

Zwischenbericht Jan Krüger

- Zwischenbericht Jan Krüger
 - Vorwort
 - Aufbau der Monorepo
 - Apps
 - Libs
 - Docs
 - Aktueller Fortschritt
 - 1. Rooftop-api (Im gesamten von mir geschrieben)
 - 2. Farmer-companion (Im gesamten von mir geschrieben)
 - 3. Datacollector
 - 4. Infrastruktur
 - 5. Sonstiges
 - Ausblick

Vorwort

Aufbau der Monorepo

Der gesammte Sourcecode des Softwareprojekt wird in einer Monorepo gemanaged. Dies bietet den Vorteil, dass der gesamte Sourcecode an einer zentralen Stelle zu finden ist und sich Code einfach teilen lässt.

Eine Monorepo benötigt allerdings auch Tooling um die Verwaltung des Sourcecodes zu vereinfachen. Dabei haben wir auf das Framework **NX** gesetzt.

Im nachfolgenden wird der Aufbau der Monorepo erklärt um das Auffinden von Sourcecode zu vereinfachen.

Die Monorepo lässt sich in 3 wichtigste Ordner unterteilen.

- apps
- libs
- docs

Apps

Der Apps Ordner enthält alle Anwendungen.

- **apps/datacollector**
 - Sourcecode für den Datacollector.
 - Geschrieben in C++ mit Arduino Framework
- **apps/dwd-weather-importer**
 - Skript zum importieren historischer Wetterdaten mittels BrightSky API
 - Geschrieben in Typescript
- **apps/farmer-companion**
 - Sourcecode für die Frontend Anwendung des Projekts
 - Geschrieben in HTML, SCSS und Typescript

- Verwendet Angular 14 und Ionic 6
- **apps/proxy**
 - Proxy Server als Brücke zwischen lokalem Netz und API
 - Geschrieben in Python
 - Voraussichtlich nicht benötigt, da ein Datacollectoren direkt mit der Cloud-API kommunizieren
- **apps/rooftop-api**
 - Cloud-API die als zentrale Api für die komplette Anwendung dient
 - Geschrieben in Typescript
 - Verwendet NestJS, TypeORM
 - Kommunikation über REST-API
 - Dokumentation gehostet unter [/api](#) mit OpenAPI 3

Libs

Hier werden Bibliotheken die in der Gesamten Monorepo verwendet werden können erstellt. Aktuell gibt es keine Bibliotheken.

Docs

Hier liegt alles an Dokumentation.

Das Design ist im Tool Figma [hier](#) eingesehen werden.

Aktueller Fortschritt

1. Rooftop-api (Im gesamten von mir geschrieben)

- Authentifizierung über JWT nach dem OAuth2 implicit Grant
 - OAuth Consumer applications Entität
 - JWTs erstellen (Access und Refresh Token)
 - Benutzer verwalten
- Passwort zurücksetzen über E-Mail
 - Anbindung an SMTP Server
 - E-Mail Templating mit Handlebars
- Datacollector Konfiguration **configurations** unterordner
 - Anlegen verschiedenster Entitäten
 - CRUD für alle Entitäten
 - REST Routen für Relationen
 - [/register](#) Route zum anmelden eines Datacollectors an die API
 - [/heartbeat](#) Route zum melden, dass ein Datacollector aktiv ist.
 - Server Sent Events, wenn sich die Konfiguration eines Datacollectors ändert.
- InfluxDB Wrapper **influx-db** unterordner
 - Wrapper um den InfluxDB Nodejs client um die Erweiterung von Typen
 - InfluxDB Service
- Sensoren Konfiguration
 - Anlegen verschiedenster Entitäten (**Sensor**, **SensorInterface**)
 - CRUD Routen für alle Entitäten
- Wetterdaten **weather** unterordner
 - Auslesen der lokalen Wetterdaten aus der InfluxDB

- Über einen Service Worker
- Eventbus für neue Wetterdaten
- Auslesen der aktuellen Wetterdaten aus der BrightSky API
 - Schreiben der DWD Wetterdaten in die InfluxDB
 - Über einen Service Worker
 - Eventbus für neue Wetterdaten
- Zusammenfassen des Wetters für den heutigen Tag
 - Bevorzugung lokaler Wetterdaten
 - Fallback auf DWD Wetterdaten
 - Vorhersage aus DWD Wetterdaten
- REST Routen für die Wetterdaten (Aktuell, Vorhersage, Heute)
- SSE Routen für Wetterdaten(Aktuell, Vorhersage)
- Beete **beds** unterordner
 - Anlegen Beds Entität
 - CRUD Routen
- Pflanzen **plants** unterordner
 - Anlegen Plants Entität
 - CRUD Routen

