**Struktur der App SolarDING:**

C:\SolarDING\

|-─ Hauptordner

│ ├ admin.py

│ ├ admin\_panel.py

│ ├ ai\_models.py

│ ├ analysis.py

│ ├ calculations.py

│ ├ check\_lines.py

│ ├ crm.py

│ ├ data\_input.py

│ ├ database.py

│ ├ de.json

│ ├ financial\_tools.py

│ ├ gui.py

│ ├ info\_platform.py

│ ├ locales.py

│ ├ map\_integration.py

│ ├ options.py

│ ├ pdf\_generator.py

│ ├ product\_db.py

│ ├ quick\_calc.py

│ ├ scenario\_manager.py

│ ├ skins.py

│ ├ three\_d\_viz.py

│ ├ utils.py

│ └ visualizer.py

**| ─ \data**

│ ├ app\_data.db

│ ├ MeineProdukte.xlsx

│ └ price\_matrix.xlsx

**└ ─ \\_\_pycache\_\_**

├ admin\_panel.cpython-313.pyc

├ analysis.cpython-313.pyc

├ calculations.cpython-313.pyc

├ crm.cpython-313.pyc

├ data\_input.cpython-313.pyc

├ database.cpython-313.pyc

├ gui.cpython-313.pyc

├ info\_platform.cpython-313.pyc

├ locales.cpython-313.pyc

├ options.cpython-313.pyc

├ product\_db.cpython-313.pyc

└ quick\_calc.cpython-313.pyc

**admin\_panel.py:**

# admin\_panel.py

# Modul für den Admin Panel Tab (F)

import streamlit as st

import pandas as pd

from typing import Dict, Any, Optional, List

import traceback

import os

from streamlit.runtime.uploaded\_file\_manager import UploadedFile # Typ-Hinweis für Datei-Upload

import io # Benötigt für StringIO, um String als Datei zu lesen

# Benötigt für den optionalen Excel Import der Produktdatenbank im Admin Bereich

import tempfile

import shutil

# Importiere Funktionen aus anderen Modulen

# Wir brauchen die Matrix Parsing Funktion von calculations.py

# Wir brauchen Admin Settings Funktionen von database.py

try:

# Importiere die Funktion mit ihrem neuen Namen (parse\_price\_matrix\_csv)

from calculations import parse\_price\_matrix\_csv

from database import load\_admin\_setting, save\_admin\_setting

db\_admin\_available = True

# print("admin\_panel.py: Erfolgreich calculations/database Funktionen importiert.") # Debugging

except ImportError as e:

st.error(f"FEHLER: Benötigte Module für Admin Panel konnten nicht geladen werden: {e}")

traceback.print\_exc()

db\_admin\_available = False

# Definiere Dummy Funktion mit dem erwarteten Namen, falls Import fehlschlägt

def parse\_price\_matrix\_csv(file\_source: Any): return None

def load\_admin\_setting(key, default=None): return default

def save\_admin\_setting(key, value): pass

def render\_admin\_panel(texts: Dict[str, str]):

"""

Rendert den Admin Panel Tab (F) der Streamlit Anwendung.

Beinhaltet Funktionen zum Laden/Verwalten von Konfigurationsdaten.

"""

# st.header(texts.get("menu\_item\_admin", "Admin (F)")) # Header wird schon in gui.py gesetzt

# Zeige Modul-Ladefehler an, falls vorhanden (kommen von gui.py, müssen hier nicht dupliziert werden)

# Kann optional hier angezeigt werden, wenn Fehler an die render Funktion übergeben werden

if not db\_admin\_available:

st.warning(texts.get("admin\_module\_unavailable", "Admin Funktionen sind aufgrund fehlender Module nicht verfügbar.")) # Neuen Textschlüssel nutzen

return # Rendering stoppen, wenn Abhängigkeiten fehlen

# --- Preis-Matrix Verwaltung ---

st.subheader(texts.get("admin\_price\_matrix\_header", "Preis-Matrix Verwaltung"))

st.info(texts.get("admin\_price\_matrix\_info", "Hier können Sie die Preis-Matrix per CSV-Datei hochladen und speichern. Die Matrix wird für die Kostenberechnung benötigt."))

# KORRIGIERT: Initialisiere uploaded\_file explizit mit None, BEVOR der uploader aufgerufen wird.

# Dies behebt den NameError in bestimmten Streamlit-Rerun-Szenarien.

uploaded\_file: Optional[UploadedFile] = None

# Streamlit File Uploader für die CSV-Datei - WEISE uploaded\_file HIER ZU

uploaded\_file = st.file\_uploader(

texts.get("admin\_upload\_matrix\_label", "CSV-Datei für Preis-Matrix hochladen"),

type=['csv'],

help=texts.get("admin\_upload\_matrix\_help", "Wählen Sie die exportierte CSV-Datei der Preis-Matrix aus.")

)

# Status der aktuell geladenen Matrix im Session State anzeigen

if 'price\_matrix\_df' in st.session\_state and st.session\_state['price\_matrix\_df'] is not None:

st.success(texts.get("admin\_matrix\_loaded\_status", "Preis-Matrix ist aktuell geladen."))

matrix\_df: pd.DataFrame = st.session\_state['price\_matrix\_df'] # Typ-Hinweis für den geladenen DF

st.write(f"Shape: {matrix\_df.shape[0]} Zeilen, {matrix\_df.shape[1]} Spalten")

st.write(texts.get("admin\_matrix\_head", "Erste 5 Zeilen:"), matrix\_df.head())

# st.write(texts.get("admin\_matrix\_tail", "Letzte 5 Zeilen:"), matrix\_df.tail()) # Optional anzeigen

# Optional: Button, um die geladene Matrix aus dem Session State zu entfernen (und DB)

if st.button(texts.get("admin\_clear\_matrix\_button", "Geladene Matrix entfernen")): # Neuen Textschlüssel nutzen

# Optional: Auch aus der Datenbank entfernen? Das wäre der "Reset"

try:

save\_admin\_setting('price\_matrix\_csv\_data', None) # Setze den DB Eintrag auf None, um ihn zu löschen

st.session\_state['price\_matrix\_df'] = None # Setze Session State auf None

st.info(texts.get("admin\_matrix\_cleared", "Matrix aus Session und Datenbank entfernt.")) # Neuen Textschlüssel nutzen

st.rerun() # Seite neu laden, um den Status zu aktualisieren

except Exception as e:

st.error(texts.get("admin\_matrix\_clear\_failure", "Fehler beim Entfernen der Matrix aus der Datenbank.").format(error=e)) # Neuer Textschlüssel

traceback.print\_exc()

else:

st.info(texts.get("admin\_matrix\_not\_loaded\_status", "Aktuell ist keine Preis-Matrix geladen."))

# Button zum Verarbeiten der hochgeladenen Datei

# Dieser Button wird nur angezeigt, wenn eine Datei hochgeladen wurde.

# KORRIGIERT:uploaded\_file ist jetzt garantiert definiert, auch wenn None, bevor die if-Bedingung kommt.

if uploaded\_file is not None:

# Den Buttontext dynamisch gestalten

process\_button\_text = texts.get("admin\_process\_uploaded\_matrix\_button", "Datei laden & speichern").replace("Datei", f"'{uploaded\_file.name}'")

if st.button(process\_button\_text):

# Rufe die Parsing Funktion aus calculations.py auf

# parse\_price\_matrix\_csv muss fähig sein, ein UploadedFile Objekt zu verarbeiten

parsed\_df = parse\_price\_matrix\_csv(uploaded\_file) # Übergib das UploadedFile Objekt

if parsed\_df is not None:

try:

# Convert DataFrame back to CSV string for saving in DB

# Use index=True to save the index (Module Quantity) to the CSV string

# Speichere es in einem konsistenten Format, z.g. Semikolon/Komma, UTF-8

csv\_string = parsed\_df.to\_csv(index=True, sep=';', decimal=',', encoding='utf-8')

# Save the string in admin\_settings

save\_admin\_setting('price\_matrix\_csv\_data', csv\_string) # Save as raw string (TEXT in DB)

st.success(texts.get("admin\_matrix\_load\_success", "Preis-Matrix '{file\_name}' erfolgreich geladen und gespeichert!").format(file\_name=uploaded\_file.name))

# Lade die Matrix auch in den Session State aus dem soeben in der DB gespeicherten Wert

# Dies stellt sicher, dass der Session State mit der persistenten Version übereinstimmt.

# Wir müssen dafür nicht die Datei erneut parsen, sondern laden aus der DB

db\_csv\_string = load\_admin\_setting('price\_matrix\_csv\_data')

if db\_csv\_string is not None:

# Rufe die Parsing Funktion mit dem String aus der DB auf (als StringIO)

# Benötigt io.StringIO, um den String als Datei zu lesen

# Die Parsing-Parameter (sep, decimal) sollten mit denen beim Speichern übereinstimmen oder parse\_price\_matrix\_csv muss robust sein.

# Da parse\_price\_matrix\_csv robust ist, sollte es klappen.

parsed\_for\_session = parse\_price\_matrix\_csv(io.StringIO(db\_csv\_string)) # Parse from string

if parsed\_for\_session is not None:

st.session\_state['price\_matrix\_df'] = parsed\_for\_session

# print("DEBUG: Matrix loaded from DB to session state after upload+save.") # Debugging

else:

# Dies sollte nicht passieren, wenn save\_admin\_setting und load\_admin\_setting korrekt arbeiten

st.error(texts.get("admin\_matrix\_load\_failure", "Fehler beim erneuten Laden der Matrix aus der Datenbank nach dem Speichern.").format(file\_name=uploaded\_file.name))

# Clear session state on failure

if 'price\_matrix\_df' in st.session\_state: del st.session\_state['price\_matrix\_df']

# Optional: Rerun to update the status display at the top

st.rerun()

except Exception as e:

st.error(texts.get("admin\_matrix\_save\_failure", "Ein Fehler ist beim Speichern der Preis-Matrix in der Datenbank aufgetreten.").format(file\_name=uploaded\_file.name))

traceback.print\_exc()

# Ensure any half-loaded or old matrix is removed on failure

if 'price\_matrix\_df' in st.session\_state: del st.session\_state['price\_matrix\_df']

else:

# parse\_price\_matrix\_csv should print errors internally, but add a generic one here

st.error(texts.get("admin\_matrix\_load\_failure", "Fehler beim Laden der Preis-Matrix '{file\_name}'. Prüfen Sie das Dateiformat und die Struktur sowie die Konsole für Details.").format(file\_name=uploaded\_file.name))

# Ensure any half-loaded or old matrix is removed on failure

if 'price\_matrix\_df' in st.session\_state: del st.session\_state['price\_matrix\_df']

st.markdown("---") # Separator

# --- Allgemeine Admin Einstellungen (Platzhalter) ---

st.subheader(texts.get("admin\_general\_settings\_header", "Allgemeine Einstellungen"))

st.info(texts.get("admin\_general\_settings\_info", "Verwaltung allgemeiner Einstellungen (Tarife, Margen, etc.) kommt hierher."))

# TODO: Implement loading/saving settings using database.load\_admin\_setting/save\_admin\_setting

if db\_admin\_available: # Nur anzeigen, wenn DB Funktionen verfügbar sind

current\_vat\_rate = load\_admin\_setting('default\_vat\_rate\_percent', 19.0) # Holen Sie den aktuellen Wert

new\_vat\_rate = st.number\_input(texts.get("admin\_setting\_vat\_rate", "Standard MwSt Rate (%)"), value=current\_vat\_rate, min\_value=0.0, max\_value=100.0, step=0.1) # Neuer Text Schlüssel

if st.button(texts.get("admin\_save\_settings\_button", "Einstellungen speichern")): # Neuer Text Schlüssel

save\_admin\_setting('default\_vat\_rate\_percent', new\_vat\_rate) # Speichern des neuen Wertes

st.success(texts.get("admin\_settings\_saved\_success", "Einstellung gespeichert!")) # Neuer Text Schlüssel

st.markdown("---") # Separator

# --- Produktdatenbank Verwaltung (Platzhalter) ---

st.subheader(texts.get("admin\_product\_db\_header", "Produktdatenbank"))

st.info(texts.get("admin\_product\_db\_info", "Verwaltung der Produktdatenbank (Import/Export, Bearbeiten) kommt hierher."))

# TODO: Implement import/export using product\_db.import\_products\_from\_excel etc.

if db\_admin\_available: # Nur anzeigen, wenn DB Funktionen verfügbar sind

# import product\_db # Importieren Sie product\_db nur bei Bedarf

uploaded\_product\_db\_file = st.file\_uploader(texts.get("admin\_upload\_product\_db\_label", "Produktdatenbank (Excel) hochladen"), type=['xlsx']) # Neuer Text Schlüssel

if uploaded\_product\_db\_file is not None:

# Sie müssten das UploadedFile Objekt temporär speichern oder direkt an product\_db.import\_products\_from\_excel übergeben

# product\_db.import\_products\_from\_excel muss evtl. angepasst werden, um Datei-Objekt zu akzeptieren oder Sie speichern temporär.

# Temporär speichern und dann Pfad übergeben

try:

# imports tempfile und shutil sind schon oben im Admin Panel Modul, wo sie gebraucht werden

with tempfile.NamedTemporaryFile(delete=False, suffix=".xlsx") as tmp\_file:

shutil.copyfileobj(uploaded\_product\_db\_file, tmp\_file)

tmp\_path = tmp\_file.name

import product\_db # Importieren Sie product\_db nur bei Bedarf

results = product\_db.import\_products\_from\_excel(tmp\_path) # Annahme: import\_products\_from\_excel akzeptiert Pfad

os.unlink(tmp\_path) # Temporäre Datei löschen

st.write(texts.get("admin\_import\_results", "Import Ergebnisse:"), results) # Neuer Text Schlüssel

except ImportError:

st.error("Produkt-Datenbank Modul nicht verfügbar (product\_db.py Import Fehler?).")

traceback.print\_exc()

except Exception as e:

st.error(f"Fehler beim Import der Produktdatenbank: {e}")

traceback.print\_exc()

st.markdown("---") # Separator

# --- Benutzerverwaltung (Platzhalter) ---

st.subheader(texts.get("admin\_user\_management\_header", "Benutzerverwaltung"))

st.info(texts.get("admin\_user\_management\_info", "Verwaltung von Benutzern und Berechtigungen kommt hierher (optional)."))

# TODO: Implement user management

# Füge neue Text Schlüssel zu texts\_fallback in gui.py hinzu:

# admin\_matrix\_save\_failure

# admin\_setting\_vat\_rate

# admin\_save\_settings\_button

# admin\_settings\_saved\_success

# admin\_upload\_product\_db\_label

# admin\_import\_results

# admin\_matrix\_clear\_failure

# imports für tempfile und shutil für Excel upload

# import tempfile

# import shutil

**Analysis.py:**

# analysis.py

# Modul für den Analyse Tab / Dashboard (A.5)

import streamlit as st

import pandas as pd

from typing import Dict, Any, Optional, List

import traceback

import math # Für float('inf') und math.isnan Prüfung

# Importiere benötigte Funktionen/Daten (falls analysis.py auf Datenbank oder Settings zugreift)

try:

# Beispiel: Wenn die Analyse Admin Settings für Anzeige braucht (z.B. Standard-Strompreissteigerung)

from database import load\_admin\_setting

# from product\_db import lookup\_product\_price # Beispiel, falls Details zu Produkten angezeigt werden

analysis\_dependencies\_available = True

except ImportError as e:

st.error(f"FEHLER: Benötigte Module für Analysis Tab konnten nicht geladen werden: {e}")

traceback.print\_exc()

analysis\_dependencies\_available = False

# Definiere Dummy Funktionen, falls Import fehlschlägt

def load\_admin\_setting(key, default=None): return default

# KORRIGIERT: render\_analysis Funktion erweitert um mehr Ergebnisse und Visualisierungen anzuzeigen

def render\_analysis(texts: Dict[str, str], results: Optional[Dict[str, Any]]):

"""

Rendert den Analyse Tab (A.5 Dashboard) der Streamlit Anwendung.

Zeigt die Berechnungsergebnisse und Visualisierungen an,

inklusive mehrjähriger Simulation.

Args:

texts: Dictionary mit den lokalisierten Texten.

results: Dictionary mit den Berechnungsergebnissen von perform\_calculations,

oder None, wenn keine Berechnungen durchgeführt wurden oder fehlerhaft waren.

"""

# Der Header wird in gui.py gesetzt, aber analysis.py kann seinen eigenen Subheader haben

# st.subheader(texts.get("dashboard\_header", "Ergebnisse und Dashboard")) # Header ist in gui.py schon da

if results is None or not isinstance(results, dict):

# Zeige Info, wenn noch keine Ergebnisse vorliegen oder das Format falsch ist

st.info(texts.get('calculation\_no\_result\_info', "Berechnungen lieferten kein Ergebnis."))

# Die Warnung vom Dummy\_perform\_calculations in gui.py wird schon global angezeigt, falls das Modul fehlt.

return # Nichts weiter rendern, wenn keine Ergebnisse da sind oder das Format falsch ist

# --- Ergebnisse anzeigen ---

# Verwenden Sie Streamlit-Layouts wie st.columns oder st.expander für bessere Struktur

# Zeige gesammelte Berechnungsfehler zuerst an (immer oben, falls vorhanden)

if results.get('calculation\_errors'):

st.error(texts.get('calculation\_errors\_header', "Es traten Fehler während der Berechnungen auf:"))

for err in results['calculation\_errors']:

st.error(err)

# Optional: Zeige auch eine allgemeine Warnung, dass Ergebnisse unvollständig/fehlerhaft sein können

# st.warning("Die Ergebnisse sind möglicherweise unvollständig oder fehlerhaft.")

# Verwenden Sie einen Expander für die Hauptübersicht oder direkt anzeigen

# with st.expander(texts.get("dashboard\_header", "Ergebnisse und Dashboard"), expanded=True): # Kann hier Header setzen

# --- Anzeige Haupt-KPIs ---

st.subheader(texts.get("dashboard\_header", "Ergebnisse und Dashboard")) # Setzen Sie den Header hier

col1, col2, col3 = st.columns(3) # 3 Spalten Layout

# Gesamtinvestition Brutto (aus Matrix + Aufpreise + MwSt)

total\_investment\_brutto = results.get('total\_investment\_brutto', 0.0)

col1.metric(

label=texts.get('total\_investment\_brutto', 'Gesamtinvestition (brutto)'),

value=f"{total\_investment\_brutto:.2f} €"

)

# Anlagengröße in kWp (kommt aus Input, in results gespeichert)

anlage\_kwp = results.get('anlage\_kwp', 0.0)

col2.metric(

label=texts.get('anlage\_size\_label', 'Anlagengröße (kWp)'),

value=f"{anlage\_kwp:.2f} kWp"

)

# Jährliche PV Produktion (geschätzt)

annual\_pv\_production\_kwh = results.get('annual\_pv\_production\_kwh', 0.0)

col3.metric(

label=texts.get('annual\_pv\_production\_kwh', 'Jahresproduktion (kWh)'),

value=f"{annual\_pv\_production\_kwh:.2f} kWh"

)

col4, col5, col6 = st.columns(3) # Weitere Spalten

# Eigenversorgungsgrad (%)

self\_supply\_rate\_percent = results.get('self\_supply\_rate\_percent', 0.0)

col4.metric(

label=texts.get('self\_supply\_rate\_percent', 'Eigenversorgungsgrad (%)'),

value=f"{self\_supply\_rate\_percent:.2f} %"

)

# Jährlicher finanzieller Vorteil (Ersparnis + Einnahmen)

annual\_financial\_benefit = results.get('annual\_financial\_benefit', 0.0)

col5.metric(

label=texts.get('annual\_financial\_benefit', 'Jährlicher finanzieller Vorteil'),

value=f"{annual\_financial\_benefit:.2f} €"

)

# Amortisationszeit (handle Inf) - Use the simulated one now

amortization\_time = results.get('amortization\_time\_years')

if isinstance(amortization\_time, float) and math.isinf(amortization\_time): # Use math.isinf for robustness

amortization\_display = texts.get("amortization\_time\_infinite", "Unendlich") # Neuer Text Schlüssel

elif isinstance(amortization\_time, (int, float)) and amortization\_time >= 0:

# Show whole years if it's close to a whole number, otherwise .2f

if abs(amortization\_time - round(amortization\_time)) < 0.01:

amortization\_display = f"{round(amortization\_time)} Jahre"

else:

amortization\_display = f"{amortization\_time:.2f} Jahre"

else:

amortization\_display = "N/A"

col6.metric(

label=texts.get('amortization\_time\_years', 'Amortisationszeit (Jahre)'),

value=amortization\_display

)

# Jährliche CO2 Einsparung

annual\_co2\_savings\_kg = results.get('annual\_co2\_savings\_kg', 0.0)

st.metric(

label=texts.get('annual\_co2\_savings\_kg', 'Jährliche CO₂-Einsparung (kg)'),

value=f"{annual\_co2\_savings\_kg:.2f} kg"

)

st.markdown("---") # Trennlinie

# --- Detaillierte Kostenübersicht ---

# Zeige nur an, wenn detaillierte Kosten im results dict vorhanden sind

if results.get('total\_investment\_netto') is not None: # Prüfen, ob Kostenkalkulation lief

st.subheader(texts.get("cost\_details\_header", "Kosten Details"))

# Erstellen eines DataFrames für die Kosten Details

# Include all the detailed cost components calculated in calculations.py

cost\_data\_list = [

(texts.get('base\_matrix\_price\_netto', 'Matrix Basispreis (netto)'), results.get('base\_matrix\_price\_netto', 0.0)),

(texts.get('cost\_modules', 'Kosten Module (Aufpreis netto)'), results.get('cost\_modules\_aufpreis\_netto', 0.0)),

(texts.get('cost\_inverter', 'Kosten Wechselrichter (Aufpreis netto)'), results.get('cost\_inverter\_aufpreis\_netto', 0.0)),

(texts.get('cost\_storage', 'Kosten Speicher (Aufpreis Produkt-DB netto)'), results.get('cost\_storage\_aufpreis\_product\_db\_netto', 0.0)), # Note about potential double count

(texts.get('cost\_accessories\_aufpreis\_netto', 'Kosten Zubehör (Aufpreis netto)'), results.get('cost\_accessories\_aufpreis\_netto', 0.0)),

(texts.get('cost\_product\_surcharges\_netto', 'Summe Produkt Aufpreise (netto)'), results.get('cost\_product\_surcharges\_netto', 0.0)), # Neue Summe

(texts.get('cost\_scaffolding\_netto', 'Kosten Gerüst (netto)'), results.get('cost\_scaffolding\_netto', 0.0)), # Neue Kosten

# (texts.get('cost\_complex\_roof\_install\_netto', 'Kosten Dachmontage (netto)'), results.get('cost\_complex\_roof\_install\_netto', 0.0)), # Neue Kosten (wenn implementiert)

(texts.get('cost\_misc\_netto', 'Sonstige Kosten (netto)'), results.get('cost\_misc\_netto', 0.0)), # Neue Kosten

(texts.get('cost\_custom\_netto', 'Kundenwunsch Kosten (netto)'), results.get('cost\_custom\_netto', 0.0)), # Neue Kosten

(texts.get('total\_additional\_costs\_netto', 'Summe zusätzliche Kosten (netto)'), results.get('total\_additional\_costs\_netto', 0.0)), # Neue Summe

(texts.get('subtotal\_netto', 'Zwischensumme (netto)'), results.get('subtotal\_netto', 0.0)), # Neue Zwischensumme

(texts.get('custom\_surcharge\_amount\_netto', 'Aufschlag (netto)'), results.get('custom\_surcharge\_amount\_netto', 0.0)), # Neuer Betrag

(texts.get('custom\_discount\_amount\_netto', 'Rabatt (netto)'), results.get('custom\_discount\_amount\_netto', 0.0)), # Neuer Betrag

(texts.get('total\_investment\_netto', 'Gesamtinvestition (netto)'), results.get('total\_investment\_netto', 0.0)), # Final Netto

(texts.get('vat\_rate\_percent', 'MwSt Rate (%)'), results.get('vat\_rate\_percent', 0.0)),

(texts.get('total\_investment\_brutto', 'Gesamtinvestition (brutto)'), results.get('total\_investment\_brutto', 0.0)), # Final Brutto

]

# Filtere Einträge mit Wert None oder leeren Strings raus, wenn sie nicht relevant sind

cost\_data\_list = [(label, value) for label, value in cost\_data\_list if value is not None and (not isinstance(value, str) or value != '')]

# Konvertiere Beträge zu formatierten Strings für die Anzeige, behalte Prozente als Zahl für spätere Nutzung falls nötig

formatted\_cost\_data = []

for label, value in cost\_data\_list:

if label == texts.get('vat\_rate\_percent'):

formatted\_value = f"{value:.1f} %"

elif label in [texts.get('custom\_surcharge\_amount\_netto'), texts.get('custom\_discount\_amount\_netto')]:

# Optional: Fügen Sie das Vorzeichen hinzu, falls nicht schon im Label

formatted\_value = f"{value:.2f} €" # Das Label sollte das +/- enthalten

elif label in [texts.get('total\_investment\_netto'), texts.get('total\_investment\_brutto')]:

formatted\_value = f"{value:.2f} €\*\*" # Endsummen hervorheben

elif isinstance(value, (int, float)):

formatted\_value = f"{value:.2f} €"

else:

formatted\_value = value # Lasse andere Typen unverändert

formatted\_cost\_data.append((label, formatted\_value))

# Zeige die Kosten als Tabelle an

cost\_df = pd.DataFrame(formatted\_cost\_data, columns=['Kostenpunkt', 'Betrag'])

st.dataframe(cost\_df, use\_container\_width=True, hide\_index=True)

# HINWEIS zur potenziellen Doppeltzählung Speicher

st.info(texts.get("analysis\_storage\_cost\_note", "Hinweis: Die 'Kosten Speicher (Aufpreis Produkt-DB netto)' werden zusätzlich zum Matrix Basispreis ausgewiesen. Prüfen Sie Ihre Preis-Matrix Definition, um eine Doppeltzählung zu vermeiden, falls der Matrixpreis bereits den Speicher abdeckt."))

# --- Visualisierungen für Kosten ---

st.subheader(texts.get("cost\_visualization\_header", "Kostenübersicht Visualisierung")) # Neuer Text Schlüssel

# Daten für das Diagramm (Hauptkomponenten)

# Nehmen Sie die Zwischensummen vor den prozentualen Anpassungen

chart\_data\_costs\_raw = {

texts.get('base\_matrix\_price\_netto', 'Matrix Basispreis'): results.get('base\_matrix\_price\_netto', 0.0),

texts.get('cost\_product\_surcharges\_netto', 'Produkt Aufpreise'): results.get('cost\_product\_surcharges\_netto', 0.0),

texts.get('cost\_accessories\_aufpreis\_netto', 'Zubehör Aufpreise'): results.get('cost\_accessories\_aufpreis\_netto', 0.0),

texts.get('total\_additional\_costs\_netto', 'Zusätzl. Kosten'): results.get('total\_additional\_costs\_netto', 0.0),

}

# Filtere Komponenten mit Kosten 0.0 für das Diagramm, es sei denn, alle sind 0

chart\_data\_costs\_filtered = {k: v for k, v in chart\_data\_costs\_raw.items() if v != 0.0} # Filtere exakt 0.0

if not chart\_data\_costs\_filtered:

# Wenn alle 0 sind, zeige eine Info und kein Diagramm

st.info(texts.get("cost\_visualization\_no\_data", "Keine Kostendaten für Visualisierung verfügbar (Gesamtkosten sind 0).")) # Neuer Text Schlüssel

else:

# Konvertiere zu DataFrame format für st.bar\_chart

chart\_df\_costs = pd.DataFrame(list(chart\_data\_costs\_filtered.items()), columns=['Komponente', 'Kosten'])

st.bar\_chart(chart\_df\_costs.set\_index('Komponente')) # Set component as index for chart labels

st.markdown("---") # Trennlinie

# --- Detaillierte Ertrags- und Verbrauchsübersicht ---

# Zeige nur an, wenn entsprechende Werte im results dict vorhanden sind

if results.get('annual\_self\_consumption\_kwh') is not None:

st.subheader(texts.get("yield\_consumption\_details\_header", "Ertrags- und Verbrauchsdetails")) # Neuer Text Schlüssel

col\_yc1, col\_yc2 = st.columns(2) # 2 Spalten

# Ertrag

col\_yc1.write(f"{texts.get('specific\_yield\_kwh\_kwp\_yr', 'Spezifischer Ertrag (kWh/kWp/Jahr)')}:\*\* {results.get('specific\_yield\_kwh\_kwp\_yr', 0.0):.2f}")

col\_yc1.write(f"{texts.get('annual\_pv\_production\_kwh', 'Jahresproduktion (kWh)')}:\*\* {results.get('annual\_pv\_production\_kwh', 0.0):.2f}")

col\_yc1.write(f"{texts.get('annual\_feed\_in\_kwh', 'Jährliche Einspeisung (kWh)')}:\*\* {results.get('annual\_feed\_in\_kwh', 0.0):.2f}")

col\_yc1.write(f"{texts.get('feed\_in\_tariff\_euro\_kwh', 'Einspeisetarif (€/kWh)')}:\*\* {results.get('feed\_in\_tariff\_euro\_kwh', 0.0):.4f}") # Mehr Dezimalstellen

# Verbrauch & Eigenverbrauch

# total\_consumption\_kwh\_yr is stored in results['total\_consumption\_kwh\_yr']

col\_yc2.write(f"{texts.get('total\_annual\_consumption\_label', 'Gesamtjahresverbrauch (kWh)')}:\*\* {results.get('total\_consumption\_kwh\_yr', 0.0):.2f}")

col\_yc2.write(f"{texts.get('annual\_self\_consumption\_kwh', 'Jährlicher Eigenverbrauch (kWh)')}:\*\* {results.get('annual\_self\_consumption\_kwh', 0.0):.2f}")

col\_yc2.write(f"{texts.get('annual\_grid\_consumption\_kwh', 'Jährlicher Netzbezug (kWh)')}:\*\* {results.get('annual\_grid\_consumption\_kwh', 0.0):.2f}")

# electricity\_price\_kwh is stored in results['electricity\_price\_kwh']

col\_yc2.write(f"{texts.get('electricity\_price\_label', 'Strompreis (€/kWh)')}:\*\* {results.get('electricity\_price\_kwh', 0.0):.4f}")

col\_yc2.write(f"{texts.get('self\_supply\_rate\_percent', 'Eigenversorgungsgrad (%)')}:\*\* {results.get('self\_supply\_rate\_percent', 0.0):.2f} %")

# --- Visualisierungen für Energiefluss ---

st.subheader(texts.get("energy\_flow\_visualization\_header", "Energiefluss Visualisierung")) # Neuer Text Schlüssel

# Daten für das Diagramm: Gesamtverbrauch vs Eigenverbrauch vs Netzbezug vs PV Produktion vs Einspeisung

# Es ist oft am klarsten zu zeigen: Consumption = Self-Consumption + Grid Import

# Und Production = Self-Consumption + Feed-in

# Zeige Verbrauch, Eigenverbrauch, Netzbezug, PV Produktion, Einspeisung

chart\_data\_energy\_raw = {

texts.get('total\_annual\_consumption\_label', 'Gesamtverbrauch'): results.get('total\_consumption\_kwh\_yr', 0.0),

texts.get('annual\_pv\_production\_kwh', 'PV Produktion'): results.get('annual\_pv\_production\_kwh', 0.0),

texts.get('annual\_self\_consumption\_kwh', 'Eigenverbrauch'): results.get('annual\_self\_consumption\_kwh', 0.0),

texts.get('annual\_grid\_consumption\_kwh', 'Netzbezug'): results.get('annual\_grid\_consumption\_kwh', 0.0),

texts.get('annual\_feed\_in\_kwh', 'Einspeisung'): results.get('annual\_feed\_in\_kwh', 0.0),

}

# Filtere 0.0 Werte für das Diagramm, es sei denn, alle sind 0

chart\_data\_energy\_filtered = {k: v for k, v in chart\_data\_energy\_raw.items() if v != 0.0}

if not chart\_data\_energy\_filtered:

# Wenn alle 0 sind, zeige eine Info und kein Diagramm

st.info(texts.get("energy\_flow\_visualization\_no\_data", "Keine Energieflussdaten für Visualisierung verfügbar (alle Werte sind 0).")) # Neuer Text Schlüssel

else:

# Konvertiere zu DataFrame format für st.bar\_chart

chart\_df\_energy = pd.DataFrame(list(chart\_data\_energy\_filtered.items()), columns=['Fluss', 'kWh pro Jahr'])

st.bar\_chart(chart\_df\_energy.set\_index('Fluss')) # Set flow as index for chart labels

st.markdown("---") # Trennlinie

# --- Wirtschaftlichkeit Details über die Laufzeit --- (Add placeholder for NPV/IRR)

# Zeige nur an, wenn Simulationsergebnisse vorhanden sind

if results.get('annual\_cash\_flows\_sim'): # Check if simulation ran and produced cash flows

st.subheader(texts.get("economy\_details\_header", "Wirtschaftlichkeitsdetails über die Laufzeit")) # Header angepasst

# NPV und IRR anzeigen

npv\_value = results.get('npv\_over\_years')

irr\_percent = results.get('irr\_percent')

col\_econ1, col\_econ2 = st.columns(2)

# NPV

if isinstance(npv\_value, (int, float)) and not math.isnan(npv\_value): # Use math.isnan

col\_econ1.metric(label=texts.get('npv\_over\_years', 'Kapitalwert (NPV)'), value=f"{npv\_value:.2f} €")

else:

col\_econ1.metric(label=texts.get('npv\_over\_years', 'Kapitalwert (NPV)'), value="N/A")

# IRR

if isinstance(irr\_percent, (int, float)) and not math.isnan(irr\_percent): # Use math.isnan

col\_econ2.metric(label=texts.get('irr\_percent', 'Interner Zinsfuß (IRR %)'), value=f"{irr\_percent:.2f} %")

else:

col\_econ2.metric(label=texts.get('irr\_percent', 'Interner Zinsfuß (IRR %)'), value="N/A")

# --- Visualisierung der finanziellen Entwicklung ---

st.subheader(texts.get("financial\_visualization\_header", "Finanzielle Entwicklung über die Laufzeit")) # Neuer Text Schlüssel

simulation\_years = results.get('simulation\_period\_years', len(results.get('cumulative\_cash\_flows\_sim', []))) # Get simulation period

years = list(range(simulation\_years + 1)) # Years from 0 to simulation\_years

# Line Chart: Kumulierter Cash Flow

# Data needs Year 0 (-Initial Investment) + Cumulative Cash Flows (Year 1 to N)

initial\_investment\_netto = results.get('total\_investment\_netto', 0.0) # Use net investment for CF analysis

cumulative\_cash\_flows\_sim = results.get('cumulative\_cash\_flows\_sim', [])

# Combine Year 0 and cumulative flows

cumulative\_cf\_data = [-initial\_investment\_netto] + cumulative\_cash\_flows\_sim

# Create DataFrame for the chart

chart\_data\_cumulative\_cf = pd.DataFrame({

texts.get("analysis\_chart\_year\_label", "Jahr"): list(range(simulation\_years + 1)), # New text key for axis label

texts.get("analysis\_chart\_cumulative\_cf\_label", "Kumulierter Cash Flow (€)"): cumulative\_cf\_data # New text key for series label

