (char *) = getMessageFromClient;
    connectHost("localhost", "9201", mess);
}
JNIEXPORT void JNICALL Java_Connection_sendMessage (JNIEnv * env, jobject jobj, jstring
string) {
     const char *nativeString = (*env)->GetStringUTFChars(env, string, 0);
    setInput(nativeString);
}
```

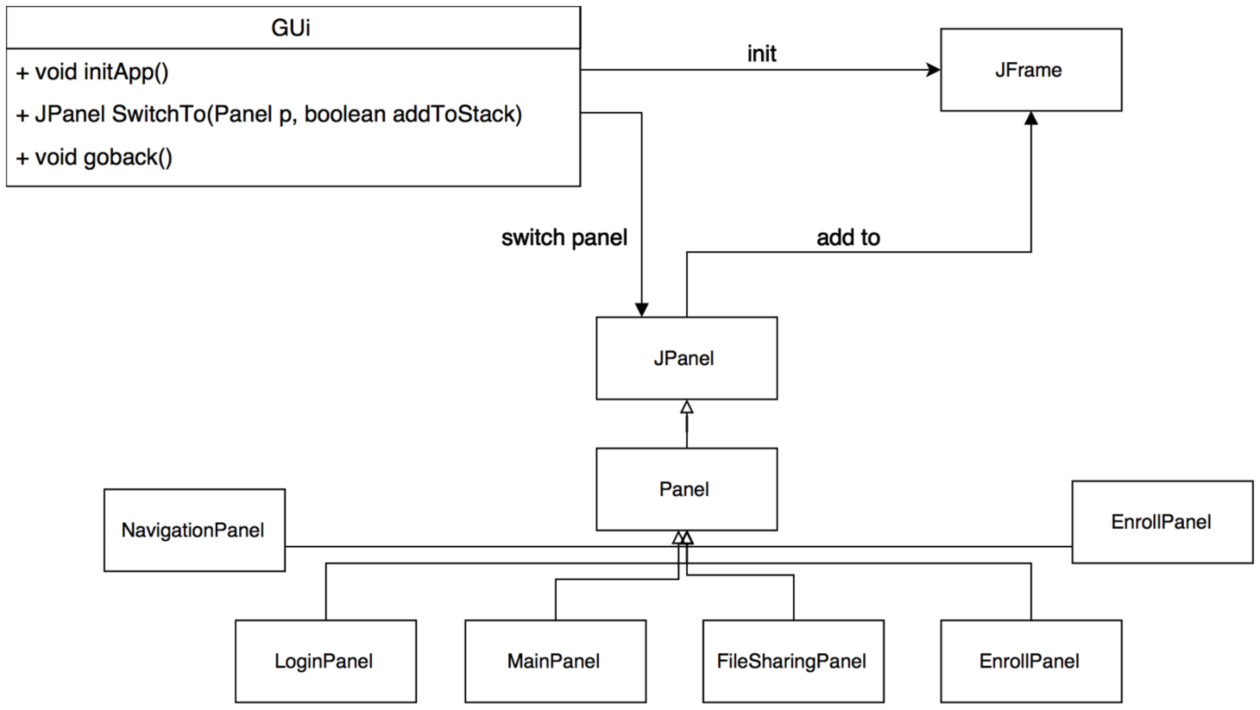
Connection.java

In the Java part, the Connection is a singleton class to play as a wrapper with all the method to JNI layer. That way, if any other classes need to send or receive message from the server, it can call through the connection class without loading the native library.

```
static {
        System.loadLibrary("clientJNI");
    }
```

```
+     public native void connectHost();
+     public native void sendMessage(String st);
+     public interface OnMessageReceivedListener {
              void onMessageReceived(String message);
       }
+     public static void setListener (OnMessageReceivedListener listener)
+     public void getMessageCallback(String message)
```

Java Swing:



Java Swing application design

The application start from GUI.java which will initialize the java swing JFrame and place the Login JPanel as its mainContentPane. From that point, each screen will know what information it need to display and request the according information. Ever screen is represented by a Panel with a Navigation Panel on top.

Extra Features Added

Apart from features such as adding course, students, professors in our role based system in the last iteration of our project we improved it by adding some extra features such as File Sharing as well as Security.

- Security : We added AES-128 security to our application for exchanging data.
    - Note: This features has already been explained in TCP Server section of this report.

- File Sharing : For file sharing we did add API to upload as well as download files. The authority to upload or download a file is decided by role of a user whether he/she is Admin, Professor or Student. The files can also be viewed in our newly modifies java swing UI along with whole new screen to upload them. The files can be maintained per course and a new database schema called filestore has been added at the backend to support this feature. This schema stores the metadata regarding file along with its path on server storage.

