Jamie Cartwright Marks

**Grade 12 IT PAT 2022 Phase 1**

**Scenario and scope:**

The topic of this IT PAT is online learning, and the creation of an application which aids in the online learning process. I decided after careful consideration that the option involving the creation of a piece of software that is used in the process of scheduling online or, more specifically, tutoring lessons was the most practical and fitting task for me, when compared to the other possible takes on online learning mentioned in the PAT outline. I chose this specific option because I know first-hand how the market for external learning and tutoring is expanding and is a useful supplementary resource for many people including a considerable amount of my peers and friends, thus creating this program would help people organise and use the newly popularised online learning process via tutoring using my software solution.

The task or purpose of this project involves creating a piece of software in, the Delphi RAD IDE, that acts as a platform for both students and teachers to interact and schedule online learning/tutoring sessions, in an easy user-friendly manner thus aiding both learners and teachers in the online learning scenario and solving the issue of scheduling discrepancies and confusion involving online lessons. It also involves creating a well thought out, well-designed and user-friendly interface to access the scheduling software.

The solution to this task for me will be to create a lightweight, easy to use piece of software for both teachers and students to utilise in scheduling and managing online learning and tutoring sessions. The software assumes that communication about session dates and times is done outside of the software (between student and teacher) and the software is used just for the scheduling of these sessions. This system would work via the use of unique, user determined usernames and passwords which are stored securely in the database along with the associated necessary information about each student and teacher. The teacher will be the main entity in the system, once logged in to their account, the teacher will schedule a meeting using the software, along with a plethora of other features like creating queries regarding their schedule all within the software, as well as resourceful html pages with links. This meeting will then be visible to the student once they log in to the software, thereafter, the student can perform actions like sorting through their upcoming sessions, changing log in details, accessing extra online material and other queries relating to their upcoming sessions. The information is entered by both teachers and students, but the teachers are the main entities in actually scheduling the lessons and confirming that the learner has attended the lessons for reason of reliability.

The scope of my project will focus mostly on the main sub-topic of the PAT which is a scheduling software for online lessons and tutoring to aid in the new trend of online learning, therefore the software will be well thought out and easy to use from all aspects and focused on the main purpose of scheduling lessons. The program will also involve a well-designed and correctly utilised database via Microsoft access. The database will be secure and have detailed but necessary information that is used within the software to help increase both the security and ease of access for all entities involved in the use of the software.

**User requirements:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Teacher** | **Learner** | **Database administrator** |
| **Role:** | * *To view scheduled lessons and schedule lessons with learners* | * *To view scheduled lessons and edit information about those lessons* | * *To delete, insert and edit records in the associated database to this program* * *To ensure the efficiency of the database and to perform maintenance on the database* |
| **Activity:** | * *View future and past lessons* * *Edit teacher log in details* * *Sort and manipulate lesson data* * *Access learner and teacher resources* * *Change activity state* * *Schedule new sessions with associated learner* | * *Edit log in details* * *View past and future scheduled lessons* * *To access the software’s educational resources* * *To check if attendance for lessons has been registered* * *To access meeting links for online lessons* | * *Delete records in the database* * *Edit records in the database* * *Insert records in the database* * *Change the fundamental structure of the database* * *Add queries and filters* * *Perform routine maintenance on the database* * *Add or remove records and associated data* * *View the database without censorship (other than passwords)* |
| **Limitations:** | * *Cannot change any of the source code or front-end of the program* * *Cannot control learner accounts or activity* * *Cannot edit the database in the program* * *Limited data access* | * *Cannot change any of the source code or front-end of the program* * *Cannot control learner accounts (that are not their own) or activity* * *Cannot control teacher activity and reliability* | * *Cannot change any of the source code or front-end of the program* * *Cannot monitor the activity of the program* * *Limited to only controlling the database of the program and associated data* * *Cannot change or view learner/teacher passwords* |

