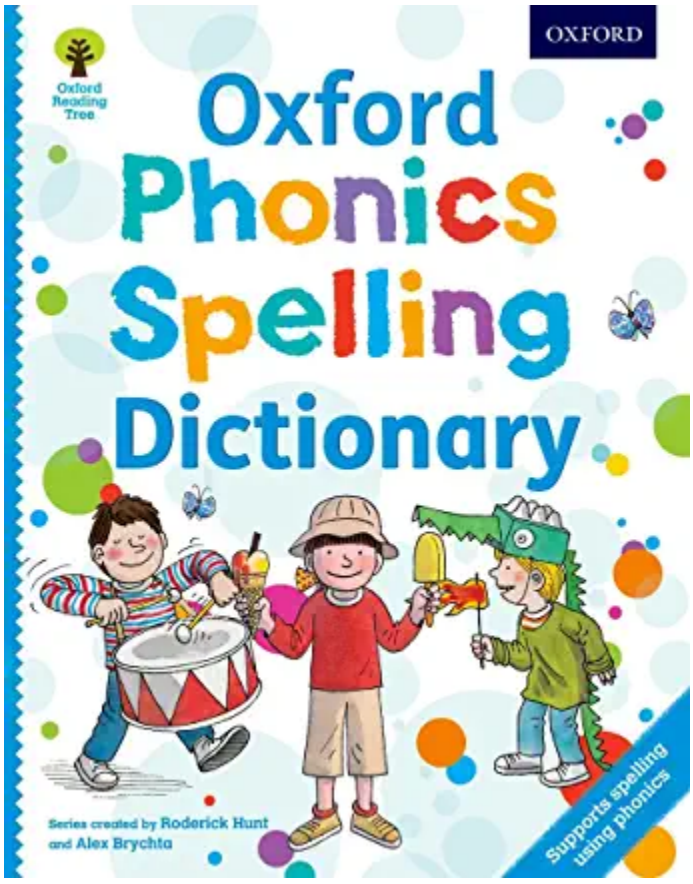
Word Structure Project

Words explained by a database (By Ian Kendall)

Have you ever looked at the first chapter of many language books? There is usually something there that explains how words are pronounced. This is the typical nod to a problem that is assumed will be addressed by a tutor who serves as a resource or reference point for the pronunciation of the reference language. For some languages, this is relatively simple. Spanish and German have very logical writing systems. English does not. The huge number of ‘sight words’ is proof of that. The best approach I have seen so far is this book.



But it only takes looks at parts of the words and leaves you to fill in the blanks yourself. My idea is to take whole words and show the phonics within each one.

What is my project?

So I started with a very simple app run on python that could take a few hundred words and run through when at random to bring students through various levels. I discovered that each word could be distilled down to a JSON element.

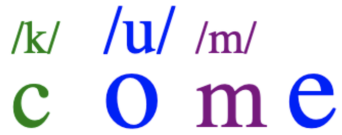
{

```
        "word": "conscious",
        "phonics": "/k//o//n//sh//u//s/",
        "links": "0,1,2,3,3,3,4,4,5",
        "irregulars": "3"
}
```

The example above shows how this is put together. The “links” entry is the same length as the word and maps each letter of the word to the sound it helps to create. The “3” in “irregulars” shows that “sci” does not usually produce the “sh” sound and can be marked in red by whatever app it appears in rather than turning the entire word into a sight word. For example:

Similarly, it covers the situations where the letters indicating the sound do not occur contiguously. This though quite standard can be quite disconcerting.

```
 {
        "word": "come",
        "phonics": "/k//u//m/",
        "links": "0,1,2,1",
        "irregulars": ""
}
```



So a backend that can provide these JSON nuggets could be a wonderful resource to teach reading in English. But there’s more. Phonics is not just good for learning to read but for memorizing numbers.

What frontend apps can use it?

I have two examples and a proposed third.

Say the word.

