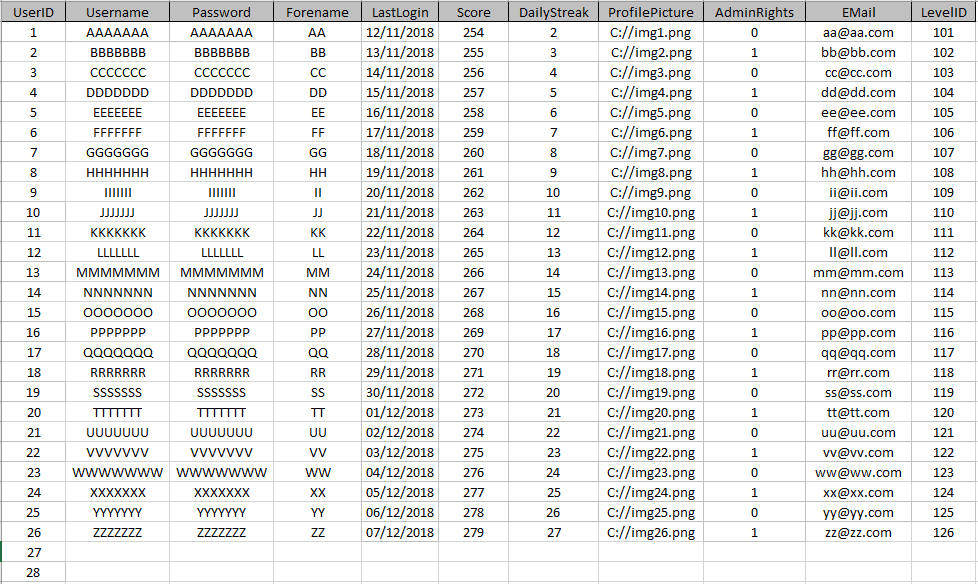
Iterative Development Testing

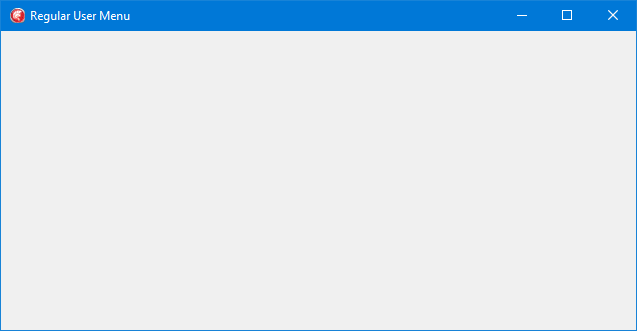
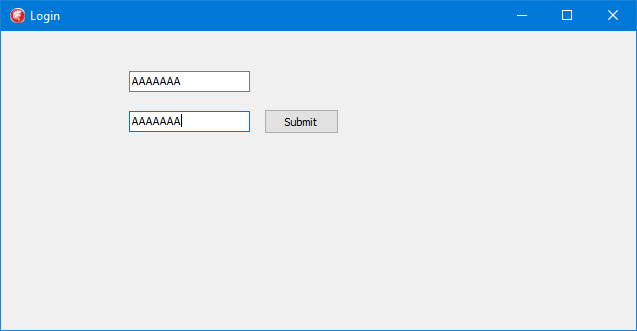
In accordance to the model of development adopted for this program, testing was performed regularly as each unit/function was completed. The results of these tests dictated the functionality of the program, and were used to make further improvements.

# 27/11/18

The first algorithm that required testing in development was the user login. Within the login form, the user is prompted to enter their login credentials in the form of a username and password. These inputs are then searched for in the ‘tblUsers’ table in the database, and the presence of both is returned to the program. If the user exists, then the main menu is shown. The user’s predetermined access rights also dictate what is shown to the user in the main menu form. The database contained the following ‘dummy’ users, which were used for testing:

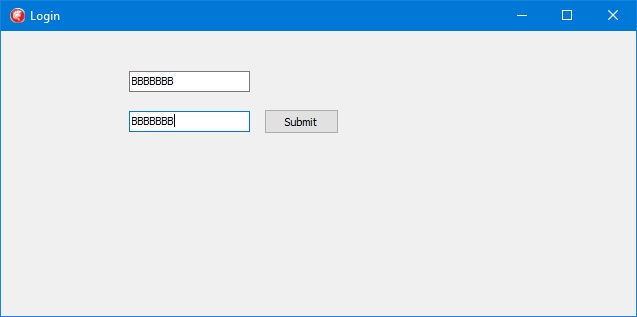
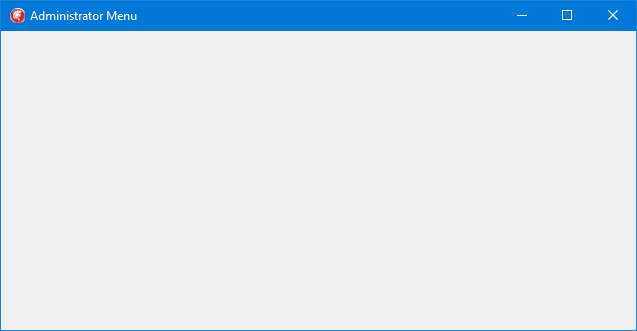
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Function Tested** | **Test Data** | **Expected Result** | **Pass/Fail** |
| 1 | Logging in with valid username and password (User) | AAAAAAA/  AAAAAAA | Log on, with ‘Regular User Menu’ caption | Pass |
| 2 | Logging in with valid username and password (Admin) | BBBBBBB/  BBBBBBB | Log on, with ‘Administrator Menu’ caption | Pass |
| 3 | Logging in with invalid username (boundary) | A/AAAAAAA or bbbbbbb/BBBBBBB or @/CCCCCCC | Log on fail, raise error message | Pass |
| 4 | Logging in with invalid password | AAAAAAA/A or BBBBBBB/bbbbbbb or CCCCCCC/c | Log on fail, raise error message | Pass |
| 5 | Logging in with blank input (abnormal) | ‘’ | Log on fail, raise error message | Pass |

Test 1 evidence:



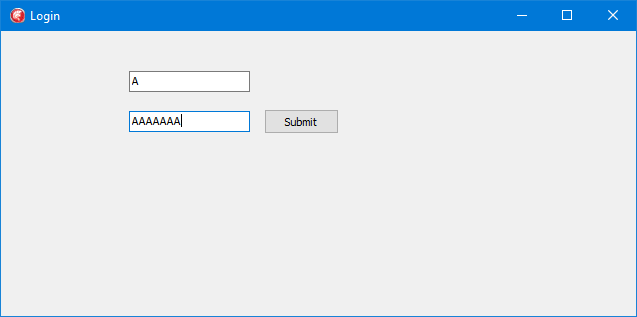
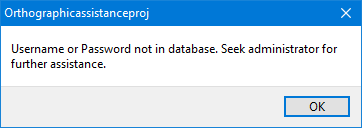
Test 1 successful, with the main menu form showing the correct caption. Note that the password entry is masked in the program. The letters are only shown for testing purposes.

