SURVEY ENGINE

CSC-17B

Dr. Mark Lehr

Royce Nguyen

INTRODUCTION:

The survey engine is a program used to create, answer, record, and display surveys to users and admins. A survey consists of different types of questions that are to be answered by the user. The admin has the ability to create surveys and review the results of the survey. A survey is used to gauge the opinion of the group that answered it in order to help the admin make decisions based on the results.

THOUGHT PROCESS ON C++ IMPLEMENTATION:

Of course, in order to follow the MVC design pattern, I initially created a class for each component. The model class is the largest class and it handles many of the operations (if not all) of the survey engine. The view class is responsible for displaying output to the user. My view class currently only displays menu-based choices. The controller class is simply the input for the system. It does input validation and returns the result to the model to handle.

ADDITIONAL CLASSES:

I added a Form class which is basically a wrapper for a survey. It contains the name of the form. The questions on the form, and the answers of the form. The questions are a separate class which contain the question itself and possible extra choices that the user can choose from (think multiple choice questions). Beyond that, there is a student class, which will contain the name and id of the student, as well as the forms that the student has already answered.

ADMIN and USER:

The admin has the permission to both create and delete surveys as well as students. The user is only able to fill out surveys and see past surveys.

DIFFICULTIES IN CODING:

Having to create the menu system is something that I do not often do, so implementing switch statements, do while menus really solidified by understanding of menu systems and how to handle going back and forth between different menus. Another thing that was very hard was the idea of saving data. Since the data in a survey engine is very intensive and hard to compress, I had to find my best solution for storing the data in a text file based database.

PSEUDOCODE:

output main menu

Take user input

Validate user input

If (login)

Attempt login

If (login successful)

If (user)

Model handle user

If (admin)

Model handle admin

Else

Output menu again

If (exit)

Save data

Exit program

Actual Code:

#include <bits/stdc++.h>

#include "Form.hpp"

#include "Model.hpp"

#include "Controller.hpp"

#include "View.hpp"

using namespace std;

int main()

{

Model sys;

View mainmenu("mainmenu.txt");

Controller mainMenuCont(mainmenu.getSize());

//do while menu loop

do

{

mainmenu.display();

mainMenuCont.TakeInput();

//name and id

string str, str2;

switch(mainMenuCont.ReturnInput())

{

case 1:

//login case

cin.ignore();

cout << "Enter first and last name: ";

getline(cin, str);

cout << "Enter ID: ";

cin >> str2;

//pass to the model class to handle

if (sys.handleLogin(str, str2))

{

break;

}

//handle the admin and student menu functions

sys.getAdmin() ? sys.handleAdmin() : sys.handleStudent();

break;

case 2:

//terminate

cout << "Program Terminated" << endl;

//save info to database;

sys.saveInfo();

exit(0);

default:

break;

}

}

while(mainMenuCont.ReturnInput() != 2);

return 0;

}

#include "Model.hpp"

#include "View.hpp"

#include "Controller.hpp"

enum QUESTION\_TYPE{ SHORT\_ANSWER, MULTIPLE\_CHOICE, CHECK\_BOX };

Model::Model()

{

//init to default values

this->loggedin = false;

this->admin = false;

this->currentid = "";

this->currentname = "";

this->currentsession = "";

this->insession = false;

ifstream in;

string str, str2;

//read in all survey names

in.open("surveynames.txt");

while (in >> str)

{

this->surveyNames.push\_back(str);

}

in.close();

//read in all valid student ids

in.open("valid\_student\_ids.txt");

while (getline(in, str))

{

in >> str2;

//push into valid ids

valid\_student\_ids.push\_back({str, str2});

//push into as sample size

sample.emplace\_back(str, str2);

in.ignore();

}

in.close();

in.open("admin\_ids.txt");

while (getline(in, str))

{

in >> str2;

//read in admin ids

valid\_admin\_ids.push\_back({str, str2});

in.ignore();

}

in.close();

//read in all surveys using constructor

for (int i = 0; i < surveyNames.size(); i++)

{

string str = surveyNames[i] + ".txt";

Form x(str);

surveys.push\_back(x);

}

//read in student survey data (already taken)

in.open("surveydata.txt");

int n, ns, nq;

in >> n;

in.ignore();

for (int i = 0; i < n; i++)

{

//read in name and id

getline(in, str);

getline(in, str2);

//read in number of surveys

in >> ns;

in.ignore();

//if zero, read in the next student

if (ns == 0)

{

continue;

}

//create a vector of forms

vector<Form> z;

for (int i = 0; i < ns; i++)

{

string s;

//read in survey name

getline(in, s);

//read in number of questions

in >> nq;

in.ignore();

