University of Pécs

Faculty of Sciences

Institute of Mathematics and Informatics

Word-Press Web Application Data Flow improved by Core Python libraries

Author:
Shang Rongxiang
Computer Science BSc

Internal supervisor:
Dr. Bornemisza Imre
Scientific Advisor

External supervisor:
Professor Jenak Ildiko
Assistant Supervisor

Pécs, 2022

Contents

Chapter I

Abstract

The difficulty of delegating and integrating them into most modern web applications currently limits reactive Data. The method of actively updating a WordPress web application through many ways provides a model for a much more finite method which we could use to form a more general, professional use procedure. First, the data in subject would have to be parsed and partitioned into manageable clusters. It would also create a migration pipeline[1] with constraints to provide an efficient pathway for our cluster model. The model may be looped if certain conditions have been met. Such conditions are inferred by observing the frequency at which we update our WordPress web application with our cluster model. I drew conclusions and comparisons up with respect to the traditional methodologies used in data migration on a WordPress platform in real world usage. There may be some limitations of the data pipeline, which will be discussed in later chapters.

[1]Data migration pipeline enhances the process of automatically moving sample production data into the Business Operations Environment (BOE).

Chapter II

Introduction

1 Topic description

Once in production or in a deployment format, posting data sources has slowly become a vital aspect in the life-cycle of a WordPress[1] application. As data pipelines became a core and integral part in the life-cycle of web applications[2], the data sources became mutable and much more complex which relied on versioning control and parsing. This concept has made developers incorporate manual and time-consuming methods in making sure their web-apps are reactive. This stems heavily particularly in web-apps that depend on continuous data delivery as opposed to single web applications that operate on static, immutable data sets. "Hundreds of members of the WordPress community help in the development phases, even though they aren't developers or programmers"[3]. It would eliminate diving into source code web-apps, however a much more streamlined method would need automated and scripting viewpoints to update the data sources. Universality of Python within large-scale enterprises, there is a targeted demographic for a much more robust and simpler data migration application. A major example would be a reactive web application which changes with respect to it's data source whilst adapting to user data and tracking systematic changes. This requires constant data fetching on both the mobile and web front. Python was initially created in late 1980s in the Netherlands which was the hope to take over ABC programming language. It was created with goals of being able to handle exception handling and communicating with Amoeba OS. Over time, newer iterations were released including Python 2.0 and Python 3.0 with comprehension of list data structure, reference counting, mutable data types. Python 2 was discontinued with version 2.7.18 in 2020[4]

[1]WordPress is a free and open-source content management system written in PHP and paired with a MySQL or MariaDB database. Features include a plugin architecture and a template system, referred to within WordPress as Themes. [1]

With the rapid growth of Python language along with it's use within the industry, non traditional use-cases that would typically not be used were being adopted. Automate the Boring Stuff with Python [5] was one of the pathways that lead to an exponential growth in Python adaptation. Data pipelines not only benefit developers but also low-code developers and non-developers accumulate large amounts of data, structure it, create back-up copies and migrate it into other extendable networks and endpoints. This thesis aims to create a data migration pipeline specifically aimed at WordPress web applications. It would specifically create a faster, modular system used for migrating data to and from an already created WordPress web app. In the event that a WordPress app[6] benefits from public data sources found on the web, data migration pipeline would offer a web crawling module that would get a specific requested data from the internet network. This is further discussed in detail in later chapters. Automating this procedure while giving the right amount of freedom and flexibility to potential users is also a key goal. Being built from the ground up to be data-centric, the thesis project is data-driven. It acts upon data clusters that can be scaled horizontally and vertically depending on the use case. The data clusters being a sub-group of our data structures would have similarities with the main group of data. This improves further parsing of data trees in certain cases. There are some limitations which are highlighted through out the thesis. Majority of the limitations are language specific due Python being an interpreted language. Irrespective of the limitations we may or may now face, the proof of concept being a practical working model should be on key goal at the end of the project. It is a symbiotic relationship, due the working model having to satisfy certain conditions and factors in order for it to be considered practical, hence a practical working model.

2 Related Work

Over the years, there has been notable similar and/or related work within the concept of Data flow pipelines. A core example, WP RSS Aggregator plugin in combination with the Feed to Post add-on specifically provides a pipeline to cross-post a RSS or Atom feed from another website. Although this majorly depends on already sourced Data sources on published websites while sacrificing originality.

3 Limitations

The purpose of this section is to highlight the limitations that come with the production of a reactive[7] data migration pipeline using Python as the core programming language. Along with the restriction that WordPress posses as opposed to a traditional web application.

3.1 Multithreaded-Based High Memory Consumption

Python offers multithreading. Although this comes at a risk of higher memory consummption. A large and long-running systems would pose difficulty in managing memory. We could always remove threads and run our code-base on a single-thread:

```python
wp = Client('http://pecstalk.wordpress.com/xmlrpc.php',
            'rongxiangshang', '107srxSRX')
post = WordPressPost()
post.title = 'New post'
post.content = 'Post Content'
post.post_status = 'publish'
post.terms_names = {
    'post_tag': ['test', 'firstpost'],
    'category': ['Introductions', 'Tests']}
post.custom_fields = []
post.custom_fields.append({
    'key': 'price',
    'value': 3})
post.custom_fields.append({
    'key': 'ok',
    'value': 'value'})
post.id = wp.call(posts.NewPost(post))
```

Code Listing II.1: Code snippet in Python

3.2 Slow At Runtime

Being further away from the hardware level[8] due to being a high-level programming language, Python code execution occurs with the help of an interpreter instead of a compiler. The interpreter in question will execute the code-base line by line – leading to a slow run-time. This is especially important as it would derail the pipeline from being run on a cloud platform. Meaning it would have to be used locally. A number of Python implementations were compared on non-numerical computations during the EuroSciPy '13. Python's performance when compared to other languages are benched-marked at the Computer Language Benchmarks Game [9] and it falls somewhat short in terms of CPU run-time seconds.
A few note-worthy limitations are:

- WordPress API not allow creation of custom Post Types. A version of this stating "Just like the default WordPress post types, you can create your own CPTs to manage any data you need, depending on what you are building."[10] has already been published.

