Morph-Day-1

- Morph Setup
- Morph Basics
  - Basics
  - Directory Structure
- Morph Framework
  - Aliases
    - Aliases for Python Functions
    - Referencing Data using Aliases
- Steps to build the Backend with Python
- Add Morph decorators
- FrontEnd-Adding Pages
- Morph Decorators
  - @morph.func
  - @morph.loaddata
    - IMPORTANT - how to get data from SQL
  - @morph.variables

Morph Setup

Run command to install morph

```
pip install morph-data
```

Create new project with morph new

```
morph new morph-starter-app
```

Use the `morph serve` command to start the Morph development server.

```
cd morph-starter-app
morph serve
```

Open the App in the Browser
Access `localhost:8080` to open the app.

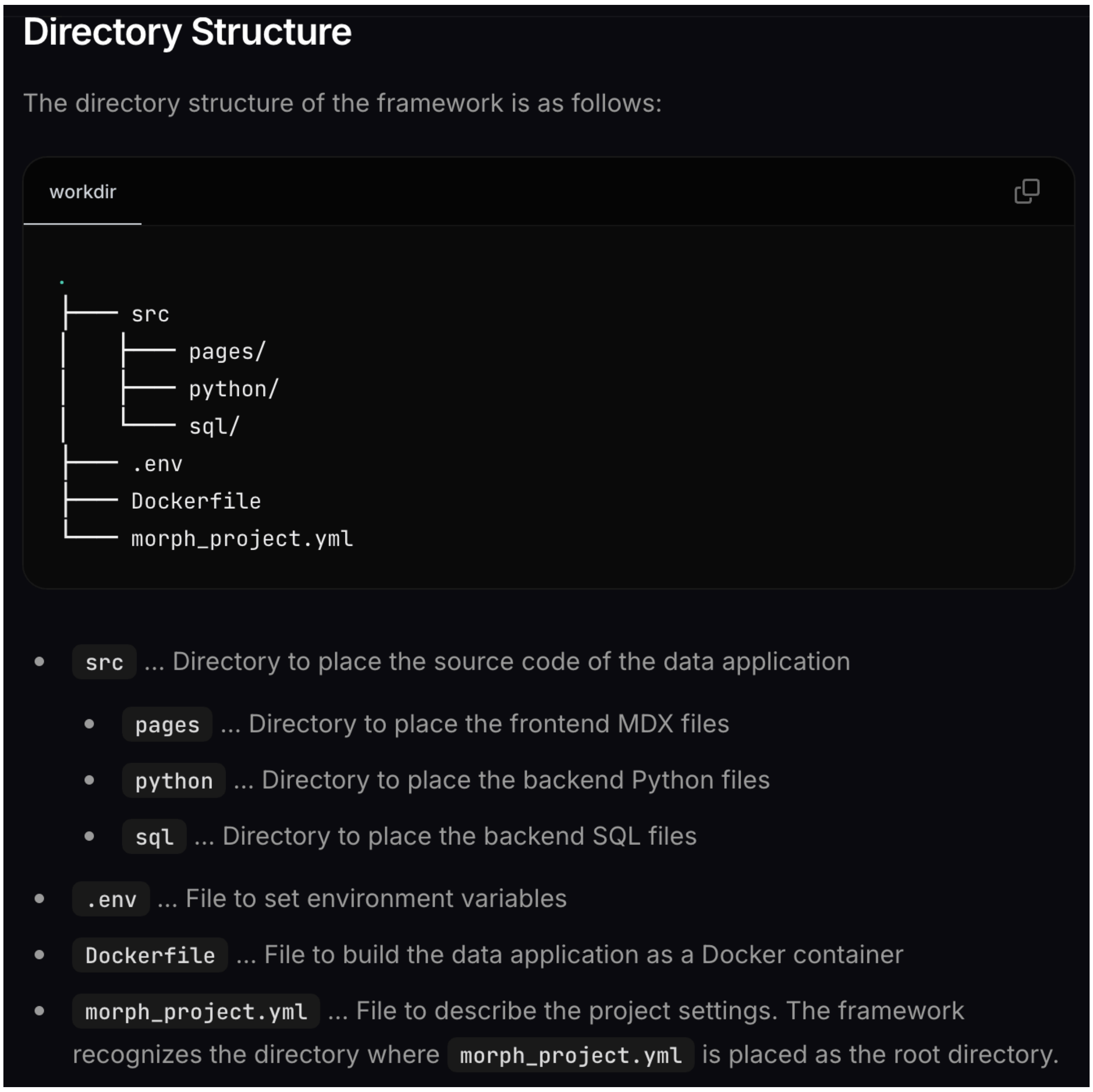
Build the App using SQL, Python, and MDX. Up to this point, you have started the basic development server. Let's build the app using SQL, Python, and MDX!

Morph Basics

Basics

1. We use Python to mess with Data
2. We also use python for the frontend, what's the frontend? It's MD files, Markdown files, but you can add cool visual stuff to it, that's why its MDX files or Markdown eXtended Files.

