

MADS - Deployment 2

It works on my machine

Raoul Grouls, 17 November 2025

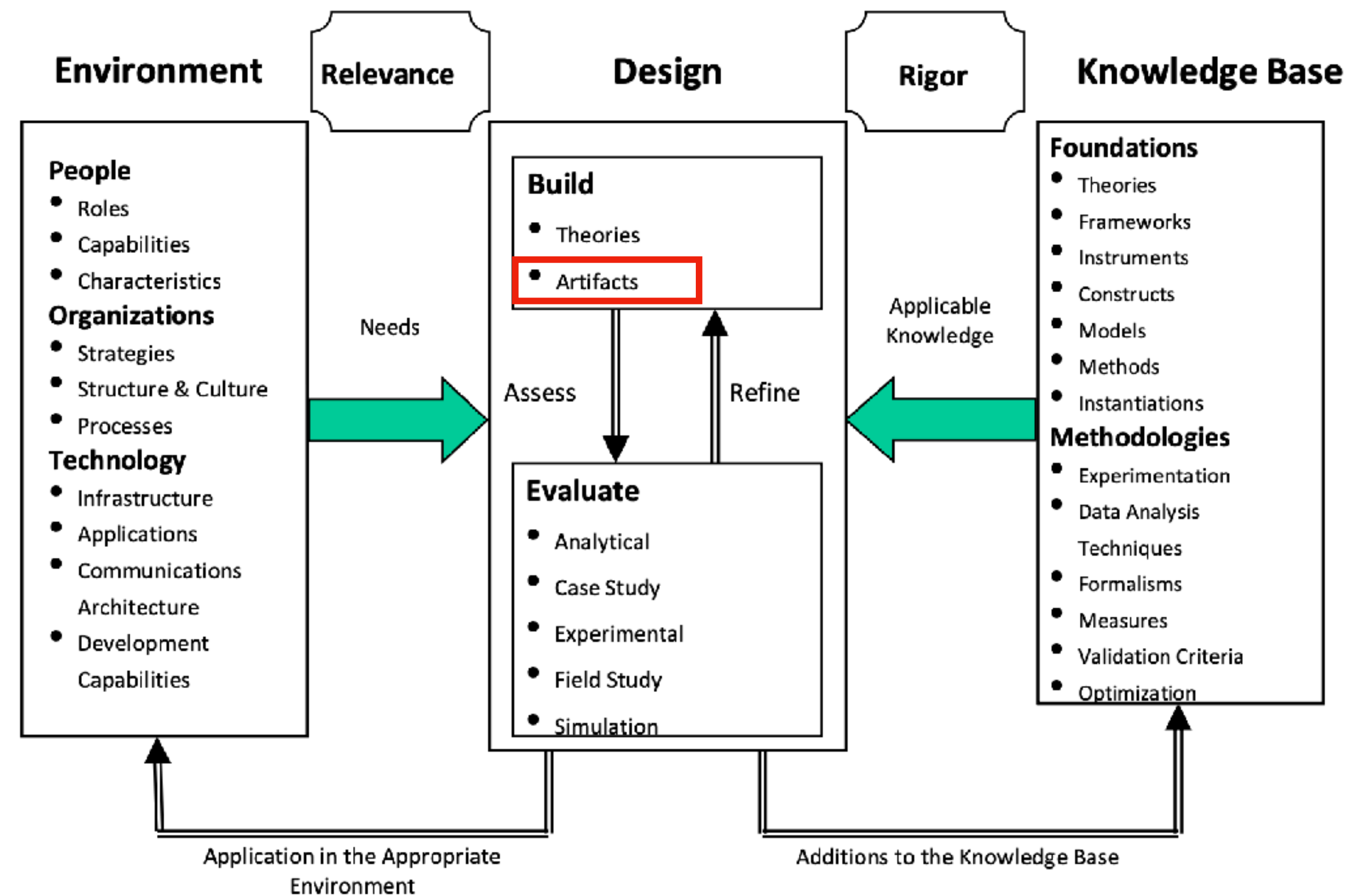


Figure 1: Design Science Research Framework (Adapted from (Hevner et al. 2004))

Ω and λ knowledge

- Descriptive (Ω) knowledge: informs the understanding of a problem, its context, and the underlying design of a solution entity.
- Prescriptive (λ) knowledge: solution design knowledge, in the form of growing design theories, informs the design of a solution entity, a design process or a design system.

