# Projekt-Dokumentation: Implementierung einer webbasierten Zeitrapport-App

## 1. Einleitung

In dieser Dokumentation wird Schritt für Schritt gezeigt, wie die zuvor spezifizierte Applikation mit Python umgesetzt wird. Erklärt werden Einrichtung, Code-Beispiele und Deployment.

## 2. Voraussetzungen

- Python 3.8+

- pip und virtuelle Umgebung (venv)

- Clockify API-Token

- Microsoft Azure Account mit Berechtigung zum Erstellen eines App Service

## 3. Projektstruktur

projekt/  
├── app/  
│ ├── \_\_init\_\_.py  
│ ├── routes.py  
│ ├── services.py  
│ └── templates/  
│ └── report\_template.html  
├── static/  
│ ├── index.html  
│ └── script.js  
├── tests/  
│ ├── test\_services.py  
│ └── test\_routes.py  
├── .env  
├── requirements.txt  
└── run.py

## 4. Einrichtung der Umgebung

- python -m venv venv

- source venv/bin/activate (Linux/macOS) oder venv\Scripts\activate (Windows)

- pip install -r requirements.txt

## 5. Clockify API Client (app/services.py)

import os  
import requests  
  
class ClockifyClient:  
 BASE\_URL = "https://api.clockify.me/api/v1"  
  
 def \_\_init\_\_(self, api\_token: str):  
 self.headers = {  
 "X-Api-Key": api\_token,  
 "Content-Type": "application/json"  
 }  
  
 def get\_time\_entries(self, workspace\_id: str, project\_id: str, start: str, end: str):  
 url = f"{self.BASE\_URL}/workspaces/{workspace\_id}/projects/{project\_id}/time-entries"  
 params = {  
 "start": start,  
 "end": end,  
 "hydrated": "true"  
 }  
 response = requests.get(url, headers=self.headers, params=params)  
 response.raise\_for\_status()  
 return response.json()

## 6. Flask Backend

# run.py  
from app import create\_app  
  
app = create\_app()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(debug=True)  
  
# app/\_\_init\_\_.py  
from flask import Flask  
from .routes import bp  
  
def create\_app():  
 app = Flask(\_\_name\_\_)  
 app.config.from\_pyfile('../.env')  
 app.register\_blueprint(bp)  
 return app  
  
# app/routes.py  
from flask import Blueprint, request, send\_file, jsonify  
from .services import ClockifyClient  
from io import BytesIO  
from .pdf\_generator import generate\_pdf  
  
bp = Blueprint('main', \_\_name\_\_)  
  
@bp.route('/generate-pdf', methods=['POST'])  
def generate\_report():  
 data = request.json  
 client = ClockifyClient(os.getenv('CLOCKIFY\_TOKEN'))  
 entries = client.get\_time\_entries(data['workspace\_id'], data['project\_id'], data['start'], data['end'])  
 pdf\_bytes = generate\_pdf(entries)  
 return send\_file(BytesIO(pdf\_bytes), attachment\_filename='report.pdf', as\_attachment=True)

## 7. PDF-Generator (app/pdf\_generator.py)

from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph  
from reportlab.lib.pagesizes import A4  
from reportlab.lib import colors  
from reportlab.lib.styles import getSampleStyleSheet  
  
def generate\_pdf(entries):  
 buffer = BytesIO()  
 doc = SimpleDocTemplate(buffer, pagesize=A4)  
 styles = getSampleStyleSheet()  
 elements = []  
  
 # Tabelle vorbereiten  
 data = [["Datum", "Projekt", "Beschreibung", "Stunden"]]  
 for e in entries:  
 data.append([e['timeInterval']['start'][:10], e['project']['name'], e['description'], e['timeInterval']['duration']])  
  
 table = Table(data)  
 table.setStyle(TableStyle([  
 ('BACKGROUND', (0,0), (-1,0), colors.grey),  
 ('GRID', (0,0), (-1,-1), 1, colors.black),  
 ]))  
 elements.append(table)  
 doc.build(elements)  
 return buffer.getvalue()

## 8. Frontend (static/index.html & script.js)

<!DOCTYPE html>  
<html lang="de">  
<head>  
<meta charset="UTF-8">  
<title>Zeitrapport</title>  
</head>  
<body>  
 <h1>Zeitrapport Generator</h1>  
 <label for="workspace">Workspace-ID:</label><input id="workspace">  
 <label for="project">Projekt-ID:</label><input id="project">  
 <label for="start">Startdatum:</label><input type="date" id="start">  
 <label for="end">Enddatum:</label><input type="date" id="end">  
 <button id="generate">PDF generieren</button>  
  
 <script src="script.js"></script>  
</body>  
</html>

// static/script.js  
document.getElementById('generate').addEventListener('click', async () => {  
 const body = {  
 workspace\_id: document.getElementById('workspace').value,  
 project\_id: document.getElementById('project').value,  
 start: document.getElementById('start').value,  
 end: document.getElementById('end').value  
 };  
 const res = await fetch('/generate-pdf', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify(body)  
 });  
 const blob = await res.blob();  
 const url = URL.createObjectURL(blob);  
 window.open(url, '\_blank');  
});

## 9. Deployment auf Microsoft Azure

- Azure CLI installieren und anmelden: az login

- Ressourcengruppe erstellen: az group create --name MyResourceGroup --location westeurope

- App Service erstellen: az appservice plan create --name MyPlan --resource-group MyResourceGroup --sku B1

- Web App erstellen: az webapp create --resource-group MyResourceGroup --plan MyPlan --name MyAppName --runtime 'PYTHON|3.8'

- Deployment via GitHub Actions oder az webapp up

- App-Einstellungen: CLOCKIFY\_TOKEN als Umgebungsvariable setzen

## 10. Tests

# tests/test\_services.py  
import pytest  
from app.services import ClockifyClient  
from unittest.mock import patch  
  
def test\_get\_time\_entries\_mocked():  
 token = "fake\_token"  
 client = ClockifyClient(token)  
 with patch('app.services.requests.get') as mock\_get:  
 mock\_get.return\_value.json.return\_value = [{"id": "1"}]  
 mock\_get.return\_value.raise\_for\_status = lambda: None  
 entries = client.get\_time\_entries("ws", "prj", "2025-01-01", "2025-01-31")  
 assert entries == [{"id": "1"}]