Software Engineering Projekt

Raka Adita, Abdulaziz al-Surabi, Felix Hartmann, Niklas Denz, Mehmet Karaca

Projektübersicht

Unser FARM2-Projekt adressiert diese Herausforderungen durch die Entwicklung einer spezialisierten Webanwendung für die automatisierte Instagram-Post-Planung. Das System basiert auf einer durchdachten Drei-Schichten-Architektur:

Frontend-Ebene

- Ein intuitives Dashboard zur Post-Verwaltung
- Visueller Kalender für die Posting-Planung
- Benutzerfreundliches Upload-System für Medieninhalte
- Vorschaufunktion für geplante Posts
- Flexibles Hashtag-Verwaltungssystem

Backend-Ebene

- Verarbeitung und Speicherung von Mediendateien
- Authentifizierung mit der Instagram-API
- Zuverlässiges Scheduling von Posts
- Verwaltung der Datenbankoperationen
- Fehlerbehandlung und Logging

Datenbank-Ebene

- Speicherung von Metadaten zu geplanten Posts, Scheduling-Informationen, Status-Updates und Logs
- PostgreSQL als robuste Basis
- Sichere Datenbankverbindungen mit Prepared Statements
- Asynchrone Prozesse für das Post-Scheduling

Technische Spezifikation und Implementierungsdetails

Systemarchitektur Die Umsetzung erfolgt mit einem FARM-Stack (Flask, React, PostgreSQL) und folgenden Komponenten:

1. **Frontend-Technologien:**

- React 18
- Material-UI
- Axios
- React Calendar
- Modernes CSS (Flexbox, Grid)

2. **Backend-Technologien:**

- Python 3.9+
- Flask
- SQLAlchemy als ORM
- APScheduler für Task-Management
- Instagrapi für Instagram-Integration

3. **Datenbank-Schema**

```
CREATE TABLE scheduled_posts (
    id SERIAL PRIMARY KEY,
    post_type VARCHAR(50) NOT NULL,
    caption TEXT NOT NULL,
    hashtags TEXT NOT NULL,
    media_path TEXT NOT NULL,
    scheduled_time TIMESTAMP NOT NULL,
    status VARCHAR(20) DEFAULT 'pending',
    error_message TEXT
);
```

Sicherheitskonzept

- Speicherung von Zugangsdaten in Umgebungsvariablen
- CORS-Konfiguration
- Validierung aller Benutzereingaben
- Sichere Dateispeicherung
- Automatische Bereinigung temporärer Dateien
- Fehlerprotokollierung

Skalierbarkeit und Performance

- Asynchrone Verarbeitung von Upload-Prozessen
- Datenbank-Indexing
- Caching-Strategien
- Lazy Loading von Medieninhalten
- Optimierte Abfragen
- Containerisierung und Load Balancing

Frontend

Die Frontend-Implementierung bietet eine moderne, benutzerfreundliche Weboberfläche zum Planen und Verwalten von Instagram-Posts.

Technologische Grundlagen

- React 18 (Hooks, reaktives UI)
- React Router (Navigation)
- Axios (HTTP-Kommunikation)

- React Calendar (Kalenderfunktionalität)
- CSS3 mit Flexbox und Grid (responsives Layout)

Komponenten und Funktionalität

Hauptkomponente (App.js)

- Formular für Hochladen/Planen von Posts
- Zustände: Typ (Bild/Video), Caption, Hashtags, Mediendatei, Zeitpunkt
- Drag-and-Drop-Funktion für Mediendateien
- Validierung und Feedback

Kalenderkomponente (ScheduledPosts.js)

- Kalenderansicht aller geplanten Posts
- Hervorhebung von Tagen mit Posts
- Detaillierte Ansicht der Posts eines ausgewählten Tages
- Löschfunktion für geplante Posts
- Karussell-Ansicht bei mehreren Posts pro Tag

Benutzeroberfläche und Interaktion

- Modernes, minimalistisches UX-Design
- Konsistentes Farbschema
- Responsive Layouts
- Sofortiges Feedback bei Interaktionen
- Tooltips, Hilfestellungen und Bestätigungsdialoge

Backend-Kommunikation

- /upload (POST, multipart/form-data)
- /scheduled_posts (GET, liefert JSON)
- /delete_scheduled_post/{id} (DELETE)

Fehlerbehandlung und Validierung

- Clientseitige Validierung
- Dateityp- und -größenprüfung
- Umgang mit Netzwerkfehlern
- Benutzerfreundliche Fehlermeldungen

Performance und Optimierung

- Lazy Loading
- Lokale Vorschau (Blob URLs)
- Effizientes State Management
- Minimierung von API-Aufrufen und lokalem Caching

Wartbarkeit und Weiterentwicklung

- Klare Komponentenstruktur
- Wiederverwendbare Funktionen/Komponenten
- Kommentierter Code und einheitliche Coding-Standards
- Erweiterbar (z. B. für weitere Social-Media-Plattformen)

Backend

Das Backend übernimmt die Verarbeitung von Medien, die Instagram-API-Anbindung sowie das zeitgesteuerte Veröffentlichen von Posts.