- API has some under the hood limitations. The wrapper being used is not a 1:1 counterpart. It's more of a 1:2 where 1 is the official API and 2 being the python wrapper API.

- The data clusters cannot be analysed due to the constraints on time complexity

This will host future limitations. There are a few other limitations that can be touched. Some rather trivia limitations would be accessibility features, technical skills for non-developers, back-ward compatibility since lots of Mac-books are using older versions for their test-cli build tools. The interpreter is usually bundled together with a Python version. Certain styles and flavours of coding simply did not exist back then with older versions of Python. An example would be list comprehension. This didn't exist in Python 2.0 but was released in Python 3.0. If a computing machine used a legacy interpreter to run the data migration application, there is a probability that it could hit an error.

Chapter III

Technologies used

1 Tools

The resulting project is made up of several core components that integrate to form our application (Discussed in Chapter II). This ranges from Core Libraries to the Programming Language used. I also dives into a flow cycle flowchart and algorithms. This chapter goes deep into the technology stack used during the thesis. A technology stack is literally a stack of corresponding pieces of software, that are stacked upon one and other in an order of magnitude ranging from build tools, API Integration, Tree parsers, Application data format right down to the programming language used. Choosing the right technology data stack can be very useful as it would not only foster in future-proofing, but also backward compatibility. This is very useful in terms of if the piece of software is built for use for a period of time or for periods in time. A tech stack is also built in accordance to the needs of the project. If there is greater need for speed over efficiency, C# would be the go to since it works close to the hardware of a computing machine. Although with efficiency and ease of use in mind, Python or Go may be better options. This does not only apply to the programming language. XML and JSON are two of the most popular markup data interchange formats available. Picking one over the other isn't a black and white line that be marked up and drawn. XML provides more than just data interchange. It gives a better and more modest approach in the data integrity of XML files. JSON on the other hand is the format for most API's and generally majority of the Internet's data interchange format.

1.1 Technology Stack

This outlines the Technology Stack commonly know as tech stack that were used to contextualise and implement the Data pipeline and wrapping the scripting application. Using a pretty data-driven stack, for the application side, services, networking, database and backup format. The Operating System is not specific to one OS. Rather it can run multiple operating systems – Mac OS, Linux OS and Windows OS. The networking section of the thesis's technology stack is vital in dealing with networking issues, setting up secured sessions between the end user and their WordPress application. It also configs and encapsulates the **Client** object[11]. The stack section that integrates some of the backup infrastructure is using a quick retrieval database like a .CSV file or a relational database such as MSSQL. This will enable end-users to interact with all the program interfaces. It is the bread and butter of the project. Also referred to as a UI project be it a GUI or script. The corresponding stack module(s) should be able to work with mutable data. The table below outlines all section of the technology stack used. Not in any real order, this technologies make up the best stack within the thesis work and factoring limitations discussed in Chapter I.

Technology	Use-case
Bash/CMD	Scripting
Python	Core Programming Language
Python-WordPress-Xmlrpc	Library To Interface With WordPress XML-RPC API
SQL Database	Local Database Storage
PyCharm	IDE Dev Environment
WordPress v5.9.3	Web Application Interface
JSON	Non-Structured Data Format
Domain & Bluehost	Domain Name And Hosting Service
BeautifulSoup	Web Crawler Core Library
SQL	Database Backup
XPath	XML Query

Table 1: Technology Stack

- **Python**: Complex systems can be bundled together into what is called a script. Scripts are majorly used to carry out specialised task. Most scripts are run once, but can be expanded and scaled whilst being run multiple times on several threads. Python is the perfect programming language for this set task in mind. It offers vesartility when needed, open source, it's portability is a plus and it's systematic nature of being Dynamical-typed.

- **Domain Name**: "When you decide to put a website online, you will need a domain name. The domain name is the "address" that identifies your site. For example,the domain name of my main site is ezseonews.com. If you type that into a web browser, you will be taken to my site"[12]

- **JSON** :"It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition"[13] Otherwise know as JavaScript Object Notation is data interchange. It does have an added benefit of being readable to individuals without any form of special adaptability. It was mainly a core aspect of the stack because of it's reinforcement in future-proofing the code-base. It does lack a few features like data-integrity assessment that XML has, the lightweight features it does have are more than enough. JSON does have lesser tendency to fail when caring data packets arcoss our application system.

- **Bash/CMD**: Due to the scripting nature of Python and being an interpreted language, it would have to be passed through an interpreter first, line by line. A Unix shell like Bash for Linus & Mac Operating systems or the standardized command-line interpreter CMD for Windows and other enterprise operating systems. There doesn't have to be a .json or config file that controls how the script should be executed. Instead, python files run from the command line as either scripts of modules as highlighted in Chapter III, section 2, subsection 2.1. There are some advantages of using an interpreter bar the limitations. Errors, both syntax and logical can be accurately thrown at to an end-user.