**Description of flow diagram:**

**TEACHER LOGIN**:

Main scheduling page launched

Splash screen launched

Home page launched

Log in as previous user or new user/register

Can edit log in details

Schedule a new session

Queries preformed; resources accessed

**LEARNER:**

Main scheduling page launched

Splash screen launched

Log in as previous user or new user/register

Home page launched

Can access learner resources if necessary

Can perform queries

Can change log in details if necessary

**DATABASE ADMINISTRATION:**

Make changes and view all tables, as well as run queries and edit data in tblTutors and tblStudents

Log in as database administrator with encrypted passcode and username

Splash screen launched

Home page launched

Can change db admin passcode if desired

**DATA STRUCTURES:**

**-DATABASE (TechnoTutorsDB):**

tblStudents:

Application

Description automatically generated with medium confidence

tblTutors:

A picture containing graphical user interface

Description automatically generated

tblSessions:

Text

Description automatically generated

Relationships:

Diagram

Description automatically generated

-**CLASS DIAGRAMS:**

Student Class:

**TStudent**

- **fUsername**: string

- **fPassword**: string

- **fFirstName**: string

- **fSurname**: string

- **fAtttendedlessons**: integer

-**fGender**: character

**Method Scopes:**

Constructor methods used to initialise object of the class and assign values to attributes

+ <<constructor>> **Create**(sUsername, sPassword, sFirstname, sSurname: string) [overloaded method pt 1]

+ <<constructor>> **Create**(sUsername, sPassword, sFirstname, sSurname: string; iAttendedlessons: integer; cGender: string) [overloaded method pt 2]

+ **GetUsername**: string

+ **GetFirstname**: string

+ **GetSurname**: string

+ **GetNumAttended**: integer

+ **GetGender**: character

+ **SetPassword**(sNewPass: string)

+ **SetUsername**(sUsername: string)

+ **IncSessions**

+ **ComparePass**(sPass: string): boolean

+ **ToString**: string

Constructor methods used to initialise object of the class and assign values to attributes, overloaded to be used for new or existing user

Accessor functions:

-returns object username

-

-returns object first name

-

-returns object number of sessions attended

-

-returns object surname

-

-returns object gender

-

-increments attended sessions by 1

-

-assigns username to object attribute

-

Mutator procedures:

-assigns password to object attribute

-

Auxiliary methods:

-function that compares inputted password to fPassword attribute

-function that returns an easy to read, informative summary of the object’s info, using the attributes

-**TEXT FILES:**

Text files will be used most noticeably in two places within this program. The first application of text files will be in the database admin section of the program. The database admin will be required to enter a password in order to access database admin controls. This password will be stored locally in a text file named “EncryptedAdminPassword.txt”, as the name suggests, this text file will contain the encrypted (from plaintext to encrypted text) of the database admin’s password. The encryption method used will be an adaptation of my 2021 grade 11 encryption/decryption PAT. When a user attempts to log in as a database admin, they will be called upon to input the plaintext version of the encrypted password saved in the text file. The encrypted password will then be read from the text file (via code) decrypted and compared to the entered password to determine whether the user is indeed a database admin and subsequently either denies or allows access to the database admin. This password can be changed within the program, as a database admin, when the password is changed, the new encrypted password will be saved in the same “EncryptedAdminPassword.txt” text file.

A text file will also be used in the main form for teachers. This text file will assume the name” Subjects.txt”. This text file will be used on the onShow event of the teacher form to populate the subjects in the text file, into a combo box, so that the teacher can choose a valid subject that complies with the text file (the acceptable subjects that Technotutors offer). The reading and subsequent loading of the text files will be done via code construct, using the logical text file under the name of “myFile”.

-**ARRAYS:**

An array with the name arrSubjects of type string will be used in both the student and tutor form of the program. The array will be accessed on the onShow event of both the student and tutor form to be populated. The array will be populated using the “Subjects.txt” text file which contains all the valid subjects that can be taught within Technotutors. I chose an array for this because of the nature of a text file (having a “key” e.g., numbers 1 to 10, and a value e.g., “Mathematics”), because the subjects are saved in the Technotutors database in the form of a SubjectID, which is a number that relates to a specific subject. In other words, it makes it easy to access the full subject name from the subject ID using <*arrSubjects*>[<*SubjectID*>]. This array is then used in the rich edit output of the student and tutor forms, to instead of a SubjectID, output a full subject name, for ease of use and understanding of the user.

