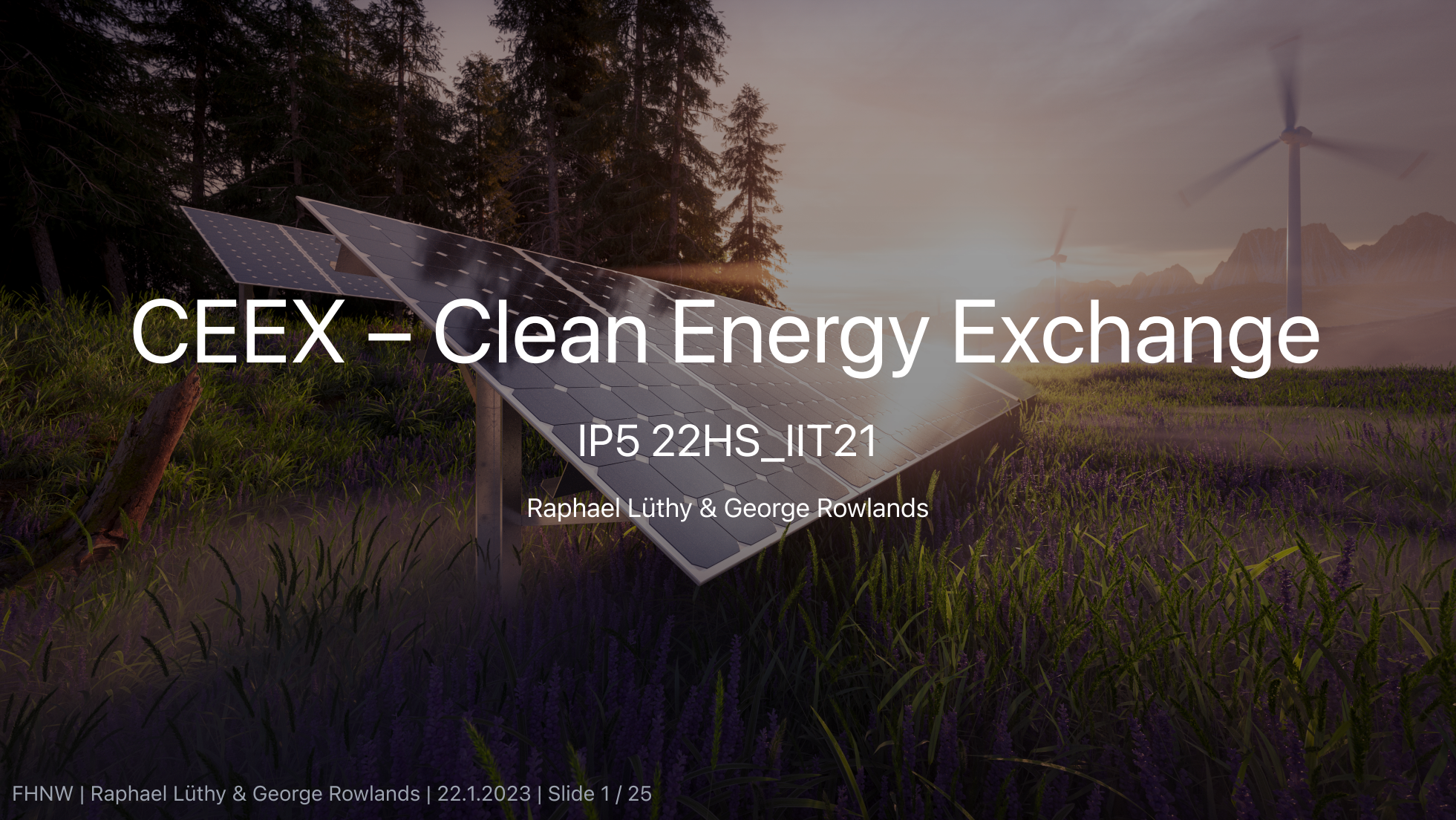


CEEX – Clean Energy Exchange

IP5 22HS_IIT21

Raphael Lüthy & George Rowlands

Table of contents

- Who are we?
- What is CEEX
- Initial State
- Problems and their Solutions
- Learnings
- Future Work
- QnA