})

st.line\_chart(chart\_data\_cumulative\_cf.set\_index(texts.get("analysis\_chart\_year\_label", "Jahr")))

# Optional: Bar Chart: Jährlicher Cash Flow

annual\_cash\_flows\_sim = results.get('annual\_cash\_flows\_sim', [])

# Data needs Year 1 to N annual cash flows

if annual\_cash\_flows\_sim: # Only show if there are annual cash flows

chart\_data\_annual\_cf = pd.DataFrame({

texts.get("analysis\_chart\_year\_label", "Jahr"): list(range(1, simulation\_years + 1)), # Start from Year 1

texts.get("analysis\_chart\_annual\_cf\_label", "Jährlicher Cash Flow (€)"): annual\_cash\_flows\_sim # New text key for series label

})

st.bar\_chart(chart\_data\_annual\_cf.set\_index(texts.get("analysis\_chart\_year\_label", "Jahr")))

# --- Simulation Details Tabelle ---

st.subheader(texts.get("simulation\_details\_header", "Simulationsdetails (Jährlich)")) # Neuer Text Schlüssel

# Create DataFrame for the detailed annual table

# Ensure all lists are the same length (simulation\_years)

if results.get('annual\_productions\_sim') and len(results['annual\_productions\_sim']) == simulation\_years: # Check if simulation lists are available and have expected length

sim\_table\_data = {

texts.get("analysis\_table\_year\_header", "Jahr"): list(range(1, simulation\_years + 1)), # New text key for column header

texts.get("annual\_pv\_production\_kwh", "Jahresproduktion (kWh)"): results.get('annual\_productions\_sim', []), # Assumed key from calculations

texts.get("total\_annual\_consumption\_label", "Gesamtverbrauch (kWh)"): [results.get('total\_consumption\_kwh\_yr', 0.0)] \* simulation\_years, # Assuming constant consumption for now

texts.get("electricity\_price\_label", "Strompreis (€/kWh)"): results.get('annual\_elec\_prices\_sim', []), # Assumed key from calculations

texts.get('feed\_in\_tariff\_euro\_kwh', 'Einspeisetarif (€/kWh)'): results.get('annual\_feed\_in\_tariffs\_sim', []), # Assumed key from calculations

texts.get("annual\_financial\_benefit", "Jährlicher Vorteil (€)"): results.get('annual\_benefits\_sim', []), # Assumed key from calculations

texts.get("annual\_maintenance\_cost\_sim", "Wartungskosten (€)"): results.get('annual\_maintenance\_costs\_sim', []), # New text key, Assumed key

texts.get("analysis\_table\_annual\_cf\_header", "Jährlicher Cash Flow (€)"): annual\_cash\_flows\_sim, # New text key

texts.get("analysis\_table\_cumulative\_cf\_header", "Kumulierter Cash Flow (€)"): cumulative\_cash\_flows\_sim, # New text key

}

# Format columns in the DataFrame for display

sim\_table\_df = pd.DataFrame(sim\_table\_data)

# Apply formatting to numeric columns

for col in [texts.get("annual\_pv\_production\_kwh"), texts.get("total\_annual\_consumption\_label"), texts.get("electricity\_price\_label"), texts.get('feed\_in\_tariff\_euro\_kwh'), texts.get("annual\_financial\_benefit"), texts.get("annual\_maintenance\_cost\_sim"), texts.get("analysis\_table\_annual\_cf\_header"), texts.get("analysis\_table\_cumulative\_cf\_header")]:

if col in sim\_table\_df.columns:

# Check if the value type requires float formatting

if sim\_table\_df[col].dtype in ['float64', 'int64']: # Check if the column contains numeric types

if col in [texts.get("electricity\_price\_label"), texts.get('feed\_in\_tariff\_euro\_kwh')]: # Price/Tariff formatting

sim\_table\_df[col] = sim\_table\_df[col].apply(lambda x: f"{x:.4f}")

elif col in [texts.get("annual\_pv\_production\_kwh"), texts.get("total\_annual\_consumption\_label")]: # KWh formatting

sim\_table\_df[col] = sim\_table\_df[col].apply(lambda x: f"{x:.2f}")

else: # Euro formatting

sim\_table\_df[col] = sim\_table\_df[col].apply(lambda x: f"{x:.2f}")

# else: # Handle non-numeric columns if any (shouldn't happen with current data)

# pass # Leave as is

st.dataframe(sim\_table\_df, use\_container\_width=True, hide\_index=True)

else:

st.info(texts.get("simulation\_details\_no\_data", "Simulationsdetails konnten nicht geladen oder berechnet werden.")) # Neuer Text Schlüssel

st.markdown("---") # Trennlinie

# --- Detaillierte Kostenübersicht ---

# Zeige nur an, wenn detaillierte Kosten im results dict vorhanden sind

# ... (This section was moved up for better flow, ensure it's not duplicated) ...

# --- CO2 Details ---

if results.get('annual\_co2\_savings\_kg') is not None:

st.subheader(texts.get("co2\_details\_header", "CO₂ Details"))

st.write(f"{texts.get('annual\_co2\_savings\_kg', 'Jährliche CO₂-Einsparung (kg)')}:\*\* {results.get('annual\_co2\_savings\_kg', 0.0):.2f} kg")

# TODO: Vergleich in Bäume, Diesel-km, Flugreisen implementieren und hier anzeigen (requires more calculation/logic)

st.markdown("---") # Trennlinie

# --- Zukunftsplanung, Szenariensimulationen, Finanzierungsvergleich (Platzhalter mit Subheadern) ---

# Diese Abschnitte bleiben Platzhalter bis zur Implementierung in calculations.py oder separaten Modulen

st.subheader(texts.get("future\_planning\_header", "Zukunftsplanung"))

st.info(texts.get("future\_planning\_info", "Prognosen für zukünftigen Verbrauch und Systemerweiterungen werden hier angezeigt (Platzhalter)."))

st.subheader(texts.get("scenario\_simulation\_header", "Szenariensimulationen"))

st.info(texts.get("scenario\_simulation\_info", "Simulationen unter verschiedenen Bedingungen (Wartung, Strompreise) kommen hierher (Platzhalter)."))

st.subheader(texts.get("financial\_comparison\_header", "Finanzierungsvergleich"))

st.info(texts.get("financial\_comparison\_info", "Vergleich PV-Anlage vs. andere Kapitalanlagen kommt hierher (Platzhalter)."))

**calculations.py:**

# calculations.py

# Enthält Importe, Konstanten und die Logik für die Projektkalkulationen.

import pandas as pd

import numpy as np

import numpy\_financial as financial\_lib

import json

from typing import Dict, Any, List, Optional

import traceback

import os

from datetime import datetime

import re

import streamlit as st # Import Streamlit für session\_state

import math

# Pfad zur Preismatrix (Wird möglicherweise nicht direkt hier verwendet, da die Matrix aus dem DB/Session State kommt)

# BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) # \_\_file\_\_ kann in manchen Umgebungen Probleme machen

# PRICE\_MATRIX\_PATH = os.path.join(BASE\_DIR, 'data', 'price\_matrix.xlsx - Tabelle1.csv')

# Dummy-Texte für Dropdown-Platzhalter

# Diese Liste wird verwendet, um auszuwählen Dropdown-Werte zu erkennen,

# die keine tatsächlichen Produktauswahlen sind.

texts\_placeholder\_values = set([

"--- Bitte wählen ---",

'Keine Module gefunden', 'Fehler',

'Keine Wechselrichter gefunden/geladen', 'Keine Speicher gefunden/geladen',

'Keine Wallboxen gefunden/geladen', 'Keine EMS gefunden/geladen',

'Keine Optimierer gefunden/geladen', 'Keine Carports gefunden/geladen',

'Keine Notstromlösungen gefunden/geladen', 'Keine Tierabwehr gefunden/geladen',

'N/A', # Füge weitere potenziell verwendete Platzhalter hinzu

])

# Datenbank-/Produkt-Module (Standard-Import mit Fallback)

try:

# Versuche die echten Funktionen zu importieren

from database import load\_admin\_setting

from product\_db import (

lookup\_product\_price,

lookup\_module\_details,

lookup\_inverter\_details,

lookup\_storage\_details,

lookup\_wallbox\_details,

lookup\_ems\_details,

lookup\_optimizer\_details,

lookup\_carport\_details,

lookup\_notstrom\_details,

lookup\_tierabwehr\_details

)

db\_functions\_available = True

except ImportError:

# Fallback-Funktionen, falls database oder product\_db nicht verfügbar sind

traceback.print\_exc()

print("WARNUNG: Datenbank- oder Produktmodule nicht gefunden. Benutze Dummy-Funktionen.")

db\_functions\_available = False

# Definiere Dummy-Funktionen, die 0.0 oder None zurückgeben

def load\_admin\_setting(key, default=None): return default

# Passe Rückgabeformate an, falls die aufrufende Funktion Dictionaries erwartet

def lookup\_product\_price(model\_name): return 0.0

def lookup\_module\_details(model\_name): return {'price\_euro': 0.0}

def lookup\_inverter\_details(model\_name): return {'price\_euro': 0.0}

def lookup\_storage\_details(model\_name): return {'price\_euro': 0.0}

def lookup\_wallbox\_details(model\_name): return 0.0

def lookup\_ems\_details(model\_name): return 0.0

def lookup\_optimizer\_details(model\_name): return 0.0

def lookup\_carport\_details(model\_name): return 0.0

def lookup\_notstrom\_details(model\_name): return 0.0

def lookup\_tierabwehr\_details(model\_name): return 0.0

# Standard-Ertragswerte pro kWp pro Jahr in kWh, basierend auf Ausrichtung und Neigung

# Die Neigungswerte sind repräsentativ; die Berechnung findet den nächstgelegenen Wert.

SPECIFIC\_YIELD\_PER\_KWp\_KWH\_YR: Dict[str, Dict[int, float]] = {

'Süd': {0:850, 15:950, 30:1000, 45:980, 60:900},

'Südost':{0:800, 15:920, 30:950, 45:930, 60:850},

'Ost':{0:750, 15:850, 30:880, 45:860, 60:780},

'Südwest':{0:800, 15:920, 30:950, 45:930, 60:850}, # Spiegelbildlich zu Südost

'West':{0:750, 15:850, 30:880, 45:860, 60:780}, # Spiegelbildlich zu Ost

'Nordwest':{0:650, 15:600, 30:550, 45:500, 60:400},

'Nord':{0:600, 15:550, 30:500, 45:450, 60:350},

'Nordost':{0:650, 15:600, 30:550, 45:500, 60:400} # Spiegelbildlich zu Nordwest

}

# Geschätzte Eigenverbrauchsrate basierend auf Speicher vorhanden (True/False)

ESTIMATED\_SELF\_CONSUMPTION\_PERCENT = { False:0.25, True:0.55 } # 25% ohne, 55% mit Speicher

# Standardwerte für mehrjährige Simulationen (Inflation, Preissteigerung, Wartung, Degradation)

ADMIN\_SETTINGS\_MULTIYEAR = {

'simulation\_period\_years': 25, # Simulationszeitraum in Jahren

'default\_elec\_price\_increase\_percent\_pa': 3.0, # Jährliche Strompreissteigerung in %

'default\_inflation\_rate\_percent\_pa': 2.0, # Jährliche Inflationsrate in %

'default\_maintenance\_cost\_fixed\_pa': 100.0, # Jährliche feste Wartungskosten in Euro

'default\_maintenance\_cost\_per\_kwp\_pa': 5.0, # Jährliche Wartungskosten pro kWp in Euro

'default\_degradation\_percent\_pa': 0.5 # Jährliche Leistungsdegradation in %

}

# --- Matrix Parsing Funktion ---

def parse\_price\_matrix\_csv(file\_source: Any) -> Optional[pd.DataFrame]:

"""

Parst eine CSV-Datenquelle (Datei-Pfad, Datei-ähnliches Objekt oder String)

in ein pandas DataFrame der Preis-Matrix Struktur.

Erwartet die Struktur der hochgeladenen Datei ('price\_matrix.xlsx - Tabelle1.csv'):

- Zeile 1: Spalten-Header (Speicher Modelle), beginnend ab der zweiten Spalte.

- Spalte 1: Zeilen-Header (Modul Anzahlen), beginnend ab der zweiten Zeile.

- Versucht verschiedene Trennzeichen (Semikolon, Komma) und Dezimalzeichen (Komma, Punkt).

- Konvertiert den Index (Modul Anzahlen) zu Integer und die Daten (Preise) zu Float.

Args:

file\_source: Die Datenquelle für die CSV (str Pfad, Datei-ähnliches Objekt oder String-Inhalt).

Returns:

Ein pandas DataFrame mit Modul Anzahlen als Index (int) und Speicher Modellen

als Spalten (str), das die Preisdaten (float) enthält.

Gibt None zurück, wenn das Parsen oder die grundlegende Strukturprüfung fehlschlägt.

"""

df = None

parsing\_successful = False

# errors: List[str] = [] # Fehlerliste für Parsing

# Define potenzielle Trennzeichen, Dezimalzeichen und Encodings, die ausprobiert werden

delimiters = [';', ',']

decimal\_separators = [',', '.']

encodings = ['utf-8', 'windows-1252', 'latin-1']

# Schleife durch alle Kombinationen, bis das Parsen erfolgreich ist

for encoding in encodings:

if parsing\_successful: break

for sep in delimiters:

if parsing\_successful: break

for dec in decimal\_separators:

try:

# pandas read\_csv kann die erste Spalte als Index und die erste Zeile als Header verwenden

# header=0 bedeutet, die 1. Zeile (Index 0) als Spaltennamen verwenden.

# index\_col=0 bedeutet, die 1. Spalte (Index 0) als Index verwenden.

df = pd.read\_csv(

file\_source,

header=0, # Nutze die erste Zeile als Spalten-Header

index\_col=0, # Nutze die erste Spalte als Index

sep=sep,

decimal=dec,

encoding=encoding,

engine='python', # 'python' Engine ist robuster bei manchen CSV-Formaten

keep\_default\_na=True, # Behandle Standard NA-Werte wie in pandas üblich

na\_values=[''] # Zusätzliche leere Strings als NaN behandeln

)

# --- Grundlegende Validierung nach dem Parsen ---

# 1. Prüfen, ob DataFrame leer ist

if df.empty:

raise ValueError("Geparster DataFrame ist leer.")

# 2. Index (Modul Anzahlen): Versuchen, zu Integer zu konvertieren

# Fehlerhafte Einträge werden zu NaN

numeric\_index = pd.to\_numeric(df.index, errors='coerce')

# Entferne Zeilen, deren Index nicht in eine Zahl umgewandelt werden konnte (z.g. leere Zeilen, Fußzeilen)

df = df[numeric\_index.notna()].copy() # Wichtig: .copy() um SettingWithCopyWarning zu vermeiden

# Der Index nach dem Filtern ist nun sicher numerisch (oder es gibt keine Zeilen mehr)

numeric\_index\_cleaned = pd.to\_numeric(df.index, errors='coerce')

if df.empty:

raise ValueError("Nach Bereinigung der Indexspalte ist der DataFrame leer.")

# Jetzt den bereinigten Index zu Integer konvertieren

df.index = numeric\_index\_cleaned.astype(int)

# 3. Spalten bereinigen (leere oder Unnamed Spalten entfernen, Leerraum trimmen)

cols = [c.strip() for c in df.columns if isinstance(c, str) and not c.strip().startswith('Unnamed')]

df = df[cols]

# Entferne auch Spalten, die nach der Bereinigung leer sind

df = df.dropna(axis=1, how='all')

if df.empty:

raise ValueError("Nach Bereinigung der Spalten ist der DataFrame leer.")

# 4. Datenzellen zu Numerisch konvertieren und NaN mit 0.0 füllen

df = df.apply(pd.to\_numeric, errors='coerce').fillna(0.0)

# Wenn wir hier ankommen, war das Parsen erfolgreich

parsing\_successful = True

return df # Erfolgreich geparsten DataFrame zurückgeben

except Exception as e:

# Speichere den Fehler für diese Kombination, aber versuche die nächste

# errors.append(f"Parsing-Versuch (Enc: {encoding}, Sep: '{sep}', Dec: '{dec}') fehlgeschlagen: {e}") # Debugging Fehlerlog

df = None # Stelle sicher, dass df None ist, wenn ein Versuch fehlschlägt

continue # Versuche die nächste Kombination

# Wenn die Schleife endet und parsing\_successful immer noch False ist

# print("FEHLER: Konnte Preis-Matrix mit keiner der getesteten Kombinationen parsen.") # Debugging Fehlerlog

# for err in errors: # Debugging: Zeige alle gesammelten Parsing-Fehler

# print(err)

return None # Keiner der Versuche war erfolgreich

# --- Matrix Lookup Funktion ---

def lookup\_matrix\_base\_price(matrix\_df: Optional[pd.DataFrame], module\_quantity: int, storage\_model\_name: Optional[str]) -> float:

"""

Sucht den Basispreis in der Preismatrix DataFrame basierend auf

Modulanzahl (Index) und Speichermodell (Spaltenname).

Ahmt die Logik der Excel-Formel INDEX(MATCH(Qty, Index), MATCH(Model, Columns)) nach,

basierend auf der Struktur der hochgeladenen CSV-Datei.

Args:

matrix\_df: Der pandas DataFrame, der die Preis-Matrix repräsentiert (geladen von parse\_price\_matrix\_csv).

Index muss Modul Anzahlen (int) enthalten, Spalten Speicher Modelle (str).

module\_quantity: Die gewählte Anzahl der PV-Module (muss exakt im DataFrame-Index existieren).

storage\_model\_name: Der gewählte Name des Batteriespeicher-Modells (muss exakt in den DataFrame-Spalten existieren,

inkl. 'No Storage' falls relevant). Kann None oder leer sein.

Returns:

Der gefundene Basispreis aus der Matrix als float.

Gibt 0.0 zurück, wenn die Matrix None/leer ist, die Menge nicht gefunden wird,

das Modell nicht gefunden wird oder der gefundene Wert ungültig ist.

"""

if matrix\_df is None or matrix\_df.empty:

# print("DEBUG: lookup\_matrix\_base\_price: Matrix DataFrame ist None oder leer.") # Debugging-Info

return 0.0 # Kann keinen Preis finden

# Bereinige den Speichermodellnamen für den Spalten-Lookup

# Handle None oder leeren String, indem wir versuchen, eine passende Spalte zu finden

storage\_model\_key = str(storage\_model\_name).strip() if storage\_model\_name is not None else ''

# Finde die Spalte, die dem Speichermodell entspricht (exakter Match, case-insensitive nach Bereinigung?)

# Basierend auf Excel MATCH(..., Range, 0) brauchen wir einen exakten Spaltennamen.

# Die Spaltennamen im DataFrame sind bereits Strings (nach parse\_price\_matrix\_csv).

# Machen wir den Lookup case-insensitiv zur Sicherheit, falls das gewünscht ist, aber verwenden den echten Spaltennamen.

# Finden des echten Spaltennamens im DF (case-insensitiv match auf bereinigtem Namen)

matching\_columns = [col for col in matrix\_df.columns if col.strip().lower() == storage\_model\_key.lower()]

# Behandle den Fall, dass kein Speicher gewählt wurde oder der Name leer ist oder ein Platzhalter ist

lookup\_col\_name = None

if not storage\_model\_key or storage\_model\_key.lower() in (name.lower() for name in texts\_placeholder\_values): # Wenn der Eingabename leer, None oder ein Platzhalter war

# Suche nach potenziellen "Ohne Speicher"-Spaltennamen (case-insensitive)

potential\_nostorage\_cols = [col for col in matrix\_df.columns if col.strip().lower() in ['no storage', 'ohne speicher', '-']] # Annahme gängiger Namen

if potential\_nostorage\_cols:

# Verwenden Sie die erste passende "Ohne Speicher" Spalte

lookup\_col\_name = potential\_nostorage\_cols[0]

# print(f"DEBUG: Keine Speicherwahl oder leer, Fallback auf Spalte '{lookup\_col\_name}'.") # Debugging

elif matching\_columns:

# Wenn ein Name eingegeben wurde und eine passende Spalte gefunden wurde

lookup\_col\_name = matching\_columns[0] # Verwenden Sie den echten Spaltennamen aus dem DataFrame

# print(f"DEBUG: Gefundene Spalte für Modell '{storage\_model\_key}': '{lookup\_col\_name}'.") # Debugging

# Prüfe, ob eine gültige Spalte zum Nachschlagen gefunden wurde

if lookup\_col\_name is None or lookup\_col\_name not in matrix\_df.columns:

# print(f"lookup\_matrix\_base\_price: Konnte keine passende Matrix-Spalte für Speichermodell '{storage\_model\_key if storage\_model\_key else 'Kein Speicher'}' finden.") # Debugging

return 0.0 # Keine passende Spalte gefunden => 0 Preis aus Matrix

# Finde die passende Zeile basierend auf der Modulanzahl (exakter Match im Index)

# Der Index ist bereits Integer (nach parse\_price\_matrix\_csv)

if module\_quantity not in matrix\_df.index:

# print(f"DEBUG: Modulanzahl {module\_quantity} nicht exakt im Matrix-Index gefunden.") # Debugging

# Optional: Nächste passende Menge finden? Bedarf klären.

# Für jetzt: Exakter Match wie in Excel MATCH(..., 0)

return 0.0 # Exakte Menge nicht in der Matrix definiert => 0 Preis

# Hole den Preis aus der gefundenen Zeile (Modulanzahl Index) und Spalte (Speichermodell Name)

try:

# .loc verwendet die Labels des Index (int) und der Spalten (str)

base\_price = matrix\_df.loc[module\_quantity, lookup\_col\_name]

# Stelle sicher, dass der Preis eine Zahl ist (fillna sollte NaN verhindern)

if pd.isna(base\_price):

# Sollte nach fillna(0.0) nicht mehr passieren, aber als Absicherung

# print(f"WARNUNG: Matrix-Preis für Menge={module\_quantity}, Modell='{lookup\_col\_name}' ist NaN nach Laden.") # Debugging

return 0.0

# Konvertiere zu float und gib zurück

return float(base\_price)

except KeyError:

# Sollte nicht auftreten, wenn vorherige Prüfungen korrekt waren, aber als Absicherung

print(f"FEHLER: KeyError während loc-Aufrufs für Menge={module\_quantity}, Modell='{lookup\_col\_name}'.")

traceback.print\_exc()

return 0.0 # Fehler beim Abrufen => kein Matrix Preis

except Exception as e:

print(f"FEHLER: Unerwarteter Fehler während des Matrix-Lookups: {e}") # Fehlerlog

traceback.print\_exc()

return 0.0 # Bei jedem anderen Fehler 0.0 zurückgeben

# --- Hauptfunktion zur Durchführung aller Berechnungen ---

def perform\_calculations(project\_data: Dict[str, Any]) -> Dict[str, Any]:

"""

Führt alle Berechnungen für ein PV-Projekt unter Berücksichtigung der Preismatrix durch.

Holt die Preis-Matrix aus dem Streamlit Session State.

Args:

project\_data: Dictionary mit allen gesammelten Eingaben aus data\_input.py

Returns:

Dictionary mit allen berechneten Ergebnissen und KPIs.

"""

# print("perform\_calculations() aufgerufen") # Debugging

results: Dict[str, Any] = {}

calculation\_errors: List[str] = [] # Liste zur Sammlung von Fehlern während der Berechnung

# --- Hole Admin Settings (für Tarife, CO2, ggf. zukünftige Margen) ---

# Nutzt load\_admin\_setting mit Fallback auf ADMIN\_SETTINGS\_MULTIYEAR oder Hardcoded Defaults

feed\_in\_tariffs = load\_admin\_setting('feed\_in\_tariffs', {}) # Default leeres Dict

co2\_factor\_g\_per\_kwh\_grid = load\_admin\_setting('co2\_factor\_grid\_g\_per\_kwh', 388.0) # Default 388 g/kWh (Stand 2022)

default\_vat\_rate\_percent = load\_admin\_setting('default\_vat\_rate\_percent', 19.0) # Default 19%

pv\_0\_vat\_threshold\_kwp = load\_admin\_setting('pv\_0\_vat\_threshold\_kwp', 30.0) # Default 30 kWp für 0% MwSt

# Multi-Year Simulation Settings (mit Fallback auf ADMIN\_SETTINGS\_MULTIYEAR)

simulation\_period\_years = int(load\_admin\_setting('simulation\_period\_years', ADMIN\_SETTINGS\_MULTIYEAR['simulation\_period\_years']))

elec\_price\_increase\_rate = load\_admin\_setting('default\_elec\_price\_increase\_percent\_pa', ADMIN\_SETTINGS\_MULTIYEAR['default\_elec\_price\_increase\_percent\_pa']) / 100.0

inflation\_rate = load\_admin\_setting('default\_inflation\_rate\_percent\_pa', ADMIN\_SETTINGS\_MULTIYEAR['default\_inflation\_rate\_percent\_pa']) / 100.0

maintenance\_cost\_fixed\_pa = load\_admin\_setting('default\_maintenance\_cost\_fixed\_pa', ADMIN\_SETTINGS\_MULTIYEAR['default\_maintenance\_cost\_fixed\_pa'])

maintenance\_cost\_per\_kwp\_pa = load\_admin\_setting('default\_maintenance\_cost\_per\_kwp\_pa', ADMIN\_SETTINGS\_MULTIYEAR['default\_maintenance\_cost\_per\_kwp\_pa'])

degradation\_rate = load\_admin\_setting('default\_degradation\_percent\_pa', ADMIN\_SETTINGS\_MULTIYEAR['default\_degradation\_percent\_pa']) / 100.0

# TODO: Load WACC or discount rate from Admin settings for NPV if different from inflation

# Load additional standard costs from Admin Settings

standard\_costs\_geruest\_pauschal = load\_admin\_setting('standard\_costs\_geruest\_pauschal', 0.0) # Beispiel: Gerüstkosten Pauschal

standard\_costs\_complex\_roof\_install\_pauschal = load\_admin\_setting('standard\_costs\_complex\_roof\_install\_pauschal', 0.0) # Beispiel: Kosten komplexes Dach

standard\_misc\_cost\_pauschal = load\_admin\_setting('default\_misc\_cost\_pauschal', 0.0) # Verschiedene Kosten Pauschal

# --- Hole notwendige Eingabedaten aus project\_data ---

project\_details = project\_data.get('project\_details', {})

# customer\_data = project\_data.get('customer\_data', {}) # Derzeit nicht für Berechnungen benötigt

anlage\_kwp = project\_details.get('anlage\_kwp', 0.0) # Kommt aus data\_input (Modulanzahl \* Modulleistung)

module\_quantity = project\_details.get('module\_quantity', 0)

selected\_storage\_name = project\_details.get('selected\_storage\_name') # Kann None sein

selected\_module\_name = project\_details.get('selected\_module\_name') # Kann None sein

selected\_inverter\_name = project\_details.get('selected\_inverter\_name') # Kann None sein

include\_additional\_components = project\_details.get('include\_additional\_components', False) # Checkbox für Zubehör

# Hole Namen der optionalen Zubehör-Komponenten aus project\_details (falls checkbox aktiv)

selected\_wallbox\_name = project\_details.get('selected\_wallbox\_name') if include\_additional\_components else None

selected\_ems\_name = project\_details.get('selected\_ems\_name') if include\_additional\_components else None

selected\_optimizer\_name = project\_details.get('selected\_optimizer\_name') if include\_additional\_components else None

selected\_carport\_name = project\_details.get('selected\_carport\_name') if include\_additional\_components else None

selected\_notstrom\_name = project\_details.get('selected\_notstrom\_name') if include\_additional\_components else None

selected\_tierabwehr\_name = project\_details.get('selected\_tierabwehr\_name') if include\_additional\_components else None

# Verbrauchs- und Ertragsdaten (für finanzielle Vorteile)

total\_annual\_consumption\_kwh = project\_details.get('total\_consumption\_kwh\_yr', 0.0)

electricity\_price\_kwh = project\_details.get('electricity\_price\_kwh', 0.0)

feed\_in\_type = project\_details.get('feed\_in\_type', 'Teileinspeisung')

roof\_orientation = project\_details.get('roof\_orientation', 'Süd')

roof\_inclination\_deg = project\_details.get('roof\_inclination\_deg', 30)

include\_storage = project\_details.get('include\_storage', False) # Speicher nur relevant für Eigenverbrauchsrate

# Additional Cost Inputs (assuming these come from data\_input.py now)

building\_height\_gt\_7m = project\_details.get('building\_height\_gt\_7m', False)

is\_complex\_roof = project\_details.get('is\_complex\_roof', False) # Angenommenes Eingabefeld für komplexes Dach

custom\_cost\_euro = project\_details.get('custom\_cost\_euro', 0.0) # Angenommenes Eingabefeld für benutzerdefinierte Kosten

custom\_surcharge\_percent = project\_details.get('custom\_surcharge\_percent', 0.0) # Angenommenes Eingabefeld für prozentualen Aufschlag

custom\_discount\_percent = project\_data.get('custom\_discount\_percent', 0.0) # Angenommenes Eingabefeld für prozentualen Rabatt (aus project\_data holen)

overfactor = project\_data.get('overfactor', 1.0) # Angenommenes Eingabefeld für Überfaktor (aus project\_data holen)

# --- 1. Kosten der Anlage (Matrix-basiert + Aufpreise) ---

base\_matrix\_price = 0.0

cost\_modules\_aufpreis = 0.0

cost\_inverter\_aufpreis = 0.0

cost\_storage\_aufpreis\_product\_db = 0.0

cost\_accessories\_aufpreis = 0.0

# Additional Costs (beyond base matrix and product lookups)

cost\_scaffolding\_netto = 0.0

cost\_complex\_roof\_install\_netto = 0.0

cost\_misc\_netto = 0.0

cost\_custom\_netto = custom\_cost\_euro # Direkte Addition der benutzerdefinierten Kosten

try: # <-- Keep the main try block for cost calculation errors

# Hole die Preismatrix aus dem Streamlit Session State

# Dies erfordert, dass die Matrix zuvor über gui.py oder admin\_panel.py geladen und gespeichert wurde!

# Wenn die Matrix nicht im Session State ist, wird lookup\_matrix\_base\_price 0.0 zurückgeben.

price\_matrix\_df = st.session\_state.get('price\_matrix\_df')

# print(f"DEBUG: Preis-Matrix aus Session State geholt: {'Verfügbar' if price\_matrix\_df is not None else 'Nicht verfügbar'}") # Debugging

if price\_matrix\_df is None or price\_matrix\_df.empty:

# Fehler, wenn Matrix nicht verfügbar ist

calculation\_errors.append("Kostenkalkulation fehlgeschlagen: Preis-Matrix ist nicht im System geladen oder leer. Bitte im Admin Bereich prüfen.")

# Gehe weiter mit 0 Kosten, aber logge den Fehler

price\_matrix\_df = pd.DataFrame() # Setze einen leeren DF, um nachfolgende Fehler in Lookups zu vermeiden

# Ermittle den Hauptbetrag aus der Matrix (jetzt immer nach dem Lade-Check)

# Der Matrixpreis wird anhand von Modulanzahl und Speichermodell gesucht.

base\_matrix\_price = lookup\_matrix\_base\_price(price\_matrix\_df, module\_quantity, selected\_storage\_name)

# Überprüfe, ob der Matrixpreis 0.0 ist, obwohl Module gewählt wurden und (falls Storage gewählt) eine Matrix-Spalte existieren sollte

# Die lookup\_matrix\_base\_price Funktion loggt bereits spezifische Lookup-Fehler.

# Hier fassen wir zusammen, falls der Endpreis 0.0 ist und das unerwartet ist.

if base\_matrix\_price == 0.0 and module\_quantity > 0:

# Prüfe, ob ein Speicher gewählt war und NICHT gefunden wurde, ODER kein Speicher gewählt war und KEINE 'Ohne Speicher' Spalte existiert

storage\_key = str(selected\_storage\_name).strip() if selected\_storage\_name is not None else ''

matrix\_cols\_lower = {col.strip().lower(): col for col in price\_matrix\_df.columns}

storage\_col\_found = storage\_key.lower() in matrix\_cols\_lower or (not storage\_key and any(col.strip().lower() in ['no storage', 'ohne speicher', '-'] for col in price\_matrix\_df.columns))

if not storage\_col\_found or module\_quantity not in price\_matrix\_df.index:

# Fehler, wenn Menge oder Modell nicht gefunden wurde, was zu 0.0 führt

if module\_quantity not in price\_matrix\_df.index:

calculation\_errors.append(f"Kostenkalkulation: Modulanzahl {module\_quantity} nicht exakt im Index der geladenen Preis-Matrix gefunden.")

if not storage\_col\_found:

calculation\_errors.append(f"Kostenkalkulation: Speichermodell '{selected\_storage\_name if selected\_storage\_name else 'Kein Speicher gewählt'}' nicht in den Spalten der geladenen Preis-Matrix gefunden (inkl. 'Ohne Speicher').")