**GUI**

*Splash screen:*

*A picture containing text, indoor, keyboard, computer

Description automatically generated*

Opens a link to an html page (in the browser) explaining what the tag introduces

*Main screen:*

Closes the program

*Graphical user interface, text, email

Description automatically generated*

Presents log-in for whichever option is chosen (hints about what each option entails are offered when hovering above each button)

Changes colour when hovering over different buttons

*Graphical user interface, application

Description automatically generatedLog-in page:*

User entered unique username used for both logging in and registering, validated fully via code

For users that have used the software before

For users that have not used the software before, once pressed the user will be greeted with some more options to make an account

User entered password username used for both logging in and registering, validated fully via code

Graphical user interface, table

Description automatically generated*Main form (similar for both students and teachers):*

Dbgrid showing database data

Schedule a new session (only available for teachers)

Change login details

Output area

‘Hints’ explaining the button function show up when hovering your curser over any button for all froms

Session controls (slightly different in learner form)

**Graphical user interface, application

Description automatically generated***Database Admin form:*

Dbgrids showing tutor info and learner info respectively and to show sql results

Database queries and SQL’s to be used by database admin

**DATA INPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| Source of Input | Data type | Format | Component used |
| Keyboard | String | Information about sessions from teacher | Edit box |
| Text file | String | Subjects for combo box for scheduling | Combo box |
| Keyboard | String | Passwords and updated information | Edit box |
| Database | String/Integer/Date time | All necessary information about teachers and students as well as sessions linking them | Rich edit |

**INPUT VALIDATION:**

|  |  |  |
| --- | --- | --- |
| **Data type** | **Input** | **Error message** |
| String | Username, checks if the username is unique according to the database of teachers or students dependant on context (when registering) | **‘**The username entered is already in use’ |
| Null | Lack thereof, If any important information like username, password or scheduling information is left out | 'Please enter a name'  Or  'Please enter a surname' |
| Integer | 0, 1 or 2 (the radio group must have an index of 0, 1 or 2 otherwise a gender has not been chosen) | ‘You have not chosen a gender’ |
| Date time | Date and time via date time picker and spin edit, if the date has passed or the time for the selected day has passed an error message is presented. | 'The selected date has passed, please select another one'  Or  'The selected time has already passed, please choose an applicable time' |

**Data Processing**

What Processing will be done:

* Validating names and show appropriate error messges
* Outputting upcoming sessions
* Decryption
* Removing non-letter characters from the inputted message
* Encryption
* Showing all sessions in a dbgrid according to created class
* Finding the average sessions attended
* Saving updated log-in information and adding it to database
* Checking password validity and show appropriate error messages

**How the processing will be done:**

Algorithms:

1. Algorithm 1 (Validating names):

***The purpose of this algorithm*:** To retrieve the name and surname of the user and check fully if it can be used for a new account

***Algorithm:***

*sName is the inputted name*

sSurname *is the inputted srname*

var

sName, sSurname: string;

iCapName, iLowName, iCapSurname, iLowSurname, I: integer;

begin

// initialization

result := True;

sName := edtName.Text;

sSurname := edtSurname.Text;

iCapName := 0;

iLowName := 0;

iCapSurname := 0;

iLowSurname := 0;

**// presence check**

if sName = '' then

begin

MessageDlg('Please enter a name', mtError, [mbOk], 0);

result := False;

exit;

end;

if sSurname = '' then

begin

MessageDlg('Please enter a surname', mtError, [mbOk], 0);

result := False;

exit;

end;

**// counting the number of lowercase and uppercase letters in the surname and lastname**

for I := 1 to length(sName) do

begin

if sName[I] in ['A' .. 'Z'] then

begin

inc(iCapName);

end;

if (sName[I] in ['a' .. 'z']) or (sName[I] in ['''', '-', '.']) then

begin

inc(iLowName);

end;

end;

for I := 1 to length(sSurname) do

begin

if sSurname[I] in ['A' .. 'Z'] then

begin

inc(iCapSurname);

end;

if (sSurname[I] in ['a' .. 'z']) or (sSurname[I] in ['''', '-', '.']) then

begin

inc(iLowSurname);

end;

end;

