We will build a simple blog app just to demonstrate the inner workings of `wsgi` in python.

WSGI (Web Server Gateway Interface) is a specification for a simple and universal interface between web servers and Python web applications or frameworks. It allows developers to write web applications that can work with any WSGI-compliant web server, providing flexibility and portability.

> WSGI is specific to Python and is not designed to work with applications or servers written in other languages. Other languages have their own standards or interfaces for handling communication between web servers and applications, such as **Rack** for Ruby or **PSR-7** for PHP.

Some popular wsgi-compliant web servers are gunicorn and daphne.

A wsgi application is simply a Python function (or callable) that accepts 2 arguments:

1. `environ`: A dictionary containing request data (e.g., headers, query strings, path information).
2. `start_response`: A callable that starts the HTTP response by sending the status code and headers.

## A minimal wsgi application

Create a new python file called `wsgi_app.py` and copy and paste the following code into it:

```python
from wsgiref.simple_server import make_server

def application(environ, start_response):
    path = environ['PATH_INFO]
    print("The requested path is ", path)

    status = '200 OK'
    headers = [('Content-Type', 'text/plain')]
    start_response(status, headers)

    return [b"Hello, World!"]

# Running the WSGI app using wsgiref
if __name__ == '__main__':
    # Create a WSGI server
    with make_server('', 8000, application) as server:
        print("Serving on port 8000...")
        server.serve_forever()
```

Run the above file, it should start a web application on port 8000.

Open the browser on http://localhost:8000 to view it. It should display a `Hello, World!` message.

Try going to http://localhost:8000/posts instead. What did you see? Check the path printed on the console of the server, has it changed?

As stated previously, the environ variable contains request information. Some popular request information and their keys in the environ are:

- Request path: `PATH_INFO`
- Request method: `REQUEST_METHOD`
- Query parameters: `QUERY_STRING`
- Content length: `CONTENT_LENGTH`

There are a lot more but just know that any thing that is passed in the request can be accessed from the environ parameter.

# Building out the blog app

You are going to replace the code already in the file created previously in order to create the blog application.

Before we get to coding, here are our objectives:

- Implement **persistent** data storage with a dictionary. This means that even when we stop the application and start it back, we should still have the data previously entered.
- Implement endpoints for listing and adding posts, viewing, updating and deleting a particular post

You are going to build out this application by following the different steps given in this document.

## Initial setup

Here we import the required packages and define some functions and variables that will be used ahead.
Replace the code in `wsgi_app.py` with the following:

- Import `re`, import `parse_qs` from `urllib.parse`, and import `make_server` from `wsgiref.simple_server`
- Create a dictionary called `data_store` with keys `posts` and `counter`. Let the posts be an empty dictionary and the counter be 0
- Define a function `render_template` that returns the following string:

```
f"""
    <!DOCTYPE html>
    <html>
    <head>
        <title>Simple Blog</title>
    </head>
    <body>
        {content}
    </body>
    </html>
    """
```

- Create a variable `new_post_form_html` and assign it the value:

```
"""
<form method='POST' action='/posts'>
    <h3>Add a new post</h3>
    <label>Title: <input type='text' name='title'></label><br>
    <label>Content: <textarea name='content'></textarea></label><br>
    <button type='submit'>Add Post</button>
</form>
"""
```

## Main application

Define a function called `application` that takes 2 arguments, `environ` and `start_response`

- In the function body, get the request method (from the environ variable) and store it in a variable called `method`
- Do same for the request path.
- Print the method and the path variables.
- Copy and paste the following code in the function:

```
    status = '200 OK'
    headers = [('Content-Type', 'text/plain')]
    start_response(status, headers)
    return [b"Hello, World!"]
```

- Copy and paste the following code after and out of the `application` function's body:

```
if __name__ == '__main__':
    print("Serving on http://127.0.0.1:8000")
    server = make_server('127.0.0.1', 8000, application)
    server.serve_forever()
```

Execute your python file. Try navigating to `/posts`, `/posts/1`. Look at the path variable you printed, what do you notice?