# Wenn der Preis 0.0 ist, obwohl Menge und Modell/Spalte gefunden wurden, ist der Wert in der Matrix 0.0

# Dies könnte ein legitimer Preis sein, daher kein Fehler hinzufügen.

# --- Berechne die optionalen Aufpreise (Preise der spezifischen Komponenten aus Produktdatenbank) ---

# Lade Preise der SPEZIFISCH ausgewählten Komponenten aus der Produktdatenbank

# Nutzt die oben importierten lookup\_\*\_details Funktionen und lookup\_product\_price

# Stelle sicher, dass die Namen nicht None oder Platzhalter sind vor dem Lookup

# Aufpreis PV Module: Preis des SPEZIFISCH gewählten Modells

# Annahme: Matrixpreis ist für ein "Standard"-Modul. Wenn ein anderes gewählt, ist dessen Preis der Aufpreis.

# TODO: Hier könnte die Logik komplexer sein (Basis-Modulpreis aus Matrix vs. gewähltes Modul)

if selected\_module\_name and str(selected\_module\_name).strip() not in texts\_placeholder\_values:

module\_details = lookup\_module\_details(selected\_module\_name)

if module\_details and 'price\_euro' in module\_details:

cost\_modules\_aufpreis = module\_details.get('price\_euro', 0.0) \* module\_quantity # Preis pro Modul \* Anzahl

# Wende ggf. Admin-Marge an (Zukunft)

# cost\_modules\_aufpreis \*= (1 + module\_markup\_percent / 100.0)

else:

# Fehler, wenn Modulname gewählt, aber Details/Preis nicht gefunden (nach DB-Lookup)

calculation\_errors.append(f"Kostenkalkulation: Details/Preis für gewähltes Modul '{selected\_module\_name}' nicht in Produktdatenbank gefunden oder Preis fehlt.")

cost\_modules\_aufpreis = 0.0 # Setze Aufpreis auf 0 im Fehlerfall

# Aufpreis Wechselrichter: Preis des SPEZIFISCH gewählten Modells

# Annahme: Matrixpreis ist für einen "Standard"-Wechselrichter. Preis des GEWÄHLTEN WR ist der Aufpreis.

# TODO: Hier könnte die Logik komplexer sein (Basis-WR aus Matrix vs. gewählter WR)

if selected\_inverter\_name and str(selected\_inverter\_name).strip() not in texts\_placeholder\_values:

inverter\_details = lookup\_inverter\_details(selected\_inverter\_name)

if inverter\_details and 'price\_euro' in inverter\_details:

cost\_inverter\_aufpreis = inverter\_details.get('price\_euro', 0.0)

# Wende ggf. Admin-Marge an (Zukunft)

# cost\_inverter\_aufpreis \*= (1 + inverter\_markup\_percent / 100.0)

else:

# Fehler, wenn WR Name gewählt, aber Details/Preis nicht gefunden

calculation\_errors.append(f"Kostenkalkulation: Details/Preis für gewählten Wechselrichter '{selected\_inverter\_name}' nicht in Produktdatenbank gefunden oder Preis fehlt.")

cost\_inverter\_aufpreis = 0.0

# Aufpreis Batteriespeicher: Preis des SPEZIFISCH gewählten Modells (aus Produktdatenbank)

# ACHTUNG: Dies addiert den Preis des SPEZIFISCHEN Speichers ZUM Matrix-Basispreis,

# der bereits durch die Wahl des Speichermodells aus der Matrix ermittelt wurde!

# DIES KANN EINE POTENZIELLE DOPPELT-ZÄHLUNG SEIN, abhängig davon, wie die Preis-Matrix definiert ist.

# WENN DIE MATRIX EINEN PAKETPREIS INKL. SPEICHER ENTHÄLT, SOLLTE DIESER AUFPREIS HIER WEGGELASSEN

# ODER ALS DIFFERENZ ZUM "Standard"-Speicher in der Matrix berechnet werden.

# Wir implementieren es hier gemäß deiner Struktur (Matrix Basis + Produkt-Aufpreise),

# aber diese Logik muss evtl. angepasst werden, je nach genauer Bedeutung der Matrix-Werte.

if include\_storage and selected\_storage\_name and str(selected\_storage\_name).strip() not in texts\_placeholder\_values:

storage\_details = lookup\_storage\_details(selected\_storage\_name)

if storage\_details and 'price\_euro' in storage\_details:

# Annahme: Preis im Produkt-DB ist der Preis des gewählten Speichermodells.

# Komplexität der Kapazität (mehrere Module?) wird hier ignoriert.

cost\_storage\_aufpreis\_product\_db = storage\_details.get('price\_euro', 0.0)

# Wende ggf. Admin-Marge an (Zukunft)

# cost\_storage\_aufpreis\_product\_db \*= (1 + storage\_markup\_percent / 100.0)

else:

# Fehler, wenn Speicher Name gewählt, aber Details/Preis nicht gefunden

calculation\_errors.append(f"Kostenkalkulation: Details/Preis für gewähltes Speichermodell '{selected\_storage\_name}' nicht in Produktdatenbank gefunden oder Preis fehlt.")

cost\_storage\_aufpreis\_product\_db = 0.0

# Aufpreise für weiteres Zubehör (Wallbox, EMS, etc.)

# Lade Preise nur, wenn das Zubehör ausgewählt wurde (und Checkbox "include\_additional\_components" aktiv war)

accessory\_prices: List[float] = []

# Helper function to look up accessory price and add error if not found or price is 0

def add\_accessory\_cost(name: Optional[str], accessory\_type: str):

if name and str(name).strip() not in texts\_placeholder\_values:

price = lookup\_product\_price(name)

if price > 0:

accessory\_prices.append(price)

elif price == 0:

calculation\_errors.append(f"Kostenkalkulation: Preis für {accessory\_type} '{name}' in Produktdatenbank ist 0.")

else:

calculation\_errors.append(f"Kostenkalkulation: Preis für {accessory\_type} '{name}' nicht in Produktdatenbank gefunden.")

add\_accessory\_cost(selected\_wallbox\_name, 'Wallbox')

add\_accessory\_cost(selected\_ems\_name, 'EMS')

add\_accessory\_cost(selected\_optimizer\_name, 'Optimierer')

add\_accessory\_cost(selected\_carport\_name, 'Carport')

add\_accessory\_cost(selected\_notstrom\_name, 'Notstrom')

add\_accessory\_cost(selected\_tierabwehr\_name, 'Tierabwehr')

# Summe der Aufpreise für Zubehör

cost\_accessories\_aufpreis = sum(accessory\_prices)

# Wende ggf. Admin-Marge für Zubehör an (Zukunft)

# cost\_accessories\_aufpreis \*= (1 + accessory\_markup\_percent / 100.0)

# Summe der Produkt-spezifischen Aufpreise (Module, Inverter, Storage aus Produkt DB)

# HINWEIS: Potenzielle Doppeltzählung Speicher beachten!

# Wenn die Matrix den Speicherpreis enthält, sollte cost\_storage\_aufpreis\_product\_db hier NICHT addiert werden.

# Für jetzt, basierend auf deiner Struktur, addieren wir es, aber das ist ein Designpunkt zur Klärung.

cost\_product\_surcharges\_netto = cost\_modules\_aufpreis + cost\_inverter\_aufpreis + cost\_storage\_aufpreis\_product\_db

# --- Berechne zusätzliche Installations- und sonstige Kosten ---

# Gerüstkosten: Pauschal, wenn Gebäude > 7m (angenommenes Kriterium)

if building\_height\_gt\_7m:

cost\_scaffolding\_netto = standard\_costs\_geruest\_pauschal

# Kosten für komplexes Dach: Pauschal, wenn als komplex markiert

if is\_complex\_roof:

cost\_complex\_roof\_install\_netto = standard\_costs\_complex\_roof\_install\_pauschal

# Verschiedene Kosten: Pauschal aus Admin Settings

cost\_misc\_netto = standard\_misc\_cost\_pauschal

# Benutzerdefinierte Kosten: Direkte Addition aus Input

cost\_custom\_netto = custom\_cost\_euro

# Summe aller zusätzlichen Kosten (Scaffolding, Complex Roof, Misc, Custom)

total\_additional\_costs\_netto = cost\_scaffolding\_netto + cost\_complex\_roof\_install\_netto + cost\_misc\_netto + cost\_custom\_netto

# --- Berechne den Netto-Zwischensumme ---

# Zwischensumme Netto = Matrix Basis + Produkt Aufpreise + Zubehör Aufpreise + Zusätzliche Kosten

subtotal\_netto = base\_matrix\_price + cost\_product\_surcharges\_netto + cost\_accessories\_aufpreis + total\_additional\_costs\_netto

# --- Wende prozentuale Aufschläge / Rabatte und Überfaktor an ---

# Berechne prozentuale Aufschläge / Rabatte

custom\_surcharge\_amount\_netto = subtotal\_netto \* (custom\_surcharge\_percent / 100.0)

custom\_discount\_amount\_netto = subtotal\_netto \* (custom\_discount\_percent / 100.0)

subtotal\_after\_surcharge\_discount = subtotal\_netto + custom\_surcharge\_amount\_netto - custom\_discount\_amount\_netto

# Wende Überfaktor an

total\_investment\_netto = subtotal\_after\_surcharge\_discount \* overfactor

# --- Berechne Brutto-Betrag (mit MwSt) ---

vat\_rate\_percent = 0.0 # Standard 0%

# MwSt nur über dem Schwellenwert anwenden

if anlage\_kwp > pv\_0\_vat\_threshold\_kwp:

vat\_rate\_percent = default\_vat\_rate\_percent

vat\_rate = vat\_rate\_percent / 100.0

total\_investment\_brutto = total\_investment\_netto \* (1 + vat\_rate)

# --- Speichere detaillierte Kosten im results Dictionary ---

results['base\_matrix\_price\_netto'] = base\_matrix\_price

results['cost\_modules\_aufpreis\_netto'] = cost\_modules\_aufpreis

results['cost\_inverter\_aufpreis\_netto'] = cost\_inverter\_aufpreis

results['cost\_storage\_aufpreis\_product\_db\_netto'] = cost\_storage\_aufpreis\_product\_db

results['cost\_accessories\_aufpreis\_netto'] = cost\_accessories\_aufpreis

results['cost\_product\_surcharges\_netto'] = cost\_product\_surcharges\_netto

results['cost\_scaffolding\_netto'] = cost\_scaffolding\_netto

results['cost\_complex\_roof\_install\_netto'] = cost\_complex\_roof\_install\_netto

results['cost\_misc\_netto'] = cost\_misc\_netto

results['cost\_custom\_netto'] = cost\_custom\_netto

results['total\_additional\_costs\_netto'] = total\_additional\_costs\_netto

results['subtotal\_netto'] = subtotal\_netto

results['custom\_surcharge\_percent'] = custom\_surcharge\_percent

results['custom\_discount\_percent'] = custom\_discount\_percent

results['custom\_surcharge\_amount\_netto'] = custom\_surcharge\_amount\_netto

results['custom\_discount\_amount\_netto'] = custom\_discount\_amount\_netto

results['overfactor'] = overfactor

results['total\_investment\_netto'] = total\_investment\_netto

results['total\_investment\_brutto'] = total\_investment\_brutto

results['vat\_rate\_percent'] = vat\_rate\_percent # Speichere den Prozentwert

except Exception as e: # Catch errors in the entire cost block

calculation\_errors.append(f"Fehler bei Kostenkalkulation (Gesamt): {e}")

traceback.print\_exc()

# Set all cost results to 0.0 or default on error

results['base\_matrix\_price\_netto'] = 0.0

results['cost\_modules\_aufpreis\_netto'] = 0.0

results['cost\_inverter\_aufpreis\_netto'] = 0.0

results['cost\_storage\_aufpreis\_product\_db\_netto'] = 0.0

results['cost\_accessories\_aufpreis\_netto'] = 0.0

results['cost\_product\_surcharges\_netto'] = 0.0

results['cost\_scaffolding\_netto'] = 0.0

results['cost\_complex\_roof\_install\_netto'] = 0.0

results['cost\_misc\_netto'] = 0.0

results['cost\_custom\_netto'] = 0.0

results['total\_additional\_costs\_netto'] = 0.0

results['subtotal\_netto'] = 0.0

results['custom\_surcharge\_percent'] = custom\_surcharge\_percent # Keep original input rate

results['custom\_discount\_percent'] = custom\_discount\_percent # Keep original input rate

results['custom\_surcharge\_amount\_netto'] = 0.0 # Calculated amount is 0 on error

results['custom\_discount\_amount\_netto'] = 0.0 # Calculated amount is 0 on error

results['overfactor'] = 1.0 # Default to 1.0 on error

results['total\_investment\_netto'] = 0.0

results['total\_investment\_brutto'] = 0.0

results['vat\_rate\_percent'] = 0.0

# --- 2. PV Ertrag Schätzung ---

try:

anlage\_kwp\_value = project\_details.get('anlage\_kwp', 0.0)

roof\_orientation = project\_details.get('roof\_orientation', 'Süd')

roof\_inclination\_deg = project\_details.get('roof\_inclination\_deg', 30)

orientation\_yields = SPECIFIC\_YIELD\_PER\_KWp\_KWH\_YR.get(roof\_orientation, SPECIFIC\_YIELD\_PER\_KWp\_KWH\_YR["Süd"])

if orientation\_yields:

# Finde die Neigung in den keys, die am nächsten zur eingegebenen Neigung liegt

nearest\_inclination = min(orientation\_yields.keys(), key=lambda x: abs(x - roof\_inclination\_deg))

specific\_yield\_per\_kwp = orientation\_yields.get(nearest\_inclination, 900.0) # Fallback auf 900, falls key nicht gefunden (unwahrscheinlich)

else:

specific\_yield\_per\_kwp = 900.0 # Fallback, falls Ausrichtung nicht gefunden

annual\_pv\_production\_kwh = anlage\_kwp\_value \* specific\_yield\_per\_kwp

results['specific\_yield\_kwh\_kwp\_yr'] = specific\_yield\_per\_kwp

results['annual\_pv\_production\_kwh'] = annual\_pv\_production\_kwh

results['anlage\_kwp'] = anlage\_kwp\_value

except Exception as e: # Catch errors in the entire yield block

calculation\_errors.append(f"Fehler bei Ertragsschätzung: {e}")

traceback.print\_exc()

results['specific\_yield\_kwh\_kwp\_yr'] = 0.0

results['annual\_pv\_production\_kwh'] = 0.0

results['anlage\_kwp'] = project\_details.get('anlage\_kwp', 0.0) # Speichere den Original-Input-Wert auch im Fehlerfall

# --- 3. Eigenverbrauch und Einspeisung ---

try:

total\_annual\_consumption\_kwh = project\_details.get('total\_consumption\_kwh\_yr', 0.0)

annual\_pv\_production\_kwh = results.get('annual\_pv\_production\_kwh', 0.0)

include\_storage = project\_details.get('include\_storage', False)

# Bestimme die Eigenverbrauchsrate basierend auf Speicher vorhanden

estimated\_self\_consumption\_rate = ESTIMATED\_SELF\_CONSUMPTION\_PERCENT.get(include\_storage, 0.25)

# Berechne Eigenverbrauch und Einspeisung.

# Der Eigenverbrauch ist das Minimum aus der geschätzten Rate der Produktion und dem tatsächlichen Verbrauch.

annual\_self\_consumption\_kwh = annual\_pv\_production\_kwh \* estimated\_self\_consumption\_rate

annual\_self\_consumption\_kwh = min(annual\_self\_consumption\_kwh, total\_annual\_consumption\_kwh, annual\_pv\_production\_kwh)

# Stellen Sie sicher, dass Eigenverbrauch nicht negativ ist

annual\_self\_consumption\_kwh = max(0.0, annual\_self\_consumption\_kwh)

annual\_feed\_in\_kwh = annual\_pv\_production\_kwh - annual\_self\_consumption\_kwh

# Stellen Sie sicher, dass Einspeisung nicht negativ ist

annual\_feed\_in\_kwh = max(0.0, annual\_feed\_in\_kwh)

annual\_grid\_consumption\_kwh = total\_annual\_consumption\_kwh - annual\_self\_consumption\_kwh

# Stellen Sie sicher, dass Netzbezug nicht negativ ist

annual\_grid\_consumption\_kwh = max(0.0, annual\_grid\_consumption\_kwh)

results['estimated\_self\_consumption\_rate'] = estimated\_self\_consumption\_rate

results['annual\_self\_consumption\_kwh'] = annual\_self\_consumption\_kwh

results['annual\_feed\_in\_kwh'] = annual\_feed\_in\_kwh

results['annual\_grid\_consumption\_kwh'] = annual\_grid\_consumption\_kwh

# Eigenversorgungsrate (%) = Eigenverbrauch / Gesamtverbrauch

self\_supply\_rate\_percent = (annual\_self\_consumption\_kwh / total\_annual\_consumption\_kwh) \* 100.0 if total\_annual\_consumption\_kwh > 0 else 0.0

results['self\_supply\_rate\_percent'] = self\_supply\_rate\_percent

results['total\_consumption\_kwh\_yr'] = total\_annual\_consumption\_kwh # Speichere den Original-Input-Wert

# Autarkiegrad (%) = Eigenverbrauch / Gesamtproduktion

autarky\_rate\_percent = (annual\_self\_consumption\_kwh / annual\_pv\_production\_kwh) \* 100.0 if annual\_pv\_production\_kwh > 0 else 0.0

results['autarky\_rate\_percent'] = autarky\_rate\_percent

except Exception as e: # Catch errors in the entire consumption split block

calculation\_errors.append(f"Fehler bei Verbrauchsverteilung: {e}")

traceback.print\_exc()

results['estimated\_self\_consumption\_rate'] = 0.0

results['annual\_self\_consumption\_kwh'] = 0.0

results['annual\_feed\_in\_kwh'] = 0.0

results['annual\_grid\_consumption\_kwh'] = 0.0

results['self\_supply\_rate\_percent'] = 0.0

results['autarky\_rate\_percent'] = 0.0

results['total\_consumption\_kwh\_yr'] = project\_details.get('total\_consumption\_kwh\_yr', 0.0) # Speichere den Original-Input-Wert

# --- 4. Jährliche Ersparnis und Einnahmen ---

try:

electricity\_price\_kwh = project\_details.get('electricity\_price\_kwh', 0.0)

feed\_in\_type = project\_details.get('feed\_in\_type', 'Teileinspeisung')

anlage\_kwp\_value = results.get('anlage\_kwp', 0.0) # Nutze den berechneten/bestätigten kWp Wert

feed\_in\_tariffs = load\_admin\_setting('feed\_in\_tariffs', {})

# Bestimme den korrekten Tarifschlüssel basierend auf Anlagetyp und Anlagengröße (kWp)

tariff\_key = None

if anlage\_kwp\_value > 0:

if feed\_in\_type == "Volleinspeisung":

if anlage\_kwp\_value <= 10:

tariff\_key = '<10kWp\_Voll'

elif anlage\_kwp\_value <= 40:

tariff\_key = '>10kWp<=40kWp\_Voll'

elif anlage\_kwp\_value <= 100:

tariff\_key = '>40kWp<=100kWp\_Voll'

else: # > 100 kWp

tariff\_key = '>100kWp\_Voll'

else: # Teileinspeisung

if anlage\_kwp\_value <= 10:

tariff\_key = '<10kWp\_Teileinspeisung'

elif anlage\_kwp\_value <= 40:

tariff\_key = '>10kWp<=40kWp\_Teileinspeisung'

elif anlage\_kwp\_value <= 100:

tariff\_key = '>40kWp<=100kWp\_Teileinspeisung'

else: # > 100 kWp

tariff\_key = '>100kWp\_Teileinspeisung'

feed\_in\_tariff\_euro\_kwh = feed\_in\_tariffs.get(tariff\_key, 0.0)

if feed\_in\_tariff\_euro\_kwh == 0.0 and anlage\_kwp\_value > 0 and tariff\_key is not None:

# Füge einen Fehler hinzu, wenn kein passender Einspeisetarif für die Anlagengröße und den Typ gefunden wurde (aber ein Schlüssel generiert wurde)

calculation\_errors.append(f"Kein passender Einspeisetarif für Anlagetyp '{feed\_in\_type}' und Größe {anlage\_kwp\_value} kWp (Schlüssel '{tariff\_key}') in den Admin-Einstellungen gefunden. Standard 0.0 €/kWh verwendet.")

elif anlage\_kwp\_value > 0 and tariff\_key is None:

# Dieser Fall sollte durch die obige Logik vermieden werden, aber als Absicherung

calculation\_errors.append(f"Interner Fehler: Konnte keinen Einspeisetarif-Schlüssel für Anlagengröße {anlage\_kwp\_value} kWp und Typ '{feed\_in\_type}' generieren.")

# Jährliche Ersparnis durch Eigenverbrauch = Eigenverbrauch (kWh) \* Strompreis (Euro/kWh)

annual\_savings\_eigenverbrauch = results.get('annual\_self\_consumption\_kwh', 0.0) \* electricity\_price\_kwh

# Jährliche Einnahmen durch Einspeisung = Einspeisung (kWh) \* Einspeisetarif (Euro/kWh)

annual\_revenue\_einspeisung = results.get('annual\_feed\_in\_kwh', 0.0) \* feed\_in\_tariff\_euro\_kwh

# Gesamter jährlicher finanzieller Vorteil

annual\_financial\_benefit\_year1 = annual\_savings\_eigenverbrauch + annual\_revenue\_einspeisung

results['feed\_in\_tariff\_euro\_kwh'] = feed\_in\_tariff\_euro\_kwh

results['annual\_savings\_eigenverbrauch'] = annual\_savings\_eigenverbrauch

results['annual\_revenue\_einspeisung'] = annual\_revenue\_einspeisung

results['annual\_financial\_benefit\_year1'] = annual\_financial\_benefit\_year1 # Vorteil im 1. Jahr

results['electricity\_price\_kwh'] = electricity\_price\_kwh # Speichere den Original-Input-Wert

except Exception as e: # Catch errors in the entire financial block (Year 1)

calculation\_errors.append(f"Fehler bei Ersparnis & Einnahmen Berechnung (Jahr 1): {e}")

traceback.print\_exc()

results['feed\_in\_tariff\_euro\_kwh'] = 0.0

results['annual\_savings\_eigenverbrauch'] = 0.0

results['annual\_revenue\_einspeisung'] = 0.0

results['annual\_financial\_benefit\_year1'] = 0.0

results['electricity\_price\_kwh'] = project\_details.get('electricity\_price\_kwh', 0.0) # Speichere den Original-Input-Wert

# --- 5. Wirtschaftlichkeitsberechnungen (Amortisation, NPV etc.) - Mehrjahressimulation ---

try:

net\_investment = results.get('total\_investment\_netto', 0.0)

# Stelle sicher, dass net\_investment eine Zahl ist

if net\_investment is None or not isinstance(net\_investment, (int, float)):

net\_investment = 0.0

# Simulation Setup

simulation\_years = simulation\_period\_years

annual\_cash\_flows\_sim = []

cumulative\_cash\_flows\_sim = []

# Starte kumulierten Cashflow mit der Anfangsinvestition (negativ)

current\_cumulative\_cf = -net\_investment

# Listen zur Speicherung der Jahreswerte für die Anzeige

annual\_productions\_sim = []

annual\_elec\_prices\_sim = []

annual\_feed\_in\_tariffs\_sim = []

annual\_benefits\_sim = []

annual\_maintenance\_costs\_sim = []

annual\_self\_consumption\_kwh\_sim = [] # Hinzugefügt für Simulation Details

annual\_feed\_in\_kwh\_sim = [] # Hinzugefügt für Simulation Details

# Initialisiere die Listen für NPV/IRR. Jahr 0 ist die Investition.

cash\_flows\_for\_npv\_irr = [-net\_investment] # Start mit Initialinvestition

# Hole die Werte aus Jahr 1 (berechnet in Schritt 3 und 4) als Basis für die Simulation

annual\_self\_consumption\_kwh\_year1 = results.get('annual\_self\_consumption\_kwh', 0.0)

annual\_feed\_in\_kwh\_year1 = results.get('annual\_feed\_in\_kwh', 0.0)

anlage\_kwp\_value = results.get('anlage\_kwp', 0.0)

electricity\_price\_kwh\_value\_year1 = results.get('electricity\_price\_kwh', 0.0) # Strompreis Jahr 1

feed\_in\_tariff\_euro\_kwh\_value\_year1 = results.get('feed\_in\_tariff\_euro\_kwh', 0.0) # Einspeisetarif Jahr 1

annual\_pv\_production\_kwh\_year1 = results.get('annual\_pv\_production\_kwh', 0.0)

estimated\_self\_consumption\_rate = results.get('estimated\_self\_consumption\_rate', 0.0)

# --- Mehrjahressimulation Loop ---

for year in range(1, simulation\_years + 1):

# Jährliche Produktion mit Degradation (exponentiell)

production\_this\_year = annual\_pv\_production\_kwh\_year1 \* ((1 - degradation\_rate) \*\* (year - 1))

production\_this\_year = max(0.0, production\_this\_year) # Produktion kann nicht negativ sein

# Strompreis mit jährlicher Steigerung

elec\_price\_this\_year = electricity\_price\_kwh\_value\_year1 \* ((1 + elec\_price\_increase\_rate) \*\* (year - 1))

# Einspeisetarif: Fest für die ersten 20 Jahre, dann 0.0 oder Marktwert (hier Annahme 0.0)

feed\_in\_tariff\_this\_year = feed\_in\_tariff\_euro\_kwh\_value\_year1

if year > 20:

feed\_in\_tariff\_this\_year = 0.0 # Annahme: keine EEG-Vergütung mehr nach 20 Jahren

# Recalculate annual financial benefit based on THIS year's prices/tariffs and PRODUCTION

# Eigenverbrauch basiert auf der DEGRADIERTEN Produktion und dem Verbrauch

estimated\_self\_consumption\_this\_year\_kwh = production\_this\_year \* estimated\_self\_consumption\_rate

# Der tatsächliche Eigenverbrauch ist das Minimum aus geschätztem Eigenverbrauch und tatsächlichem Verbrauch

actual\_self\_consumption\_this\_year\_kwh = min(estimated\_self\_consumption\_this\_year\_kwh, total\_annual\_consumption\_kwh)

# Stellen Sie sicher, dass Eigenverbrauch nicht höher ist als die Produktion dieses Jahres

actual\_self\_consumption\_this\_year\_kwh = min(actual\_self\_consumption\_this\_year\_kwh, production\_this\_year)

actual\_self\_consumption\_this\_year\_kwh = max(0.0, actual\_self\_consumption\_this\_year\_kwh) # Kann nicht negativ sein

# Einspeisung dieses Jahres = Produktion dieses Jahres - tatsächlicher Eigenverbrauch dieses Jahres

actual\_feed\_in\_this\_year\_kwh = production\_this\_year - actual\_self\_consumption\_this\_year\_kwh

actual\_feed\_in\_this\_year\_kwh = max(0.0, actual\_feed\_in\_this\_year\_kwh) # Kann nicht negativ sein

# Berechne die finanziellen Vorteile für dieses Jahr

saved\_cost\_this\_year = actual\_self\_consumption\_this\_year\_kwh \* elec\_price\_this\_year

revenue\_this\_year = actual\_feed\_in\_this\_year\_kwh \* feed\_in\_tariff\_this\_year

benefit\_this\_year = saved\_cost\_this\_year + revenue\_this\_year

# Calculate annual maintenance cost and apply inflation

maintenance\_cost\_this\_year = (maintenance\_cost\_fixed\_pa + maintenance\_cost\_per\_kwp\_pa \* anlage\_kwp\_value) \* ((1 + inflation\_rate) \*\* (year - 1))

# Calculate annual cash flow (Benefit - Maintenance)

annual\_cf = benefit\_this\_year - maintenance\_cost\_this\_year

# Store for simulation results display

annual\_cash\_flows\_sim.append(annual\_cf)

annual\_productions\_sim.append(production\_this\_year)

annual\_elec\_prices\_sim.append(elec\_price\_this\_year)

annual\_feed\_in\_tariffs\_sim.append(feed\_in\_tariff\_this\_year)

annual\_benefits\_sim.append(benefit\_this\_year)

annual\_maintenance\_costs\_sim.append(maintenance\_cost\_this\_year)

annual\_self\_consumption\_kwh\_sim.append(actual\_self\_consumption\_this\_year\_kwh)

annual\_feed\_in\_kwh\_sim.append(actual\_feed\_in\_this\_year\_kwh)

# Calculate cumulative cash flow

current\_cumulative\_cf += annual\_cf

cumulative\_cash\_flows\_sim.append(current\_cumulative\_cf)

# Add annual cash flow to the list for NPV/IRR (starts from Year 1)

cash\_flows\_for\_npv\_irr.append(annual\_cf)

# --- NPV Calculation ---

# Use the inflation rate as the discount rate (simplification)

# TODO: Use a proper discount rate / WACC from Admin settings

discount\_rate\_for\_npv = inflation\_rate # Placeholder

# financial\_lib.npv requires cash flows starting from Year 1. Add Year 0 separately.

# Ensure discount rate is > -1

if discount\_rate\_for\_npv <= -1.0:

npv\_value = float('nan') # Cannot calculate NPV with rate <= -1

calculation\_errors.append("Fehler bei NPV Berechnung: Diskontierungsrate (Inflationsrate) ist <= -100%.")

elif len(cash\_flows\_for\_npv\_irr) > 1: # Benötigt mindestens 2 Cashflows (Jahr 0 und Jahr 1)

try:

# np.npv calculates NPV from Year 1 onwards, so add Year 0 (-net\_investment) separately

npv\_value = financial\_lib.npv(discount\_rate\_for\_npv, cash\_flows\_for\_npv\_irr[1:]) + cash\_flows\_for\_npv\_irr[0]

except Exception as e:

npv\_value = float('nan')

calculation\_errors.append(f"Fehler bei NPV Berechnung: {e}")

traceback.print\_exc()

else: # Kein Cashflow außer initialer Investition

npv\_value = net\_investment \* -1 # NPV ist nur die negative Investition

if net\_investment > 0:

calculation\_errors.append("Warnung: Keine positiven Cashflows nach der Initialinvestition. NPV ist nur die negative Investition.")

# --- IRR Calculation ---

# financial\_lib.irr requires at least one positive and one negative cash flow

irr\_percent = float('nan') # Default to NaN

try:

if len(cash\_flows\_for\_npv\_irr) > 1 and (any(cf > 0 for cf in cash\_flows\_for\_npv\_irr) and any(cf < 0 for cf in cash\_flows\_for\_npv\_irr)):

irr\_value = financial\_lib.irr(cash\_flows\_for\_npv\_irr) # Returns a rate (e.g., 0.05 for 5%)

# Überprüfe auf extrem hohe oder ungültige IRR-Werte, die von np.irr manchmal zurückgegeben werden

if abs(irr\_value) > 100 or not np.isfinite(irr\_value): # Z.B. > 10000 %

irr\_percent = float('nan')

calculation\_errors.append("Warnung: Berechneter IRR ist extrem hoch oder ungültig.")

else:

irr\_percent = irr\_value \* 100.0

# else: irr\_percent remains NaN (keine ausreichenden Cashflows für Berechnung)

except Exception: # Catch potential errors in financial\_lib.irr calculation (e.g., no convergence)

irr\_percent = float('nan') # Indicate calculation failed

calculation\_errors.append("Fehler bei IRR Berechnung: Konnte internen Zinsfuß nicht berechnen (keine Konvergenz oder unzureichende Cashflows).")

# --- Amortization Time Refinement ---

# Find the first year where cumulative cash flow is >= 0

amortization\_time\_sim = float('inf') # Default zu unendlich

if net\_investment > 0: # Amortisation nur relevant für positive Investition

# Füge den Start (Jahr 0) mit dem negativen Investitionswert hinzu

cumulative\_cf\_with\_start = [-net\_investment] + cumulative\_cash\_flows\_sim

for year\_idx, cumulative\_cf in enumerate(cumulative\_cf\_with\_start):

# year\_idx ist der Index in der Liste (0 für Jahr 0, 1 für Jahr 1, etc.)

# Das tatsächliche Jahr ist year\_idx

if cumulative\_cf >= 0:

if year\_idx == 0:

# Amortisation in Jahr 0 bedeutet, die Anfangsinvestition war <= 0

amortization\_time\_sim = 0.0

else:

# Interpolation zwischen dem letzten negativen und dem ersten nicht-negativen Punkt

last\_negative\_year\_idx = year\_idx - 1

# Stelle sicher, dass der Index gültig ist, bevor zugegriffen wird

if last\_negative\_year\_idx >= 0:

last\_negative\_cumulative\_cf = cumulative\_cf\_with\_start[last\_negative\_year\_idx]

# Stelle sicher, dass der Index für annual\_cash\_flows\_sim gültig ist (beginnt bei Index 0 für Jahr 1)

if (year\_idx - 1) < len(annual\_cash\_flows\_sim):

current\_year\_cf = annual\_cash\_flows\_sim[year\_idx - 1] # Cashflow des Jahres, in dem Amortisation erreicht wird (Index 0 = Jahr 1 CF)

if current\_year\_cf > 0: # Nur interpolieren, wenn der Cashflow in diesem Jahr positiv ist

# Amortisationszeit = Letztes negatives Jahr + Betrag, der noch amortisiert werden muss / Cashflow in diesem Jahr

# Letztes negatives Jahr ist year\_idx - 1

# Betrag, der noch amortisiert werden muss, ist der absolute Wert des kumulierten Cashflows am Ende des letzten negativen Jahres

amortization\_time\_sim = last\_negative\_year\_idx + abs(last\_negative\_cumulative\_cf) / current\_year\_cf

else:

# Wenn der Cashflow nicht positiv ist, wird der Rückstand nicht aufgeholt

amortization\_time\_sim = float('inf') # Bleibt unendlich

else:

# Sollte nicht passieren, wenn cumulative\_cash\_flows\_sim korrekt befüllt ist,

# aber als Absicherung gegen Indexfehler

amortization\_time\_sim = float('inf')

calculation\_errors.append(f"Interner Fehler bei Amortisationsberechnung: Ungültiger Index für Jahres-Cashflow in Jahr {year\_idx}.")

else:

# Sollte nur bei year\_idx == 0 passieren, der oben behandelt wird

amortization\_time\_sim = float('inf')

break # Amortisation gefunden, Schleife beenden

# Wenn die Schleife endet und amortization\_time\_sim immer noch inf ist,

# bedeutet das, dass der kumulierte Cashflow nie >= 0 wurde innerhalb der Simulationsjahre.

# Dies wird bereits durch den initialen Wert von float('inf') korrekt behandelt.

elif net\_investment <= 0:

# Wenn die Investition 0 oder negativ ist, ist die Amortisationszeit 0 oder nicht sinnvoll (inf)

# Wenn die Initialinvestition <= 0 ist UND der jährliche Benefit > 0 ist, ist es sofort amortisiert.

if results.get('annual\_financial\_benefit\_year1', 0.0) > 0:

amortization\_time\_sim = 0.0

else:

amortization\_time\_sim = float('inf') # Kein Benefit, keine Amortisation

results['simulation\_period\_years'] = simulation\_years

results['annual\_cash\_flows\_sim'] = annual\_cash\_flows\_sim # Store the list

results['cumulative\_cash\_flows\_sim'] = cumulative\_cash\_flows\_sim # Store the list

results['npv\_over\_years'] = npv\_value # Store NPV (scalar)

results['irr\_percent'] = irr\_percent # Store IRR (scalar %)

results['amortization\_time\_years'] = amortization\_time\_sim # Update amortization time

# Store simulation details lists for analysis display

results['annual\_productions\_sim'] = annual\_productions\_sim

results['annual\_elec\_prices\_sim'] = annual\_elec\_prices\_sim

results['annual\_feed\_in\_tariffs\_sim'] = annual\_feed\_in\_tariffs\_sim

results['annual\_benefits\_sim'] = annual\_benefits\_sim

results['annual\_maintenance\_costs\_sim'] = annual\_maintenance\_costs\_sim

results['annual\_self\_consumption\_kwh\_sim'] = annual\_self\_consumption\_kwh\_sim

results['annual\_feed\_in\_kwh\_sim'] = annual\_feed\_in\_kwh\_sim

except Exception as e: # Catch errors in the entire economic analysis block

calculation\_errors.append(f"Fehler bei Wirtschaftlichkeitsberechnung (Simulation): {e}")

traceback.print\_exc()

results['simulation\_period\_years'] = simulation\_period\_years # Store period even on error

results['annual\_cash\_flows\_sim'] = []

results['cumulative\_cash\_flows\_sim'] = []

results['npv\_over\_years'] = float('nan') # Set NPV to NaN on error

results['irr\_percent'] = float('nan') # Set IRR to NaN on error

results['amortization\_time\_years'] = float('inf') # Default to infinite on error

# Also reset/clear simulation detail lists on error

results['annual\_productions\_sim'] = []

results['annual\_elec\_prices\_sim'] = []

results['annual\_feed\_in\_tariffs\_sim'] = []

results['annual\_benefits\_sim'] = []

results['annual\_maintenance\_costs\_sim'] = []

results['annual\_self\_consumption\_kwh\_sim'] = []

results['annual\_feed\_in\_kwh\_sim'] = []

# --- 6. CO2 Einsparung ---

try:

co2\_factor\_g\_per\_kwh\_grid = load\_admin\_setting('co2\_factor\_grid\_g\_per\_kwh', 388.0) # Standardwert nutzen

# Gesamte produzierte PV-Energie (pro Jahr) multipliziert mit dem CO2-Faktor des Netzstroms (da diese Menge Netzstrom ersetzt)

# Konvertierung von Gramm zu Kilogramm ( / 1000)

results['annual\_co2\_savings\_kg'] = (results.get('annual\_pv\_production\_kwh', 0.0) \* co2\_factor\_g\_per\_kwh\_grid) / 1000.0

except Exception as e: # Catch errors in the entire CO2 block

calculation\_errors.append(f"Fehler bei CO2 Berechnung: {e}")

traceback.print\_exc()

results['annual\_co2\_savings\_kg'] = 0.0

# --- 7. Zukunftsplanung (WP/E-Auto) --- (Platzhalter)

# Diese Logik müsste hier implementiert werden, um den zukünftigen Verbrauch zu simulieren

# und dessen Einfluss auf Eigenverbrauch/Netzbezug und finanzielle Vorteile zu berechnen.

results['future\_consumption\_analysis'] = "Platzhalter für Zukunftsplanung"

# --- 8. Szenariensimulationen --- (Platzhalter)

# Logik zur Durchführung und zum Vergleich verschiedener Szenarien (z.B. Speichergröße, Verbrauchsprofile)

results['scenario\_simulation\_results'] = "Platzhalter für Szenariensimulationen"

# --- 9. Finanzierungsvergleich (Placeholder) ---

# Logik zur Berechnung und zum Vergleich von Finanzierungsoptionen (Kredit vs. Eigenkapital)

results['financial\_comparison\_results'] = "Platzhalter für Finanzierungsvergleich"

# --- Anzeige gesammelter Fehler ---

results['calculation\_errors'] = calculation\_errors # Füge die gesammelten Fehler hinzu

# print("perform\_calculations() beendet") # Debugging

return results

# --- Ende des calculations.py Codes ---

**Crm.py:**

# crm.py

# Modul für das Customer Relationship Management (CRM)

import streamlit as st

import sqlite3 # Direktimport für SQLite-spezifische Fehlerbehandlung

import re # Importiere re für die Regex-Suche nach ID im Kundennamen

from typing import Dict, List, Optional, Any

from datetime import datetime # Für Zeitstempel

import traceback # Importiere traceback für Fehlerdetails

import pandas as pd # Für die Anzeige in der Tabelle

# Importiere die Funktion, die die Datenbankverbindung bereitstellt, aus database.py

# Stelle sicher, dass database.py importierbar ist

try:

# Verwende absolute Imports oder stelle sicher, dass dein PYTHONPATH korrekt ist.

# 'from database import ...' geht davon aus, dass das Modul 'database' im Python-Pfad ist.

# Wenn gui.py und database.py im selben Ordner liegen und du 'streamlit run gui.py'

# in diesem Ordner ausführst, sollte der Import 'from database import ...' funktionieren.

from database import get\_db\_connection as real\_get\_db\_connection # Benötigen get\_db\_connection

# Optional: Dummy-Funktion für den Fall, dass database.py nicht geladen werden kann

# Stelle sicher, dass die importierte Funktion auch wirklich callable ist, sonst ist der Import vielleicht fehlgeschlagen

if not callable(real\_get\_db\_connection): raise ImportError("Imported get\_db\_connection is not callable.")

