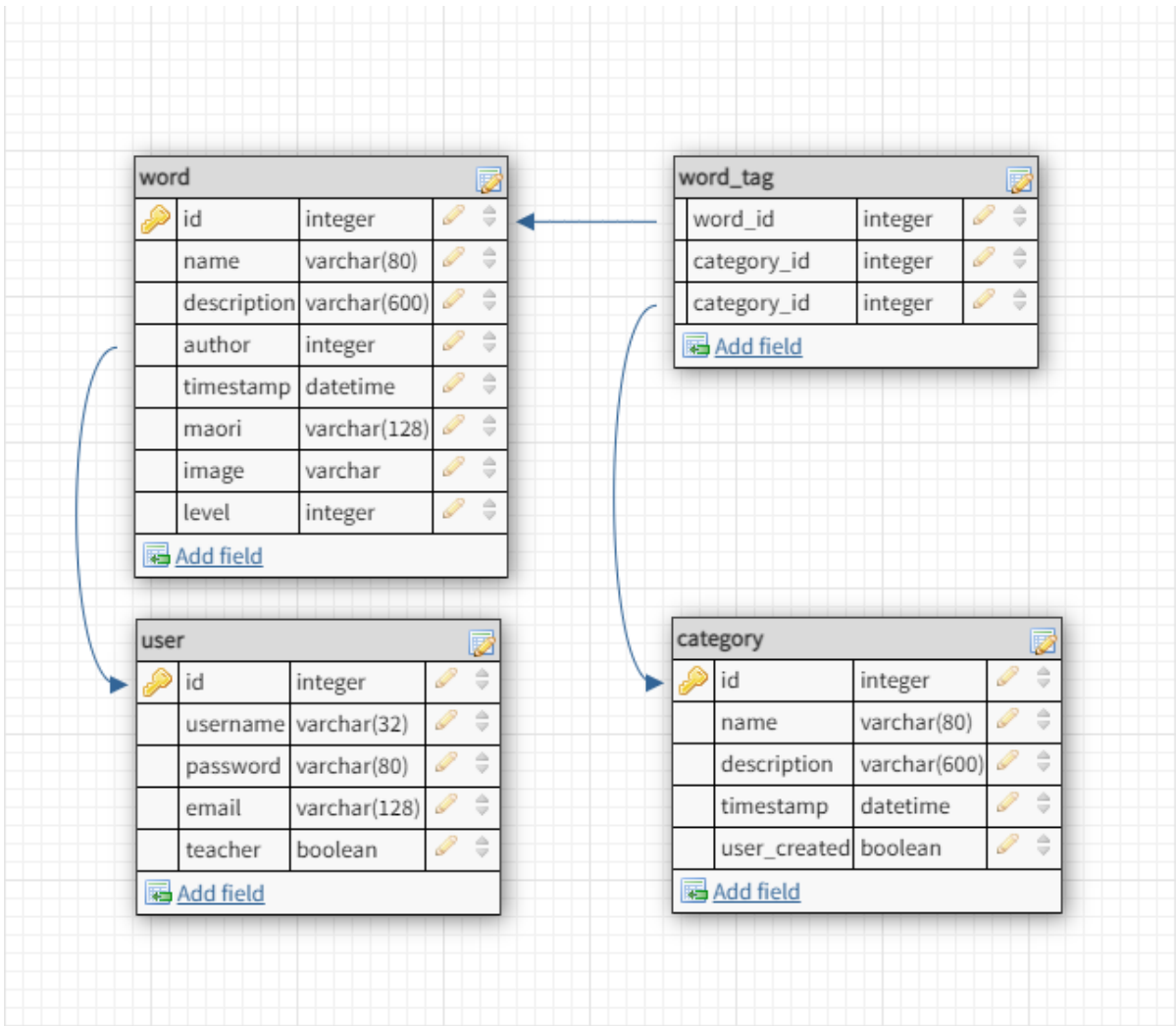


13DTS Web App Documentation

Martyn Gascoigne - 13RKH

Database Schema:

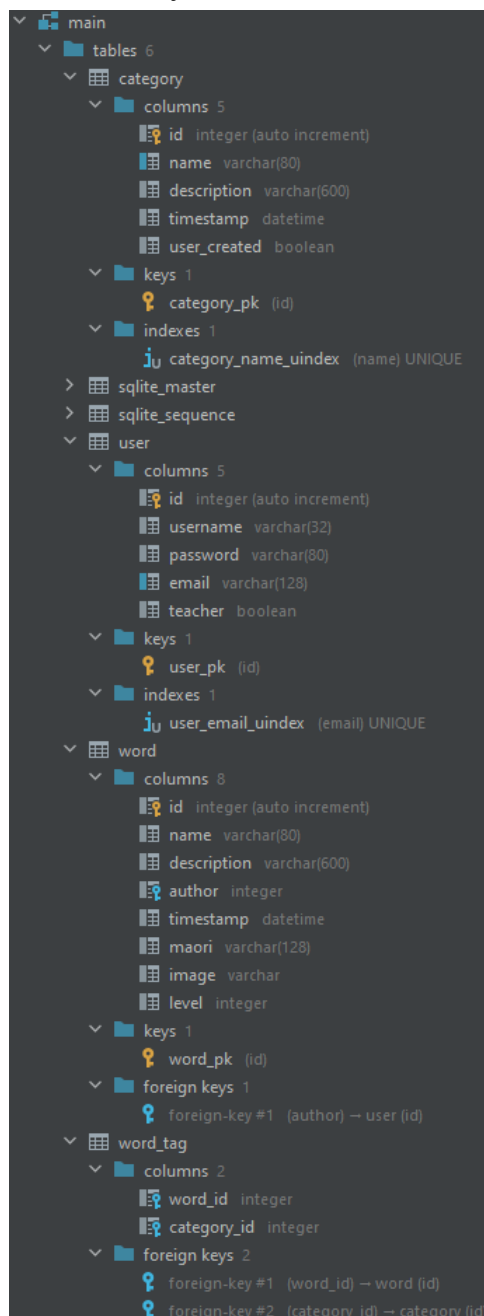


Database Justification:

id (includes word_id and category_id)	username	password	name	maori	teacher
integer - I decided to use an integer for all of the IDs within the database as I can apply the "Auto Increment" tag to these to quickly and easily generate a unique new id for each table entry.	varchar(32) - I decided to use a varchar with the maximum length 32 for the account username as I figured this would be a good length while also not being too long to mess up any formatting.	varchar(80) - I decided to use a varchar with maximum length 80 as I figured this length would be good for password hashing to create a more secure user account system. The password must also be longer than 8 characters.	varchar(80) - I decided to use a varchar with maximum length 80 as I figured this length would cover all of the English words being entered into the dictionary. I decided on 80 characters as I thought that it would be a good safeguard to allow the user some extra characters to work with just in case.	varchar(128) - I decided to use a varchar with a maximum length 128 - longer than the varchar for English. I decided this was a good value as Maori words tend to be longer than English words, and I wanted to allow for this by providing users with extra space.	boolean - I decided to use a boolean for this value as this just has to be either true or false (the user can be either a teacher or not a teacher).