## List all posts

In the `application` function, after creating and printing the `method` and `path` do the following:

- Write an `if` statement that checks if the path is `'/posts'` and the method is `'GET'`
- In the if block, we are going to create html content for the posts if there are any else we put a message.
  Write a nested if statement that checks if the length of `data_store['posts'].items()` is greater than 0. In the block of this statement, paste the following:

```
posts_html = "".join([
            f"<p><a href='/posts/{post_id}'>{data['title']}</a>: {data['content']
[:50]}...</p>"
            for post_id, data in data_store.items()
        ])
```

- Write an `else` statement for the if statement **created above**. In the block of the else, put the following: `posts_html = 'There are currently no posts, use the form to add some'`
- In the block of the first if statement written (the one that checks the path and method) put the following:
    - Create a variable, `response_body` and assign it the value `render_template(new_post_form_html + posts_html)`
    - A variable `status` with value `'200 OK'`
    - A variable `headers` with value `[('Content-Type', 'text/html')]`
    - Replace the following code at the end of the function:

```
status = '200 OK'
headers = [('Content-Type', 'text/plain')]
```

```
start_response(status, headers)

return [b"Hello, World!"]
```

```
  With this
```

```
start_response(status, headers)
return [response_body.encode('utf-8')]
```

Now stop the application and run it back.

Navigate to `/posts` to see a form and a message saying there are no posts.

If you fill the form, nothing will be added. This is cause we have not yet worked on the logic for adding items.

## Add a post

Inside the `application` function after the first if block (the one that checks the path and the method), write an else if statement that checks if the path is `'/posts'` and the method is `'POST'`.

- In the `elif` block, write the following block:

```
try:
        pass
except Exception as e:
        response_body = render_template(f"Error: {e}")
        status = '400 Bad Request'
        headers = [('Content-Type', 'text/html')]
```

The rest of the code of this section should be put into the body of the try block.

- Get the content length using the following code: `int(environ.get('CONTENT_LENGTH', 0))`, store it in a variable called `content_length`
- Get the request body using the following code: `body = environ['wsgi.input'].read(content_length).decode('utf-8')`
- Copy and paste the following code:

```
post_data = parse_qs(body)
new_title = post_data.get('title', [''])[0]
```

```
new_content = post_data.get('content', [''])[0]
```

- Write an if block that checks if the `new_title` and the `new_content` are not empty.
  - In this block, create a `new_id` variable and set it to the `data_store['counter'] + 1`
  - Add the `new_id` variable as a key to the `data_store['posts']` dictionary. Let its value be a dictionary with keys and values `'title': new_title` and `'content': new_content` respectively.
  - Set the `data_store['counter']` variable to `new_id`
- Out of the if block, paste the following code, but still in the try block:

```
environ['REQUEST_METHOD'] = 'GET'
return application(environ, start_response)
```

## Display a single post

In the `application` function, after the previous `elif` block. Write another `elif` block that checks if the path has a number and if the method is GET. Here's the code:
`elif re.match(r"^/posts/\\d+$", path) and method == 'GET':`

- In the elif block, get the post id using the following code `int(path.split('/')[-1])`, store it a variable `post_id`
- Check if the post_id is in the `data_store['posts']` with if statement. In the if block, get the post from the dictionary using `data_store['posts'][post_id]` dictionary and store it in a variable called `post`
  - Create a variable `form_html` and assign it the following value:

```
f"""
<form method='POST' action='/posts/{post_id}?' method='PATCH'>
        <h3>Edit Post {post_id}</h3>
        <label>Title: <input type='text' name='title' value='{post['title']}'>
</label><br>
        <label>Content: <textarea name='content'>{post['content']}</textarea>
</label><br>
        <button type='submit'>Update Post</button>
</form>
"""
```

**NB**: We are still in the `if` block that checks if the `post_id` is in the `data_store['posts']`

- Create a variable `post_content` with the following value: `f"<h1>{post['title']}</h1><p>{post['content']}</p>{form_html}"`

- Paste the following code at the end of the if block

```
response_body = render_template(post_content)
status = '200 OK'
headers = [('Content-Type', 'text/html')]
```

- Put an `else` statement after the `if` block created in the 2nd step above. Paste the following code into it:

```
response_body = render_template("Post not found.")
status = '404 Not Found'
headers = [('Content-Type', 'text/html')]
```

## Exercise

Write the endpoints to update and delete a post from the `data_store`. The method for deletion is `PATCH` and that for deletion is `DELETE`.

The `DELETE` endpoint should return us to the list of posts endpoint.

The `PATCH` endpoint should return us to the post detail endpoint