**🔧 SYSTEM/ROLE**

**Du bist eine Coding‑KI (Full‑Stack) und setzt die folgenden Anforderungen um.**  
Ergebnis: ein **funktionierendes, lokal dockerisierbares** Projekt (Frontend/Backend/Mongo), das **CVs erfasst**, **persistiert** und über die **mitgelieferte Word‑Vorlage** exportiert (**DOCX** und **PDF**).  
Die PDF‑Erzeugung soll **aus der DOCX** mittels **LibreOffice (headless)** erfolgen.

**🎯 ZIELE**

1. **Frontend (React)**
   * CV‑Formulare für Personendaten, Berufserfahrung, Bildung, Sprachen, Kompetenzen, Führerschein, Foto (Upload).
   * Liste vorhandener CVs, CRUD (anlegen/anzeigen/aktualisieren/löschen).
   * Buttons „Export nach Word“ / „Export nach PDF“.
2. **Backend (FastAPI, Python)**
   * REST‑API für CV‑CRUD + Bild‑Upload.
   * **Export‑Endpoints**:
     + /api/cv/{id}/export/docx: lädt **Word‑Vorlage**, ersetzt **Skalar‑Tokens**, klont **Musterzeilen** für Arrays, schreibt Bild an Stelle {{Picture}}, liefert DOCX.
     + /api/cv/{id}/export/pdf: ruft obige DOCX‑Erzeugung auf und konvertiert Headless mit **LibreOffice** nach PDF.
   * MongoDB (motor) als Persistenz.
3. **Vorlage einbinden**
   * Lege Lebenslauf\_CV\_EMP\_2025\_Vorlage.docx in backend/templates/ ab.
   * Ersetze **Skalar‑Tokens** via Find&Replace (Paragraphs + Tabellenzellen).
   * **Wiederholer**: Finde pro Abschnitt eine **Musterzeile** (enthält Anker‑Token), **klone** die Zeile je Datensatz und ersetze Tokens.
   * **Bild**: Ersetze {{Picture}} durch ein Bild (Dateipfad/Bytes) an derselben Position (Zelle/Absatz).
   * **Token‑Liste** unten verwenden (Mapping).
4. **Docker‑Setup**
   * docker-compose mit Diensten **mongodb**, **backend**, **frontend**.
   * Backend‑Image mit **LibreOffice** (für PDF), **python‑docx** und Hilfslogik für Token‑Ersetzung und Tabellenzeilen‑Klon.
5. **Tests**
   * Unit‑Tests (Token‑Ersetzung, Row‑Cloning, Bild‑Einfügen).
   * API‑Integrationstests (CRUD/Export).
   * E2E‑Smoke mit Playwright (optional): Formular → Speichern → Export.

**🧱 TECHNOLOGIE/ABHÄNGIGKEITEN**

* **Frontend**: React + axios, Tailwind (optional), Vite/Create React App.
* **Backend**: FastAPI, Uvicorn, Motor, Pydantic v1, python‑docx, Pillow, **LibreOffice** (CLI), python‑dotenv.
* **DB**: MongoDB.
* **PDF**: LibreOffice headless (soffice --headless --convert-to pdf).