description	author	timestamp	image	level	user_created
varchar(600) - I decided to use a varchar with the maximum length 600. This is to allow for a good length for users to	integer - The author uses an integer for the same reason the IDs use one. The author column directly references the ID value from the user table and as such requires the same value type.	datetime - This is an obvious choice as this value is used for keeping track of the time a word has been edited/submitted.	varchar - I decided to use a varchar with no maximum length. This is to account for all of the possible image names. I didn't want to restrict this as I felt this would help with the longevity of the project.	integer - I decided to use an integer for the level as the level of the word is displayed on the page, so having this as a numerical value is an obvious choice.	boolean - I decided to use a boolean for this value as this just has to be either true or false (the category can be created by a user or not created by a user).

Database in PyCharm:

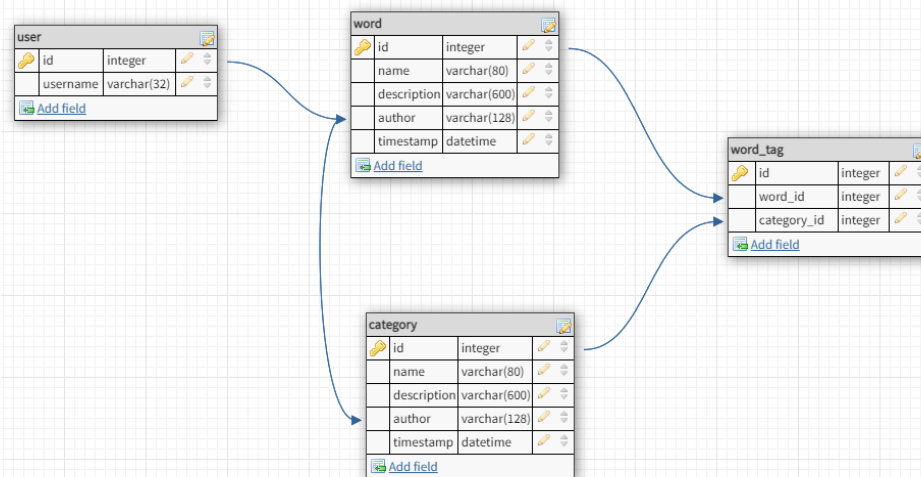


[Link to the Github repository](#)

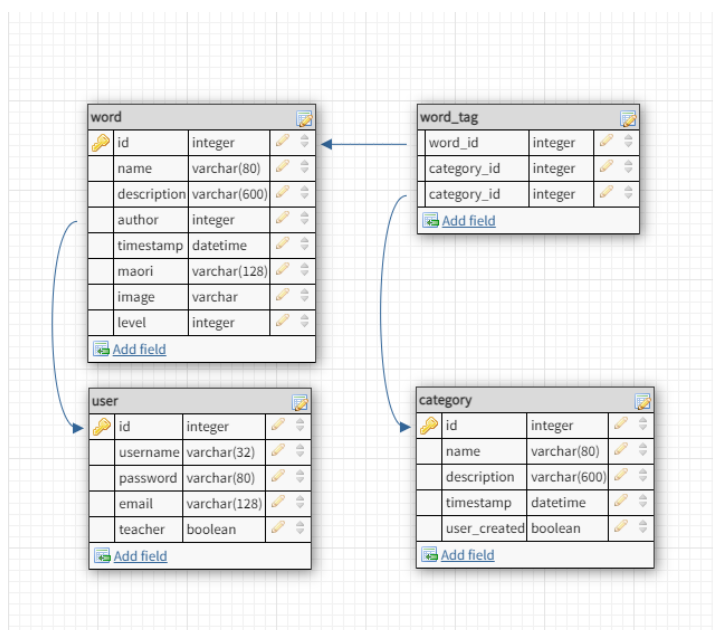
Iterative Developments

Improvement #1: Database schema

Initial schema:



Revised schema:



My initial schema was devised before I began programming my dictionary. This led to some less-than-ideal decisions in terms of the tables and their contents, such as with the user table, which was missing a lot of columns regarding the user information. Other areas of the schema are also unfit for purpose in the final product as they are missing columns or contain unnecessary pieces of data.

However, the initial schema influenced the final schema and it served as a good reference.

User table:

Initially, the user table only contained 2 values - id and username. Initially, this seemed acceptable, however in developing the app I realised that this would be inappropriate for saving and storing the user accounts. Because of this realisation, I decided to add a handful of new columns, mainly for the sign-in process for the accounts. The email and password columns store the entered user data from the user's sign-in form details. The user's password is hashed for extra security, and unhashed when they try to sign in. This adds security to the sign-in process. The teacher value is used to determine whether or not the user can create/edit/delete words and categories. I made these changes to improve the functionality of the database.

Word table:

The word table stayed relatively the same, with only a few additions / changes, a result of realisations during the development process. The author value changed its value type from varchar(128) to integer. This is because I decided to use the user ID to keep track of the author rather than the username, making it more succinct to fetch the user data, as the user ID is sure to be unique due to the way the value is set up in the user table. I added 3 extra columns to account for displaying the word data on the page. In the initial schema, I neglected to add the

Maori, image and level, which are all things that should be displayed on the page. I added these columns, which improved the functionality of the site. Because I changed the value type of the author to integer, I set up a foreign key referencing the ID from the user table, ensuring no word has an undefined author account.

Category table:

Similarly to the word table, the category table was relatively unchanged. However, after progressing with the initial schema I quickly realised some more functionality would be required regarding user permissions to delete categories and words, so made some adjustments to the schema. The first change was getting rid of the author column. Initially, I figured the category table would act in a similar fashion to the word table, displaying the author of the category, along with a description of what was contained within it and a list of its contents. However, I realised an author value for the category seemed redundant as the focus was on the users adding content into the category, not the category itself (as it serves as more of a container for the data and as such should not be the main focus). My second adjustment was adding in a “user_created” column. This boolean value is used to determine whether or not a category was created by a user or added to the database manually. If this value in the corresponding row of a chosen ID is true, the category has been created by a user and as such will display the delete option to users with teacher permissions. This is to ensure that the initial categories added to the system won't be removable by users with teacher permissions to prevent any accidental deletion of the core words by users.

