# Anforderungsliste und Anleitung zur Entwicklung einer webbasierten App

## 1. Technische Anforderungsliste

### 1.1 Funktionale Anforderungen

- Auswahl von Kunde/Projekt über eine Dropdown-Liste oder Suchfeld.

- Auswahl eines Zeitraums (Start- und Enddatum) mittels Kalender-Picker.

- Abruf von Zeiteinträgen über die Clockify API anhand der gewählten Parameter.

- Generierung einer PDF-Datei mit stundenweiser Auflistung der Zeiteinträge.

- Downloadmöglichkeit bzw. Anzeige der generierten PDF im Browser.

### 1.2 Nicht-funktionale Anforderungen

- Backend: Python Flask Framework.

- Frontend: HTML5, CSS3, JavaScript (vanilla oder Framework nach Wahl).

- PDF-Generierung: Bibliothek z.B. ReportLab oder WeasyPrint.

- API-Kommunikation: REST-Client für Clockify (JSON).

- Hosting: Microsoft Azure (App Service, Blob Storage ggf.).

- Sicherheit: HTTPS, sichere Speicherung des Clockify API-Tokens.

- Skalierbarkeit: Deployment via Azure DevOps oder GitHub Actions.

- Dokumentation: README mit Setup- und Deploy-Anleitung.

## 2. Detaillierte Anleitung zum Projektabschluss

### 2.1 Vorbereitung

- Azure-Account erstellen (falls nicht vorhanden).

- Clockify-Konto und API-Token besorgen.

- Lokale Entwicklungsumgebung einrichten: Python 3.8+, pip, Virtualenv.

### 2.2 Backend mit Flask aufsetzen

- Virtuelle Umgebung erstellen: python -m venv venv

- Abhängigkeiten installieren: Flask, requests, python-dotenv, ReportLab/WeasyPrint

- Projektstruktur anlegen: /app mit \_\_init\_\_.py, routes.py, services.py, models.py

- Umgebungsvariablen konfigurieren (.env): CLOCKIFY\_TOKEN, AZURE\_CREDENTIALS

- API-Service implementieren: ClockifyClient in services.py mit Methoden get\_time\_entries(start, end, project\_id)

### 2.3 PDF-Generierung implementieren

- PDF-Template in HTML/CSS entwerfen (für WeasyPrint) oder direkt ReportLab-Code schreiben.

- PDF-Service in services.py: generate\_pdf(data) → BytesIO.

- Route in routes.py: /generate-pdf, nimmt Parameter entgegen und liefert PDF-Response.

### 2.4 Frontend entwickeln

- HTML-Formular mit Dropdowns (Kunde/Projekt) und Datumspickern erstellen.

- Fetch-API in JavaScript: POST/GET an Flask-Endpunkte.

- PDF im Browser anzeigen: window.open oder Download-Link.

- Fehlerbehandlung und Ladeindikator ergänzen.

### 2.5 Testing und lokale Ausführung

- Unit-Tests für Clockify-Client und PDF-Generator schreiben (pytest).

- Integrationstest: komplette PDF-Erzeugung lokal testen.

- Frontend-Tests manuell: verschiedene Eingaben durchspielen.

### 2.6 Deployment auf Microsoft Azure

- Azure App Service anlegen (Linux/Python).

- Azure CLI installieren und Login: az login.

- Deployment-Slot oder GitHub Actions einrichten.

- Automatisches Deployment aus dem Git-Repo konfigurieren.

- Umgebungsvariablen in Azure App Service setzen.

- App starten und Endpunkte testen.

### 2.7 Wartung und Erweiterung

- Dokumentation aktualisieren (README, API-Dokument).

- Logging und Monitoring (Azure Application Insights).

- Fehlerdiagnose: Logs auswerten.

- Feature-Requests planen: z.B. Export in CSV, E-Mail-Versand der PDF.