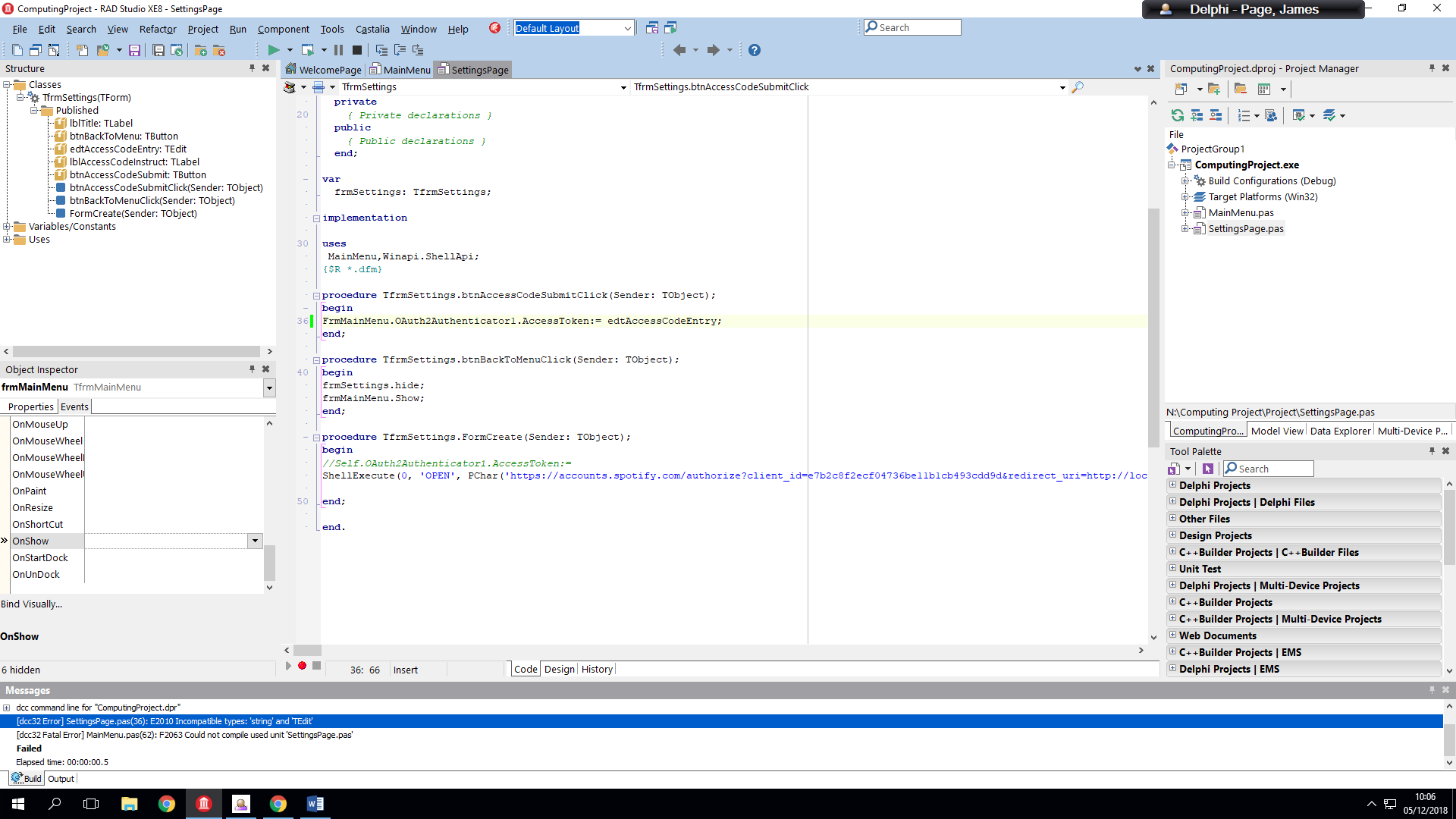
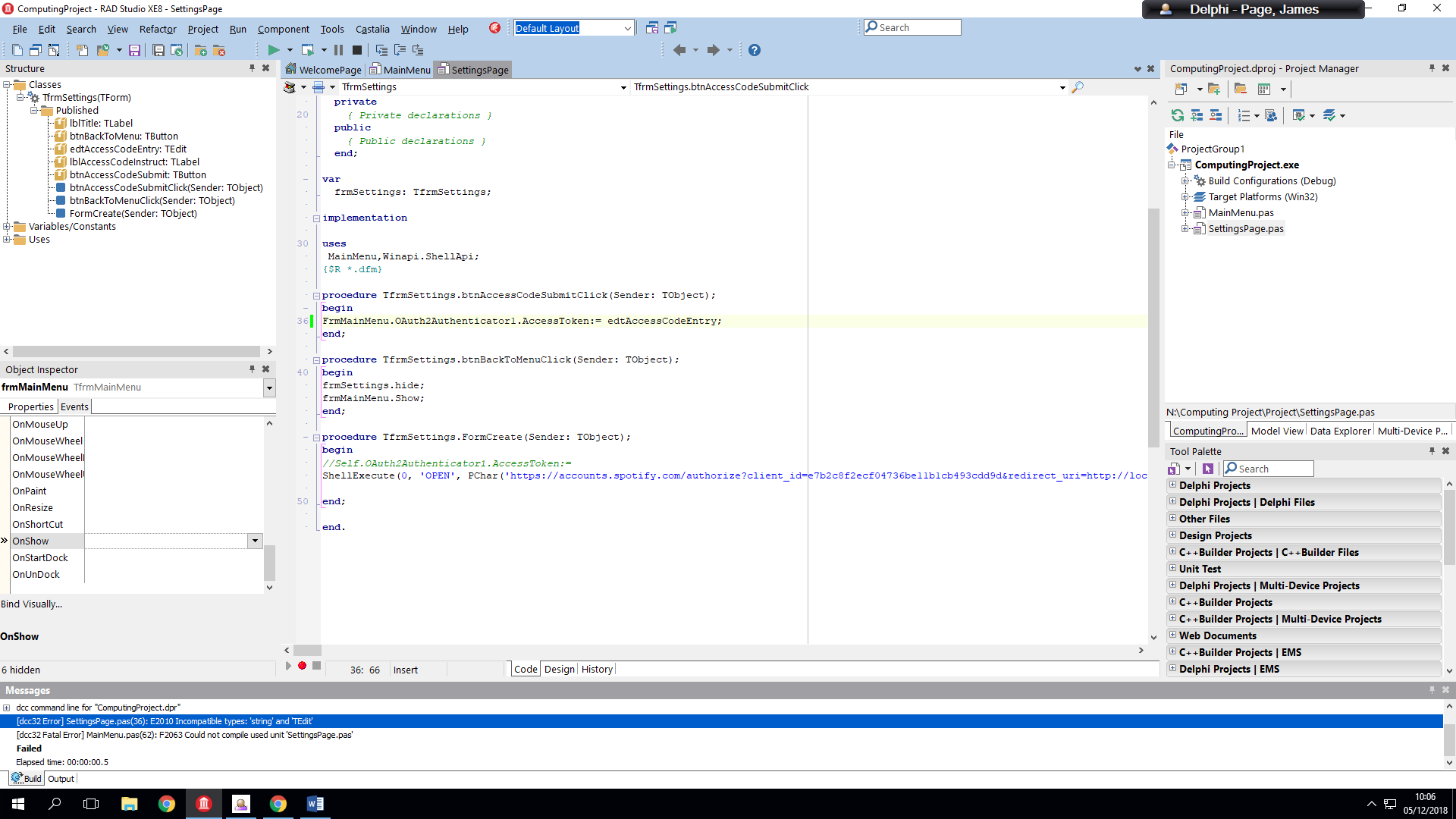
Development