- **Python-WordPress-Xmlrpc** : It is possible to work with the PHP/Native form of the WordPress API, using a wrapper would greatly eradicate primitive structures. As of Wordpress 3.5+, XML-RPC API is on by default and cannot be turned off. This helps by removing an edge case."The WordPress REST API provides an interface for applications to interact with your WordPress site by sending and receiving data as JSON (JavaScript Object Notation) objects. It is the foundation of the WordPress Block Editor, and can likewise enable your theme, plugin or custom application to present new, powerful interfaces for managing and publishing your site content."[14]

- **Bundle Package**: In order to make the codebase modular and much easier to be downloaded and shared, packaging the project is a solution for this.

- **BeautifulSoup**: The initial scope for this thesis's proof of principle was majorly on data migration pipelines. Although without valuable data sources, data migration pipelines are rendered useless. A mitigate, is to use an initial data loader that is capable of parsing and retrieving clean data from complicated html responses. This would make responses much more adaptive to the environment.

```python
#!/usr/bin/env python3
from bs4 import BeautifulSoup # BeautifulSoup library
from urllib.request import urlopen # A core library for requests
with urlopen('https://wordcount.com/') as response:
    cleanData = BeautifulSoup(response, 'html.parser')
    for element in cleanData.find_all('element'):
        print(element.get('href', '/'))
```

Code Listing III.1: BeautifulSoup in action. Able to create a parse tree from html responses

It is an integral part of web harvesting, otherwise known as web scraping. The parse tree could get exceptional large with nested trees, but doesn't affect the accuracy or performance of the library.

- **XPath**: XPath meaning XML Path Language. It is an expression language similar to JSON that is meant to handle query runs of XML documents. It can also be used to pick node sets in an XML document.

2 Implementation

A switch case could be used to determine if a backup is needed but at the initial stage, there has to be a credible data source. This could can be a user-derived data source from a .CSV or SQL database. In the event that a user-derived data source is unavailable or not needed, data scraping data sources can be harvested all round the internet provided no external issues occur. This will allow the user in a use-case to web crawl through web shops, news update shops, RSS daily feeds of blog websites, general immigration laws from immigration websites with a parse tree provided. This would data would then be cleaned and parsed. The WordPress Python wrapper otherwise called Python-WordPress-Xmlrpc provides a simple yet powerful method to access the WordPress API using a Python environment. On the stage of securing our data, we enter into the execution phase. A secure session is created over the Python-WordPress-Xmlrpc using credential secrets. Although the session only last during run-time. After every successful run, at run-time a new session is created. This is so as to preserve continuity across the API for a specific session. The following and planned prompt would be for users of the end product to pick a choice of a formatted backup in either .CSV or SQL database. During the creation of the secure session highlighted above, a Client object which contains the necessary headers and OAuth required to perform extended WordPress specified data updates, edit posts, create posts, get posts, get pages and update the WordPress build itself. The programming language Python integrates well with the lightweight JSON object types. This objects can be used across different points within the code-base.

```python
data={
    'name':'picture.jpg',
    'type':'image/jpeg',#mimetype
}
```

Code Listing III.2: Code snippet in Python

There is a step down from a .json format, but it does act like a **JSON** object it it's own right. The variable called *data* is actually a dictionary. The major difference is that a dictionary is a data structure based off Python and on the other hand, JSON is a serialization format.

The main way of retrieving an entry from a document is to use a string unique key to get it's corresponding data that it holds.

```
#input
name = data['name']
-----------------------------------
#output
picture.jpg
```

Code Listing III.3: Code snippet in Python

Going back on track about the implementation on create the Data migration pipelines. There is a complexity in using the WordPress API to interact with the exposed endpoints. In the rare point that a response body is sent back, the object time is drastically different. Although there is a Python RESTful Server bridging PHP. A major concern was a test server in this case or a live server would be required. This further adds a middle-layer into our ever growing simple format. Adding that would go agaainst the purpose of the thesis. It would add anther RESTful API which may lead to CORS error which means Cross-Origin Resource Sharing. It tries to use the same origin policy for request that come from another domain name different from the initial domain name used. CORS is a core implementation that helps prevent CORS error. The error itself is issued when a separate web page or app is requesting data sources from another web app in another domain which may be different from the initial domain used for the initial request made. It is further talked about "Before CORS, developers would need to go to great lengths to access APIs from JavaScript clients in the browser"[15] A major example of how CORS error could occur:

- A web app that uses a WordPress API.
- When trying to access the WordPress API, it poses issues since the endpoints do not extend to python.
- Using a Python RESTful Server bridging PHP would send off a request.
- Although Another request will be sent off to the WordPress API in this case having a different domain name as the initial request using the Server bridging

2.1 Programming Languages

Python Interpreter Since majority of the code-base will be written in Python – there would be an Interpreter involved which is usually determined, but not limited to the versioning of Python used. We would use the currently stable version of Python [Python V 3.0] and it's interpreter.

```
$ python hello.py # Terminal command for running Python scripts
$ python3 -m hello # Terminal command for running Python modules
```

Code Listing III.4: Terminal command for running Python scripts and modules

The main part of this approach is the WordPress API. It utilizes the below mentioned algorithm highlighted in chapter III, section 2, subsection 2.2 in order to create the power the data migration pipelines. It can run independently of other libraries or environment tweaks. The python wrapper for the WordPress API is Python-WordPress-Xmlrpc library and can be used directly within Python scripts. There are a number of classes exposed which acts as an interface to use the data migration pipeline.

- **wordpressClient**: This class (*wordpressClient*) accepts three parameters. The parameters are secrets. With the three parameters, namely username, password, domain. Together it establishes a secure session with the credentials. It returns a Client[16] object. The Client object is a WordPress API custom object, which will be used to interact using it's extended methods.*username* refers to an actual WordPress user's sign-in credential username. "The workhorse for managing WordPress users in code is the WPUser class"[17]. This can be interchanged with Client class. The *password* refers to an actual WordPress user's sign-in credential password. Also, *domain* is needed as a beacon to the full domain name of a WordPress website.