Directory Structure



Directory Structure

The directory structure of the framework is as follows:

workdir

```
.
├── src
│   ├── pages/
│   ├── python/
│   └── sql/
├── .env
├── Dockerfile
└── morph_project.yml
```

- `src` ... Directory to place the source code of the data application
  - `pages` ... Directory to place the frontend MDX files
  - `python` ... Directory to place the backend Python files
  - `sql` ... Directory to place the backend SQL files
- `.env` ... File to set environment variables
- `Dockerfile` ... File to build the data application as a Docker container
- `morph_project.yml` ... File to describe the project settings. The framework recognizes the directory where `morph_project.yml` is placed as the root directory.

Screenshot 2025-02-18 at 11.32.48 PM.png

Morph Framework

Introduces you to how Morph works, how to use Morph, where to add files and whatnot.

Aliases

Aliases are a mechanism for assigning identifiers to Python functions, SQL queries, etc. throughout the Morph framework so that they can be easily referenced from other files. Using aliases, you can combine functions and queries spread across multiple files, build data pipelines, and reference data from a front-end built with markdown.

Aliases for Python Functions

```
@morph.func(name="example_dataframe") # Set up alias
def main(context):
    #...
    return df
```

1. ! IF no name is specified, then the function name is taken as the alias

```
@morph.func # If no alias is set, the function name will be used as the alias
def example_dataframe(context: MorphGlobalContext):
    #...
    return df
```

Referencing Data using Aliases

To reference data using an alias in a Python function, use the `@morph.load_data("alias")` decorator.

```
import morph
from morph import MorphGlobalContext

@morph.func
@morph.load_data("example_data") # Load data using alias
def example_dataframe(context: MorphGlobalContext):
    df = context.data["example_data"]
    return df
```

Loading the same data in the MARKDOWN File (frontend)

1. ! When using data-referencing components such as `<DataTable>` or `<Embed>`, specify the alias to reference the data.

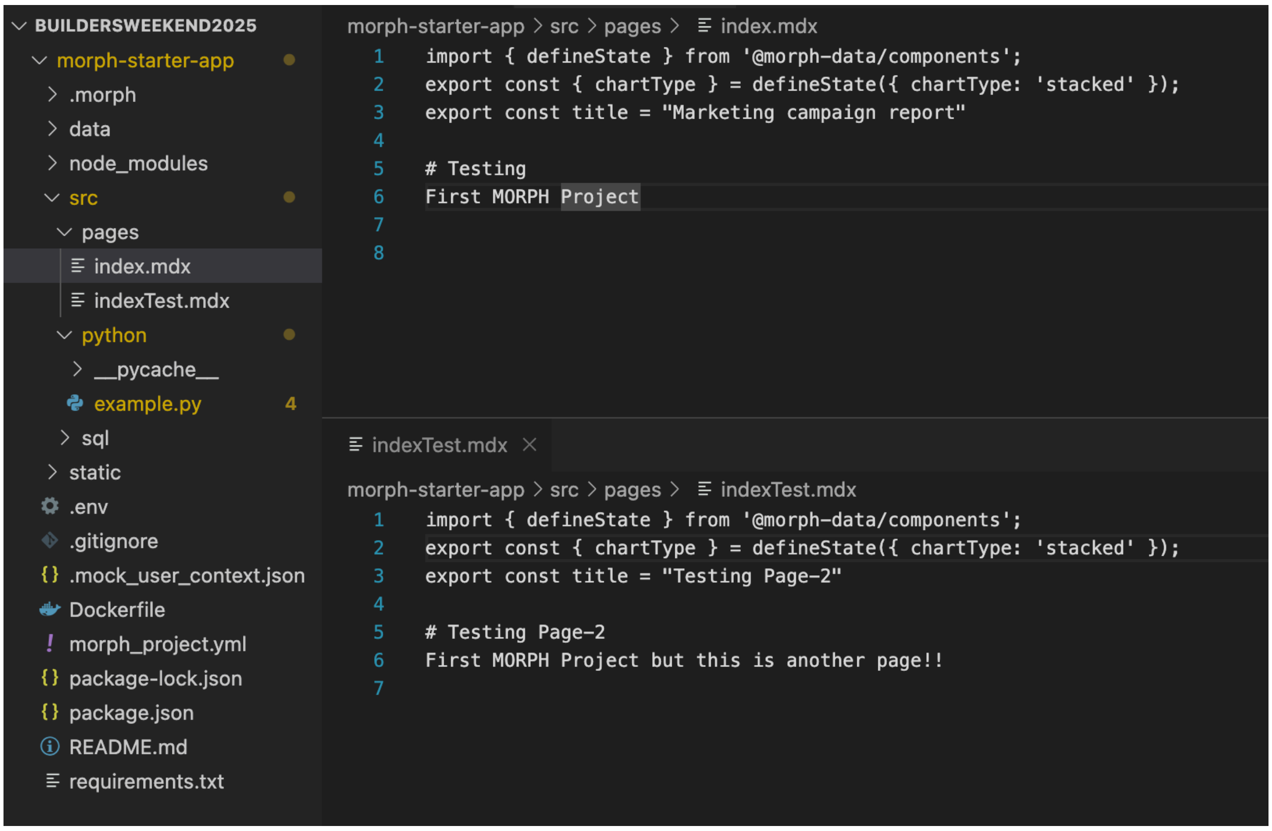
```
# Data Table

Specify the alias to reference the data as follows.

<DataTable data="example_dataframe" />
```