Test 2 evidence:



Test 2 successful, with the main menu form showing the correct caption.

Test 3 evidence:



Test 3 successful, presenting the user with an error message and a description of the error. **However, this description does not offer common solutions to such an input error. For example, a user may not be aware that the username is case-sensitive.**

Code:

uses

Globalsetup, Mainmenu;

procedure TloginForm.FormCreate(Sender: TObject);

var

DatabaseName:textfile;

text:string;

dlg:Topendialog;

filename:string;

connection:string;

connectstring:string;

begin

loginForm.Position:=poscreencenter;

assignfile(databasename,'filelocation.txt');

reset(databasename);

readln(databasename,text);

closefile(databasename);

showmessage(text);

try

if fileexists(text) then

connection:=text

else

begin

dlg:=topendialog.Create(nil);

try

if dlg.Execute() then

begin

filename:=dlg.filename;

showmessage(filename);

connection:=(filename);

rewrite(databasename);

writeln(databasename,connection);

closefile(databasename);

end

else

showmessage('No file has been selected.')

finally

dlg.Free();

end;

end;

finally

end;

connectstring:='provider=microsoft.ace.oledb.16.0;data source ='+connection+';persist security info=false;';

setconnectionstring(connectstring);

adotblUsers.Active:=false;

adotblusers.ConnectionString:=connectstring;

adotblusers.TableName:='tblUsers';

adotblusers.Active:=true;

end;

procedure TloginForm.submitButtonClick(Sender: TObject);

begin

if not ADOtblUsers.Locate('Username',usernameEdit.Text,[]) then

showmessage('Username or Password not in database. Seek administrator for further assistance.')

else

if passwordEdit.Text = ADOtblUsers['Password'] then

begin

setglobalpasslevel(ADOtblUsers['AdminRights']);

mainmenuForm.Show;

loginForm.Hide;

end

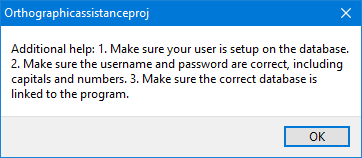
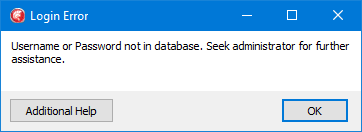
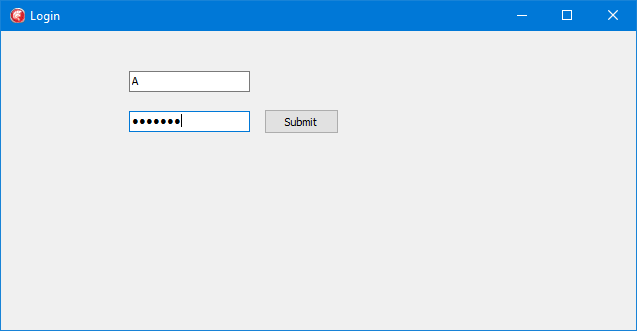
else

showmessage('Username or Password not in database. Seek administrator for further assistance.');

end;

end.

Improvements:



In order to display a more helpful error report to the user, a custom error dialog box was created. This allowed for a more coherent error report, as well as the option to view additional help which may aid the user in fixing the problem.

Additional help shows common solutions to the problem. This provides a user interface which is clearer and easier to use.

Code

procedure TfrmLogin.imgLoginClick(Sender: TObject);

var

DailyStreak: integer;

begin

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

begin

setglobalerrorcode('loginblankerror.txt');

frmError.Show;

end

else

begin

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

begin

setglobalerrorcode('loginerror.txt');

frmError.Show;

end

procedure TerrorForm.FormShow(Sender: TObject);

begin

errorForm.Caption := globalerrorcode[0];

errorMemo.Lines.Clear;

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

if length(globalerrorcode[2]) > 1 then

helpButton.Visible := True

else

helpButton.Visible := False;

end;

procedure TerrorForm.helpButtonClick(Sender: TObject);

begin

showmessage(globalerrorcode[2]);

end;

procedure TerrorForm.okButtonClick(Sender: TObject);

begin

errorForm.Hide;

end;

procedure TerrorForm.FormHide(Sender: TObject);

begin

LoginForm.usernameEdit.Text := '';

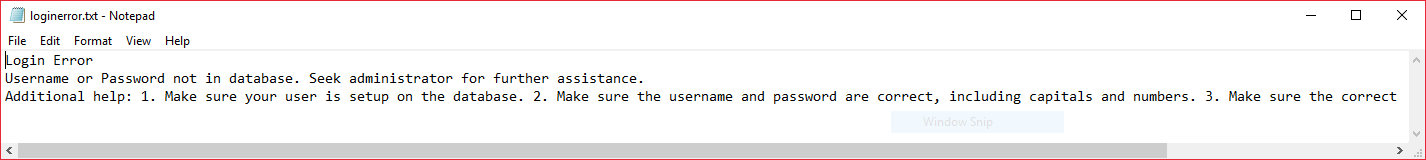
LoginForm.passwordEdit.Text := '';

end;

A separate error form was created, which displays error information based on a custom error file inputted into the form. ‘globalerrorcode’ defines the error information based on the file, before the error form is shown. The array describes the caption, initial error info and additional help. This form can therefore be used for any error in the program.

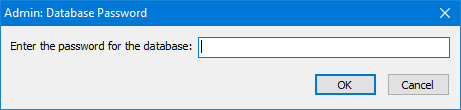
Login Form

Error Form



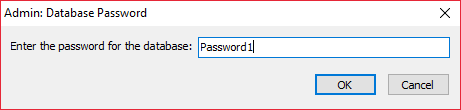
# 30/11/18

The second algorithm that required testing was the algorithm to provide access to the database, using the db password. Since the database is encrypted with this password, and contains sensitive user information, it is important that the data entry is handled correctly.