- **chinaUniversities**:*chinaUniversities* takes in a uniform serialized URL string variable and harvest for credible datum. The URL would have to redirect to a website that already has some form of data source. It does not return the data source itself, but as a HTML object. The class does extend with more methods that pinpoint what kind of data sources will be needed. A Wikipedia article about super-symmetry is an ideal target candidate.

- **databaseClient**:*databaseClient* recieves a database object.

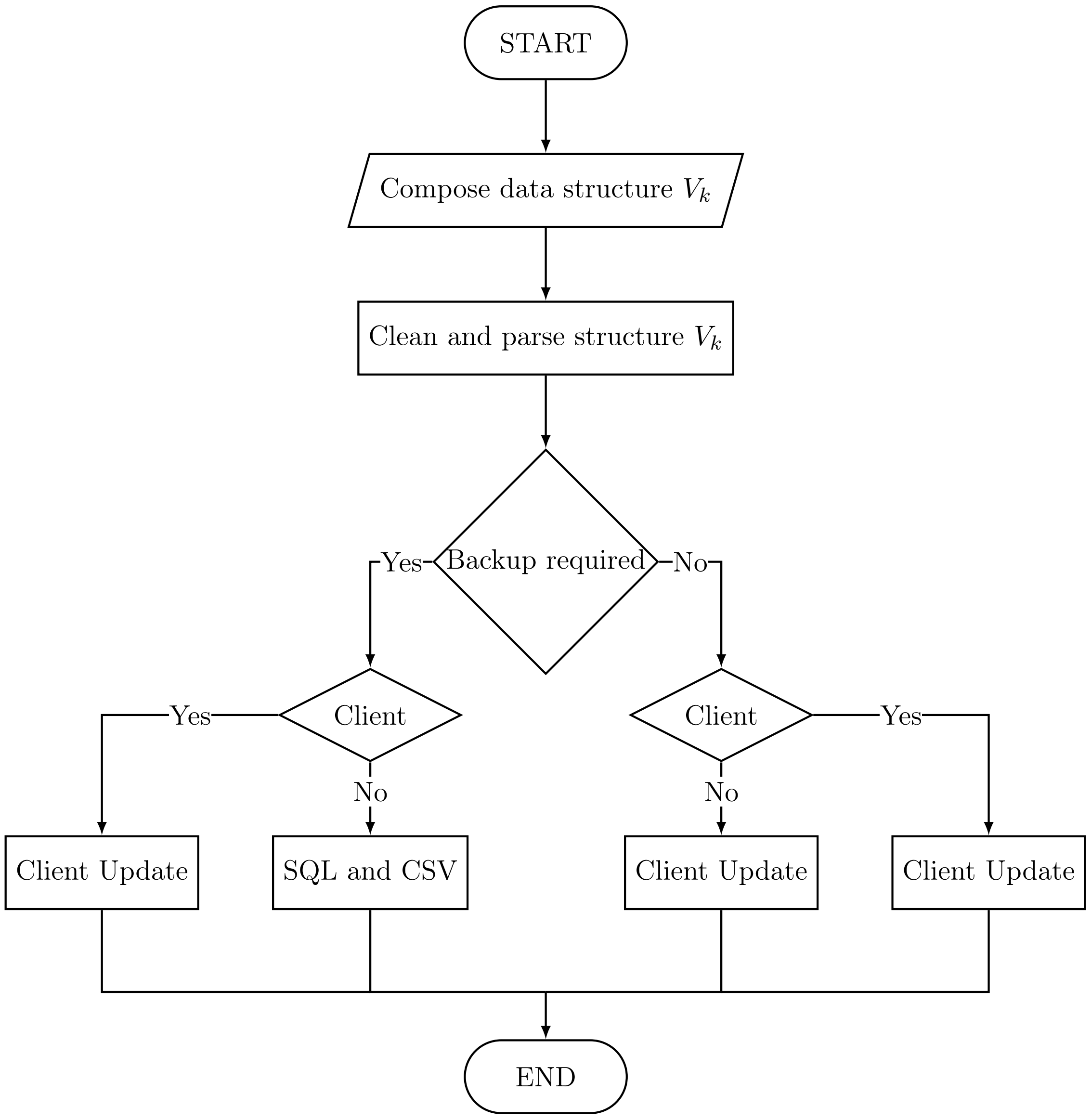
```python
db=pymysql.connect(host='localhost',
                        user=self.user,
                        password=self.,
                        database='TESTDB')
```

Code Listing III.5: Code snippet in Python

It majorly handles the re-formatting and storing of backup parsed data sources during the data migration process. It uses an SQL database as it's base and not a non-SQL database. This is due to how SQL databases scales vertically and not horizontally.

2.2 Algorithm for Data pipeline

This subsection will show how exactly the data migration pipelines are used to update the pages of WordPress web applications. It shows a step by step pseudo code used from the initialization to the finalization of the data migration process. It would also highlight some key methodologies for manipulating the data sources into data clusters. A flowchart is used to display the entire life-cycle of the data migration pipeline. The a cases which branch off into other cases. Each cases has a switch that re-directs the flow cycle to a new direction. Irrespective of the flow used, all flows will inevitably stop at the END node. The initial node START is the beginning of the flow cycle. Flows can be identical, depending on the use-case of the user. There is also a RETRIEVAL & PARSE algorithm discussed below. It features the algorithm used in a node branch of the flowchart.