- Improved UI : As decided earlier we also added new UI screens to upload and download file to view along with improvement to existing screens to make them more user friendly.

- Test and improve server/ client stabilities : We also made our project demo ready

along with UI, backend features and tested it regressively for more stability using manual testing.

Future Plan

- Integrate more APIs with UI, for example, filtering, sorting and searching.

- Add more feature if possible, for example, email service, grading system, post announcement, and allow group forming for students.

- Enable securities key exchange/ hash password

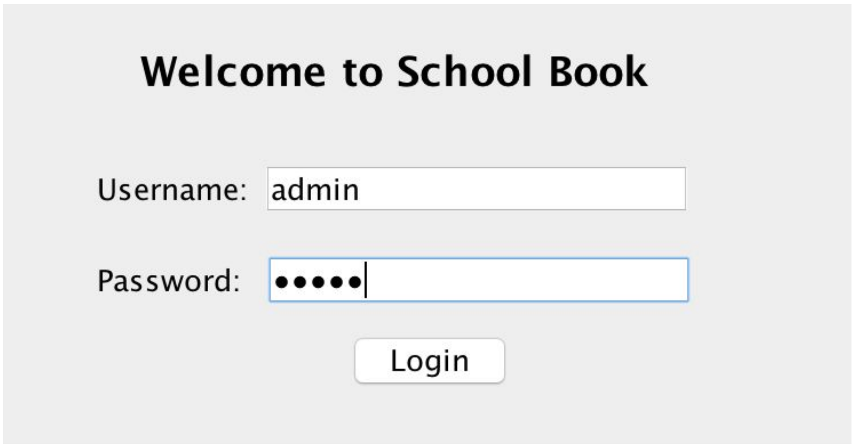
Challenges

- Efficiently design the server commands
- Integrate different layouts
- 3 different system: Window, Mac, Ubuntu
- Feasibility of Libraries JSON-C, mcrypt, etc
- Interaction with DB (Query Builder)
- Inclusion of AES-128 Block Cipher and its impact on Socket code
- Efficient communicate between Server and Client using tcp

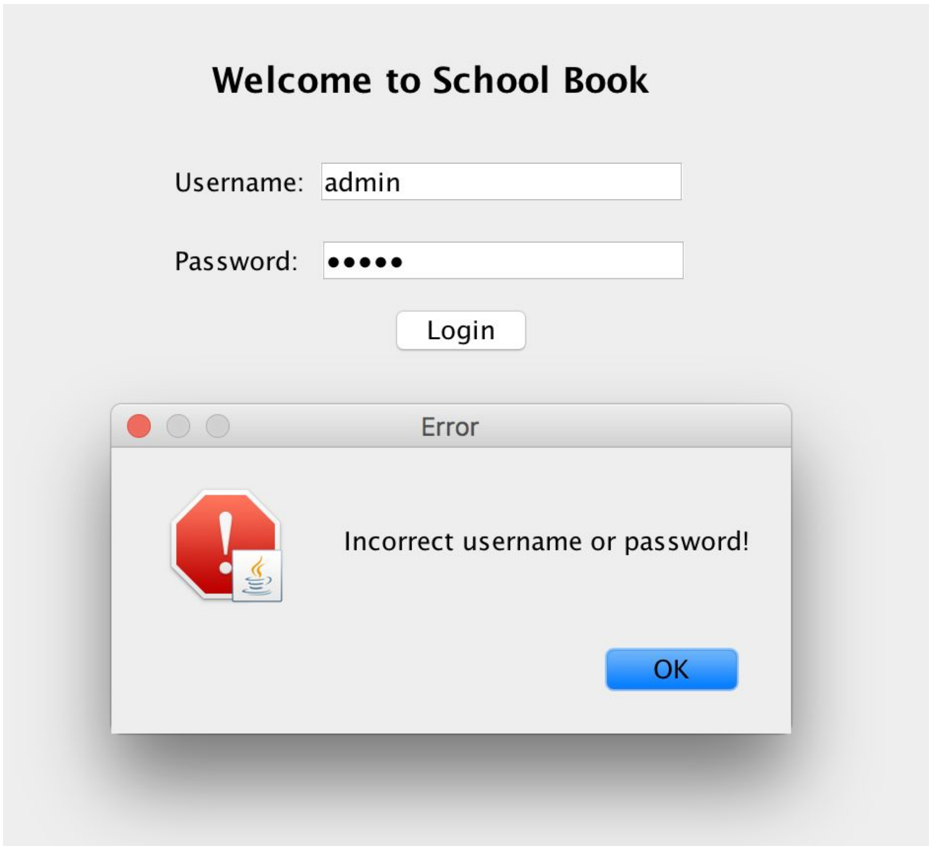
- Passing data and callback from C client to Java GUI

Snapshots and Application Flow

Login screen

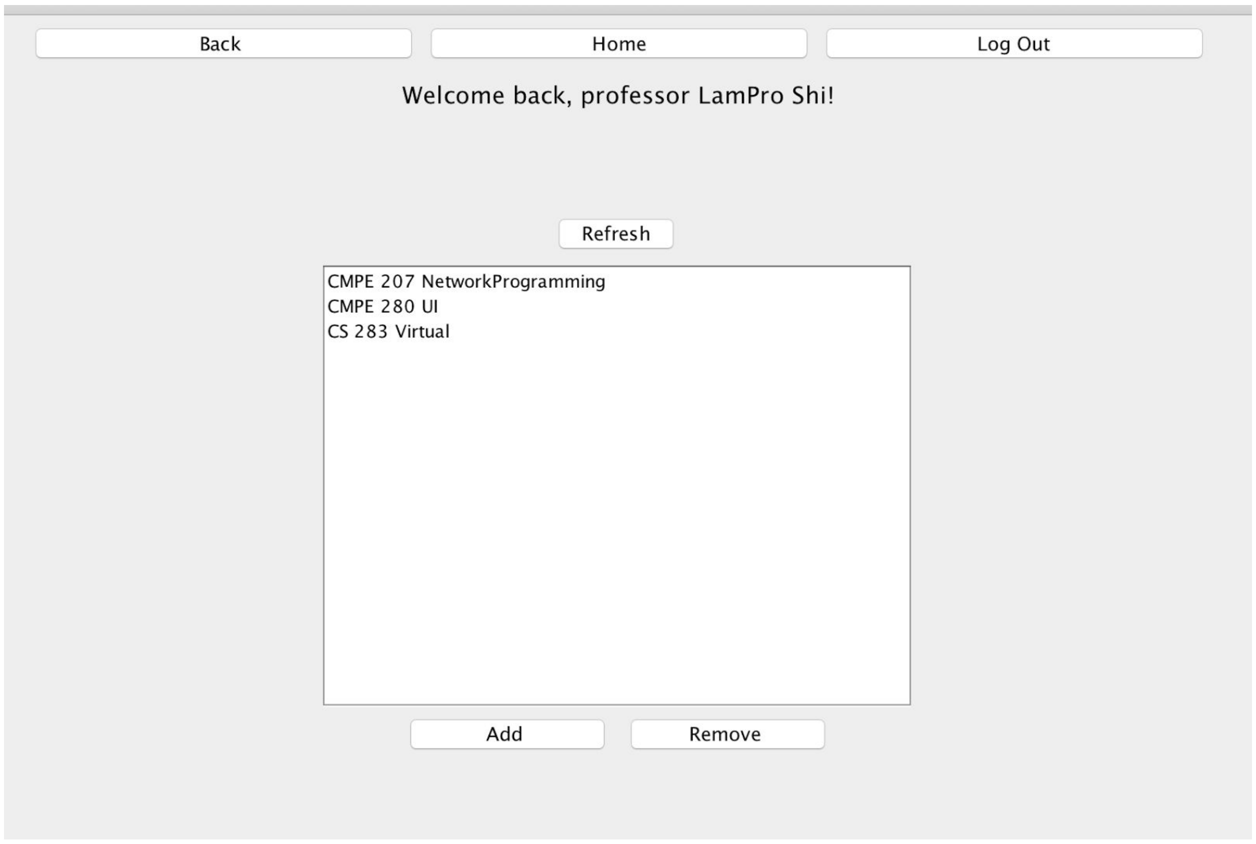


Wrong username/password



Incorrect username or password!