Who are we

Raphi & Groog

What is CEEX

Initial State

Initial State

Code Base

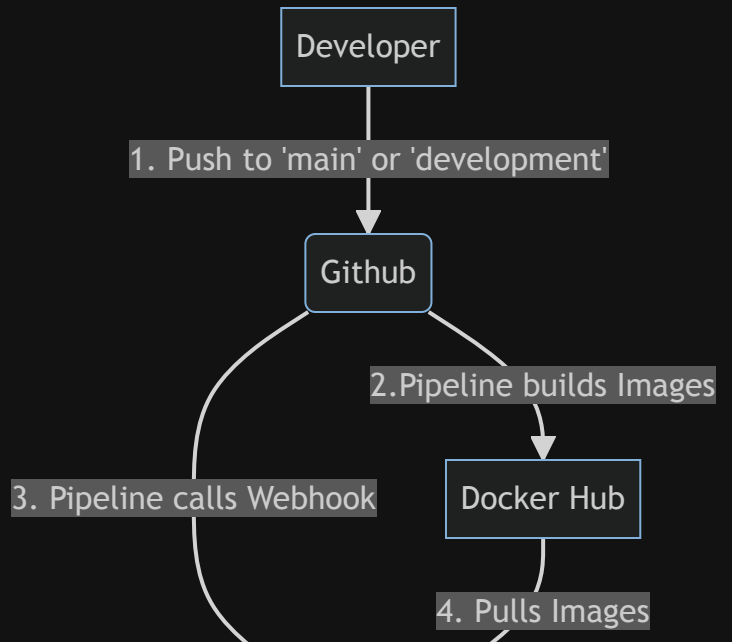
- Hard Coded Components
- Hard Coded Values

```
<mat-option value="option1">current Week</mat-option>
<mat-option value="option2">current Month</mat-option>
<mat-option value="option3">last Month</mat-option>
<mat-option value="option4">last 3 Month</mat-option>
<mat-option value="option4">last Year</mat-option>
```

```
export class DataService {
    baseurl = 'http://127.0.0.1:8000/';
    resetUrl = this.baseurl + 'auth/request-reset-ema
...
}
```

Deployment & Architecture

- Server
- Pipeline
- SSL / Reverse Proxy



Initial State

Hard Coded Components

```
<mat-card class="list-element">
    <img id="img1" src="../../../assets/solarHouse.png" alt="customerImage">
    <div class="list-element__info ">
        <div class="horizontal_info">
            <h2>Max Mustermann</h2>
            <img src="../../../assets/solar.png" alt="solarEnergy">
        </div>
        <span id="amount">200kWh</span>
        <div class="horizontal_info last_row ">
            <div>
                <img id="pin" src="../../../assets/pin.png" alt="pin">
                <span id="distance">300m</span>
            </div>
            <span id="horizontal_info">0.18 CHF / kWh</span>
        </div>
    </div>
</mat-card>
```

Initial State

Hard Coded

```
export class DataService {
    baseurl = 'http://127.0.0.1:8000/';
    resetUrl = this.baseurl + 'auth/request-reset-email/';
...
}
```

Issues

- Maintainability
  - -> Hard to keep track where to change values
- Reusability
  - -> Can’t be reused if different values are needed
- Deployability
  - -> Deployments need to change values at runtime

Initial State

Containerization, Pipeline and Deployment

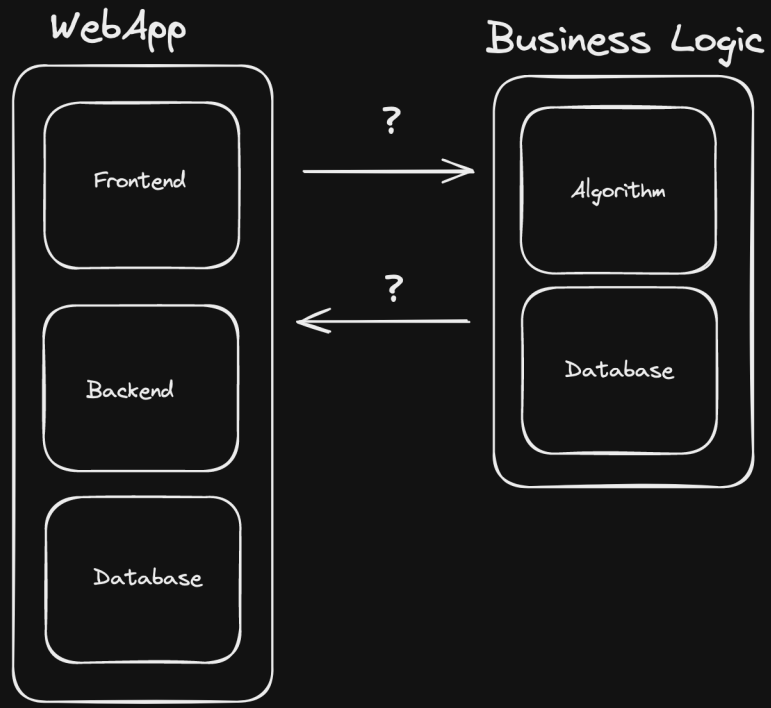
- No Containerization
- No Pipeline
- No Server

backend	Update README.md	5 months ago
frontend	Update README.md	5 months ago
.gitignore	updated gitignore with pycache	6 months ago
README.md	Update README.md	9 months ago