API request development:

Problem:



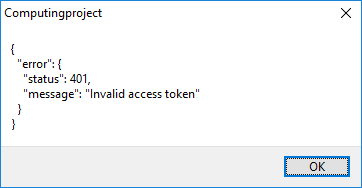
When asking the user to enter the access code from redirect, when trying to apply the text to the Oauth2 object.

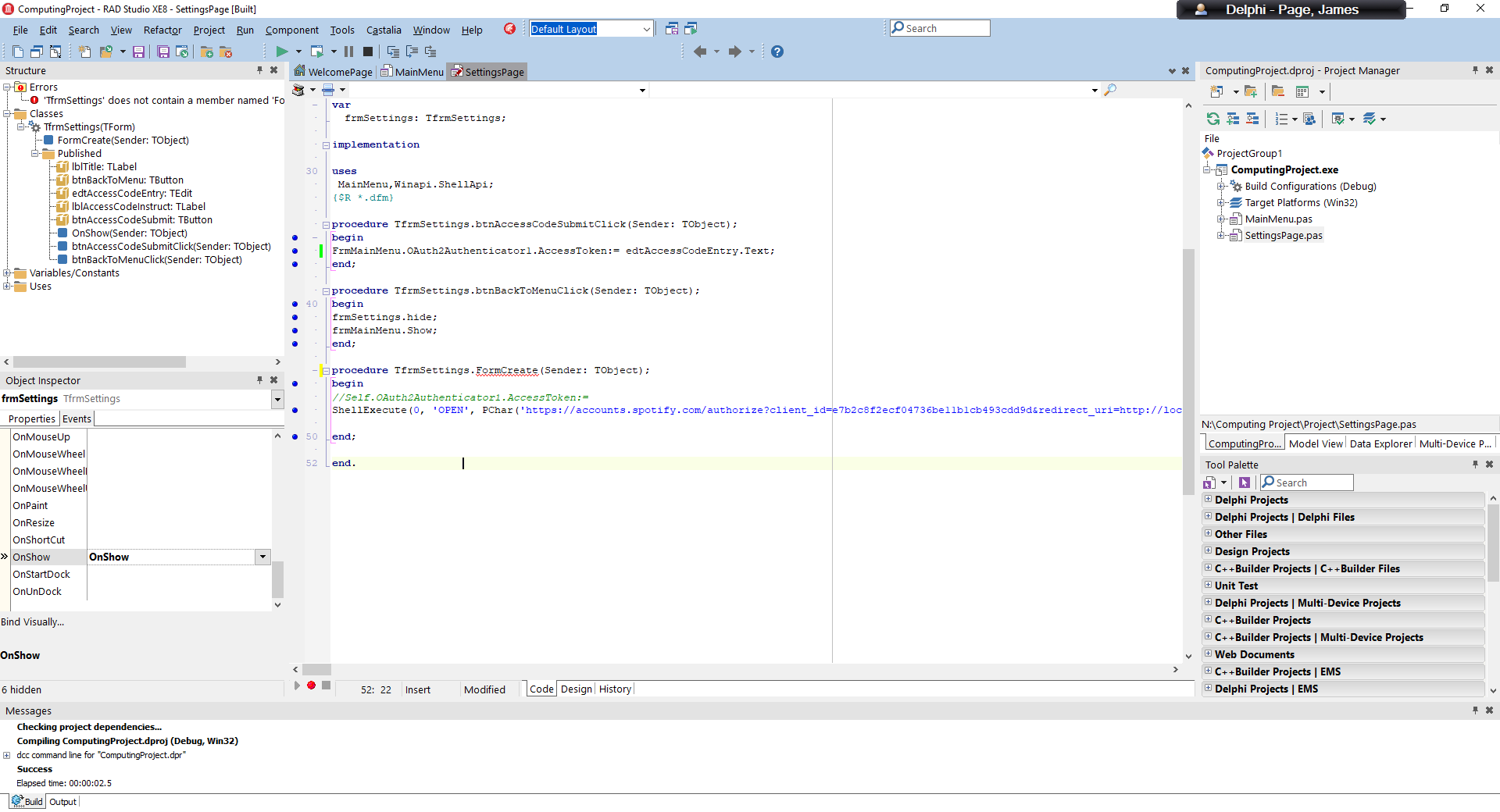
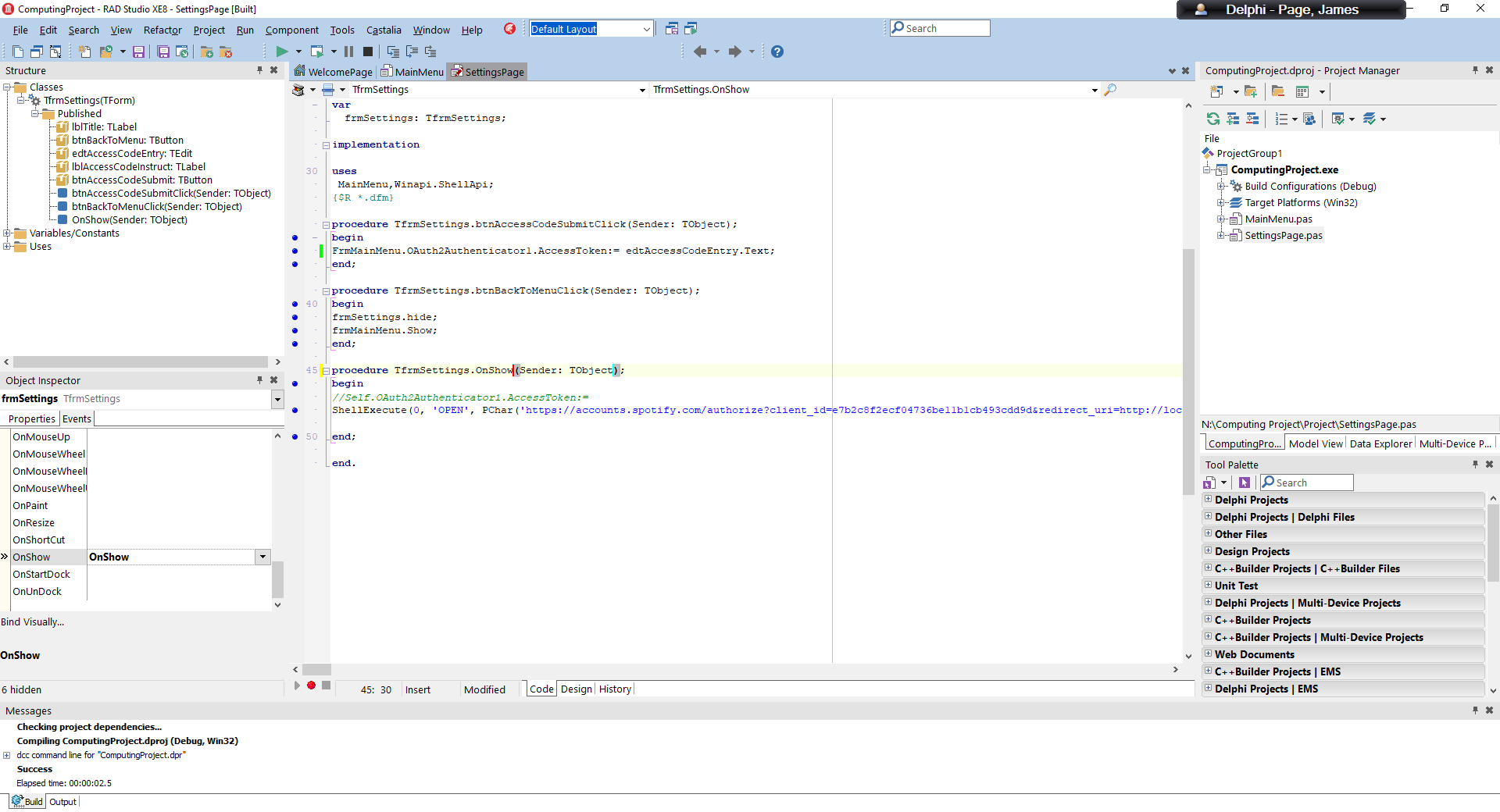
Fix required: *Added edtAccessCodeEntry.Text*

Problem:

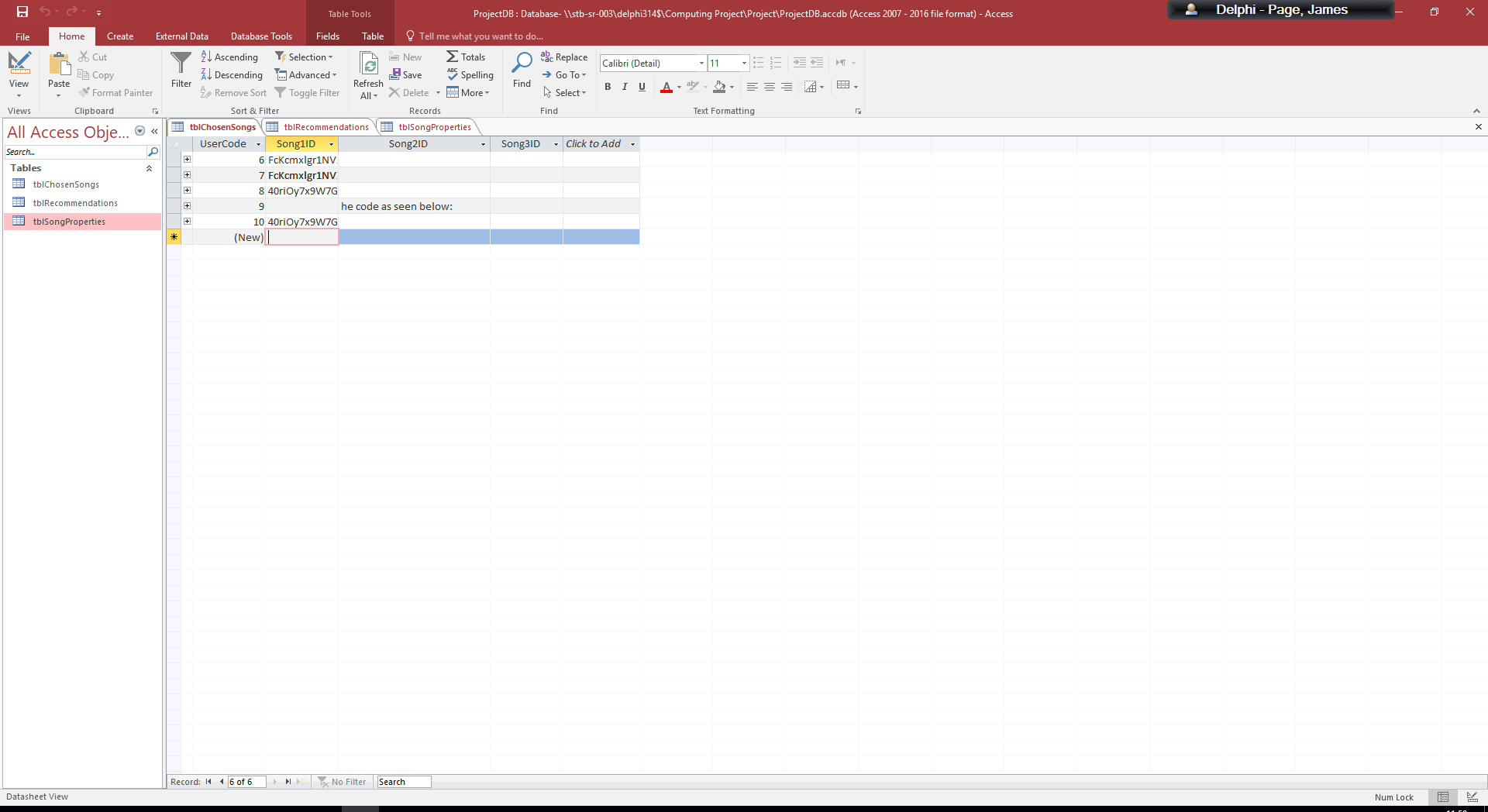
When running the program, the intended outcome is that the settings menu will show and a redirect will pop-up allowing the user to enter the access code returned by the API.

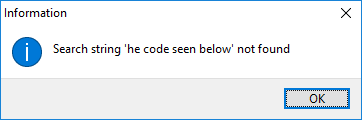
Instead the redirect pops up but the settings menu does not open so when running the program, the user will get a 401 error.



Fix Required: Rather than running it on FormCreate it should be OnShow:

Problem:



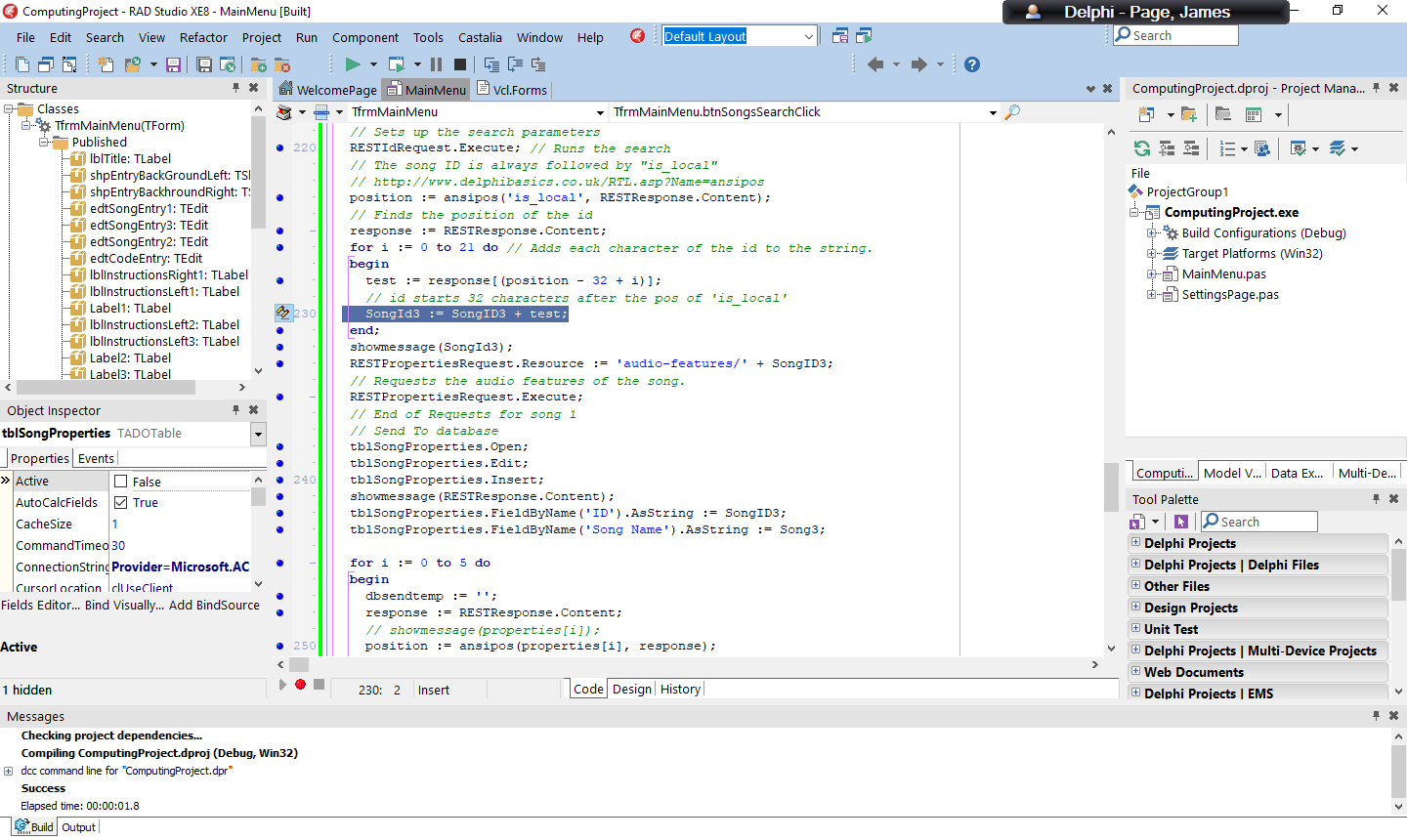


Testing API request + transfer into database:

|  |  |  |
| --- | --- | --- |
| **Test Data** | **Expected Outcome** | **Outcome** |
| Entering 1 song name in field 1 | Shows the returned values in show message and then adds a record to the song properties table. | Show message successful |
| Entering 2 song names | Shows correct request data in show message and then adds 2 new records to the song properties table. | Show message successful indicating successful API request.    First song gets added correctly but blank record created for the second song.  It then adds the correct record next time the program runs, and then until the table is cleared everything is added on a 1 procedure delay. |

Solution:

I found that the problem originated from the third song requests being made, as the code was not correctly fetching the ID and the requests were returning null values which was not working correctly when attempting to format them for database use. After changing one line (highlighted below) to correctly update the songId3 variable it works correctly adding in values are searched.