Word tag table:

The final table of my database schema, the word tag table, changed the least out of all of the tables from the initial to final schema. I decided to remove the ID column as I found it to be redundant, as I don't need to keep track of the rows themselves, rather, the contents of each row. The word tag table itself would seem redundant at first glance, as you would assume I could just insert the values into another column within the word table. However, I decided to go about it with a separate table so that the same word could appear in multiple different categories without requiring duplicates of said word. The final schema was shortened to just two columns; “word_id” and “category_id”. These values, as described in their names, take the ID of the word and ID of the category the word is to be displayed in respectively. This is a simple solution to getting a singular word appearing in multiple categories as the program can just reference this table and search for entries with a “category_id” matching that of the current category the user is in. There are two foreign keys for this table, one referencing the ID from the word table, and the other referencing the ID from the category table to ensure there are no values referencing words/categories that don't exist.

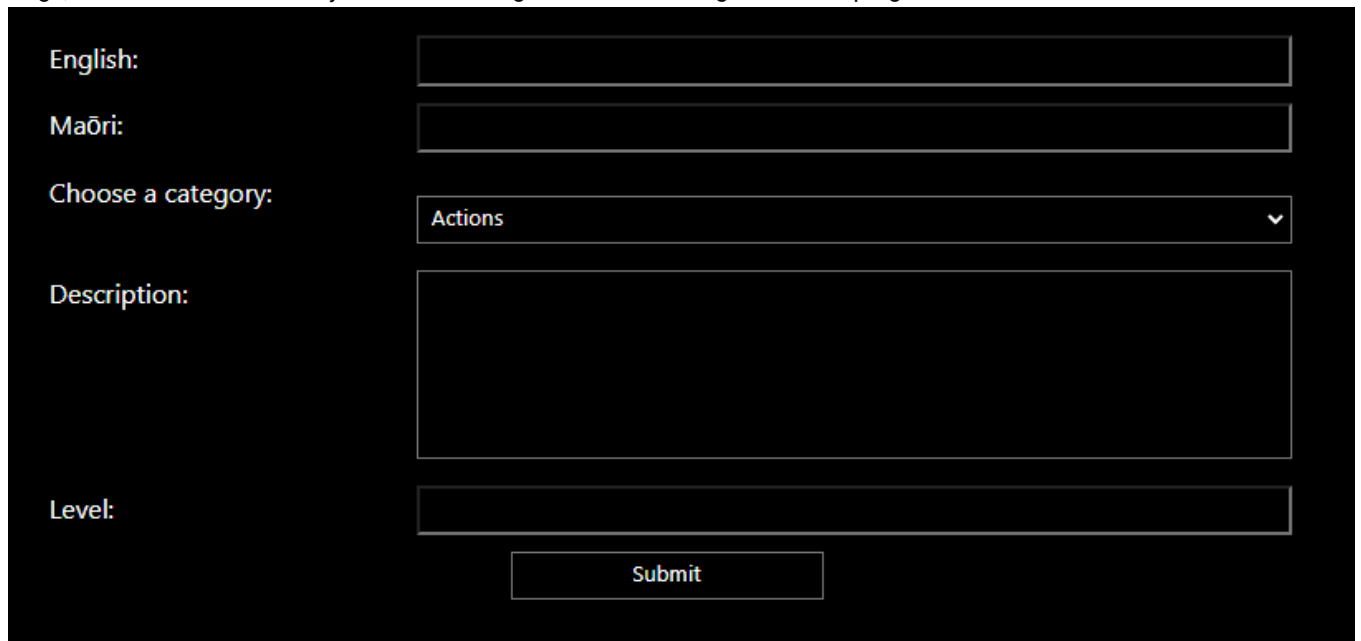
I believe my final schema is a good representation of the requirements of the project, as well as being concise and succinct. It references the initial schema well, and further builds off of it to create an overall effective outcome.

Improvement #2: Changes to UI and Layout

Over the course of developing the dictionary app, I made some improvements to the overall user interface and layout of the app to make it more user-friendly.

Example #1: Level field in add / edit word page

Initially, this field was represented by a text box that only accepted integers. This worked, but I felt it was quite clunky and felt quite unintuitive, as the user was left to do a lot of the work by either typing in the value themselves or clicking the up/down arrows that appeared within the box when hovered over. This also left the issue of users being able to type in values outside of the desired 1-10 level range. These outlier case values are caught by the program, which sets them within the designated range, but I found overall this system was lacking the intuition I sought from the program.



The screenshot shows a form with the following elements:

- English:** A text input field.
- Maōri:** A text input field.
- Choose a category:** A dropdown menu with "Actions" selected.
- Description:** A large text area.
- Level:** A text input field.
- Submit** button.

Thus, I opted to use radio buttons to display the year levels, which I found to be a more clear and understandable way of displaying the level options.



The screenshot shows the same form as before, but with the following changes:

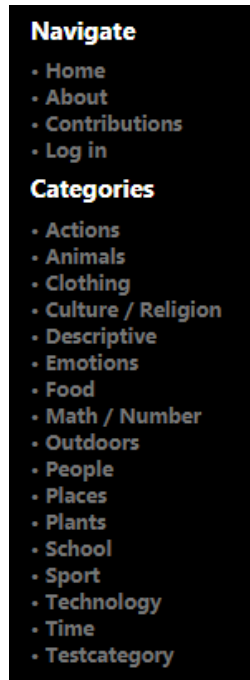
- Year Level:** A row of radio buttons labeled 1 through 10. The radio button for "1" is selected.
- Submit** button.

I feel like this change has greatly improved the user-experience for adding / editing words, as there is a visual indicator for the selected option, as well as a better grasp on the range of year levels the user can select.

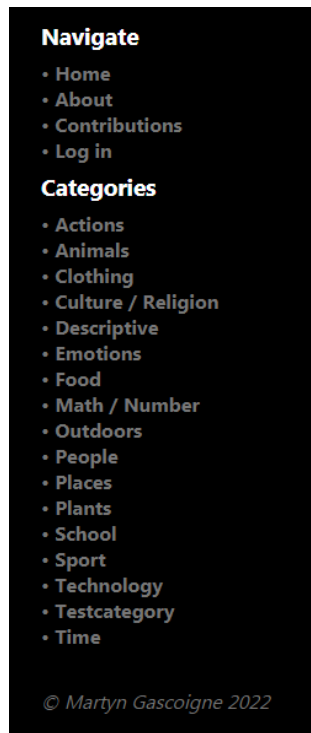
Example #2: Ordered categories in alphabetical order

Originally, the categories in the navigation menu weren't sorted, ordered by the most recent addition to the database. However, I felt this was an un-intuitive way of handling the navigation between categories, as with many websites it is common to display data in alphabetical order, and so following this design standard I would thus be improving the functionality and user-experience with my website.