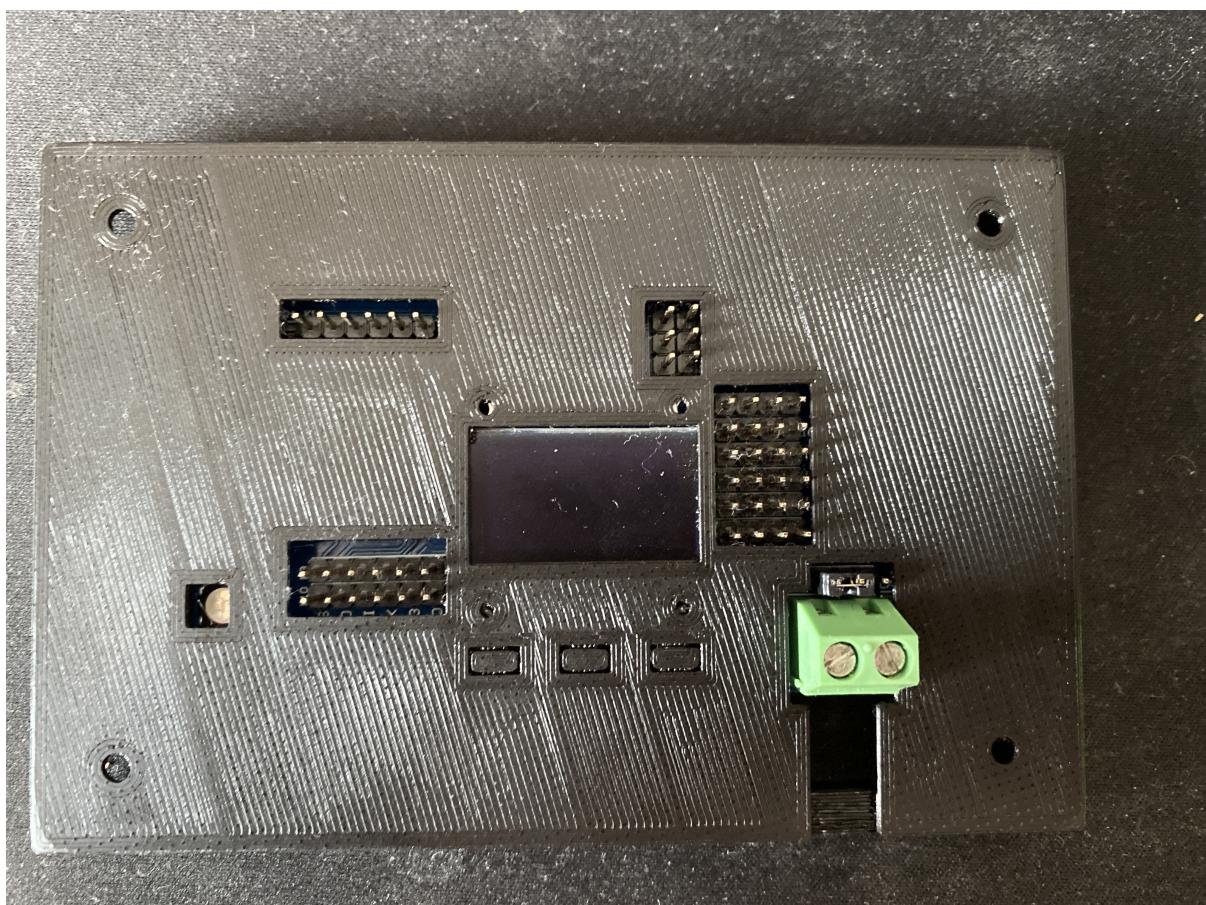
2. Farmer-companion (Im gesamten von mir geschrieben)

Der gesamte Sourcecode ist im **src/app** unterordner zu finden

- Templating verschiedensten Seiten
 - Landing Page
 - Login Seite
 - Passwort zurücksetzen Seiten
 - Startseite
 - Einstellungsseiten
 - Inkl. Unterseiten
 - Sidebar Navigation
- Funktionalität verschiedenster Seiten
 - Startseite
 - Wetter Widget (Aktuelles Wetter)
 - Navigation Slider
 - Boards Seite
 - Anbindung an API
 - Speichern der Daten
 - Laden von Bestandsdaten
 - Sidebar Navigation
- Widgets
 - Wetter Widget (Aktuelles Wetter)
 - Wetter Widget (7 Tage Vorhersage)
 - Link Slider (Slider-Carousel für Navigation)
- Einbindung übersetzungs Bibliothek **ngx-translate**
 - Einbindung auf allen Seiten
 - Übersetzung aller Seiten auf Englisch und Deutsch

3. Datacollector

- Planung der Kommunikation zwischen API und Datacollector
 - Initiale Konfiguration
 - Heartbeat
 - Sensor Konfiguration
- Design eines Gehäuses für die Platine in Fusion 360
 - Gedrucktes Gehäuse



- [Link](#) zum 3D Modell

4. Infrastruktur

- Hilfe bei Aufräumen des VPS
- Installation und Einrichtung PostgresDB
- Bereitstellung und deployment der Rooftop-API als Dockercontainer über GitHub

5. Sonstiges

- Unterstützende Funktion beim Design (Initiales Design)
- Übersicht aller Seiten und Funktionen der Apö
- "Gesamtbild" über die Anwendung und die Fähigkeiten
- Ausarbeitung Navigationskonzept
 - Unterseiten
 - Navigationsart etc...
 - Navigationspfade
- Einrichtung NX Monorepo
 - Einrichtung DevOps Tools in der Monrepo

- Linter (eslint)
- Fomatter (prettier)
- Commitlinting
 - In Kombination mit Husky (Git Hooks)
- Versionierung und Changelog generierung (semver)
- Einrichtung GitHub

Für Rückfragen stehe ich gerne zur Verfügung.

Ausblick

Im weiteren Verlauf des Projekts werde ich mich Fullstack mit der Rooftop-Api und dem Farmer-Companion beschäftigen. Aktuell arbeite ich an der Funktionalität aller Einstellungsseiten. Sobald die Datacollectoren soweit sind ihre gemessenen Daten zu senden, werde ich dies Priorisieren und die Darstellung der Daten in die App ermöglichen. Anschließend werde ich die manuelle Bewässerung ermöglichen. Zuletzt werde ich das Automatisierungsframework implementieren, sodass Automatisierungsregeln ähnlich wie in [Home Assistant](#) eingestellt werden können.

Schwierigkeiten sehe ich aktuell nur bei dem Design der Automatisierungen, wie diese effizient in die Datenbank gespeichert werden und über einen Service Worker ausgeführt werden können.