WEIHENSTEPHAN · TRIESDORF
University of Applied Sciences

Hochschule Weihenstephan-Triesdorf
Wintersemester 24/25

Anwendung digitaler Technologien in der Prozessindustrie

Gruppe 4: Gnalian Samuel und Sterk Franziska

Erstellt mit Copilot

Inhaltsverzeichnis

1. Anforderungen Nutzer und Entwickler – Spezifikationen Datenprodukt

1. **Einführung:**
   Das Ziel des Projekts ist die Entwicklung einer App zur Überwachung von Umgebungszuständen. Dabei werden Sensordaten wie Temperatur, Luftfeuchtigkeit und Luftdruck erfasst, über einen MQTT-Broker bereitgestellt, in einer PostgreSQL-Datenbank gespeichert und über ein Streamlit-Dashboard visualisiert.

2. **Anforderungen der Nutzer:**
   - Zielgruppe & Nutzergruppen:

Tabelle 1: Beschreibung der Zielgruppen

Nutzer	**Beschreibung**
Allgemeine Nutzer	Möchten die Sensordaten in einer benutzerfreundlichen Oberfläche einsehen
Technisch versierte Nutzer	Benötigen detaillierte Datenanalysen und Filtermöglichkeiten

   - Funktionale Anforderungen:

Tabelle 2: Anforderungen an App

Anforderung	**Beschreibung**
Echtzeit-Datenvisualisierung	Nutzer sollen Wetterdaten in Echtzeit über das Dashboard abrufen können
Filtern & Sortieren	Daten sollen nach Zeiträumen und Parametern (Temperatur, Luftfeuchtigkeit etc.) filterbar sein
Benachrichtigungen bei Grenzwertüberschreitungen	Nutzer sollen visuelle Warnungen erhalten, wenn Temperatur- oder Feuchtigkeitsschwellen über- oder unterschritten werden
Sichere Anmeldung	Nur authentifizierte Nutzer sollen Zugriff auf die App haben
Responsive Design	Die App soll auf Desktops, Tablets und Smartphones nutzbar sein

3. **Anforderungen der Entwickler:**
    - Technische Anforderungen:

Tabelle 3: Technische Anforderungen an die App

Kategorie	**Anforderung**
Frameworks & Bibliotheken	Streamlit, Plotly, Pandas, Paho-MQTT, Requests, Psycopg2, SQLAlchemy, Dotenv
Programmiersprache	Python 3.x
Datenbank	PostgreSQL
Datenquelle	OpenWeatherMap API
Datenkommunikation	MQTT-Protokoll
Containerisierung	Docker
Versionskontrolle	GitHub

- Architektur des Datenprodukts
    1) Datenquelle: OpenWeatherMap API ruft Wetterdaten für eine bestimmte Stadt (München) ab.
    2) Datenübertragung: Die Daten werden über einen MQTT-Broker gesendet.
    3) Datenverarbeitung: Ein Python-Skript empfängt die Daten und speichert sie in einer PostgreSQL-Datenbank.
    4) Visualisierung: Ein Streamlit-Dashboard ruft die Daten aus der Datenbank ab und stellt sie in Diagrammen dar.
- Sicherheitsanforderungen:
    - Zugriffsbeschränkung durch Benutzer-Login.
    - Eingeschränkte Datenbankrechte für Nutzer.

4. **Spezifikation des Datenprodukts**
    - Datenmodell:

Tabelle 4: Beschreibung der Felder

Feldname	**Typ**	**Beschreibung**
timestamp	TIMESTAMP	Zeitpunkt der Messung
temperature	FLOAT	Temperatur in °C
humidity	FLOAT	Luftfeuchtigkeit in %
pressure	FLOAT	Luftdruck in hPa

- Datenfluss:
    1) Wetterdaten werden alle 10 Minuten von der API abgerufen.
    2) Die Daten werden als JSON-Nachricht über MQTT veröffentlicht.
    3) Ein Subscriber empfängt die Nachricht und speichert sie in PostgreSQL.
    4) Das Dashboard ruft die Daten ab und stellt sie grafisch dar.

