# Annotated code

# Globalsetup form

unit Globalsetup;

interface

uses

sysutils;

var

globalconnectionstring: string;

globalpasslevel: integer;

globalerrorcode: array [0 .. 2] of string;

globalusername: string;

globaldbpassword: string;

globalgamelevel: integer;

const

LexiconSize: array [1 .. 12] of integer = (333, 333, 304, 304, 257, 257, 304,

304, 252, 252, 228, 228);

procedure setconnectionstring(x: string);

procedure setglobalpasslevel(a: integer);

procedure setglobalerrorcode(filename: string);

procedure setglobalusername(x: string);

procedure setglobaldbpassword(x: string);

procedure setglobalgamelevel(a: integer);

implementation

uses

Login;

// Procedure to set global var globalconnectionstring

procedure setconnectionstring(x: string);

begin

globalconnectionstring := x;

end;

// Procedure to set global var globalpasslevel

procedure setglobalpasslevel(a: integer);

begin

globalpasslevel := a;

end;

// Procedure to set global array globalerrorcode

procedure setglobalerrorcode(filename: string);

var

ErrorCode: textfile;

text: string;

i: integer;

begin

assignfile(ErrorCode, filename);

reset(ErrorCode);

for i := 0 to 2 do

begin

readln(ErrorCode, text);

globalerrorcode[i] := text;

end;

closefile(ErrorCode);

end;

// Procedure to set global var globalusername

procedure setglobalusername(x: string);

begin

globalusername := x;

end;

// Procedure to set global var globalpassword

procedure setglobaldbpassword(x: string);

begin

globaldbpassword := x;

end;

// Procedure to set global var globalgamelevel

procedure setglobalgamelevel(a: integer);

begin

globalgamelevel := a;

end;

end.

# Login form

unit Login;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Mask, Vcl.ExtCtrls,

Data.DB, Data.Win.ADODB, Vcl.Imaging.pngimage, Vcl.Buttons, Vcl.OleServer,

SpeechLib\_TLB, DateUtils;

type

TfrmLogin = class(TForm)

edtUsername: TEdit;

edtPassword: TMaskEdit;

adotblUsers: TADOTable;

imgLogin: TImage;

lblLogin: TLabel;

imgExit: TImage;

lblExit: TLabel;

lblLoginBtn: TLabel;

pnlLogin: TPanel;

imgSpaggy: TImage;

imgSpeechBubble: TImage;

lblSpeechBubble: TLabel;

procedure FormCreate(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure edtPasswordKeyPress(Sender: TObject; var Key: Char);

procedure imgLoginClick(Sender: TObject);

procedure imgExitClick(Sender: TObject);

procedure edtUsernameKeyPress(Sender: TObject; var Key: Char);

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmLogin: TfrmLogin;

implementation

{$R \*.dfm}

uses

Globalsetup, Mainmenu, Error;

procedure TfrmLogin.FormCreate(Sender: TObject);

var

DatabaseName: textfile;

text: string;

dlg: Topendialog;

filename: string;

connection: string;

connectstring: string;

password: string;

begin

frmLogin.Position := poscreencenter;

// Open filelocation.txt and store the database location contained within it

assignfile(DatabaseName, 'filelocation.txt');

reset(DatabaseName);

readln(DatabaseName, text);

closefile(DatabaseName);

// If the file is a database and exists then add the location to the connection string

if (fileexists(text) and (extractfileext(text) = '.accdb')) then

connection := text

else

// If the database doesnt exist then create open dialog to allow user to choose location

begin

dlg := Topendialog.Create(nil);

try

if dlg.Execute() then

begin

filename := dlg.filename;

showmessage(filename);

connection := (filename);

// Rewrite filelocation.txt to store the new database location

rewrite(DatabaseName);

writeln(DatabaseName, connection);

// If the user closes the dialog box then inform them that no selection has been made

showmessage('No file has been selected.')

finally

dlg.Free();

end;

end;

// Create input dialog to enter database password and append the connection string

password:= InputBox('Admin: Database Password', #31'Enter the password for the database:','');

setglobaldbpassword(password);

// Create connection string. This will currently only work for .accdb files

connectstring := 'provider=microsoft.ace.oledb.16.0;data source =' +

connection + ';Jet OLEDB:Database Password=' + password + ';';

// Run procedure in Globalsetup unit to save global connection string

// This can then be used by other forms/units

setconnectionstring(connectstring);

adotblUsers.Active := false;

adotblUsers.ConnectionString := connectstring;

adotblUsers.TableName := 'tblUsers';

end;

procedure TfrmLogin.FormShow(Sender: TObject);

begin

// When form is loaded reset input boxes

edtUsername.text := '';

edtPassword.text := '';

edtUsername.SetFocus;

// Attempt to connect to database ..

try

adotblUsers.Active := true;

// If this raises an error (eg doesnt exist) then show custom error dialog

except

setglobalerrorcode('dbpassworderror.txt');

frmError.Show;

end;

end;

procedure TfrmLogin.imgExitClick(Sender: TObject);

begin

Application.Terminate();

Exit;

end;

procedure TfrmLogin.imgLoginClick(Sender: TObject);

var

DailyStreak: integer;

begin

// If inputs are blank then show custom error dialog

if (length(edtUsername.text) = 0) or (length(edtPassword.text) = 0) then

begin

setglobalerrorcode('loginblankerror.txt');

frmError.Show;

end

else

begin

// If inputted username doesnt exist in the database then show custom error dialog

if not adotblUsers.Locate('Username', edtUsername.text, []) then

begin

setglobalerrorcode('loginerror.txt');

frmError.Show;

end

else if edtPassword.text = adotblUsers['Password'] then

begin

// Set admin rights and username in the Globalsetup unit

setglobalpasslevel(adotblUsers['AdminRights']);

setglobalusername(edtUsername.text);

adotblUsers.Locate('Username', edtUsername.text, []);

// Get daily streak from database and increment if last login was yesterday, otherwise reset

DailyStreak := adotblUsers.FieldByName('DailyStreak').AsInteger;

if adotblUsers.FieldByName('LastLogin').AsDateTime = Yesterday then

inc(DailyStreak)

else if adotblUsers.FieldByName('LastLogin').AsDateTime <> date then

DailyStreak := 0;

// Post amendments to the database and refresh

adotblUsers.Edit;

adotblUsers.FieldByName('DailyStreak').AsInteger := DailyStreak;

adotblUsers.FieldByName('LastLogin').AsDateTime := date;

adotblUsers.Post;

adotblUsers.Refresh;

frmMainmenu.Show;

frmLogin.Hide;

end

else

// If inputted password is incoorrect then show custom error dialog

begin

setglobalerrorcode('loginerror.txt');

frmError.Show;

end;

end;

end;

procedure TfrmLogin.edtPasswordKeyPress(Sender: TObject; var Key: Char);

begin

if ord(Key) = VK\_RETURN then

begin

Key := #0; // prevent beeping

imgLoginClick(Sender);

end;

end;

procedure TfrmLogin.edtUsernameKeyPress(Sender: TObject; var Key: Char);

begin

// If the user presses the return key then move focus to next input (tab)

if ord(Key) = VK\_RETURN then

begin

Key := #0; // prevent beeping

edtPassword.SetFocus;

end;

end;

end.

closefile(DatabaseName);

end

else

// If the user closes the dialog box then inform them that no selection has been made

showmessage('No file has been selected.')

finally

dlg.Free();

end;

end;

// Create input dialog to enter database password and append the connection string

password:= InputBox('Admin: Database Password', #31'Enter the password for the database:','');

setglobaldbpassword(password);

// Create connection string. This will currently only work for .accdb files

connectstring := 'provider=microsoft.ace.oledb.16.0;data source =' +

connection + ';Jet OLEDB:Database Password=' + password + ';';

// Run procedure in Globalsetup unit to save global connection string

// This can then be used by other forms/units

setconnectionstring(connectstring);

adotblUsers.Active := false;

adotblUsers.ConnectionString := connectstring;

adotblUsers.TableName := 'tblUsers';

end;

procedure TfrmLogin.FormShow(Sender: TObject);

begin

// When form is loaded reset input boxes

edtUsername.text := '';

edtPassword.text := '';

edtUsername.SetFocus;

// Attempt to connect to database ..

try

adotblUsers.Active := true;

// If this raises an error (eg doesnt exist) then show custom error dialog

except

setglobalerrorcode('dbpassworderror.txt');

frmError.Show;

end;

end;

procedure TfrmLogin.imgExitClick(Sender: TObject);

begin

Application.Terminate();

Exit;

end;

procedure TfrmLogin.imgLoginClick(Sender: TObject);

var

DailyStreak: integer;

begin

// If inputs are blank then show custom error dialog

if (length(edtUsername.text) = 0) or (length(edtPassword.text) = 0) then

begin

setglobalerrorcode('loginblankerror.txt');

frmError.Show;

end

else

begin

// If inputted username doesnt exist in the database then show custom error dialog

if not adotblUsers.Locate('Username', edtUsername.text, []) then

begin

setglobalerrorcode('loginerror.txt');

frmError.Show;

end

else if edtPassword.text = adotblUsers['Password'] then

begin

// Set admin rights and username in the Globalsetup unit

setglobalpasslevel(adotblUsers['AdminRights']);

setglobalusername(edtUsername.text);

adotblUsers.Locate('Username', edtUsername.text, []);

// Get daily streak from database and increment if last login was yesterday, otherwise reset

DailyStreak := adotblUsers.FieldByName('DailyStreak').AsInteger;

if adotblUsers.FieldByName('LastLogin').AsDateTime = Yesterday then

inc(DailyStreak)

else if adotblUsers.FieldByName('LastLogin').AsDateTime <> date then

DailyStreak := 0;

// Post amendments to the database and refresh

adotblUsers.Edit;

adotblUsers.FieldByName('DailyStreak').AsInteger := DailyStreak;

adotblUsers.FieldByName('LastLogin').AsDateTime := date;

adotblUsers.Post;

adotblUsers.Refresh;

frmMainmenu.Show;

frmLogin.Hide;

end

else

// If inputted password is incoorrect then show custom error dialog

begin

setglobalerrorcode('loginerror.txt');

frmError.Show;

end;

end;

end;

procedure TfrmLogin.edtPasswordKeyPress(Sender: TObject; var Key: Char);

begin

if ord(Key) = VK\_RETURN then

begin

Key := #0; // prevent beeping

imgLoginClick(Sender);

end;

end;

procedure TfrmLogin.edtUsernameKeyPress(Sender: TObject; var Key: Char);

begin

// If the user presses the return key then move focus to next input (tab)

if ord(Key) = VK\_RETURN then

begin

Key := #0; // prevent beeping

edtPassword.SetFocus;

end;

end;

end.