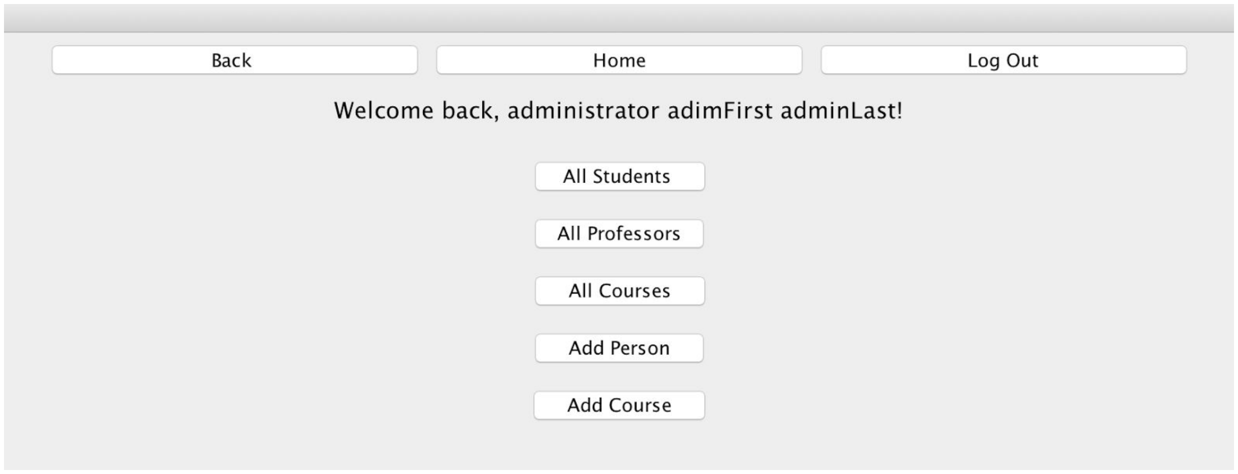
Professor logged in screen



Welcome back, professor LamPro Shi!

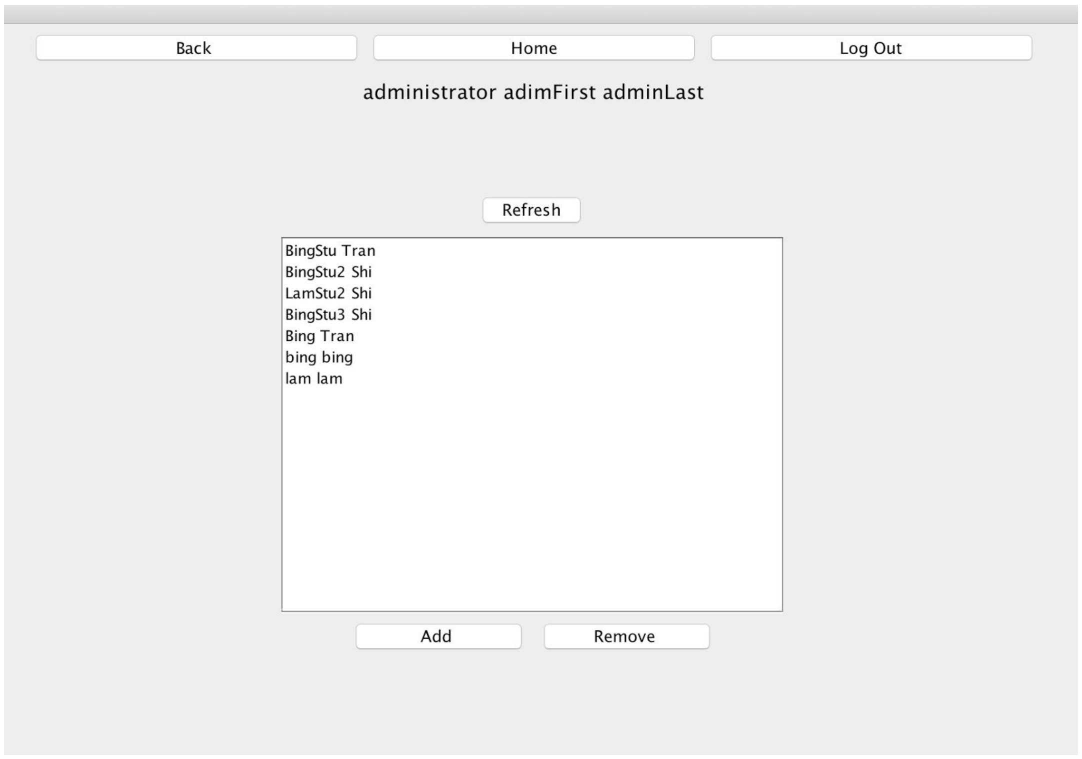
CMPE 207 NetworkProgramming
CMPE 280 UI
CS 283 Virtual

Administrator logged in screen



Welcome back, administrator adimFirst adminLast!