Initial State

Architecture

- What Side does what?
- Who owns which data?
- How does the communication between Customer and Business Logic work?
- Who owns the Authentication?

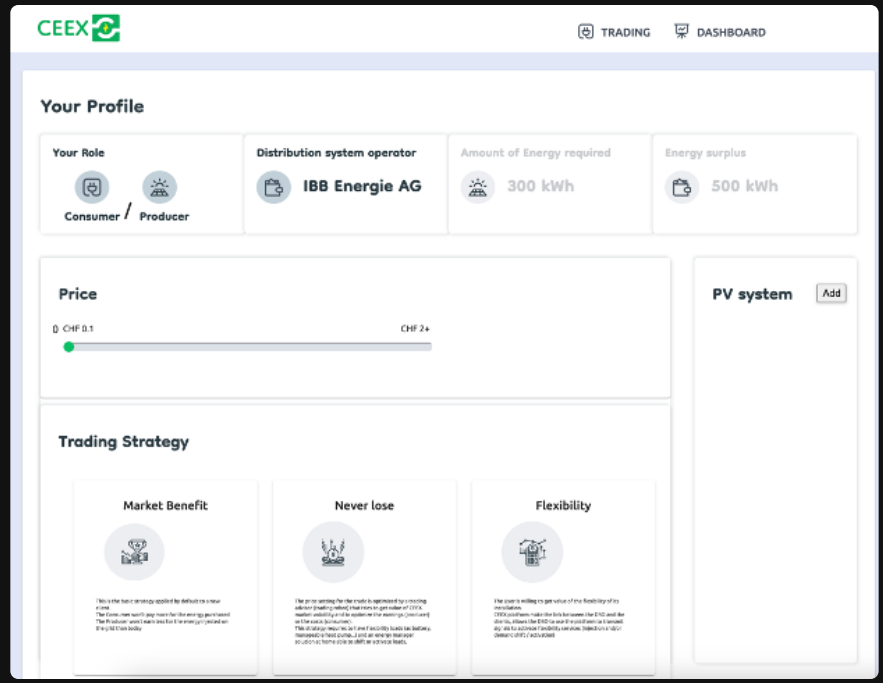


Initial State

Usability

- A lot of different fields
- Functionalities with insufficient titles
- No clear distinction of importance
- No clear indication of what interactive