# Mainmenu form

unit Mainmenu;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Imaging.pngimage, Vcl.ExtCtrls,

Vcl.StdCtrls, Data.DB, Data.Win.ADODB, Vcl.ComCtrls;

type

TfrmMainmenu = class(TForm)

imgAdminOptions: TImage;

imgSettings: TImage;

imgLogout: TImage;

lblMainMenu: TLabel;

lblLogout: TLabel;

lblAdminOptions: TLabel;

lblSettings: TLabel;

shpProfilePic: TShape;

imgProfilePic: TImage;

lblUsername: TLabel;

adotblUsers: TADOTable;

adoqryUsers: TADOQuery;

lblScore: TLabel;

lblDailyStreak: TLabel;

lblLastLogin: TLabel;

timerClock: TTimer;

lblTime: TLabel;

lblDate: TLabel;

pnlMain: TPanel;

imgLvl1: TImage;

imgLvl2: TImage;

imgLvl3: TImage;

imgLvl9: TImage;

imgLvl10: TImage;

imgLvl11: TImage;

imgLvl4: TImage;

imgLvl12: TImage;

imgLvl5: TImage;

imgLvl6: TImage;

imgLvl7: TImage;

imgLvl8: TImage;

pb1: TProgressBar;

pb2: TProgressBar;

pb3: TProgressBar;

pb4: TProgressBar;

pb5: TProgressBar;

pb6: TProgressBar;

pb7: TProgressBar;

pb8: TProgressBar;

pb9: TProgressBar;

pb10: TProgressBar;

pb11: TProgressBar;

pb12: TProgressBar;

imgSpaggy: TImage;

procedure FormCreate(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure imgAdminOptionsClick(Sender: TObject);

procedure imgLogoutClick(Sender: TObject);

procedure imgSettingsClick(Sender: TObject);

procedure timerClockTimer(Sender: TObject);

procedure imgLvl1Click(Sender: TObject);

procedure imgLvl2Click(Sender: TObject);

procedure imgLvl3Click(Sender: TObject);

procedure imgLvl4Click(Sender: TObject);

procedure imgLvl5Click(Sender: TObject);

procedure imgLvl6Click(Sender: TObject);

procedure imgLvl7Click(Sender: TObject);

procedure imgLvl8Click(Sender: TObject);

procedure imgLvl9Click(Sender: TObject);

procedure imgLvl10Click(Sender: TObject);

procedure imgLvl11Click(Sender: TObject);

procedure imgLvl12Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmMainmenu: TfrmMainmenu;

CurrentDir: string;

ImageDir: string;

UserProgress: array [1 .. 12] of integer;

implementation

{$R \*.dfm}

uses

Login, Globalsetup, AdminOptions, Settings, Error, Level;

procedure TfrmMainmenu.FormClose(Sender: TObject; var Action: TCloseAction);

begin

frmLogin.Show;

end;

procedure TfrmMainmenu.FormCreate(Sender: TObject);

begin

frmMainmenu.Scaled := true;

frmMainmenu.Position := poscreencenter;

end;

procedure TfrmMainmenu.FormShow(Sender: TObject);

var

i, x: integer;

pb: TProgressBar;

begin

// Display admin options to the user if they have administrator rights

case globalpasslevel of

0:

begin

frmMainmenu.Caption := 'Spaggy''s Spelling';

imgAdminOptions.Visible := false;

lblAdminOptions.Visible := false;

end;

1:

begin

frmMainmenu.Caption := 'Spaggy''s Spelling (Admin)';

imgAdminOptions.Visible := true;

lblAdminOptions.Visible := true;

end;

end;

with adoqryUsers do

begin

Active := false;

ConnectionString := globalconnectionstring;

SQL.Clear;

// Run SQL to gather all fields linked to the username

SQL.Add('Select \* from tblUsers where Username = "' +

globalusername + '";');

Active := true;

// Set focus to the record

First;

CurrentDir := GetCurrentDir;

// Set image directory (contained within program folder) from main program directory

// (SizeOf(Pointer)\*8) returns the windows platform at runtime (x86/x64)

ImageDir := StringReplace(CurrentDir, 'Win' + (inttostr(SizeOf(Pointer) \* 8)

) + '\Debug', 'Profile Pictures\', [rfIgnoreCase]);

// Load the user's profile picture to imgProfilePic

imgProfilePic.Picture.LoadFromFile

(ImageDir + adoqryUsers['ProfilePicture']);

// Load the user's profile info and display within corresponding labels

lblScore.Caption := 'Score: ' + floattostr(adoqryUsers['Score']);

lblDailyStreak.Caption := 'Daily Streak: ' +

floattostr(adoqryUsers['DailyStreak']);

lblLastLogin.Caption := 'Last Login: ' +

datetostr(adoqryUsers['LastLogin']);

// Store user's level progress to local array

for i := 1 to 12 do

UserProgress[i] := FieldByName('Level' + inttostr(i)).AsInteger;

end;

lblUsername.Caption := globalusername;

for x := 1 to 2 do

// Set positions for each progress bar in the main menu according to level progress

for i := 1 to 12 do

begin

pb := TProgressBar(frmMainmenu.FindComponent('pb' + inttostr(i)));

pb.Position := trunc((UserProgress[i] / LexiconSize[i]) \* 100);

// Set progress bar colour to green if full (completed level)

if pb.Position = 100 then

pb.State := pbsNormal;

end;

end;

procedure TfrmMainmenu.imgAdminOptionsClick(Sender: TObject);

begin

frmAdminoptions.Show;

frmMainmenu.Hide;

end;

procedure TfrmMainmenu.imgLogoutClick(Sender: TObject);

begin

frmLogin.Show;

frmMainmenu.Hide;

end;

procedure TfrmMainmenu.imgLvl10Click(Sender: TObject);

begin

// Grant access to level if incomplete and previous level has been partially complete

if ((pb9.Position > 10) and (pb10.Position < 100)) then

begin

setglobalgamelevel(10);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl11Click(Sender: TObject);

begin

if ((pb10.Position > 10) and (pb11.Position < 100)) then

begin

setglobalgamelevel(11);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl12Click(Sender: TObject);

begin

if ((pb11.Position > 10) and (pb12.Position < 100)) then

begin

setglobalgamelevel(12);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl1Click(Sender: TObject);

begin

if pb1.Position < 100 then

begin

setglobalgamelevel(1);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl2Click(Sender: TObject);

begin

if ((pb1.Position > 10) and (pb2.Position < 100)) then

begin

setglobalgamelevel(2);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl3Click(Sender: TObject);

begin

if ((pb2.Position > 10) and (pb3.Position < 100)) then

begin

setglobalgamelevel(3);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl4Click(Sender: TObject);

begin

if ((pb3.Position > 10) and (pb4.Position < 100)) then

begin

setglobalgamelevel(4);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl5Click(Sender: TObject);

begin

if ((pb4.Position > 10) and (pb5.Position < 100)) then

begin

setglobalgamelevel(5);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl6Click(Sender: TObject);

begin

if ((pb5.Position > 10) and (pb6.Position < 100)) then

begin

setglobalgamelevel(6);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl7Click(Sender: TObject);

begin

if ((pb6.Position > 10) and (pb7.Position < 100)) then

begin

setglobalgamelevel(7);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl8Click(Sender: TObject);

begin

if ((pb7.Position > 10) and (pb8.Position < 100)) then

begin

setglobalgamelevel(8);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl9Click(Sender: TObject);

begin

if ((pb8.Position > 10) and (pb9.Position < 100)) then

begin

setglobalgamelevel(9);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgSettingsClick(Sender: TObject);

begin

frmSettings.Show;

frmMainmenu.Hide;

end;

procedure TfrmMainmenu.timerClockTimer(Sender: TObject);

begin

// Display time and date

lblTime.Caption := timetostr(Time);

lblDate.Caption := datetostr(Date);

end;

end.

var

frmMainmenu: TfrmMainmenu;

CurrentDir: string;

ImageDir: string;

UserProgress: array [1 .. 12] of integer;

implementation

{$R \*.dfm}

uses

Login, Globalsetup, AdminOptions, Settings, Error, Level;

procedure TfrmMainmenu.FormClose(Sender: TObject; var Action: TCloseAction);

begin

frmLogin.Show;

end;

procedure TfrmMainmenu.FormCreate(Sender: TObject);

begin

frmMainmenu.Scaled := true;

frmMainmenu.Position := poscreencenter;

end;

procedure TfrmMainmenu.FormShow(Sender: TObject);

var

i, x: integer;

pb: TProgressBar;

begin

// Display admin options to the user if they have administrator rights

case globalpasslevel of

0:

begin

frmMainmenu.Caption := 'Spaggy''s Spelling';

imgAdminOptions.Visible := false;

lblAdminOptions.Visible := false;

end;

1:

begin

frmMainmenu.Caption := 'Spaggy''s Spelling (Admin)';

imgAdminOptions.Visible := true;

lblAdminOptions.Visible := true;

end;

end;

with adoqryUsers do

begin

Active := false;

ConnectionString := globalconnectionstring;

SQL.Clear;

// Run SQL to gather all fields linked to the username

SQL.Add('Select \* from tblUsers where Username = "' +

globalusername + '";');

Active := true;

// Set focus to the record

First;

CurrentDir := GetCurrentDir;

// Set image directory (contained within program folder) from main program directory

// (SizeOf(Pointer)\*8) returns the windows platform at runtime (x86/x64)