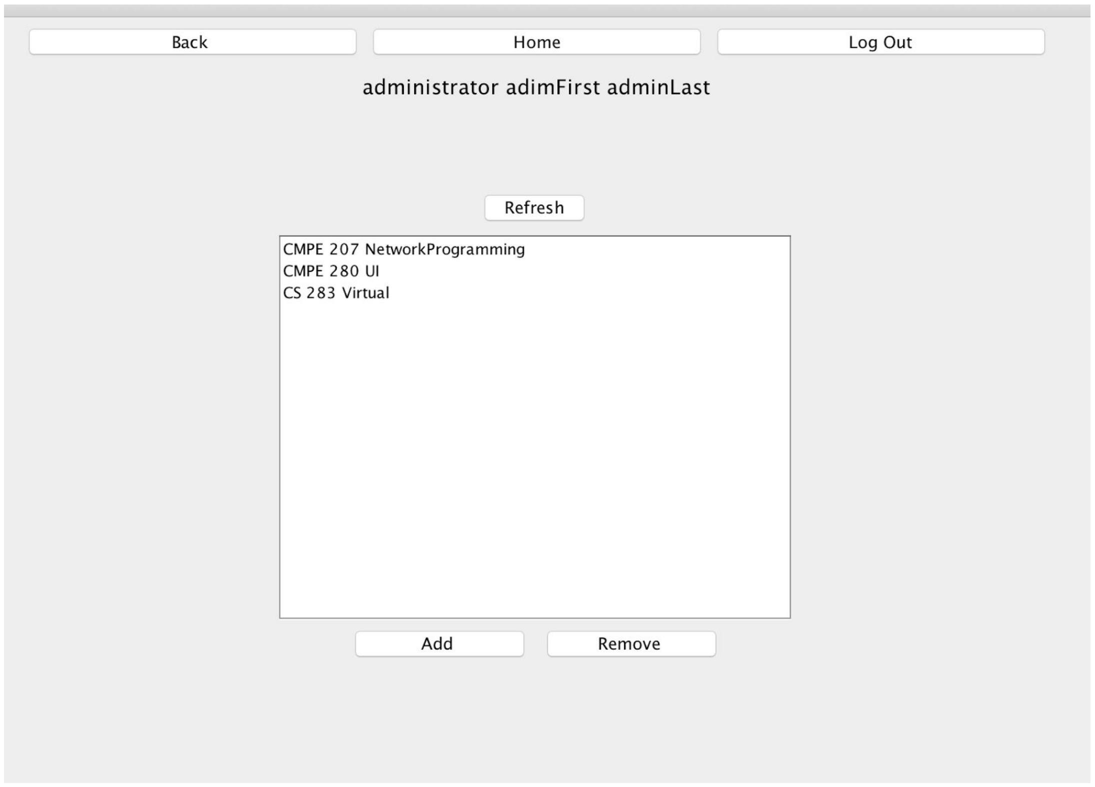
Admin click on "all students"



administrator adimFirst adminLast

```
BingStu Tran
BingStu2 Shi
LamStu2 Shi
BingStu3 Shi
Bing Tran
bing bing
lam lam
```

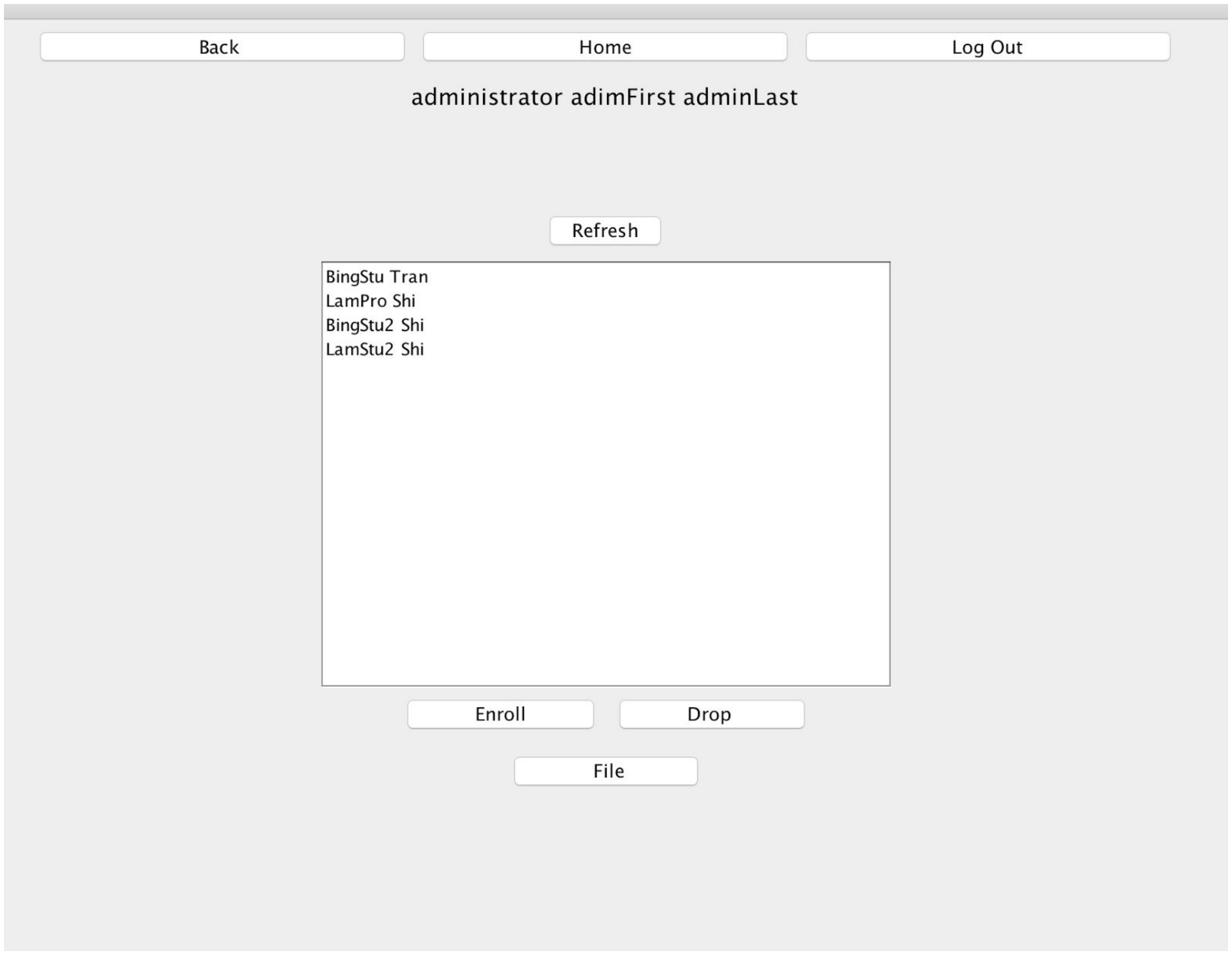
Admin click on "all courses"