START
Compose data structure V_k
Clean and parse structure V_k
Backup required
Yes
No
Client
Yes
No
Client
No
Yes
Client Update
SQL and CSV
Client Update
Client Update
END

Algorithm 1 A general data RETRIEVAL&PARSE algorithm

```
Funct Post($WP, P$)
 1: Set the mutable data list $[\mathcal{L}]_{\mathcal{M}} := [P]$ and the final list $[\mathcal{L}]_{\mathcal{F}} := []$
 2: while ( $\mathcal{L}_W \neq \emptyset$ ) do
 3:     Select a time interval $T$ from $_{Datetime}$                    ▷ Selection rule
 4:     Compute $sourcing(X)$                                            ▷ Data sourcing rule
 5:     if $T$ Elapse then                                               ▷ Convergence rule
 6:         for $i = length(X), \ldots, P$ do
 7:             if $X[i]$ satisfies the parsed data criterion then       ▷ Termination rule
 8:                 Store $X[i]$ in $\mathcal{L}_W$
 9:             else
10:                 Store $X[j]$ in $\mathcal{L}_W$
11:             end if
12:         end for
13:     end if
14: end while
15: $\mathcal{L}_Q = \mathcal{L}_W$
16: return $\mathcal{L}_Q$
```

3 Proficiency

My thesis topic clearly based on a Python package, states the data migration flow. The optimal proficiency limit would be to gather data sources, which can be manipulated into a backup database[18], backup .csv file structure and can be used for many data pipeline updates as mentioned in the Introduction.

3.1 Data Formats

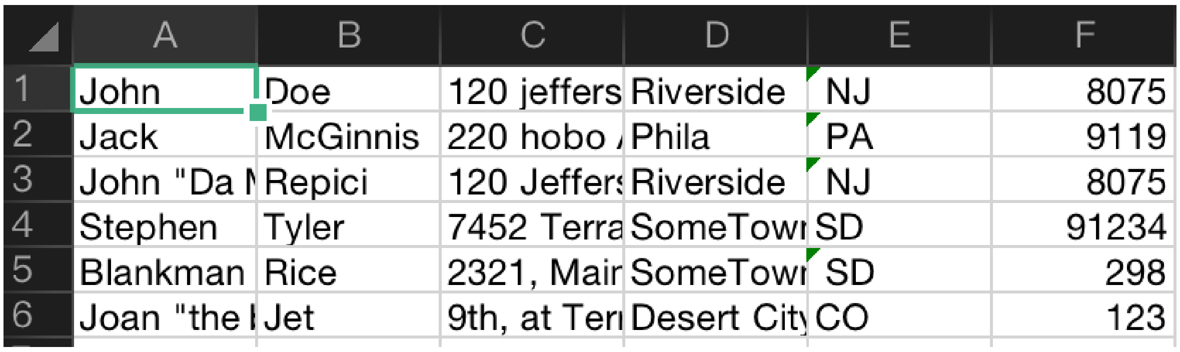
Data migration pipelines heavily depend on the structure and format of the initial data provided. During run-time, the data structure provided is parsed in UTF-8 to avoid reading issues and have one universal encoding format. This also means that data formats are strictly formatted once for each data migration. In the presence of this, multi-threading is a viable solution for handling various data formats at once. A thread is a separate flow of execution. This can allow a user host to run two task at the same time. This will not break the syntax or the semantics[19] of the program as shown in this here. There is a limitation mentioned in chapter II, section 3, subsection 3.1. As the time complexity grows, so those the space complexity. Although for a sizable number of N being the length of elements going through the data migration pipeline.

```python
from time import sleep, perf_counter
from threading import Thread

def task():
    print('Starting a task...')
    sleep(1)
    print('done')

start_time = perf_counter()

# create two new threads
t1 = Thread(target=task)
t2 = Thread(target=task)

# start the threads
t1.start()
t2.start()

# wait for the threads to complete
t1.join()
t2.join()

end_time = perf_counter()

print(f'It took {end_time- start_time: 0.2f} second(s) to complete.')
```

Code Listing III.6: Code snippet in Python

3.2 Data Backup Formats

Somewhere along the data migration pipelines, the approach is capable of packaging and exporting the data structures into an external database or a comma-separated values file. The initial data source will be parsed and reformatted as a post object for WordPress web application. The data sources can be parsed and serialised into JavaScript Object Notation which will enable advanced manipulation within database environments. This will enable multiple pathways for the data source formats. The comma-separated values file can be represented graphically using Google Sheets, Word and WPS Office.
Once the data sources have been parsed, they can be backed-up into a CSV format.

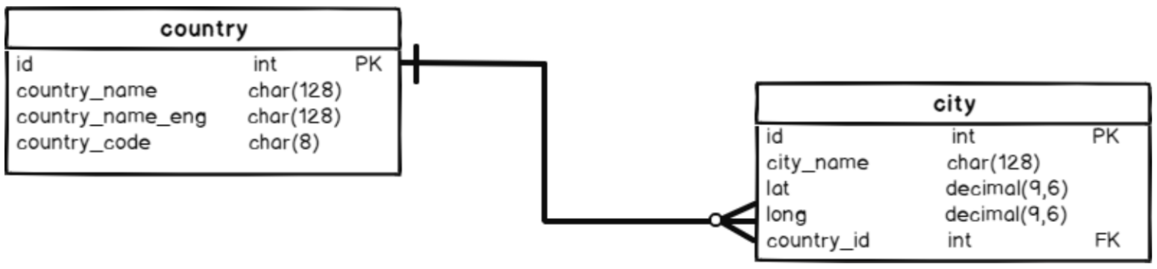


	A	B	C	D	E	F
1	John	Doe	120 jeffers	Riverside	NJ	8075
2	Jack	McGinnis	220 hobo ,	Phila	PA	9119
3	John "Da M	Repici	120 Jeffers	Riverside	NJ	8075
4	Stephen	Tyler	7452 Terra	SomeTowr	SD	91234
5	Blankman	Rice	2321, Mair	SomeTowr	SD	298
6	Joan "the l	Jet	9th, at Teri	Desert City	CO	123