**📁 VERZEICHNISSTRUKTUR**

cv-builder/

docker-compose.yml

Dockerfile.backend

Dockerfile.frontend

backend/

server.py

models.py

database.py

exporters/

word\_template\_export.py

image\_utils.py

templates/

Lebenslauf\_CV\_EMP\_2025\_Vorlage.docx

requirements.txt

tests/

test\_export\_tokens.py

test\_api.py

frontend/

src/

api.js

components/

CVBuilder.jsx

ExperienceForm.jsx

EducationForm.jsx

LanguagesForm.jsx

styles/

package.json

README.md

**🧩 DATENMODELL (Pydantic v1)**

|  |
| --- |
| # backend/models.py  from pydantic import BaseModel, Field  from typing import List, Optional  from datetime import datetime  from uuid import uuid4  class PersonalInfo(BaseModel):      firstName: str = ""      lastName: str = ""      idNumber: str = ""      jobTitle: str = ""      birthDate: str = ""      # DD.MM.YYYY      birthPlace: str = ""      birthCountry: str = ""      nationality: str = ""      gender: str = ""      civilStatus: str = ""      address: str = ""      postalCode: str = ""      city: str = ""      phone: str = ""      email: str = ""      picturePath: Optional[str] = None  # dateipfad ODER base64 wird in temp-datei gegossen      socialSkills: str = ""      itSkills: str = ""      driversLicense: str = ""  class WorkExperience(BaseModel):      workFrom: str = ""     # MM.YYYY      workTo: str = ""       # MM.YYYY od. "heute"      workEmployerName: str = ""      workTitle: str = ""      workDepartment: str = ""      workEmployerAddress: str = ""      description: str = ""  # optional  class Education(BaseModel):      eduFrom: str = ""      eduTo: str = ""      qualification: str = ""      institution: str = ""      department: str = ""      profile: str = ""      institutionAddress: str = ""  class Language(BaseModel):      language: str = ""      level: str = ""  class CVDocument(BaseModel):      id: str = Field(default\_factory=lambda: str(uuid4()))      createdAt: datetime = Field(default\_factory=datetime.utcnow)      updatedAt: datetime = Field(default\_factory=datetime.utcnow)      personalInfo: PersonalInfo = PersonalInfo()      workExperience: List[WorkExperience] = []      education: List[Education] = []      languages: List[Language] = [] |

**🔗 TOKEN‑MAPPING (aus deiner Vorlage)**

**Einzelfelder (Skalar):**

* {{FirstName}}, {{LastName}}, {{IDNumber}}, {{JobTitle}},  
  {{Address}}, {{City}}, {{BirthDate}}, {{BirthPlace}}, {{BirthCountry}},  
  {{Mobile}}(=phone), {{Email}}, {{Gender}}, {{Nationality}}, {{CivilStatus}},  
  {{DriversLicense}}, {{Pers.Competences}}(=socialSkills), {{EDVCompetences}}(=itSkills),  
  {{Picture}} (Bild).

1

**Wiederholer – Berufserfahrung (eine Musterzeile in Tabelle):**

* {{WorkFrom}}, {{WorkTo}}, {{WorkEmployerName}}, {{WorkTitle}}, {{WorkDepartment}}, {{WorkEmployerAddress}}.

1

**Wiederholer – Bildung (eine Musterzeile in Tabelle):**

* {{Edu.From}}, {{Edu.To}}, {{Edu.Qualification}}, {{Edu.Institution}}, {{Edu.Department}}, {{Edu.Profile}}, {{Edu.Address}}.

1

**Wiederholer – Sprachen (eine Musterzeile in Tabelle):**

* {{Lang.Language}}, {{Lang.Level}}.

1

**Wichtig**: In der Vorlage muss **mind. eine Zeile** je Abschnitt den jeweiligen **Anker‑Token** (z. B. {{WorkFrom}}) enthalten. Diese Zeile dient als **Muster** und wird pro Datensatz **geklont**.

1

**🖥️ BACKEND: API‑SPEZIFIKATION**

**Basis‑Pfad:** /api

* POST /cv – CV anlegen (Body = CVDocument ohne id) → CVDocument.
* GET /cv/{id} – CV laden → CVDocument.
* PUT /cv/{id} – CV aktualisieren (Partial/Full) → CVDocument.
* DELETE /cv/{id} – Löschen.
* POST /upload/image – multipart, gibt { imagePath | base64 } zurück (Server speichert als temp‑Datei).
* POST /cv/{id}/export/docx – liefert DOCX als Stream/Attachment.
* POST /cv/{id}/export/pdf – liefert PDF als Stream/Attachment.

**Persistenz**

* MongoDB: DB cv\_builder, Collection cv\_documents, \_id = id.