*Was previously*

*‘SongId1:=SongID3+test;’*

*Due to an error when copying from previous code.*

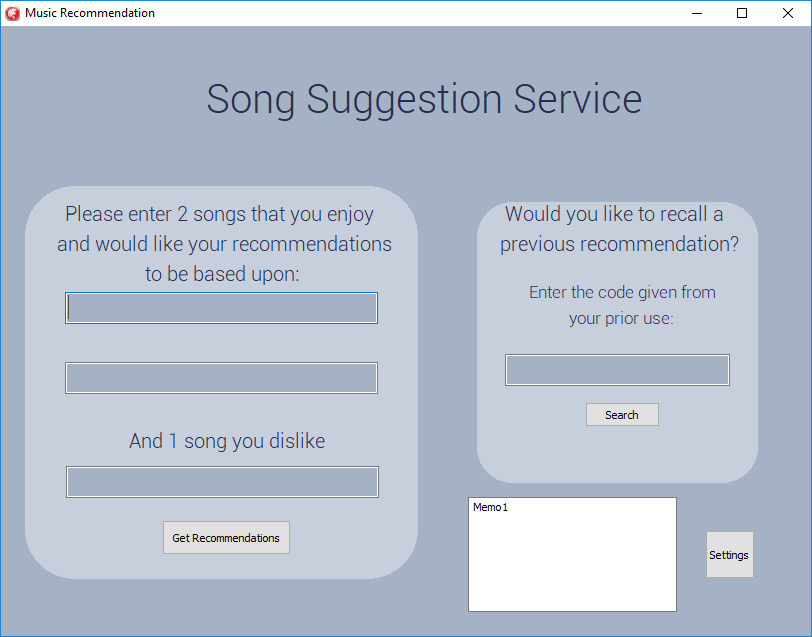
Testing the full API Request:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Data** | **Expected Outcome** | **Outcome** | **Test Pass?** |
| Normal Data  (i.e 3 correct Song Names that are new to the song property table). | Show message for each song indicating a success and the song is added to both the chosen songs table under a unique user ID and the songs are added to song properties table. |  | Pass |
| Normal Data (i.e Correct song names that already exist in the table) | The chosen songs table adds a new id and adds a new set of songs that have been chosen without having and change on the song properties table. |  | Pass |
|  |  |  |  |

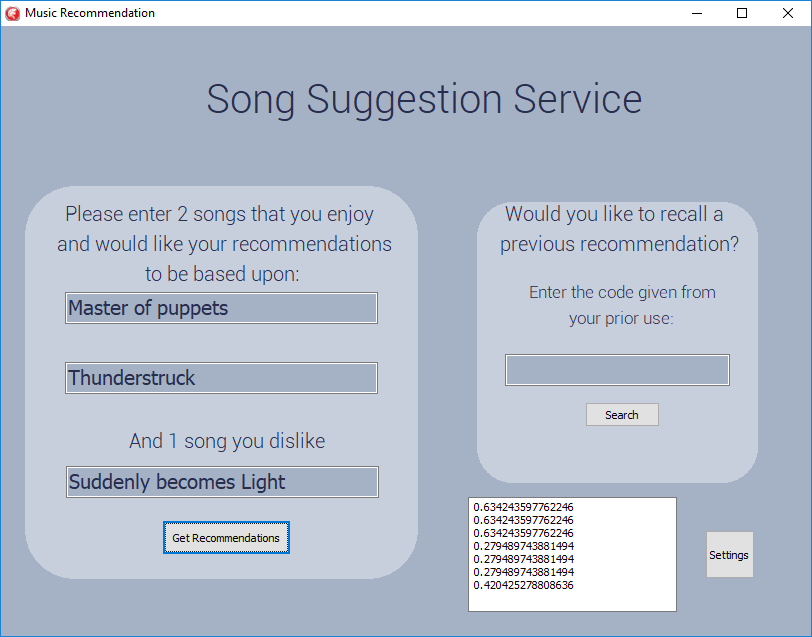
Neural Network training:

When developing the training procedure for the neural network I realised that in order for the program to best be able to learn it would need to have a song to find values that didn’t the user didn’t like as well as songs that they did like to increase the accuracy of the training.