5. **Fazit:**
   Das Datenprodukt erfüllt die Anforderungen sowohl für Nutzer (intuitive Visualisierung, Filtermöglichkeiten, Warnsysteme) als auch für Entwickler (skalierbare Architektur, sichere Speicherung). Die Nutzung von Docker und GitHub ermöglicht eine einfache Bereitstellung und Wartung.

2. Sicherheit und Architektur des Datenprodukts, Testplan Sicherheit

1. **Einführung:**
   Das Datenprodukt ist eine Anwendung zur Erfassung, Speicherung und Visualisierung von Wetterdaten. Dabei müssen sowohl die Datenintegrität, Verfügbarkeit und Zugriffssicherheit gewährleistet werden. Die Architektur besteht aus einer MQTT-Datenquelle, einer PostgreSQL-Datenbank und einer Webanwendung zur Datenvisualisierung.
2. **Sicherheitsmaßnahmen:**
   Um das Datenprodukt gegen unbefugten Zugriff, Manipulation und Datenverlust zu schützen, sind folgende Maßnahmen vorgesehen:

2.1 Netzwerk- und Kommunikationssicherheit:

Tabelle 5: Sicherheitsmaßnahmen für Netzwerk- und Kommunikationssicherheit

Sicherheitsmaßnahme	**Beschreibung**
MQTT-Authentifizierung	Der MQTT-Broker benötigt Authentifizierung, um nur autorisierte Clients zuzulassen

2.2 Zugriffskontrolle & Authentifizierung:

Tabelle 6: Sicherheitsmaßnahmen zur Zugriffskontrolle und Authentifizierung

Sicherheitsmaßnahme	**Beschreibung**
Benutzer-Login mit Passwörtern	Nutzer müssen sich anmelden, bevor sie auf das Dashboard zugreifen können
Sichere Passwortspeicherung	Anmeldedaten werden nicht sichtbar für User in einer externen Konfigurationsdatei gespeichert (.env-Datei)
VPN-Verbindung Erforderlich	Zugriff auf die Daten ist nur möglich, wenn der Nutzer mit dem VPN der Universität, oder direkt mit dem WLAN der Universität verbunden ist

2.2.1 Kontrolle:

Falscher Benutzername – richtiges Passwort:

Es war nicht möglich sich, mit einem falschen Benutzernamen anzumelden. Es kam eine Fehlermeldung, dass entweder das Passwort oder der Benutzername falsch ist.