Vom Brocke, J., Hevner, A., & Maedche, A. (2020). Introduction to design science research. In *Design science research. Cases* (pp. 1-13). Cham: Springer International Publishing.

Artefacts we build

- Trained model weights (.pth files) and BPE-tokenizer files
- Docker images that serve the model
- Docker compose networks that run the pipeline / dashboard
- Wheel files that publish the code to create and serve the model
- Makefile automate the creation of all the artefacts

Docker Fundamentals: Key Concepts

Dockerfile -> builds -> **Image** -> runs -> **Container**

You should have produced something like this

```
1 FROM python:3.12-slim
2 WORKDIR /app
3 RUN pip install --no-cache-dir requests loguru
4 COPY ingest/ingest.py .
5
6 CMD ["python", "ingest.py"]
```

```
1 FROM python:3.11-slim
2 WORKDIR /app
3 RUN pip install --no-cache-dir pandas loguru pyarrow
4 COPY preprocess/preprocess.py .
5
6 CMD ["python", "preprocess.py"]
```

```
1 FROM python:3.12-slim
2 COPY model/requirements.txt .
3 RUN pip install --no-cache-dir -r requirements.txt
4
5 WORKDIR /app
6 COPY model/*.py .
7 EXPOSE 8000
8
9 ENTRYPOINT ["uvicorn", "serve:app", "--host", "0.0.0.0", "--port", "8000"]
```

This will work.

However...

```
1 build:
2 |     docker build -t mads-ingest -f ingest/ingest.Dockerfile .
3 |     docker build -t mads-preprocess -f preprocess/preprocess.Dockerfile .
4 |     docker build -t mads-model -f model/serve.Dockerfile .
5 |
6 run:
7 |     docker run \
8 |         -v ./data:/app/data \
9 |         -v ./logs:/app/logs mads-ingest
10 |     docker run \
11 |         -v ./data:/app/data \
12 |         -v ./logs:/app/logs mads-preprocess
13 |     docker run \
14 |         -v ./data:/app/data \
15 |         -v ./logs:/app/logs \
16 |         -p 8000:8000 \
17 |         mads-model
18 |
19 clean:
20 |     rm -rf data/*
```


- But using docker-compose is better
- It makes it easier to combine multiple docker containers

```
1 services:
2   ingest:
3     build:
4       context: .
5       dockerfile: ingest/ingest.Dockerfile
6     volumes:
7       - ./data:/app/data
8       - ./logs:/app/logs
9
10  preprocess:
11    build:
12      context: .
13      dockerfile: preprocess/preprocess.Dockerfile
14    volumes:
15      - ./data:/app/data
16      - ./logs:/app/logs
17    depends_on:
18      - ingest
19
20  model:
21    build:
22      context: .
23      dockerfile: model/serve.Dockerfile
24    volumes:
25      - ./data:/app/data
26      - ./logs:/app/logs
27    ports:
28      - "8000:8000"
29    depends_on:
30      - preprocess
31
32    healthcheck:
33      test: ["CMD-SHELL", "curl -f http://localhost:8000/health && exit 0 || exit 1"]
34      interval: 10s
35      timeout: 10s
36      retries: 3
37      start_period: 10s
38    restart: unless-stopped
39
```

Docker compose

Services Architecture:

- Three connected services: ingest, preprocess, and model
- Pipeline flow: ingest → preprocess → model

Common Patterns:

- Each service has dedicated Dockerfile
- Shared volumes for data and logs
- Volume mounts persist data to host machine