**// checking if the names are capitilised**

if iCapName = 0 then

begin

MessageDlg('Please capitilise your name correctly', mtError, [mbOk], 0);

result := False;

exit;

end;

if iCapSurname = 0 then

begin

MessageDlg('Please capitilise your surname correctly', mtError, [mbOk], 0);

result := False;

exit;

end;

// checking if there are any foreign characters in the names

if ((iLowName + iCapName) <> length(sName)) then

begin

MessageDlg('The entered name has (a) foreign character(s) in it', mtError,

[mbOk], 0);

result := False;

exit;

end;

if ((iLowSurname + iCapSurname) <> length(sSurname)) then

begin

MessageDlg('The entered surname has (a) foreign character(s) in it',

mtError, [mbOk], 0);

result := False;

exit;

end;

end;

1. Algorithm 2 (Upcoming sessions output):

**The purpose of this algorithm:** To output all of the future scheduled sessions to the student using the student object to find related session data

**Algorithm:**

*iSubject is the subject number which correlates to the text file*

var

iSubject: integer;

begin

FormatRichedit(redStudent); **// formatting richedit**

redStudent.Lines.Add(#9 + #9 + 'UPCOMING SESSIONS:'); **// title**

redStudent.Lines.Add('|Tutor Username|' + #9 + '|Session Date|' + #9 +

'|Subjectname|' + #9 + '|Meeting Link|' + #9 + '|Session Time|');

**// headings**

redStudent.Lines.Add( // line to seperate info from headings

'----------------------------------------------------------------------------------------------------------------------------');

conTechno.dbconnection; **// database connection**

tblSessions.First; // begining of database

while not tblSessions.Eof do

begin

if (tblSessions['StudentUsername'] = objStudent.GetUsername) AND **// if the client is the wanted client and the date of the session is later than todays date**

(tblSessions['SessionDate'] > Today) then

**// all session records relation to a certain student**

begin

iSubject := tblSessions['SubjectID']; **// reading subject number to extract** **subject name from array later on**

if tblSessions['Meetinglink'] = null then

redStudent.Lines.Add(tblSessions['TutorUsername'] + #9 + datetostr

(tblSessions['SessionDate']) + #9 + arrSubjects[iSubject]

+ #9 + 'No link provided' + #9 + TimeToStr

(tblSessions['SessionTime'])) **// output if meeting link isnt provided**

else

redStudent.Lines.Add(tblSessions['TutorUsername'] + #9 + datetostr

(tblSessions['SessionDate']) + #9 + arrSubjects[iSubject]

+ #9 + tblSessions['Meetinglink'] + #9 + TimeToStr

(tblSessions['SessionTime'])); **// outputed if meeting link is provided**

end; // next record in db

tblSessions.Next;

end;

end;

1. Algorithm 3 (Decryption):

***The purpose of this algorithm*:** To decrypt database admin password stored in text file to check if the entered admin password is correct

***Algorithm:***

*m is the loop counter variable*

*iIndex is the position of desired letter in key*

*sKey is the key used to decrypt, same as the key used to encrypt the data*

*sOutput is the plaintext output*

var

m, iIndex: integer;

sOutput: string;

const

sValid: string = 'abcdefghijklmnopqrstuvwxyz ';

begin

// decrypting according to the key

for m := 1 to length(EncryptedText) do **// m assinged the value one to the** length of the cyphered text

begin

iIndex := pos(EncryptedText[m], sKey); **// saved index is the position of the** **letter at point m of the encrypted text, in the key**

sOutput := sOutput + sValid[iIndex]; **// above index used in normal** alpahabet (svalid) to decrypt

end;

result := sOutput;

end;

1. Algorithm 4 (Checking password validity and show appropriate error messages

**The purpose of this algorithm:** A multipurpose procedure used to check if the user entered password in the main page is valid according to the password restrictions outlined in the program. The algorithm checks thoroughly and efficiently whether the password doesn’t fit any single restriction, and will show an appropriate descriptive error message