# Wenn der Import und Check erfolgreich, weisen wir die reale Funktion zu

get\_db\_connection\_safe\_crm = real\_get\_db\_connection

except ImportError as e:

# Bei Importfehler weisen wir Dummy-Funktion zu und geben Fehlermeldung aus

def get\_db\_connection\_safe\_crm():

#print(f"crm.py: Importfehler für database.py: {e}. Dummy DB-Verbindung.") # Logge den Fehler im Terminal

# Streamlit Fehler anzeigen, aber nur einmal pro Lauf? -> Nein, besser global in gui.py

return None

# Füge Fehler zur globalen Liste in gui.py hinzu, wenn möglich

try:

from gui import import\_errors as global\_import\_errors

global\_import\_errors.append(f"crm.py: Importfehler für database.py: {e}. CRM DB Funktionen nicht verfügbar.")

except ImportError:

pass # Globaler Handler nicht verfügbar

except Exception as e\_global:

pass # Fehler beim Zugriff auf globalen Handler

except Exception as e:

# Bei anderen Fehlern beim Laden weisen wir Dummy-Funktionen zu und geben Fehlermeldung aus

def get\_db\_connection\_safe\_crm():

#print(f"crm.py: Fehler beim Laden von database.py: {e}. Dummy DB-Verbindung.") # Logge den Fehler im Terminal

# Streamlit Fehler anzeigen...

return None

# Füge Fehler zur globalen Liste in gui.py hinzu

try:

from gui import import\_errors as global\_import\_errors

global\_import\_errors.append(f"crm.py: Fehler im database.py Modul beim Laden: {e}. CRM DB Funktionen nicht verfügbar.")

except ImportError:

pass

except Exception as e\_global:

pass

# --- Hilfsfunktionen zum Laden von Daten ---

def load\_all\_customers() -> List[Dict[str, Any]]:

"""Lädt alle Kunden aus der Datenbank."""

conn = get\_db\_connection\_safe\_crm() # Nutze die '\_safe' Funktion (kann None sein)

if conn is None:

# Zeige Fehler in der UI, wenn keine Verbindung möglich ist, aber erst IN render\_crm

# st.error("Datenbankverbindung für Kundenliste nicht verfügbar.") # Kein st.error hier!

print("load\_all\_customers: Keine Datenbankverbindung verfügbar (get\_db\_connection\_safe\_crm gab None zurück).") # Logge

return [] # Leere Liste bei Fehler

cursor = conn.cursor()

try:

cursor.execute("SELECT \* FROM customers ORDER BY last\_name, first\_name")

rows = cursor.fetchall()

# conn.close() # Verbindung nicht hier schließen, wird von gui.py gemacht wenn übergeben

return [dict(row) for row in rows] # Konvertiere Rows zu Dictionaries

except Exception as e:

# Zeige Fehler in der UI, wenn DB-Abfrage fehlschlägt

# st.error(f"Fehler beim Laden der Kunden aus der Datenbank: {e}") # Kein st.error hier!

print(f"load\_all\_customers: Fehler beim Laden der Kunden: {e}") # Logge

traceback.print\_exc()

return [] # Leere Liste bei Fehler

finally:

if conn: conn.close() # Verbindung am Ende schließen

def load\_projects\_by\_customer\_id(customer\_id: int) -> List[Dict[str, Any]]:

"""Lädt alle Projekte für einen spezifischen Kunden anhand seiner ID."""

conn = get\_db\_connection\_safe\_crm() # Nutze die '\_safe' Funktion

if conn is None:

# st.error("Datenbankverbindung für Projektliste nicht verfügbar.") # Kein st.error hier!

print("load\_projects\_by\_customer\_id: Keine Datenbankverbindung verfügbar (get\_db\_connection\_safe\_crm gab None zurück).") # Logge

return [] # Leere Liste bei Fehler

cursor = conn.cursor()

try:

# Hier müssen wir die Spalte 'anlage\_kwp' aus der projects Tabelle laden,

# um sie in der CRM-Übersicht anzeigen zu können.

# Stelle sicher, dass die Spalte 'anlage\_kwp' in der projects Tabelle in database.py existiert (haben wir hinzugefügt).

# Lade auch den Modulnamen für die Anzeige

cursor.execute("""

SELECT p.\*, -- Alle Spalten aus projects

(SELECT model\_name FROM products WHERE id = p.selected\_module\_id) as module\_model\_name -- Modulname aus products

FROM projects p

WHERE p.customer\_id = ?

ORDER BY p.created\_at DESC

""", (customer\_id,))

rows = cursor.fetchall()

# conn.close() # Verbindung nicht hier schließen

return [dict(row) for row in rows] # Konvertiere Rows zu Dictionaries

except Exception as e:

# st.error(f"Fehler beim Laden der Projekte für Kunde {customer\_id}: {e}") # Kein st.error hier!

print(f"load\_projects\_by\_customer\_id: Fehler beim Laden der Projekte für Kunde {customer\_id}: {e}") # Logge

traceback.print\_exc()

return [] # Leere Liste bei Fehler

finally:

if conn: conn.close() # Verbindung am Ende schließen

# --- Speicher Funktionen (werden von gui.py aufgerufen) ---

# Diese Funktionen speichern Daten in der Datenbank.

def save\_customer(conn, customer\_data: Dict[str, Any]) -> int: # Rückgabe der ID oder -1 bei Fehler

"""

Speichert Kundendaten in der 'customers' Tabelle.

Akzeptiert eine Datenbankverbindung (conn) und ein Dictionary mit Kundendaten.

Gibt die ID des neuen Kunden zurück oder -1 bei Fehler.

"""

# Die Datenbankverbindung (conn) wird von gui.py geöffnet und übergeben.

# Wir verwenden diese übergebene Verbindung und schließen sie NICHT hier,

# das Schließen übernimmt der Aufrufer (gui.py).

if conn is None:

print("save\_customer: Keine gültige Datenbankverbindung übergeben.")

return -1 # Indicate failure

cursor = conn.cursor()

try:

# SQL INSERT Statement

# Stellen Sie sicher, dass die übergebenen Schlüssel mit den Spaltennamen übereinstimmen

cursor.execute("""

INSERT INTO customers (type, salutation, title, first\_name, last\_name,

address, house\_number, zip\_code, city, state, region,

num\_persons, email, phone\_landline, phone\_mobile)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (

customer\_data.get('type'),

customer\_data.get('salutation'),

customer\_data.get('title'),

customer\_data.get('first\_name'),

customer\_data.get('last\_name'),

customer\_data.get('address'),

customer\_data.get('house\_number'), # Muss evtl. aus address extrahiert werden ODER Eingabefeld in data\_input anpassen

customer\_data.get('zip\_code'),

customer\_data.get('city'),

customer\_data.get('state'),

customer\_data.get('region'),

customer\_data.get('num\_persons'),

customer\_data.get('email'),

customer\_data.get('phone\_landline'),

customer\_data.get('phone\_mobile')

))

# conn.commit() # Commit am Ende der Transaktion, nicht hier

#print(f"Kunde '{customer\_data.get('first\_name')} {customer\_data.get('last\_name')}' zum Einfügen vorbereitet.") # Debugging

# Rückgabe der ID des neu eingefügten Datensatzes

return cursor.lastrowid

except sqlite3.IntegrityError as e:

print(f"save\_customer: Integrity Error - Kunde existiert möglicherweise bereits oder Daten sind ungültig: {e}")

traceback.print\_exc()

# conn.rollback() # Rollback nur bei Bedarf hier, oder gui.py macht es bei Fehler

return -1 # Indicate failure

except Exception as e:

print(f"save\_customer: Fehler beim Einfügen von Kundendaten: {e}")

traceback.print\_exc()

# conn.rollback() # Rollback nur bei Bedarf hier

return -1 # Indicate failure

def save\_project(conn, project\_details: Dict[str, Any]) -> int: # Rückgabe der ID oder -1 bei Fehler

"""

Speichert Projektdaten in der 'projects' Tabelle.

Akzeptiert eine Datenbankverbindung (conn) und ein Dictionary mit Projektdaten (inkl. customer\_id).

Gibt die ID des neuen Projekts zurück oder -1 bei Fehler.

"""

if conn is None:

print("save\_project: Keine gültige Datenbankverbindung übergeben.")

return -1 # Indicate failure

cursor = conn.cursor()

try:

# Stelle sicher, dass customer\_id in project\_details vorhanden ist

customer\_id = project\_details.get('customer\_id')

if customer\_id is None:

print("save\_project: customer\_id fehlt in den Projektdetails.")

return -1 # Indicate failure

# Stelle sicher, dass project\_name gesetzt ist (provisorisch, sollte vom Benutzer kommen oder generiert werden)

project\_name = project\_details.get('project\_name', f"Projekt Kunde {customer\_id}")

# SQL INSERT Statement

# ACHTUNG: Hier müssen alle Spalten aufgeführt werden, die wir speichern wollen!

# Aktuell: Nur die aus A.1, A.2, und Basis A.3 (Module, EV, HP)

# Später HINZUFÜGEN: selected\_inverter\_id, include\_storage, selected\_storage\_id, selected\_storage\_capacity\_kwh

# Stelle sicher, dass die Spaltennamen hier genau denen in database.py CREATE TABLE entsprechen!

# Fügen wir hier auch die Felder für Inverter und Storage hinzu, auch wenn sie noch None sind,

# um das SQL-Statement stabil zu halten, wenn data\_input.py die Felder hinzufügt.

cursor.execute("""

INSERT INTO projects (customer\_id, project\_name, anlage\_type, feed\_in\_type,

consumption\_household\_kwh\_yr, consumption\_heating\_kwh\_yr,

costs\_household\_euro\_mo, costs\_heating\_euro\_mo,

roof\_type, build\_year, roof\_covering\_type, free\_roof\_area\_sqm,

roof\_orientation, roof\_inclination\_deg, building\_height\_gt\_7m,

module\_quantity, selected\_module\_id, anlage\_kwp, -- Anlage\_kwp wird in data\_input berechnet und übergeben

future\_ev, future\_hp,

selected\_inverter\_id, include\_storage, selected\_storage\_id, selected\_storage\_capacity\_kwh -- Felder für WR/Speicher

)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (

customer\_id,

project\_name,

project\_details.get('anlage\_type'),

project\_details.get('feed\_in\_type'),

project\_details.get('consumption\_household\_kwh\_yr'),

project\_details.get('consumption\_heating\_kwh\_yr'),

project\_details.get('costs\_household\_euro\_mo'),

project\_details.get('costs\_heating\_euro\_mo'),

project\_details.get('roof\_type'),

project\_details.get('build\_year'),

project\_details.get('roof\_covering\_type'),

project\_details.get('free\_roof\_area\_sqm'),

project\_details.get('roof\_orientation'),

project\_details.get('roof\_inclination\_deg'),

project\_details.get('building\_height\_gt\_7m'),

project\_details.get('module\_quantity'),

project\_details.get('selected\_module\_id'), # ID des Modells speichern

project\_details.get('anlage\_kwp'), # kWp Wert speichern

project\_details.get('future\_ev', False), # Standard auf False setzen, falls nicht vorhanden

project\_details.get('future\_hp', False), # Standard auf False setzen

project\_details.get('selected\_inverter\_id'), # WR ID speichern (kann None sein)

project\_details.get('include\_storage', False), # Speicherbool speichern

project\_details.get('selected\_storage\_id'), # Speicher ID speichern (kann None sein)

project\_details.get('selected\_storage\_capacity\_kwh') # Eingegebene Speicher Kapazität speichern (kann None sein)

))

# conn.commit() # Commit am Ende der Transaktion in gui.py

# Rückgabe der ID des neu eingefügten Datensatzes

return cursor.lastrowid

except sqlite3.IntegrityError as e:

print(f"save\_project: Integrity Error - Projekt existiert möglicherweise bereits oder Daten sind ungültig: {e}")

traceback.print\_exc()

return -1 # Indicate failure

except Exception as e:

print(f"save\_project: Fehler beim Einfügen von Projektdaten: {e}")

traceback.print\_exc()

return -1 # Indicate failure

# --- Render Funktion für den CRM Tab (C) ---

def render\_crm(texts, get\_db\_connection\_func=None): # get\_db\_connection\_func wird von gui.py übergeben

"""

Implementierung der Benutzeroberfläche für den CRM Tab (C).

Zeigt eine Liste der Kunden und deren Projekte an.

"""

st.header(texts.get("menu\_item\_crm", "CRM")) # Nutze den Titel aus den Texten

# --- CRM Benutzeroberfläche ---

st.subheader(texts.get("crm\_customer\_list\_header", "Kundenliste")) # Neuer Header aus Texten

# Lade alle Kunden beim Rendern des Tabs

# Die Ladefunktion load\_all\_customers() gibt bei Fehlern eine leere Liste zurück

customers = load\_all\_customers() # Nutze die Ladefunktion

if not customers:

# Ladefunktion zeigt bei DB-Fehler schon eine Konsolenmeldung, hier in UI

st.info(texts.get("crm\_no\_customers\_found", "Keine Kunden in der Datenbank gefunden."))

# Zeige eine zusätzliche Meldung, wenn die Ladefunktion leer zurückgab ABER DB verfügbar sein sollte

# Füge einen kleinen Check hinzu, um nicht bei jedem Neuladen des Tabs die Warning zu spammen

if 'crm\_db\_warning\_shown' not in st.session\_state:

if get\_db\_connection\_safe\_crm is not None:

conn\_check = get\_db\_connection\_safe\_crm() # Versuche eine Verbindung zu bekommen

if conn\_check: # Wenn Verbindung möglich war, aber keine Kunden gefunden

st.warning("Datenbank scheint leer zu sein. Bitte speichern Sie neue Projekte im 'Eingabe (A)' Tab.")

conn\_check.close()

else: # Wenn keine Verbindung möglich war

st.error(texts.get("db\_connection\_unavailable", "Datenbankverbindung nicht verfügbar. Kundenliste kann nicht geladen werden."))

st.session\_state['crm\_db\_warning\_shown'] = True # Markiere, dass die Warnung/Fehler gezeigt wurde

else: # Wenn die Safe-Funktion selbst der Dummy ist

st.error(texts.get("db\_connection\_unavailable", "Datenbankverbindung nicht verfügbar. Kundenliste kann nicht geladen werden."))

st.session\_state['crm\_db\_warning\_shown'] = True # Markiere

else:

# Wenn Kunden gefunden wurden, setze die Warnung zurück, falls vorhanden

if 'crm\_db\_warning\_shown' in st.session\_state:

del st.session\_state['crm\_db\_warning\_shown']

# Zeige Kunden in einer Selectbox

customer\_options\_display = [f"{c.get('last\_name', 'N/A')}, {c.get('first\_name', 'N/A')} (ID: {c.get('id')})" for c in customers]

# Option "--- Kunden auswählen ---" hinzufügen

customer\_options\_display.insert(0, texts.get("crm\_select\_customer\_placeholder", "--- Kunden auswählen ---")) # Füge Platzhalter hinzu

# Use session state for selectbox to remember selection

# Initialisiere den session state Wert sicher

if "crm\_selected\_customer\_display" not in st.session\_state or st.session\_state["crm\_selected\_customer\_display"] not in customer\_options\_display:

st.session\_state["crm\_selected\_customer\_display"] = customer\_options\_display[0] # Setze Standard auf ersten Eintrag

try:

initial\_index\_crm\_customer = customer\_options\_display.index(st.session\_state["crm\_selected\_customer\_display"])

except ValueError:

initial\_index\_crm\_customer = 0 # Fallback wenn gespeicherter Wert nicht mehr existiert

selected\_customer\_display = st.selectbox(

texts.get("crm\_select\_customer\_label", "Kunden auswählen"),

options=customer\_options\_display,

index=initial\_index\_crm\_customer,

key="crm\_customer\_selectbox" # Session State Key für die Auswahl

)

# Streamlit aktualisiert st.session\_state["crm\_customer\_selectbox"] automatisch

# Finde den ausgewählten Kunden anhand der ID im String

selected\_customer\_id = None

selected\_customer\_details = None # Speichere auch das Kunden-Dict

if selected\_customer\_display != texts.get("crm\_select\_customer\_placeholder", "--- Kunden auswählen ---"):

# Extrahiere die ID aus dem String (ID: X)

match = re.search(r'\(ID: (\d+)\)', selected\_customer\_display)

if match:

selected\_customer\_id = int(match.group(1)) # Konvertiere ID zu Integer

# Finde das volle Kunden-Dictionary anhand der ID in der geladenen Liste

selected\_customer\_details = next((c for c in customers if c.get('id') == selected\_customer\_id), None)

# Wenn Kunde nicht gefunden wird (obwohl ID extrahiert), logge Fehler

if selected\_customer\_details is None:

print(f"crm.py: Fehler - Kunde mit ID {selected\_customer\_id} nach Auswahl im Dropdown nicht in geladener Liste gefunden.")

# --- Anzeige der Kundendetails und Projekte ---

if selected\_customer\_details: # Prüfe, ob ein Kunde ausgewählt und gefunden wurde

st.subheader(texts.get("crm\_customer\_details\_header", "Kundendetails")) # Header

# Anzeige der Kundendetails (Nutze get() mit Fallback für fehlende Daten)

st.write(f"\*\*{texts.get('crm\_name\_label', 'Name')}:\*\* {selected\_customer\_details.get('salutation', '')} {selected\_customer\_details.get('title', '')} {selected\_customer\_details.get('first\_name', 'N/A')} {selected\_customer\_details.get('last\_name', 'N/A')}")

st.write(f"\*\*{texts.get('crm\_type\_label', 'Typ')}:\*\* {selected\_customer\_details.get('type', 'N/A')}")

st.write(f"\*\*{texts.get('crm\_contact\_label', 'Kontakt')}:\*\* E-Mail: {selected\_customer\_details.get('email', 'N/A')}, Festnetz: {selected\_customer\_details.get('phone\_landline', 'N/A')}, Mobil: {selected\_customer\_details.get('phone\_mobile', 'N/A')}")

st.write(f"\*\*{texts.get('crm\_address\_label', 'Adresse')}:\*\* {selected\_customer\_details.get('address', 'N/A')} {selected\_customer\_details.get('house\_number', '')}, {selected\_customer\_details.get('zip\_code', 'N/A')} {selected\_customer\_details.get('city', 'N/A')}")

st.write(f"\*\*{texts.get('crm\_location\_label', 'Region')}:\*\* {selected\_customer\_details.get('region', 'N/A')}, {selected\_customer\_details.get('state', 'N/A')}")

st.write(f"\*\*{texts.get('crm\_num\_persons\_label', 'Personen im Haushalt')}:\*\* {selected\_customer\_details.get('num\_persons', 'N/A')}")

st.write(f"\*\*ID:\*\* {selected\_customer\_details.get('id', 'N/A')}") # Kunden ID anzeigen

st.markdown("---") # Trennlinie

st.subheader(texts.get("crm\_projects\_header", "Projekte")) # Header

# Lade Projekte für den ausgewählten Kunden

# Die Ladefunktion load\_projects\_by\_customer\_id gibt bei Fehlern eine leere Liste zurück

projects = load\_projects\_by\_customer\_id(selected\_customer\_id) # Nutze die Ladefunktion

if not projects:

st.info(texts.get("crm\_no\_projects\_found", "Keine Projekte für diesen Kunden gefunden."))

else:

# Zeige Projekte in einer Tabelle

project\_display\_data = []

for p in projects:

display\_row = {

# Nutze get() mit Fallback für fehlende Daten

texts.get('crm\_project\_name\_label', 'Projektname'): p.get('project\_name', 'N/A'),

texts.get('crm\_anlage\_type\_label', 'Anlagentyp'): p.get('anlage\_type', 'N/A'),

texts.get('crm\_feed\_in\_type\_label', 'Einspeisetyp'): p.get('feed\_in\_type', 'N/A'),

texts.get('crm\_module\_model\_label', 'Modul Modell'): p.get('module\_model\_name', 'N/A'), # Zeige den Modulnamen (aus SELECT Statement)

texts.get('crm\_module\_quantity\_label', 'Anzahl Module'): p.get('module\_quantity', 'N/A'), # Anzahl Module

texts.get('crm\_anlage\_kwp\_label', 'Anlagengröße (kWp)'): p.get('anlage\_kwp', 0.0), # Display kWp

# Füge weitere wichtige Projektdetails hinzu, die in der DB gespeichert werden

texts.get('crm\_build\_year\_label', 'Baujahr'): p.get('build\_year', 'N/A'), # Füge Baujahr hinzu

texts.get('crm\_roof\_type\_label', 'Dachart'): p.get('roof\_type', 'N/A'), # Füge Dachart hinzu

texts.get('crm\_created\_at\_label', 'Erstellt am'): p.get('created\_at', 'N/A') # Datum

# texts.get('crm\_status\_label', 'Status'): p.get('status', 'N/A'), # Später aus crm\_data Tabelle

# texts.get('crm\_project\_id\_label', 'Projekt ID'): p.get('id', 'N/A') # Projekt ID anzeigen

}

# Formatiere kWp Wert, wenn es eine Zahl ist

kwp\_val = display\_row[texts.get('crm\_anlage\_kwp\_label', 'Anlagengröße (kWp)')]

if isinstance(kwp\_val, (int, float)):

display\_row[texts.get('crm\_anlage\_kwp\_label', 'Anlagengröße (kWp)')] = f"{kwp\_val:.2f}" # Formatiere als String

elif kwp\_val is None:

display\_row[texts.get('crm\_anlage\_kwp\_label', 'Anlagengröße (kWp)')] = 'N/A' # Oder ein anderer Fallback

# Format created\_at if possible

created\_at\_val = display\_row[texts.get('crm\_created\_at\_label', 'Erstellt am')]

if isinstance(created\_at\_val, str) and created\_at\_val != 'N/A':

try:

# Passe das Format an, falls nötig (SQLite speichert als Text '%Y-%m-%d %H:%M:%S')

dt\_obj = datetime.strptime(created\_at\_val, '%Y-%m-%d %H:%M:%S')

display\_row[texts.get('crm\_created\_at\_label', 'Erstellt am')] = dt\_obj.strftime('%d.%m.%Y %H:%M') # Neues Format

except ValueError:

pass # Bei Fehler bleibt der Originalstring

project\_display\_data.append(display\_row)

# Konvertiere die Liste von Dictionaries in ein Pandas DataFrame für die Streamlit Tabelle

# Zeige nur relevante Spalten in der finalen Tabelle

df\_projects = pd.DataFrame(project\_display\_data)

# Definiere die Reihenfolge und Auswahl der Spalten für die Anzeige

display\_columns = [

texts.get('crm\_project\_name\_label', 'Projektname'),

texts.get('crm\_anlage\_type\_label', 'Anlagentyp'),

texts.get('crm\_feed\_in\_type\_label', 'Einspeisetyp'),

texts.get('crm\_module\_model\_label', 'Modul Modell'),

texts.get('crm\_module\_quantity\_label', 'Anzahl Module'),

texts.get('crm\_anlage\_kwp\_label', 'Anlagengröße (kWp)'),

texts.get('crm\_build\_year\_label', 'Baujahr'),

texts.get('crm\_roof\_type\_label', 'Dachart'),

# texts.get('crm\_free\_area\_label', 'Dachfläche (m²)'), # Ggf. zu viele Spalten, optional hinzufügen

texts.get('crm\_created\_at\_label', 'Erstellt am')

]

# Stelle sicher, dass nur vorhandene Spalten angezeigt werden (falls z.B. ein Textkey fehlt)

display\_columns = [col for col in display\_columns if col in df\_projects.columns]

st.dataframe(df\_projects[display\_columns], use\_container\_width=True) # Zeige nur die ausgewählten Spalten

st.markdown("---")

# Buttons für Projektaktionen (später implementieren)

st.write(texts.get("crm\_project\_actions\_header", "Projektaktionen (Placeholder)"))

# Wähle ein Projekt aus der Tabelle aus, um Aktionen zu ermöglichen (komplexer)

# Temporär Buttons deaktiviert

col\_edit, col\_delete, col\_offer = st.columns(3)

with col\_edit: st.button(texts.get("crm\_edit\_project\_button", "Projekt bearbeiten"), disabled=True)

with col\_delete: st.button(texts.get("crm\_delete\_project\_button", "Projekt löschen"), disabled=True)

with col\_offer: st.button(texts.get("crm\_generate\_offer\_button", "Angebot erstellen"), disabled=True) # Feature 14

# Füge Buttons für Kundenaktionen (später implementieren)

st.markdown("---")

st.write(texts.get("crm\_customer\_actions\_header", "Kundenaktionen (Placeholder)"))

# Temporär Buttons deaktiviert

col\_edit\_cust, col\_delete\_cust, col\_follow\_up = st.columns(3)

with col\_edit\_cust: st.button(texts.get("crm\_edit\_customer\_button", "Kunden bearbeiten"), disabled=True)

with col\_delete\_cust: st.button(texts.get("crm\_delete\_customer\_button", "Kunden löschen"), disabled=True)

with col\_follow\_up: st.button(texts.get("crm\_set\_follow\_up\_button", "Wiedervorlage setzen"), disabled=True) # Teil von Feature C - Kalender/Follow-up

# --- Speicher Funktionen (bleiben hier, sind aber nicht Teil der CRM Render Funktion) ---

# save\_customer und save\_project sind hier definiert und werden von gui.py aufgerufen.

# --- Optional: Test beim direkten Ausführen (nur zum Debuggen) ---

# Dieser Block wird nur ausgeführt, wenn du python crm.py direkt startest

if \_\_name\_\_ == "\_\_main\_\_":

print("--- Testlauf für crm.py ---")

# Für diesen Test brauchen wir eine DB-Verbindung und eine initialisierte DB

try:

# Stelle sicher, dass die DB existiert und initialisiert ist.

# Führe `python database.py` einmalig manuell aus, falls noch nicht geschehen.

import database

# database.init\_db() # Sicherstellen, dass Tabellen da sind (sollte schon passiert sein)

print("Datenbank Initialisierung/Check abgeschlossen.")

conn\_test = database.get\_db\_connection()

if conn\_test:

print("\nDatenbankverbindung im crm.py Test verfügbar.")

# Beispiel Kundendaten (Nur um Speicherfunktionen zu testen, nicht zum Laden im Render Test)

# test\_customer\_data = {

# 'type': 'Privat',

# 'salutation': 'Herr',

# 'title': None,

# 'first\_name': 'Max',

# 'last\_name': 'Mustermann',

# 'address': 'Musterstraße',

# 'house\_number': '1a',

# 'zip\_code': '12345',

# 'city': 'Musterstadt',

# 'state': 'Bayern',

# 'region': 'Süd',

# 'num\_persons': 4,

# 'email': 'max.mustermann@example.com',

# 'phone\_landline': '089-123456',

# 'phone\_mobile': '0170-7890123'

# }

# Beispiel Projektdaten (minimal)

# test\_project\_details\_min = {

# 'project\_name': 'Musterprojekt 1',

# 'anlage\_type': 'Neuanlage',

# 'feed\_in\_type': 'Teileinspeisung',

# 'consumption\_household\_kwh\_yr': 4000.0,

# 'consumption\_heating\_kwh\_yr': 1500.0,

# 'costs\_household\_euro\_mo': 90.0,

# 'costs\_heating\_euro\_mo': 30.0,

# 'roof\_type': 'Satteldach',

# 'build\_year': 2010,

# 'roof\_covering\_type': 'Tonziegel',

# 'free\_roof\_area\_sqm': 60.0,

# 'roof\_orientation': 'Süd',

# 'roof\_inclination\_deg': 35,

# 'building\_height\_gt\_7m': False,

# 'module\_quantity': 15,

# 'selected\_module\_id': 1, # Annahme: Modul mit ID 1 existiert

# 'anlage\_kwp': 6.0, # Beispiel kWp

# 'future\_ev': True,

# 'future\_hp': False,

# 'selected\_inverter\_id': None,

# 'include\_storage': False,

# 'selected\_storage\_id': None,

# 'selected\_storage\_capacity\_kwh': None

# }

# print("\nVersuche, Testkunden zu speichern...")

# customer\_id = save\_customer(conn\_test, test\_customer\_data) # Nutze die save\_customer Funktion

# if customer\_id != -1:

# print(f"Testkunde erfolgreich gespeichert mit ID: {customer\_id}")

# print("\nVersuche, Testprojekt zu speichern...")

# # Füge die erhaltene customer\_id zu den Projektdetails hinzu

# test\_project\_details\_min['customer\_id'] = customer\_id

# project\_id = save\_project(conn\_test, test\_project\_details\_min) # Nutze die save\_project Funktion

# if project\_id != -1:

# print(f"Testprojekt erfolgreich gespeichert mit ID: {project\_id}")

# else:

# print("Fehler beim Speichern des Testprojekts.")

# else:

# print("Fehler beim Speichern des Testkunden.")

# conn\_test.commit() # Hier committen wir die Transaktion für den Test

# conn\_test.close() # Und schließen die Verbindung

# --- Teste Ladefunktionen ---

print("\nTeste load\_all\_customers...")

all\_cust = load\_all\_customers() # Rufe die Ladefunktion auf

print(f"Gefundene Kunden: {len(all\_cust)}")

for c in all\_cust:

# Drucke nur relevante Felder zur Übersicht

print(f" ID: {c.get('id')}, Name: {c.get('first\_name')} {c.get('last\_name')}, Ort: {c.get('city')}")

if all\_cust:

first\_cust\_id = all\_cust[0].get('id')

print(f"\nTeste load\_projects\_by\_customer\_id für Kunde ID {first\_cust\_id}...")

cust\_projects = load\_projects\_by\_customer\_id(first\_cust\_id) # Rufe die Ladefunktion auf

print(f"Gefundene Projekte für Kunde {first\_cust\_id}: {len(cust\_projects)}")

for p in cust\_projects:

# Drucke nur relevante Felder zur Übersicht

print(f" ID: {p.get('id')}, Name: {p.get('project\_name')}, Anlagentyp: {p.get('anlage\_type')}, kWp: {p.get('anlage\_kwp')}, Modul: {p.get('module\_model\_name')}")

# --- Teste Render Funktion direkt (benötigt Streamlit Kontext) ---

# Dies funktioniert nicht durch direktes Ausführen von crm.py,

# da st.\* Funktionen nur in einer Streamlit App laufen.

# Um render\_crm zu testen, muss die Funktion in gui.py aufgerufen werden.

# print("\nTeste render\_crm Funktion (benötigt Streamlit Kontext)...")

# try:

# # Simulieren des Aufrufs von gui.py

# class DummyTextsForCrmTest: # Mock für das texts Dictionary

# def get(self, key, default): return default

#

# render\_crm(DummyTextsForCrmTest(), database.get\_db\_connection) # Rufe render\_crm auf (mit Dummy Texten und realer DB-Funktion)

# print("render\_crm aufgerufen (Output nur in Streamlit App sichtbar).")

# except Exception as e:

# print(f"FEHLER beim Testen von render\_crm: {e}")

# traceback.print\_exc()

conn\_test.close() # Verbindung am Ende schließen

else:

print("\nFEHLER: Keine Datenbankverbindung für Test verfügbar.")

except ImportError:

print("\nFEHLER: Kann database.py nicht importieren, CRM Test nicht möglich.")

except Exception as e:

print(f"\nEin Fehler ist während des CRM Testlaufs aufgetreten: {e}")

traceback.print\_exc()

print("\n--- Testlauf beendet ---")

**data\_input.py:**

# data\_input.py

# Eingabemasken für Kundendaten, Verbrauchsdaten und Bedarfsanalyse

import streamlit as st

import pandas as pd

import os

import re # Importiere re für das Parsen der Adresse

from typing import Dict, Any, Optional, List # Importiere List für Typ-Hints

import json

import traceback

# --- Import von Modulen, die Daten oder Logik bereitstellen ---

# Wir machen die Imports robuster, indem wir Dummy-Funktionen definieren,# falls der Import des Datenbankmoduls fehlschlägt.# Wichtige: Diese Dummy-Funktionen müssen existieren, bevor der Importversuch kommt.# Definiere Dummy-Datenbankfunktionen als Fallback für data\_input# Diese Dummies geben nur leere Listen oder None zurück und rufen KEINE st.error/warning hier auf Top-Level

def Dummy\_get\_db\_connection\_input():

# st.error("Datenbankverbindung nicht verfügbar. Dropdown-Listen können nicht geladen werden.") # MOVED TO RENDER FUNCTION IF NEEDED

return None

def Dummy\_list\_products\_input(\*args, \*\*kwargs):

# st.warning("Produktdaten können nicht geladen werden (Datenbankfehler).") # MOVED TO RENDER FUNCTION IF NEEDED

# print("Dummy\_list\_products\_input called") # Debugging

return [] # Immer eine leere Liste zurückgeben, damit Iteration möglich ist

def Dummy\_get\_product\_by\_model\_name\_input(\*args, \*\*kwargs):

# st.warning("Produktdetails können nicht geladen werden (Datenbankfehler).") # MOVED TO RENDER FUNCTION IF NEEDED

# print("Dummy\_get\_product\_by\_model\_name\_input called") # Debugging

return None

def Dummy\_load\_admin\_setting\_input(key, default=None):

# print(f"Dummy load\_admin\_setting\_input called for key: {key}") # For debugging dummy usage

# Stellt sicher, dass zumindest Standardlisten zurückgegeben werden, wenn Admin-Settings nicht geladen werden können

if key == 'salutation\_options': return ['Herr', 'Frau', 'Familie']

if key == 'title\_options': return ['Dr.', 'Prof.', 'Mag.', 'Ing.', None]

if key == 'bundesland\_options': return ['Bitte Admin prüfen (DB Fehler)']

if key == 'region\_options': return ['Bitte Admin prüfen (DB Fehler)']

if key == 'dachart\_options': return ['Bitte Admin prüfen (DB Fehler)']

if key == 'dachdeckung\_options': return ['Bitte Admin prüfen (DB Fehler)']

return default # Fallback für andere Schlüssel

# Versuche nun, die reale Datenbankfunktion zu importieren# Weisen die importierte Funktion oder die Dummy-Funktion den Variablen zu.# Die Fehlerbehandlung mit st.error/warning erfolgt erst IN der render\_data\_input Funktion.

try:

# WICHTIG: Verwende absolute Imports oder stelle sicher, dass dein PYTHONPATH korrekt ist.

# 'from database import ...' geht davon aus, dass das Modul 'database' im Python-Pfad ist.

# Wenn gui.py und database.py im selben Ordner liegen und du 'streamlit run gui.py'

# in diesem Ordner ausführst, sollte der Import 'from database import ...' funktionieren.

from database import get\_db\_connection as real\_get\_db\_connection, init\_db as real\_init\_db, save\_admin\_setting as real\_save\_admin\_setting, load\_admin\_setting as real\_load\_admin\_setting

# Check if imported objects are callable/valid (optional, aber gut)

if not callable(real\_get\_db\_connection): raise ImportError("Imported get\_db\_connection is not callable.")

if not callable(real\_load\_admin\_setting): raise ImportError("Imported load\_admin\_setting is not callable.")