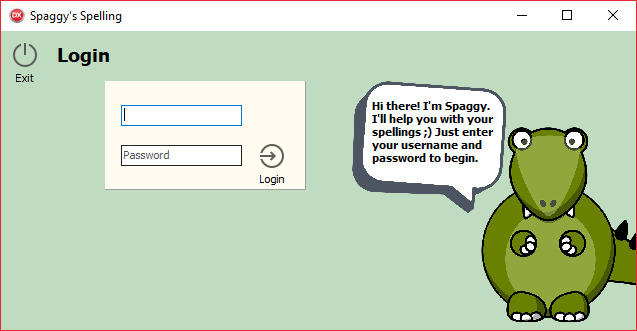


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Function Tested** | **Test Data** | **Expected Result** | **Pass/Fail** |
| 1 | Entering valid db password (normal) | Password1 | Access to database, presented with logon form | Pass |
| 2 | Entering invalid db password (normal) | Password2 | No access, raise error message, presented with logon form | Pass |
| 3 | Entering invalid db password (boundary) | ‘;’ or ‘@~|#’ | No access, raise error message, presented with logon form | Pass |
| 4 | Entering blank db password (abnormal) | ‘’ | No access, raise error message, presented with logon form | Pass |

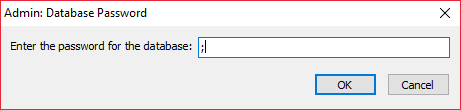
Test 1 evidence:



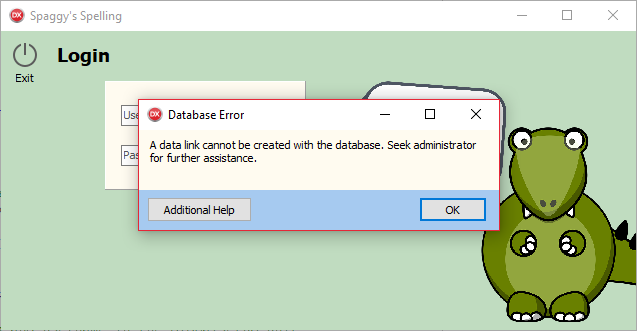
Test 1 successful, with a successful connection made to the database and the logon form presented. **However, the data entry is not masked, despite being a password. This allows over-lookers to steal the password.**



Test 3 evidence:



Test 3 successful, with no access granted and logon screen presented. The existence of the semi-colon may cause the connection string to be mis-interpretted to end in ‘;;’ syntax error, similar to SQL inject. However, the program is unaffected.



Code:

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;

Improvements:

**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 + ';';

By changing the InputQuery function to InputBox, the dialog screen is more customizable, allowing for a password mask. The data entry is now masked:

# 

# 10/12/18

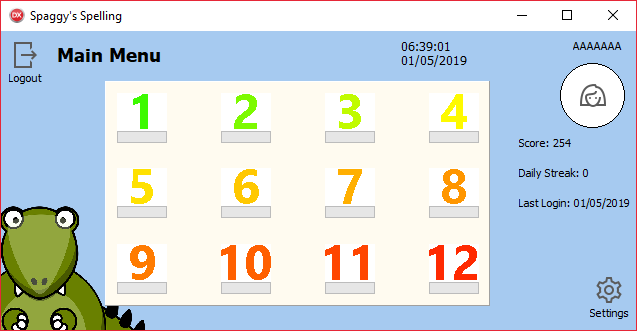
Displaying user information is an important aspect of the MainMenu form. If this data was displayed incorrectly, then the user would be misinformed of their progress. While the data within the database should not exceed normal bounds, being controlled by the program and not user entry, it should still be tested.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Function Tested** | **Test Data** | **Expected Result** | **Pass/Fail** |
| 1 | Displaying expected user information (normal) | Score: 254, DailyStreak:20, LastLogin: 01/05/19 | All data displayed correctly | Pass |
| 2 | Displaying large values for user information (boundary) | Score: 254000000  DailyStreak:365  LastLogin: 02/05/98 | No access, raise error message, presented with logon form | Pass |

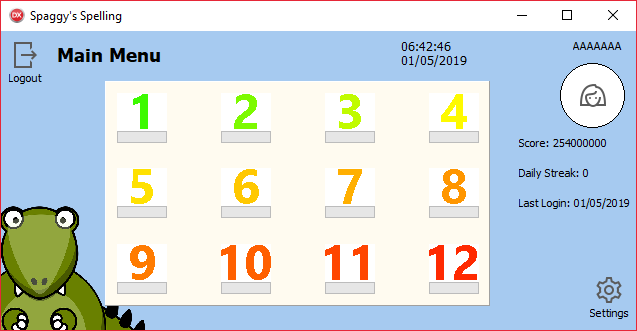
Note that tests could not be run on abnormal data, as this data cannot be entered into the database. This means that the validation rules set by the database, e.g. long integer max size, correctly safeguard against potential errors.

Test 1 evidence:

Test 1 successful, with all information being displayed correctly. Note that the daily streak has been correctly set to zero as the last login wasn’t from the day before.

 Test 2 evidence:

Test 2 successful, with all information being displayed correctly. Note that the daily streak has been correctly set to zero as the last login wasn’t from the day before. LastLogin has also been correctly reset to the current day.



Code:

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;

// 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;