Originally when it didn’t have a value to make low, it would find values very quickly that where large multipliers that meant any song gave a very positive value (it was very good at recognising if a song was a song but not being able to differentiate between different genres and sounds).

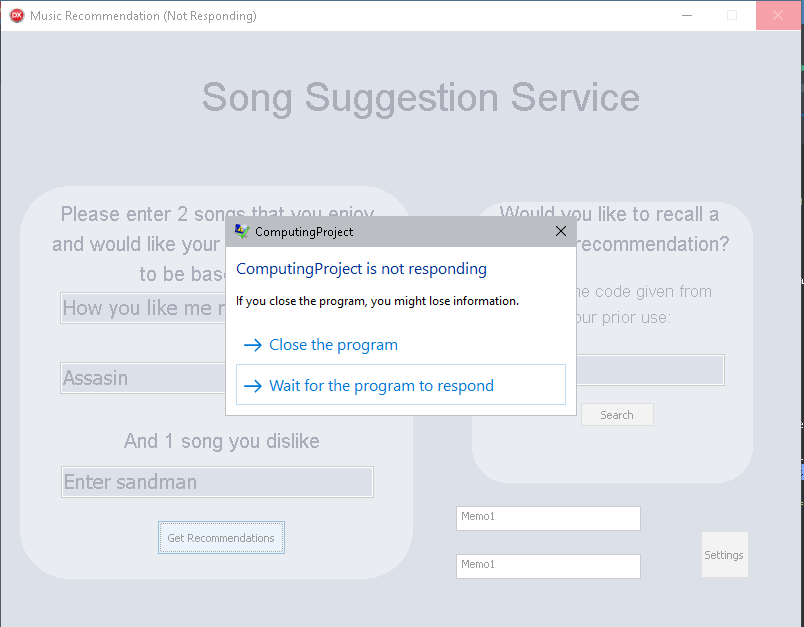
With the current version the neural network struggles to find a set of values that meet the criteria set apart from a few sets of songs which it can find quite easily. The songs that do work are 2 similar songs and the most contrasting style of song as possible. With these scenarios I can demonstrate the proof of concept of the program but unless I redesigned the program to be a lot more efficient it would be unlikely to work for all songs.