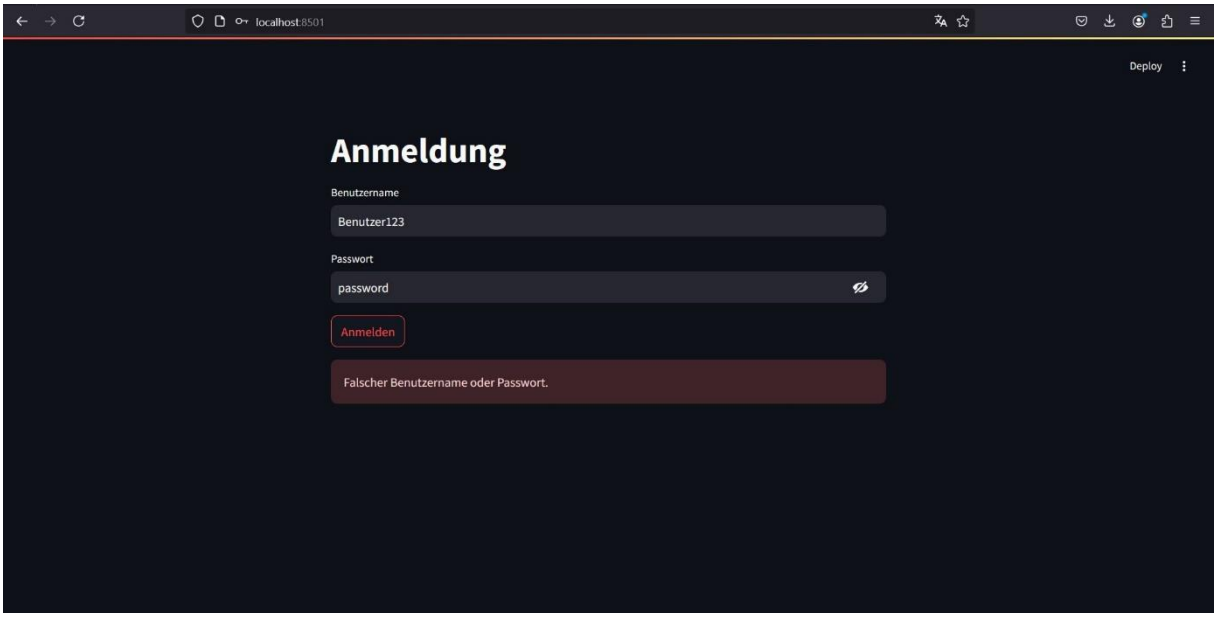


Abbildung 1: Fehlermeldung bei falschem Benutzernamen

Richtiger Benutzername – falsches Passwort:

Es war nicht möglich sich, mit einem falschen Passwort anzumelden. Es kam, wie bei dem falschen Benutzernamen, eine Fehlermeldung, dass entweder das Passwort oder der Benutzername falsch ist.

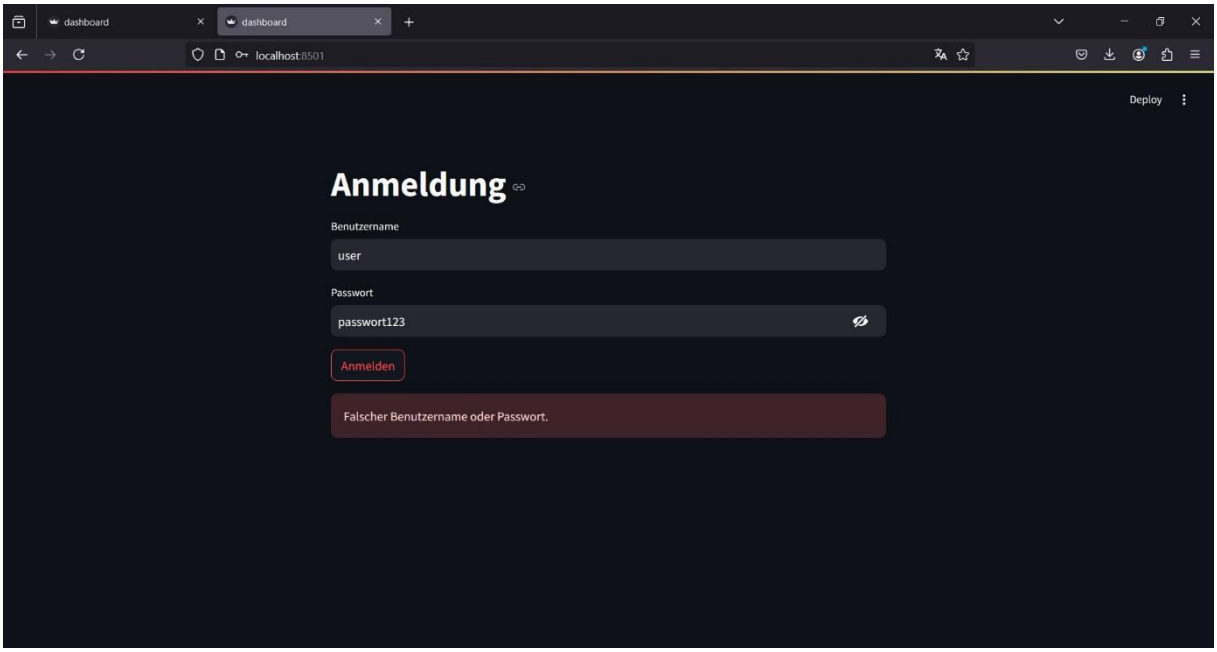


Abbildung 2: Fehlermeldung bei falschem Passwort

Ohne VPN-Verbindung / Ohne Verbindung zu Universitäts-WLAN:

Es war möglich auf die Website zuzugreifen und sich anzumelden. Es gab eine Bestätigung, dass die Anmeldung erfolgreich war. Nach Aktualisierung der Website folgte allerdings eine Fehlermeldung. Es war nicht möglich auf Daten zuzugreifen.