# Wenn der Import und der Check erfolgreich sind, weisen wir die realen Funktionen unseren Variablen zu

get\_db\_connection\_safe = real\_get\_db\_connection # Umbenannt, um Verwechslung mit Dummy zu vermeiden

load\_admin\_setting\_safe = real\_load\_admin\_setting # Umbenannt

except ImportError as e:

# Bei Importfehler, weise die Dummy-Funktionen zu

get\_db\_connection\_safe = Dummy\_get\_db\_connection\_input

load\_admin\_setting\_safe = Dummy\_load\_admin\_setting\_input

# Logge den Fehler, aber verwende hier kein st.error

print(f"data\_input.py: Importfehler für database.py: {e}. Dummy DB Funktionen werden genutzt.")

except Exception as e:

# Bei anderen Fehlern beim Laden, weise Dummies zu

get\_db\_connection\_safe = Dummy\_get\_db\_connection\_input

load\_admin\_setting\_safe = Dummy\_load\_admin\_setting\_input

print(f"data\_input.py: Fehler beim Laden von database.py: {e}. Dummy DB Funktionen werden genutzt.")

# Versuche nun, die reale Produktdatenbankfunktion zu importieren (benötigt database)

try:

# WICHTIG: Stelle sicher, dass product\_db.py im selben Ordner liegt wie gui.py und der Import funktioniert

from product\_db import list\_products as real\_list\_products, get\_product\_by\_model\_name as real\_get\_product\_by\_model\_name

# Check if imported objects are callable/valid

if not callable(real\_list\_products): raise ImportError("Imported list\_products is not callable.")

if not callable(real\_get\_product\_by\_model\_name): raise ImportError("Imported get\_product\_by\_model\_name is not callable.")

# Wenn der Import und der Check erfolgreich sind, weisen wir die realen Funktionen zu.

list\_products\_safe = real\_list\_products # Umbenannt

get\_product\_by\_model\_name\_safe = real\_get\_product\_by\_model\_name # Umbenannt

except ImportError as e:

# Bei Importfehler, weise die Dummy-Funktionen zu.

list\_products\_safe = Dummy\_list\_products\_input

get\_product\_by\_model\_name\_safe = Dummy\_get\_product\_by\_model\_name\_input

# Logge den Fehler, aber verwende hier kein st.error

print(f"data\_input.py: Importfehler für product\_db.py: {e}. Dummy Produkt Funktionen werden genutzt.")

except Exception as e:

# Bei anderen Fehlern beim Laden, weise Dummies zu.

list\_products\_safe = Dummy\_list\_products\_input

get\_product\_by\_model\_name\_safe = Dummy\_get\_product\_by\_model\_name\_input

print(f"data\_input.py: Fehler im product\_db.py Modul beim Laden: {e}. Dummy Produkt Funktionen werden genutzt.")

# --- Funktion zum Rendern der Eingabemasken ---

def render\_data\_input(texts: Dict[str, str]) -> Optional[Dict[str, Any]]:

"""

Rendert die Eingabemasken für Kundendaten und Bedarfsanalyse.

Lädt die notwendigen Listen aus der Datenbank (falls verfügbar) IN DIESER FUNKTION.

Fügt einen "Bitte wählen" Platzhalter zu Dropdowns hinzu.

Gibt ein Dictionary mit den gesammelten Daten zurück.

"""

# Dictionary zum Speichern aller Eingaben

inputs: Dict[str, Any] = {}

inputs['customer\_data'] = {} # Für A.1 Kundendaten

inputs['project\_details'] = {} # Für A.2, A.3 etc.

# --- Laden von Listen für Dropdowns aus der Datenbank (INNERHALB der Render Funktion) ---

# Dies wird jetzt IN der Render-Funktion gemacht, nachdem st.set\_page\_config in gui.py gelaufen ist.

# Wir nutzen die '\_safe' Funktionen, die oben zugewiesen wurden (echt oder Dummy).

# Setze Default-Fallback-Listen am Anfang der Funktion

SALUTATION\_OPTIONS = ['Keine Anreden gefunden']

TITLE\_OPTIONS = ['Keine Titel gefunden']

BUNDESLAND\_OPTIONS = ['Keine Bundesländer gefunden']

REGION\_OPTIONS = ['Keine Regionen gefunden']

DACHART\_OPTIONS = ['Keine Dacharten gefunden']

DACHDECKUNG\_OPTIONS = ['Keine Dachdeckungen gefunden']

MODULE\_LIST\_MODELS = ['Keine Module gefunden'] # Default Fallback für Produktdaten

INVERTER\_LIST\_MODELS = ['Keine Wechselrichter gefunden']

STORAGE\_LIST\_MODELS = ['Keine Speicher gefunden']

WALLBOX\_LIST\_MODELS = ['Keine Wallboxen gefunden']

EMS\_LIST\_MODELS = ['Keine EMS gefunden']

OPTIMIZER\_LIST\_MODELS = ['Keine Optimierer gefunden']

CARPORT\_LIST\_MODELS = ['Keine Carports gefunden'] # Neue Liste

NOTSTROM\_LIST\_MODELS = ['Keine Notstromlösungen gefunden'] # Neue Liste

TIERABWEHR\_LIST\_MODELS = ['Keine Tierabwehr gefunden'] # Neue Liste

db\_available = False # Flag, ob DB-Zugriff möglich ist

conn = None # Verbindung initial None

try:

# Versuche, eine echte Verbindung zu bekommen, wenn die Funktion nicht der Dummy ist

if get\_db\_connection\_safe is not Dummy\_get\_db\_connection\_input:

conn = get\_db\_connection\_safe() # Nutze die real\_get\_db\_connection Funktion (falls verfügbar)

if conn: # Prüfe, ob die Verbindung erfolgreich war (auch bei realer Funktion kann connect fehlschlagen)

db\_available = True

# Lade Listen NUR, wenn DB-Verbindung erfolgreich

# Nutze die real\_load\_admin\_setting Funktion (falls verfügbar)

# Stellen Sie sicher, dass die geladenen Optionen auch Listen sind oder geben Sie einen Fallback zurück

SALUTATION\_OPTIONS = load\_admin\_setting\_safe('salutation\_options', ['Herr', 'Frau', 'Familie'])

if not isinstance(SALUTATION\_OPTIONS, list): SALUTATION\_OPTIONS = ['Fehler beim Laden Anreden']

TITLE\_OPTIONS = load\_admin\_setting\_safe('title\_options', ['Dr.', 'Prof.', 'Mag.', 'Ing.', None])

if not isinstance(TITLE\_OPTIONS, list): TITLE\_OPTIONS = ['Fehler beim Laden Titel']

BUNDESLAND\_OPTIONS = load\_admin\_setting\_safe('bundesland\_options', ['Baden-Württemberg', 'Bayern', 'Berlin', 'Brandenburg', 'Bremen', 'Hamburg', 'Hessen', 'Mecklenburg-Vorpommern', 'Niedersachsen', 'Nordrhein-Westfalen', 'Rheinland-Pfalz', 'Saarland', 'Sachsen', 'Sachsen-Anhalt', 'Schleswig-Holstein', 'Thüringen'])

if not isinstance(BUNDESLAND\_OPTIONS, list): BUNDESLAND\_OPTIONS = ['Fehler beim Laden Bundesländer']

REGION\_OPTIONS = load\_admin\_setting\_safe('region\_options', ['Nord', 'Ost', 'Süd', 'West', 'Mitte'])

if not isinstance(REGION\_OPTIONS, list): REGION\_OPTIONS = ['Fehler beim Laden Regionen']

DACHART\_OPTIONS = load\_admin\_setting\_safe('dachart\_options', ['Satteldach', 'Satteldach mit Gaube', 'Pultdach', 'Flachdach', 'Walmdach', 'Krüppelwalmdach', 'Zeltdach', 'Sonstiges'])

if not isinstance(DACHART\_OPTIONS, list): DACHART\_OPTIONS = ['Fehler beim Laden Dacharten']

DACHDECKUNG\_OPTIONS = load\_admin\_setting\_safe('dachdeckung\_options', ['Frankfurter Pfannen', 'Trapezblech', 'Tonziegel', 'Bieberschwanz', 'Schiefer', 'Bitumen', 'Ethernit', 'Schindeln', 'Sonstiges'])

if not isinstance(DACHDECKUNG\_OPTIONS, list): DACHDECKUNG\_OPTIONS = ['Fehler beim Laden Dachdeckungen']

# Lade Produktdaten Listen NUR wenn die list\_products\_safe Funktion nicht der Dummy ist

if list\_products\_safe is not Dummy\_list\_products\_input:

MODULE\_LIST\_MODELS = [p.get('model\_name', f"ID:{p.get('id', 'N/A')}") for p in list\_products\_safe(category='Modul')] or ['Keine Module gefunden/geladen']

INVERTER\_LIST\_MODELS = [p.get('model\_name', f"ID:{p.get('id', 'N/A')}") for p in list\_products\_safe(category='Wechselrichter')] or ['Keine Wechselrichter gefunden/geladen']

STORAGE\_LIST\_MODELS = [p.get('model\_name', f"ID:{p.get('id', 'N/A')}") for p in list\_products\_safe(category='Batteriespeicher')] or ['Keine Speicher gefunden/geladen']

WALLBOX\_LIST\_MODELS = [p.get('model\_name', f"ID:{p.get('id', 'N/A')}") for p in list\_products\_safe(category='Wallbox')] or ['Keine Wallboxen gefunden/geladen']

EMS\_LIST\_MODELS = [p.get('model\_name', f"ID:{p.get('id', 'N/A')}") for p in list\_products\_safe(category='Energiemanagementsystem')] or ['Keine EMS gefunden/geladen']

OPTIMIZER\_LIST\_MODELS = [p.get('model\_name', f"ID:{p.get('id', 'N/A')}") for p in list\_products\_safe(category='Leistungsoptimierer')] or ['Keine Optimierer gefunden/geladen']

CARPORT\_LIST\_MODELS = [p.get('model\_name', f"ID:{p.get('id', 'N/A')}") for p in list\_products\_safe(category='Carport')] or ['Keine Carports gefunden/geladen'] # Neue Liste laden

NOTSTROM\_LIST\_MODELS = [p.get('model\_name', f"ID:{p.get('id', 'N/A')}") for p in list\_products\_safe(category='Notstromversorgung')] or ['Keine Notstromlösungen gefunden/geladen'] # Neue Liste laden

TIERABWEHR\_LIST\_MODELS = [p.get('model\_name', f"ID:{p.get('id', 'N/A')}") for p in list\_products\_safe(category='Tierabwehrschutz')] or ['Keine Tierabwehr gefunden/geladen'] # Neue Liste laden

else:

# Wenn list\_products\_safe der Dummy ist, bleiben die Listen die Default-Fallbacks

pass # Nichts tun, Fallbacks sind schon gesetzt

# Wenn get\_db\_connection\_safe der Dummy IST ODER conn None ist, zeige die Fehlermeldung

if not db\_available:

st.error(texts.get("db\_connection\_unavailable", "Datenbankverbindung nicht verfügbar. Dropdown-Listen können nicht geladen werden.")) # Zeige Fehler nur hier

except Exception as e:

# Dieser Block fängt Fehler beim \*Nutzen\* der (eventuell Dummy-) DB-Funktionen innerhalb der Render-Funktion ab

st.error(f"{texts.get('error\_loading\_dropdowns', 'Fehler beim Vorbereiten der Dropdown-Listen:')} {e}")

traceback.print\_exc()

# Listen bleiben Fallback-Standardwerte (die am Anfang der Funktion gesetzt wurden)

finally:

if conn:

conn.close()

# --- Füge "Bitte wählen" Platzhalter zu den Dropdown-Listen hinzu ---

please\_select\_text = texts.get("please\_select\_option", "--- Bitte wählen ---")

# Listen, die immer einen Platzhalter bekommen sollen

lists\_to\_prefix = [

SALUTATION\_OPTIONS,

TITLE\_OPTIONS,

BUNDESLAND\_OPTIONS,

REGION\_OPTIONS,

DACHART\_OPTIONS,

DACHDECKUNG\_OPTIONS,

MODULE\_LIST\_MODELS,

INVERTER\_LIST\_MODELS,

STORAGE\_LIST\_MODELS,

WALLBOX\_LIST\_MODELS,

EMS\_LIST\_MODELS,

OPTIMIZER\_LIST\_MODELS,

CARPORT\_LIST\_MODELS,

NOTSTROM\_LIST\_MODELS,

TIERABWEHR\_LIST\_MODELS

]

for option\_list in lists\_to\_prefix:

# Prüfe, ob die Liste nicht leer ist ODER ob sie bereits den Platzhalter enthält

if option\_list and option\_list[0] != please\_select\_text:

option\_list.insert(0, please\_select\_text)

elif not option\_list:

# Wenn die Liste leer ist, füge nur den Platzhalter hinzu

option\_list.append(please\_select\_text)

st.subheader(texts.get("customer\_data\_header", "Kundendaten"))

# 1. Kundendaten / Grunddaten (A.1)

with st.expander(texts.get("customer\_data\_header", "Kundendaten"), expanded=True):

col1, col2, col3 = st.columns(3)

with col1:

# Anlagentyp: Hier möchten wir wahrscheinlich keinen "Bitte wählen", da es nur 2 Optionen gibt und eine immer Standard ist

session\_state\_anlage\_type = st.session\_state.get('anlage\_type', 'Neuanlage')

try: initial\_index\_anlage\_type = ['Neuanlage', 'Bestandsanlage'].index(session\_state\_anlage\_type)

except ValueError: initial\_index\_anlage\_type = 0 # Fallback

inputs['project\_details']['anlage\_type'] = st.selectbox(texts.get("anlage\_type\_label", "Anlagentyp"), options=['Neuanlage', 'Bestandsanlage'], index=initial\_index\_anlage\_type, key='anlage\_type') # Aus A.1

with col2:

# Einspeisetyp: Auch hier wahrscheinlich kein "Bitte wählen"

session\_state\_feed\_in\_type = st.session\_state.get('feed\_in\_type', 'Teileinspeisung')

try: initial\_index\_feed\_in\_type = ['Teileinspeisung', 'Volleinspeisung'].index(session\_state\_feed\_in\_type)

except ValueError: initial\_index\_feed\_in\_type = 0 # Fallback

inputs['project\_details']['feed\_in\_type'] = st.selectbox(texts.get("feed\_in\_type\_label", "Einspeisetyp"), options=['Teileinspeisung', 'Volleinspeisung'], index=initial\_index\_feed\_in\_type, key='feed\_in\_type') # Aus A.1

with col3:

# Kundentyp: Auch hier wahrscheinlich kein "Bitte wählen"

session\_state\_customer\_type = st.session\_state.get('customer\_type', 'Privat')

try: initial\_index\_customer\_type = ['Privat', 'Gewerblich'].index(session\_state\_customer\_type)

except ValueError: initial\_index\_customer\_type = 0 # Fallback

inputs['customer\_data']['type'] = st.selectbox(texts.get("customer\_type\_label", "Kundentyp"), options=['Privat', 'Gewerblich'], index=initial\_index\_customer\_type, key='customer\_type') # Aus A.1

col4, col5, col6 = st.columns(3)

with col4:

# Anrede: Hier wollen wir "Bitte wählen"

options\_salutation = SALUTATION\_OPTIONS # Liste wurde oben mit Placeholder ergänzt

session\_state\_salutation = st.session\_state.get('salutation', please\_select\_text) # Default ist der Platzhalter

try: initial\_index\_salutation = options\_salutation.index(session\_state\_salutation) if session\_state\_salutation in options\_salutation else 0

except ValueError: initial\_index\_salutation = 0 # Fallback

inputs['customer\_data']['salutation'] = st.selectbox(texts.get("salutation\_label", "Anrede"), options=options\_salutation, index=initial\_index\_salutation, key='salutation') # Aus A.1

if inputs['customer\_data']['salutation'] == please\_select\_text: inputs['customer\_data']['salutation'] = None # Speichere None, wenn Platzhalter gewählt

with col5:

# Titel: Hier wollen wir "Bitte wählen"

options\_title = TITLE\_OPTIONS # Liste wurde oben mit Placeholder ergänzt

session\_state\_title = st.session\_state.get('title', please\_select\_text) # Default ist der Platzhalter (da None jetzt auch eine Option ist)

try: initial\_index\_title = options\_title.index(session\_state\_title) if session\_state\_title in options\_title else 0

except ValueError: initial\_index\_title = 0 # Fallback

inputs['customer\_data']['title'] = st.selectbox(texts.get("title\_label", "Titel"), options=options\_title, index=initial\_index\_title, key='title') # Aus A.1

if inputs['customer\_data']['title'] == please\_select\_text or inputs['customer\_data']['title'] == texts.get("none\_option", "Kein Titel"): inputs['customer\_data']['title'] = None # Speichere None, wenn Platzhalter oder "Kein Titel" gewählt

with col6:

inputs['customer\_data']['first\_name'] = st.text\_input(texts.get("first\_name\_label", "Vorname"), value=st.session\_state.get('first\_name', ''), key='first\_name') # Use session state and key

col7, col8 = st.columns(2)

with col7:

inputs['customer\_data']['last\_name'] = st.text\_input(texts.get("last\_name\_label", "Nachname"), value=st.session\_state.get('last\_name', ''), key='last\_name') # Use session state and key

with col8:

inputs['customer\_data']['num\_persons'] = st.number\_input(texts.get("num\_persons\_label", "Anzahl Personen im Haushalt"), min\_value=1, value=st.session\_state.get('num\_persons', 1), key='num\_persons') # Aus A.1

# Use session state to remember parsed address components (Initialize if they don't exist)

if 'address\_parsed' not in st.session\_state: st.session\_state['address\_parsed'] = ''

if 'house\_number\_parsed' not in st.session\_state: st.session\_state['house\_number\_parsed'] = ''

if 'zip\_code\_parsed' not in st.session\_state: st.session\_state['zip\_code\_parsed'] = ''

if 'city\_parsed' not in st.session\_state: st.session\_state['city\_parsed'] = ''

if 'state\_parsed' not in st.session\_state: st.session\_state['state\_parsed'] = '' # Add state\_parsed

if 'region\_parsed' not in st.session\_state: st.session\_state['region\_parsed'] = '' # Add region\_parsed

# Initialisieren Sie Session State für die separaten Felder nur, wenn sie noch nicht existieren

# Der value-Parameter im Widget wird den Wert aus dem Session State laden

if 'full\_address' not in st.session\_state: st.session\_state['full\_address'] = ''

if 'address' not in st.session\_state: st.session\_state['address'] = ''

if 'house\_number' not in st.session\_state: st.session\_state['house\_number'] = ''

if 'zip\_code' not in st.session\_state: st.session\_state['zip\_code'] = ''

if 'city' not in st.session\_state: st.session\_state['city'] = ''

# State und Region werden unten über Selectboxen verwaltet

if 'email' not in st.session\_state: st.session\_state['email'] = ''

if 'phone\_landline' not in st.session\_state: st.session\_state['phone\_landline'] = ''

if 'phone\_mobile' not in st.session\_state: st.session\_state['phone\_mobile'] = ''

# Neues Feld für volle Adresse + Button zum Parsen (Feature A.1 - automatische Koordinaten)

full\_address\_input = st.text\_input(texts.get("full\_address\_label", "Komplette Adresse"),

value=st.session\_state.get('full\_address', ''), # Nutze Session State als Value

help=texts.get("full\_address\_help", "Fügen Sie hier die komplette Adresse ein (z.B. 'Musterweg 18, 12345 Musterstadt, Deutschland')"),

key='full\_address') # Nutze Key für Session State Management

# Button zum Parsen/Übernehmen - DIESER BUTTON LÖST EIN NEULADEN AUS!

if st.button(texts.get("parse\_address\_button", "Daten aus Adresse übernehmen")):

# HIER kommt später die Logik zum Parsen der Adresse (z.B. mit regex oder Geocoding API VOR dem Geocoding selbst)

# Für jetzt nur einfacher Versuch, PLZ, Ort und Ggf. Straße zu übernehmen

# Parsing-Ergebnisse im Session State speichern

parsed\_zip\_city = re.search(r'(\d{5})\s+(.+)', full\_address\_input) # Einfaches Muster PLZ Ort

if parsed\_zip\_city:

st.session\_state['zip\_code\_parsed'] = parsed\_zip\_city.group(1)

st.session\_state['city\_parsed'] = parsed\_zip\_city.group(2).split(',')[0].strip() # Nimm ersten Teil vor Komma als Ort

st.success("PLZ und Ort erfolgreich geparst.")

else:

st.warning("PLZ und Ort konnten nicht automatisch geparst werden.")

# Sehr einfacher Versuch, Straße und Hausnummer zu trennen (funktioniert nicht immer)

# Annahme: Straße und Hausnummer sind der erste Teil vor dem ersten Komma

first\_part = full\_address\_input.split(',')[0].strip()

parsed\_street\_hnr = re.search(r'(.+)\s+(\d+[a-zA-Z]?)$', first\_part) # Strasse Hausnr am Ende des ersten Teils

if parsed\_street\_hnr:

st.session\_state['address\_parsed'] = parsed\_street\_hnr.group(1).strip() # Speichere Straße

st.session\_state['house\_number\_parsed'] = parsed\_street\_hn.group(2).strip() # Speichere Hausnummer separat # <<< Fehlerhafte Zeile

st.success("Straße und Hausnummer erfolgreich geparst.")

else:

# Wenn Trennung nicht klappt, speichere ganze erste Teil als Adresse und leere Hausnummer

st.session\_state['address\_parsed'] = first\_part

st.session\_state['house\_number\_parsed'] = ''

st.warning("Straße und Hausnummer konnten nicht eindeutig getrennt werden.")

# TODO: Logik zum Parsen von Bundesland/Region hinzufügen und in session state speichern (state\_parsed, region\_parsed)

# WICHTIG: Nach dem Parsen müssen wir die Widgets UNTEN, die diese geparsten Werte als Default-Value verwenden,

# neu rendern lassen, was Streamlit automatisch nach einem Button-Klick macht.

# Wir setzen KEINE Werte direkt in inputs['customer\_data'] HIER, da der Widget-Zustand maßgeblich ist.

pass # Dummy pass nach dem Button-Logic

# Separate Felder für PLZ, Ort etc. anzeigen.

# Die values dieser Felder lesen JETZT aus dem Session State.

# st.session\_state['zip\_code\_parsed'] und st.session\_state['city\_parsed']

# haben höhere Priorität als st.session\_state['zip\_code'] und st.session\_state['city']

# als Default-Value, wenn der Parse-Button gedrückt wurde.

col9, col10, col11 = st.columns(3)

with col9:

# Nutze geparsten Wert als Value, wenn vorhanden, sonst Session State, sonst leer

# Der key='zip\_code' sorgt dafür, dass Streamlit den aktuellen Wert in st.session\_state['zip\_code'] speichert

inputs['customer\_data']['zip\_code'] = st.text\_input(texts.get("zip\_code\_label", "PLZ"), value=st.session\_state.get('zip\_code\_parsed', st.session\_state.get('zip\_code', '')), key='zip\_code')

with col10:

# Nutze geparsten Wert als Value, wenn vorhanden, sonst Session State, sonst leer

# Der key='city' sorgt dafür, dass Streamlit den aktuellen Wert in st.session\_state['city'] speichert

inputs['customer\_data']['city'] = st.text\_input(texts.get("city\_label", "Ort"), value=st.session\_state.get('city\_parsed', st.session\_state.get('city', '')), key='city')

with col11:

# Use session state for selectbox

# Ensure BUNDESLAND\_OPTIONS is not empty

options\_bundesland = BUNDESLAND\_OPTIONS # Liste wurde oben mit Placeholder ergänzt

# Der geparste Bundesland-Wert müsste auch in Session State gespeichert werden

session\_state\_state = st.session\_state.get('state\_parsed', st.session\_state.get('state', options\_bundesland[0] if options\_bundesland else None))

# Finde den Index des Werts im Session State in der Liste mit Placeholder (Index 0)

try:

# Suche im Session State Wert in der Liste options\_bundesland

# Wenn gefunden, nimm den Index. Wenn nicht (auch None), Fallback auf 0 (Platzhalter)

initial\_index\_state = options\_bundesland.index(session\_state\_state) if session\_state\_state is not None and session\_state\_state in options\_bundesland else 0

except ValueError: initial\_index\_state = 0 # Fallback

# Der key='state' sorgt dafür, dass Streamlit den aktuellen Wert in st.session\_state['state'] speichert

inputs['customer\_data']['state'] = st.selectbox(texts.get("state\_label", "Bundesland"), options=options\_bundesland, index=initial\_index\_state, key='state') # Aus A.1

# Wenn Platzhalter gewählt, speichere None

if inputs['customer\_data']['state'] == please\_select\_text: inputs['customer\_data']['state'] = None

# Separate Felder für Straße und Hausnummer

# Standardwerte kommen hier auch aus dem Session State (geparst oder manuell)

# Nutze geparsten Wert als Value, wenn vorhanden, sonst Session State, sonst leer

# Der key='address' sorgt dafür, dass Streamlit den aktuellen Wert in st.session\_state['address'] speichert

inputs['customer\_data']['address'] = st.text\_input(texts.get("street\_label", "Straße"), value=st.session\_state.get('address\_parsed', st.session\_state.get('address', '')), key='address')

# Nutze geparsten Wert als Value, wenn vorhanden, sonst Session State, sonst leer

# Der key='house\_number' sorgt dafür, dass Streamlit den aktuellen Wert in st.session\_state['house\_number'] speichert

inputs['customer\_data']['house\_number'] = st.text\_input(texts.get("house\_number\_label", "Hausnummer"), value=st.session\_state.get('house\_number\_parsed', st.session\_state.get('house\_number', '')), key='house\_number')

# Use session state for selectbox (Region)

options\_region = REGION\_OPTIONS # Liste wurde oben mit Placeholder ergänzt

# Nutze geparsten Wert als Value, wenn vorhanden, sonst Session State, sonst Default-Option

session\_state\_region = st.session\_state.get('region\_parsed', st.session\_state.get('region', options\_region[0] if options\_region else None))

try:

initial\_index\_region = options\_region.index(session\_state\_region) if session\_state\_region is not None and session\_state\_region in options\_region else 0

except ValueError: initial\_index\_region = 0 # Fallback

# Der key='region' sorgt dafür, dass Streamlit den aktuellen Wert in st.session\_state['region'] speichert

inputs['customer\_data']['region'] = st.selectbox(texts.get("region\_label", "Region"), options=options\_region, index=initial\_index\_region, key='region') # Aus A.1

# Wenn Platzhalter gewählt, speichere None

if inputs['customer\_data']['region'] == please\_select\_text: inputs['customer\_data']['region'] = None

col12, col13, col14 = st.columns(3)

with col12:

inputs['customer\_data']['email'] = st.text\_input(texts.get("email\_label", "E-Mail"), value=st.session\_state.get('email', ''), key='email') # Use session state and key

with col13:

inputs['customer\_data']['phone\_landline'] = st.text\_input(texts.get("phone\_landline\_label", "Telefon (Festnetz)"), value=st.session\_state.get('phone\_landline', ''), key='phone\_landline') # Use session state and key

with col14:

inputs['customer\_data']['phone\_mobile'] = st.text\_input(texts.get("phone\_mobile\_label", "Telefon (Mobil)"), value=st.session\_state.get('phone\_mobile', ''), key='phone\_mobile') # Aus A.1

# Button für Luftbilder (A.3 / map\_integration.py - Implementierung kommt später)

st.button(texts.get("show\_aerial\_images\_button", "Luftbilder anzeigen (Placeholder)"), disabled=True) # Beispiel Platzhalter

# 2. Bedarfsanalyse (A.2)

st.subheader(texts.get("consumption\_analysis\_header", "Bedarfsanalyse"))

# 2.1 Verbräuche und Kosten (A.2.1)

with st.expander(texts.get("consumption\_costs\_header", "Verbräuche und Kosten"), expanded=True):

col15, col16 = st.columns(2)

with col15:

inputs['project\_details']['consumption\_household\_kwh\_yr'] = st.number\_input(texts.get("annual\_consumption\_kwh\_label", "Jahresverbrauch Haushalt (kWh)"), min\_value=0.0, value=st.session\_state.get('consumption\_household\_kwh\_yr', 3500.0), key='consumption\_household\_kwh\_yr') # Aus A.2.1

with col16:

inputs['project\_details']['consumption\_heating\_kwh\_yr'] = st.number\_input(texts.get("annual\_heating\_kwh\_optional\_label", "Jahresverbrauch Heizung (kWh, optional)"), min\_value=0.0, value=st.session\_state.get('consumption\_heating\_kwh\_yr', 0.0), key='consumption\_heating\_kwh\_yr') # Aus A.2.1

# Ergebnis: Gesamtverbrauch jährlich (Wird hier berechnet zur Anzeige, wird im analysis Modul richtig genutzt)

total\_consumption\_kwh\_yr = inputs['project\_details'].get('consumption\_household\_kwh\_yr', 0) + inputs['project\_details'].get('consumption\_heating\_kwh\_yr', 0)

st.info(f"{texts.get('total\_annual\_consumption\_label', 'Gesamtjahresverbrauch (kWh)')}: {total\_consumption\_kwh\_yr:.2f}")

inputs['project\_details']['total\_consumption\_kwh\_yr'] = total\_consumption\_kwh\_yr # Auch ins dict speichern (das ist OK, da inputs[] nicht session state ist)

col17, col18 = st.columns(2)

with col17:

inputs['project\_details']['costs\_household\_euro\_mo'] = st.number\_input(texts.get("monthly\_costs\_household\_label", "Monatliche Kosten Haushalt (€)"), min\_value=0.0, value=st.session\_state.get('costs\_household\_euro\_mo', 80.0), key='costs\_household\_euro\_mo') # Aus A.2.1

with col18:

inputs['project\_details']['costs\_heating\_euro\_mo'] = st.number\_input(texts.get("monthly\_costs\_heating\_optional\_label", "Monatliche Kosten Heizung (€, optional)"), min\_value=0.0, value=st.session\_state.get('costs\_heating\_euro\_mo', 0.0), key='costs\_heating\_euro\_mo') # Aus A.2.1

# Ergebnis: Gesamtkosten jährlich (Wird hier berechnet zur Anzeige)

total\_costs\_euro\_yr = (inputs['project\_details'].get('costs\_household\_euro\_mo', 0) + inputs['project\_details'].get('costs\_heating\_euro\_mo', 0)) \* 12

st.info(f"{texts.get('total\_annual\_costs\_label', 'Gesamtjahreskosten (€)')}: {total\_costs\_euro\_yr:.2f}")

inputs['project\_details']['total\_costs\_euro\_yr'] = total\_costs\_euro\_yr # Auch ins dict speichern

# Ergebnis: Stromtarif (Wird hier berechnet zur Anzeige, im analysis Modul richtig genutzt)

# Vermeide Division durch Null

electricity\_price\_kwh = (total\_costs\_euro\_yr / total\_consumption\_kwh\_yr) if total\_consumption\_kwh\_yr > 0 else 0.0

st.info(f"{texts.get('electricity\_price\_label', 'Strompreis (€/kWh)')}: {electricity\_price\_kwh:.4f}")

inputs['project\_details']['electricity\_price\_kwh'] = electricity\_price\_kwh # Speichern für spätere Berechnungen

# Button/Feld: Verbrauchswerte unbekannt (A.2.1) - Implementierung kommt später

st.button(texts.get("consumption\_unknown\_button", "Verbrauchswerte unbekannt? (Lastprofile)"), disabled=True) # Beispiel Platzhalter

# 2.2 Daten des Gebäudes (A.2.2)

with st.expander(texts.get("building\_data\_header", "Daten des Gebäudes"), expanded=True):

col19, col20 = st.columns(2)

with col19:

inputs['project\_details']['build\_year'] = st.number\_input(texts.get("build\_year\_label", "Baujahr des Hauses"), min\_value=1800, value=st.session\_state.get('build\_year', 2000), step=1, key='build\_year') # Aus A.2.2

# Automatische Meldung zur Zählerschrank/Elektrik Kompatibilität (Logik basierend auf Eingabe)

if inputs['project\_details']['build\_year'] != 0: # Zeige Meldung nur wenn Baujahr gesetzt ist

if inputs['project\_details']['build\_year'] < 1960:

st.warning(texts.get("build\_year\_warning\_old", "❗️ Zählerschrank/Hauselektrik muss wahrscheinlich geprüft/erneuert werden. Mehrkosten möglich!"))

elif inputs['project\_details']['build\_year'] < 2000:

st.info(texts.get("build\_year\_info\_mid", "ℹ️ Zählerschrank/Hauselektrik sollte geprüft werden. Ggf. Anpassungen nötig."))

else:

st.success(texts.get("build\_year\_success\_new", "✅ Zählerschrank/Hauselektrik wahrscheinlich kompatibel. Geringe Mehrkosten erwartet."))

with col20:

# Use session state for selectbox

options\_roof\_type = DACHART\_OPTIONS # Liste wurde oben mit Placeholder ergänzt

session\_state\_roof\_type = st.session\_state.get('roof\_type', please\_select\_text) # Default ist der Platzhalter

try: initial\_index\_roof\_type = options\_roof\_type.index(session\_state\_roof\_type) if session\_state\_roof\_type in options\_roof\_type else 0

except ValueError: initial\_index\_roof\_type = 0 # Fallback

inputs['project\_details']['roof\_type'] = st.selectbox(texts.get("roof\_type\_label", "Dachart"), options=options\_roof\_type, index=initial\_index\_roof\_type, key='roof\_type') # Aus A.2.2

if inputs['project\_details']['roof\_type'] == please\_select\_text: inputs['project\_details']['roof\_type'] = None # Speichere None, wenn Platzhalter gewählt

col21, col22 = st.columns(2)

with col21:

# Use session state for selectbox

options\_roof\_covering = DACHDECKUNG\_OPTIONS # Liste wurde oben mit Placeholder ergänzt

session\_state\_roof\_covering = st.session\_state.get('roof\_covering\_type', please\_select\_text) # Default ist der Platzhalter

try: initial\_index\_roof\_covering = options\_roof\_covering.index(session\_state\_roof\_covering) if session\_state\_roof\_covering in options\_roof\_covering else 0

except ValueError: initial\_index\_roof\_covering = 0 # Fallback

inputs['project\_details']['roof\_covering\_type'] = st.selectbox(texts.get("roof\_covering\_label", "Dachdeckungsart"), options=options\_roof\_covering, index=initial\_index\_roof\_covering, key='roof\_covering\_type') # Aus A.2.2

if inputs['project\_details']['roof\_covering\_type'] == please\_select\_text: inputs['project\_details']['roof\_covering\_type'] = None # Speichere None, wenn Platzhalter gewählt

# Automatische Meldung zur Eignung/Aufpreis (Logik basierend auf Auswahl)

if inputs['project\_details']['roof\_covering\_type'] in ['Schiefer', 'Bitumen', 'Ethernit']: # Beispiele für potenziell höhere Kosten

st.warning(texts.get("roof\_covering\_warning", "❗️ Bei dieser Dachdeckungsart können höhere Montagekosten anfallen."))