ImageDir := StringReplace(CurrentDir, 'Win' + (inttostr(SizeOf(Pointer) \* 8)

) + '\Debug', 'Profile Pictures\', [rfIgnoreCase]);

// Load the user's profile picture to imgProfilePic

imgProfilePic.Picture.LoadFromFile

(ImageDir + adoqryUsers['ProfilePicture']);

// Load the user's profile info and display within corresponding labels

lblScore.Caption := 'Score: ' + floattostr(adoqryUsers['Score']);

lblDailyStreak.Caption := 'Daily Streak: ' +

floattostr(adoqryUsers['DailyStreak']);

lblLastLogin.Caption := 'Last Login: ' +

datetostr(adoqryUsers['LastLogin']);

// Store user's level progress to local array

for i := 1 to 12 do

UserProgress[i] := FieldByName('Level' + inttostr(i)).AsInteger;

end;

lblUsername.Caption := globalusername;

for x := 1 to 2 do

// Set positions for each progress bar in the main menu according to level progress

for i := 1 to 12 do

begin

pb := TProgressBar(frmMainmenu.FindComponent('pb' + inttostr(i)));

pb.Position := trunc((UserProgress[i] / LexiconSize[i]) \* 100);

// Set progress bar colour to green if full (completed level)

if pb.Position = 100 then

pb.State := pbsNormal;

end;

end;

procedure TfrmMainmenu.imgAdminOptionsClick(Sender: TObject);

begin

frmAdminoptions.Show;

frmMainmenu.Hide;

end;

procedure TfrmMainmenu.imgLogoutClick(Sender: TObject);

begin

frmLogin.Show;

frmMainmenu.Hide;

end;

procedure TfrmMainmenu.imgLvl10Click(Sender: TObject);

begin

// Grant access to level if incomplete and previous level has been partially complete

if ((pb9.Position > 10) and (pb10.Position < 100)) then

begin

setglobalgamelevel(10);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl11Click(Sender: TObject);

begin

if ((pb10.Position > 10) and (pb11.Position < 100)) then

begin

setglobalgamelevel(11);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl12Click(Sender: TObject);

begin

if ((pb11.Position > 10) and (pb12.Position < 100)) then

begin

setglobalgamelevel(12);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl1Click(Sender: TObject);

begin

if pb1.Position < 100 then

begin

setglobalgamelevel(1);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl2Click(Sender: TObject);

begin

if ((pb1.Position > 10) and (pb2.Position < 100)) then

begin

setglobalgamelevel(2);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl3Click(Sender: TObject);

begin

if ((pb2.Position > 10) and (pb3.Position < 100)) then

begin

setglobalgamelevel(3);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl4Click(Sender: TObject);

begin

if ((pb3.Position > 10) and (pb4.Position < 100)) then

begin

setglobalgamelevel(4);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl5Click(Sender: TObject);

begin

if ((pb4.Position > 10) and (pb5.Position < 100)) then

begin

setglobalgamelevel(5);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl6Click(Sender: TObject);

begin

if ((pb5.Position > 10) and (pb6.Position < 100)) then

begin

setglobalgamelevel(6);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl7Click(Sender: TObject);

begin

if ((pb6.Position > 10) and (pb7.Position < 100)) then

begin

setglobalgamelevel(7);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl8Click(Sender: TObject);

begin

if ((pb7.Position > 10) and (pb8.Position < 100)) then

begin

setglobalgamelevel(8);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl9Click(Sender: TObject);

begin

if ((pb8.Position > 10) and (pb9.Position < 100)) then

begin

setglobalgamelevel(9);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgSettingsClick(Sender: TObject);

begin

frmSettings.Show;

frmMainmenu.Hide;

end;

procedure TfrmMainmenu.timerClockTimer(Sender: TObject);

begin

// Display time and date

lblTime.Caption := timetostr(Time);

lblDate.Caption := datetostr(Date);

end;

end.

procedure TfrmMainmenu.imgLvl11Click(Sender: TObject);

begin

if ((pb10.Position > 10) and (pb11.Position < 100)) then

begin

setglobalgamelevel(11);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl12Click(Sender: TObject);

begin

if ((pb11.Position > 10) and (pb12.Position < 100)) then

begin

setglobalgamelevel(12);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl1Click(Sender: TObject);

begin

if pb1.Position < 100 then

begin

setglobalgamelevel(1);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl2Click(Sender: TObject);

begin

if ((pb1.Position > 10) and (pb2.Position < 100)) then

begin

setglobalgamelevel(2);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl3Click(Sender: TObject);

begin

if ((pb2.Position > 10) and (pb3.Position < 100)) then

begin

setglobalgamelevel(3);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl4Click(Sender: TObject);

begin

if ((pb3.Position > 10) and (pb4.Position < 100)) then

begin

setglobalgamelevel(4);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl5Click(Sender: TObject);

begin

if ((pb4.Position > 10) and (pb5.Position < 100)) then

begin

setglobalgamelevel(5);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl6Click(Sender: TObject);

begin

if ((pb5.Position > 10) and (pb6.Position < 100)) then

begin

setglobalgamelevel(6);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl7Click(Sender: TObject);

begin

if ((pb6.Position > 10) and (pb7.Position < 100)) then

begin

setglobalgamelevel(7);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl8Click(Sender: TObject);

begin

if ((pb7.Position > 10) and (pb8.Position < 100)) then

begin

setglobalgamelevel(8);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl9Click(Sender: TObject);

begin

if ((pb8.Position > 10) and (pb9.Position < 100)) then

begin

setglobalgamelevel(9);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgSettingsClick(Sender: TObject);

begin

frmSettings.Show;

frmMainmenu.Hide;

end;

procedure TfrmMainmenu.timerClockTimer(Sender: TObject);

begin

// Display time and date

lblTime.Caption := timetostr(Time);

lblDate.Caption := datetostr(Date);

end;

end.

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgLvl9Click(Sender: TObject);

begin

if ((pb8.Position > 10) and (pb9.Position < 100)) then

begin

setglobalgamelevel(9);

frmLevel.Show;

frmMainmenu.Hide;

end

else

showmessage('Level is locked. Try playing a previous level.');

end;

procedure TfrmMainmenu.imgSettingsClick(Sender: TObject);

begin

frmSettings.Show;

frmMainmenu.Hide;

end;

procedure TfrmMainmenu.timerClockTimer(Sender: TObject);

begin

// Display time and date

lblTime.Caption := timetostr(Time);

lblDate.Caption := datetostr(Date);

end;

end.

# Level form

unit Level;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Imaging.pngimage, Vcl.ExtCtrls,

Vcl.StdCtrls, Vcl.OleServer, SpeechLib\_TLB, Data.DB, Data.Win.ADODB;

type

TfrmLevel = class(TForm)

lblHome: TLabel;

imgHome: TImage;

pnlLevel: TPanel;

lblEnter: TLabel;

edtGuess: TEdit;

btnSubmit: TButton;

btnSkip: TButton;

imgAudio: TImage;

btnRepeat: TButton;

SpVoice: TSpVoice;

imgResult: TImage;

lblResult: TLabel;

adotblUsers: TADOTable;

lblProg1: TLabel;

lblProg2: TLabel;

lblProg3: TLabel;

lblAnswer: TLabel;

cbVoices: TComboBox;

imgSpaggy: TImage;

Image1: TImage;

procedure imgHomeClick(Sender: TObject);

procedure btnSubmitClick(Sender: TObject);

procedure btnRepeatClick(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure FormCreate(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure btnSkipClick(Sender: TObject);

procedure edtGuessKeyPress(Sender: TObject; var Key: Char);

procedure UpdateProgress(x: integer; a: boolean);

procedure FormDestroy(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmLevel: TfrmLevel;

word: string;

guess: string;

CurrentDir: string;

LexiconDir: string;

Lexicon: textfile;

IconDir: string;

count, guesses: integer;

SOToken: ISpeechObjectToken;

SOTokens: ISpeechObjectTokens;

sl: TStringList;

implementation

{$R \*.dfm}

uses

Mainmenu, Globalsetup;

procedure TfrmLevel.FormClose(Sender: TObject; var Action: TCloseAction);

begin

// Close ADO object and Lexicon file properly before closing

adotblUsers.Close;

closefile(Lexicon);

SpVoice.Volume := 0;

frmMainmenu.Show;

end;

procedure TfrmLevel.FormCreate(Sender: TObject);

var

i: integer;

begin

frmLevel.Position := poscreencenter;

// Gather speech object tokens for all available voices (SpeechAPI)

SOTokens := SpVoice.GetVoices('', '');

// Populate cbVoices with voices

for i := 0 to (SOTokens.count - 1) do

begin

SOToken := SOTokens.Item(i);

cbVoices.Items.AddObject(SOToken.GetDescription(0), TObject(SOToken));

//Increment descriptor reference count to ensure it's not destroyed

SOToken.\_AddRef;

end;

end;

procedure TfrmLevel.FormDestroy(Sender: TObject);

begin

// Free TStringList object so that it is properly removed

sl.Free;

end;

procedure TfrmLevel.FormShow(Sender: TObject);

var

i: integer;

begin

// Using ADO object, locate the user's data and load previous level progress to count

with adotblUsers do

begin

Active := false;

ConnectionString := globalconnectionstring;

TableName := 'tblUsers';

Active := true;

Open;

Locate('Username', globalusername, []);

count := FieldByName('Level' + inttostr(globalgamelevel)).AsInteger;

end;

// Reset GUI labels

guesses := 0;

lblResult.Caption := '';

imgResult.Visible := false;

lblProg1.Caption := formatfloat('000', count);

lblProg3.Caption := inttostr(LexiconSize[globalgamelevel]);

CurrentDir := GetCurrentDir;

// Set icon and lexicon directory (contained within program folder) from main program directory

// (SizeOf(Pointer)\*8) returns the windows platform at runtime (x86/x64)

IconDir := StringReplace(CurrentDir, 'Win' + (inttostr(SizeOf(Pointer) \* 8)) +

'\Debug', 'Icons\', [rfIgnoreCase]);