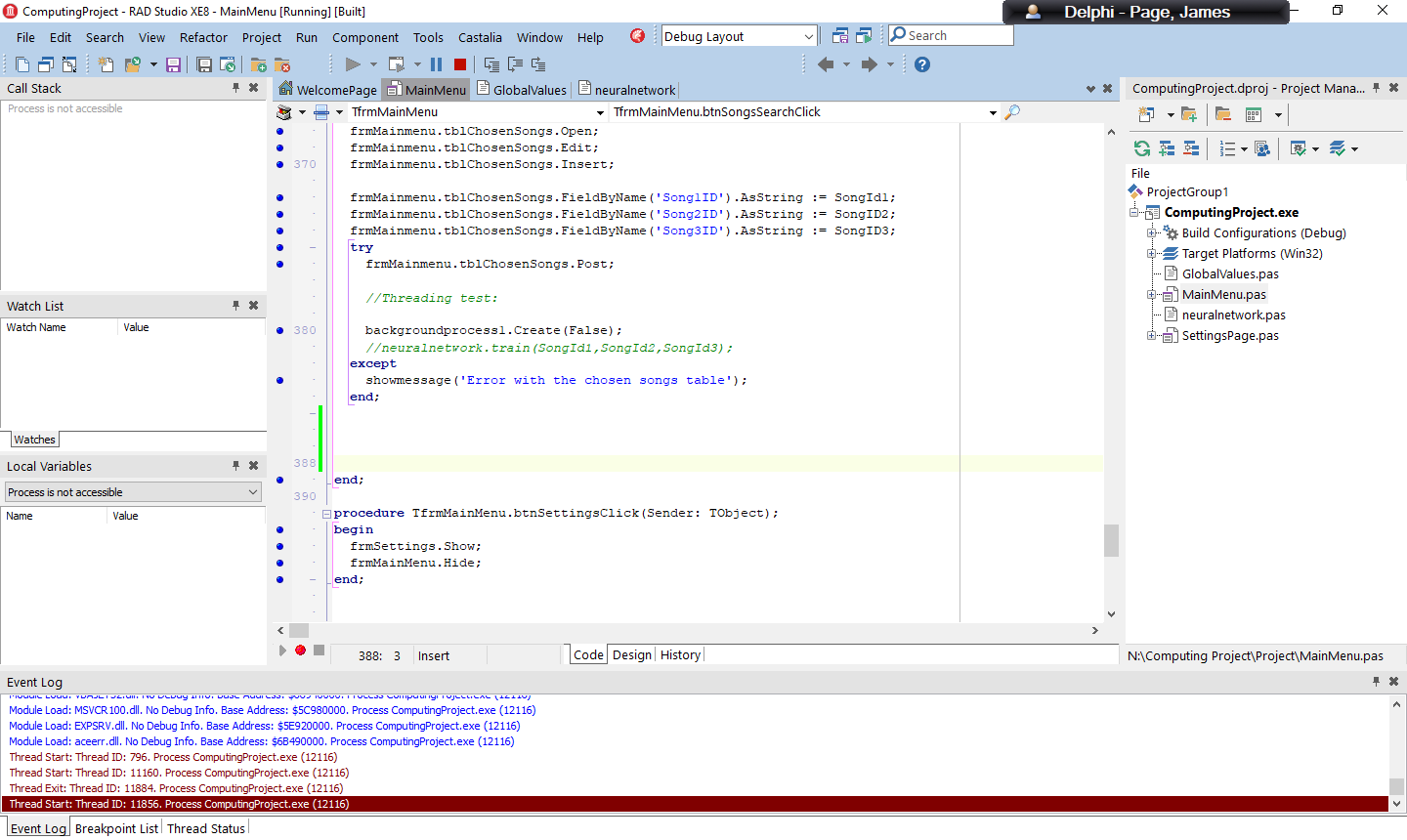
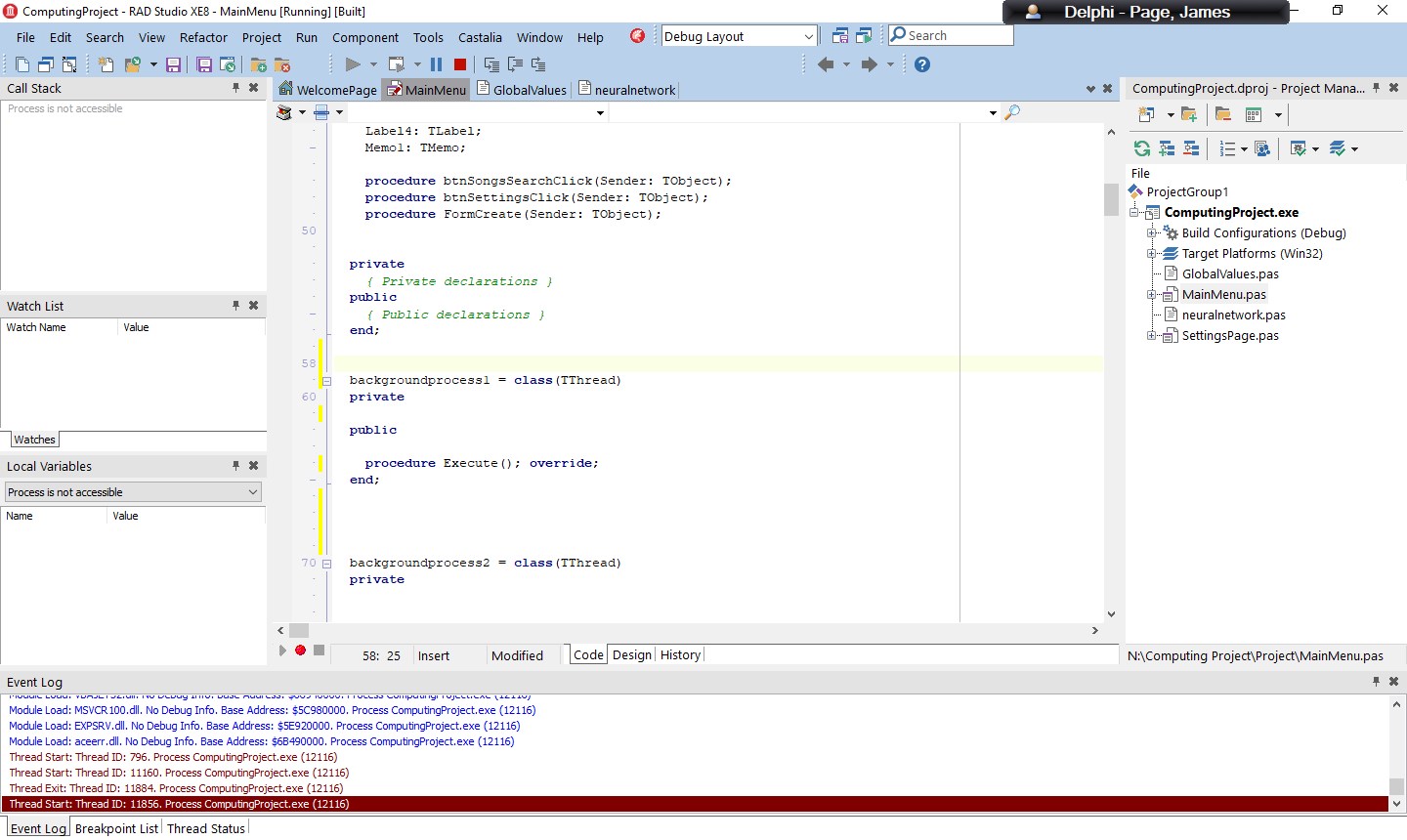
Above is the redesigned main screen to accommodate for the disliked song entry instead of 3 songs the user likes, they enter 2 positive songs and 1 negative that the don’t want. In the bottom right is a memo box used to monitor if the code is actually running, below you can see that when it is running it updates showing each of the values that are output when it is testing to find a valid setup for the network.