**Algorithm:**

*iCapLetterCount* *is the counted number of capital letters in the entered password*

*iNumberCount* *is the number of number characters entered in the user entered password*

*sPassword is the user entered password from the main form*

*iLowerLetterCount is is the counted number of lowercase letters in the entered password*

iCapLetterCount := 0;

iNumberCount := 0;

iLowerLetterCount := 0;

result := True;

**// checking the length of the password is valid**

if length(sPassword) <= 8 then

begin

MessageDlg('Your password is too short, it must be 8 or more characters',

mtError, [mbOk], 0);

result := False;

exit;

end;

if length(sPassword) > 20 then

begin

MessageDlg(

'Your password is too long, it may not be more than 20 characters',

mtError, [mbOk], 0);

result := False;

exit;

end;

**// counting the number of letters and numbers in the password**

for I := 1 to length(sPassword) do

begin

if sPassword[I] in ['a' .. 'z'] then

begin

inc(iLowerLetterCount);

end

else if sPassword[I] in ['A' .. 'Z'] then

begin

inc(iCapLetterCount);

end

else if sPassword[I] in ['1' .. '9'] then

begin

inc(iNumberCount);

end;

end;

if (iCapLetterCount + iLowerLetterCount) < 1 then

begin

MessageDlg('Your password is must have atleast one letter', mtError,

[mbOk], 0);

result := False;

exit;

end;

if iNumberCount < 1 then

begin

MessageDlg('Your password must have atleast one number', mtError, [mbOk],

0);

result := False;

exit;

end;

if iCapLetterCount < 1 then

begin

MessageDlg('Your password must have atleast 1 uppercase letter', mtError,

[mbOk], 0);

result := False;

exit;

end;

if (iLowerLetterCount + iCapLetterCount + iNumberCount) <> length(sPassword)

then

begin

MessageDlg('Your password may only consist of letters and number', mtError,

[mbOk], 0);

result := False;

exit;

end;

1. Algorithm 5 (finding the average, max and min sessions attended using code construct and the database):

**The purpose of this algorithm:** To go through all the users in the tblStudents table and to thereafter find the average number of sessions attended by the students using a total and count variable and then subsequently finding the minimum and maximum sessions attended (two algorithms combined for space reasons)

**Algorithm:**

*iCount is the counted number of students in the database used to find the average*

*rTotal is the total sessions attended by all students in the database*

*iMax is the maximum attended sessions recorded*

*iMin is the minimum attended sessions recorded*

**// calculating average attended lessons**

tblStudents.first; **// start of database records**

iCount := 0; // instantiation of icount

while not tblSTudents.Eof do

begin

if tblSTudents['AttendedSessions'] <> null then

**// if an attended value is entered**

begin

rTotal := rTotal + tblSTudents['AttendedSessions'];

**// add record's attended value to total**

inc(iCount);

end;

tblSTudents.Next;

end;

showmessage('The average student attended lessons is: ' + floattostrf

(rTotal / iCount, fffixed, 5, 2) + ' lessons');

**// output + finding average**

iMin := 10000; **// high max value so first real max is set to the max**

iMax := 0; **// low min value so first real min is set to the min**

tblSTudents.first;

while not tblSTudents.Eof do

begin

if tblSTudents['AttendedSessions'] > iMax then **// checking for new max**

begin

iMax := tblSTudents['AttendedSessions'];

end;

if tblSTudents['AttendedSessions'] < iMin then **// checking for new min**

begin

iMin := tblSTudents['AttendedSessions'];

end;

tblSTudents.Next

end;

showmessage('The max attended lessons is: ' + inttostr(iMax)); **// output**

showmessage('The min attended lesson is: ' + inttostr(iMin)); **// output**

**Data output**

|  |  |  |
| --- | --- | --- |
| **What** | **Format** | **Component** |
| New user information | String, Character, integer | Database |
| Encrypted database password | String | Textfile |
| Decrypted database password | String | Message box, TextFile |
| Database query results for database admin | String | Richedit |
| Information about learners, tutors and sessions | String, integer, character, date/time | dbGrid |