What the menu looked like originally:



What the menu now looks like:



Although it doesn't appear much has changed, Time and Testcategory have been switched around, a result of me ordering the list alphabetically. I believe this change has subtly improved the user-experience of my site, as it is now more intuitive to navigate the category menus as I have made the menu follow a more standard website design convention of listing data alphabetically.

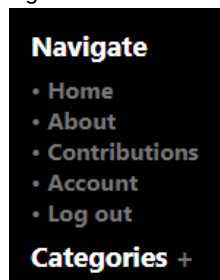
Developmental Testing

Test #1: Expected

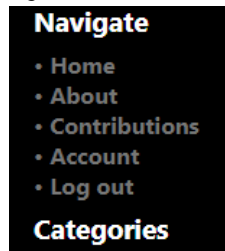
I am testing to see if the teacher can add words and categories, but non-teachers can not.

If the user has teacher permissions, new options will be displayed to them on the website, such as the “Add word” and “Add category” prompts, as well as the option to delete categories and edit words.

Signed in as teacher:



Signed in as non-teacher:



The plus icon (which can be clicked on to add a category) is only visible to a user with teacher permissions.

Signed in as teacher:

A form for creating a new category, visible only to teachers. It includes a 'Navigate' sidebar with a 'Categories +' link. The main form has fields for 'English:' and 'Description:', a 'Submit' button, and a list of existing categories: Actions, Animals, Clothing, and Culture / Religion.

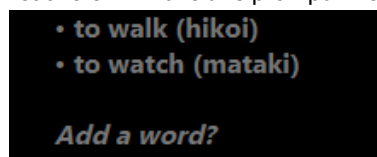
Signed in as non-teacher:

10.50.20.38:5000/?error=No+permission

If a teacher clicks on the plus icon or enters the link “/add_category”, they will be taken to the page pictured above. If a non-teacher tries this, they will be redirected to the homepage with the error message “No+permission”.

Signed in as teacher:

Teachers will have this prompt when viewing a category, allowing them to add a word.



Upon clicking they will be taken to this page:

A form for creating a new word. It includes a 'Navigate' sidebar with a 'Categories +' link. The main form has fields for 'English:', 'Māori:', 'Choose a category:' (with a dropdown menu), 'Description:', 'Level:', and a 'Submit' button.

Signed in as non-teacher:

Non-teachers don't see the prompt.

- to walk (hikoi)
- to watch (mataki)

Upon entering the link:

10.50.20.38:5000/?error=No+permission

Signed in as teacher:

kaukau - to bathe - Delete this word?
To get clean using water.
Level: 7
Date Added: 2022-04-29 16:37:31.312698
Edited By: Asd



English:
Māori:
Description:
Level:

Teachers see the option to edit the word, as well as delete it.

Signed in as non-teacher:

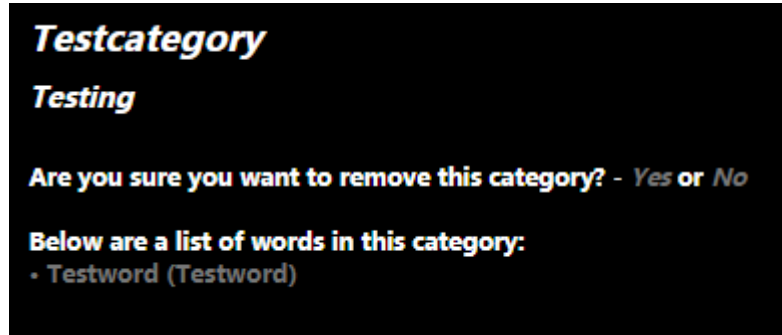
kaukau - to bathe
To get clean using water.
Level: 7
Date Added: 2022-04-29 16:37:31.312698
Edited By: Asd



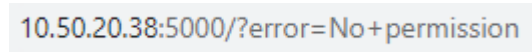
Neither option is visible to non-teacher accounts.

Signed in as teacher:

If the user clicks on the delete category button, it'll take them to this page provided they are a teacher.



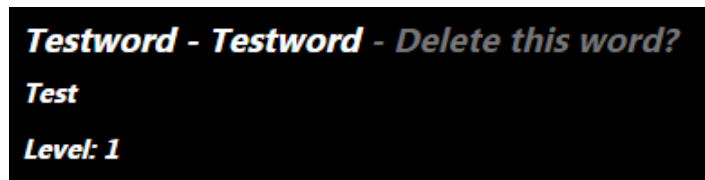
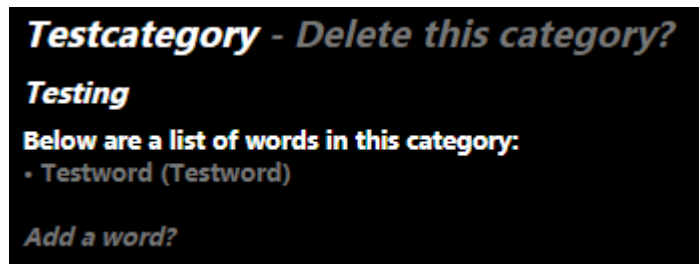
Signed in as non-teacher:



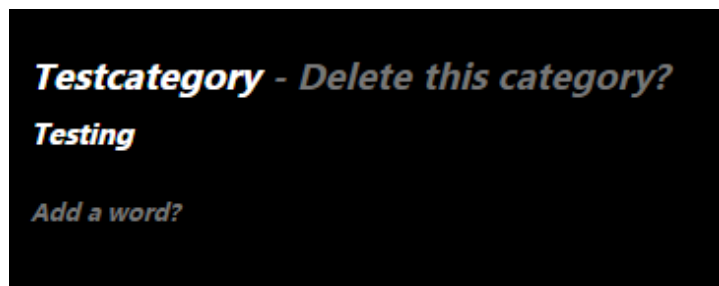
Overall, this system works as expected. The non-teacher accounts are restricted correctly and are limited to basic actions on the site. Anything that would alter or modify the experience for other users is limited to accounts with teacher permissions. Currently, teacher status is enabled on the signup page but if I were to take this website further I would create a new system for requesting teacher access, rather than leaving it up to the user entirely.

Test #2: Expected

I am testing to see if, when the call to remove a word from a database is made, the data is correctly removed from the tables within the database.