Technologische Grundlagen

- Flask (Web-Framework)
- PostgreSQL (relationale Datenbank)
- SQLAlchemy (ORM)
- APScheduler (Scheduler für zeitgesteuerte Tasks)
- Instagrapi (Instagram-Integration)

Systemarchitektur

API-Layer

- /upload: POST-Endpoint für Mediendateien + Metadaten
- /scheduled_posts: GET-Endpoint für geplante Posts
- /delete_scheduled_post/: DELETE-Endpoint zum Entfernen

Datenbank-Layer

- scheduled_posts-Tabelle verwaltet Metadaten, Status, Fehlerlogs etc

```
CREATE TABLE scheduled_posts (
    id SERIAL PRIMARY KEY,
    post_type VARCHAR(50) NOT NULL,
    caption TEXT NOT NULL,
    hashtags TEXT NOT NULL,
    media_path TEXT NOT NULL,
    scheduled_time TIMESTAMP NOT NULL,
    status VARCHAR(20) DEFAULT 'pending'
);
```

Scheduler-Komponente

- Überprüfung geplanter Posts alle 2 Minuten
- Automatische Veröffentlichung zum geplanten Zeitpunkt
- Fehlerbehandlung, Statusaktualisierung, Bereinigung temporärer Dateien

Implementierungsdetails

Medienverarbeitung

- Validierung (Dateitypen, Größe, Format)
- Speicherung (eindeutige Dateinamen, temporäre Ordner)
- Bereinigung nach erfolgreicher Verarbeitung

Instagram-Integration

- Sichere Authentifizierung (Umgebungsvariablen)
- Unterstützung von Bild- und Videouploads
- Fehlerbehandlung bei Rate Limits und API-Limits
- Wiederholungsversuche bei temporären Fehlern

Fehlerbehandlung

- Mehrstufig: Eingabevalidierung, Logging, HTTP-Statuscodes, Scheduler-Fehlerprotokollierung

Sicherheitsaspekte

- Sichere Speicherung von Zugangsdaten (z. B. in .env)
- Verschlüsselte DB-Verbindung
- CORS-Regeln
- Rate-Limits und validierte Dateitypen

Performance-Optimierung

- Indexierung der Datenbank
- Prepared Statements und Connection Pooling
- Asynchrone Upload-Prozesse
- Caching bei Bedarf

Wartung und Monitoring

- Logging (Fehler, Aktivität, Performance)
- Health-Checks (Scheduler, DB, Instagram API)
- Automatische Bereinigung alter Daten
- Erweiterbar für zusätzliche Funktionen (Analytics, weitere Plattformen)

Systemarchitektur und Technologie-Stack

- Flask 3.1.0, PostgreSQL 14.0+, APScheduler 3.11.0, Instagrapi 2.1.3
- Python 3.9+
- Weitere Bibliotheken: Flask-CORS, Psycopg2-binary, python-dotenv, Pillow, Moviepy

Datenbank

1. Datenbankarchitektur

1.1 Übersicht

- PostgreSQL als DBMS
- Speichert alle anstehenden und bereits veröffentlichte Instagram-Posts
- Vorteile: ACID-Konformität, Transaktionssicherheit, Indexierung, JSON-Unterstützung

1.2 Systemanforderungen

- PostgreSQL 14.0 oder höher
- Mind. 2 GB RAM, ~10 GB Speicherplatz empfohlen
- Ausreichend Platz für Backups und Logs

2. Datenbankschema

2.1 Tabelle scheduled_posts

- id: Primärschlüssel (SERIAL)
- post_type: (z. B. image oder video)
- caption: Beschreibung des Posts
- hashtags: Hashtags
- media_path: Pfad zur Mediendatei
- scheduled_time: Zeitpunkt der Veröffentlichung
- status: Status (pending, uploaded, etc.)

```
CREATE TABLE scheduled_posts (
    id SERIAL PRIMARY KEY,
    post_type VARCHAR(50) NOT NULL,
    caption TEXT NOT NULL,
    hashtags TEXT NOT NULL,
    media_path TEXT NOT NULL,
    scheduled_time TIMESTAMP NOT NULL,
    status VARCHAR(20) DEFAULT 'pending',
);
```

3. Wichtige Datenbankoperationen

3.1 Einfügen geplanter Posts

```
cursor.execute(
    "INSERT INTO scheduled_posts (post_type, caption, hashtags,
media_path, scheduled_time) VALUES (%s, %s, %s, %s, %s)",
    (post_type, caption, hashtags, media_path, scheduled_time)
)
```

3.2 Abrufen und Verarbeiten

```
SELECT * FROM scheduled_posts
WHERE scheduled_time <= NOW() AND status = 'pending';
```

Bei Erfolg -> Status auf uploaded setzen.

3.3 Löschen

```
cursor.execute("DELETE FROM scheduled_posts WHERE id = %s", (post_id,))
```

Instagrapi-Integration

1. Übersicht

Die Integration erfolgt über Instagrapi (v2.1.3):

- Zuverlässig, umfangreich, aktive Community
- Fehlerbehandlung und dokumentierte Schnittstellen

2. Installation und Konfiguration

```
pip install instagrapi==2.1.3
```

3. Post-Management

```
def upload_post():
    try:
        post_type = request.form.get("post_type")
        caption = request.form.get("caption", "")
        hashtags = request.form.get("hashtags", "")
        file = request.files.get("media")
        scheduled_time = request.form.get("scheduled_time")

        if not file:
            return jsonify({"status": "error", "message": "No media file
provided"}), 400
```

Unit-Tests

Überblick über die Unit-Tests

`unittest_validateImage.py`

- Tests für die Bildverarbeitungs- und Validierungslogik

`test_integration.py`

- Überprüft die Interaktion mehrerer Komponenten (z. B. Upload-Endpoint)

behave

- Tests von der Anwendung durch Selenium-Test