-> Items need more explanation

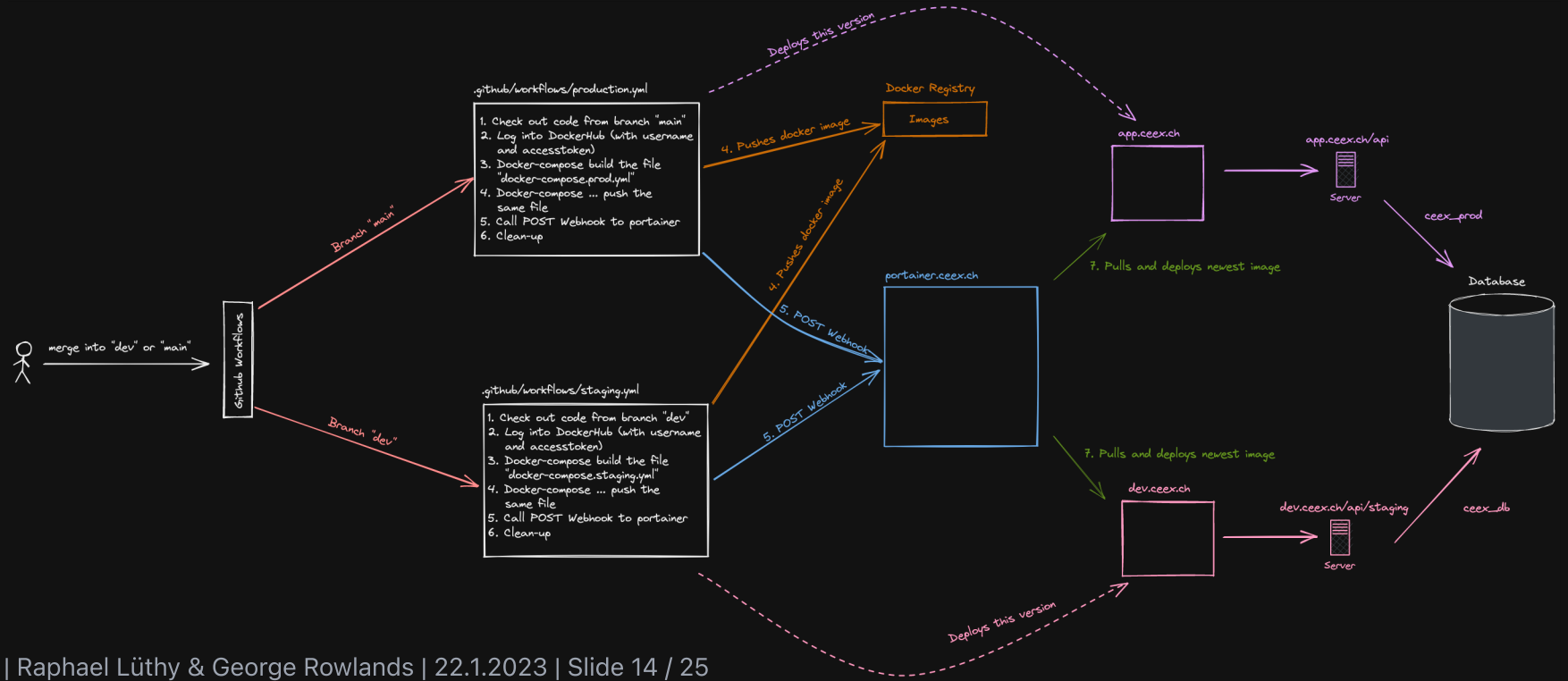


Solutions

Deployment & Containerization

Deployment & Containerization

Server setup

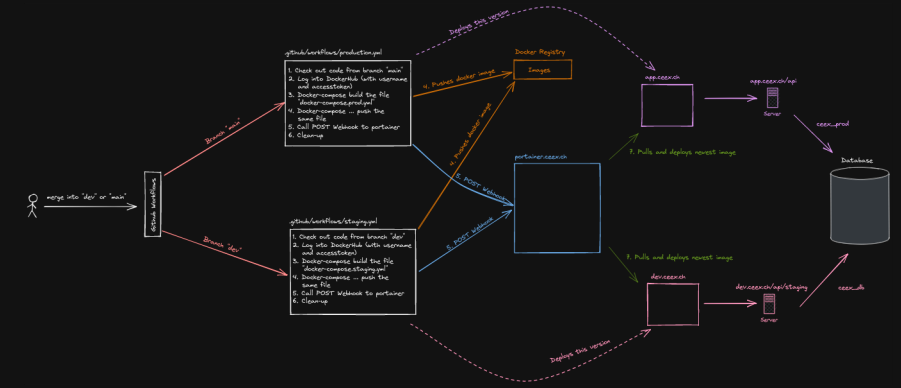


Deployment & Containerization

Server setup

Steps to take:

- Dockerfiles (Containerize Application)
- Server
- Reverse Proxy (traefik)
- Portainer
- Pipeline



Deployment & Containerization

Dockerfiles

Frontend:

```
FROM node:14-alpine as build
WORKDIR /app
RUN npm install -g @angular/cli@13.3.10
RUN npm install -g serve@14.1.2
COPY ./frontend/package.json ./
RUN npm install --force
COPY ./frontend .
RUN ng build --configuration=production
  --output-path=dist
EXPOSE 3000
CMD ["serve", "./dist", "-p", "3000"]
```

Backend:

```
FROM python:3.10-alpine3.15
  # stops stdout and stderr streams being buffered
ENV PYTHONUNBUFFERED 1
WORKDIR /app/backend
COPY backend/requirements.txt /app/backend/
RUN ["pip", "install", "-r", "requirements.txt"]
COPY backend/ceex_api /app/backend/
EXPOSE 8000
```

Deployment & Containerization

Server

Actual Server:

Reverse Proxy:

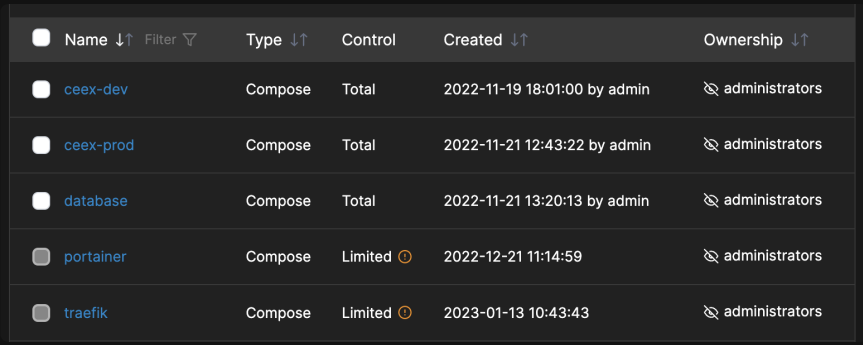
Switch Engines Website

Traefik Documentation

Deployment & Containerization

Portainer

- Control of Images / Containers
- Redeployment of latest Image via Webhook
- Stacks



Name	Type	Control	Created	Ownership
ceex-dev	Compose	Total	2022-11-19 18:01:00 by admin	administrators
ceex-prod	Compose	Total	2022-11-21 12:43:22 by admin	administrators
database	Compose	Total	2022-11-21 13:20:13 by admin	administrators
portainer	Compose	Limited	2022-12-21 11:14:59	administrators
traefik	Compose	Limited	2023-01-13 10:43:43	administrators

Deployment & Containerization

Portainer: "Stack"

- Specifies Docker Images
- Connect the backend to the database
- Expose UI and API to Reverse Proxy

```yaml
version: '3.9'
services:
    ui:
        image: 'raphaelluethy/ceex:ui-prod-latest'
        networks:
          - proxy
    api:
        image: 'raphaelluethy/ceex:api-prod-latest'
        networks:
            - proxy
            - database

networks:
    proxy:
        name: 'proxy'
        external: true
    database:
        name: 'database'
        external: true
```

Deployment & Containerization

Pipeline

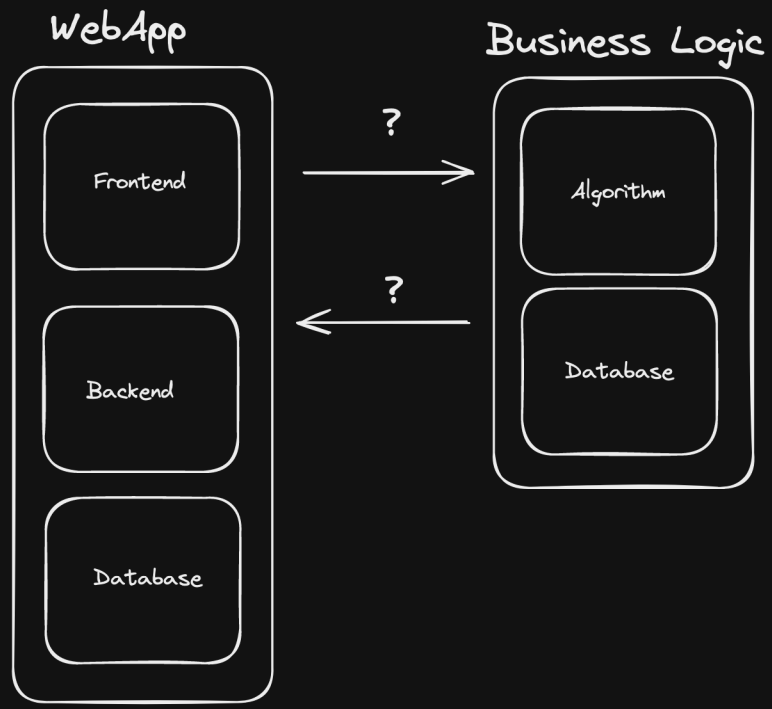
1. Push to Branch "main" (production) or "development" (staging)
2. Spin up Ubuntu VM
3. Checkout the Source Code
4. Login to Docker Hub
5. Build the Docker Image
6. Push the Docker Image to Docker Hub
7. Call Webhook of Portainer
8. Cleanup

```yaml
on:
  push:
    branches:
      - main
jobs:
  build_push_containers:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2
      - name: Login docker
        uses: docker/login-action@v1
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}
      - name: Build Docker image
        run: docker compose -f
          docker-compose.prod.yml build
      - name: Push Docker image
        run: docker compose -f
          docker-compose.prod.yml push
      - name: Call webhook
        uses: satak/webrequest-action@master
        with:
          url: ${{ secrets.WEBHOOK_PROD_URL }}
          method: POST
      Here comes cleanup...
```

Architecture

Original Problems:

- What Side does what?
- Who owns which data?
- How does the communication between Customer and Business Logic work?
- Who owns the Authentication?