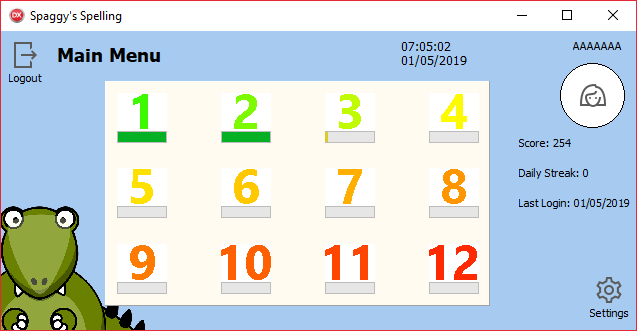
# 14/12/18

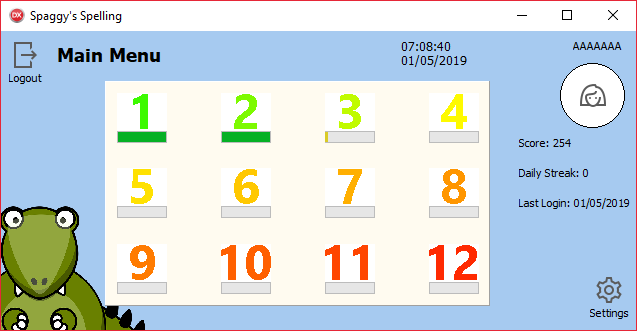
For the user to track their progress, a progress bar was put at the bottom of each level. This shows how far the user is into each level at a glance. Since there are currently no validation rules on user progress into each level, this could cause errors in the program, especially if the admin manually enters abnormal data into the database.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Function Tested** | **Test Data** | **Expected Result** | **Pass/Fail** |
| 1 | Progress bar with standard user progress (normal) | 333,333,20,0(..) | Progress bars correctly display progress | Pass |
| 2 | Progress bar with user progress greater than lexicon size (boundary) | 334,350,20,0(..) | Progress bars full for first two levels, no error raised | Pass |
| 3 | Progress bar with user progress containing abnormal data | 20,-132,-3333,0(..) | Progress bars empty for negative progress, no error raised | Pass |
| 4 | Progress bar with user progress containing abnormal data | -20,-132,-3333,0(..) | Progress bars empty for negative progress, no error raised | Fail |

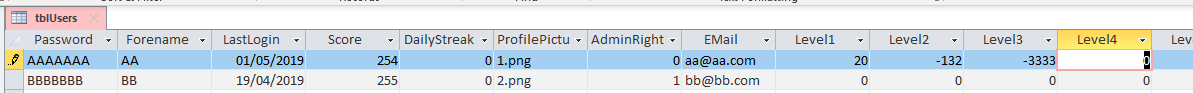
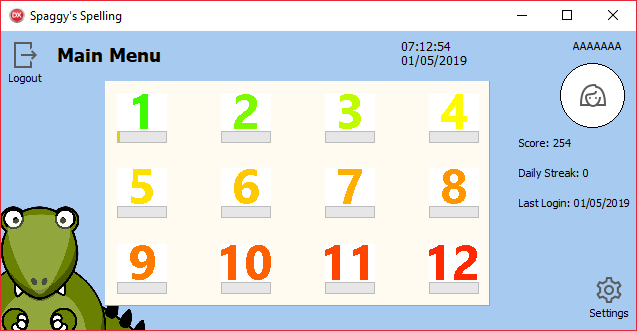
Test 1 evidence:

Test 1 successful, with all the progress bars correctly displaying progress into each level.

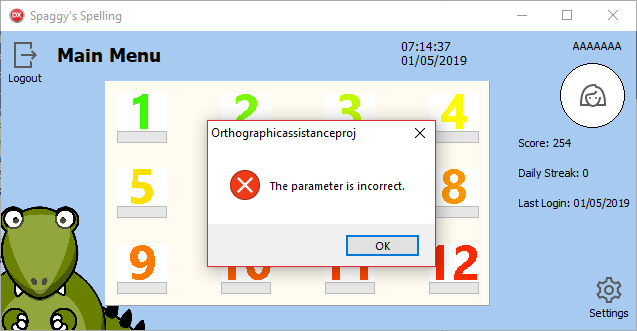


Test 2 evidence:

Test 2 successful, with all the progress bars correctly displaying progress into each level. No error raised.

Test 3 evidence:

Test 3 successful, with all the progress bars correctly displaying progress into each level. No error raised.

Test 4 evidence:

Test 4 fails, with all the progress bars correctly displaying progress into each level, but error raised when opening a level with negative progress.

Code:

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)));

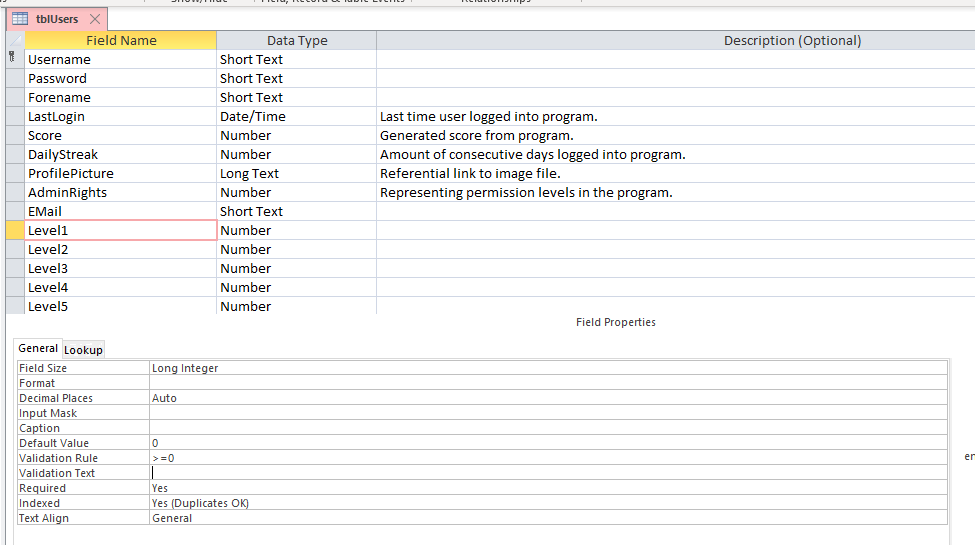
pb.Position := trunc((UserProgress[i] / LexiconSize[i]) \* 100);

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

if pb.Position = 100 then

pb.State := pbsNormal;

end;

Improvements:

By adding the validation rule ‘>=0’ for all levels, negative progress cannot exist in the database. This prevents errors in trying to access levels. This validation rule is a better fix than writing validation into the program, as this abnormal data won’t exist in typical use. Note that there is no upper limit on these levels, as the lexicon size is subject to change and a large user progress doesn’t return any harmful errors.

# 19/12/18

Upon linking the setting elements to the config file, it became apparent that abnormal data could easily persist in the unlocked config file. The user/admin can easily change the values in the config file, which may return errors in the HeadsetSettings form.