//create a vector of answers

vector<string> a(nq);

for (int i = 0; i < nq; i++)

{

//read in each answer

getline(in, a[i]);

}

z.emplace\_back(s, a);

}

//push the survey to the appropriate student

this->pushSurvey(str, str2, z);

}

in.close();

};

bool Model::handleLogin(string name, string id)

{

if (id != "" && name != "")

{

//iterate through all the ids

for (int i = 0; i < valid\_student\_ids.size(); i++)

{

//check if the student ids match

if (valid\_student\_ids[i] == make\_pair(name, id))

{

this->loggedin = true;

this->currentid = id;

this->currentname = name;

this->admin = false;

}

}

for (int i = 0; i < valid\_admin\_ids.size(); i++)

{

//iterate through all the admin ids

if (valid\_admin\_ids[i] == make\_pair(name, id))

{

this->loggedin = true;

this->currentid = id;

this->admin = true;

}

}

}

//return if the login is successfull or unsucessful

if (this->loggedin)

{

cout << "Login Successful" << endl;

cout << (this->admin ? "Admin" : "Student") << endl;

return false;

}

else

{

cout << "Login Unsuccessful" << endl;

}

return true;

}

void Model::handleStudent()

{

//output student menu and take input

View studentMenu("studentmenu.txt");

Controller studentMenuCont(studentMenu.getSize());

studentMenu.display();

studentMenuCont.TakeInput();

//do while student menu loop with switch case

while(this->studentSwitch(studentMenuCont.ReturnInput()))

{

studentMenu.display();

studentMenuCont.TakeInput();

}

};

void Model::handleAdmin()

{

//output admin menu and take input

View adminMenu("adminmenu.txt");

Controller adminMenuCont(adminMenu.getSize());

adminMenu.display();

adminMenuCont.TakeInput();

//do while admin menu loop with switch case

while (this->adminSwitch(adminMenuCont.ReturnInput()))

{

adminMenu.display();

adminMenuCont.TakeInput();

}

};

bool Model::adminSwitch(int choice)

{

//self explanatory

switch(choice)

{

case 1:

this->createSurvey();

return true;

case 2:

this->deleteSurvey();

return true;

case 3:

this->viewSurveyResults();

return true;

default:

return false;

}

};

bool Model::studentSwitch(int choice)

{

//also self explanatory

switch(choice)

{

case 1:

{

this->fillOutSurvey();

return true;

}

case 2:

{

this->viewPreviousSurvey();

return true;

}

case 3:

{

return false;

}

}

};

void Model::createSurvey()

{

string name;

//take in the name of the survey

cin.ignore();

cout << "Enter the name of the survey: ";

getline(cin, name);

ofstream out("surveynames.txt", ios::app);

out << name << endl;

out.close();

name += ".txt";

//create a file for the survey

//init

View questionType("types.txt");

Controller questionTypeCont(questionType.getSize());

int n;

//read in and output the number of questions

cout << "How many questions are in this survey: ";

cin >> n;

out.open(name, ios::app);

out << n << endl;

out.close();

for (int i = 0; i < n; i++)

{

//take in the question type and create the question based on the type

questionType.display();

questionTypeCont.TakeInput();

this->createQuestionSwitch(questionTypeCont.ReturnInput()-1, name);

}

};

bool Model::createQuestionSwitch(int choice, string filename)

{

cin.ignore();

ofstream out;

out.open(filename, ios::app);

string q, str;

vector<string> a;

cout << "Enter the question: ";

getline(cin, q);

//short answer just needs the question

//multiple choice and check box need the extra choices for the user to choose

switch(choice)

{

case SHORT\_ANSWER:

{

out << SHORT\_ANSWER << endl;

out << q << endl;

out.close();

return true;

}

case MULTIPLE\_CHOICE:

{

out << MULTIPLE\_CHOICE << endl;

cout << "Enter one choice (-1 to stop): ";

getline(cin, str);

while (str != "-1")

{

a.push\_back(str);

cout << "Enter one choice (-1 to stop): ";

getline(cin, str);

}

out << a.size() << endl;

for (int i = 0; i < a.size(); i++)

{

out << a[i] << endl;

}

out.close();

return true;

}

case CHECK\_BOX:

{

out << CHECK\_BOX << endl;

cout << "Enter one choice (-1 to stop): ";

getline(cin, str);

while (str != "-1")

{

a.push\_back(str);

cout << "Enter one choice (-1 to stop): ";

getline(cin, str);

}

out << a.size() << endl;

for (int i = 0; i < a.size(); i++)

{

out << a[i] << endl;

}

out.close();

return true;

}

default:

{

out.close();

return false;

}

}

};

void Model::deleteSurvey()

{

int choice;

View delSurveyMenu("surveynames.txt");