If the user has permission (is a teacher), they will have this prompt. This ensures that only authorised users can modify the website.



After the user clicks the delete button, the word disappears from the category page. I expect that all of the data in the database regarding this word will disappear.

The word we just removed:

10.50.20.38:5000/word/304

The word isn't visible in the words table

299	soon	Pending	2	2022-05-04	11:43:49.000493	akoune1	noimage.png
300	sun	Pending	2	2022-05-04	11:43:49.008493	ra	noimage.png
301	tomorrow	Pending	2	2022-05-04	11:43:49.008493	apopo	noimage.png
302	week	Pending	2	2022-05-17	10:01:42.357768	wiki	noimage.png

Nor is it visible in the word tag table

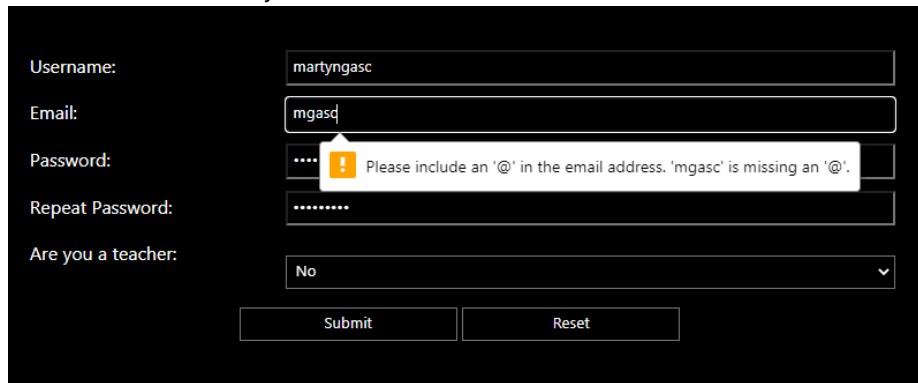
word_id ↕	category_id ↕
278	15
279	15
280	15
281	15
282	15
283	15
284	15
285	15
286	15
287	15
288	15
289	15
290	16
291	16
292	16
293	16
294	16
295	16
296	16
297	16
298	16
299	16
300	16
301	16
302	16

This means the data has been correctly removed, meaning word deletion all works correctly.

Test #3: Invalid

I am testing to see what happens if the user submits invalid data into the signup/login form, such as an email that isn't in the database or an incorrect password, or, if they try to sign up with an email that is already in the database, it denies their request.

User enters an incorrectly formatted email:



The screenshot shows a web form with the following fields: Username (filled with 'martyngasc'), Email (filled with 'mgasc'), Password (filled with '....'), Repeat Password (filled with '.....'), and Are you a teacher? (dropdown menu with 'No' selected). A red error message box is displayed over the Password field, stating: 'Please include an '@' in the email address. 'mgasc' is missing an '@'.'. At the bottom of the form are two buttons: 'Submit' and 'Reset'.

The program informs the user that they need to enter a properly formatted email. This ensures that if I included a system within the program to send an email to the users with updates (for example), there wouldn't be any invalid emails within the database that the program would try to send an email to, ensuring the errors are prevented.

If the user tries to sign up but doesn't match the passwords:

`10.50.20.38:5000/signup?error=Passwords+dont+match`

If the passwords aren't long enough:

`10.50.20.38:5000/signup?error=Passwords+must+be+8+characters+or+more`

This ensures the user's account is more secure as they have to confirm the password that they've entered, as well as creating a minimum length requirement to make the accounts overall more secure.

If the user logs in with an invalid email or incorrect password:

`10.50.20.38:5000/login?error=Email+or+password+incorrect`

I decided to combine these errors into one error message to deter people trying to find a user's email and then guess their password.

My program correctly prevents errors with the login and signup page, to ensure the user doesn't get any bugs when attempting to use the site.

Signup password checking and hashing:

```
# Check passwords
if pass1 != pass2:
    return redirect('/signup?error=Passwords+dont+match')

# Make sure it has appropriate length
if len(pass1) < 8:
    return redirect('/signup?error=Passwords+must+be+8+characters+or+more')

# Hash Password
hashed_pass = bcrypt.generate_password_hash(pass1)
```

Login code:

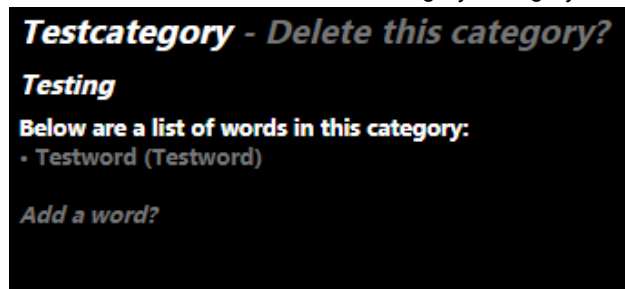
```
# Attempt to log in
try:
    userid = user_data[0][0]
    username = user_data[0][1]
    db_password = user_data[0][2]
    teacher_status = user_data[0][3]
except IndexError:
    return redirect('/login?error=Email+or+password+incorrect')

# Check hashed password against entered password
if not bcrypt.check_password_hash(db_password, password):
    return redirect(request.referrer + '?error=Email+or+password+incorrect')
```

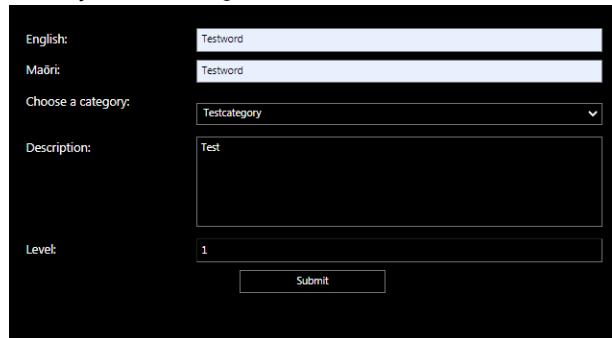
Test #4: Invalid

I am testing to see if the user can enter multiple words with the same name / data into a category.

The word "Testword" is in the "Testcategory" category already.



If we try and add it again with the same data:



We get this error message:

10.50.20.38:5000/?error=Word+already+exists+in+database

This ensures that there aren't duplicates of the same word spread throughout the website. The program informs the user that the word is already in the database. They'll be able to go back to the add word page and change the data appropriately. I decided this feature would be important, as if a user were to add multiple of the same word to the database, they would have to go through and edit every instance of the word when they went to update it. To remedy this, I incorporated the "Word tags" system, which lets one word be added to multiple categories. This means that updating one word will change it globally throughout all of the categories / instances.