Code:

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]);

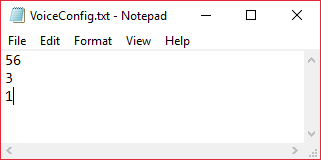
tbRate.Position := strtoint(sl[1]);

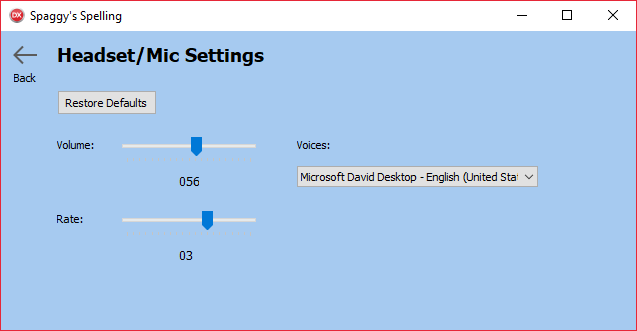
cbVoices.ItemIndex := strtoint(sl[2]);

end;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Function Tested** | **Test Data** | **Expected Result** | **Pass/Fail** |
| 1 | Config file loaded with typical values (normal) | 56,3,1 | UI elements correctly show config | Pass |
| 2 | Config file loaded with max values (boundary) | 100,-10,2 | UI elements correctly show config | Pass |
| 3 | Config file loaded with values that cannot be set (abnormal) | -20,20,20 | UI elements maxed out, no error raised | Pass |

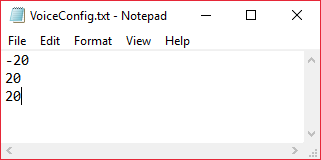
Test 1 evidence:

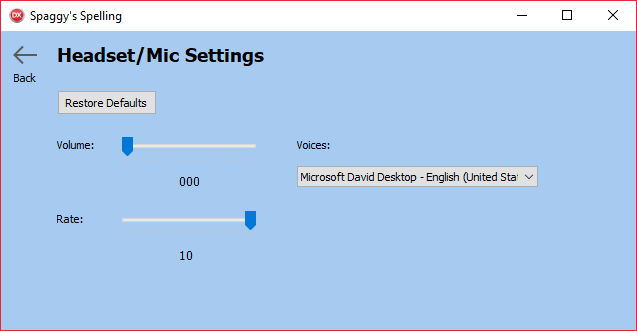
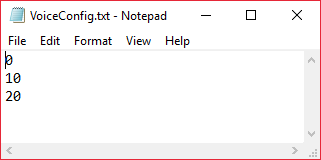




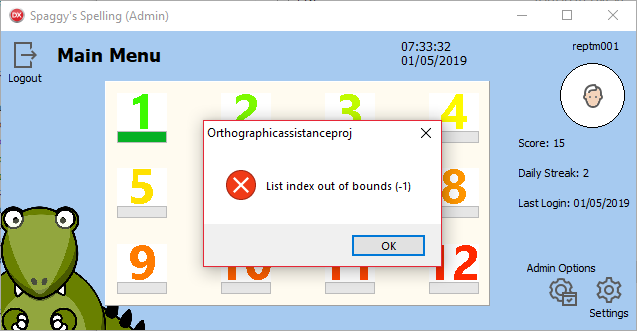
Test 1 successful, with all the UI elements correctly displaying the configuration.

Test 3 evidence:





Test 3 successful, with all the UI elements maxed out and no error raised. Through the onChange event, maxing out the elements resets config file to max values, **other than the index.**



**When the level is loaded with an abnormal voice index, the following error is shown.**

Improvements:

This section checks to see if the stored index is greater than the number of voices available, and sets it to the max index. This way, no errors are raised.

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

begin

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

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

end;

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

end;

# 06/01/19

Testing is performed on the view/hide passwords button on the ViewUsers form, to ensure that the check on the db password is carried out correctly. If this function is broken, then an admin can view passwords on a database they shouldn’t have access to.

Code:

procedure TfrmViewusers.SpeedButton1Click(Sender: TObject);

var

password:string;

begin

InputQuery('Admin: Database Password',

'Enter the password for the database:', password);

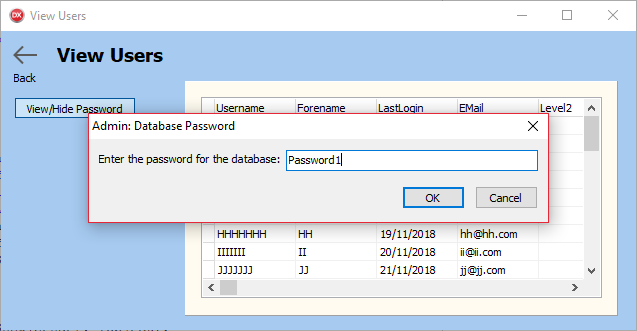
if AnsiContainsStr(globalconnectionstring,password) then

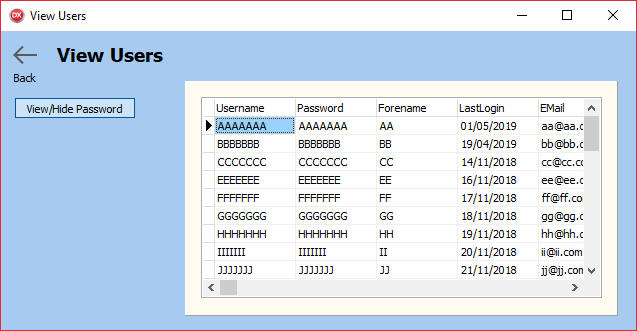
dbGrid.Columns[1].Visible:=SpeedButton1.Down

else

end;

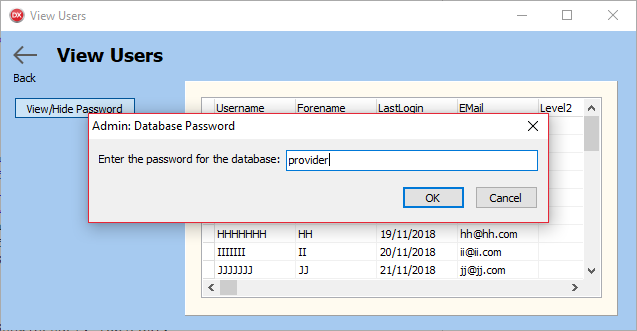
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Function Tested** | **Test Data** | **Expected Result** | **Pass/Fail** |
| 1 | ViewPassword button with correct db password (normal) | Password1 | Passwords in dbGrid shown correctly | Pass |
| 2 | ViewPassword button with incorrect db password (normal) | Password2 | Access not granted, passwords not shown | Pass |
| 3 | ViewPassword button with incorrect db password (boundary) | ‘provider’ (contained within connection string) | Access not granted, passwords not shown | Fail |

Test 1 evidence:

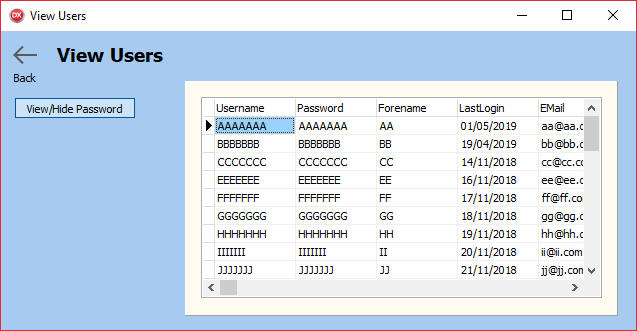


Test 1 successful, with access granted and passwords correctly shown.