Key Features:

- Model service exposes port 8000
- Health checks on model service

- Automatic restart policy (restart: unless-stopped)
- Service dependencies ensure correct startup order

Volume Configuration:

- ./data:/app/data - shared data directory
- ./logs:/app/logs - centralized logging

Health Checks (model):

- Used in `docker ps` to report health
- 10s intervals with 3 retries
- 10s startup grace period
- Curl test to /health endpoint

Docker compose commands

`docker compose up`

- Starts all services defined in compose file
- Shows logs in terminal
- Creates networks/volumes if needed
- Builds images if not present

`docker compose up -d`

- Starts services in detached mode
- Runs in background
- No logs in terminal
- Returns control to shell

`docker compose down`

- Stops all running containers
- Removes containers and networks
- Preserves volumes by default
- Add `-v` flag to remove volumes

`docker ps`

- Shows running containers
- Displays: Container ID, Image, Command, Status, Ports
- Add `-a` flag (for “all”) to show stopped containers
- Useful for checking container health

uv-example

An example where

- curl is installed
- uv is installed to speed up installation of dependencies

```
18 # Base image
17 FROM python:3.12
16
15 # this updates dependencies, installs curl and
14 # rm -rf /var removes files only necessary during installation
13 RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/*
12
11 WORKDIR /app
10
9 ADD --chmod=755 https://astral.sh/uv/install.sh /install.sh
8 RUN /install.sh && rm /install.sh
7
6 COPY ./requirements.txt .
5 RUN /root/.local/bin/uv pip install --system --no-cache -r requirements.txt
4
3 COPY test.py test.py
2 ENTRYPOINT ["python", "test.py"]
1
```

Smaller torch builds

Dockerfile:

```
FROM raoulgrouls/torch-python-slim:py3.12-torch2.8.0-arm64-uv0.9.8
```

pyproject.toml:

```
[[tool.uv.index]]  
name = "pytorch"  
url = "https://download.pytorch.org/whl/cpu"  
explicit = true
```

Makefile

- **Target:** File to be created or action to perform
- **Prerequisites:** Files or targets needed before recipes can run
- **Recipe:** Commands (indented with TAB) that make runs to create target or perform action
- `.PHONY` tells make these targets don't create actual files

```
target: prerequisites  
    recipe  
    recipe  
    ...
```

Makefile

- Checks if target (\$@) is missing or older than source (\$<)
- Creates directory and copies only if needed
- Automatic variables:
 - \$< (first prerequisite),
 - \$@ (target name)

Chain of dependencies:

- Run depends on clean and build
- Build depends on img/clustering.png
- Make resolves these automatically in correct order

```
29 img/clustering.png: .. /img/clustering.png
30 |     mkdir -p img
31 |     cp $< $@
32 |
33 build: img/clustering.png
34 |     @echo "${YELLOW}Building Docker image ... $(NC)"
35 |     docker build -t $(IMAGE_NAME):$(IMAGE_TAG) .
36 |     @echo "${GREEN}Build complete!$(NC)"
37
```

```
35 run: clean build
36 |     @echo "${YELLOW}Starting container ... $(NC)"
37 |     docker run -d -p $(PORT):$(PORT) --name $(IMAGE_NAME) $(IMAGE_NAME):$(IMAGE_TAG)
38 |     @echo "${GREEN}Service running on http://localhost:$(PORT)$(NC)"
39 |     @echo "${GREEN}View image on on http://localhost:$(PORT)/show$(NC)"
40
```