Prevention code:

```
# Get list of word names
query = "SELECT name FROM word"
cur = con.cursor()
cur.execute(query)

word_data = cur.fetchall()

# Check if word being inserted already exists in the word table
for find_word in word_data:
    if find_word[0].strip().lower() == word_name.strip().lower():
        return redirect('/?error=Word+already+exists+in+database')
```

We query the name of all of the words in our "word" table and then loop through the query and compare the word's name to the submitted "add word" form. If it finds that the words are the same, it'll redirect the user to the home page before the program inserts the data into the table.

Test #5: Boundary

I am testing to see if I can enter numbers outside of the specified range into the year level of a word when adding it to the database.

The level pictured on the edit page:

English:	<input type="text" value="to bathe"/>
Maōri:	<input type="text" value="kaukau"/>
Description:	<input type="text" value="To get clean using water."/>
Level:	<input type="text" value="7"/>
<input type="button" value="Submit"/>	

If I was to try and add a word:

English:	<input type="text" value="Testword"/>
Maōri:	<input type="text" value="Testword"/>
Choose a category:	<input type="text" value="TestCategory"/>
Description:	<input type="text" value="Test"/>
Level:	<input type="text" value="99999999"/>
<input type="button" value="Submit"/>	

It will automatically clamp the level

Testword - Testword - Delete this word?
Test
Level: 10
Date Added: 2022-05-25 11:17:48.701296
Edited By: Asd
<div>Image Unavailable <sorry></div>

As pictured above, the level panel accepts an integer from 1-10. If a value outside of this range is submitted, it'll automatically confine it within this range. This ensures that no cases outside of this boundary will be accepted, or at least left unaltered before they're submitted into the database. I could make the program throw an error if the level is outside of the range but I prefer this way. I can shift the maximum and minimum year level values quickly and easily within the code if I wanted to allow for more difficult words.

Test #6: Boundary

I am testing to see what happens when a user tries to both view a word that doesn't exist / delete a word that doesn't exist.

Initially, I had this code:

```

@app.route('/remove_word/<word>')
def render_word_remove_page(word):
    # Ensure user has appropriate permission
    if not is_logged_in():
        return redirect('/?error=Not+logged+in')

    if not is_teacher():
        return redirect('/?error=No+permission')

    # Create connection to database
    con = create_connection(DB_NAME)

    # Delete word from the word table at <word>
    query = "DELETE FROM word WHERE id=?"
    cur = con.cursor()

    cur.execute(query, (word,))

    # Commit word deletion
    con.commit()

    # Delete word tag
    query = "DELETE FROM word_tag WHERE word_id=?"
    cur = con.cursor()

    cur.execute(query, (word,))

    # Commit changes and close database
    con.commit()
    con.close()

    return redirect('/')

```

After trying to test deleting a word, the program would throw an error. I narrowed the error down to being a result of the foreign key within the word tag table attempting to reference a word that no-longer existed, due to it being deleted before the word tag entry was deleted. To remedy this, I changed the order of deletion so that the word tag was removed before the word itself, preventing the error.

```

@app.route('/remove_word/<word>')
def render_word_remove_page(word):
    # Ensure user has appropriate permission
    if not is_logged_in():
        return redirect('/?error=Not+logged+in')

    if not is_teacher():
        return redirect('/?error=No+permission')

    # Create connection to database
    con = create_connection(DB_NAME)

    # Delete word tag
    query = "DELETE FROM word_tag WHERE word_id=?"
    cur = con.cursor()

    cur.execute(query, (word,))

    # Delete word from the word table at <word>
    query = "DELETE FROM word WHERE id=?"
    cur = con.cursor()

    cur.execute(query, (word,))

    # Commit changes and close database
    con.commit()
    con.close()

    return redirect('/')

```

After fixing this issue, the program worked as expected and I could carry out my testing. Attempting to access a word outside of the range of word IDs:

🔍 192.168.178.37:5000/word/6000|

🔍 Maōri Dictionary | Home - 192.168.178.37:5000/word/6000

The program prevents an error from arising and redirects the user to the homepage with this error:

⚠️ Not secure | 192.168.178.37:5000/?error=No+such+word

This is because, should the program not redirect the user, it would attempt to grab data from a word that doesn't exist in the first place, returning an error.

Trying to delete a word outside of the range of word IDs:

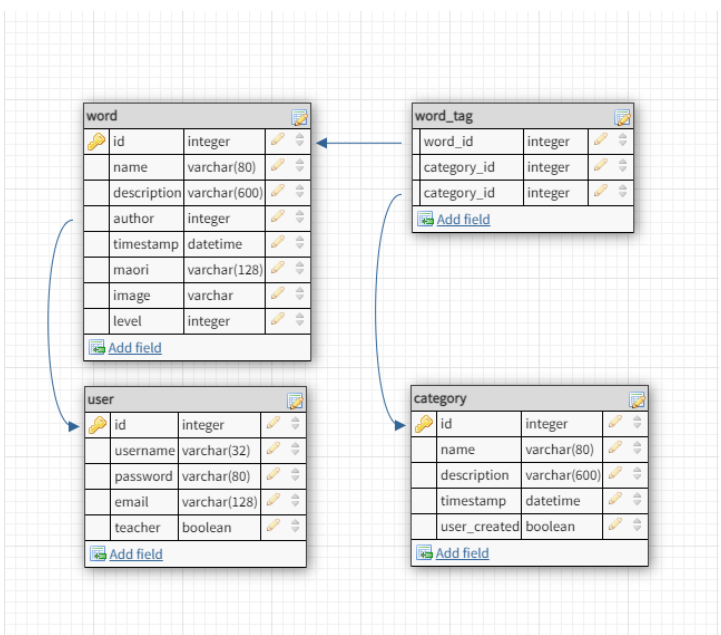
🔍 192.168.178.37:5000/remove_word/6000|


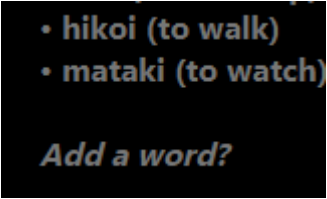

🔍 192.168.178.37:5000/remove_word/6000


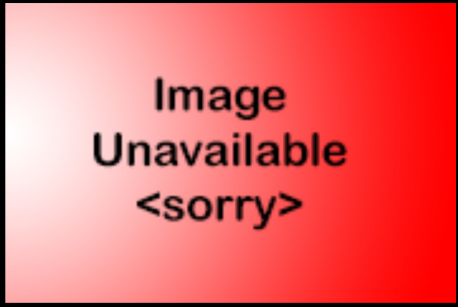
Once again, the program lets the user know that the word they're attempting to remove doesn't exist in the first place.