Architecture

Data Agreement

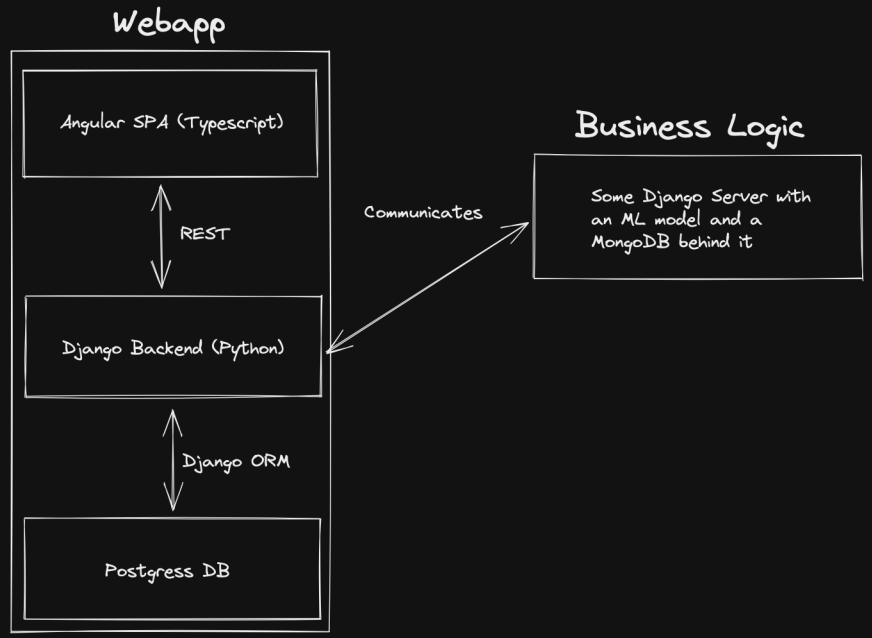
Webapp:

- Own Authentication Layer
- Stores what user belongs to which ***Node***
- Provides Proxy for user to communicate with Business Logic

Business Logic:

- Home of the Algorithm
- Stores all the data related to trading

Node: Maps the actual household to the graph of the algorithm



Proxy

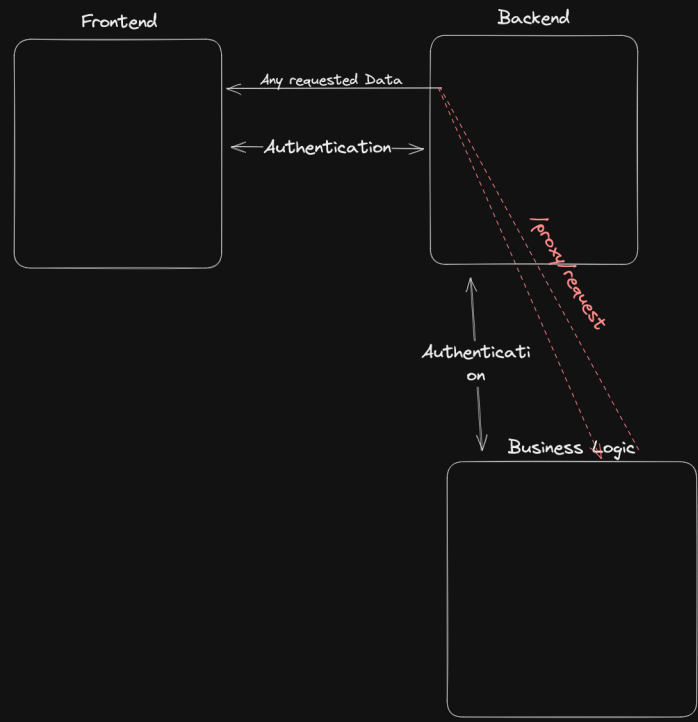
Frontend:

```
return this.http.post(`${this.proxyBaseUrl}`, {
    url: `node/${nodeId}/
            metrics?startDate=${startDateTime}
            &endDate=${endDateTime}`,
    method: 'GET',
});
```

Backend:

```
response = requests.request(
    method=method,
    url=f'{remote_url}{request_url}',
    allow_redirects=False
    )

return JsonResponse(response.json(), safe=False)
```



Thank you for listening 

Any Questions?