Test 3 evidence:



**Test 3 fails, with access granted and passwords correctly shown, despite providing an incorrect password.**



Improvements:

This error was caused due to the function ‘AnsiContainsStr’ comparing the input to the entire connection string. Therefore any text that appears in the connection string would grant access. To fix this, the db password is defined as a global variable at login and this is used for comparison. The same input now no longer works.

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);

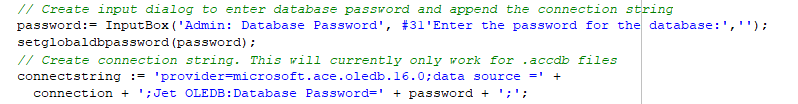
if password = globaldbpassword then

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

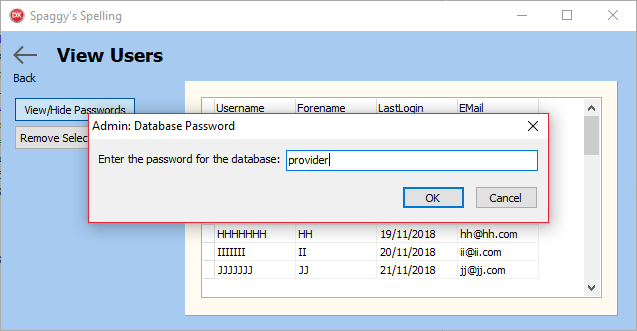
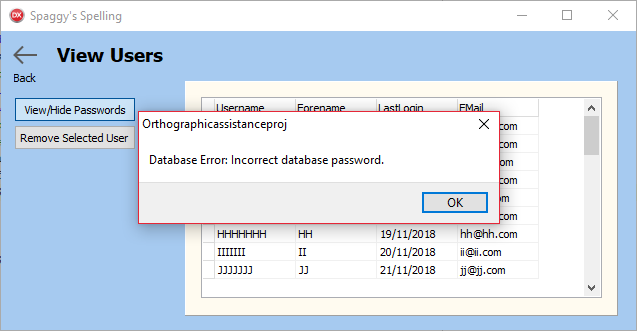
else

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

end;



setglobaldbpassword runs at login and stores the database password.



# 07/01/19

As well as the view/hide passwords function, the remove user function also requires some verification of identity. Following on from the testing made on the view/hide button, updates were made to the checking of the db password. However, the message dialog that makes a confirmation on deleting a user must be tested, since the action is so sensitive:

procedure TfrmViewusers.btnRemoveUserClick(Sender: TObject);

var

password: string;

buttonSelected: integer;

user: string;

begin

(..)

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 if buttonSelected = mrCancel then

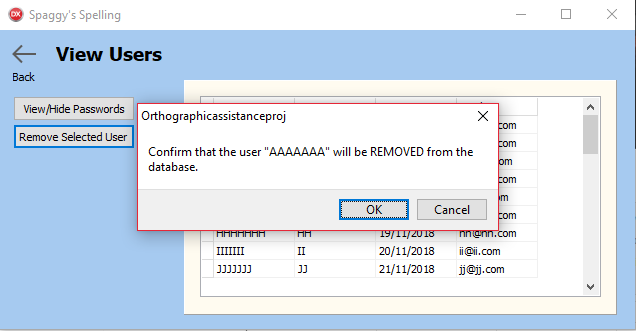
showmessage('"' + user + '" has NOT been removed from the database.');

end

end;

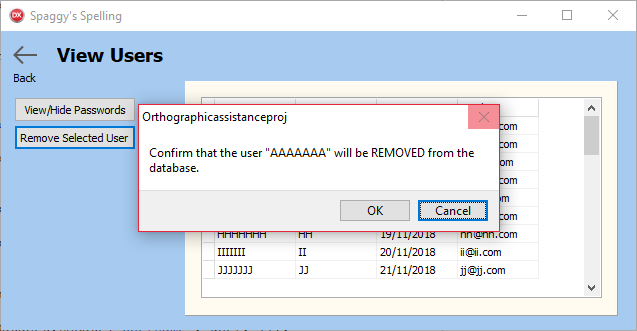
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Function Tested** | **Test Data** | **Expected Result** | **Pass/Fail** |
| 1 | buttonSelected with ‘ok’ pressed (normal) | OK pressed | User deleted | Pass |
| 2 | buttonSelected with ‘cancel’ pressed (normal) | CANCEL pressed | User not deleted | Pass |
| 3 | Dialog box closed by user (boundary) | Dialog closed | User not deleted | Pass |

Test 1 evidence:



Test 1 successful, with the user deleted.

When OK is pressed, the user ‘AAAAAAA’ is removed from the database.

Test 3 evidence:

Test 3 successful, with the user not being deleted. **However, the message informing the admin of this action is not shown. They wouldn’t know the result of the action, causing unnecessary concern.**

When the dialog is closed, the user ‘AAAAAAA’ is not removed from the database.

Improvements:

To fix this issue, the 2nd if statement was removed. This way, any function made to the dialog box other than pressing ok results in the action being aborted and the message being displayed.

procedure TfrmViewusers.btnRemoveUserClick(Sender: TObject);

var

password: string;

buttonSelected: integer;

user: string;

begin

(..)