Straattaal

- artefacts: contains all artefacts we create

```
•
├── artefacts
│   ├── config.json
│   ├── history.txt
│   ├── model.pth
│   └── tokenizer.json
├── assets
│   └── straattaal.txt
├── backend
│   ├── static
│   │   ├── index.html
│   │   └── styles.css
│   ├── app.py
│   ├── requirements.txt
│   └── utils.py
├── dist
│   ├── slanggen-0.4-py3-none-any.whl
│   └── slanggen-0.4.tar.gz
├── logs
│   ├── app.log
│   └── main.log
├── src
│   └── slanggen
│       ├── __init__.py
│       ├── custom_logger.py
│       ├── datatools.py
│       ├── main.py
│       └── models.py
├── Dockerfile
├── Makefile
├── README.md
├── pyproject.toml
├── requirements-dev.lock
├── requirements.lock
└── slanggen.toml
```


Straattaal

- assets: data we obtained

```
•
├── artefacts
│   ├── config.json
│   ├── history.txt
│   ├── model.pth
│   └── tokenizer.json
├── assets
│   └── straattaal.txt
├── backend
│   ├── static
│   │   ├── index.html
│   │   └── styles.css
│   ├── app.py
│   ├── requirements.txt
│   └── utils.py
├── dist
│   ├── slanggen-0.4-py3-none-any.whl
│   └── slanggen-0.4.tar.gz
├── logs
│   ├── app.log
│   └── main.log
├── src
│   └── slanggen
│       ├── __init__.py
│       ├── custom_logger.py
│       ├── datatools.py
│       ├── main.py
│       └── models.py
├── Dockerfile
├── Makefile
├── README.md
├── pyproject.toml
├── requirements-dev.lock
├── requirements.lock
└── slanggen.toml
```