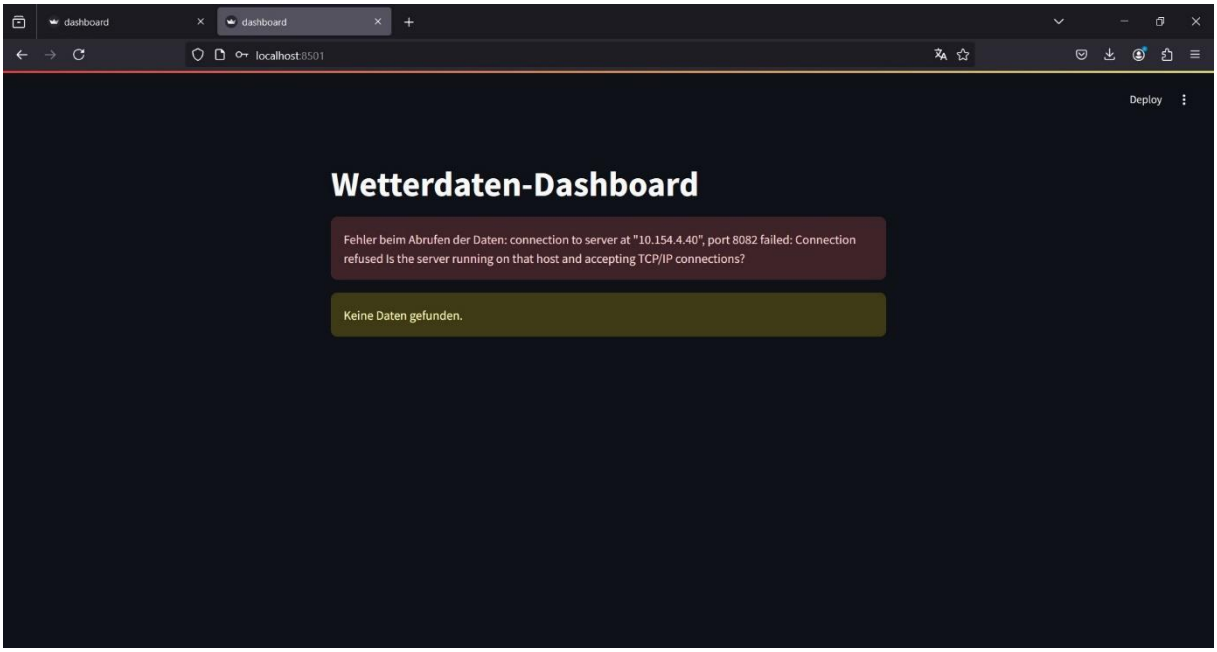


Abbildung 3: Fehlermeldung bei fehlender VPN-Verbindung

2.3 Datenbanksicherheit

Tabelle 7: Sicherheitsmaßnahmen für die Datenbanksicherheit

Sicherheitsmaßnahme	**Beschreibung**
SQL-Injection verhindern	Verwendung vorbereiteter SQL-Statements (prepared statements) zur Abfrage der Daten.
Leserechte für Standard-Nutzer	Nutzer können nur Daten abrufen, aber nicht verändern oder löschen.

2.3.1 Kontrolle:

SQL-Injection in Textfelder:

Es war nicht möglich eine SQL-Injection zu tätigen, da die Datum-Felder nicht die dazu benötigten Zeichen zulassen. In die Uhrzeit-Felder lassen sich keine Zeichen eintragen, da diese nur eine Eingabe über ein Dropdown-Menü ermöglichen.