Source: https://people.sc.fsu.edu/ jburkardt/data/csv/csv.html

Figure 1: CSV structure for data backup

Conventionally, it can be stored in a relational database.



country		
id	int	PK
country_name	char(128)	
country_name_eng	char(128)	
country_code	char(8)	

city		
id	int	PK
city_name	char(128)	
lat	decimal(9,6)	
long	decimal(9,6)	
country_id	int	FK

Source: https://people.sc.fsu.edu/ jburkardt/data/csv/csv.html/

Figure 2: Sample example of relational database for data backups

Chapter IV

Discussion of results

1 Comparisons

Once the data pipeline has been initialized, drawing up comparisons with the native methods provided by WordPress can easily be made. The comparisons would be on time complexity. Both methods at their roots both execute some form of code-base. Although, there is the human factor involved. This factor will not go unnoticed as it is also a decided and not a negligible factor. The proof of principle for this thesis has been met in the sense of developer satisfaction. To draw out a major difference between each method, thee time complexity can be measured. Time complexity can be defined as the number of times a code block or code cluster is run for a data set. The data set in this case would be our JSON data object(s). The code cluster would be the a start cycle of each methods cycle. It should be noted that time complexity should not be seen as a black and white bar. It is only one piece of the puzzle. The space complexity can also be a metric used in comparisons of both methods. Space complexity is the metric of hardware memory used by a code cluster or algorithm.

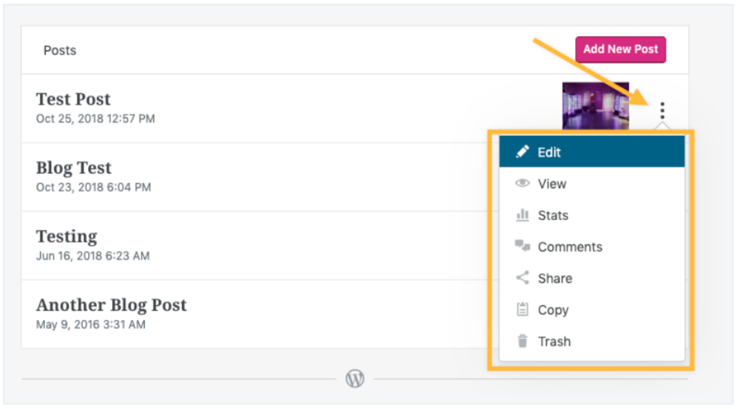
1.1 Using the native methods[20]

In this thesis, as proof of principle, there are objectively two methods that can be used to provide data updates. Native Dashboard – refers to the readily provided methodology to add posts, delete pages, add data contents to the hierarchy of WordPress web applications.
The latter, Data Migration Pipeline is the fundamental project of this thesis. The two methods compared in terms of time complexity or big O notation:

Methodology	*Time complexity.*
Native Dashboard	O(n2) – Quadratic Time.
Data Migration Pipeline	O(n) – Linear Time.

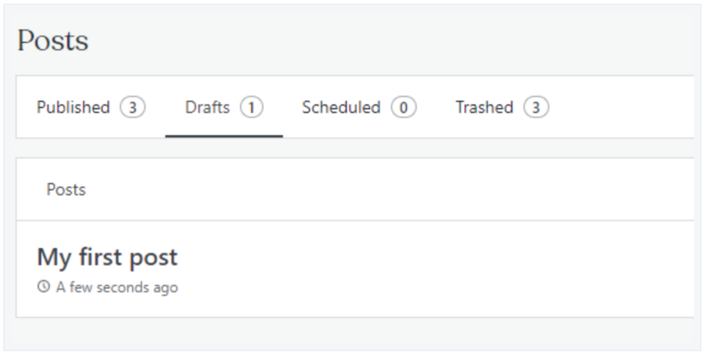
Table 2: Observations and Comparisons

This implementation of comparison is from an optimal perspective. Looking at the overall complexity of each method can give another perspective.



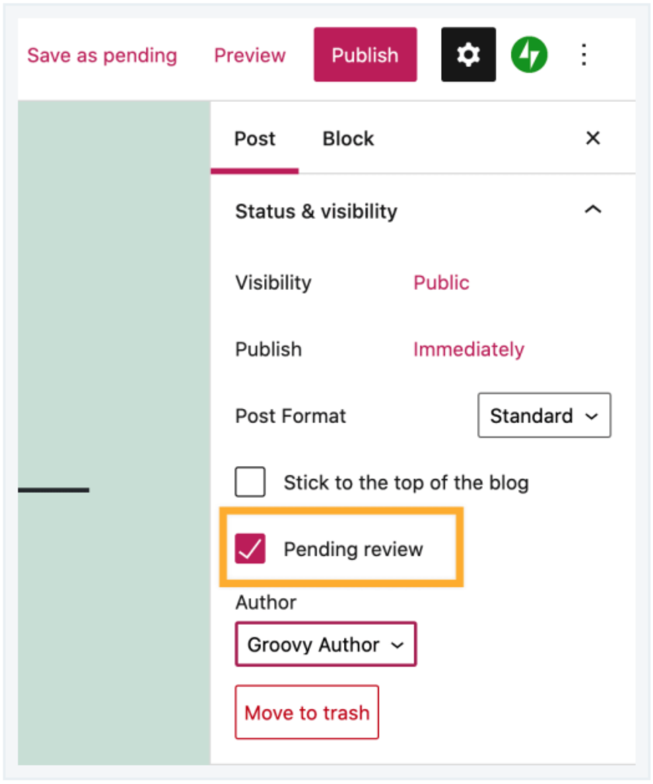
Source: https://en-support.files.wordpress.com/2019/05/ellipses-menu-posts-wordpress.png