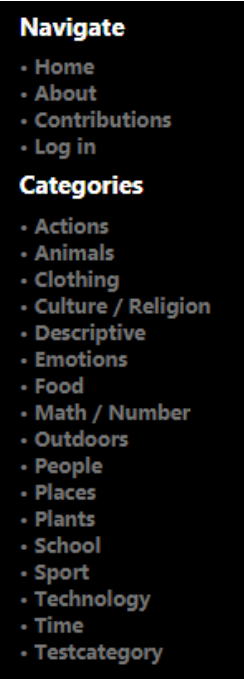
⚠️ Not secure | 192.168.178.37:5000/?error=No+such+word

Functional Testing

What function are you testing?	Proof of functionality	Is the function Successful or unsuccessful?
Database should have a workable table structure which includes multiple tables, with at least two tables being linked together.		Successful - the database is linked together through the use of foreign keys and uses a consistent naming structure, as well as variable structure.
It should allow authorised users to add, and delete words from the dictionary via customised input forms in a	Add category: Categories +	Successful - only authorised users with teacher permissions can view the extra buttons to

<p>password protected admin area.</p>	<div data-bbox="515 73 1233 304">  </div> <p>Add word:</p> <div data-bbox="515 369 842 566">  </div> <div data-bbox="515 575 1233 920">  </div>	<p>add, remove and edit categories and words. These pages are also restricted to these users, and the program will reject those without these permissions.</p>
<p>Data should be sanitised and validated where appropriate</p>	<p>Add word form:</p> <pre data-bbox="515 987 1233 1220"># Only run this code if the form is being posted if request.method == "POST": word_name = request.form['word_name'].strip() word_maori = request.form['word_maori'].strip() word_desc = request.form['word_desc'].strip() cat_id = request.form['category'].strip() word_level = max(0, min(10, int(request.form['word_level'])))</pre> <p>Sign-up form:</p> <pre data-bbox="515 1285 1233 1749"># Post form if request.method == "POST": # Show form in console print(request.form) # Get data username = request.form.get('username').strip().title() email = request.form.get('email').strip().lower() pass1 = request.form.get('pass1') pass2 = request.form.get('pass2') teacher = request.form.get('teacher') # Check passwords if pass1 != pass2: return redirect('/signup?error=Passwords+dont+match') # Make sure it has appropriate length if len(pass1) < 8: return redirect('/signup?error=Passwords+must+be+8+characters+or+more')</pre>	<p>Successful - Only data that should be cleaned up will be touched by the program, leaving the remainder alone. Things such as word names and usernames are cleaned up as having inconsistencies with these could result in a poorer user-experience, whereas data such as passwords which are meant to be each user's key to their account is untouched so that the formatting of their password is entirely up to them.</p>
<p>If an image is not available for a given entry, a placeholder image should be used.</p>	<p>Has an image:</p>	<p>Successful - appropriately displays an image if it's available otherwise displays no image.</p>

	<div><p><i>ngeru - cat</i> Furry pet which eats mice, birds and cat food. Level: 4 Date Added: 2022-05-04 11:43:49.008493 Edited By: Asd</p></div> <p>No image provided:</p> <div><p><i>kaukau - to bathe</i> To get clean using water. Level: 7 Date Added: 2022-04-29 16:37:31.312698 Edited By: Asd</p></div>	
System should be comprehensively tested to ensure that it works as expected.	<div>192.168.178.37:5000/?error=No+such+word</div> <div>10.50.20.38:5000/?error=No+permission</div> <div>192.168.178.37:5000/?error=No+such+category</div> <div>192.168.178.37:5000/?error=Not+logged+in</div> <div>192.168.178.37:5000/?error=Word+already+exists+in+database</div> <div>192.168.178.37:5000/?error=Already+logged+in</div> <div>192.168.178.37:5000/login?error=Email+or+password+incorrect</div> <div>192.168.178.37:5000/?error=Account+unavailable</div> <div>192.168.178.37:5000/signup?error=Passwords+dont+match</div> <div>192.168.178.37:5000/signup?error=Passwords+must+be+8+characters+or+more</div> <div>192.168.178.37:5000/signup?error=Email+is+already+used</div>	Successful - appropriately returns an error message and redirects the user before the program can present an error. The program functions as expected.

<p>Pages should be designed so that the material is easy to read and understand</p>	 <p>toroa - albatross <i>Large bird that lives near the sea.</i> Level: 3 Date Added: 2022-05-04 11:43:49.008493 Edited By: Asd</p> <p>Image Unavailable <sorry></p>	<p>Successful - the page displays the data simply and effectively, making it easily understandable by all.</p>
<p>The site's navigation should be intuitive.</p>	 <p>Navigate</p> <ul style="list-style-type: none"> • Home • About • Contributions • Log in <p>Categories</p> <ul style="list-style-type: none"> • Actions • Animals • Clothing • Culture / Religion • Descriptive • Emotions • Food • Math / Number • Outdoors • People • Places • Plants • School • Sport • Technology • Time • Testcategory <p>All of the categories and navigation options are readily available and easy to view. These items are visible to all users regardless of permissions.</p>	<p>Successful - correctly shows all information in a very clear format.</p>

Animals

Cricket

Below are a list of words in this category:

- toroa (albatross)
- kararehe (animal)
- pea (bear)
- pi (bee)
- manu (bird)
- purerehua (butterfly)
- kawhe (calf)
- ngeru (cat)
- heihei (chicken)
- kau (cow)
- kuri (dog)
- tuna (eel)
- huruhuru (feather)
- ika (fish)
- rango (fly)
- poraka (frog)
- hoiho (horse)
- kiwi (kiwi)
- moko (lizard)
- mokai (pet)
- poaka (pig)
- kioere (rat)
- ngarara (reptile)
- hipi (sheep)
- ngata (snail)
- pungawerewere (spider)
- tukutuku (spider web)
- tohora (whale)
- kotuku (white heron)
- noke (worm)

If you select a category, it'll display the name of each word in both English and Maōri.

Recent Contributions

Here is a list of the recent contributions:

- Testword (Testword) was created or modified
- wiki (week) was created or modified
- hari (to carry) was created or modified
- hopu (to catch) was created or modified
- horoi (to clean) was created or modified
- kake (to climb) was created or modified
- korerorero (to discuss) was created or modified
- ruku (to dive) was created or modified
- moemoea (to dream) was created or modified
- rere (to escape) was created or modified

User Account - Asd

This is the user page of Asd. Below is a list of their recent contributions and edits.