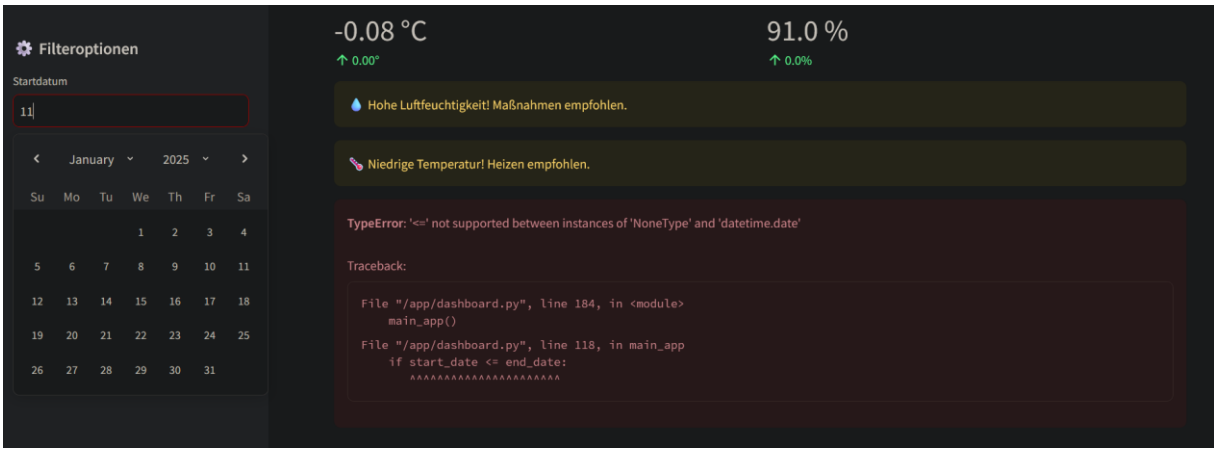


Abbildung 4: Fehlermeldung beim Versuch von SQL-Injection

Datenveränderung der eingeblendeten Datensätze:

Die Daten in der Tabelle konnten nicht verändert oder gelöscht werden, wenn man als Standart-Nutzer angemeldet ist. Es war nicht möglich Text in die Felder der Tabelle einzugeben.

3. Architektur des Datenprodukts

3.1 Überblick

Die Architektur besteht aus vier Hauptkomponenten:

1) Datenquelle: OpenWeatherMap API liefert Wetterdaten.
2) Datenübertragung: MQTT-Broker verarbeitet die Sensordaten.
3) Datenbank: PostgreSQL speichert die Daten langfristig.
4) Dashboard: Eine Streamlit-Web-App visualisiert die Daten und ermöglicht Benutzerinteraktionen.

3.2 Datenfluss

1) Das Python-Skript ruft die Wetterdaten von OpenWeatherMap ab.
2) Die Daten werden als MQTT-Nachricht veröffentlicht.
3) Ein MQTT-Subscriber empfängt die Daten und speichert sie in PostgreSQL.
4) Das Streamlit-Dashboard ruft die Daten aus der Datenbank ab und zeigt sie in Diagrammen an.

3.3 Systemdiagramm

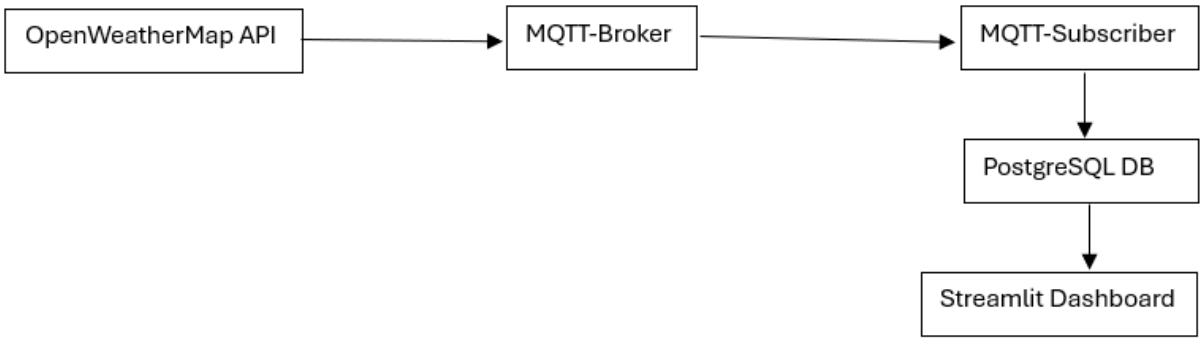


Abbildung 5: Systemdiagramm

4. **Testplan Sicherheit**
   Für die Absicherung des Systems sind verschiedene Tests notwendig:

Tabelle 8: Sicherheitstests und Ergebnisse

Testfall	Beschreibung	Erwartetes Ergebnis
SQL-Injection-Test	Versucht, SQL-Injection durch unsichere Eingaben zu testen	SQL-Abfrage wird nicht manipuliert
MQTT-Sicherheitstest	Ein nicht registrierter Client versucht, sich mit dem Broker zu verbinden	Verbindung wird abgelehnt

5. **Fazit**
   Das Sicherheitskonzept umfasst Netzwerksicherheit, Zugriffskontrolle, und Datenbanksicherheit. Der Testplan stellt sicher, dass bekannte Schwachstellen getestet und behoben werden. Damit wird die Vertraulichkeit, Integrität und Verfügbarkeit des Systems gewährleistet.

3. Berechtigungskonzept, Testplan Berechtigung des Datenprodukts

1. **Berechtigungskonzept**
   Das Berechtigungskonzept stellt sicher, dass verschiedene Benutzergruppen unterschiedliche Zugriffsrechte haben. Dadurch wird verhindert, dass unautorisierte Nutzer sicherheitskritische Änderungen an den Daten vornehmen können.

1.1 Benutzerrollen:

Tabelle 9: Berechtigungen des Nutzers

Rolle	Berechtigung	Beispielhafte Aktionen
Standard-Nutzer	Zugriff auf das Dashboard, kann Daten einsehen	Wetterdaten anzeigen, Filteroptionen nutzen

1.2 Implementierung der Berechtigungen

1) Benutzerverwaltung:
    - Nutzer müssen sich mit einem Benutzernamen und Passwort anmelden.
    - Passwörter werden nicht sichtbar in einer externen Konfigurationsdatei gespeichert (.env-Datei).
2) Berechtigungen in der Datenbank:
    - Admin hat Lese- und Schreibrechte.
    - Standard-Nutzer hat nur Leserechte.
    - Gast-Nutzer hat eingeschränkten Zugriff auf allgemeine Daten.

2. **Testplan Berechtigung:**
   Um sicherzustellen, dass das Berechtigungskonzept korrekt funktioniert, wird ein Testplan mit verschiedenen Szenarien erstellt.

Tabelle 10: Testplan Berechtigung und Ergebnisse

Testfall	**Beschreibung**	**Erwartetes Ergebnis**
Passwort-Hashing funktioniert	Ein gespeichertes Passwort wird überprüft	Passwort wird korrekt validiert und nicht als Klartext gespeichert
Sitzung bleibt aktiv	Nutzer bleibt nach Login aktiv und wird nicht unerwartet ausgeloggt	Sitzung bleibt bestehen, bis sie abläuft oder sich der Nutzer abmeldet