The proposed app illustrated above checks and improves your reading for given levels. This is especially good for spelling bee preparation or just word learning. But why stop there. To date I have managed to compile over 1700 words for this system. It's time to expand a little further.

Hangman

This is a well known parlor game for word guessing. It's purpose, since Victorian times, was to teach spelling and word recognition. I would like to expand this and I now have enough words to do it. Have you heard of Harry Lorayne or Tony Buzan. They have written books on a major system that uses words as numbers so that sentences can be used to memorize large numbers. While it is easy to translate in your head from the words to the number it is difficult to go the other way. Unless you have a phonics database. Using SQL code shown (this is only a small part of it) I was able to convert phonics into mnemonics:

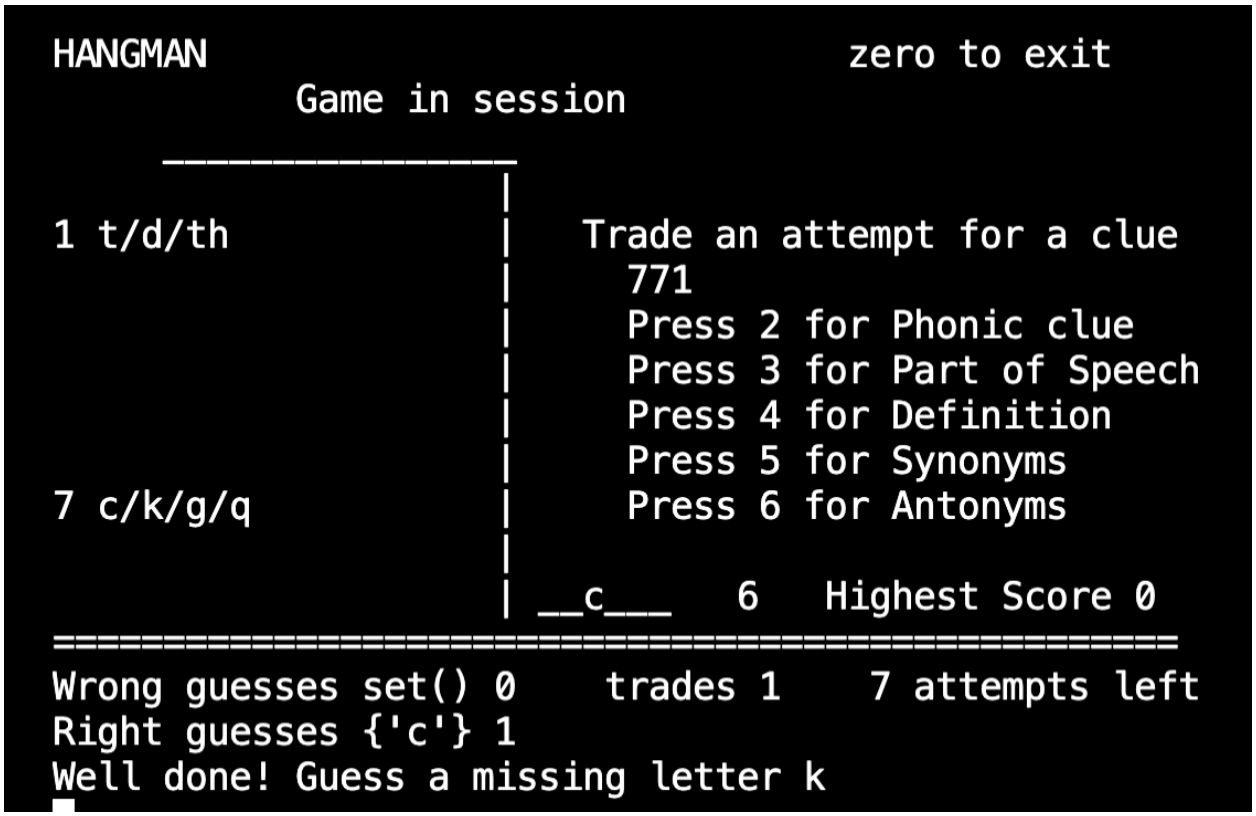
```
CREATE OR REPLACE FUNCTION edit_hash(search_item text, replace_with text)
returns void
language plpgsql
as
$$
declare
	like_item text;
BEGIN
	like_item = concat('%', search_item, '%');
	UPDATE catalog_wordstructure
	SET hash = REPLACE(hash, search_item, replace_with)
	WHERE hash LIKE like_item;
END;
$$;

select edit_hash('/a/', '');
select edit_hash('/aa/', '');
select edit_hash('/ai/', '');
select edit_hash('/air/', '4');
select edit_hash('/ar/', '4');
select edit_hash('/au/', '');
select edit_hash('/aw/', '');
select edit_hash('/awl/', '5');
select edit_hash('/b/', '9');
select edit_hash('/d/', '1');
select edit_hash('/dh/', '1');
```