LexiconDir := StringReplace(CurrentDir, 'Win' + (inttostr(SizeOf(Pointer) \* 8)

) + '\Debug', 'Lexicon\', [rfIgnoreCase]);

// Create TStringList object, and populate the list with the values in the

// Config file: 1st line = volume .. etc...

sl := TStringList.Create;

sl.LoadFromFile(LexiconDir + 'VoiceConfig.txt');

// Set volume and rate of SpVoice to values in config file (SpeechAPI)

SpVoice.Volume := strtoint(sl[0]);

SpVoice.Rate := strtoint(sl[1]);

cbVoices.ItemIndex := strtoint(sl[2]);

// Set voice of SpVoice to the object in cbVoices that corresponds to the

// Index in the config file

SOToken := ISpeechObjectToken

(Pointer(cbVoices.Items.Objects[cbVoices.ItemIndex]));

SpVoice.Voice := SOToken;

// Assign the current game level file from the lexicon directory to the variable Lexicon

assignfile(Lexicon, LexiconDir + inttostr(globalgamelevel) + '.txt');

reset(Lexicon);

// Set the current word by moving through the words before it in the lexicon

// This will depend on current level progress through 'count'

for i := 0 to count do

readln(Lexicon, word);

// Using the SpVoice component of the SpeechAPI package, output the word to be tested

// Use asynchronous flag to return the command as soon as the speak request is queued

SpVoice.Speak(word, 1);

end;

// UpdateProgress procedure updates the user's data for current level progress

// 'a' parameter is false for skipped answers, to correctly reflect user's score

procedure TfrmLevel.UpdateProgress(x: integer; a: boolean);

var

score: integer;

begin

adotblUsers.Edit;

// Set the value of the current level progress to x

adotblUsers.FieldByName('Level' + inttostr(globalgamelevel)).AsInteger := x;

score := adotblUsers.FieldByName('Score').AsInteger;

// Only increment user's score if answer hasn't been skipped

if a then

adotblUsers.FieldByName('Score').AsInteger := (score + 1);

adotblUsers.Post;

adotblUsers.Refresh;

// Display current level progress to the user

lblProg1.Caption := formatfloat('000', x);

end;

procedure TfrmLevel.btnSubmitClick(Sender: TObject);

begin

guess := edtGuess.Text;

edtGuess.Text := '';

lblAnswer.Visible := false;

imgResult.Visible := true;

// If the guess matches the word then update progress and move onto next word

if UpperCase(Trim(guess)) = UpperCase(word) then

begin

// Display image informing the user that their guess is correct

imgResult.Picture.LoadFromFile(IconDir + 'icons8\_Happy\_64px.png');

lblResult.Caption := 'Correct';

inc(count);

UpdateProgress(count, true);

// If 'count' is equal to the size of the current lexicon then the level is complete

if count = LexiconSize[globalgamelevel] then

begin

showmessage('Level Completed.');

frmLevel.Close;

end

else

// Else speak the next word

begin

guesses := 0;

readln(Lexicon, word);

SpVoice.Speak(word, 1);

edtGuess.SetFocus;

end;

end

else

// Else display image informing the user that their guess is incorrect

begin

imgResult.Picture.LoadFromFile(IconDir + 'icons8\_Sad\_64px.png');

lblResult.Caption := 'Incorrect';

// Increment guesses for each incorrect guess

inc(guesses);

// If the user makes 3 failed attempts then display correct answer

if guesses = 3 then

begin

lblAnswer.Visible := true;

edtGuess.Text := word;

imgResult.Visible := false;

lblResult.Caption := '';

end;

SpVoice.Speak(word, 1);

edtGuess.SetFocus;

end;

end;

procedure TfrmLevel.btnSkipClick(Sender: TObject);

begin

// Increment count and update user's data, skipping the current word

inc(count);

UpdateProgress(count, false);

// Speak next word

readln(Lexicon, word);

SpVoice.Speak(word, 1);

lblResult.Caption := '';

imgResult.Visible := false;

edtGuess.SetFocus;

end;

procedure TfrmLevel.btnRepeatClick(Sender: TObject);

begin

SpVoice.Speak(word, 1);

edtGuess.SetFocus;

end;

procedure TfrmLevel.edtGuessKeyPress(Sender: TObject; var Key: Char);

begin

lblResult.Caption := '';

imgResult.Visible := false;

if ord(Key) = VK\_RETURN then

begin

Key := #0; // prevent beeping

btnSubmitClick(Sender);

end;

end;

procedure TfrmLevel.imgHomeClick(Sender: TObject);

begin

frmMainmenu.Show;

// Do not hide, close to properly disconnect objects

frmLevel.Close;

end;

end.

frmLevel.Position := poscreencenter;

// Gather speech object tokens for all available voices (SpeechAPI)

SOTokens := SpVoice.GetVoices('', '');

// Populate cbVoices with voices

for i := 0 to (SOTokens.count - 1) do

begin

SOToken := SOTokens.Item(i);

cbVoices.Items.AddObject(SOToken.GetDescription(0), TObject(SOToken));

//Increment descriptor reference count to ensure it's not destroyed

SOToken.\_AddRef;

end;

end;

procedure TfrmLevel.FormDestroy(Sender: TObject);

begin

// Free TStringList object so that it is properly removed

sl.Free;

end;

procedure TfrmLevel.FormShow(Sender: TObject);

var

i: integer;

begin

// Using ADO object, locate the user's data and load previous level progress to count

with adotblUsers do

begin

Active := false;

ConnectionString := globalconnectionstring;

TableName := 'tblUsers';

Active := true;

Open;

Locate('Username', globalusername, []);

count := FieldByName('Level' + inttostr(globalgamelevel)).AsInteger;

end;

// Reset GUI labels

guesses := 0;

lblResult.Caption := '';

imgResult.Visible := false;

lblProg1.Caption := formatfloat('000', count);

lblProg3.Caption := inttostr(LexiconSize[globalgamelevel]);

CurrentDir := GetCurrentDir;

// Set icon and lexicon directory (contained within program folder) from main program directory

// (SizeOf(Pointer)\*8) returns the windows platform at runtime (x86/x64)

IconDir := StringReplace(CurrentDir, 'Win' + (inttostr(SizeOf(Pointer) \* 8)) +

'\Debug', 'Icons\', [rfIgnoreCase]);

LexiconDir := StringReplace(CurrentDir, 'Win' + (inttostr(SizeOf(Pointer) \* 8)

) + '\Debug', 'Lexicon\', [rfIgnoreCase]);

// Create TStringList object, and populate the list with the values in the

// Config file: 1st line = volume .. etc...

sl := TStringList.Create;

sl.LoadFromFile(LexiconDir + 'VoiceConfig.txt');

// Set volume and rate of SpVoice to values in config file (SpeechAPI)

SpVoice.Volume := strtoint(sl[0]);

SpVoice.Rate := strtoint(sl[1]);

cbVoices.ItemIndex := strtoint(sl[2]);

// Set voice of SpVoice to the object in cbVoices that corresponds to the

// Index in the config file

SOToken := ISpeechObjectToken

(Pointer(cbVoices.Items.Objects[cbVoices.ItemIndex]));

SpVoice.Voice := SOToken;

// Assign the current game level file from the lexicon directory to the variable Lexicon

assignfile(Lexicon, LexiconDir + inttostr(globalgamelevel) + '.txt');

reset(Lexicon);

// Set the current word by moving through the words before it in the lexicon

// This will depend on current level progress through 'count'

for i := 0 to count do

readln(Lexicon, word);

// Using the SpVoice component of the SpeechAPI package, output the word to be tested

// Use asynchronous flag to return the command as soon as the speak request is queued

SpVoice.Speak(word, 1);

end;

// UpdateProgress procedure updates the user's data for current level progress

// 'a' parameter is false for skipped answers, to correctly reflect user's score

procedure TfrmLevel.UpdateProgress(x: integer; a: boolean);

var

score: integer;

begin

adotblUsers.Edit;

// Set the value of the current level progress to x

adotblUsers.FieldByName('Level' + inttostr(globalgamelevel)).AsInteger := x;

score := adotblUsers.FieldByName('Score').AsInteger;

// Only increment user's score if answer hasn't been skipped

if a then

adotblUsers.FieldByName('Score').AsInteger := (score + 1);

adotblUsers.Post;

adotblUsers.Refresh;

// Display current level progress to the user

lblProg1.Caption := formatfloat('000', x);

end;

procedure TfrmLevel.btnSubmitClick(Sender: TObject);

begin

guess := edtGuess.Text;

edtGuess.Text := '';

lblAnswer.Visible := false;

imgResult.Visible := true;

// If the guess matches the word then update progress and move onto next word

if UpperCase(Trim(guess)) = UpperCase(word) then

begin

// Display image informing the user that their guess is correct

imgResult.Picture.LoadFromFile(IconDir + 'icons8\_Happy\_64px.png');

lblResult.Caption := 'Correct';

inc(count);

UpdateProgress(count, true);

// If 'count' is equal to the size of the current lexicon then the level is complete

if count = LexiconSize[globalgamelevel] then

begin

showmessage('Level Completed.');

frmLevel.Close;

end

else

// Else speak the next word

begin

guesses := 0;

readln(Lexicon, word);

SpVoice.Speak(word, 1);

edtGuess.SetFocus;

end;

end

else

// Else display image informing the user that their guess is incorrect

begin

imgResult.Picture.LoadFromFile(IconDir + 'icons8\_Sad\_64px.png');

lblResult.Caption := 'Incorrect';

// Increment guesses for each incorrect guess

inc(guesses);

// If the user makes 3 failed attempts then display correct answer

if guesses = 3 then

begin

lblAnswer.Visible := true;

edtGuess.Text := word;

imgResult.Visible := false;

lblResult.Caption := '';

end;

SpVoice.Speak(word, 1);

edtGuess.SetFocus;

end;

end;

procedure TfrmLevel.btnSkipClick(Sender: TObject);

begin

// Increment count and update user's data, skipping the current word

inc(count);

UpdateProgress(count, false);