# Füge weitere Bedingungen hinzu, z.B. für Trapezblech spezielle Montagesysteme

# elif inputs['project\_details']['roof\_covering\_type'] == 'Trapezblech':

# st.info(texts.get("roof\_covering\_trapez\_info", "ℹ️ Spezielle Trapezblechmontage notwendig."))

elif inputs['project\_details']['roof\_covering\_type'] not in [please\_select\_text, 'Keine Dachdeckungen gefunden', 'Fehler', None]: # Zeige Erfolg nur, wenn eine echte Option gewählt ist

st.success(texts.get("roof\_covering\_info", "✅ Dachbelegung voraussichtlich problemlos möglich."))

with col22:

inputs['project\_details']['free\_roof\_area\_sqm'] = st.number\_input(texts.get("free\_roof\_area\_label", "Freie Dachfläche (m²)"), min\_value=0.0, value=st.session\_state.get('free\_roof\_area\_sqm', 50.0), key='free\_roof\_area\_sqm') # Aus A.2.2

col23, col24 = st.columns(2)

with col23:

# Use session state for selectbox

# Ausrichtung hat feste Optionen, füge Platzhalter hinzu

options\_orientation = ['Süd', 'Südost', 'Ost', 'Südwest', 'West', 'Nordwest', 'Nord', 'Nordost']

options\_orientation\_with\_placeholder = [please\_select\_text] + options\_orientation

session\_state\_orientation = st.session\_state.get('roof\_orientation', please\_select\_text) # Default ist Platzhalter

try: initial\_index\_orientation = options\_orientation\_with\_placeholder.index(session\_state\_orientation) if session\_state\_orientation in options\_orientation\_with\_placeholder else 0

except ValueError: initial\_index\_orientation = 0 # Fallback

inputs['project\_details']['roof\_orientation'] = st.selectbox(texts.get("roof\_orientation\_label", "Dachausrichtung"), options=options\_orientation\_with\_placeholder, index=initial\_index\_orientation, key='roof\_orientation') # Aus A.2.2

if inputs['project\_details']['roof\_orientation'] == please\_select\_text: inputs['project\_details']['roof\_orientation'] = None # Speichere None, wenn Platzhalter gewählt

with col24:

inputs['project\_details']['roof\_inclination\_deg'] = st.number\_input(texts.get("roof\_inclination\_label", "Dachneigung (Grad)"), min\_value=0, max\_value=60, value=st.session\_state.get('roof\_inclination\_deg', 30), key='roof\_inclination\_deg') # Aus A.2.2

# Button für Gebäudehöhe über 7m (A.2.2) - Logik kommt später

inputs['project\_details']['building\_height\_gt\_7m'] = st.checkbox(texts.get("building\_height\_gt\_7m\_label", "Gebäudehöhe > 7 Meter (Gerüst erforderlich)"), value=st.session\_state.get('building\_height\_gt\_7m', False), key='building\_height\_gt\_7m') # Aus A.2.2

# 2.3 Zukünftiger Mehrverbrauch (A.2.3) - Implementierung kommt später

st.markdown("---") # Trennlinie

st.subheader(texts.get("future\_consumption\_header", "Zukünftiger Mehrverbrauch"))

st.info(texts.get("future\_consumption\_info", "Optionen für zukünftiges E-Auto oder Wärmepumpe werden hier später aktiviert (schaltet Sektion A.6 frei)."))

# 3. Auswahl der Technik (A.3) - Basis-Moduleingabe hier, weitere Details später

st.subheader(texts.get("technology\_selection\_header", "Auswahl der Technik"))

with st.expander(texts.get("technology\_selection\_header", "Auswahl der Technik"), expanded=True):

col25, col26 = st.columns(2)

with col25:

# Eingabefeld: Anzahl der PV Module (A.3)

inputs['project\_details']['module\_quantity'] = st.number\_input(texts.get("module\_quantity\_label", "Anzahl PV Module"), min\_value=0, value=st.session\_state.get('module\_quantity', 20), key='module\_quantity') # Aus A.3

with col26:

# Dropdown: Modell der PV Module (A.3) - dynamisch basierend auf DB

# Sicherstellen, dass MODULE\_LIST\_MODELS nicht leer ist, bevor als Optionen übergeben wird

# Laden der Liste ist bereits oben in der Funktion passiert.

options\_module = MODULE\_LIST\_MODELS # Liste wurde oben mit Placeholder ergänzt

# Use session state to remember selection

session\_state\_module = st.session\_state.get('selected\_module\_name', please\_select\_text) # Default ist Platzhalter

# Finde den Index des Werts im Session State in der Liste mit Placeholder (Index 0)

try:

initial\_index\_module = options\_module.index(session\_state\_module) if session\_state\_module in options\_module else 0

except ValueError: initial\_index\_module = 0 # Fallback

inputs['project\_details']['selected\_module\_name'] = st.selectbox(texts.get("module\_model\_label", "PV Modul Modell"), options=options\_module, index=initial\_index\_module, key='selected\_module\_name') # Aus A.3, speichere in Session State

# Optional: Anzeige der Leistung des gewählten Moduls (A.3)

# Benötigt get\_product\_by\_model\_name Funktion (nutzt die \_safe Version)

module\_details = None

# Finde Modul-Details NUR wenn ein sinnvoller Name gewählt wurde UND die Funktion verfügbar ist

selected\_module\_name = inputs['project\_details']['selected\_module\_name'] # Der Wert kommt jetzt aus dem Selectbox

if selected\_module\_name and selected\_module\_name not in [please\_select\_text, 'Keine Module gefunden', 'Keine Module verfügbar', 'Fehler'] and get\_product\_by\_model\_name\_safe is not Dummy\_get\_product\_by\_model\_name\_input:

try:

# Suche das Produkt anhand des ausgewählten Namens in der DB

module\_details = get\_product\_by\_model\_name\_safe(selected\_module\_name) # Nutze die reale oder Dummy Funktion

# Stelle sicher, dass es auch wirklich ein Modul ist (optional, falls Namen nicht eindeutig)

if module\_details and module\_details.get('category', '').lower() != 'modul':

# found something, but it's not a module - ignore details

module\_details = None

except Exception as e:

st.warning(f"Fehler beim Laden von Moduldetails für '{selected\_module\_name}': {e}")

traceback.print\_exc()

if module\_details:

inputs['project\_details']['selected\_module\_id'] = module\_details.get('id') # Speichere ID

inputs['project\_details']['selected\_module\_capacity\_w'] = module\_details.get('capacity\_w', 0.0) # Speichere Leistung

st.info(f"{texts.get('module\_capacity\_label', 'Leistung pro Modul (W)')}: {module\_details.get('capacity\_w', 0.0):.2f} W")

else:

inputs['project\_details']['selected\_module\_id'] = None

inputs['project\_details']['selected\_module\_capacity\_w'] = 0.0

# Zeige Info nur wenn kein Modul gefunden ODER Details nicht geladen werden konnten

if selected\_module\_name in [please\_select\_text, 'Keine Module gefunden', 'Keine Module verfügbar', 'Fehler']:

st.info(texts.get('module\_details\_not\_found', 'Modul Details nicht gefunden'))

elif selected\_module\_name is not None: # Wenn ein Name gewählt ist, aber Details fehlen (z.B. DB-Fehler)

st.warning(texts.get('module\_details\_not\_loaded', f'Details für Modul "{selected\_module\_name}" konnten nicht geladen werden.'))

# Sonst, wenn selected\_module\_name None ist (Liste leer), zeige auch nichts

# Ergebnis: Anlagengröße (kWp) (A.3)

# Wird hier berechnet zur Anzeige, im analysis Modul richtig genutzt

module\_quantity = inputs['project\_details'].get('module\_quantity', st.session\_state.get('module\_quantity', 0)) # Nutze session state als Fallback für Quantity

module\_capacity\_w = inputs['project\_details'].get('selected\_module\_capacity\_w', 0.0)

anlage\_kwp = (module\_quantity \* module\_capacity\_w) / 1000.0

st.info(f"{texts.get('anlage\_size\_label', 'Anlagengröße (kWp)')}: {anlage\_kwp:.2f} kWp")

inputs['project\_details']['anlage\_kwp'] = anlage\_kwp # Auch ins dict speichern

# --- Auswahlfelder für Wechselrichter und Speicher hinzufügen (A.3) ---

st.markdown("---")

st.subheader(texts.get("inverter\_selection\_header", "Wechselrichter Auswahl")) # Neuer Header

# Dropdown: Modell des Wechselrichters (A.3)

options\_inverter = INVERTER\_LIST\_MODELS # Liste wurde oben mit Placeholder ergänzt

session\_state\_inverter = st.session\_state.get('selected\_inverter\_name', please\_select\_text) # Default ist Platzhalter

try: initial\_index\_inverter = options\_inverter.index(session\_state\_inverter) if session\_state\_inverter in options\_inverter else 0

except ValueError: initial\_index\_inverter = 0 # Fallback

inputs['project\_details']['selected\_inverter\_name'] = st.selectbox(texts.get("inverter\_model\_label", "Wechselrichter Modell"), options=options\_inverter, index=initial\_index\_inverter, key='selected\_inverter\_name') # Aus A.3

# Optional: Anzeige der Leistung des gewählten Wechselrichters (A.3)

inverter\_details = None

selected\_inverter\_name = inputs['project\_details']['selected\_inverter\_name'] # Der Wert kommt jetzt aus dem Selectbox

if selected\_inverter\_name and selected\_inverter\_name not in [please\_select\_text, 'Keine Wechselrichter gefunden/geladen', 'Keine Wechselrichter verfügbar', 'Fehler'] and get\_product\_by\_model\_name\_safe is not Dummy\_get\_product\_by\_model\_name\_input:

try:

inverter\_details = get\_product\_by\_model\_name\_safe(selected\_inverter\_name)

if inverter\_details and inverter\_details.get('category', '').lower() not in ['wechselrichter', 'inverter']:

inverter\_details = None # Gefunden, aber keine WR Kategorie

except Exception as e:

st.warning(f"Fehler beim Laden von WR Details für '{selected\_inverter\_name}': {e}")

traceback.print\_exc()

if inverter\_details:

inputs['project\_details']['selected\_inverter\_id'] = inverter\_details.get('id')

inputs['project\_details']['selected\_inverter\_power\_kw'] = inverter\_details.get('power\_kw', 0.0)

st.info(f"{texts.get('inverter\_power\_label', 'Leistung Wechselrichter (kW)')}: {inverter\_details.get('power\_kw', 0.0):.2f} kW")

else:

inputs['project\_details']['selected\_inverter\_id'] = None

inputs['project\_details']['selected\_inverter\_power\_kw'] = 0.0

if selected\_inverter\_name in [please\_select\_text, 'Keine Wechselrichter gefunden/geladen', 'Keine Wechselrichter verfügbar', 'Fehler']:

st.info(texts.get('inverter\_details\_not\_found', 'Wechselrichter Details nicht gefunden'))

elif selected\_inverter\_name is not None:

st.warning(texts.get('inverter\_details\_not\_loaded', f'Details für WR "{selected\_inverter\_name}" konnten nicht geladen werden.'))

st.markdown("---")

st.subheader(texts.get("storage\_selection\_header", "Batteriespeicher Auswahl")) # Neuer Header

# Checkbox: Batteriespeicher einplanen (A.3)

inputs['project\_details']['include\_storage'] = st.checkbox(texts.get("include\_storage\_label", "Batteriespeicher einplanen"), value=st.session\_state.get('include\_storage', False), key='include\_storage') # Aus A.3

# Felder für Speicher nur anzeigen, wenn Checkbox aktiv

if inputs['project\_details']['include\_storage']:

col\_storage\_model, col\_storage\_capacity = st.columns(2)

with col\_storage\_model:

# Dropdown: Modell des Speichers (A.3)

options\_storage = STORAGE\_LIST\_MODELS # Liste wurde oben mit Placeholder ergänzt

session\_state\_storage\_model = st.session\_state.get('selected\_storage\_name', please\_select\_text) # Default ist Platzhalter

try: initial\_index\_storage = options\_storage.index(session\_state\_storage\_model) if session\_state\_storage\_model in options\_storage else 0

except ValueError: initial\_index\_storage = 0 # Fallback

inputs['project\_details']['selected\_storage\_name'] = st.selectbox(texts.get("storage\_model\_label", "Speicher Modell"), options=options\_storage, index=initial\_index\_storage, key='selected\_storage\_name') # Aus A.3

# Optional: Anzeige der Kapazität des gewählten Speichermodells (A.3)

storage\_details = None

selected\_storage\_name = inputs['project\_details']['selected\_storage\_name'] # Der Wert kommt jetzt aus dem Selectbox

if selected\_storage\_name and selected\_storage\_name not in [please\_select\_text, 'Keine Speicher gefunden/geladen', 'Keine Speicher verfügbar', 'Fehler'] and get\_product\_by\_model\_name\_safe is not Dummy\_get\_product\_by\_model\_name\_input:

try:

storage\_details = get\_product\_by\_model\_name\_safe(selected\_storage\_name)

if storage\_details and storage\_details.get('category', '').lower() not in ['batteriespeicher', 'storage']:

storage\_details = None # Gefunden, aber keine Speicher Kategorie

except Exception as e:

st.warning(f"Fehler beim Laden von Speicher Details für '{selected\_storage\_name}': {e}")

traceback.print\_exc()

if storage\_details:

inputs['project\_details']['selected\_storage\_id'] = storage\_details.get('id')

inputs['project\_details']['selected\_storage\_capacity\_kwh\_from\_model'] = storage\_details.get('capacity\_kwh', 0.0)

st.info(f"{texts.get('storage\_capacity\_model\_label', 'Kapazität Modell (kWh)')}: {storage\_details.get('capacity\_kwh', 0.0):.2f} kWh")

else:

inputs['project\_details']['selected\_storage\_id'] = None

inputs['project\_details']['selected\_storage\_capacity\_kwh\_from\_model'] = 0.0

if selected\_storage\_name in [please\_select\_text, 'Keine Speicher gefunden/geladen', 'Keine Speicher verfügbar', 'Fehler']:

st.info(texts.get('storage\_details\_not\_found', 'Speicher Details nicht gefunden'))

elif selected\_storage\_name is not None:

st.warning(texts.get('storage\_details\_not\_loaded', f'Details für Speicher "{selected\_storage\_name}" konnten nicht geladen werden.'))

with col\_storage\_capacity:

# Eingabefeld: Gewünschte Speicherkapazität (manuell) (A.3)

# Erlaubt Eingabe unabhängig vom Modell (z.B. mehrere Module des Modells)

inputs['project\_details']['selected\_storage\_capacity\_kwh'] = st.number\_input(texts.get("storage\_capacity\_manual\_label", "Gewünschte Gesamtkapazität (kWh)"), min\_value=0.0, value=st.session\_state.get('selected\_storage\_capacity\_kwh', 5.0), key='selected\_storage\_capacity\_kwh') # Aus A.3

else: # Wenn Batteriespeicher NICHT eingeplant

# Setze die Speicher-relevanten Eingaben auf None oder 0, wenn die Checkbox nicht aktiv ist

inputs['project\_details']['selected\_storage\_name'] = None

inputs['project\_details']['selected\_storage\_id'] = None

inputs['project\_details']['selected\_storage\_capacity\_kwh\_from\_model'] = 0.0

inputs['project\_details']['selected\_storage\_capacity\_kwh'] = 0.0 # Gewünschte Kapazität auf 0 setzen, wenn nicht eingeplant

# --- Checkbox und Auswahlfelder für weiteres Zubehör hinzufügen (A.3) ---

st.markdown("---")

st.subheader(texts.get("additional\_components\_header", "Zusätzliche Komponenten")) # Geänderter Header

# Checkbox: Zusätzliche Komponenten einplanen

inputs['project\_details']['include\_additional\_components'] = st.checkbox(texts.get("include\_additional\_components\_label", "Zusätzliche Komponenten einplanen"), value=st.session\_state.get('include\_additional\_components', False), key='include\_additional\_components')

# Felder für Zusatzkomponenten nur anzeigen, wenn Checkbox aktiv

if inputs['project\_details']['include\_additional\_components']:

st.markdown("---") # Trennlinie innerhalb des Expanders

st.subheader(texts.get("other\_components\_header", "Standard Zubehör (Wallbox, EMS, Optimierer)")) # Unter-Header

# Wallbox Auswahl

options\_wallbox = WALLBOX\_LIST\_MODELS # Liste wurde oben mit Placeholder ergänzt

session\_state\_wallbox = st.session\_state.get('selected\_wallbox\_name', please\_select\_text) # Default ist Platzhalter

try: initial\_index\_wallbox = options\_wallbox.index(session\_state\_wallbox) if session\_state\_wallbox in options\_wallbox else 0

except ValueError: initial\_index\_wallbox = 0 # Fallback

inputs['project\_details']['selected\_wallbox\_name'] = st.selectbox(texts.get("wallbox\_model\_label", "Wallbox Modell"), options=options\_wallbox, index=initial\_index\_wallbox, key='selected\_wallbox\_name')

if inputs['project\_details']['selected\_wallbox\_name'] == please\_select\_text: inputs['project\_details']['selected\_wallbox\_name'] = None # Speichere None

# TODO: Logik zum Laden/Anzeigen von Wallbox-Details (power\_kw etc.) und Speichern der ID hinzufügen

# EMS Auswahl

options\_ems = EMS\_LIST\_MODELS # Liste wurde oben mit Placeholder ergänzt

session\_state\_ems = st.session\_state.get('selected\_ems\_name', please\_select\_text) # Default ist Platzhalter

try: initial\_index\_ems = options\_ems.index(session\_state\_ems) if session\_state\_ems in options\_ems else 0

except ValueError: initial\_index\_ems = 0 # Fallback

inputs['project\_details']['selected\_ems\_name'] = st.selectbox(texts.get("ems\_model\_label", "Energiemanagementsystem Modell"), options=options\_ems, index=initial\_index\_ems, key='selected\_ems\_name')

if inputs['project\_details']['selected\_ems\_name'] == please\_select\_text: inputs['project\_details']['selected\_ems\_name'] = None # Speichere None

# TODO: Logik zum Laden/Anzeigen von EMS-Details hinzufügen

# Optimierer Auswahl

options\_optimizer = OPTIMIZER\_LIST\_MODELS # Liste wurde oben mit Placeholder ergänzt

session\_state\_optimizer = st.session\_state.get('selected\_optimizer\_name', please\_select\_text) # Default ist Platzhalter

try: initial\_index\_optimizer = options\_optimizer.index(session\_state\_optimizer) if session\_state\_optimizer in options\_optimizer else 0

except ValueError: initial\_index\_optimizer = 0 # Fallback

inputs['project\_details']['selected\_optimizer\_name'] = st.selectbox(texts.get("optimizer\_model\_label", "Leistungsoptimierer Modell"), options=options\_optimizer, index=initial\_index\_optimizer, key='selected\_optimizer\_name')

if inputs['project\_details']['selected\_optimizer\_name'] == please\_select\_text: inputs['project\_details']['selected\_optimizer\_name'] = None # Speichere None

# TODO: Logik zum Laden/Anzeigen von Optimierer-Details hinzufügen

st.markdown("---") # Trennlinie

st.subheader(texts.get("other\_additional\_components\_header", "Sonstige Zusatzkomponenten")) # Neuer Unter-Header

# Carport Auswahl

options\_carport = CARPORT\_LIST\_MODELS # Liste wurde oben mit Placeholder ergänzt

session\_state\_carport = st.session\_state.get('selected\_carport\_name', please\_select\_text) # Default ist Platzhalter

try: initial\_index\_carport = options\_carport.index(session\_state\_carport) if session\_state\_carport in options\_carport else 0

except ValueError: initial\_index\_carport = 0 # Fallback

inputs['project\_details']['selected\_carport\_name'] = st.selectbox(texts.get("carport\_model\_label", "Carport Modell"), options=options\_carport, index=initial\_index\_carport, key='selected\_carport\_name')

if inputs['project\_details']['selected\_carport\_name'] == please\_select\_text: inputs['project\_details']['selected\_carport\_name'] = None # Speichere None

# TODO: Logik zum Laden/Anzeigen von Carport-Details hinzufügen

# Notstrom Auswahl

options\_notstrom = NOTSTROM\_LIST\_MODELS # Liste wurde oben mit Placeholder ergänzt

session\_state\_notstrom = st.session\_state.get('selected\_notstrom\_name', please\_select\_text) # Default ist Platzhalter

try: initial\_index\_notstrom = options\_notstrom.index(session\_state\_notstrom) if session\_state\_notstrom in options\_notstrom else 0

except ValueError: initial\_index\_notstrom = 0 # Fallback

inputs['project\_details']['selected\_notstrom\_name'] = st.selectbox(texts.get("notstrom\_model\_label", "Notstromversorgung Modell"), options=options\_notstrom, index=initial\_index\_notstrom, key='selected\_notstrom\_name')

if inputs['project\_details']['selected\_notstrom\_name'] == please\_select\_text: inputs['project\_details']['selected\_notstrom\_name'] = None # Speichere None

# TODO: Logik zum Laden/Anzeigen von Notstrom-Details hinzufügen

# Tierabwehr Auswahl

options\_tierabwehr = TIERABWEHR\_LIST\_MODELS # Liste wurde oben mit Placeholder ergänzt

session\_state\_tierabwehr = st.session\_state.get('selected\_tierabwehr\_name', please\_select\_text) # Default ist Platzhalter

try: initial\_index\_tierabwehr = options\_tierabwehr.index(session\_state\_tierabwehr) if session\_state\_tierabwehr in options\_tierabwehr else 0

except ValueError: initial\_index\_tierabwehr = 0 # Fallback

inputs['project\_details']['selected\_tierabwehr\_name'] = st.selectbox(texts.get("tierabwehr\_model\_label", "Tierabwehrschutz Modell"), options=options\_tierabwehr, index=initial\_index\_tierabwehr, key='selected\_tierabwehr\_name')

if inputs['project\_details']['selected\_tierabwehr\_name'] == please\_select\_text: inputs['project\_details']['selected\_tierabwehr\_name'] = None # Speichere None

# TODO: Logik zum Laden/Anzeigen von Tierabwehr-Details hinzufügen

else: # Wenn zusätzliche Komponenten NICHT eingeplant

# Setze die zusätzlichen Eingaben auf None oder 0, wenn die Checkbox nicht aktiv ist

inputs['project\_details']['selected\_wallbox\_name'] = None

# TODO: Setze auch die ID und spezifischen Werte (power, capacity) auf None/0

inputs['project\_details']['selected\_ems\_name'] = None

# TODO: Setze auch die ID und spezifischen Werte (power, capacity) auf None/0

inputs['project\_details']['selected\_optimizer\_name'] = None

# TODO: Setze auch die ID und spezifischen Werte (power, capacity) auf None/0

inputs['project\_details']['selected\_carport\_name'] = None

# TODO: Setze auch die ID und spezifischen Werte (power, capacity) auf None/0

inputs['project\_details']['selected\_notstrom\_name'] = None

# TODO: Setze auch die ID und spezifischen Werte (power, capacity) auf None/0

inputs['project\_details']['selected\_tierabwehr\_name'] = None

# TODO: Setze auch die ID und spezifischen Werte (power, capacity) auf None/0

# --- Rückgabe der gesammelten Eingaben ---

# Gib das Dictionary mit allen gesammelten Werten zurück

# Die Haupt-GUI (gui.py) kann diese Daten dann weiterverarbeiten oder speichern

return inputs

# --- Optional: Test beim direkten Ausführen (nur zum Debuggen) ---

if \_\_name\_\_ == "\_\_main\_\_":

# Um data\_input.py alleine zu testen, muss database.py importierbar sein

# und die init\_db() Funktion aus database.py muss gelaufen sein, um die DB-Datei zu erstellen.

# Dummy\_load\_translations wird hier verwendet, da render\_data\_input texts benötigt.

class DummyTextsForInputTest:

def get(self, key, default): return default

st.set\_page\_config(layout="wide")

st.title("Data Input Modul Test")

# Lade Dummy-Texte für den Testlauf

dummy\_texts = DummyTextsForInputTest() # Nutze eine Dummy Klasse, die .get() hat

# Rufe die Render Funktion auf

collected\_data = render\_data\_input(dummy\_texts)

st.write("---")

st.subheader("Gesammelte Daten (zur Kontrolle):")

st.write(collected\_data)

**database.py:**

# database.py

# Modul für Datenbankverbindungen und -initialisierung

import sqlite3

import os

import traceback # Import traceback for detailed error logging

import json # Importiere json für das Speichern/Laden komplexer Werte in admin\_settings

from typing import List, Dict, Any # Importiere Typing-Hints

from datetime import datetime # Für Zeitstempel im Testblock

# Definiere den Pfad zur Datenbankdatei

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

DB\_PATH = os.path.join(BASE\_DIR, 'data', 'app\_data.db')

# Stelle sicher, dass der data-Ordner existiert

DATA\_DIR = os.path.join(BASE\_DIR, 'data')

if not os.path.exists(DATA\_DIR):

os.makedirs(DATA\_DIR)

print(f"Datenverzeichnis '{DATA\_DIR}' erstellt.") # Debugging

def get\_db\_connection():

"""Stellt eine Verbindung zur SQLite-Datenbank her."""

try:

conn = sqlite3.connect(DB\_PATH)

# Setze row\_factory auf sqlite3.Row, um Spaltennamen als Dictionary-Schlüssel verwenden zu können

conn.row\_factory = sqlite3.Row

# print(f"Datenbankverbindung erfolgreich zu {DB\_PATH}") # Debugging

return conn

except Exception as e:

print(f"Fehler beim Verbinden zur Datenbank {DB\_PATH}: {e}")

traceback.print\_exc() # Detaillierten Traceback ausgeben

return None

def init\_db():

"""Initialisiert die Datenbank: Erstellt Tabellen und fügt fehlende Spalten hinzu."""

print("init\_db() aufgerufen - Starte Datenbankinitialisierung und Schema-Check...") # Debugging

conn = get\_db\_connection()

if conn is None:

print("Datenbankinitialisierung fehlgeschlagen: Keine Verbindung.")

return

cursor = conn.cursor()

try:

# Tabelle für Admin-Einstellungen

cursor.execute("""

CREATE TABLE IF NOT EXISTS admin\_settings (

key TEXT PRIMARY KEY,

value TEXT

)

""")

# Tabelle für Produkte (Module, Speicher, Wechselrichter, etc.)

cursor.execute("""

CREATE TABLE IF NOT EXISTS products (

id INTEGER PRIMARY KEY AUTOINCREMENT,

category TEXT NOT NULL, -- e.g., 'module', 'storage', 'inverter', 'wallbox'

brand TEXT,

model\_name TEXT UNIQUE NOT NULL, -- Modellname sollte eindeutig sein

capacity\_w REAL, -- z.B. 450 (Wp) für Module, Leistung für WR/Wallbox in W (kann später in kW umgerechnet werden)

capacity\_kwh REAL, -- z.B. 10 (kWh) für Speicher

power\_kw REAL, -- Leistung in kW (Du hattest power\_kw, behalten wir bei, muss aber konsistent genutzt werden)

price\_euro REAL DEFAULT 0.0,

warranty\_years REAL,

length\_m REAL, -- Für Module

width\_m REAL, -- Für Module

weight\_kg REAL,

efficiency\_percent REAL,-- Für Module, WR

origin\_country TEXT,

description TEXT,

pros TEXT, -- Gespeichert als JSON-String []

cons TEXT, -- Gespeichert als JSON-String []

rating REAL, -- z.B. 1-5 Sterne

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

)

""")

# Index für schnelles Nachschlagen nach Modellname und Kategorie

cursor.execute("CREATE INDEX IF NOT EXISTS idx\_products\_model\_name ON products (model\_name)")

cursor.execute("CREATE INDEX IF NOT EXISTS idx\_products\_category ON products (category)")

# Tabelle für Kunden

cursor.execute("""

CREATE TABLE IF NOT EXISTS customers (

id INTEGER PRIMARY KEY AUTOINCREMENT,

type TEXT, -- z.B. 'Privat', 'Gewerblich'

salutation TEXT, -- z.B. 'Herr', 'Frau'

title TEXT, -- z.B. 'Dr.'

first\_name TEXT,

last\_name TEXT,

address TEXT, -- Straße + Hausnummer (oder nur Straße)

house\_number TEXT, -- Separate Hausnummer falls geparst

zip\_code TEXT,

city TEXT,

state TEXT, -- Bundesland

region TEXT,

num\_persons INTEGER,

email TEXT,

phone\_landline TEXT,

phone\_mobile TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

)

""")

# Index für schnelles Nachschlagen nach Nachname, Firmenname, PLZ

# Annahme: company\_name wird im Type='Gewerblich' Fall im last\_name oder einem separaten Feld gespeichert

cursor.execute("CREATE INDEX IF NOT EXISTS idx\_customers\_last\_name ON customers (last\_name)")

cursor.execute("CREATE INDEX IF NOT EXISTS idx\_customers\_zip\_code ON customers (zip\_code)")

# cursor.execute("CREATE INDEX IF NOT EXISTS idx\_customers\_company\_name ON customers (company\_name)") # Falls separate Spalte für Firmenname

# Tabelle für Projekte

cursor.execute("""

CREATE TABLE IF NOT EXISTS projects (

id INTEGER PRIMARY KEY AUTOINCREMENT,

customer\_id INTEGER,

project\_name TEXT,

project\_status TEXT, -- HINZUGEFÜGT: Definition der Spalte HIER

-- Verbrauchsdaten

consumption\_household\_kwh\_yr REAL,

consumption\_heating\_kwh\_yr REAL,

costs\_household\_euro\_mo REAL,

costs\_heating\_euro\_mo REAL,

-- Gebäudedaten

roof\_type TEXT,

build\_year INTEGER,

roof\_covering\_type TEXT,

free\_roof\_area\_sqm REAL,

roof\_orientation TEXT,

roof\_inclination\_deg REAL,

building\_height\_gt\_7m BOOLEAN, -- 1 für Ja, 0 für Nein

notes TEXT, -- Allgemeine Notizen zum Projekt

project\_details\_json TEXT, -- Speichert alle Projektdetails als JSON (Alternative oder Ergänzung zu Spalten)

-- Spalten für ausgewählte Produkte und Features

anlage\_kwp REAL,

module\_quantity INTEGER,

selected\_module\_id INTEGER, -- FK zu products.id

selected\_inverter\_id INTEGER, -- FK zu products.id

include\_storage BOOLEAN,

selected\_storage\_id INTEGER, -- FK zu products.id

selected\_storage\_capacity\_kwh REAL, -- Eingegebene Speicherkapazität

future\_ev BOOLEAN,

future\_hp BOOLEAN,

include\_additional\_components BOOLEAN,

selected\_wallbox\_id INTEGER, -- FK zu products.id

selected\_ems\_id INTEGER, -- FK zu products.id

selected\_optimizer\_id INTEGER, -- FK zu products.id

selected\_carport\_id INTEGER, -- FK zu products.id

selected\_notstrom\_id INTEGER, -- FK zu products.id

selected\_tierabwehr\_id INTEGER, -- FK zu products.id

-- Weitere zukünftige Spalten hier definieren

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (customer\_id) REFERENCES customers (id) ON DELETE CASCADE

-- FKs für ausgewählte Produkte können hier hinzugefügt werden (erfordert ALTER TABLE oder separate Migration)

)

""")

# Index für schnelles Nachschlagen nach Kundennummer und Projektstatus

cursor.execute("CREATE INDEX IF NOT EXISTS idx\_projects\_customer\_id ON projects (customer\_id)")

# Dieser Index wird jetzt NACH der CREATE TABLE Anweisung für project\_status erstellt

# Stellen Sie sicher, dass die Spalte existiert, BEVOR der Index erstellt wird.

# Die Migration unten fügt die Spalte hinzu, falls nötig. Der Index kommt danach.

cursor.execute("CREATE INDEX IF NOT EXISTS idx\_projects\_status ON projects (project\_status)")

# --- Schema-Migrationen: Spalten hinzufügen, falls sie fehlen ---

print("Starte Schema-Migrations-Check...") # Debugging

# Hole die aktuellen Spaltennamen der projects Tabelle

cursor.execute("PRAGMA table\_info(projects)")

columns = [col[1] for col in cursor.fetchall()] # Hole die Spaltennamen

# Liste der Spalten, die hinzugefügt werden sollen, falls sie fehlen

columns\_to\_add = {

'project\_status': 'TEXT', # <<< DIESE SPALTE WIRD DURCH MIGRATION HINZUGEFÜGT FALLS FEHLEND

'anlage\_kwp': 'REAL',

'selected\_module\_id': 'INTEGER', # Füge FK Spalte hinzu

'selected\_inverter\_id': 'INTEGER', # Füge FK Spalte hinzu

'include\_storage': 'BOOLEAN', # Füge Spalte hinzu

'selected\_storage\_id': 'INTEGER', # Füge FK Spalte hinzu

'selected\_storage\_capacity\_kwh': 'REAL', # Füge Spalte hinzu

'future\_ev': 'BOOLEAN', # Füge Spalte hinzu

'future\_hp': 'BOOLEAN', # Füge Spalte hinzu

'include\_additional\_components': 'BOOLEAN', # Füge Spalte hinzu

'selected\_wallbox\_id': 'INTEGER', # Füge FK Spalte hinzu

'selected\_ems\_id': 'INTEGER', # Füge FK Spalte hinzu

'selected\_optimizer\_id': 'INTEGER', # Füge FK Spalte hinzu

'selected\_carport\_id': 'INTEGER', # Füge FK Spalte hinzu

'selected\_notstrom\_id': 'INTEGER', # Füge FK Spalte hinzu

'selected\_tierabwehr\_id': 'INTEGER' # Füge FK Spalte hinzu

# Füge hier alle zukünftigen Spalten hinzu

}

for col\_name, col\_type in columns\_to\_add.items():

if col\_name not in columns:

try:

print(f"Versuche Migration: Füge Spalte '{col\_name}' ({col\_type}) zur Tabelle 'projects' hinzu...") # Debugging

cursor.execute(f"ALTER TABLE projects ADD COLUMN {col\_name} {col\_type}")

print(f"Migration erfolgreich: Spalte '{col\_name}' hinzugefügt.") # Debugging

# Füge auch die fehlenden FOREIGN KEYs hinzu, aber das ist komplexer und wird später gemacht,

# wenn wir eine robustere Migrationsstrategie haben. Für jetzt nur die Spalten.

except Exception as e:

print(f"Fehler bei Migration (Spalte {col\_name}): {e}")

traceback.print\_exc() # Detaillierten Traceback ausgeben

else:

# print(f"Spalte '{col\_name}' existiert bereits.") # Zu gesprächig

pass

# Füge Standard-Admin-Einstellungen hinzu, falls sie fehlen

initial\_settings = {

'salutation\_options': ['Herr', 'Frau', 'Familie'],

'title\_options': ['Dr.', 'Prof.', 'Mag.', 'Ing.', None], # None für 'Kein Titel'

'bundesland\_options': ['Baden-Württemberg', 'Bayern', 'Berlin', 'Brandenburg', 'Bremen', 'Hamburg', 'Hessen', 'Mecklenburg-Vorpommern', 'Niedersachsen', 'Nordrhein-Westfalen', 'Rheinland-Pfalz', 'Saarland', 'Sachsen', 'Sachsen-Anhalt', 'Schleswig-Holstein', 'Thüringen'],

'region\_options': ['Nord', 'Ost', 'Süd', 'West', 'Mitte', 'Nord-West', 'Süd-West', 'Nord-Ost', 'Süd-Ost'], # Beispiel Regionen, erweiterbar

'dachart\_options': ['Satteldach', 'Satteldach mit Gaube', 'Pultdach', 'Flachdach', 'Walmdach', 'Krüppelwalmdach', 'Zeltdach', 'Sonderform', 'Keine Angabe'],

'dachdeckung\_options': ['Frankfurter Pfanne (Beton)', 'Tonziegel (Biberschwanz, Flachziegel etc.)', 'Trapezblech', 'Wellbitumen / Dachpappe', 'Schiefer', 'Eternit / Faserzement', 'Blechdach (Stehfalz)', 'Schindeln', 'Sonstiges', 'Keine Angabe'],

# Beispiele für Standardkosten (Werte sind Platzhalter)

'standard\_costs\_geruest\_pauschal\_gt\_7m': 500.0, # Pauschale, wenn >7m

'standard\_costs\_geruest\_pro\_modul\_gt\_7m': 10.0, # Kosten pro Modul, wenn >7m

'standard\_costs\_misc\_pauschal': 200.0, # Sonstige Pauschalkosten

'standard\_costs\_misc\_pro\_kwp': 50.0, # Sonstige Kosten pro kWp

# Beispiel Einspeisevergütung (Werte sind Platzhalter, aktuell Januar 2025)

# Siehe https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Energie/Unternehmen\_Institutionen/ErneuerbareEnergien/Photovoltaik/LeistungsgrenzenZeitreihen/AnzPV2024/EEG2023\_AnzPV\_Jan2025.pdf?\_\_blob=publicationFile&v=2

'feed\_in\_tariffs': {

'<10kWp\_Teil': 0.081, # ct/kWh / 100

'>10kWp<=40kWp\_Teil': 0.070,

'>40kWp<=100kWp\_Teil': 0.058,

'<10kWp\_Voll': 0.129,

'>10kWp<=40kWp\_Voll': 0.108,

'>40kWp<=100kWp\_Voll': 0.086

},

# Weitere finanzielle Standardwerte (Platzhalter)

'default\_vat\_rate\_percent': 19.0, # 19% Standard-MwSt

'pv\_0\_vat\_threshold\_kwp': 30.0, # Schwelle für 0% MwSt PV (<= 30 kWp)

'default\_strompreis\_pro\_kwh': 0.35, # Standard Strompreis für Vergleiche, falls nicht eingegeben

'default\_strompreissteigerung\_prozent\_pa': 3.0, # Jährliche Strompreissteigerung %

'default\_inflation\_rate\_prozent\_pa': 2.0, # Jährliche Inflationsrate %

'default\_maintenance\_cost\_euro\_pa': 100.0, # Jährliche Wartungskosten Pauschal

'default\_maintenance\_cost\_pro\_kwp\_pa': 5.0 # Jährliche Wartungskosten pro kWp

}

for key, value in initial\_settings.items():

cursor.execute("SELECT value FROM admin\_settings WHERE key = ?", (key,))

if cursor.fetchone() is None:

# Speichere Listen und Dictionaries als JSON

value\_to\_store = json.dumps(value) if isinstance(value, (dict, list, bool, int, float, str)) else str(value)

cursor.execute("INSERT INTO admin\_settings (key, value) VALUES (?, ?)",

(key, value\_to\_store))

print(f"Standardeinstellung '{key}' hinzugefügt.") # Debugging

# else: # Debugging: Zeigt, dass Einstellung schon da ist

# print(f"Standardeinstellung '{key}' existiert bereits.")

conn.commit()

print("Datenbankinitialisierung und Schema-Check/Migration abgeschlossen.") # Debugging

except Exception as e:

print(f"Fehler während der Datenbankinitialisierung: {e}")

traceback.print\_exc() # Detaillierten Traceback ausgeben

conn.rollback() # Änderungen rückgängig machen

finally:

if conn: conn.close() # Verbindung schließen!

# --- Funktionen zum Speichern und Laden von Admin-Einstellungen ---

# Diese Funktionen müssen außerhalb von init\_db() definiert sein, um importierbar zu sein

def save\_admin\_setting(key: str, value: Any):

"""Speichert eine einzelne Einstellung im Adminbereich."""

conn = get\_db\_connection()

if conn is None:

print("save\_admin\_setting: Keine Datenbankverbindung verfügbar.")

return

cursor = conn.cursor()

try:

# Werte, die keine einfachen Typen sind, als JSON speichern

value\_to\_store = json.dumps(value) if isinstance(value, (dict, list, bool, int, float, str)) else str(value)

cursor.execute("INSERT OR REPLACE INTO admin\_settings (key, value) VALUES (?, ?)",

(key, value\_to\_store))

conn.commit()

# print(f"Admin Setting '{key}' gespeichert.") # Debugging

except Exception as e:

print(f"Fehler beim Speichern von Admin Setting '{key}': {e}")

traceback.print\_exc()

conn.rollback()

finally:

if conn: conn.close()

def load\_admin\_setting(key: str, default: Any = None) -> Any:

"""Lädt eine Einstellung aus dem Adminbereich."""

conn = get\_db\_connection()

if conn is None:

# print("load\_admin\_setting: Keine Datenbankverbindung verfügbar.") # Zu gesprächig

return default # Gib Default zurück, wenn keine Verbindung möglich

cursor = conn.cursor()

try:

cursor.execute("SELECT value FROM admin\_settings WHERE key = ?", (key,))

row = cursor.fetchone()

if row:

value\_str = row['value']

# Versuche, JSON zu parsen, ansonsten als String zurückgeben

try:

# Laden als JSON versucht, handhabt Listen, Dictionaries, Boolesche Werte, Zahlen

return json.loads(value\_str)

except json.JSONDecodeError:

# Wenn JSON Decode fehlschlägt, ist es ein einfacher String oder ein anderer nicht-JSON Wert

return value\_str

return default # Gib den Standardwert zurück, wenn der Schlüssel nicht gefunden wird

except Exception as e:

print(f"Fehler beim Laden von Admin Setting '{key}': {e}")

traceback.print\_exc()

return default # Gib Default zurück bei Fehler

finally:

if conn: conn.close()

def load\_all\_admin\_settings() -> Dict[str, Any]:

"""Lädt alle Einstellungen aus dem Adminbereich."""

conn = get\_db\_connection()

if conn is None:

print("load\_all\_admin\_settings: Keine Datenbankverbindung verfügbar.")

return {}

cursor = conn.cursor()

settings = {}

try:

cursor.execute("SELECT key, value FROM admin\_settings")

rows = cursor.fetchall()

for row in rows:

key = row['key']

value\_str = row['value']

try:

settings[key] = json.loads(value\_str)

except json.JSONDecodeError:

settings[key] = value\_str

return settings

except Exception as e:

print(f"Fehler beim Laden aller Admin Settings: {e}")

traceback.print\_exc()

return {}

finally:

if conn: conn.close()

def save\_all\_admin\_settings(settings: Dict[str, Any]):

"""Speichert ein Dictionary von Einstellungen im Adminbereich."""

conn = get\_db\_connection()

if conn is None:

print("save\_all\_admin\_settings: Keine Datenbankverbindung verfügbar.")

return

cursor = conn.cursor()

try:

# Lösche alle alten Einstellungen und füge die neuen ein

# Optional: Lösche nur Einstellungen, die im übergebenen Dictionary sind, um andere nicht zu beeinflussen

# For now, simple delete all:

cursor.execute("DELETE FROM admin\_settings")

for key, value in settings.items():

value\_to\_store = json.dumps(value) if isinstance(value, (dict, list, bool, int, float, str)) else str(value)

cursor.execute("INSERT OR REPLACE INTO admin\_settings (key, value) VALUES (?, ?)",

(key, value\_to\_store))

conn.commit()

# print("Alle Admin Settings gespeichert.") # Debugging

except Exception as e:

print(f"Fehler beim Speichern aller Admin Settings: {e}")

traceback.print\_exc()

conn.rollback()

finally:

if conn: conn.close()

# --- Initialisiere die Datenbank beim Laden des Moduls ---

# Dieser Block wird nur ausgeführt, wenn die Datei direkt gestartet wird ODER

# wenn sie importiert wird und die Datei 'app\_data.db' noch nicht existiert.

# Die init\_db() Funktion kümmert sich jetzt um Tabellen UND Spalten.

# Ein Aufruf von init\_db() beim Laden des Moduls ist riskant, wenn andere Module

# zur gleichen Zeit importiert werden und DB-Zugriff benötigen.

# Es ist oft besser, init\_db() explizit einmal am Anfang der Hauptanwendung (z.B. in gui.py oder einem separaten start\_app.py) aufzurufen.

# Für jetzt lassen wir den Check beim Laden, was in einer einfachen Streamlit App oft funktioniert.

print("Database module loaded. Calling init\_db()...") # Debugging

try:

init\_db()

print("init\_db() finished.") # Debugging

except Exception as e:

print(f"FEHLER während init\_db() beim Modul-Laden: {e}")

traceback.print\_exc()

# Optional: Test beim direkten Ausführen (nur zum Debuggen)

if \_\_name\_\_ == "\_\_main\_\_":

print("--- Testlauf für database.py ---")

print(f"Datenbankpfad: {DB\_PATH}")

# init\_db() # Wird oben bereits ausgeführt, wenn Datei nicht existiert oder immer beim Laden.

# Man könnte hier init\_db() explizit aufrufen, um sicherzustellen, dass es beim Test läuft.

# init\_db() # Rufe es explizit auf, um sicherzustellen, dass Tabellen/Spalten da sind.

conn = get\_db\_connection()

if conn:

print("Testverbindung erfolgreich hergestellt.")

conn.close()

else:

print("Testverbindung fehlgeschlagen.")

print("\nTeste save/load\_admin\_setting...")

test\_key = "test\_setting\_single"

test\_value = "simple\_value"

save\_admin\_setting(test\_key, test\_value)

loaded\_value = load\_admin\_setting(test\_key, default="Not Found")

print(f"Gespeicherter Wert für '{test\_key}': {test\_value}")

print(f"Geladener Wert für '{test\_key}': {loaded\_value}")

print(f"Wert korrekt geladen? {loaded\_value == test\_value}")

test\_key\_json = "test\_setting\_json"

test\_value\_json = {"example\_list": [1, 2, 3], "example\_bool": True, "example\_string": "hello", "nested\_dict": {"a": 1, "b": 2}}

save\_admin\_setting(test\_key\_json, test\_value\_json)

loaded\_value\_json = load\_admin\_setting(test\_key\_json, default={})

print(f"\nGespeicherter JSON Wert für '{test\_key\_json}': {test\_value\_json}")

print(f"Geladener JSON Wert für '{test\_key\_json}': {loaded\_value\_json}")

print(f"Wert korrekt geladen? {loaded\_value\_json == test\_value\_json}")

print("\nTeste load\_all\_admin\_settings...")

# Füge ein paar mehr Settings hinzu, um load\_all zu testen

save\_admin\_setting("another\_setting", 12345)

save\_admin\_setting("a\_boolean\_setting", True)

all\_settings = load\_all\_admin\_settings()

print("Alle geladenen Admin Settings:", all\_settings)

print("\n--- Testlauf beendet ---")

**gui.py:**

# gui.py

# Hauptdatei für die Streamlit Anwendung "Ömer´s DingsDa für Solar"

import streamlit as st

import pandas as pd

import json

from datetime import datetime

import io

import math

import re

import sqlite3 # SQLite wird oft direkt in Modulen verwendet

import traceback # Importiert, um Fehler zu loggen

import os

from typing import Dict, Any, List, Optional

# Importiere inspect, um Funktionssignaturen zu prüfen, optional aber nützlich

# import inspect # Nicht nötig mit der vereinfachten Fehlerbehandlung

# --- Globale Variablen für Import-Fehler ---

# Diese Liste sammelt alle Fehler beim Laden externer Module.

import\_errors: List[str] = []

# --- Definiere Dummy-Funktionen als Fallback ---

# Diese Funktionen simulieren das Verhalten der echten Funktionen, falls der Import fehlschlägt.

# Sie sind hier auf der obersten Ebene definiert und werden später den Modulvariablen zugewiesen.

def Dummy\_render\_module(\*args, \*\*kwargs):

"""Generische Dummy-Renderfunktion für UI-Module."""

module\_name = kwargs.get('module\_name', 'Unbekanntes Modul')

st.warning(f"{module\_name} Modul nicht geladen oder fehlerhaft. Ansicht nicht verfügbar.")

def Dummy\_perform\_calculations(project\_data: Dict[str, Any]) -> Dict[str, Any]:

"""Dummy-Funktion, wenn calculations.py nicht geladen werden konnte."""

return {"calculation\_errors": ["Berechnungsmodul nicht geladen oder Fehler beim Laden."]}

def Dummy\_get\_db\_connection():

"""Dummy-Funktion für Datenbankverbindung."""

return None

def Dummy\_save\_admin\_setting(key: str, value: Any):

"""Dummy-Funktion zum Speichern von Admin-Einstellungen."""

pass # Dummy tut nichts

def Dummy\_load\_admin\_setting(key: str, default: Any = None) -> Any:

"""Dummy-Funktion zum Laden von Admin-Einstellungen mit Fallback-Werten."""

# Stellt sicher, dass zumindest Standardlisten zurückgegeben werden, wenn Admin-Settings nicht geladen werden können

if key == 'salutation\_options': return ['Herr', 'Frau', 'Familie', texts\_fallback.get("none\_option", "Kein Titel")] # Nutze Fallback Texte

if key == 'title\_options': return ['Dr.', 'Prof.', 'Mag.', 'Ing.', texts\_fallback.get("none\_option", "Kein Titel")]

if key == 'bundesland\_options': return [texts\_fallback.get("please\_select\_option", "--- Bitte wählen ---"), texts\_fallback.get("db\_connection\_unavailable", "DB Fehler")]

if key == 'region\_options': return [texts\_fallback.get("please\_select\_option", "--- Bitte wählen ---"), texts\_fallback.get("db\_connection\_unavailable", "DB Fehler")]

if key == 'dachart\_options': return [texts\_fallback.get("please\_select\_option", "--- Bitte wählen ---"), texts\_fallback.get("db\_connection\_unavailable", "DB Fehler")]

if key == 'dachdeckung\_options': return [texts\_fallback.get("please\_select\_option", "--- Bitte wählen ---"), texts\_fallback.get("db\_connection\_unavailable", "DB Fehler")]

# Dummy tarife für tests, wenn admin settings nicht gehen (Werte von 2024 beibehalten)

if key == 'feed\_in\_tariffs': return {'<10kWp\_Teil': 0.081, '>10kWp<=40kWp\_Teil': 0.070, '>40kWp<=100kWp\_Teil': 0.058, '<10kWp\_Voll': 0.129, '>10kWp<=40kWp\_Voll': 0.108, '>40kWp<=100kWp\_Voll': 0.086}

if key == 'co2\_factor\_grid\_g\_per\_kwh': return 388.0

if key == 'standard\_costs\_geruest\_pauschal\_gt\_7m': return 500.0

if key == 'standard\_misc\_cost\_pauschal': return 200.0

if key == 'default\_vat\_rate\_percent': return 19.0

if key == 'pv\_0\_vat\_threshold\_kwp': return 30.0

return default # Fallback für andere Schlüssel

def Dummy\_load\_translations(lang\_code: str) -> dict:

"""Dummy-Funktion für Lokalisierung mit festen Fallback-Texten."""

return texts\_fallback # Gib immer die festen Fallback-Texte zurück

def Dummy\_save\_customer(conn: Any, customer\_data: Dict[str, Any]) -> int:

"""Dummy-Funktion zum Speichern eines Kunden."""

return -1 # Dummy return -1 für Fehler

def Dummy\_save\_project(conn: Any, project\_data: Dict[str, Any]) -> int:

"""Dummy-Funktion zum Speichern eines Projekts."""

return -1 # Dummy return -1 für Fehler

# KORRIGIERT: Dummy-Funktion für render\_analysis mit korrekter Signatur

def Dummy\_render\_analysis(\*args, \*\*kwargs):

"""Dummy-Renderfunktion für Analysis Modul, die Argumente akzeptiert."""

module\_name = texts\_fallback.get("menu\_item\_analysis", "Analyse") # Annahme Text Schlüssel

st.warning(f"{module\_name} Modul nicht geladen oder fehlerhaft. Ansicht nicht verfügbar.")

# Optional: Anzeige der übergebenen Ergebnisse im Dummy, falls vorhanden

results = kwargs.get('results') or (args[1] if len(args) > 1 else None) # Versuche results zu finden (Positional arg index 1)

if results and results.get('calculation\_errors'):

st.error(texts\_fallback.get('calculation\_errors\_header', "Es traten Fehler während der Berechnungen auf:"))

for err in results['calculation\_errors']:

st.error(err)

elif results:

st.info("Dummy-Analyse: Ergebnisse wurden berechnet, können aber nicht detailliert angezeigt werden.")

# --- Fallback-Texte für Lokalisierung (werden verwendet, wenn locales.py nicht geladen werden kann) ---

# KORRIGIERT: Alle neuen Schlüssel aus admin\_panel.py und analysis.py hinzugefügt

texts\_fallback = {

"app\_title": "Ömer´s DingsDa für Solar (DEVELOPMENT)",

"menu\_item\_input": "Eingabe (A)",

"menu\_item\_quick\_calc": "Schnellberechnung (B)",

"menu\_item\_crm": "CRM (C)",

"menu\_item\_info\_platform": "Info-Plattform (D)",

"menu\_item\_options": "Optionen (E)",

"menu\_item\_admin": "Admin (F)",

"import\_errors\_title": "⚠ Modul-Ladefehler:",

"save\_project\_button": "Projekt speichern",

"save\_success": "Projekt erfolgreich gespeichert!",

"save\_failure": "Fehler beim Speichern des Projekts.",

"save\_project\_button\_disabled\_tooltip": "Speichern nicht möglich aufgrund fehlender Datenbank- oder CRM-Funktionen.",

"db\_connection\_unavailable": "Datenbankverbindung nicht verfügbar.", # Text für data\_input/crm DB Fehler

"please\_select\_option": "--- Bitte wählen ---", # Für Dropdowns in data\_input

"none\_option": "Kein Titel", # Für Titel Dropdown

# --- Texte für A.1, A.2, A.3 (werden von data\_input genutzt) ---

"customer\_data\_header": "Kundendaten",

"consumption\_analysis\_header": "Bedarfsanalyse",

"consumption\_costs\_header": "Verbräuche und Kosten",

"building\_data\_header": "Daten des Gebäudes",

"customer\_type\_label": "Kundentyp",

"anlage\_type\_label": "Anlagentyp",

"feed\_in\_type\_label": "Einspeisetyp",

"salutation\_label": "Anrede",

"title\_label": "Titel",

"first\_name\_label": "Vorname",

"last\_name\_label": "Nachname",

"num\_persons\_label": "Anzahl Personen",

"full\_address\_label": "Komplette Adresse", # Neu

"full\_address\_help": "Fügen Sie hier die komplette Adresse ein (z.g. 'Musterweg 18, 12345 Musterstadt, Deutschland')", # Neu

"parse\_address\_button": "Daten aus Adresse übernehmen", # Neu

"street\_label": "Straße", # Neu

"house\_number\_label": "Hausnummer", # Neu

"zip\_code\_label": "PLZ",

"city\_label": "Ort",

"state\_label": "Bundesland",

"region\_label": "Region",

"email\_label": "E-Mail",

"phone\_landline\_label": "Tel. (Festnetz)",

"phone\_mobile\_label": "Tel. (Mobil)",

"annual\_consumption\_kwh\_label": "Jahresverbrauch Haushalt (kWh)",

"annual\_heating\_kwh\_optional\_label": "Jahresverbrauch Heizung (kWh, optional)",

"total\_annual\_consumption\_label": "Gesamtjahresverbrauch (kWh)",

"monthly\_costs\_household\_label": "Monatliche Kosten Haushalt (€)",

"monthly\_costs\_heating\_optional\_label": "Monatliche Kosten Heizung (€, optional)",

"total\_annual\_costs\_label": "Gesamtjahreskosten (€)",

"electricity\_price\_label": "Strompreis (€/kWh)",

"consumption\_unknown\_button": "Verbrauch unbekannt?",

"build\_year\_label": "Baujahr des Hauses", # Angepasst

"build\_year\_warning\_old": "❗ Zählerschrank/Hauselektrik alt. Prüfung nötig!",

"build\_year\_info\_mid": "ℹ Zählerschrank/Hauselektrik sollte geprüft werden.",

"build\_year\_success\_new": "✅ Hauselektrik wahrscheinlich kompatibel.",

"roof\_type\_label": "Dachart",

"roof\_covering\_label": "Dachdeckungsart", # Angepasst

"roof\_covering\_warning": "❗ Bei dieser Dachdeckungsart können höhere Montagekosten anfallen.",

"roof\_covering\_info": "✅ Dachbelegung voraussichtlich problemlos.",

"roof\_orientation\_label": "Dachausrichtung", # Angepasst

"roof\_inclination\_label": "Dachneigung (Grad)", # Angepasst

"building\_height\_gt\_7m\_label": "Gebäudehöhe > 7 Meter (Gerüst erforderlich)", # Angepasst

"future\_consumption\_header": "Zukünftiger Mehrverbrauch",

"future\_consumption\_info": "Optionen für E-Auto/WP werden hier später aktiviert.",

"technology\_selection\_header": "Auswahl der Technik",

"module\_quantity\_label": "Anzahl PV Module", # Angepasst

"module\_model\_label": "PV Modul Modell", # Angepasst

"module\_capacity\_label": "Leistung pro Modul (W)", # Angepasst

"module\_details\_not\_found": "Modul Details nicht gefunden",

"module\_details\_not\_loaded": "Details konnten nicht geladen werden.",

"anlage\_size\_label": "Anlagengröße (kWp)", # Angepasst

"inverter\_selection\_header": "Wechselrichter Auswahl",

"inverter\_model\_label": "Wechselrichter Modell",

"inverter\_power\_label": "Leistung Wechselrichter (kW)", # Angepasst

"inverter\_details\_not\_found": "Wechselrichter Details nicht gefunden", # Angepasst

"inverter\_details\_not\_loaded": "Details für Wechselrichter konnten nicht geladen werden.", # Angepasst

"storage\_selection\_header": "Batteriespeicher Auswahl",

"include\_storage\_label": "Batteriespeicher einplanen",

"storage\_model\_label": "Speicher Modell",

"storage\_capacity\_model\_label": "Kapazität Modell (kWh)",

"storage\_details\_not\_found": "Speicher Details nicht gefunden",

"storage\_details\_not\_loaded": "Details für Speicher konnten nicht geladen werden.", # Angepasst

"storage\_capacity\_manual\_label": "Gewünschte Gesamtkapazität (kWh)",

"additional\_components\_header": "Zusätzliche Komponenten", # Geänderter Text

"include\_additional\_components\_label": "Zusätzliche Komponenten einplanen", # Neuer Text

"other\_components\_header": "Standard Zubehör (Wallbox, EMS, Optimierer)", # Neuer Unter-Header

"other\_additional\_components\_header": "Sonstige Zusatzkomponenten", # Neuer Unter-Header

"wallbox\_model\_label": "Wallbox Modell",

"ems\_model\_label": "Energiemanagementsystem Modell", # Angepasst

"optimizer\_model\_label": "Leistungsoptimierer Modell", # Angepasst

"carport\_model\_label": "Carport Modell", # Neu

"notstrom\_model\_label": "Notstromversorgung Modell", # Neu

"tierabwehr\_model\_label": "Tierabwehrschutz Modell", # Neu

# --- Texte für CRM (werden von crm.py genutzt) ---

"crm\_customer\_list\_header": "Kundenliste",

"crm\_select\_customer\_placeholder": "--- Kunden auswählen ---",

"crm\_select\_customer\_label": "Kunden auswählen",

"crm\_no\_customers\_found": "Keine Kunden in der Datenbank gefunden.",

"crm\_customer\_details\_header": "Kundendetails",

"crm\_name\_label": "Name",

"crm\_type\_label": "Typ",

"crm\_contact\_label": "Kontakt",

"crm\_address\_label": "Adresse",

"crm\_location\_label": "Region", # Angepasst von "Location"

"crm\_num\_persons\_label": "Personen im Haushalt",

"crm\_projects\_header": "Projekte",

"crm\_no\_projects\_found": "Keine Projekte für diesen Kunden gefunden.",

"crm\_project\_name\_label": "Projektname",

"crm\_anlage\_type\_label": "Anlagentyp",

"crm\_feed\_in\_type\_label": "Einspeisetyp",

"crm\_module\_model\_label": "Modul Modell",

"crm\_module\_quantity\_label": "Anzahl Module",

"crm\_anlage\_kwp\_label": "Anlagengröße (kWp)", # Neu für CRM Anzeige

"crm\_inverter\_model\_label": "WR Modell", # Neu für CRM Anzeige

"crm\_storage\_model\_label": "Speicher Modell", # Neu für CRM Anzeige

"crm\_storage\_capacity\_label": "Speicher Kapazität (kWh)", # Neu für CRM Anzeige

"crm\_build\_year\_label": "Baujahr", # Neu für CRM Anzeige

"crm\_roof\_type\_label": "Dachart", # Neu für CRM Anzeige

"crm\_created\_at\_label": "Erstellt am", # Neu für CRM Anzeige

"crm\_project\_actions\_header": "Projektaktionen (Placeholder)",

"crm\_edit\_project\_button": "Projekt bearbeiten",

"crm\_delete\_project\_button": "Projekt löschen",

"crm\_generate\_offer\_button": "Angebot erstellen",

"crm\_customer\_actions\_header": "Kundenaktionen (Placeholder)",

"crm\_edit\_customer\_button": "Kunden bearbeiten",

"crm\_delete\_customer\_button": "Kunden löschen",

"crm\_set\_follow\_up\_button": "Wiedervorlage setzen",

# Neue Schlüssel für zusätzliche Kategorien in CRM Anzeige

"crm\_wallbox\_model\_label": "Wallbox Modell",

"crm\_ems\_model\_label": "EMS Modell",

"crm\_optimizer\_model\_label": "Optimierer Modell",

"crm\_carport\_model\_label": "Carport Modell",

"crm\_notstrom\_model\_label": "Notstrom Modell",

"crm\_tierabwehr\_model\_label": "Tierabwehrschutz Modell",

# --- Texte für A.5 Ergebnisse ---

"dashboard\_header": "Ergebnisse und Dashboard",

"results\_preliminary\_header": "--- Berechnungsergebnisse ---", # Header für vorläufige Anzeige (Text angepasst)

"calculation\_errors\_header": "Es traten Fehler während der Berechnungen auf:",

"calculation\_no\_result\_info": "Berechnungen lieferten kein Ergebnis.",

"calculations\_module\_unavailable": "Berechnungsmodul nicht verfügbar. Ergebnisse können nicht angezeigt werden.", # Angepasst

# Text Keys für einzelne Ergebniswerte (können automatisch generiert werden oder hier spezifisch)

"total\_investment\_netto": "Gesamtinvestition (netto)",

"total\_investment\_brutto": "Gesamtinvestition (brutto)",

"cost\_modules": "Kosten Module",

"cost\_inverter": "Kosten Wechselrichter",

"cost\_storage": "Kosten Speicher",

"installation\_cost": "Installationskosten",

"vat\_rate\_percent": "MwSt Rate (%)",

"specific\_yield\_kwh\_kwp\_yr": "Spezifischer Ertrag (kWh/kWp/Jahr)",

"annual\_pv\_production\_kwh": "Jahresproduktion (kWh)",

"estimated\_self\_consumption\_rate": "Eigenverbrauchsanteil (geschätzt)",

"annual\_self\_consumption\_kwh": "Jährlicher Eigenverbrauch (kWh)",

"annual\_feed\_in\_kwh": "Jährliche Einspeisung (kWh)",

"annual\_grid\_consumption\_kwh": "Jährlicher Netzbezug (kWh)",

"self\_supply\_rate\_percent": "Eigenversorgungsgrad (%)",

"feed\_in\_tariff\_euro\_kwh": "Einspeisetarif (€/kWh)",

"annual\_savings\_eigenverbrauch": "Jährliche Ersparnis (Eigenverbrauch)",

"annual\_revenue\_einspeisung": "Jährliche Einnahmen (Einspeisung)",

"annual\_financial\_benefit": "Jährlicher finanzieller Vorteil",

"amortization\_time\_years": "Amortisationszeit (Jahre)",

"npv\_over\_years": "Kapitalwert (NPV) über Jahre",

"irr\_percent": "Interner Zinsfuß (IRR %)",

"annual\_co2\_savings\_kg": "Jährliche CO₂-Einsparung (kg)",

# Neue Text Schlüssel für Admin Panel

"admin\_price\_matrix\_header": "Preis-Matrix Verwaltung",

"admin\_price\_matrix\_info": "Hier können Sie die Preis-Matrix per CSV-Datei hochladen und speichern. Die Matrix wird für die Kostenberechnung benötigt.",

"admin\_upload\_matrix\_label": "CSV-Datei für Preis-Matrix hochladen",

"admin\_upload\_matrix\_help": "Wählen Sie die exportierte CSV-Datei der Preis-Matrix aus.",

"admin\_matrix\_loaded\_status": "Preis-Matrix ist aktuell geladen.",

"admin\_matrix\_head": "Erste 5 Zeilen:",

"admin\_matrix\_not\_loaded\_status": "Aktuell ist keine Preis-Matrix geladen.",

"admin\_process\_uploaded\_matrix\_button": "Datei laden & speichern", # Basis-Text für Button

"admin\_matrix\_load\_success": "Preis-Matrix '{file\_name}' erfolgreich geladen und gespeichert!", # Mit Platzhalter

"admin\_matrix\_load\_failure": "Fehler beim Laden der Preis-Matrix '{file\_name}'. Prüfen Sie das Dateiformat und die Struktur sowie die Konsole für Details.", # Mit Platzhalter

"admin\_general\_settings\_header": "Allgemeine Einstellungen",

"admin\_general\_settings\_info": "Verwaltung allgemeiner Einstellungen (Tarife, Margen, etc.) kommt hierher.",

"admin\_product\_db\_header": "Produktdatenbank",

"admin\_product\_db\_info": "Verwaltung der Produktdatenbank (Import/Export, Bearbeiten) kommt hierher.",

"admin\_user\_management\_header": "Benutzerverwaltung",

"admin\_user\_management\_info": "Verwaltung von Benutzern und Berechtigungen kommt hierher (optional).",

"admin\_module\_unavailable": "Admin Funktionen sind aufgrund fehlender Module nicht verfügbar.",

"admin\_clear\_matrix\_button": "Geladene Matrix entfernen",

"admin\_matrix\_cleared": "Matrix aus Session entfernt.",

"admin\_matrix\_save\_failure": "Ein Fehler ist beim Speichern der Preis-Matrix in der Datenbank aufgetreten.", # New text key

"admin\_setting\_vat\_rate": "Standard MwSt Rate (%)", # Placeholder

"admin\_save\_settings\_button": "Einstellungen speichern", # Placeholder

"admin\_settings\_saved\_success": "Einstellung gespeichert!", # Placeholder

"admin\_upload\_product\_db\_label": "Produktdatenbank (Excel) hochladen", # Placeholder

"admin\_import\_results": "Import Ergebnisse:", # Placeholder

"admin\_matrix\_clear\_failure": "Fehler beim Entfernen der Matrix aus der Datenbank.", # Placeholder

# Neue Text Schlüssel für Quick Calc Panel

"quick\_calc\_info": "Die Schnellberechnung ermöglicht Schätzungen mit wenigen Eingaben (Platzhalter).", # Placeholder

"quick\_calc\_power\_input": "Gewünschte PV-Leistung (kWp)", # Placeholder

"quick\_calc\_consumption\_input": "Geschätzter Jahresverbrauch (kWh)", # Placeholder

"quick\_calc\_calculate\_button": "Berechnen", # Placeholder

"quick\_calc\_results\_header": "Ergebnisse Schnellberechnung", # Placeholder

"quick\_calc\_estimated\_cost": "Geschätzte Kosten", # Placeholder

"quick\_calc\_estimated\_production": "Geschätzte Jahresproduktion", # Placeholder

"quick\_calc\_details\_note": "Detaillierte Berechnung und Wirtschaftlichkeitsanalyse finden Sie im Tab A.", # Placeholder

# Neuer Text Schlüssel für Info Platform Panel

"info\_platform\_content\_placeholder": "Inhalte zur Info-Plattform werden hier geladen und angezeigt (Platzhalter).", # Placeholder

# Neuer Text Schlüssel für Options Panel

"options\_content\_placeholder": "Die App-Einstellungen können hier konfiguriert werden (Platzhalter).", # Placeholder

"options\_language\_header": "Sprache auswählen", # Placeholder

"options\_language\_label": "Sprache", # Placeholder

"options\_save\_load\_header": "Speicherverhalten", # Placeholder

"options\_auto\_save": "Projekte automatisch speichern", # Placeholder

"options\_save\_settings\_button": "Optionen speichern", # Placeholder

"options\_settings\_saved\_success": "Optionen gespeichert!", # Placeholder

# Potenziell benötigte zukünftige Schlüssel für analysis.py Details (schon oben bei A.5)

"cost\_details\_header": "Kosten Details", # Platzhalter

"yield\_consumption\_details\_header": "Ertrags- und Verbrauchsdetails", # Placeholder