Figure 3: Viewing WordPress posts from the dashboard



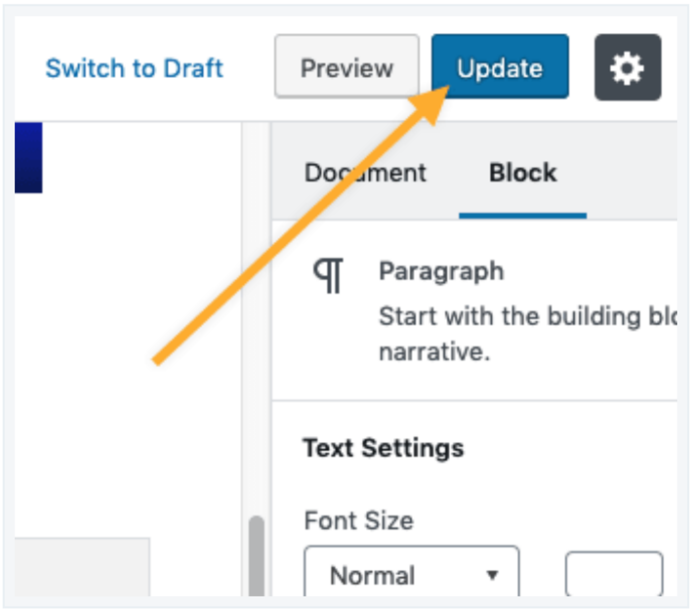
Source: https://en-support.files.wordpress.com/2021/02/post-drafts.png

Figure 4: Screen showing post drafts



Source: https://en-support.files.wordpress.com/2021/02/posts-pending-review-1.png?w=496

Figure 5: Post drafts under review



Source: https://en-support.files.wordpress.com/2019/05/posts-update-post.png

Figure 6: Edit and update screen

The optimal target in this case would be in updating an already existing Post with the simple JSON data:

```
{
   "data":{
      "data_content": {
            "post_content":"Some form of content",
            "post_content":"Exponential",
            "post_content":"Working with linear algebra",
            "post_content":"CS",
            "post_content":"Gross",
            "post_content":"New console release",
            "post_content":"Inflammation",
                }
      "data_ID": {
            "post_id":"101",
            "post_id":"102",
            "post_id":"103",
            "post_id":"104",
            "post_id":"105",
            "post_id":"106",
            "post_id":"a_unique_id",
            }
   }
}
```

Code Listing IV.1: JSON tree data

2 Future-Proofing

In order to optimise this application so as reduce any form of drastic changes in the foreseeable future, a predictive model would have too be implemented. The results at a certain point in time could most definitely change over the course of time. Code-bases should be able to run as new hardware and technological advancements are made. This includes backward compatibility, the ability to add and remove new features without breaking the entire build of the code-base. It's greatly crucial in contributing to a readable and robust code-base. A major reason why JSON is the preferred data object used all across the thesis project. Irrespective of how much cutting edge technology improves, they will be able to interact on software level with JavaScript Object Notation. This ensures that the code-base will not be termed as " legacy code ". Legacy code has strong ties to old code. Not directly in age of the code, but on how obsolete the code is in respect to current technological terms and it's ability to interact with other technology.

Chapter V

Conclusions

In order to demonstrate a proof-of-principle for my thesis topic, a live execution needed to be performed. The limitations and proficiency in utilizing a data migration pipeline for dynamic data structures. The proficiency highlighted in chapter III, section 3 ranging from manipulating data sources, re-formatting data sources and passing it through data pipelines. On the other hand, the limitations highlighted in chapter II, section 3 ranging from Mulithreaded-base memory consumption, stem from the high-level of Python and how far it is from hardware level compared to C++ or C. There were also various technical limitations that WordPress as a technological entity entailed. One major area of improvement in the future, would be the adaptation of updating pages at a system level instead of individual data clusters. WordPress does not have a solid pipeline, instead it performs a Query to fetch posts and get context. A hard coded template file is issued based on the context. The only mutable data are variables that accept data changes. This limits the way WordPress pages can be updated. This may need a whole revamp of the WordPress standardization. WordPress uses XML-RPC in order to interface, more specifically the WordPress API. The wordpress-xmlrpc library which is the python implementation of the WordPress API. It is not a 1:1 comparison though, since it aims at easy integration into the ecosystem. Some approaches might not have a counterpart when using wordpress-xmlrpc. In some rare cases, said functions may be needed to be accessed. The python counterpart could benefit from a 1:1 revamp. This will allow for other components to be included.
In conclusion, the final outcome can be inferred from the proof of principle. The initial goal as discussed in chapter I was met to a certain degree. Further research and work would have to be carried out in other to have full flexibility of WordPress web applications – but then again, that is a limitation of the WordPress.

Chapter VI

References

[1] Adakiko. *WordPressWordPress (WP, WordPress.org)*. 2001. URL: https://en.wikipedia.org/wiki/WordPress (visited on 03/12/2022).