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 then

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

end

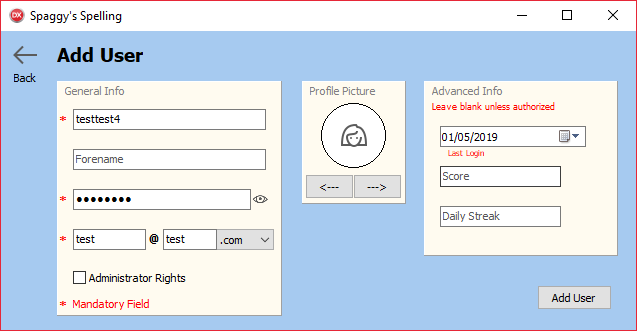
end;

# 14/01/19

The AddUser form, as mentioned in the development diary, contains validation on the client side to prevent breaking validation rules on the server/database side. Testing must therefore be performed on the data entry to ensure these validation rules are met. Otherwise, an error will be made to the database, rendering the program useless. Errors in this form have a direct impact on the functionality of the program, so this testing must be thorough.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Function Tested** | **Test Data** | **Expected Result** | **Pass/Fail** |
| 1 | Adding user with standard information. Optional fields filled (normal) | testtest3,test,testtest ,test@test.com, 3.png, 01/05/19,20 ,20 | User added with correct information | Pass |
| 2 | Adding user with standard information. (normal) | testtest4,testtest ,test@test.com | User added with correct information ,optional fields: default values | Pass |
| 3 | Adding user that meets max validation rules (boundary) | testtesttesttes, testtesttesttes, testtest@testtesttest test.com | User added with correct information | Pass |
| 4 | Adding user that does not meet validation rules (boundary) | test, testtesttesttest, @.com | User not added, admin informed of errors | Pass |
| 5 | Adding user with blank fields (boundary) | ‘’ blank fields | User not added, admin informed of errors | Pass |
| 6 | Adding user with unusual characters and information (abnormal) | ‘@#/?><”’,’ @#/?><”’, @@@.com | User added with correct information | Pass |

Code:

Test 2 evidence:

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;

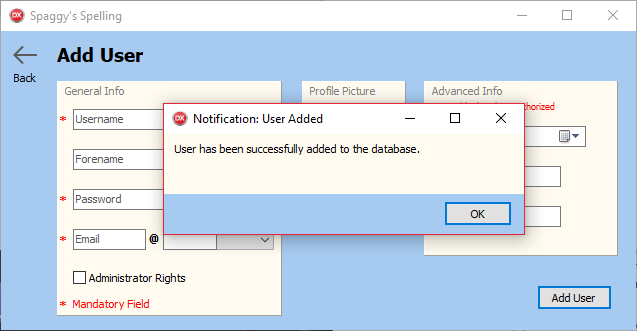
end

else

lblEmailError.Visible := false;

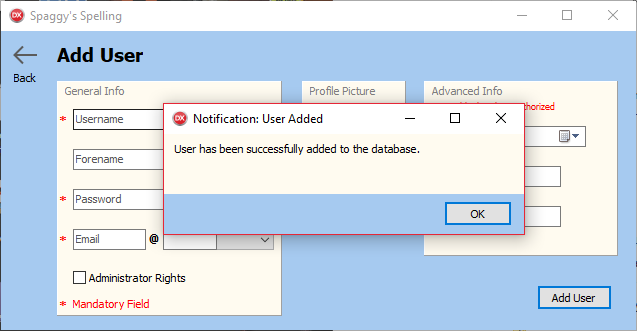
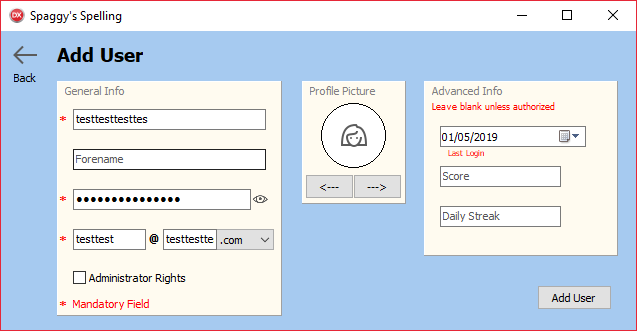
if not Error then

(..)



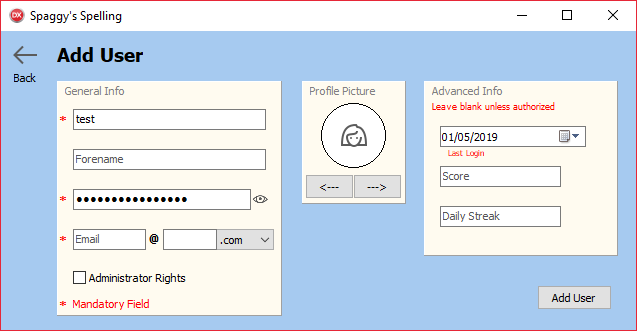
Test 2 successful, with the user being added to the database with the correct information.

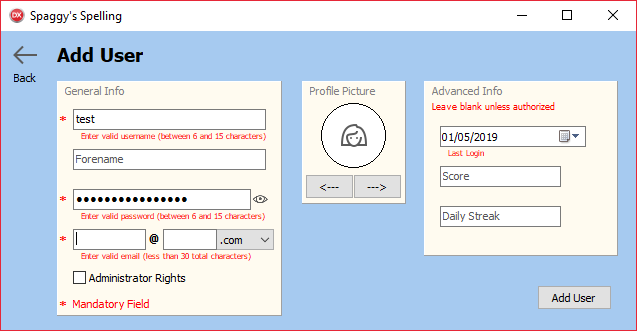
Test 3 evidence:



Test 3 successful, with the user being added to the database with the correct information.

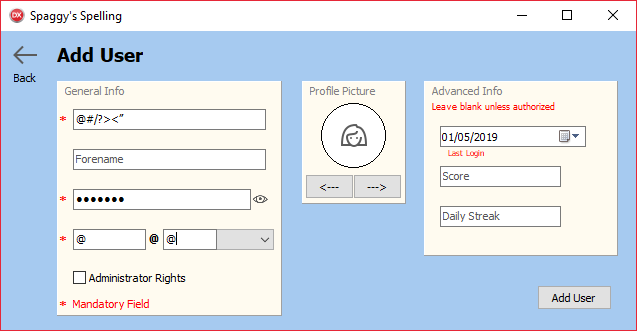
Test 4 evidence:

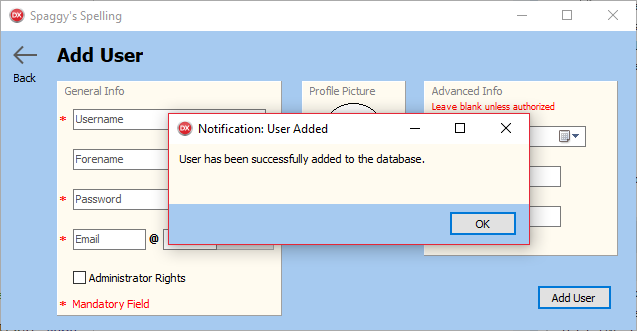




Test 4 successful, with the admin being informed of their errors. User is not added to the database.