// Speak next word

readln(Lexicon, word);

SpVoice.Speak(word, 1);

lblResult.Caption := '';

imgResult.Visible := false;

edtGuess.SetFocus;

end;

procedure TfrmLevel.btnRepeatClick(Sender: TObject);

begin

SpVoice.Speak(word, 1);

edtGuess.SetFocus;

end;

procedure TfrmLevel.edtGuessKeyPress(Sender: TObject; var Key: Char);

begin

lblResult.Caption := '';

imgResult.Visible := false;

if ord(Key) = VK\_RETURN then

begin

Key := #0; // prevent beeping

btnSubmitClick(Sender);

end;

end;

procedure TfrmLevel.imgHomeClick(Sender: TObject);

begin

frmMainmenu.Show;

// Do not hide, close to properly disconnect objects

frmLevel.Close;

end;

end.

guesses := 0;

readln(Lexicon, word);

SpVoice.Speak(word, 1);

edtGuess.SetFocus;

end;

end

else

// Else display image informing the user that their guess is incorrect

begin

imgResult.Picture.LoadFromFile(IconDir + 'icons8\_Sad\_64px.png');

lblResult.Caption := 'Incorrect';

// Increment guesses for each incorrect guess

inc(guesses);

// If the user makes 3 failed attempts then display correct answer

if guesses = 3 then

begin

lblAnswer.Visible := true;

edtGuess.Text := word;

imgResult.Visible := false;

lblResult.Caption := '';

end;

SpVoice.Speak(word, 1);

edtGuess.SetFocus;

end;

end;

procedure TfrmLevel.btnSkipClick(Sender: TObject);

begin

// Increment count and update user's data, skipping the current word

inc(count);

UpdateProgress(count, false);

// Speak next word

readln(Lexicon, word);

SpVoice.Speak(word, 1);

lblResult.Caption := '';

imgResult.Visible := false;

edtGuess.SetFocus;

end;

procedure TfrmLevel.btnRepeatClick(Sender: TObject);

begin

SpVoice.Speak(word, 1);

edtGuess.SetFocus;

end;

procedure TfrmLevel.edtGuessKeyPress(Sender: TObject; var Key: Char);

begin

lblResult.Caption := '';

imgResult.Visible := false;

if ord(Key) = VK\_RETURN then

begin

Key := #0; // prevent beeping

btnSubmitClick(Sender);

end;

end;

procedure TfrmLevel.imgHomeClick(Sender: TObject);

begin

frmMainmenu.Show;

// Do not hide, close to properly disconnect objects

frmLevel.Close;

end;

end.

# Settings form

unit Settings;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Imaging.pngimage, Vcl.ExtCtrls,

Vcl.StdCtrls;

type

TfrmSettings = class(TForm)

imgHome: TImage;

lblSettings: TLabel;

lblHome: TLabel;

pnlHeadsetSettings: TPanel;

imgHeadsetSettings: TImage;

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure FormCreate(Sender: TObject);

procedure imgHomeClick(Sender: TObject);

procedure pnlHeadsetSettingsClick(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmSettings: TfrmSettings;

implementation

{$R \*.dfm}

uses

Mainmenu, HeadsetSettings;

procedure TfrmSettings.FormClose(Sender: TObject; var Action: TCloseAction);

begin

frmMainmenu.Show;

end;

procedure TfrmSettings.FormCreate(Sender: TObject);

begin

frmSettings.Position := poscreencenter;

end;

procedure TfrmSettings.imgHomeClick(Sender: TObject);

begin

frmMainmenu.Show;

frmSettings.Hide;

end;

procedure TfrmSettings.pnlHeadsetSettingsClick(Sender: TObject);

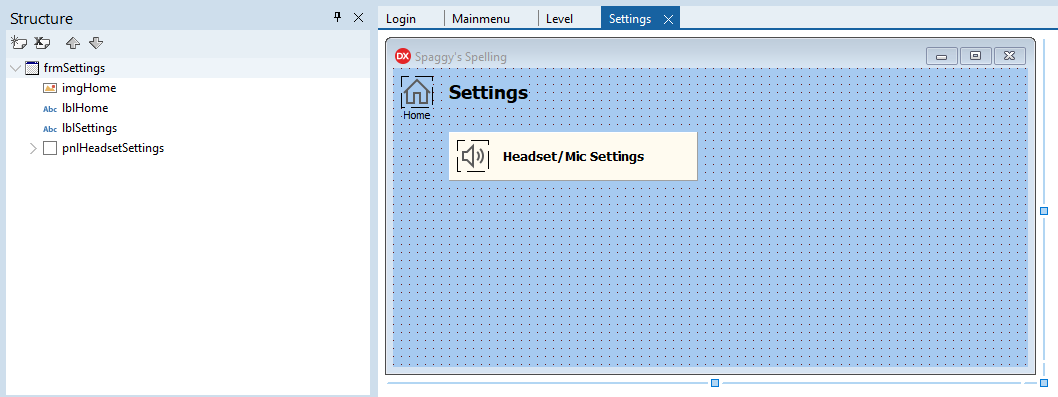
begin

frmHeadsetsettings.Show;

frmSettings.Hide;

end;

end.



# HeadsetSettings form

unit HeadsetSettings;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Imaging.pngimage,

Vcl.ExtCtrls, Vcl.ComCtrls, Vcl.OleServer, SpeechLib\_TLB;

type

TfrmHeadsetsettings = class(TForm)

imgBack: TImage;

lblBack: TLabel;

lblViewUsers: TLabel;

SpVoice: TSpVoice;

tbVolume: TTrackBar;

lblTBVolPos: TLabel;

btnRestore: TButton;

cbVoices: TComboBox;

tbRate: TTrackBar;

lblTBRatePos: TLabel;

lblVolume: TLabel;

lblRate: TLabel;

lblVoices: TLabel;

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure FormCreate(Sender: TObject);

procedure imgBackClick(Sender: TObject);

procedure tbVolumeChange(Sender: TObject);

procedure tbRateChange(Sender: TObject);

procedure FormDestroy(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure cbVoicesChange(Sender: TObject);

procedure btnRestoreClick(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmHeadsetsettings: TfrmHeadsetsettings;

SOToken: ISpeechObjectToken;

SOTokens: ISpeechObjectTokens;

LexiconDir: string;

CurrentDir: string;

sl: TStringList;

implementation

{$R \*.dfm}

uses

Settings;

procedure TfrmHeadsetsettings.btnRestoreClick(Sender: TObject);

begin

// Set trackbars and combobox to default values

// This will trigger onChange event for each, posting changes to config file

tbVolume.Position := 100;

tbRate.Position := 0;

cbVoices.ItemIndex := 0;

sl[2] := inttostr(cbVoices.ItemIndex);

sl.SaveToFile(LexiconDir + 'VoiceConfig.txt');

end;

procedure TfrmHeadsetsettings.cbVoicesChange(Sender: TObject);

begin

// On change of voice, post cb index to config file. This index will be used

// For an identical combobox on the level form

sl[2] := inttostr(cbVoices.ItemIndex);

sl.SaveToFile(LexiconDir + 'VoiceConfig.txt');

end;

procedure TfrmHeadsetsettings.FormClose(Sender: TObject;

var Action: TCloseAction);

begin

frmSettings.Show;

end;

procedure TfrmHeadsetsettings.FormCreate(Sender: TObject);

var

i: integer;

begin

frmHeadsetsettings.Position := poscreencenter;

// Gather speech object tokens for all available voices (SpeechAPI)

SOTokens := SpVoice.GetVoices('', '');

// Populate cbVoices with voices

for i := 0 to (SOTokens.Count - 1) do

begin

SOToken := SOTokens.Item(i);

cbVoices.Items.AddObject(SOToken.GetDescription(0), TObject(SOToken));

//Increment descriptor reference count to ensure it's not destroyed

SOToken.\_AddRef;

end;

end;

procedure TfrmHeadsetsettings.FormDestroy(Sender: TObject);

begin

// Free TStringList object so that it is properly removed

sl.Free;

end;

procedure TfrmHeadsetsettings.FormShow(Sender: TObject);

begin

CurrentDir := GetCurrentDir;

LexiconDir := StringReplace(CurrentDir, 'Win' + (inttostr(SizeOf(Pointer) \* 8)

) + '\Debug', 'Lexicon\', [rfIgnoreCase]);

// Create TStringList object, and populate the list with the values in the

// Config file: 1st line = volume .. etc...

sl := TStringList.Create;

sl.LoadFromFile(LexiconDir + 'VoiceConfig.txt');

tbVolume.Position := strtoint(sl[0]);

tbRate.Position := strtoint(sl[1]);

// If the stored index is greater than the voices available, set to max index

if strtoint(sl[2])>SOTokens.Count then

begin

sl[2]:=inttostr(SOTokens.Count);

sl.SaveToFile(LexiconDir + 'VoiceConfig.txt');

end;

cbVoices.ItemIndex := strtoint(sl[2]);

end;

procedure TfrmHeadsetsettings.imgBackClick(Sender: TObject);

begin

frmSettings.Show;

frmHeadsetsettings.Hide;

end;

procedure TfrmHeadsetsettings.tbRateChange(Sender: TObject);

begin

// On change of rate, post new rate to config file

lblTBRatePos.Caption := (formatfloat('00', tbRate.Position));

sl[1] := inttostr(tbRate.Position);

sl.SaveToFile(LexiconDir + 'VoiceConfig.txt');

end;

procedure TfrmHeadsetsettings.tbVolumeChange(Sender: TObject);

begin

// On change of volume, post new rate to config file

lblTBVolPos.Caption := (formatfloat('000', tbVolume.Position));

sl[0] := inttostr(tbVolume.Position);

sl.SaveToFile(LexiconDir + 'VoiceConfig.txt');

end;

end.

procedure TfrmHeadsetsettings.FormCreate(Sender: TObject);

var

i: integer;

begin

frmHeadsetsettings.Position := poscreencenter;

// Gather speech object tokens for all available voices (SpeechAPI)

SOTokens := SpVoice.GetVoices('', '');