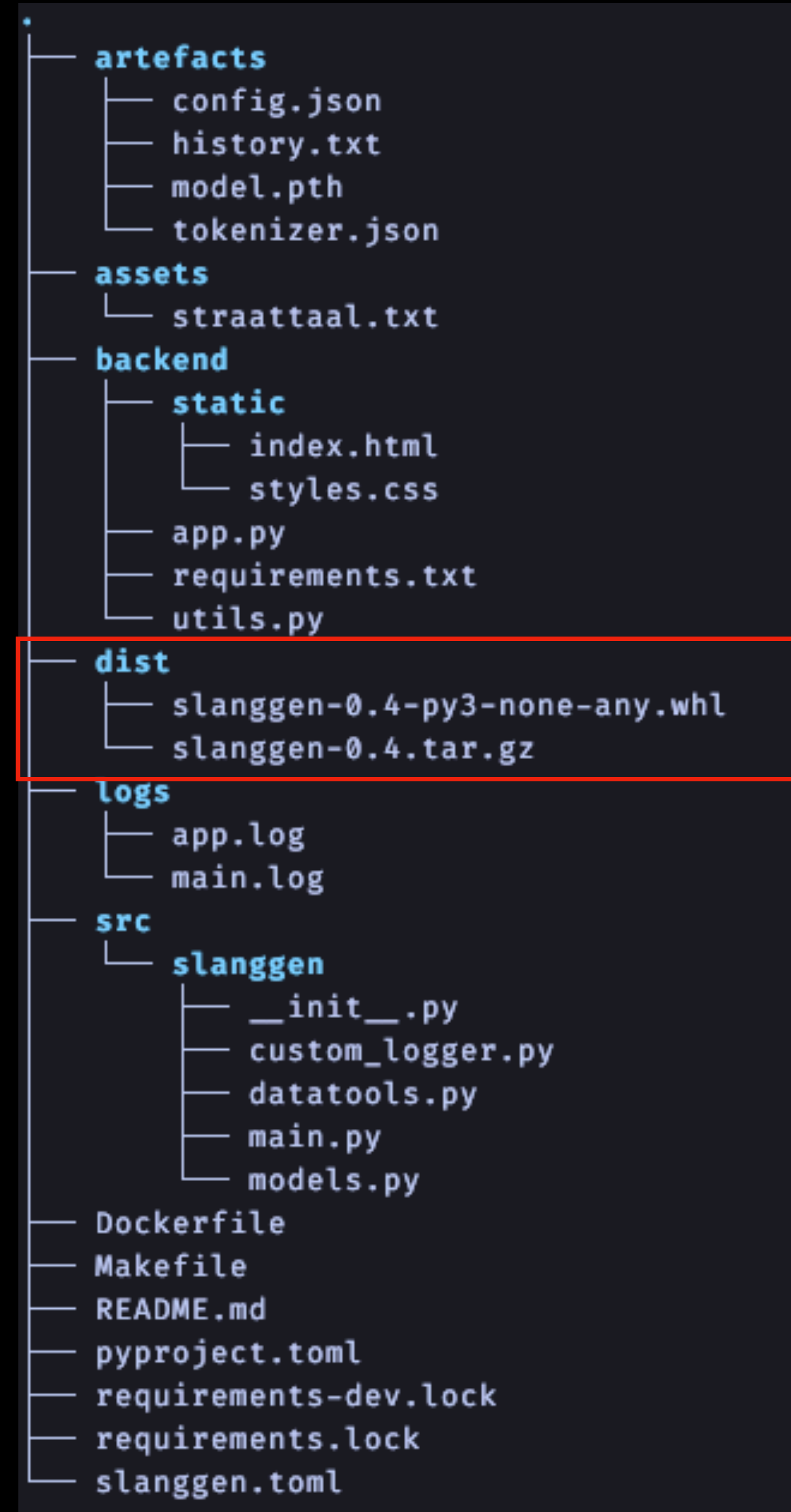
Straattaal

- src/slanggen: python src code for scraping, preprocessing and training a PyTorch model

```
•
├── artefacts
│   ├── config.json
│   ├── history.txt
│   ├── model.pth
│   └── tokenizer.json
├── assets
│   └── straattaal.txt
├── backend
│   ├── static
│   │   ├── index.html
│   │   └── styles.css
│   ├── app.py
│   ├── requirements.txt
│   └── utils.py
├── dist
│   ├── slanggen-0.4-py3-none-any.whl
│   └── slanggen-0.4.tar.gz
├── logs
│   ├── app.log
│   └── main.log
├── src
│   └── slanggen
│       ├── __init__.py
│       ├── custom_logger.py
│       ├── datatools.py
│       ├── main.py
│       └── models.py
├── Dockerfile
├── Makefile
├── README.md
├── pyproject.toml
├── requirements-dev.lock
├── requirements.lock
└── slanggen.toml
```

Straattaal

- We can publish the src/slanggen code as a wheel file.
- We can install the wheel in the Dockerfile
- This makes it easier to import the model from `models.py` without worrying about dependencies



Straattaal

`dist/` is the distribution directory containing packaged versions

`slanggen-0.4-py3-none-any.whl`

- Wheel format (.whl) - faster to install than source distributions
- py3: Python 3 compatible
- any: Works on any platform