administrator adimFirst adminLast

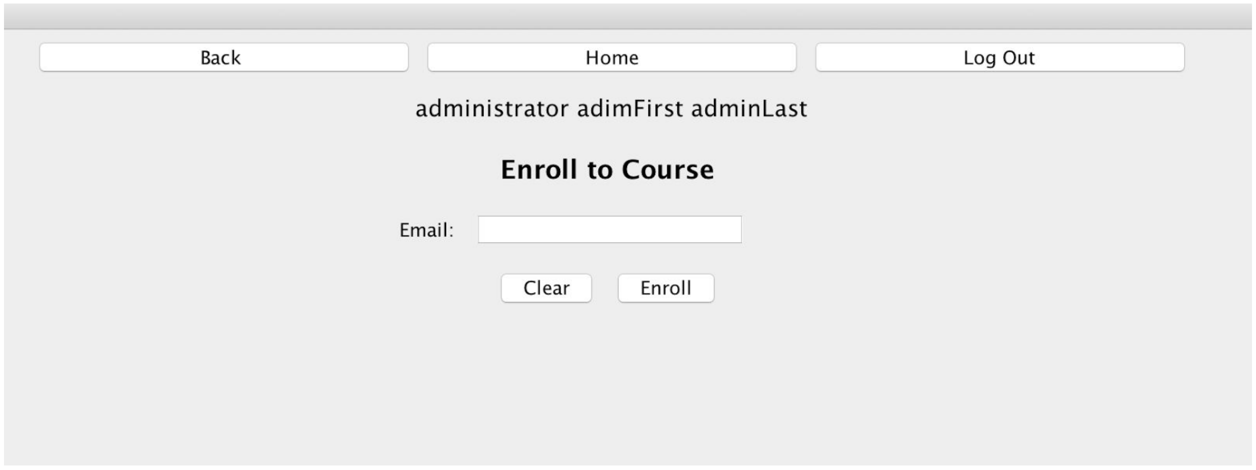
```
CMPE 207 NetworkProgramming
CMPE 280 UI
CS 283 Virtual
```

Double click on a course

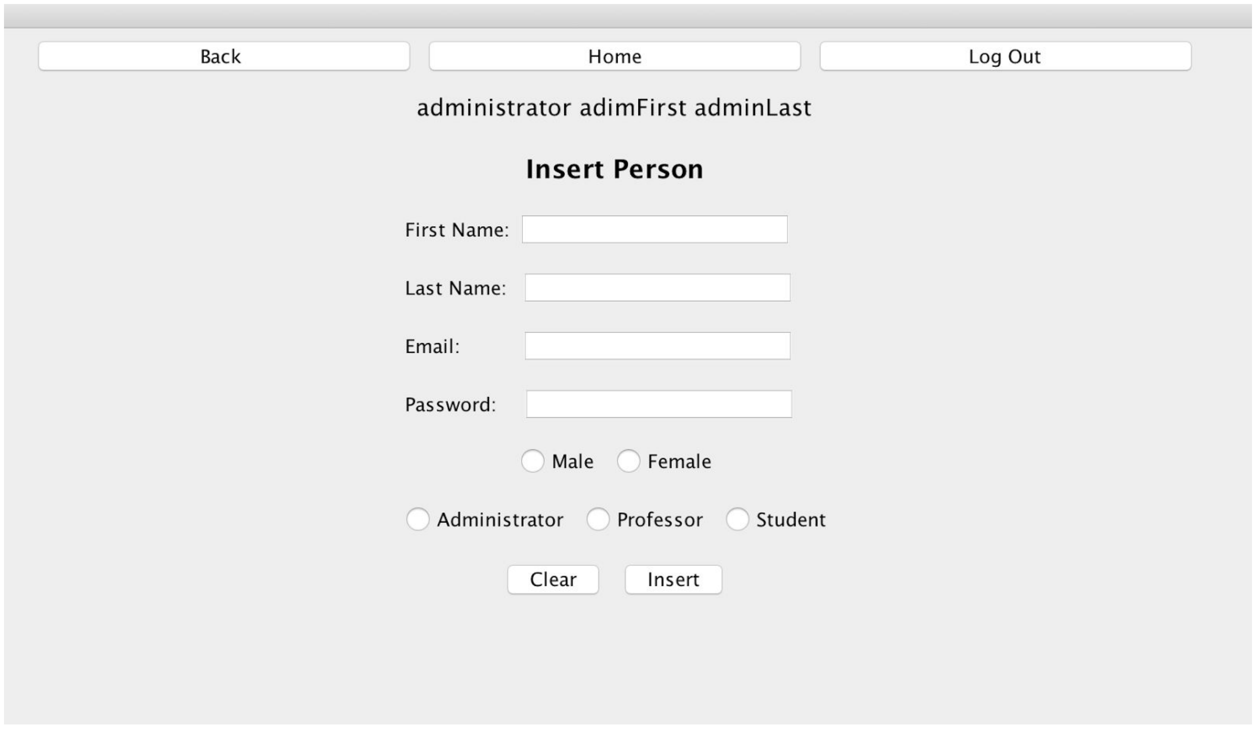


BingStu Tran
LamPro Shi
BingStu2 Shi
LamStu2 Shi

Enroll a student to a course