// Populate cbVoices with voices

for i := 0 to (SOTokens.Count - 1) do

begin

SOToken := SOTokens.Item(i);

cbVoices.Items.AddObject(SOToken.GetDescription(0), TObject(SOToken));

//Increment descriptor reference count to ensure it's not destroyed

SOToken.\_AddRef;

end;

end;

procedure TfrmHeadsetsettings.FormDestroy(Sender: TObject);

begin

// Free TStringList object so that it is properly removed

sl.Free;

end;

procedure TfrmHeadsetsettings.FormShow(Sender: TObject);

begin

CurrentDir := GetCurrentDir;

LexiconDir := StringReplace(CurrentDir, 'Win' + (inttostr(SizeOf(Pointer) \* 8)

) + '\Debug', 'Lexicon\', [rfIgnoreCase]);

// Create TStringList object, and populate the list with the values in the

// Config file: 1st line = volume .. etc...

sl := TStringList.Create;

sl.LoadFromFile(LexiconDir + 'VoiceConfig.txt');

tbVolume.Position := strtoint(sl[0]);

tbRate.Position := strtoint(sl[1]);

// If the stored index is greater than the voices available, set to max index

if strtoint(sl[2])>SOTokens.Count then

begin

sl[2]:=inttostr(SOTokens.Count);

sl.SaveToFile(LexiconDir + 'VoiceConfig.txt');

end;

cbVoices.ItemIndex := strtoint(sl[2]);

end;

procedure TfrmHeadsetsettings.imgBackClick(Sender: TObject);

begin

frmSettings.Show;

frmHeadsetsettings.Hide;

end;

procedure TfrmHeadsetsettings.tbRateChange(Sender: TObject);

begin

// On change of rate, post new rate to config file

lblTBRatePos.Caption := (formatfloat('00', tbRate.Position));

sl[1] := inttostr(tbRate.Position);

sl.SaveToFile(LexiconDir + 'VoiceConfig.txt');

end;

procedure TfrmHeadsetsettings.tbVolumeChange(Sender: TObject);

begin

// On change of volume, post new rate to config file

lblTBVolPos.Caption := (formatfloat('000', tbVolume.Position));

sl[0] := inttostr(tbVolume.Position);

sl.SaveToFile(LexiconDir + 'VoiceConfig.txt');

end;

end.

# AdminOptions form

unit AdminOptions;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Imaging.pngimage, Vcl.ExtCtrls,

Vcl.StdCtrls;

type

TfrmAdminoptions = class(TForm)

imgHome: TImage;

imgAddUser: TImage;

pnlAddUser: TPanel;

pnlViewUsers: TPanel;

imgViewUsers: TImage;

lblHome: TLabel;

lblAdminOptions: TLabel;

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure imgHomeClick(Sender: TObject);

procedure pnlAddUserClick(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure pnlViewUsersClick(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmAdminoptions: TfrmAdminoptions;

implementation

{$R \*.dfm}

uses

Mainmenu, AddUser, ViewUsers;

procedure TfrmAdminoptions.FormClose(Sender: TObject; var Action: TCloseAction);

begin

frmMainmenu.Show;

end;

procedure TfrmAdminoptions.FormCreate(Sender: TObject);

begin

frmAdminoptions.Position := poscreencenter;

end;

procedure TfrmAdminoptions.imgHomeClick(Sender: TObject);

begin

frmMainmenu.Show;

frmAdminoptions.Hide;

end;

procedure TfrmAdminoptions.pnlAddUserClick(Sender: TObject);

begin

frmAdduser.Show;

frmAdminoptions.Hide;

end;

procedure TfrmAdminoptions.pnlViewUsersClick(Sender: TObject);

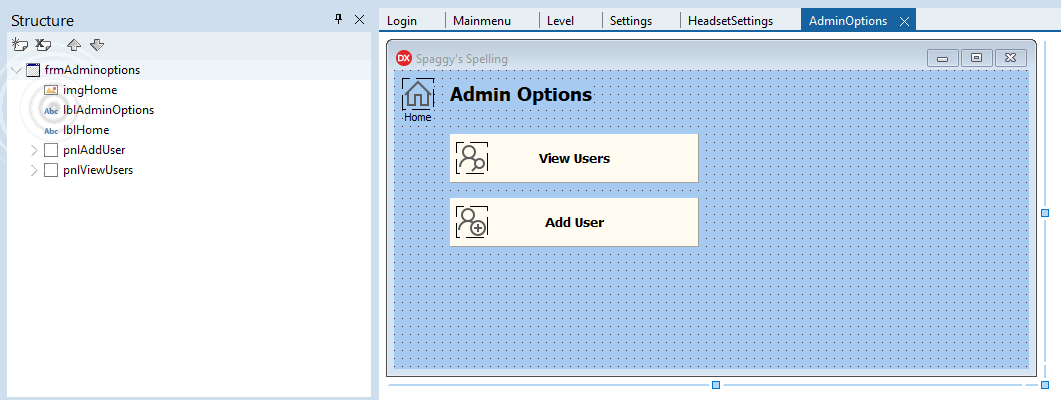
begin

frmViewusers.Show;

frmAdminoptions.Hide;

end;

end.



# AddUser form

unit AddUser;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Data.Win.ADODB, Data.DB,

Vcl.StdCtrls, Vcl.Grids, Vcl.DBGrids, Vcl.Imaging.pngimage, Vcl.ComCtrls;

type

TfrmAdduser = class(TForm)

imgBack: TImage;

adotblUsers: TADOTable;

pnlAddUserForm: TPanel;

edtUsername: TEdit;

edtPassword: TEdit;

edtForename: TEdit;

edtEmailPre: TEdit;

ckbxAdmin: TCheckBox;

cbEmail: TComboBox;

lblEmail: TLabel;

btnAddUser: TButton;

edtEmailSuf: TEdit;

lblAddUser: TLabel;

lblBack: TLabel;

lblUsernameError: TLabel;

lblPasswordError: TLabel;

lblForenameError: TLabel;

lblEmailError: TLabel;

lblManFieldUsername: TLabel;

lblManFieldPassword: TLabel;

lblManFieldEmail: TLabel;

lblManFieldStar: TLabel;

lblManField: TLabel;

shpProfilePic: TShape;

btnProfilePicRight: TButton;

btnProfilePicLeft: TButton;

lblProfilePic: TLabel;

pnlAdvancedForm: TPanel;

dtpLastLogin: TDateTimePicker;

edtScore: TEdit;

edtDailyStreak: TEdit;

lblAdvSettings: TLabel;

lblAdvNotice: TLabel;

lblGenSettings: TLabel;

Label1: TLabel;

imgProfilePic: TImage;

imgViewPassword: TImage;

pnlProfilePicForm: TPanel;

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure FormCreate(Sender: TObject);

procedure imgBackClick(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure btnAddUserClick(Sender: TObject);

procedure btnProfilePicRightClick(Sender: TObject);

procedure btnProfilePicLeftClick(Sender: TObject);

procedure imgViewPasswordMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure imgViewPasswordMouseUp(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure ClearForms();

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmAdduser: TfrmAdduser;

CurrentPic: Integer;

CurrentDir: string;

ImageDir: string;

implementation

{$R \*.dfm}

uses

AdminOptions, GlobalSetup, Error;

// ClearForms procedure clears/resets all input boxes in 'add user' form

procedure TfrmAdduser.ClearForms();

begin

edtUsername.Text := '';

lblUsernameError.Visible := false;

edtForename.Text := '';

lblForenameError.Visible := false;

edtPassword.Text := '';

lblPasswordError.Visible := false;

edtEmailPre.Text := '';

edtEmailSuf.Text := '';

cbEmail.ItemIndex := -1;

lblEmailError.Visible := false;

ckbxAdmin.State := cbUnchecked;

dtpLastLogin.Date := Date;

edtScore.Text := '';

edtDailyStreak.Text := '';

end;

procedure TfrmAdduser.btnAddUserClick(Sender: TObject);

var

i: Integer;

Error: boolean;

vInt: Integer;

vStr: string;

begin

// Set Error variable to false. If there is one error in input validation then Error is true

// Input validation corresponds to the validation rules set in the database (or if input blank)

// For each input error, display error hint to user and set focus to the input box

Error := false;

if (edtUsername.Text = '') or (Length(edtUsername.Text) < 6) or

(Length(edtUsername.Text) > 15) then

begin

lblUsernameError.Visible := true;

edtUsername.SetFocus;

Error := true;

end

else

lblUsernameError.Visible := false;

if (Length(edtForename.Text) > 20) then

begin

lblForenameError.Visible := true;

edtForename.SetFocus;

Error := true;

end

else

lblForenameError.Visible := false;

if (edtPassword.Text = '') or (Length(edtPassword.Text) < 6) or

(Length(edtPassword.Text) > 15) then

begin

lblPasswordError.Visible := true;

edtPassword.SetFocus;

Error := true;

end

else

lblPasswordError.Visible := false;

if (edtEmailPre.Text = '') or (edtEmailSuf.Text = '') or

(cbEmail.ItemIndex = -1) or

((Length(edtEmailPre.Text) + Length(edtEmailSuf.Text) + 6) > 30) then

begin

lblEmailError.Visible := true;

edtEmailPre.SetFocus;

Error := true;

end

else

lblEmailError.Visible := false;

if not Error then

begin

adotblUsers.First;

// Check new username against existing usernames. If it already exists, raise an error

repeat

if uppercase(adotblUsers['Username']) = uppercase(edtUsername.Text) then

Error := true;

adotblUsers.Next;

until adotblUsers.Eof or Error;

// If there are no validation errors and username is unique then append new record

if not Error then

begin

adotblUsers.Close;

adotblUsers.Open;

adotblUsers.Append;

adotblUsers['Username'] := edtUsername.Text;

adotblUsers['Password'] := edtPassword.Text;

adotblUsers['Forename'] := edtForename.Text;

adotblUsers['LastLogin'] := dtpLastLogin.Date;