Controller delSurvey(surveys.size());

delSurveyMenu.display();

delSurvey.TakeInput();

//take user input and erase from the vector of surveys

//cannot delete the actual survey info file yet nor from students

surveys.erase(surveys.begin() + delSurvey.ReturnInput()-1);

};

void Model::viewSurveyResults()

{

//output all results of the surveys

for (int i = 0; i < sample.size(); i ++)

{

sample[i].getSurveyResults();

cout << endl;

}

};

void Model::fillOutSurvey()

{

View viewSurvey("surveynames.txt");

Controller viewSurveyCont(viewSurvey.getSize());

viewSurvey.display();

viewSurveyCont.TakeInput();

Form fill(surveyNames[viewSurveyCont.ReturnInput()-1] + ".txt");

//take in the survey answers

fill.takeInput();

//push to the sample group data

for (int i = 0; i < sample.size(); i++)

{

if (sample[i].getName() == currentname && sample[i].getID() == currentid)

{

sample[i].insertSurvey(fill);

}

}

};

void Model::viewPreviousSurvey()

{

//iterate through all surveys

for (int i = 0; i < sample.size(); i++)

{

//if matching to the user's id and name

if (sample[i].getName() == this->currentname && sample[i].getID() == this->currentid)

{

//output results entered by the user

sample[i].getSurveyResults();

}

}

};

void Model::saveInfo()

{

ofstream out;

//output all survey names so the model knows which files to read in for data

out.open("surveynames.txt");

for (int i = 0; i < surveyNames.size(); i++)

{

out << surveyNames[i] << endl;

}

out.close();

//for each survey name

for (int i = 0; i < surveys.size(); i++)

{

//open the survey data file

out.open(surveys[i].getName() + ".txt");

//read in how many questions to look for

out << surveys[i].getQuestionSize() << endl;

//iterate through each question

for (int j = 0; j < surveys[i].getQuestionSize(); j++)

{

//output the type of question

out << surveys[i].getQuestion(j).getType() << endl;

//output the literal question string

out << surveys[i].getQuestion(j).getLitQuestion() << endl;

//if there are extra question elements

if (surveys[i].getQuestion(j).getExtraQSize() != 0)

{

//output the size of the extra elements

out << surveys[i].getQuestion(j).getExtraQSize() << endl;

//iterate through each extra q

for (int k = 0; k < surveys[i].getQuestion(j).getExtraQSize(); k++)

{

//output the question element

out << surveys[i].getQuestion(j).getExtraQ(k) << endl;

}

}

}

}

out.close();

//for each survey, output the student answers

out.open("surveydata.txt");

out << sample.size() << endl;

for (int i = 0; i < sample.size(); i++)

{

out << sample[i].saveResults();

}

out.close();

};

void Model::pushSurvey(string name, string id, vector<Form> a)

{

for (int i = 0; i < sample.size(); i++)

{

//push a survey to the correct student.

if (sample[i].getName() == name && sample[i].getID() == id)

{

sample[i].pushSurvey(a);

}

}

};

#include "Question.hpp"

enum QUESTION\_TYPE{ SHORT\_ANSWER, MULTIPLE\_CHOICE, CHECK\_BOX };

Question::Question(int t, string q)

{

//question type and question initializer

question = q;

type = t;

};

Question::Question(int t, string q, vector<string> qs)

{

//type, question and extra qs if the question is check box or multiple choice

type = t;

question = q;

extraq = qs;

};

string Question::getQuestion()

{

string str = "";

//based on the type, output respectively

switch (type)

{

case SHORT\_ANSWER:

{

//case

str += this->question;

break;

}

case MULTIPLE\_CHOICE:

case CHECK\_BOX:

{

//output the question and then the extra values that go with the multiple choice and check box questions.

str += this->question;

for (int i = 0; i < extraq.size(); i++)

{

str += '\n';

str += '1' + i;

str += ". ";

str += extraq[i];

}

break;

}

default:

{

break;

}

};

return str;

};

//basic getters.

string Question::getExtraQ(int choice)

{

return extraq[choice];

};

int Question::getExtraQSize()

{

return extraq.size();

};

#include "Surveyee.hpp"

Surveyee::Surveyee(string sname, string sid)

{

//name and id constructor

this->name = sname;

this->id = sid;

};

void Surveyee::getSurveyResults()

{

//iterate through each answered survey and output the results

for (int i = 0; i < answered.size(); i++)

{

cout << answered[i].getName() << endl;

answered[i].outputAnswers();

}

};

void Surveyee::insertSurvey(Form x)

{

//push back a form that has been filled out

answered.push\_back(x);

};

string Surveyee::saveResults()

{

string str = "";