Test 6 evidence:





Test 6 successful, with the user being added to the database with the correct information.

These validation algorithms function correctly, as shown through the testing of normal, boundary and abnormal data entry. These algorithms can therefore be relied upon to protect the database from raising validation errors.

# 23/01/19

The level form is the form where the most data entry happens. It requires the user to input their guess for each word that is spoken to them. Since the result of this data entry impacts the user’s score and progression through the game, it is important that the input is handled carefully.

Code:

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(guess) = UpperCase(word) then

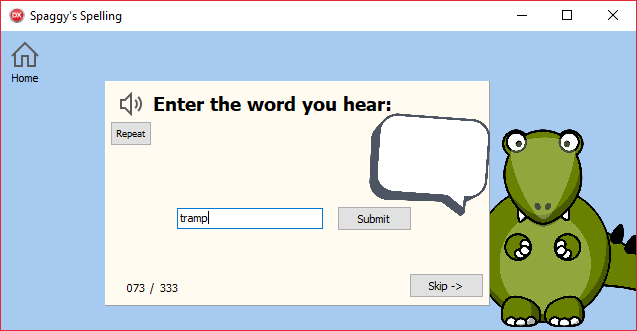
begin

(..)

end;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Function Tested** | **Test Data** | **Expected Result** | **Pass/Fail** |
| 1 | edtGuess with correct answer in lowercase (normal) | tramp | Happy face shown, guess is correct | Pass |
| 2 | edtGuess with correct answer in uppercase (normal) | LIMP | Happy face shown, guess is correct | Pass |
| 3 | edtGuess with incorrect answer in lowercase (normal) | subaru | Sad face shown, guess is incorrect | Pass |
| 4 | edtGuess with correct answer and whitespace (boundary) |  | Happy face shown, guess is correct | Fail |

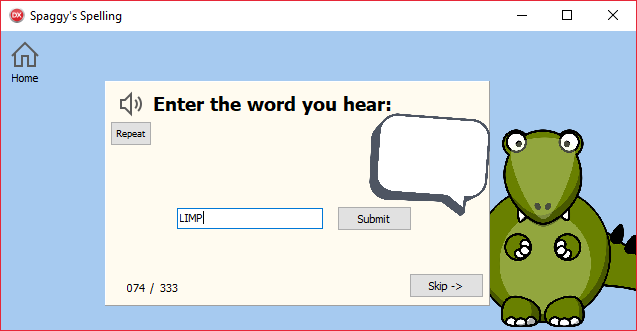
Test 1 evidence:



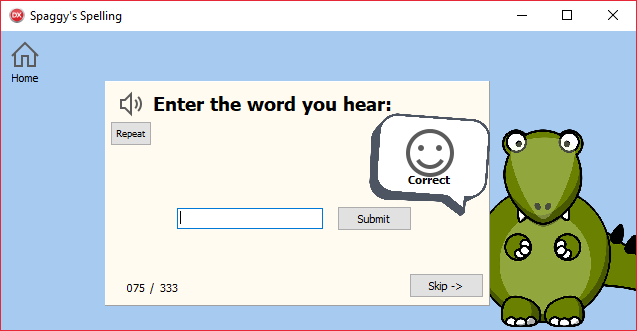
Test 1 successful, with guess being accepted, progress updated, and happy face shown.



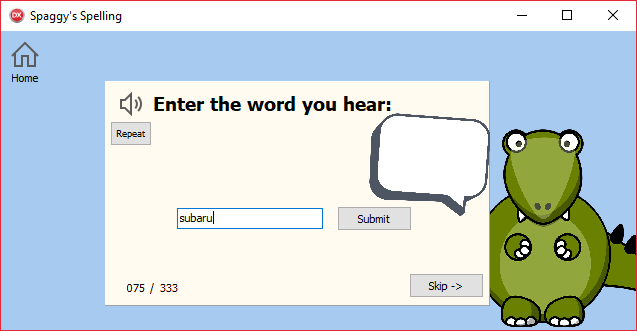
Test 2 evidence:



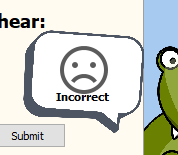
Test 2 successful, with guess being accepted, progress updated, and happy face shown.



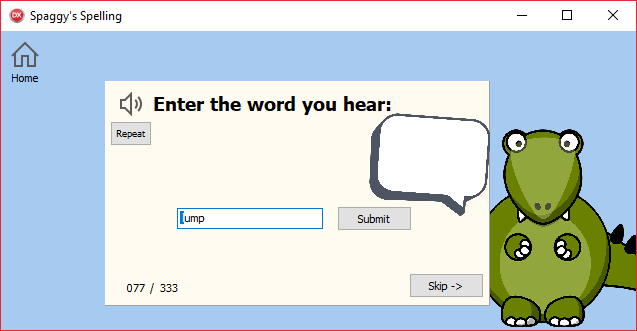
Test 3 evidence:



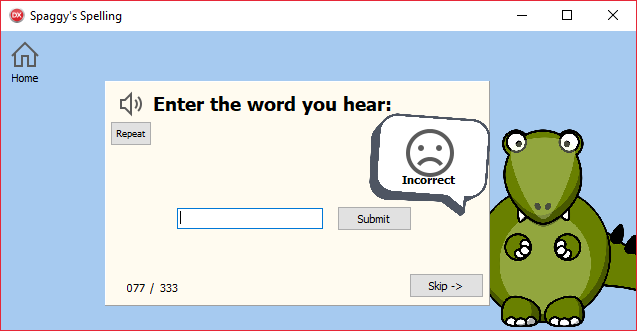
Test 3 successful, with guess not being accepted, progress updated, and sad face shown.



Test 4 evidence:



**Test 4 fails, with guess not being accepted, despite entering in the correct answer. A whitespace should not affect the answer.**



Improvements:

if UpperCase(guess) = UpperCase(word) then

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

In order to remove whitespaces from the start or end of the guess, the function ‘Trim’ was used. Now the program will ignore whitespaces.