// Only append advanced info if entered (optional field)

if edtScore.GetTextLen > 0 then

adotblUsers['Score'] := strtoint(edtScore.Text)

else

adotblUsers['Score'] := 0;

if edtDailyStreak.GetTextLen > 0 then

adotblUsers['DailyStreak'] := strtoint(edtDailyStreak.Text)

else

adotblUsers['DailyStreak'] := 0;

adotblUsers['ProfilePicture'] := inttostr(CurrentPic) + '.png';

if ckbxAdmin.Checked then

adotblUsers['AdminRights'] := 1

else

adotblUsers['AdminRights'] := 0;

adotblUsers['EMail'] := edtEmailPre.Text + '@' + edtEmailSuf.Text +

cbEmail.Text;

// Repeatedly present input box until a valid school year is entered

// This completes levels that are below the user's current level of knowledge

repeat

repeat

vStr := InputBox('User Info',

'Enter the school year the user is currently attending (0 for reception, max year 5):',

'0');

until TryStrToInt(vStr, vInt);

until (vInt >= 0) and (vInt <= 5);

// Set user progress to default (0 for each level)

for i := 1 to 12 do

adotblUsers['Level' + inttostr(i)] := 0;

// Depending on user input, mark levels below their current knowledge as complete

for i := 1 to (vInt \* 2) do

adotblUsers['Level' + inttostr(i)] := LexiconSize[i];

// Post changes to database and reload

adotblUsers.Post;

adotblUsers.Refresh;

// Inform admin that the user has been successfully added (not an error)

setglobalerrorcode('addusernotification.txt');

frmError.Show;

ClearForms;

end

else

begin

setglobalerrorcode('addusernametakenerror.txt');

frmError.Show;

end;

end

end;

procedure TfrmAdduser.btnProfilePicLeftClick(Sender: TObject);

begin

if CurrentPic = 1 then

CurrentPic := 14

else

Dec(CurrentPic);

imgProfilePic.Picture.LoadFromFile(ImageDir + inttostr(CurrentPic) + '.png');

end;

procedure TfrmAdduser.btnProfilePicRightClick(Sender: TObject);

begin

if CurrentPic = 14 then

CurrentPic := 1

else

Inc(CurrentPic);

imgProfilePic.Picture.LoadFromFile(ImageDir + inttostr(CurrentPic) + '.png');

end;

procedure TfrmAdduser.FormClose(Sender: TObject; var Action: TCloseAction);

begin

frmAdminoptions.Show;

end;

procedure TfrmAdduser.FormCreate(Sender: TObject);

begin

frmAdduser.Position := poscreencenter;

end;

procedure TfrmAdduser.FormShow(Sender: TObject);

begin

ClearForms;

edtScore.NumbersOnly := true;

edtDailyStreak.NumbersOnly := true;

adotblUsers.Active := false;

adotblUsers.ConnectionString := globalconnectionstring;

adotblUsers.TableName := 'tblUsers';

adotblUsers.Active := true;

CurrentDir := GetCurrentDir;

ImageDir := StringReplace(CurrentDir, 'Win' + (inttostr(SizeOf(Pointer) \* 8))

+ '\Debug', 'Profile Pictures\', [rfIgnoreCase]);

CurrentPic := 1;

imgProfilePic.Picture.LoadFromFile(ImageDir + inttostr(CurrentPic) + '.png');

end;

procedure TfrmAdduser.imgViewPasswordMouseDown(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

begin

edtPassword.PasswordChar := #0;

end;

procedure TfrmAdduser.imgViewPasswordMouseUp(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

begin

edtPassword.PasswordChar := '\*';

end;

procedure TfrmAdduser.imgBackClick(Sender: TObject);

begin

frmAdminoptions.Show;

frmAdduser.Hide;

end;

end.

uses

AdminOptions, GlobalSetup, Error;

// ClearForms procedure clears/resets all input boxes in 'add user' form

procedure TfrmAdduser.ClearForms();

begin

edtUsername.Text := '';

lblUsernameError.Visible := false;

edtForename.Text := '';

lblForenameError.Visible := false;

edtPassword.Text := '';

lblPasswordError.Visible := false;

edtEmailPre.Text := '';

edtEmailSuf.Text := '';

cbEmail.ItemIndex := -1;

lblEmailError.Visible := false;

ckbxAdmin.State := cbUnchecked;

dtpLastLogin.Date := Date;

edtScore.Text := '';

edtDailyStreak.Text := '';

end;

procedure TfrmAdduser.btnAddUserClick(Sender: TObject);

var

i: Integer;

Error: boolean;

vInt: Integer;

vStr: string;

begin

// Set Error variable to false. If there is one error in input validation then Error is true

// Input validation corresponds to the validation rules set in the database (or if input blank)

// For each input error, display error hint to user and set focus to the input box

Error := false;

if (edtUsername.Text = '') or (Length(edtUsername.Text) < 6) or

(Length(edtUsername.Text) > 15) then

begin

lblUsernameError.Visible := true;

edtUsername.SetFocus;

Error := true;

end

else

lblUsernameError.Visible := false;

if (Length(edtForename.Text) > 20) then

begin

lblForenameError.Visible := true;

edtForename.SetFocus;

Error := true;

end

else

lblForenameError.Visible := false;

if (edtPassword.Text = '') or (Length(edtPassword.Text) < 6) or

(Length(edtPassword.Text) > 15) then

begin

lblPasswordError.Visible := true;

edtPassword.SetFocus;

Error := true;

end

else

lblPasswordError.Visible := false;

if (edtEmailPre.Text = '') or (edtEmailSuf.Text = '') or

(cbEmail.ItemIndex = -1) or

((Length(edtEmailPre.Text) + Length(edtEmailSuf.Text) + 6) > 30) then

begin

lblEmailError.Visible := true;

edtEmailPre.SetFocus;

Error := true;

end

else

lblEmailError.Visible := false;

if not Error then

begin

adotblUsers.First;

// Check new username against existing usernames. If it already exists, raise an error

repeat

if uppercase(adotblUsers['Username']) = uppercase(edtUsername.Text) then

Error := true;

adotblUsers.Next;

until adotblUsers.Eof or Error;

// If there are no validation errors and username is unique then append new record

if not Error then

begin

adotblUsers.Close;

adotblUsers.Open;

adotblUsers.Append;

adotblUsers['Username'] := edtUsername.Text;

adotblUsers['Password'] := edtPassword.Text;

adotblUsers['Forename'] := edtForename.Text;

adotblUsers['LastLogin'] := dtpLastLogin.Date;

// Only append advanced info if entered (optional field)

if edtScore.GetTextLen > 0 then

adotblUsers['Score'] := strtoint(edtScore.Text)

else

adotblUsers['Score'] := 0;

if edtDailyStreak.GetTextLen > 0 then

adotblUsers['DailyStreak'] := strtoint(edtDailyStreak.Text)

else

adotblUsers['DailyStreak'] := 0;

adotblUsers['ProfilePicture'] := inttostr(CurrentPic) + '.png';

if ckbxAdmin.Checked then

adotblUsers['AdminRights'] := 1

else

adotblUsers['AdminRights'] := 0;

adotblUsers['EMail'] := edtEmailPre.Text + '@' + edtEmailSuf.Text +

cbEmail.Text;

// Repeatedly present input box until a valid school year is entered

// This completes levels that are below the user's current level of knowledge

repeat

repeat

vStr := InputBox('User Info',

'Enter the school year the user is currently attending (0 for reception, max year 5):',

'0');

until TryStrToInt(vStr, vInt);

until (vInt >= 0) and (vInt <= 5);

// Set user progress to default (0 for each level)

for i := 1 to 12 do

adotblUsers['Level' + inttostr(i)] := 0;

// Depending on user input, mark levels below their current knowledge as complete

for i := 1 to (vInt \* 2) do

adotblUsers['Level' + inttostr(i)] := LexiconSize[i];

// Post changes to database and reload

adotblUsers.Post;

adotblUsers.Refresh;

// Inform admin that the user has been successfully added (not an error)

setglobalerrorcode('addusernotification.txt');

frmError.Show;

ClearForms;

end

else

begin

setglobalerrorcode('addusernametakenerror.txt');

frmError.Show;

end;

end

end;

procedure TfrmAdduser.btnProfilePicLeftClick(Sender: TObject);

begin

if CurrentPic = 1 then

CurrentPic := 14

else

Dec(CurrentPic);

imgProfilePic.Picture.LoadFromFile(ImageDir + inttostr(CurrentPic) + '.png');

end;

procedure TfrmAdduser.btnProfilePicRightClick(Sender: TObject);

begin

if CurrentPic = 14 then

CurrentPic := 1

else

Inc(CurrentPic);

imgProfilePic.Picture.LoadFromFile(ImageDir + inttostr(CurrentPic) + '.png');

end;

procedure TfrmAdduser.FormClose(Sender: TObject; var Action: TCloseAction);

begin

frmAdminoptions.Show;

end;

procedure TfrmAdduser.FormCreate(Sender: TObject);

begin

frmAdduser.Position := poscreencenter;

end;

procedure TfrmAdduser.FormShow(Sender: TObject);

begin

ClearForms;

edtScore.NumbersOnly := true;

edtDailyStreak.NumbersOnly := true;

adotblUsers.Active := false;

adotblUsers.ConnectionString := globalconnectionstring;

adotblUsers.TableName := 'tblUsers';

adotblUsers.Active := true;

CurrentDir := GetCurrentDir;

ImageDir := StringReplace(CurrentDir, 'Win' + (inttostr(SizeOf(Pointer) \* 8))

+ '\Debug', 'Profile Pictures\', [rfIgnoreCase]);

CurrentPic := 1;

imgProfilePic.Picture.LoadFromFile(ImageDir + inttostr(CurrentPic) + '.png');

end;

procedure TfrmAdduser.imgViewPasswordMouseDown(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

begin

edtPassword.PasswordChar := #0;

end;

procedure TfrmAdduser.imgViewPasswordMouseUp(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

begin

edtPassword.PasswordChar := '\*';

end;

procedure TfrmAdduser.imgBackClick(Sender: TObject);

begin

frmAdminoptions.Show;

frmAdduser.Hide;

end;

end.