Insert user screen



administrator adimFirst adminLast

Insert Person

First Name:

Last Name:

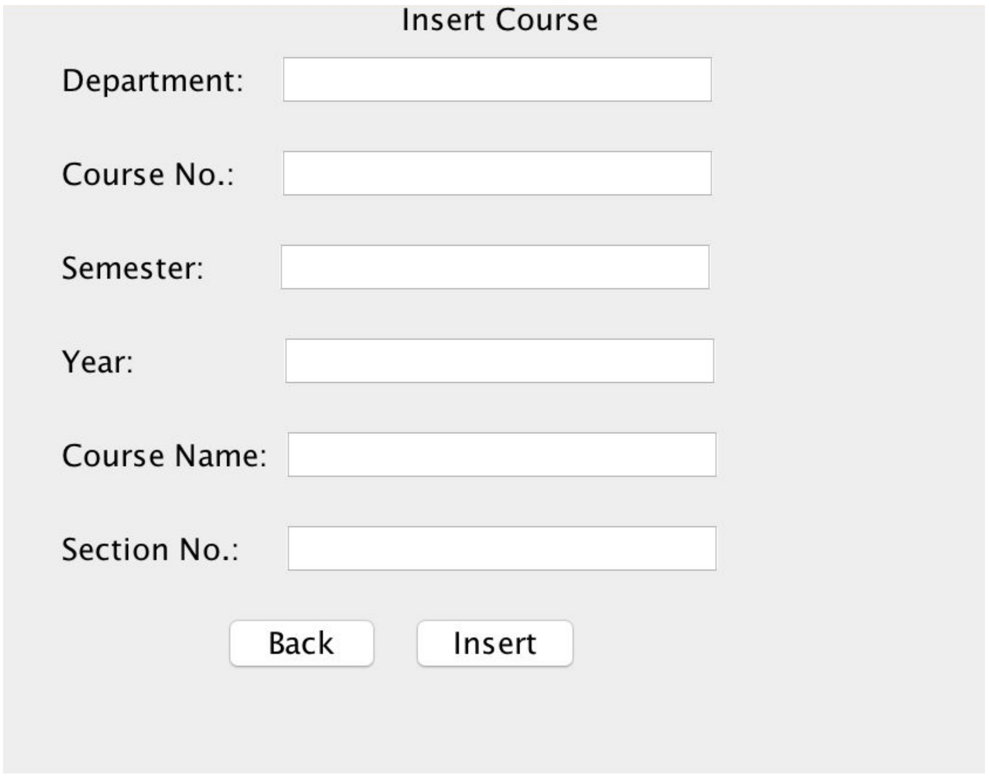
Email:

Password:

( ) Male ( ) Female

( ) Administrator ( ) Professor ( ) Student

Insert course screen


Insert Course