Thus, for those conversant or attempting to learn this type of mnemonics, the hangman game is useful. In order to facilitate this, the JSON element has been expanded to the following:


```
{
        "id": 319,
        "word": "came",
        "phonics": "/k//ai//m/",
        "links": "0,1,2,1",
        "irregulars": "",
        "hash": "73"
}
```


The hash is based on consonant sounds. The gutteral sounds of “k” and “g” are hashed to “7”. The “m” sound is hashed to “3”.



In a hangman game linked to this database the mnemonic hash can be used as a clue. As you can see phonics, synonyms etc can be used as clues too. These were drawn from an external api (https://api.dictionaryapi.dev/api/v2/entries/en/)

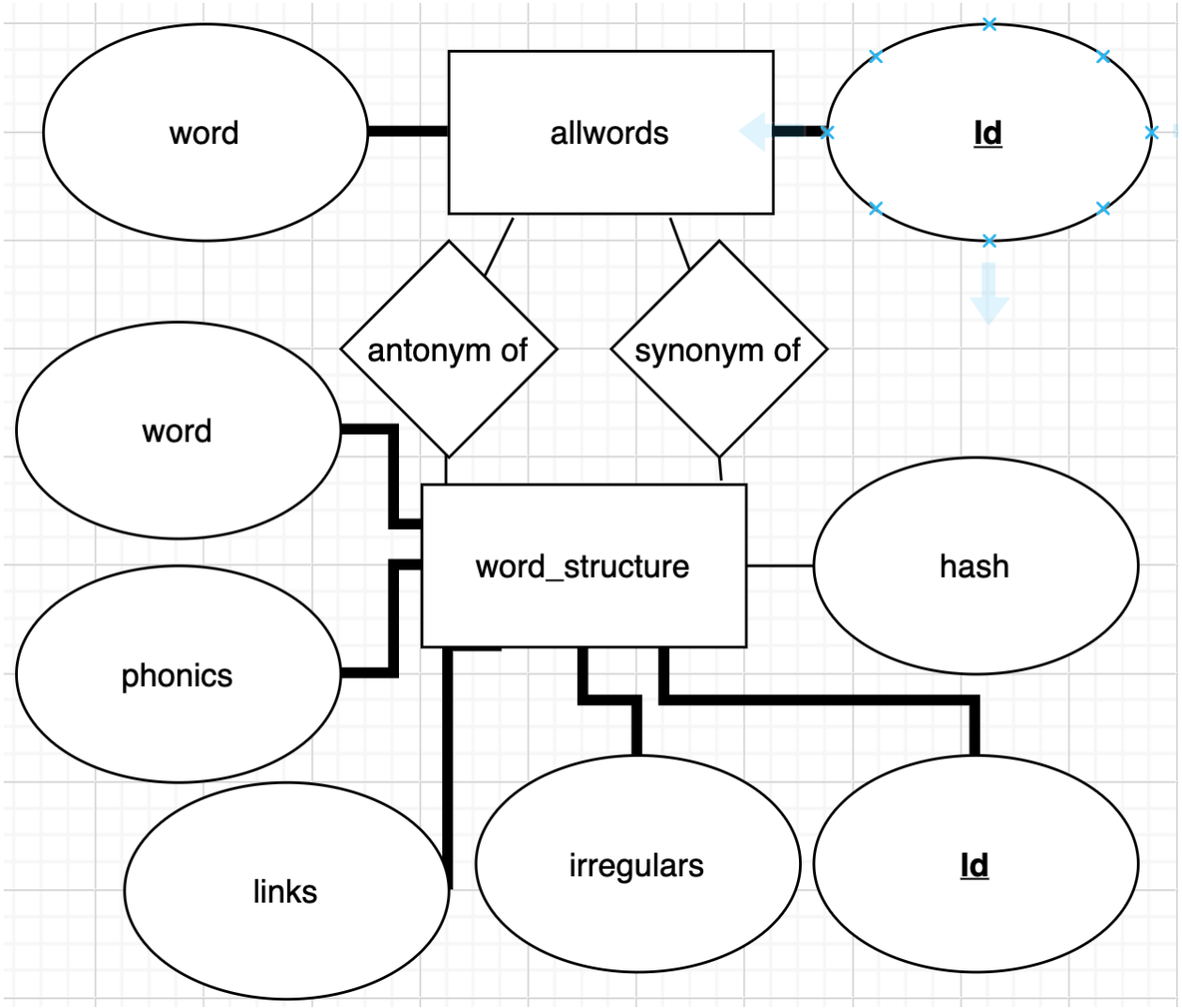
Reading companion and Mnemonics generator.