**🧠 BACKEND: TEMPLATE‑EXPORT – ALGORITHMUS**

**Libs**: python-docx, Pillow (für Bild), Standard‑Lib.

**DOCX‑Erzeugung**:

1. **Vorlage kopieren/öffnen**: Lade backend/templates/Lebenslauf\_CV\_EMP\_2025\_Vorlage.docx.
2. **Skalare ersetzen**:
   * Iteriere **alle Absätze** + **alle Zellen** in **allen Tabellen**.
   * Für jeden Text, führe **string‑basiertes Replace** für jeden Token aus (Achtung: python-docx kann Runs splitten → implementiere eine „merge runs & replace“‑Hilfsroutine oder ersetze in Tabellenzellen/Absätzen per rekonstruierter Text‑Zuweisung).
3. **Bild einfügen** ({{Picture}}):
   * Finde die Zelle/den Absatz, lösche den Token‑Text und füge add\_run().add\_picture(stream, width=...) ein (Bytes aus Datei/Base64).
4. **Wiederholer** (Erfahrung/Bildung/Sprachen):
   * **Finde** die **erste Tabellenzelle** mit dem **Anker‑Token** (z. B. {{WorkFrom}}).
   * Hole row = cell.\_tc.tr bzw. via table.rows[row\_idx].
   * **Ersetze** in der **Musterzeile** die Tokens mit dem **ersten Datensatz**.
   * Für **weitere Datensätze**: **dupliziere** die Zeile (XML‑Clone von row.\_tr), **füge** sie darunter ein, **ersetze** Tokens.
   * Tipp: Schreibe util‑Funktionen find\_cell\_containing(token), clone\_row(table, row\_idx), replace\_tokens\_in\_row(row, mapping).
5. **Speichern in Memory** und als Response streamen.

**PDF‑Konvertierung**:

* Lege **Temp‑Pfad** an, speichere DOCX dorthin.
* Führe soffice --headless --convert-to pdf --outdir <tmp> <docx> aus.
* Streame die PDF‑Datei, lösche Temp‑Artefakte.

**🧪 TESTS**

* **Unit** (pytest):
  + test\_export\_tokens.py:
    - Skalar‑Replace (Paragraph + Table Cell).
    - Zeilenklon (N Einträge erzeugen N Zeilen).
    - Fallback bei **leeren Arrays**: kein Fehler, nur keine zusätzlichen Zeilen.
    - Bild‑Einfügen (prüfe, dass InlineShapes/Bild läuft – notfalls Smoke).
* **Integration**:
  + test\_api.py: POST /cv → GET /cv/{id} → POST /export/docx → prüfe, dass gültige DOCX‑Bytes kommen; danach POST /export/pdf → gültige PDF‑Bytes.
* **Manual/E2E**:
  + Starte docker‑compose, öffne Frontend, erfasse Datensätze, exportiere und prüfe, dass Vorlage gefüllt ist.

**🧱 BACKEND – CODE‑SKIZZEN**

**requirements.txt**

|  |
| --- |
| fastapi==0.110.0  uvicorn[standard]==0.27.1  motor==3.3.2  pydantic==1.10.13  python-dotenv==1.0.1  python-multipart==0.0.9  python-docx==0.8.11  Pillow==10.0.0 |

**Dockerfile.backend** (LibreOffice für PDF)

|  |
| --- |
| FROM python:3.11-slim  RUN apt-get update && apt-get install -y libreoffice && rm -rf /var/lib/apt/lists/\*  WORKDIR /app  COPY backend/requirements.txt .  RUN pip install --no-cache-dir -r requirements.txt  COPY backend/ /app  CMD ["uvicorn", "server:app", "--host", "0.0.0.0", "--port", "8001"] |