4. Programmablauf und Datenfluss, Design GUI

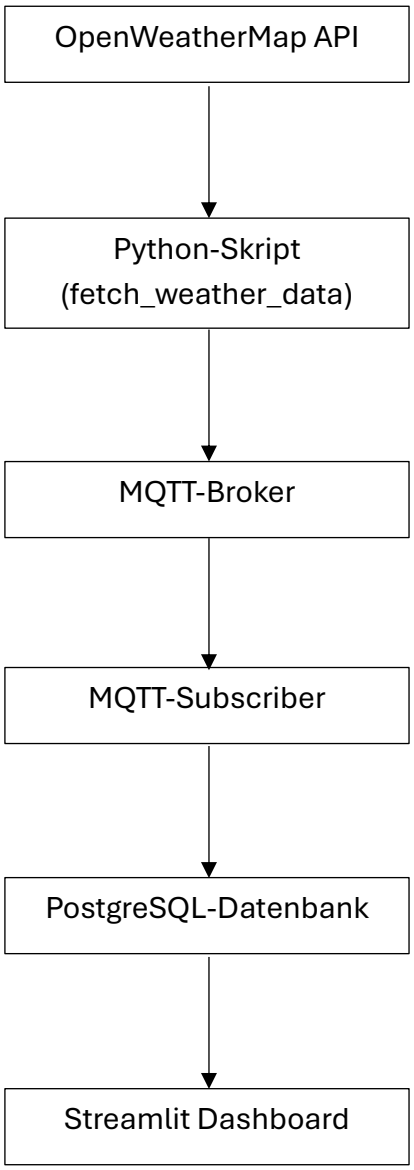
1. **Programmablauf**
   Das System besteht aus mehreren miteinander verbundenen Komponenten, die in einer definierten Reihenfolge arbeiten. Die folgenden Schritte beschreiben den Ablauf der Anwendung:
   1) Abruf von Wetterdaten:
      - Die OpenWeatherMap API wird in festen Intervallen (600 Sekunden) abgefragt.
      - Die Antwort enthält Temperatur, Luftfeuchtigkeit und Luftdruck.
   2) MQTT-Publishing:
      - Die Wetterdaten werden in JSON-Format umgewandelt.
      - Die Daten werden über den MQTT-Broker mit Authentifizierung (MQTT_USER, MQTT_PASSWORD) veröffentlicht.
   3) MQTT-Subscription & Speicherung in der Datenbank:
      - Der MQTT-Subscriber empfängt die Wetterdaten.

- Die Daten werden überprüft und dann in die PostgreSQL-Datenbank geschrieben.

4) Datenbankabfrage & Streamlit-Dashboard:
   - Die Web-App authentifiziert Benutzer.
   - Daten werden aus der PostgreSQL-Datenbank abgerufen und visualisiert.
   - Nutzer können die Daten filtern und analysieren.

2. Datenfluss

Der Datenfluss beschreibt, wie die Daten durch das System bewegt und verarbeitet werden.



3. Gui-Design (Streamlit)

Das Dashboard ist die Hauptschnittstelle für Benutzer und bietet eine intuitive und interaktive Visualisierung.

3.1 Designübersicht

- Login-Bereich: Nur authentifizierte Benutzer dürfen auf die App zugreifen.
- Hauptbereich: Zeigt die wichtigsten Messwerte mit Warnhinweisen.
- Filter-Optionen: Auswahl von Zeiträumen und spezifischen Parametern.
- Diagramme: Zeigt Temperatur-, Luftfeuchtigkeits- und Druckverläufe.
- Tabellarische Ansicht: Zeigt die Rohdaten an.

3.2 Gui-Komponenten

Tabelle 11: Komponenten des Graphical User Interface

Komponente	**Beschreibung**
Login-Bereich	Benutzer müssen sich anmelden, bevor sie Daten sehen.
Hauptdashboard	Anzeige der aktuellen Wetterwerte mit Warnmeldungen.
Seitliches Menü	Ermöglicht das Filtern nach Zeitraum und Sensordaten.
Diagramme (Plotly)	Zeigt Temperatur-, Luftfeuchtigkeits- und Druckverläufe.
Tabellarische Daten	Ermöglicht eine detaillierte Ansicht der Messwerte.

3.3 Visualisierungsmethoden

- st.metric() für aktuelle Werte und Abweichungen.
- px.line() (Plotly) für Liniendiagramme mit Zeitverlauf.
- st.dataframe() für tabellarische Darstellungen.

4. **Fazit**
   Das System ermöglicht eine automatisierte, sichere und visuell ansprechende Wetterdatenerfassung. Die GUI bietet intuitive Interaktionen und ermöglicht datengetriebene Entscheidungen.