//output the name and id

str += this->name + '\n';

str += this->id + '\n';

//output the number of surveys that they answered

str += to\_string((int)answered.size()) + '\n';

for (int i = 0; i < answered.size(); i++)

{

//output the name of the survey

str += answered[i].getName() + '\n';

//output the number of answered

str += to\_string((int)answered[i].getAnswerSize()) + '\n';

for (int j = 0; j < answered[i].getAnswerSize(); j++)

{

//output the string answer

str += answered[i].getAnswer(j);

}

}

return str;

}

void Surveyee::pushSurvey(vector <Form> a)

{

//push a vector of forms. reading from database.

this->answered = a;

};

#include "View.hpp"

View::View(string filename)

{

//read in the valid choices from a file that was passed from the constructor

ifstream in;

in.open(filename);

string str;

while (getline(in, str))

{

choices.push\_back(str);

}

}

void View::display()

{

cout << endl;

//output the choices. the number is handled by the system and not the file

for (int i = 0; i < choices.size(); i++)

{

cout << i+1 << ": " << choices[i] << endl;

}

}

int View::getSize()

{

//get for the controller for validation of the input

return (int)this->choices.size();

}

#include "Controller.hpp"

Controller::Controller(int end)

{

//high end of valid entries

max = end;

}

void Controller::TakeInput()

{

cin >> choice;

//while loop validation

while (choice < 1 || choice > max)

{

cout << "Try again: ";

cin >> choice;

}

}

int Controller::ReturnInput()

{

//basic getter, why am i even commenting this?

return this->choice;

}

#include "Form.hpp"

enum QUESTION\_TYPE{ SHORT\_ANSWER, MULTIPLE\_CHOICE, CHECK\_BOX };

Form::Form(string filename)

{

this->formName = filename.substr(0, filename.size()-4);

ifstream in;

in.open(filename);

//read in the size of the questions needed to iterate through

int size;

in >> size;

for (int i = 0; i < size; i++)

{

int type, num;

string str;

in >> type;

in.ignore();

//based on the question type

switch(type)

{

//only have to take in the literal question

case SHORT\_ANSWER:

{

getline(in, str);

questions.emplace\_back(SHORT\_ANSWER, str);

break;

}

//have to take in the extra elements

case MULTIPLE\_CHOICE:

{

string s;

getline(in, str);

in >> num;

in.ignore();

vector<string> qs;

for (int i = 0; i < num; i++)

{

getline(in, s);

qs.push\_back(s);

}

questions.emplace\_back(MULTIPLE\_CHOICE, str, qs);

break;

}

//same as mult choice

case CHECK\_BOX:

{

string s;

getline(in, str);

in >> num;

in.ignore();

vector<string> qs;

for (int i = 0; i < num; i++)

{

getline(in, s);

qs.push\_back(s);

}

questions.emplace\_back(CHECK\_BOX, str, qs);

break;

}

default:

{

cout << "Error in reading in question type" << endl;

break;

}

}

}

in.close();

answers = vector<vector<string>>((int)questions.size(), vector<string>());

}

Form::Form(string sname, vector<string> a)

{

//constructor for a form that already has the answers

this->formName = sname;

this->answers = vector<vector<string>> (a.size(), vector<string>());

for (int i = 0; i < a.size(); i++)

{

answers[i].push\_back(a[i]);

}

};

void Form::takeInput()

{

cin.ignore();

for (int i = 0; i < questions.size(); i++)

{

cout << questions[i].getQuestion() << endl;

string ans;

int choice;

switch(questions[i].getType())

{

//just take the short answer response

case SHORT\_ANSWER:

{

getline(cin, ans);

answers[i].push\_back(ans);

break;

}

//take in only one choice

case MULTIPLE\_CHOICE:

{

cin >> choice;

answers[i].push\_back(questions[i].getExtraQ(--choice));

break;

}

//take in many choices until -1 is reached

case CHECK\_BOX:

{

cin >> choice;

while (choice != -1)

{

answers[i].push\_back(questions[i].getExtraQ(--choice));

cin >> choice;

}

break;

}

default:

{

break;

}

}

}

};

string Form::getAnswer(int i)

{

//output the answers

string str = "";

for (int j = 0; j < answers[i].size(); j++)

{

str += answers[i][j] + " \n"[j+1==(int)answers[i].size()];

}

return str;

};

void Form::outputAnswers()

{

//iterate through each

for (int i = 0; i < answers.size(); i++)

{

cout << getAnswer(i) << endl;

}

};

//all basic getters

int Form::getQuestionType(int i)

{

return questions[i].getType();

};

int Form::getQuestionSize()

{

return this->questions.size();

};

int Form::getAnswerSize()

{

return this->answers.size();

}