**Exporter‑Eintrittspunkt** backend/exporters/word\_template\_export.py (Skelett)

|  |
| --- |
| from docx import Document  from io import BytesIO  import subprocess, tempfile, os  from .image\_utils import resolve\_image\_stream  from typing import Dict, List, Tuple  # ---- public API ----  def generate\_docx\_from\_template(cv: Dict, template\_path: str) -> bytes:      doc = Document(template\_path)      # 1) scalars      replace\_scalars(doc, scalar\_mapping(cv))      # 2) picture      insert\_picture\_if\_present(doc, cv.get("personalInfo", {}))      # 3) repeaters      fill\_repeater(doc, anchor="{{WorkFrom}}", rows=cv.get("workExperience", []), token\_order=[          "{{WorkFrom}}","{{WorkTo}}","{{WorkEmployerName}}","{{WorkTitle}}","{{WorkDepartment}}","{{WorkEmployerAddress}}"      ])      fill\_repeater(doc, anchor="{{Edu.From}}", rows=cv.get("education", []), token\_order=[          "{{Edu.From}}","{{Edu.To}}","{{Edu.Qualification}}","{{Edu.Institution}}","{{Edu.Department}}","{{Edu.Profile}}","{{Edu.Address}}"      ])      fill\_repeater(doc, anchor="{{Lang.Language}}", rows=cv.get("languages", []), token\_order=[          "{{Lang.Language}}","{{Lang.Level}}"      ])      buf = BytesIO()      doc.save(buf)      return buf.getvalue()  def convert\_docx\_to\_pdf\_bytes(docx\_bytes: bytes) -> bytes:      with tempfile.TemporaryDirectory() as tmp:          docx\_path = os.path.join(tmp, "out.docx")          pdf\_path  = os.path.join(tmp, "out.pdf")          with open(docx\_path, "wb") as f: f.write(docx\_bytes)          subprocess.run(["soffice","--headless","--convert-to","pdf","--outdir",tmp,docx\_path], check=True)          with open(pdf\_path, "rb") as f: return f.read()  # ---- helpers (implement robustly) ----  def scalar\_mapping(cv: Dict) -> Dict[str,str]:      p = cv.get("personalInfo", {})      return {          "{{FirstName}}": p.get("firstName",""),          "{{LastName}}":  p.get("lastName",""),          "{{IDNumber}}":  p.get("idNumber",""),          "{{JobTitle}}":  p.get("jobTitle",""),          "{{Address}}":   p.get("address",""),          "{{City}}":      p.get("city",""),          "{{BirthDate}}": p.get("birthDate",""),          "{{BirthPlace}}":p.get("birthPlace",""),          "{{BirthCountry}}": p.get("birthCountry",""),          "{{Mobile}}":    p.get("phone",""),          "{{Email}}":     p.get("email",""),          "{{Gender}}":    p.get("gender",""),          "{{Nationality}}": p.get("nationality",""),          "{{CivilStatus}}": p.get("civilStatus",""),          "{{DriversLicense}}": p.get("driversLicense",""),          "{{Pers.Competences}}": p.get("socialSkills",""),          "{{EDVCompetences}}":   p.get("itSkills","")      }  def replace\_scalars(doc: Document, mapping: Dict[str,str]): ...  def insert\_picture\_if\_present(doc: Document, pi: Dict): ...  def fill\_repeater(doc: Document, anchor: str, rows: List[Dict], token\_order: List[str]): ... |

Implementiere replace\_scalars so, dass du **alle Absätze** und **jede Tabellenzelle** durchgehst, den vollständigen Zellentext (Runs mergen) ersetzt und wieder zurückschreibst.  
fill\_repeater: Suche anchor → ermittle table,row\_idx → ersetze Tokens in Musterzeile für den **ersten Datensatz**, **klone** Zeile für die restlichen Datensätze (XML‑Kopie), ersetze Tokens je Zeile.  
insert\_picture\_if\_present: Suche {{Picture}} in Zellen/Absätzen, lösche Marker und füge Bild via add\_run().add\_picture(...) (aus Datei/Base64) ein.

**🌐 FRONTEND – API‑CLIENT (axios)**

|  |
| --- |
| // frontend/src/api.js  import axios from "axios";  const API = (import.meta.env.VITE\_API\_BASE || "") + "/api";  export const cvApi = {    createCV: (cv) => axios.post(`${API}/cv`, cv).then(r => r.data),    getCV: (id) => axios.get(`${API}/cv/${id}`).then(r => r.data),    updateCV: (id, cv) => axios.put(`${API}/cv/${id}`, cv).then(r => r.data),    deleteCV: (id) => axios.delete(`${API}/cv/${id}`).then(r => r.data),    listCVs: () => axios.get(`${API}/cvs`).then(r => r.data),    uploadImage: (file) => {      const fd = new FormData(); fd.append("image", file);      return axios.post(`${API}/upload/image`, fd, { headers: { "Content-Type":"multipart/form-data" }}).then(r=>r.data);    },    exportDocx: (id) =>      axios.post(`${API}/cv/${id}/export/docx`, {}, { responseType: "blob" }),    exportPdf: (id) =>      axios.post(`${API}/cv/${id}/export/pdf`, {}, { responseType: "blob" }),  }; |

**UI‑Hinweise**

* CVBuilder.jsx: Formsektionen + Wiederholer (Experience/Education/Languages) als dynamische Listen (Add/Remove).
* Export‑Buttons rufen exportDocx/pdf auf, erzeugen Download‑Link (Blob → URL.createObjectURL).
* Bild‑Upload: uploadImage aufrufen; Backend liefert Dateipfad, im CV personalInfo.picturePath setzen.