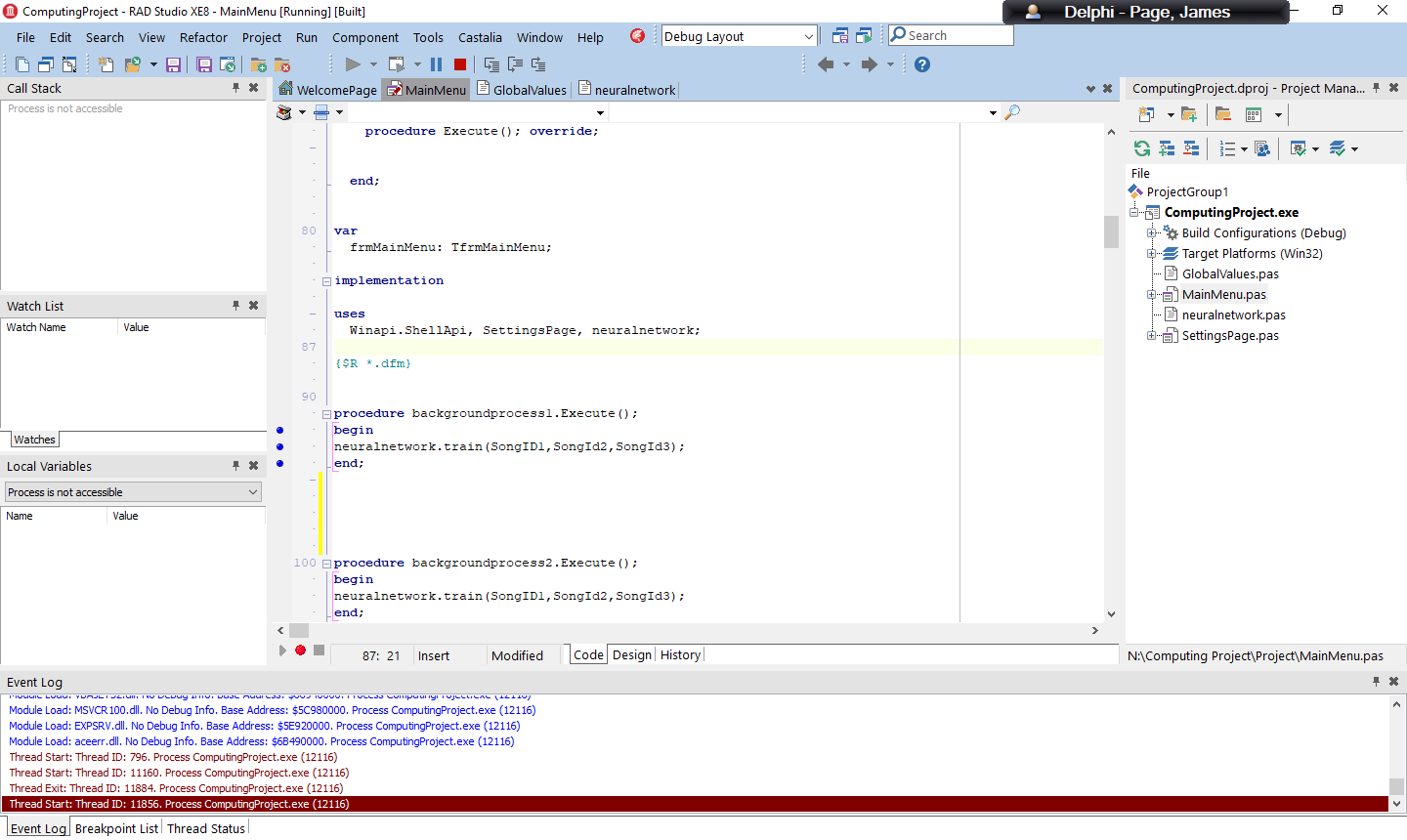


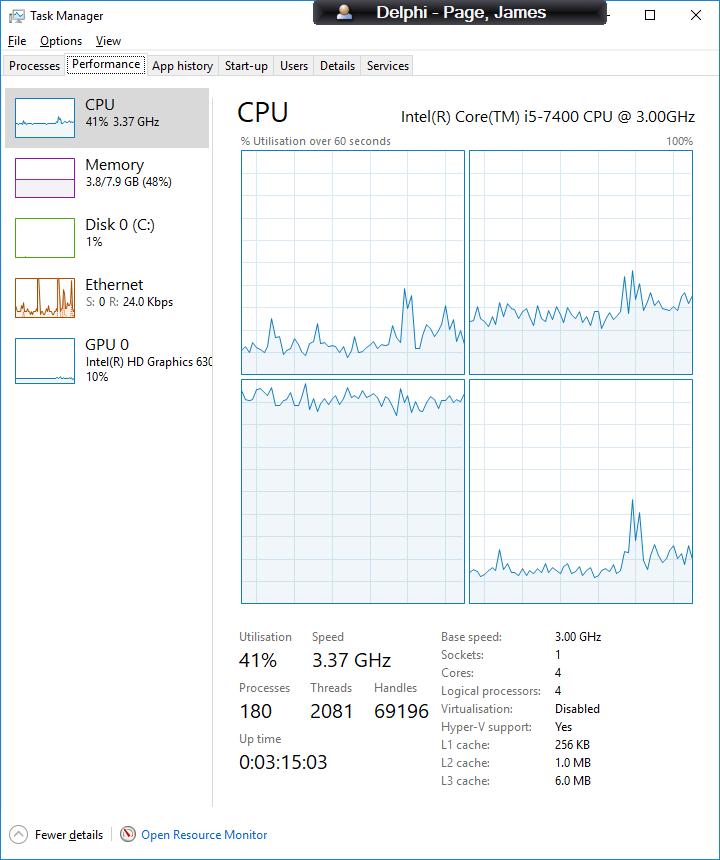
Problems encountered with the training procedure

The procedure that trains the network is a long string of calculations that repeat over and over until it finds a valid configuration. The problem with this is that a standard Delphi program is ran in a single processing thread one command after another, this is fine for simple programs but when Windows finds a program that is running a long process with no breaks for new input, such as an infinite loop, the program will crash and become ‘not responding’.

To circumvent this issue and allow the program to continue running while the calculations are made, I had to add in a background process using Delphi’s TThread functionality.





To do this I had to create a TThread object and replace the point where I would normally call the train procedure with a call to create the backgroundprocess object. This then runs the execute procedure which simply runs the train procedure. This creates the process and runs in the background on a different processing thread (it tends to max out one processor core on task manager).

After fixing the problems with the threading I found that even when the program did not crash while running there were some situations that would not find a setup for the neural network. In response to this I decided to rethink the algorithm that was training the neural network.