`slanggen-0.4.tar.gz`

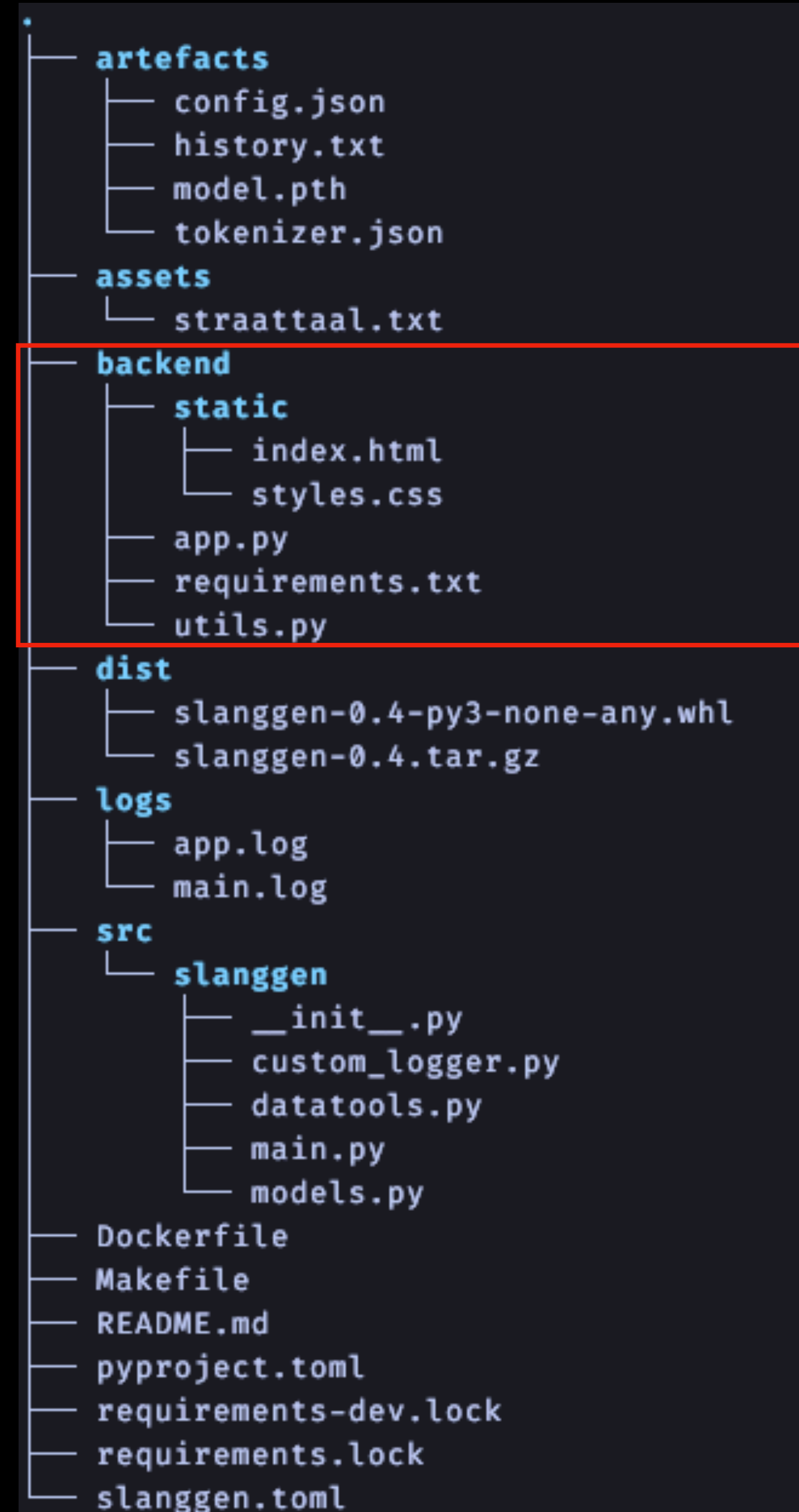
- Source distribution (sdist)
- Contains raw source code and build instructions
- Fallback if wheel installation fails
- Required for PyPI distribution

Both files represent version 0.4 of the slanggen package, just in different formats.



Straattaal

- backend: python code that uses the trained model for inference
- We can access slanggen (from `slanggen import models`) because we installed the package from the wheel
- Includes static folder with css/html ; you could separate this as a stand alone docker container for the frontend



Installing dist

```
RUN --mount=source=dist,target=/dist PYTHONDONTWRITEBYTECODE=1 pip install --no-cache-dir /dist/*.whl
```

- RUN: Dockerfile instruction that executes commands during image build
- `--mount=source=dist,target=/dist`: Temporarily mounts local `dist` directory to `/dist` in container during build (why is that?)
- `PYTHONDONTWRITEBYTECODE=1`: Prevents Python from creating `.pyc` files
 - Container images should be immutable and `.pyc` files create varying builds
 - Adds unnecessary size to container images
 - Can cause permission issues in some container setups
- `pip install --no-cache-dir /dist/*.whl`: Installs all wheel files from mounted `/dist` directory without caching