[2] B. Messenlehner and J. Coleman. *Building Web Apps with WordPress: WordPress as an Application Framework*. O'Reilly Media, 2014, p. 2. ISBN: 9781449364793. URL: https://books.google.hu/books?id=QUROAwAAQBAJ.

[3] L. Sabin-Wilson et al. *WordPress All-in-One For Dummies*. –For dummies. Wiley, 2011, p. 39. ISBN: 9781118048672. URL: https://books.google.hu/books?id=pvo6GNXy47cC.

[4] Benjamin Peterson. *Python InsiderPython Insider: Python 2.7.18, the last release of Python 2*. 2020. URL: https://pythoninsider.blogspot.com/2020/04/python-2718-last-release-of-python-2.html (visited on 01/10/2022).

[5] E. Dijkstra. "Classics in Software Engineering". In: ed. by Edward Nash Yourdon. Upper Saddle River, NJ, USA: Yourdon Press, 1979. Chap. Go to Statement Considered Harmful, pp. 27–33. ISBN: 0-917072-14-6. URL: http://dl.acm.org/citation.cfm?id=1241515.1241518.

[6] B. Messenlehner and J. Coleman. *Building Web Apps with WordPress: WordPress as an Application Framework*. O'Reilly Media, 2014, p. 11. ISBN: 9781449364793. URL: https://books.google.hu/books?id=QUROAwAAQBAJ.

[7] B. Messenlehner and J. Coleman. *Building Web Apps with WordPress: WordPress as an Application Framework*. O'Reilly Media, 2014, p. 11. ISBN: 9781449364793. URL: https://books.google.hu/books?id=QUROAwAAQBAJ.

[8] L. Ramalho. *Fluent Python: Clear, Concise, and Effective Programming*. O'Reilly, 2015. ISBN: 9781491946237. URL: https://books.google.hu/books?id=luRHjwEACAAJ.

[9] Doug Bagley. *The Computer Language Benchmarks Game*. 2020. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python.html (visited on 01/21/2022).

[10] B. Messenlehner and J. Coleman. *Building Web Apps with WordPress: WordPress as an Application Framework*. O'Reilly Media, 2014, p. 126. ISBN: 9781449364793. URL: https://books.google.hu/books?id=QUROAwAAQBAJ.

[11] B. Messenlehner and J. Coleman. *Building Web Apps with WordPress: WordPress as an Application Framework*. O'Reilly Media, 2014, pp. 155–158. ISBN: 9781449364793. URL: https://books.google.hu/books?id=QUROAwAAQBAJ.

[12] A.J. Williams. *WordPress for Beginners 2021: A Visual Step-By-Step Guide to Mastering WordPress*. Amazon Digital Services LLC - KDP Print US, 2020, pp. 30–31. ISBN: 9798584887780. URL: https://books.google.hu/books?id=o78fzgEACAAJ.

[13] Glenn E. Krasner and Stephen T. Pope. "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80". In: *J. Object Oriented Program.* 1.3 (Aug. 1988), pp. 26–49. ISSN: 0896-8438. URL: http://dl.acm.org/citation.cfm?id=50757.50759.

[14] WordPress Org. *REST API Handbook*. 2018. URL: https://developer.wordpress.org/rest-api/ (visited on 02/28/2022).

[15] M. Hossain. *CORS in Action: Creating and consuming cross-origin APIs*. Manning, 2014. ISBN: 9781638353256. URL: https://books.google.hu/books?id=uTkzEAAAQBAJ.

[16] B. Messenlehner and J. Coleman. *Building Web Apps with WordPress: WordPress as an Application Framework*. O'Reilly Media, 2014, p. 158. ISBN: 9781449364793. URL: https://books.google.hu/books?id=QUROAwAAQBAJ.

[17] B. Messenlehner and J. Coleman. *Building Web Apps with WordPress: WordPress as an Application Framework*. O'Reilly Media, 2014, p. 156. ISBN: 9781449364793. URL: https://books.google.hu/books?id=QUROAwAAQBAJ.

[18] B. Messenlehner and J. Coleman. *Building Web Apps with WordPress: WordPress as an Application Framework*. O'Reilly Media, 2014, pp. 23–25. ISBN: 9781449364793. URL: https://books.google.hu/books?id=QUROAwAAQBAJ.

[19] Eric Matthes. *Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming*. 2nd. No Starch Press, 2019. ISBN: 9781593279295.

[20] B. Messenlehner and J. Coleman. *Building Web Apps with WordPress: WordPress as an Application Framework*. O'Reilly Media, 2014, p. 172. ISBN: 9781449364793. URL: `https://books.google.hu/books?id=QUROAwAAQBAJ`.

List of Figures

List of Tables

UNIVERSITY OF PECS
Faculty of Sciences
Instituion ID FI58544

Declaration of Originality

Student's Name: Shang Rongxiang

Neptun code: B27SW1

Title of Thesis: WordPress Web Application Data Flow improved by Core Python Libraies

Hereby, I, a student of the Faculty of Sciences at the University of Pecs, in full knowledge of my liability, declare that this thesis is **my own original work.** I have clearly referenced all sources (both from printed and electronic sources) in the text in accordance with international requirements of copyright.

I acknowledge that it is considered plagiarism
- to cite verbatim without using quotation marks and crediting the author
- to paraphrase the source material without citing the source
- to indicate another author's published ideas as my own.

I declare that I have understood the definition of plagiarism and I accept that if my thesis violates copyrights, its assessment shall be fail (1), and the major director of the program shall initiate a disciplinary procedure in front of the Dean according to Article 59 (14) of the Code of Studies and Examinations of the University of Pecs.

Pecs, 29 day, 04 month, 2022 year

[illegible]
signed by student