Department:

Course No.:

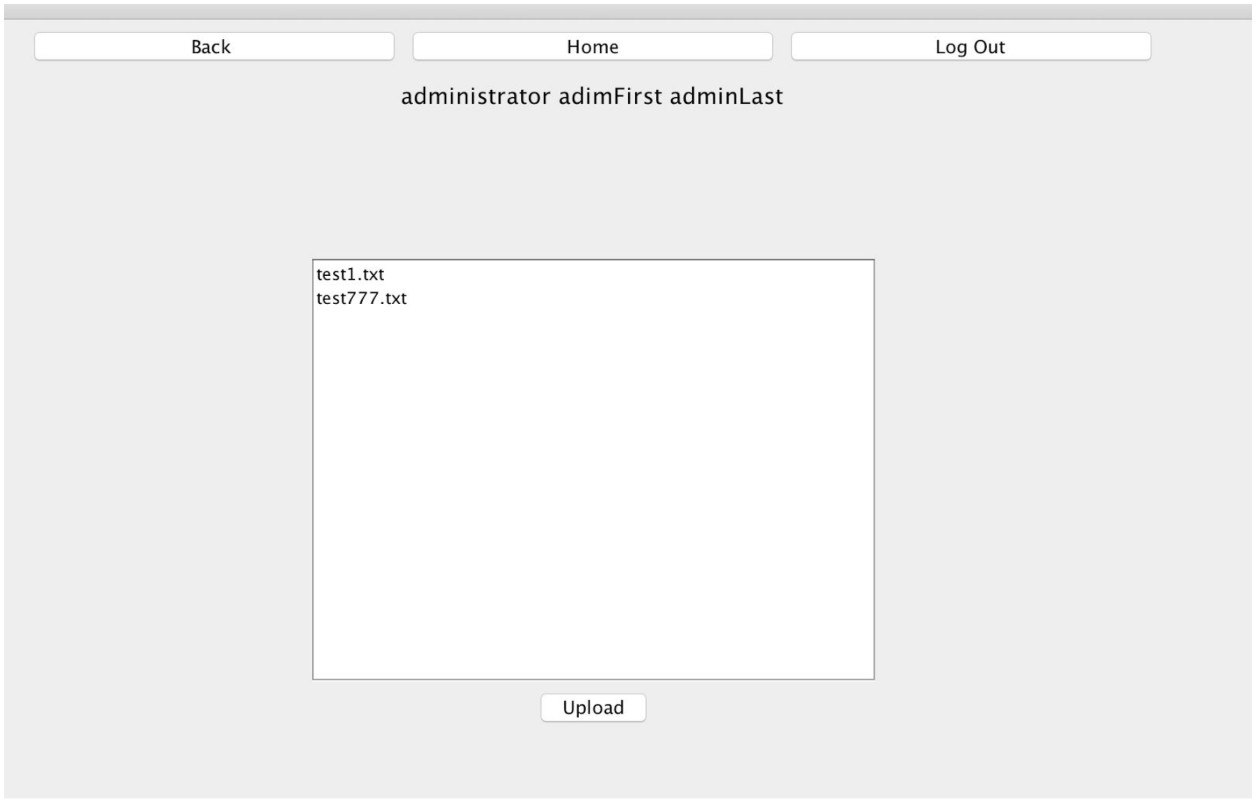
Semester:

Year:

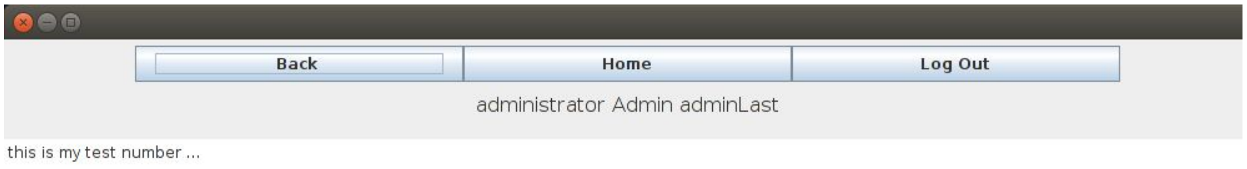
Course Name:

Section No.:

__File upload/download screen__



__Double clicked the file to download/ view screen__



this is my test number ...