5. Programmcode App mit streamlit; Testplan App

1. **Programmcode mit Streamlit**
   - Siehe GitHub Repository für die Anwendung
2. **Testplan App**

Tabelle 12: Testplan der App

Testfall	**Beschreibung**	**Erwartetes Ergebnis**
Login-Test (korrekte Daten)	Nutzer meldet sich mit korrekten Zugangsdaten an	Zugang wird gewährt

Login-Test (falsche Daten)	Nutzer gibt falsches Passwort ein	Fehlermeldung erscheint
Datenbankverbindung	App greift auf PostgreSQL zu	Daten werden korrekt geladen
MQTT-Datenfluss	Sensordaten werden via MQTT empfangen	Daten erscheinen in der App
Filterung nach Zeitraum	Nutzer wählt einen bestimmten Zeitraum aus	Diagramme und Tabelle zeigen die gefilterten Daten
Visualisierung der Werte	Diagramme für Temperatur, Feuchtigkeit, Druck werden angezeigt	Korrekte Werte und Trends sind sichtbar
Warnhinweise	Temperatur oder Luftfeuchtigkeit überschreiten Schwellwerte	Warnmeldung erscheint in der UI

6. Handbuch für Benutzer und Entwickler

1. **Benutzerhandbuch**

1.1 Einführung

Das Wetter-Dashboard ermöglicht die Visualisierung von Wetterdaten wie Temperatur, Luftfeuchtigkeit und Luftdruck. Die Daten werden in Echtzeit über MQTT empfangen und in einer PostgreSQL-Datenbank gespeichert.

1.2 Starten der Anwendung

Schritt 1: Anmeldung

1) Öffne das Dashboard über den Webbrowser (z. B. http://10.154.4.40:8088/ oder falls die App lokal läuft, http://localhost:8088/).
2) Melde dich mit deinem Benutzernamen und Passwort an. (Default-User: BN=user123; PW=password)
3) Falls das Passwort falsch ist, erscheint eine Fehlermeldung.

Schritt 2: Wetterdaten anzeigen

- Nach erfolgreicher Anmeldung siehst du aktuelle Wetterdaten.
- Die Hauptanzeige zeigt:
  - Temperatur
  - Luftfeuchtigkeit
  - Luftdruck

Schritt 3: Filterung und Analyse

- Im Seitenmenü kannst du:
  - Ein Start- und Enddatum wählen.
  - Nach spezifischen Zeiträumen filtern.

- Diagramme und tabellarische Daten anzeigen.

Schritt 4: Warnhinweise verstehen

- Falls bestimmte Schwellenwerte überschritten werden:
  - Temperatur zu hoch/niedrig → Empfehlung zu Heizen oder Lüften
  - Luftfeuchtigkeit zu hoch → Empfehlung zur Entlüftung

Schritt 5: Beenden der Anwendung

- Um sich abzumelden, schließe einfach das Fenster.

2. Entwicklerhandbuch

2.1 Systemübersicht

Die Anwendung besteht aus mehreren Komponenten:

1) Python-Skript (database.py)
   - Ruft Wetterdaten von der API ab
   - Sendet die Daten über MQTT
   - Speichert die Daten in PostgreSQL
2) Streamlit Dashboard (dashboard.py)
   - Liest die Wetterdaten aus der Datenbank
   - Stellt sie grafisch dar
   - Ermöglicht Filterung
3) MQTT-Broker
   - Verteilt die Wetterdaten an andere Komponenten
4) PostgreSQL-Datenbank
   - Speichert die Sensordaten

7. Quellen

- https://docs.streamlit.io/
- https://www.datacamp.com/de/tutorial/tutorial-postgresql-python?utm_source=chatgpt.com
- https://www.geeksforgeeks.org/python-os-getenv-method/
- https://www.sqlalchemy.org/
- https://home.openweathermap.org/
- https://pypi.org/project/requests/
- https://lukianovihor.medium.com/python-environment-variables-using-dotenv-library-71529ad0e9c3
- https://github.com/KIrony404/DaBivSim

- https://chatgpt.com/
  - Prompts:
    - Wie benutze ich getenv aus der bibliothek dotenv korrekt?
    - Wie stelle ich mit SQLALchemy eine dbverbindung in python her?
    - Wie kann ich in Streamlit einen Live-Update-Modus aktivieren, der die Daten alle 10 Sekunden neu lädt?
    - Wie kann ich mein System in Docker-Containern bereitstellen, um es auf einem Server laufen zu lassen?
    - Wie kann ich Fehler in MQTT-Nachrichten besser behandeln, um fehlerhafte Daten nicht in die Datenbank zu schreiben?