Steps to build the Backend with Python

1. Add Python Packages
    1. Add the necessary packages to `requirements.txt` or `poetry.toml`.
2. Define Python Functions and Add Morph Decorators
    1. *By adding Morph decorators to regular Python function definitions, you can set aliases*. This allows you to reference the results of these functions from other Python, SQL, or Markdown files.
    ```python
    import pandas as pd
    import morph
    from morph import MorphGlobalContext

# Add Morph decorators

@morph.func(name="example_dataframe")
@morph.load_data("example_data")
def example_dataframe(context: MorphGlobalContext):
df = context.data["example_data"]
return df

```
3. Test Python Functions
    1. Use the `morph run` command to execute functions for unit testing.
    2. For checking the function defined above
```shell
morph run example_data
```

4. Start the Development Server
    1. By starting the development server, you can check the web application locally.

FrontEnd-Adding Pages

1. Follows MDX Format, similar to MD files but with more cool features such as visual styling, graphs, etc.
2. ! To add pages/stuff, simply add more documents to the "src/pages" folder.
3. 

morph-starter-app > src > pages > index.mdx

```
import { defineState } from '@morph-data/components';
export const { chartType } = defineState({ chartType: 'stacked' });
export const title = "Marketing campaign report"

# Testing
First MORPH Project
```

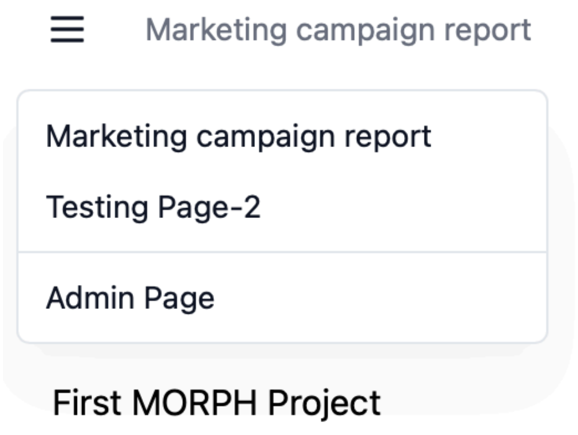
indexTest.mdx

morph-starter-app > src > pages > indexTest.mdx

```
import { defineState } from '@morph-data/components';
export const { chartType } = defineState({ chartType: 'stacked' });
export const title = "Testing Page-2"

# Testing Page-2
First MORPH Project but this is another page!!
```

Screenshot 2025-02-19 at 0.46.12 AM.png

4. You can see them on your website/app as follows-



Screenshot 2025-02-19 at 0.46.34 AM.png

Morph Decorators

@morph.func

By adding the `@morph.func` annotation, you can register the function to be executed with morph.

```python
@morph.func(
    name: str | None = None,
    description: str | None = None,
    output_type: Optional[
        Literal["dataframe", "csv", "visualization", "markdown", "json"]
    ] = None,
    result_cache_ttl: int = 0,
)
def func_name(context):
    # write your code here
```

Example

```python
@morph.func(
    name="example_python_cell",
    description="Example Python cell",
)
def get_data_from_database(context):
    # write your code here
    return pd.DataFrame({
        "name": ["John", "Doe"],
        "age": [20, 30],
    })
```

@morph.load_data

By adding the `@morph.load_data` annotation, you can store the output result of the function in context.data and use it.

```python
@morph.load_data(
    name: str,
)
def func_name(context):
    # write your code here
```

```
@morph.func
@morph.load_data("example_sql_cell")
def get_data_from_database(context):
    sql_result_df = context.data["example_sql_cell"]
    # write your code here using sql_result_df
    return sql_result_df
```

IMPORTANT - how to get data from SQL

In the starter example given,

```
@morph.func(name="example_dataframe") # Give the Function an Alias
@morph.load_data("example_data")      # This is used to load data using an
ALIAS from /src/sql folder
def example_dataframe(context: MorphGlobalContext):
    df = context.data["example_data"]
    return df
```

example_data is an SQL query file. Check this out

```
{{
    config(
        name = "example_data",
        connection = "DUCKDB"
    )
}}

select
    Date as data,
    Source as source,
    Traffic as traffic,
    Orders as orders
from
    read_csv_auto("./data/Traffic_Orders_Demo_Data.csv")
where
    Date >= '2023-01-01'
```

I'm thinking, we can just get our data as a CSV file, and just read it into the dataframe.

@morph.variables

You can declare variables passed from outside with the `@morph.variables` annotation.

```
@morph.variables(
    var_name: str, default: Optional[Any] = None
)
def func_name(context):
    # write your code here
```

Example of how to use-

```
@morph.func
@morph.variables("var1")
def get_data_from_database(context):
    var1_data = context.vars["var1"]
    # write your code here using var1_data
    return pd.DataFrame({})
```