**🐳 DOCKER & COMPOSE**

**docker-compose.yml**

|  |
| --- |
| version: "3.8"  services:    mongodb:      image: mongo:7      container\_name: cv\_mongo      restart: unless-stopped      ports: ["27017:27017"]      volumes: ["mongodb\_data:/data/db"]    backend:      build:        context: .        dockerfile: Dockerfile.backend      container\_name: cv\_backend      environment:        - MONGO\_URL=mongodb://mongodb:27017/cv\_builder        - DB\_NAME=cv\_builder      volumes:        - ./backend:/app      ports: ["8001:8001"]      depends\_on: [mongodb]    frontend:      build:        context: .        dockerfile: Dockerfile.frontend      container\_name: cv\_frontend      environment:        - VITE\_API\_BASE=http://localhost:8001      ports: ["3000:3000"]      depends\_on: [backend]  volumes:    mongodb\_data: |

**✅ AKZEPTANZKRITERIEN**

* docker-compose up --build startet alle Dienste ohne Fehler.
* **Frontend:**
  + CV anlegen, Einträge für Erfahrung/Bildung/Sprachen hinzufügen, Bild hochladen.
  + CV speichern → in Liste sichtbar → öffnen/aktualisieren/löschen.
  + „Export Word“: lädt eine **DOCX**, in der **alle Tokens** korrekt **ersetzt**, **Zeilen geklont** und **Bild** eingesetzt sind.
  + „Export PDF“: lädt ein **PDF**, das dem DOCX entspricht.
* **Tests**: pytest grün, (optional) E2E‑Smoke erfolgreich.

**🧪 TESTDATEN (Beispiel‑Payload)**

|  |
| --- |
| {    "personalInfo": {      "firstName":"Max","lastName":"Mustermann","idNumber":"EMP-2025-001",      "jobTitle":"Softwareentwickler","birthDate":"15.03.1990","birthPlace":"Hamburg",      "birthCountry":"Deutschland","nationality":"deutsch","gender":"männlich",      "civilStatus":"verheiratet","address":"Musterstraße 123","postalCode":"12345",      "city":"Berlin","phone":"+49 123 456789","email":"max@example.com",      "socialSkills":"Teamfähigkeit, Kommunikation","itSkills":"Python, React, Docker",      "driversLicense":"B","picturePath":"/app/media/photos/max.jpg"    },    "workExperience": [      {        "workFrom":"01.2022","workTo":"12.2023","workEmployerName":"Beispiel GmbH",        "workTitle":"Entwickler","workDepartment":"IT","workEmployerAddress":"Musterweg 1, Berlin",        "description":"Backend-APIs, CI/CD"      }    ],    "education":[      {        "eduFrom":"10.2010","eduTo":"09.2014","qualification":"B.Sc. Informatik",        "institution":"TU Berlin","department":"Fakultät IV","profile":"Softwaretechnik",        "institutionAddress":"Straße des 17. Juni 135, Berlin"      }    ],    "languages":[{"language":"Deutsch","level":"C2"},{"language":"Englisch","level":"C1"}]  } |

**⚠️ FEHLERBEHEBUNG – HINWEISE**

* **„Index außerhalb des gültigen Bereichs“** im Export deutet i. d. R. auf **leere Arrays**, **falsch erkannte Musterzeilen** oder **Off‑by‑one beim Spaltenmapping**. In der Python‑Implementierung genauso wie in VBA robust gegen **leere Listen** und **nicht gefundene Tokens** arbeiten (vor Klonen prüfen).
* Falls die Vorlage Tokens **über mehrere Runs** verteilt, beim Replace zunächst den **Zellentext** als Ganzes behandeln (Runs zusammenführen oder Text der Zelle direkt überschreiben).

**📎 VORLAGE‑BESONDERHEIT**

Die Platzhalter in deiner Word‑Datei folgen dem Schema oben – insbesondere **einzelne Tokens** für Skalarfelder und **Anker‑Tokens** in Tabellenzeilen für wiederholte Listen (Erfahrung/Bildung/Sprachen). Diese Struktur wird vom Exporter **ohne Änderung** der Vorlage unterstützt, indem er **Zeilen dupliziert** und Tokens ersetzt.

1

**Liefere** am Ende bitte:

* Den vollständigen **Quellcode** gemäß Struktur,
* docker-compose.yml, Dockerfile.backend, Dockerfile.frontend,
* Testläufe (Screens/Logs) und eine kurze **README** mit Start/Stop/Export‑Hinweisen.
* Aufgrund mangelnder Coding-Kenntnisse, wird eine Schritt-für-Schritt-Anleitung benötigt, um die App am Ende am lokalen Computer „zusammenzubauen“ und als fertige Anwendung z. B. EXE-Datei für den Endbenutzer startbereit zu machen.