- Testword (Testword) was created or modified by Asd
- hari (to carry) was created or modified by Asd
- hopu (to catch) was created or modified by Asd
- horoi (to clean) was created or modified by Asd
- kake (to climb) was created or modified by Asd
- korerorero (to discuss) was created or modified by Asd
- ruku (to dive) was created or modified by Asd
- moemoea (to dream) was created or modified by Asd
- rere (to escape) was created or modified by Asd
- makere (to fall) was created or modified by Asd

The other pages that include words being listed are laid out in the same format, which also builds the website's consistency. Overall, I find this UI very intuitive.

	<div data-bbox="523 85 847 241"> <p>toroa - albatross <i>Large bird that lives near the sea.</i> Level: 3 Date Added: 2022-05-04 11:43:49.008493 Edited By: Asd</p> </div> <div data-bbox="523 250 1222 725"> </div> <p>The word page itself displays information in a very clear, understandable way.</p>	
<p>The site should allow all users to view the vocab for each topic.</p>	<p>As shown above, all of the content is visible to all users. It doesn't matter whether or not the user has teacher permissions, nor whether or not they're logged in, they will still be able to view the content.</p> <div data-bbox="515 940 1027 1478"> <div data-bbox="515 940 713 1478"> <p>Navigate</p> <ul style="list-style-type: none"> • Home • About • Contributions • Log in <p>Categories</p> <ul style="list-style-type: none"> • Actions • Animals • Clothing • Culture / Religion • Descriptive • Emotions • Food • Math / Number • Outdoors • People • Places • Plants • School • Sport • Technology • Time • Testcategory </div> <div data-bbox="748 940 1027 1478"> <p>Animals</p> <p>Cricket</p> <p>Below are a list of words in this category:</p> <ul style="list-style-type: none"> • toroa (albatross) • kararehe (animal) • pea (bear) • pi (bee) • manau (bird) • puerahua (butterfly) • kawhe (calf) • ngeru (cat) • heihei (chicken) • kau (cow) • kuri (dog) • tuna (eel) • huruhuru (feather) • ika (fish) • rango (fly) • poraka (frog) • hohe (horse) • kiwi (kiwi) • moko (lizard) • mokai (pet) • poaka (pig) • klore (rat) • ngarara (reptile) • hipi (sheep) • ngata (snail) • pungawerewere (spider) • tukutuku (spider web) • tohora (whale) • kotuku (white heron) • noke (worm) </div> </div>	<p>Successful - The content is visible to all users no matter what. Other features such as creating words and modifying them are locked behind teacher permissions, requiring an account, but to view pages there are no restrictions.</p>
<p>Details for each word should be linked to a separate page and should include the word in English, the Te Reo translation, a brief definition, date of entry, Year Level, an image and the author of the record.</p>	<div data-bbox="515 1547 1211 2179"> <div data-bbox="534 1559 874 1715"> <p>ngeru - cat <i>Furry pet which eats mice, birds and cat food.</i> Level: 4 Date Added: 2022-05-04 11:43:49.008493 Edited By: Asd</p> </div> <div data-bbox="534 1718 1198 2175"> </div> </div>	<p>Successful - Demonstrates the desired function appropriately.</p>

	<p>When viewing a word page, the data of the individual word will be displayed, including the English word, Maōri word, description, year level, date added and the most recent editor / author.</p>	
<p>Users with admin access should be prompted to confirm actions which cannot be undone</p>		<p>Successful - the user is presented with a confirm page, as well as showing the user a list of the words in the category to ensure they're sure they're deleting the correct one.</p>
<p>Ensure that your program is well documented with the use of appropriate variable/module names and organised comments that describe code function and behaviour.</p>		<p>Successful - the code is well documented and variable names are consistent and coherent. Functions are documented with their purpose and code is commented thoroughly.</p>
<p>Ensure that you follow python programming conventions.</p>		<p>Successful - the program follows PEP-8 conventions. The only program-perceived errors are of duplicated (where it wouldn't make sense to integrate into a function) and of a supposed variable referenced before assignment ()</p>

	<pre> # Ensure user has appropriate permission if not is_logged_in(): return redirect('/?error=Not+logged+in') if not is_teacher(): return redirect('/?error=No+permission') # Connect to database con = create_connection(DB_NAME) # Ensures the selected category has been created by a user query = "SELECT user_created FROM category WHERE id=?" cur = con.cursor() cur.execute(query, (cat,)) created_by_user = cur.fetchall() # If the query returns nothing, this category must not exist if not created_by_user: return redirect('/?error=No+such+category') </pre> <p>I believe these to be irrelevant and they do not impact the final project in any way whatsoever.</p>	
Ensure that you comprehensively test and debug your program, recording the process.	I have recorded my testing (refer to above developmental testing) and fixed an array of bugs. I've locked down edge-cases and constrained the program to boundaries so as to prevent errors from occurring (through the use of checking permissions, redirects, ensuring data is accessible and preventing users from going further if it isn't, etc.)	Successful - I believe overall my program is robust and effective. It has achieved what I set out to achieve with it and has followed the guidelines set for it efficiently and effectively.

Implications:

I have chosen to write about these implications:

- Functionality
- Aesthetics
- Sustainability and future proofing

Sustainability and future proofing: I believe my outcome is sustainable and future-proof as it can be updated quickly and easily from the browser by users with teacher permissions as well as manually editing the database itself from the backend. The implication covers the topics of keeping the outcome relevant, up-to-date, and improving it when and if it needs to be improved. I believe that my dictionary does all of these effectively. Because users with teacher permissions can directly update the words in the database, there is no need for someone to manage the database from the backend, adding words as they see fit, as this can all be done by people using the site. This ensures that the dictionary can be kept up-to-date with new words, as well as remaining relevant to people who are looking to learn Maori.

Aesthetics: I believe my outcome demonstrates the Aesthetics implication effectively. The provided .css file I felt was quite difficult to see and was too much on the eyes. As a result, I opted for a more simple, monochromatic colour palette. This is much easier on the eyes and the text stands out as it's high-contrast white against black. The interactions themselves remain consistent throughout the pages. For example, links being hovered turning white to indicate the user can click on them and names are displayed in a consistent manner (Maori then English). As a result of this, I believe the dictionary follows the aesthetic implication very closely and adheres to the guidelines it sets.

Functionality: I believe my outcome is functional as it's useful and usable by anyone. The user doesn't require an account to access the words, and the whole program has been tested and secured. The user interface is easily understandable and the individual pages are extremely easy to navigate. I believe it adheres to the end-user requirements, and as demonstrated by my functional testing, follows the guidelines set out by the marker. Because of this, I believe my dictionary is an example of a function outcome.