Imagine having a reading tutor by your side while you read. I think with over 1700 word structures to train a machine learning algorithm, it should be possible to build an app that can recognise the phonics setup of any word and the mnemonic hash.

Project Features

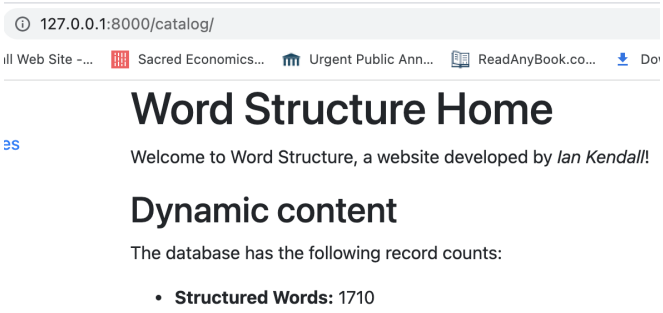
Database Structure

The basic structure of the data is as follows:



On the local docker-compose postgreSQL database I have 1710 words in the word structure table.

Basic Website



You can observe the public website at
http://ec2-3-14-99-127.us-east-2.compute.amazonaws.com:8000/catalog/

However, I do not have the full 1710 on it for now.

http://ec2-3-14-99-127.us-east-2.compute.amazonaws.com:8000/catalog/wordstructures/
Shows all the word structures that are up there as a web page.

They can be queried by Id number are the following location:
http://ec2-3-14-99-127.us-east-2.compute.amazonaws.com:8000/catalog/wordstructure/6

API functionality

The same data can be viewed through the api with some added functionality.

http://ec2-3-14-99-127.us-east-2.compute.amazonaws.com:8000/catalog/worddata/
Offers GET and POST functionality

http://ec2-3-14-99-127.us-east-2.compute.amazonaws.com:8000/catalog/worddata/6
Id search offers GET, PUT and DELETE

http://ec2-3-14-99-127.us-east-2.compute.amazonaws.com:8000/catalog/worddata/as
Word search only offers GET.

Challenges

The course was mostly about deploying existing code so the code to create the app had to be pieced together from multiple sources. I must confess that my first run through the exercises and workshops was somewhat superficial so I did not recognize the full value of the material until I had to put this project together.