for i := 1 to 12 do

adotblUsers['Level' + inttostr(i)] := 0;

// Depending on user input, mark levels below their current knowledge as complete

for i := 1 to (vInt \* 2) do

adotblUsers['Level' + inttostr(i)] := LexiconSize[i];

// Post changes to database and reload

adotblUsers.Post;

adotblUsers.Refresh;

// Inform admin that the user has been successfully added (not an error)

setglobalerrorcode('addusernotification.txt');

frmError.Show;

ClearForms;

end

else

begin

setglobalerrorcode('addusernametakenerror.txt');

frmError.Show;

end;

end

end;

procedure TfrmAdduser.btnProfilePicLeftClick(Sender: TObject);

begin

if CurrentPic = 1 then

CurrentPic := 14

else

Dec(CurrentPic);

imgProfilePic.Picture.LoadFromFile(ImageDir + inttostr(CurrentPic) + '.png');

end;

procedure TfrmAdduser.btnProfilePicRightClick(Sender: TObject);

begin

if CurrentPic = 14 then

CurrentPic := 1

else

Inc(CurrentPic);

imgProfilePic.Picture.LoadFromFile(ImageDir + inttostr(CurrentPic) + '.png');

end;

procedure TfrmAdduser.FormClose(Sender: TObject; var Action: TCloseAction);

begin

frmAdminoptions.Show;

end;

procedure TfrmAdduser.FormCreate(Sender: TObject);

begin

frmAdduser.Position := poscreencenter;

end;

procedure TfrmAdduser.FormShow(Sender: TObject);

begin

ClearForms;

edtScore.NumbersOnly := true;

edtDailyStreak.NumbersOnly := true;

adotblUsers.Active := false;

adotblUsers.ConnectionString := globalconnectionstring;

adotblUsers.TableName := 'tblUsers';

adotblUsers.Active := true;

CurrentDir := GetCurrentDir;

ImageDir := StringReplace(CurrentDir, 'Win' + (inttostr(SizeOf(Pointer) \* 8))

+ '\Debug', 'Profile Pictures\', [rfIgnoreCase]);

CurrentPic := 1;

imgProfilePic.Picture.LoadFromFile(ImageDir + inttostr(CurrentPic) + '.png');

end;

procedure TfrmAdduser.imgViewPasswordMouseDown(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

begin

edtPassword.PasswordChar := #0;

end;

procedure TfrmAdduser.imgViewPasswordMouseUp(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

begin

edtPassword.PasswordChar := '\*';

end;

procedure TfrmAdduser.imgBackClick(Sender: TObject);

begin

frmAdminoptions.Show;

frmAdduser.Hide;

end;

end.

# ViewUsers form

unit ViewUsers;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.StrUtils,

System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Imaging.pngimage, Vcl.ExtCtrls,

Data.DB, Vcl.Grids, Vcl.DBGrids, Data.Win.ADODB, Vcl.StdCtrls, Vcl.Buttons;

type

TfrmViewusers = class(TForm)

imgBack: TImage;

adotblUsers: TADOTable;

dbGrid: TDBGrid;

lblBack: TLabel;

lblViewUsers: TLabel;

pnlViewUsers: TPanel;

DataSource: TDataSource;

sbtnViewPassword: TSpeedButton;

btnRemoveUser: TButton;

adoqryDelete: TADOQuery;

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure FormCreate(Sender: TObject);

procedure imgBackClick(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure sbtnViewPasswordClick(Sender: TObject);

procedure btnRemoveUserClick(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmViewusers: TfrmViewusers;

implementation

{$R \*.dfm}

uses

AdminOptions, Globalsetup;

procedure TfrmViewusers.btnRemoveUserClick(Sender: TObject);

var

password: string;

buttonSelected: integer;

user: string;

begin

// Only remove user if admin has correctly entered db password

InputQuery('Admin: Database Password', 'Enter the password for the database:',

password);

if password = globaldbpassword then

begin

// Set user variable to the username of the selected row on dbGrid

user := dbGrid.Fields[0].AsString;

// Present user with confirmation dialog

buttonSelected := messagedlg('Confirm that the user "' + user +

'" will be REMOVED from the database.', mtCustom, mbOKCancel, 0);

// Locate and delete user if user presses ok on confirmation dialog

if buttonSelected = mrOK then

begin

dbGrid.DataSource.DataSet.Locate('Username', user, []);

dbGrid.DataSource.DataSet.Delete;

showmessage('"' + user + '" has been removed from the database.');

end

else

showmessage('"' + user + '" has NOT been removed from the database.');

end

else

showmessage('Database Error: Incorrect database password.');

end;

procedure TfrmViewusers.FormClose(Sender: TObject; var Action: TCloseAction);

begin

frmAdminoptions.Show;

end;

procedure TfrmViewusers.FormCreate(Sender: TObject);

begin

frmViewusers.Position := poscreencenter;

end;

procedure TfrmViewusers.FormShow(Sender: TObject);

var

i: integer;

ColumnArray: array of integer;

begin

adotblUsers.ConnectionString := globalconnectionstring;

adoqryDelete.ConnectionString := globalconnectionstring;

adotblUsers.TableName := 'tblUsers';

adotblUsers.Active := true;

// Only show neccessary columns in the database, e.g. hide user passwords

ColumnArray := [1, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20];

for i := 0 to 9 do

dbGrid.Columns[i].Width := 80;

for i in ColumnArray do

dbGrid.Columns[i].Visible := false;

end;

procedure TfrmViewusers.imgBackClick(Sender: TObject);

begin

frmAdminoptions.Show;

frmViewusers.Hide;

end;

procedure TfrmViewusers.sbtnViewPasswordClick(Sender: TObject);

var

password: string;

begin

// Only show user passwords once admin has correctly entered db password

InputQuery('Admin: Database Password', 'Enter the password for the database:',

password);

if password = globaldbpassword then

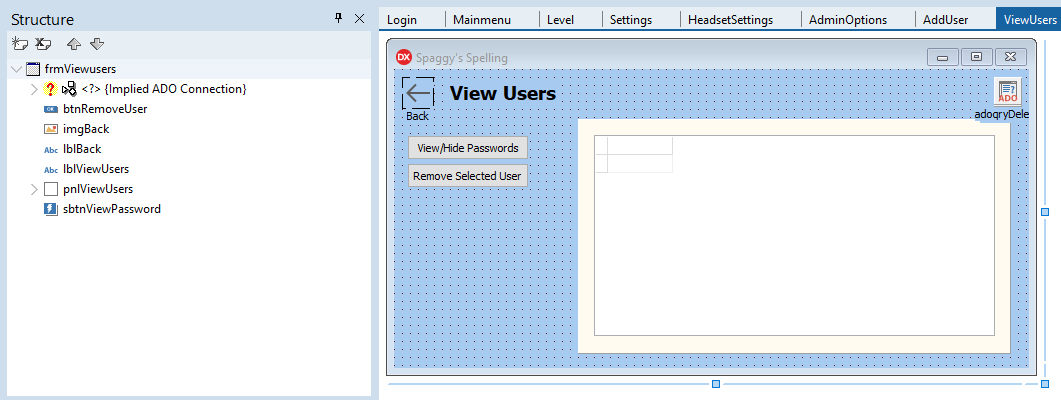
dbGrid.Columns[1].Visible := sbtnViewPassword.Down

else

showmessage('Database Error: Incorrect database password.');

end;

end.



begin

frmViewusers.Position := poscreencenter;

end;

procedure TfrmViewusers.FormShow(Sender: TObject);

var

i: integer;

ColumnArray: array of integer;

begin

adotblUsers.ConnectionString := globalconnectionstring;

adoqryDelete.ConnectionString := globalconnectionstring;

adotblUsers.TableName := 'tblUsers';

adotblUsers.Active := true;

// Only show neccessary columns in the database, e.g. hide user passwords

ColumnArray := [1, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20];

for i := 0 to 9 do

dbGrid.Columns[i].Width := 80;

for i in ColumnArray do

dbGrid.Columns[i].Visible := false;

end;

procedure TfrmViewusers.imgBackClick(Sender: TObject);

begin

frmAdminoptions.Show;

frmViewusers.Hide;

end;

procedure TfrmViewusers.sbtnViewPasswordClick(Sender: TObject);

var

password: string;

begin

// Only show user passwords once admin has correctly entered db password

InputQuery('Admin: Database Password', 'Enter the password for the database:',

password);

if password = globaldbpassword then

dbGrid.Columns[1].Visible := sbtnViewPassword.Down

else

showmessage('Database Error: Incorrect database password.');

end;

end.

# Error form

unit Error;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.StdCtrls;

type

TfrmError = class(TForm)

shpWhitebox: TShape;

btnOk: TButton;

btnHelp: TButton;

memError: TMemo;

procedure FormShow(Sender: TObject);

procedure btnHelpClick(Sender: TObject);

procedure btnOkClick(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure FormHide(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmError: TfrmError;

implementation

{$R \*.dfm}

uses

GlobalSetup, Login;

procedure TfrmError.FormCreate(Sender: TObject);

begin

frmError.Position := poscreencenter;

end;

procedure TfrmError.FormShow(Sender: TObject);

begin

// Set form caption and error message depending on error file

frmError.Caption := globalerrorcode[0];

memError.Lines.Clear;

memError.Lines.Add(globalerrorcode[1]);

// Provide option for additional help if error file contains it

if length(globalerrorcode[2]) > 1 then

btnHelp.Visible := True

else

btnHelp.Visible := False;

end;

procedure TfrmError.btnHelpClick(Sender: TObject);

begin

showmessage(globalerrorcode[2]);

end;

procedure TfrmError.btnOkClick(Sender: TObject);

begin

// Terminate program if database error raised (user cannot login)

if frmError.Caption = 'Database Error' then

begin

Application.Terminate();

Exit;

end;

frmError.Hide;

end;

procedure TfrmError.FormHide(Sender: TObject);

begin

frmLogin.edtUsername.Text := '';

frmLogin.edtPassword.Text := '';

frmLogin.edtUsername.SetFocus;

end;

end.