Resources

Because I wanted access to the database, both to observe the result of my data modeling and to insert the necessary data, I repeated the procedure from workshop 1 up to the point of implementing some database functionality and security encryption.
https://learn.nucamp.co/mod/assign/view.php?id=4201

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Models
Explains how to build a library, so I modified some code from this to build the models.

At every step I backed up to https://github.com/IanKendall/project2

This allowed me to reverse direction more than once.

docker compose down --rmi all
docker compose up -d
docker compose exec web python manage.py makemigrations --noinput

```
Ians-MBP:project2 iankendall$ docker compose exec web python manage.py makemigrations --noinputMigrations for 'catalog':
  catalog/migrations/0001_initial.py
    - Create model AllWord
    - Create model WordStructure
    - Create model Synonym
    - Create model Antonym
```

docker compose exec web python manage.py migrate --noinput

```
Ians-MBP:project2 iankendall$ docker compose exec web python manage.py migrate --noinput
Operations to perform:
  Apply all migrations: admin, auth, catalog, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying catalog.0001_initial... OK
  Applying sessions.0001_initial... OK
```

docker compose exec web python manage.py createsuperuser

```
Ians-MBP:project2 iankendall$ docker compose exec web python manage.py createsuperuser
Username (leave blank to use 'root'):
Email address: iankendall0@gmail.com
Password:
Password (again):
Superuser created successfully.
```

I used the docker compose commands shown as a sort of compile button every time I made a minor change to the models.py

Then I switched to udemy course for a backend api and django.

This finally allows me to decode the contents of the url.py and view.py files copied from tutorials.zip and turn them to my own purposes.

```
urlpatterns = [
   path('', catalog_views.index, name='index'),
   path('wordstructures/', catalog_views.WordStructureView.as_view(),
name='wordstructures'),
   path('wordstructure/<int:pk>', catalog_views.WordStructureDetailView.as_view(),
name='wordstructure-detail'),
   path('hello-view/', catalog_views.HelloApiView),
   path('worddata/', catalog_views.worddata_list),
   path('worddata/<int:pk>', catalog_views.worddata_detail),
   path('worddata/<str:wd>', catalog_views.worddata_search_word),
```

I noticed that websites have .as_view() on the end. APIs do not. The views reference methods in views.py not the classes.

Once the entire website was working locally, I made use of the week2 exercise: https://learn.nucamp.co/mod/book/view.php?id=5147&chapterid=5634

This provided the instructions necessary to get an AWS version up and running from the latest github version.

Updating from the repository was very similar to updating locally

```
docker-compose down --rmi all

docker-compose build

docker-compose push

docker-compose up -d
```

```
docker-compose exec web python manage.py makemigrations --noinput

docker-compose exec web python manage.py migrate --noinput

docker-compose exec web python manage.py createsuperuser
```

To provide some contents for the public version I pulled from the local website and pushed to the aws website. I would like to find a more efficient way of doing this.

Forward direction

The next challenge is to expand on this work.

Expanding the backend itself

How do I make the backend better?

Making the database bigger

The database currently has 1710 entries. The backend obviously needs more data. I can use the data I have to train a machine learning algorithm to produce new entries.

Extra database features

Synonyms, Antonyms, Definitions etc. These database tables are already present but as yet there is no data in them. I also need to build views that could add data to them.

Creating apps to use the database.

I already have the basis for 2 apps. It would probably make sense to bring out versions of the existing front ends for websites and devices.

React and React Native

These seem good candidates for developing web sites and device apps. The initial code would be reusable but could be adapted to android or apple templates.

License the backend to other apps

My imagination cannot encompass all possible applications so I should figure out how to license out the backend to whoever wants to build more.

Conclusion

Now that I have the basics of DevOps I can use these skills to develop this and other applications. I think it was worth the effort. I learned a lot.