"economy\_details\_header": "Wirtschaftlichkeitsdetails", # Placeholder

"co2\_details\_header": "CO₂ Details", # Placeholder

"future\_planning\_header": "Zukunftsplanung", # Placeholder

"future\_planning\_info": "Prognosen für zukünftigen Verbrauch und Systemerweiterungen werden hier angezeigt (Platzhalter).", # Placeholder

"scenario\_simulation\_header": "Szenariensimulationen", # Placeholder

"scenario\_simulation\_info": "Simulationen unter verschiedenen Bedingungen (Wartung, Strompreise) kommen hierher (Platzhalter).", # Placeholder

"financial\_comparison\_header": "Finanzierungsvergleich", # Placeholder

"financial\_comparison\_info": "Vergleich PV-Anlage vs. andere Kapitalanlagen kommt hierher (Platzhalter).", # Placeholder

"amortization\_time\_infinite": "Unendlich", # Text für unendliche Amortisationszeit

"analysis\_storage\_cost\_note": "Hinweis: Die 'Kosten Speicher (Aufpreis Produkt-DB netto)' werden zusätzlich zum Matrix Basispreis ausgewiesen. Prüfen Sie Ihre Preis-Matrix Definition, um eine Doppeltzählung zu vermeiden, falls der Matrixpreis bereits den Speicher abdeckt.", # Hinweis Text

"cost\_visualization\_header": "Kostenübersicht Visualisierung", # New header for cost chart

"energy\_flow\_visualization\_header": "Energiefluss Visualisierung", # New header for energy flow chart

"cost\_visualization\_no\_data": "Keine Kostendaten für Visualisierung verfügbar (Gesamtkosten sind 0).", # New info text

"energy\_flow\_visualization\_no\_data": "Keine Energieflussdaten für Visualisierung verfügbar (alle Werte sind 0).", # New info text

"simulation\_details\_no\_data": "Simulationsdetails konnten nicht geladen oder berechnet werden.", # New info text for table

# Keys for detailed cost breakdown table headers/labels (from calculations.py results)

# These keys are used in analysis.py to format the cost table dataframe.

# They map to the keys used in the results dictionary.

# We define them here as they are part of the UI text elements.

# Some are already defined above (e.g. 'cost\_modules'), ensure they are consistent.

# 'base\_matrix\_price\_netto' -> use texts.get('base\_matrix\_price\_netto', 'Matrix Basispreis (netto)')

# 'cost\_modules\_aufpreis\_netto' -> use texts.get('cost\_modules', 'Kosten Module')

# 'cost\_inverter\_aufpreis\_netto' -> use texts.get('cost\_inverter', 'Kosten Wechselrichter')

# 'cost\_storage\_aufpreis\_product\_db\_netto' -> use texts.get('cost\_storage', 'Kosten Speicher')

# 'cost\_accessories\_aufpreis\_netto' -> use texts.get('cost\_accessories\_aufpreis\_netto', 'Kosten Zubehör')

# 'cost\_product\_surcharges\_netto' -> use texts.get('cost\_product\_surcharges\_netto', 'Summe Produkt Aufpreise')

# 'cost\_scaffolding\_netto' -> use texts.get('cost\_scaffolding\_netto', 'Kosten Gerüst')

# 'cost\_misc\_netto' -> use texts.get('cost\_misc\_netto', 'Sonstige Kosten')

# 'cost\_custom\_netto' -> use texts.get('cost\_custom\_netto', 'Kundenwunsch Kosten')

# 'total\_additional\_costs\_netto' -> use texts.get('total\_additional\_costs\_netto', 'Summe zusätzliche Kosten')

# 'subtotal\_netto' -> use texts.get('subtotal\_netto', 'Zwischensumme')

# 'custom\_surcharge\_amount\_netto' -> use texts.get('custom\_surcharge\_amount\_netto', 'Aufschlag')

# 'custom\_discount\_amount\_netto' -> use texts.get('custom\_discount\_amount\_netto', 'Rabatt')

# 'total\_investment\_netto' -> use texts.get('total\_investment\_netto', 'Gesamtinvestition (netto)')

# 'vat\_rate\_percent' -> use texts.get('vat\_rate\_percent', 'MwSt Rate (%)')

# 'total\_investment\_brutto' -> use texts.get('total\_investment\_brutto', 'Gesamtinvestition (brutto)')

# Keys for simulation table headers/labels (from calculations.py results)

"analysis\_chart\_year\_label": "Jahr", # Axis label for charts

"analysis\_chart\_cumulative\_cf\_label": "Kumulierter Cash Flow (€)", # Series label for cumulative CF chart

"analysis\_chart\_annual\_cf\_label": "Jährlicher Cash Flow (€)", # Series label for annual CF chart

"simulation\_details\_header": "Simulationsdetails (Jährlich)", # Header for simulation table

"analysis\_table\_year\_header": "Jahr", # Column header for Year in table

"analysis\_table\_annual\_cf\_header": "Jährlicher Cash Flow (€)", # Column header for annual CF in table

"analysis\_table\_cumulative\_cf\_header": "Kumulierter Cash Flow (€)", # Column header for cumulative CF in table

"annual\_maintenance\_cost\_sim": "Wartungskosten (€)", # Column header for maintenance in table

}

# --- Importversuche für reale Module ---

# Weisen Sie globale Variablen initial den Dummy-Funktionen zu

render\_data\_input = Dummy\_render\_module

perform\_calculations = Dummy\_perform\_calculations

render\_analysis = Dummy\_render\_analysis # KORRIGIERT: Nutze die Dummy\_render\_analysis mit richtiger Signatur

render\_crm = Dummy\_render\_module

save\_customer = Dummy\_save\_customer

save\_project = Dummy\_save\_project

render\_info\_platform = Dummy\_render\_module # Dummy für info\_platform

render\_options = Dummy\_render\_module # Dummy für options

render\_admin\_panel = Dummy\_render\_module # Dummy für admin\_panel

render\_quick\_calc = Dummy\_render\_module # Dummy für quick\_calc

get\_db\_connection = Dummy\_get\_db\_connection

save\_admin\_setting = Dummy\_save\_admin\_setting

load\_admin\_setting = Dummy\_load\_admin\_setting

load\_translations = Dummy\_load\_translations

# Dummy für parse\_price\_matrix\_csv, falls calculations import fehlschlägt

parse\_price\_matrix\_csv = lambda file\_source: None # Muss hier existieren, falls import fehlschlägt

# Importiere Datenbankfunktionen ZUERST, da sie oft von anderen Modulen benötigt werden.

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe

try:

# WICHTIG: Stelle sicher, dass database.py im selben Ordner liegt wie gui.py und der Import funktioniert

from database import get\_db\_connection as real\_get\_db\_connection, init\_db as real\_init\_db, save\_admin\_setting as real\_save\_admin\_setting, load\_admin\_setting as real\_load\_admin\_setting

# Prüfe, ob importierte Objekte aufrufbar/gültig sind

if not callable(real\_get\_db\_connection): raise ImportError("Imported get\_db\_connection is not callable.")

if not callable(real\_load\_admin\_setting): raise ImportError("Imported load\_admin\_setting is not callable.")

# init\_db wird normalerweise von database.py selbst beim Laden aufgerufen, oder wir könnten es hier aufrufen

# real\_init\_db() # Optional, kann hier explizit aufgerufen werden, um sicherzustellen, dass es passiert

get\_db\_connection = real\_get\_db\_connection

save\_admin\_setting = real\_save\_admin\_setting

load\_admin\_setting = real\_load\_admin\_setting

# print("DEBUG: gui.py: Erfolgreich database.py Funktionen importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von database.py: {e}. Datenbankfunktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Importfehler database.py: {e}") # Debugging

# Dummy Funktionen bleiben aktiv (initial zugewiesen)

except Exception as e:

import\_errors.append(f"Fehler im database.py Modul beim Laden: {e}. Datenbankfunktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Fehler im Laden database.py: {e}") # Debugging

# Dummy Funktionen bleiben aktiv

# Importiere Lokalisierung als Nächstes

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe

try:

from locales import load\_translations as real\_load\_translations

if not callable(real\_load\_translations): raise ImportError("Imported load\_translations is not callable.")

load\_translations = real\_load\_translations # Überschreibe den Dummy

# print("DEBUG: gui.py: Erfolgreich locales.py Funktion importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von locales.py: {e}. Fallback-Texte werden verwendet.")

# print(f"DEBUG: gui.py: Importfehler locales.py: {e}") # Debugging

# Dummy load\_translations bleibt aktiv (initial zugewiesen)

except Exception as e:

import\_errors.append(f"Fehler im locales.py Modul beim Laden: {e}. Fallback-Texte werden verwendet.")

# print(f"DEBUG: gui.py: Fehler im Laden locales.py: {e}") # Debugging

# Dummy load\_translations bleibt aktiv

# Importiere Produktdatenbankfunktionen (benötigt database)

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe

try:

from product\_db import list\_products as real\_list\_products, get\_product\_by\_model\_name as real\_get\_product\_by\_model\_name

# Prüfen Sie diese Funktionen optional auf Aufrufbarkeit, wenn nötig

# print("DEBUG: gui.py: Erfolgreich product\_db.py Funktionen importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von product\_db.py: {e}. Produktfunktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Importfehler product\_db.py: {e}") # Debugging

except Exception as e:

import\_errors.append(f"Fehler im product\_db.py Modul beim Laden: {e}. Produktfunktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Fehler im Laden product\_db.py: {e}") # Debugging

# Importiere CRM Funktionen (benötigt database)

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe

try:

from crm import render\_crm as real\_render\_crm, save\_customer as real\_save\_customer, save\_project as real\_save\_project

if not callable(real\_render\_crm): raise ImportError("Imported render\_crm is not callable.")

if not callable(real\_save\_customer): raise ImportError("Imported save\_customer is not callable.")

if not callable(real\_save\_project): raise ImportError("Imported save\_project is not callable.")

render\_crm = real\_render\_crm

save\_customer = real\_save\_customer

save\_project = real\_save\_project

# print("DEBUG: gui.py: Erfolgreich crm.py Funktionen importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von crm.py: {e}. CRM-Funktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Importfehler crm.py: {e}") # Debugging

except Exception as e:

import\_errors.append(f"Fehler im crm.py Modul beim Laden: {e}. CRM-Funktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Fehler im Laden crm.py: {e}") # Debugging

# Importiere Data Input Modul (kann product\_db und calculations benötigen)

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe

try:

from data\_input import render\_data\_input as real\_render\_data\_input

if not callable(real\_render\_data\_input): raise ImportError("Imported render\_data\_input is not callable.")

render\_data\_input = real\_render\_data\_input

# print("DEBUG: gui.py: Erfolgreich data\_input.py Funktion importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von data\_input.py: {e}. Eingabefunktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Importfehler data\_input.py: {e}") # Debugging

except Exception as e:

import\_errors.append(f"Fehler im data\_input.py Modul beim Laden: {e}. Eingabefunktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Fehler im Laden data\_input.py: {e}") # Debugging

# Importiere Calculations Modul (kann product\_db und database benötigen)

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe und Import von parse\_price\_matrix\_csv

try:

from calculations import perform\_calculations as real\_perform\_calculations, parse\_price\_matrix\_csv as real\_parse\_price\_matrix\_csv

if not callable(real\_perform\_calculations): raise ImportError("Imported perform\_calculations is not callable.")

if not callable(real\_parse\_price\_matrix\_csv): raise ImportError("Imported parse\_price\_matrix\_csv is not callable.") # Prüfen

perform\_calculations = real\_perform\_calculations # Weise die reale Funktion zu

parse\_price\_matrix\_csv = real\_parse\_price\_matrix\_csv # KORRIGIERT: Zuweisung der realen Funktion

# print("DEBUG: gui.py: Erfolgreich calculations.py Funktionen importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von calculations.py: {e}. Berechnungsfunktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Importfehler calculations.py: {e}") # Debugging

# Dummy perform\_calculations bleibt aktiv

# Dummy parse\_price\_matrix\_csv bleibt aktiv

# Definiere hier einen Dummy für parse\_price\_matrix\_csv, falls der Import fehlschlägt

def parse\_price\_matrix\_csv(file\_source: Any): # Muss hier existieren, falls import fehlschlägt

print("DEBUG: Dummy parse\_price\_matrix\_csv called (calculations import failed)")

return None

except Exception as e:

import\_errors.append(f"Fehler im calculations.py Modul beim Laden: {e}. Berechnungsfunktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Fehler im Laden calculations.py: {e}") # Debugging

# Dummy perform\_calculations bleibt aktiv

# Dummy parse\_price\_matrix\_csv bleibt aktiv

# Definiere hier einen Dummy für parse\_price\_matrix\_csv

def parse\_price\_matrix\_csv(file\_source: Any): # Muss hier existieren, falls import fehlschlägt

print("DEBUG: Dummy parse\_price\_matrix\_csv called (calculations import failed)")

return None

# Importiere Analysis Modul (wird calculation results verarbeiten/anzeigen, kann database benötigen)

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe und Nutzung der Dummy\_render\_analysis

try:

from analysis import render\_analysis as real\_render\_analysis

if not callable(real\_render\_analysis): raise ImportError("Imported render\_analysis is not callable.")

render\_analysis = real\_render\_analysis # Weise die reale Funktion zu

# print("DEBUG: gui.py: Erfolgreich analysis.py Funktion importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von analysis.py: {e}. Analyse-Funktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Importfehler analysis.py: {e}") # Debugging

render\_analysis = Dummy\_render\_analysis # KORRIGIERT: Weise die Dummy Funktion zu

except Exception as e:

import\_errors.append(f"Fehler im analysis.py Modul beim Laden: {e}. Analyse-Funktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Fehler im Laden analysis.py: {e}") # Debugging

render\_analysis = Dummy\_render\_analysis # KORRIGIERT: Weise die Dummy Funktion zu

# Importiere Info Platform Modul

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe

try:

from info\_platform import render\_info\_platform as real\_render\_info\_platform

if not callable(real\_render\_info\_platform): raise ImportError("Imported render\_info\_platform is not callable.")

render\_info\_platform = real\_render\_info\_platform

# print("DEBUG: gui.py: Erfolgreich info\_platform.py Funktion importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von info\_platform.py: {e}. Info-Plattform Funktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Importfehler info\_platform.py: {e}") # Debugging

except Exception as e:

import\_errors.append(f"Fehler im info\_platform.py Modul beim Laden: {e}. Info-Plattform Funktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Fehler im Laden info\_platform.py: {e}") # Debugging

# Importiere Options Modul (kann database benötigen)

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe

try:

from options import render\_options as real\_render\_options

if not callable(real\_render\_options): raise ImportError("Imported render\_options is not callable.")

render\_options = real\_render\_options

# print("DEBUG: gui.py: Erfolgreich options.py Funktion importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von options.py: {e}. Optionen Funktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Importfehler options.py: {e}") # Debugging

except Exception as e:

import\_errors.append(f"Fehler im options.py Modul beim Laden: {e}. Optionen Funktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Fehler im Laden options.py: {e}") # Debugging

# Importiere Admin Panel Modul (kann database benötigen)

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe

try:

from admin\_panel import render\_admin\_panel as real\_render\_admin\_panel # KORRIGIERT: Verwenden Sie den korrekten Aliasnamen

if not callable(real\_render\_admin\_panel): raise ImportError("Imported render\_admin\_panel is not callable.")

render\_admin\_panel = real\_render\_admin\_panel # Weise die reale Funktion zu

# print("DEBUG: gui.py: Erfolgreich admin\_panel.py Funktion importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von admin\_panel.py: {e}. Admin Panel Funktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Importfehler admin\_panel.py: {e}") # Debugging

# Dummy render\_admin\_panel bleibt aktiv

except Exception as e:

import\_errors.append(f"Fehler im admin\_panel.py Modul beim Laden: {e}. Admin Panel Funktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Fehler im Laden admin\_panel.py: {e}") # Debugging

# Dummy render\_admin\_panel bleibt aktiv

# Importiere Quick Calc Modul (kann product\_db und calculations benötigen)

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe

try:

from quick\_calc import render\_quick\_calc as real\_render\_quick\_calc # KORRIGIERT: Verwenden Sie den korrekten Aliasnamen

if not callable(real\_render\_quick\_calc): raise ImportError("Imported render\_quick\_calc is not callable.")

render\_quick\_calc = real\_render\_quick\_calc # Weise die reale Funktion zu

# print("DEBUG: gui.py: Erfolgreich quick\_calc.py Funktion importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von quick\_calc.py: {e}. Schnellberechnungsfunktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Importfehler quick\_calc.py: {e}") # Debugging

except Exception as e:

import\_errors.append(f"Fehler im quick\_calc.py Modul beim Laden: {e}. Schnellberechnungsfunktionen nicht verfügbar.")

# print(f"DEBUG: gui.py: Fehler im Laden quick\_calc.py: {e}") # Debugging

# Importiere Lokalisierung zuletzt

# KORRIGIERT: Konsistente Fehlerbehandlung für Importe

try:

from locales import load\_translations as real\_load\_translations

if not callable(real\_load\_translations): raise ImportError("Imported load\_translations is not callable.")

load\_translations = real\_load\_translations # Überschreibe den Dummy

# print("DEBUG: gui.py: Erfolgreich locales.py Funktion importiert.") # Debugging

except ImportError as e:

import\_errors.append(f"Fehler beim Laden von locales.py: {e}. Fallback-Texte werden verwendet.")

# print(f"DEBUG: gui.py: Importfehler locales.py: {e}") # Debugging

# Dummy load\_translations bleibt aktiv (initial zugewiesen)

except Exception as e:

import\_errors.append(f"Fehler im locales.py Modul beim Laden: {e}. Fallback-Texte werden verwendet.")

# print(f"DEBUG: gui.py: Fehler im Laden locales.py: {e}") # Debugging

# Dummy load\_translations bleibt aktiv

# Add a new function to load persistent data on startup

# Benötigt load\_admin\_setting (global verfügbar) und parse\_price\_matrix\_csv (global verfügbar)

# Benötigt auch io

import io # Importiere io

def load\_persistent\_data():

"""

Lädt persistente Daten (z.B. Preis-Matrix) aus der Datenbank in den Session State beim App-Start.

Diese Funktion wird einmalig beim Start der Main-App in gui.py aufgerufen.

"""

# Prüfen Sie, ob die Preis-Matrix nicht bereits im Session State geladen ist

# Dies ist nützlich bei Hot-Reloads während der Entwicklung

if 'price\_matrix\_df' in st.session\_state and st.session\_state['price\_matrix\_df'] is not None:

# Matrix ist bereits geladen, nichts tun

# print("DEBUG: Preis-Matrix bereits im Session State beim Start vorhanden.") # Debugging

return

# Benötigt load\_admin\_setting (global verfügbar, Dummy oder Real)

# Benötigt parse\_price\_matrix\_csv (global verfügbar, Dummy oder Real)

# Stelle sicher, dass parse\_price\_matrix\_csv verfügbar ist (wird durch import block oben gehandhabt)

# Lade die Preis-Matrix als CSV String aus den Admin Settings

db\_csv\_string = load\_admin\_setting('price\_matrix\_csv\_data')

if db\_csv\_string is not None and isinstance(db\_csv\_string, str):

try:

# Parse den CSV String zurück in einen DataFrame

# io.StringIO ermöglicht pandas, einen String wie eine Datei zu lesen

# Benutzen Sie die globale parse\_price\_matrix\_csv Funktion

# Stellen Sie sicher, dass parse\_price\_matrix\_csv fähig ist, StringIO zu verarbeiten

# (Unsere Implementierung ist es)

parsed\_df = parse\_price\_matrix\_csv(io.StringIO(db\_csv\_string))

if parsed\_df is not None:

st.session\_state['price\_matrix\_df'] = parsed\_df

# print("DEBUG: Preis-Matrix erfolgreich von DB in Session State geladen beim Start.") # Debugging

else:

# Fehler beim Parsen der DB-Daten

print("FEHLER: Konnte Preis-Matrix String aus DB beim Start nicht parsen. Datenstruktur evtl. beschädigt.")

# Session State bleibt None, Admin Bereich wird Status 'nicht geladen' anzeigen.

except Exception as e:

print(f"FEHLER: Unerwarteter Fehler beim Laden/Parsen der Preis-Matrix aus DB beim Start: {e}")

traceback.print\_exc()

# Session State bleibt None.

# Wenn db\_csv\_string None ist, bleibt st.session\_state['price\_matrix\_df'] ebenfalls None (Initialwert),

# was korrekt ist, da keine Matrix in der DB gefunden wurde. Admin Bereich wird dies anzeigen.

# --- Hauptfunktion der Anwendung ---

def main(texts: dict):

"""

Hauptfunktion, die die Streamlit App strukturiert.

"""

# set\_page\_config muss der erste Streamlit-Befehl sein, der aufgerufen wird.

# Da Imports oben schon Streamlit-Befehle ausführen könnten (obwohl sie es idealerweise nicht sollten),

# stellen wir sicher, dass dies hier der erste st.\* Befehl ist, der die App-Konfiguration setzt.

# Eine robustere Prüfung, ob die Seite bereits konfiguriert wurde.

try:

st.set\_page\_config(layout="wide", page\_title=texts.get("app\_title", "Solar App"))

except st.errors.StreamlitAPIException as e:

# Fängt den Fehler ab, wenn set\_page\_config nicht am Anfang aufgerufen wurde

if "set\_page\_config()" in str(e):

# Dies ist der erwartete Fehler beim Hot-Reloading oder wenn vorher st.\* Befehle kamen.

# print("DEBUG: StreamlitAPIException beim set\_page\_config abgefangen (erwartet bei Rerun).") # Debugging

pass # Ignoriere diesen spezifischen Fehler

else:

# Andere unerwartete StreamlitAPIExceptions

st.error(f"Unerwarteter Streamlit Fehler bei set\_page\_config: {e}")

traceback.print\_exc()

# Initialisiere session\_state falls nötig, bevor darauf zugegriffen wird

if 'current\_project\_id' not in st.session\_state:

st.session\_state['current\_project\_id'] = None # ID des aktuell geladenen/bearbeiteten Projekts

# Fügen Sie hier weitere globale Session States hinzu

# z.g. für den aktiven Tab, falls Sie die Navigation über Session State steuern möchten

# if 'active\_tab' not in st.session\_state:

# st.session\_state['active\_tab'] = texts.get("menu\_item\_input", "Eingabe (A)")

# KORRIGIERT: Lade persistente Daten nach der Session State Initialisierung, aber VOR dem Titel/UI Rendering

load\_persistent\_data()

st.title(texts.get("app\_title", "Ömer´s DingsDa für Solar"))

# Anzeige der Import-Fehler ganz oben, falls vorhanden

if import\_errors:

st.error(texts.get("import\_errors\_title", "⚠ Modul-Ladefehler:"))

for error in import\_errors:

st.error(error)

st.warning("Einige App-Teile funktionieren möglicherweise nicht korrekt. Bitte fehlende Dateien prüfen/korrigieren.")

# Hauptnavigation (Tabs) für die Bereiche A-F

# Die Texte für die Tabs kommen aus dem texts Dictionary

tab\_titles = [

texts.get("menu\_item\_input", "Eingabe (A)"), # A: Detaillierte Kalkulation

texts.get("menu\_item\_quick\_calc", "Schnellberechnung (B)"), # B: Schnellberechnung

texts.get("menu\_item\_crm", "CRM (C)"), # C: CRM

texts.get("menu\_item\_info\_platform", "Info-Plattform (D)"), # D: Informationsplattform

texts.get("menu\_item\_options", "Optionen (E)"), # E: Optionen

texts.get("menu\_item\_admin", "Admin (F)") # F: Adminbereich

]

# Nutzen Sie Session State für die Tab-Auswahl, falls Sie die aktive Registerkarte beibehalten möchten

# Aktiver Tab wird beim Klick auf einen Tab aktualisiert

# index=0 wählt standardmäßig den ersten Tab aus

tab\_a, tab\_b, tab\_c, tab\_d, tab\_e, tab\_f = st.tabs(tab\_titles)

# --- Inhalte der Tabs rendern ---

# Rufe die render Funktion des jeweiligen Moduls auf

# Übergebe das texts Dictionary und ggf. andere notwendige Argumente (wie DB-Verbindung)

with tab\_a:

# Header anzeigen

st.header(texts.get("menu\_item\_input", "Eingabe (A) - Detaillierte Kalkulation"))

# Rufe die render Funktion aus data\_input.py auf (kann Dummy sein)

# Die render\_data\_input Funktion ist das texts Dictionary bekannt

# data\_input.py wird sich die DB-Verbindung und Produktdaten-Funktionen selbst holen müssen

project\_data = render\_data\_input(texts) # Sammelt die Eingaben

# Speichern Button nur anzeigen, wenn data\_input und die notwendigen DB/CRM Funktionen real sind

# Prüfe die Funktionen, die direkt für den Speichervorgang in DIESER GUI-Funktion benötigt werden

if render\_data\_input is not Dummy\_render\_module and \

save\_customer is not Dummy\_save\_customer and \

save\_project is not Dummy\_save\_project and \

get\_db\_connection is not Dummy\_get\_db\_connection:

# Echter Speicherbutton

# Button-Click wird nur verarbeitet, wenn der Button nicht disabled ist (was er hier nicht ist)

if st.button(texts.get("save\_project\_button", "Projekt speichern")):

# Stelle sicher, dass Daten gesammelt wurden und die minimalen benötigten Felder da sind

if project\_data and project\_data.get('customer\_data') and project\_data.get('project\_details'):

conn = None # Verbindung initial None

try:

conn = get\_db\_connection() # Nutze die reale oder Dummy Funktion (hier sollte sie real sein)

if conn: # Prüfe, ob die Verbindung echt ist und geöffnet werden konnte

customer\_data = project\_data.get("customer\_data", {})

project\_details\_to\_save = project\_data.get("project\_details", {})

# Zuerst Kunden speichern/aktualisieren, um customer\_id zu bekommen

# save\_customer erwartet eine offene Verbindung und die Daten

# save\_customer sollte INSERT OR REPLACE oder ON CONFLICT machen und die ID zurückgeben

customer\_id = save\_customer(conn, customer\_data) # RUFE save\_customer AUF

if customer\_id is not None and customer\_id != -1:

# Füge die customer\_id zu den Projektdetails hinzu

project\_details\_to\_save['customer\_id'] = customer\_id

# Generiere Projektnamen-Fallback HIER, falls project\_name fehlt

if not project\_details\_to\_save.get('project\_name'): # Prüfe auf None, leeren String, etc.

customer\_name\_fallback = f"{customer\_data.get('first\_name', '').strip()} {customer\_data.get('last\_name', '').strip()}".strip() # Sicher trimmen

if customer\_name\_fallback:

project\_details\_to\_save['project\_name'] = f"Projekt {customer\_name\_fallback} - {datetime.now().strftime('%Y-%m-%d %H:%M')}" # Kürzere Zeit

else:

project\_details\_to\_save['project\_name'] = f"Projekt - {datetime.now().strftime('%Y-%m-%d %H:%M')}" # Kürzere Zeit

# Jetzt Projekt speichern

# save\_project erwartet eine offene Verbindung, die Daten und die customer\_id

# save\_project muss die project\_id zurückgeben oder -1 bei Fehler

project\_id = save\_project(conn, project\_details\_to\_save) # RUFE save\_project AUF

if project\_id is not None and project\_id != -1:

st.success(texts.get("save\_success", "Projekt erfolgreich gespeichert!"))

st.session\_state['current\_project\_id'] = project\_id # Speichere die neue Projekt-ID im Session State

conn.commit() # <<< Commit am Ende der Transaktion, wenn alles erfolgreich war

# print("DEBUG: gui.py: Projekt gespeichert und committed.") # Debugging

# Optional: Streamlit neu laden, um UI (z.g. CRM Liste) zu aktualisieren

# st.rerun()

else: # save\_project returned -1 oder None

st.error(texts.get("save\_failure", "Fehler beim Speichern der Projektdaten."))

conn.rollback() # <<< Rollback, wenn Projektspeicherung fehlschlägt

# print("DEBUG: gui.py: Rollback nach save\_project Fehler.") # Debugging

else: # save\_customer returned -1 oder None

st.error(texts.get("save\_failure", "Fehler beim Speichern der Kundendaten."))

conn.rollback() # <<< Rollback, wenn Kundenspeicherung fehlschlägt

# print("DEBUG: gui.py: Rollback nach save\_customer Fehler.") # Debugging

else: # get\_db\_connection() returned None

st.error(texts.get("db\_connection\_unavailable", "Keine Datenbankverbindung verfügbar für Speichervorgang."))

# print("DEBUG: gui.py: Keine DB-Verbindung für Speichervorgang verfügbar.") # Debugging

except Exception as e:

# Dieser Block fängt Fehler WÄHREND des Speicherns/Committens ab (z.g. beim commit selbst)

st.error(f"{texts.get('save\_failure', 'Ein Fehler ist beim Speichern aufgetreten.')} Details: {e}")

traceback.print\_exc() # Zeige detaillierten Fehler im Terminal/Log

# Rollback sollte idealerweise hier auch passieren, wenn conn geöffnet und nicht schon gerollt wurde

if conn:

try:

conn.rollback() # Sicherstellen, dass im Fehlerfall ein Rollback versucht wird

# print("DEBUG: gui.py: Rollback im allgemeinen Fehler-Handler.") # Debugging

except Exception as rb\_e:

# print(f"DEBUG: gui.py: Fehler während Rollback: {rb\_e}") # Debugging

pass # Ignore errors during rollback

finally:

if conn: # Ensure the main connection is closed

conn.close()

# print("DEBUG: gui.py: Datenbankverbindung nach Speichervorgang geschlossen.") # Debugging

else: # project\_data oder notwendige Unter-Dicts fehlen

st.warning("Keine oder unvollständige Daten zum Speichern vorhanden.")

else:

# Zeige den Button als disabled und mit Tooltip, wenn Speichern nicht möglich ist

st.button(texts.get("save\_project\_button", "Projekt speichern"),

help=texts.get("save\_project\_button\_disabled\_tooltip", "Speichern nicht möglich aufgrund fehlender Datenbank- oder CRM-Funktionen."),

disabled=True)

st.markdown("---") # Trennlinie vor den Ergebnissen

# --- 5. Allgemeine Berechnungen und Ergebnisse (A.5) ---

# Header wird nun direkt in analysis.py gerendert (oder kann hier bleiben, je nach Design)

# st.subheader(texts.get("dashboard\_header", "Ergebnisse und Dashboard")) # Header für Ergebnisse

# Führe Berechnungen mit den gesammelten Daten durch

# perform\_calculations erwartet project\_data

# Das Modul muss erfolgreich importiert werden

# KORRIGIERT: Vergleich mit der initial zugewiesenen Dummy-Funktion

calculation\_results = None # Ergebnis initial None setzen

# Importiere Streamlit hier, falls perform\_calculations es benötigt (für Session State)

# Der Import erfolgt ohnehin am Anfang des Moduls, aber hier explizit für Klarheit

# import streamlit as st # Bereits global importiert

# Rufe perform\_calculations auf, wenn das Modul verfügbar ist

if perform\_calculations is not Dummy\_perform\_calculations:

# Rufe die Berechnungen auf (sie sammeln Ergebnisse und Fehler)

# Stelle sicher, dass project\_data nicht leer ist, bevor Berechnungen aufgerufen werden

# project\_data enthält auch die ID des aktuell geladenen/bearbeiteten Projekts

if project\_data and project\_data.get('project\_details'): # Mindestens project\_details müssen da sein

# perform\_calculations wird hier aufgerufen (auch wenn die Ergebnisse noch nicht voll angezeigt werden)

# Stelle sicher, dass perform\_calculations st.session\_state nutzen kann (Streamlit Kontext oder simuliert)

calculation\_results = perform\_calculations(project\_data)

# else:

# st.info("Bitte füllen Sie die Eingabefelder aus, um die Berechnungen zu starten.") # Diese Info wird im Analysis Tab besser platziert

# Zeige die Ergebnisse an, indem du render\_analysis aufrufst

# render\_analysis wird texts und calculation\_results (kann None sein) erhalten

# render\_analysis wird selbst entscheiden, ob Ergebnisse da sind und wie sie angezeigt werden.

render\_analysis(texts, calculation\_results) # Übergib texts und die berechneten Ergebnisse (oder None)

with tab\_b:

st.header(texts.get("menu\_item\_quick\_calc", "Schnellberechnung (B)"))

# Rufe die render Funktion aus quick\_calc.py auf (kann Dummy sein)

render\_quick\_calc(texts, module\_name=texts.get("menu\_item\_quick\_calc", "Schnellberechnung")) # Übergib Modulname als Keyword Arg

with tab\_c:

st.header(texts.get("menu\_item\_crm", "CRM (C)"))

# Rufe die render Funktion aus crm.py auf (kann Dummy sein)

# crm.py wird sich die DB-Verbindung selbst holen müssen

if render\_crm is not Dummy\_render\_module:

render\_crm(texts)

else:

Dummy\_render\_module(module\_name=texts.get("menu\_item\_crm", "CRM")) # Nutze generische Dummy

with tab\_d:

st.header(texts.get("menu\_item\_info\_platform", "Info-Plattform (D)"))

# Rufe die render Funktion aus info\_platform.py auf (kann Dummy sein)

render\_info\_platform(texts, module\_name=texts.get("menu\_item\_info\_platform", "Info-Plattform")) # Übergib Modulname

with tab\_e:

st.header(texts.get("menu\_item\_options", "Optionen (E)"))

# Rufe die render Funktion aus options.py auf (kann Dummy sein)

render\_options(texts, module\_name=texts.get("menu\_item\_options", "Optionen")) # Übergib Modulname

with tab\_f:

st.header(texts.get("menu\_item\_admin", "Admin (F)"))

# Rufe die render Funktion aus admin\_panel.py auf (kann Dummy sein)

# Admin Panel braucht texts und load\_admin\_setting/save\_admin\_setting (die global zugewiesen sind)

render\_admin\_panel(texts) # render\_admin\_panel erhält nur texts, holt sich andere Abhängigkeiten selbst

# --- Startpunkt der Anwendung ---

if \_\_name\_\_ == "\_\_main\_\_":

# Lade die Texte für die Benutzeroberfläche

# Nutze hier 'de' als Standard. Später aus Optionen (E) laden

# Die globale Variable load\_translations wurde oben initialisiert

# Entweder ist sie die reale Funktion oder der Dummy.

# Beginnen Sie immer mit den Fallback-Texten als Basis

loaded\_texts = texts\_fallback.copy()

try:

# Versuchen Sie, die reale Lokalisierungsfunktion aufzurufen, WENN sie verfügbar ist

# Prüfen Sie, ob load\_translations die reale Funktion ist und aufrufbar

if load\_translations is not Dummy\_load\_translations:

real\_texts = load\_translations("de") # Lade spezifische Sprache (hartcodiert 'de' vorerst)

# Wenn die reale Funktion ein gültiges Dictionary zurückgibt, mergen Sie es mit den Fallbacks

if isinstance(real\_texts, dict) and real\_texts:

# Aktualisieren Sie loaded\_texts mit den realen Texten (überschreibt Fallbacks bei Konflikt)

loaded\_texts.update(real\_texts)

# print("DEBUG: Reale Texte geladen und mit Fallbacks gemerged.") # Debugging

# else:

# print("DEBUG: Reale load\_translations gab leeres/ungültiges Dict zurück. Fallbacks aktiv.") # Debugging

# pass # loaded\_texts bleibt der Fallback

except Exception as e:

# Dieser Block fängt Fehler ab, die beim AUSFÜHREN von load\_translations auftreten (nicht beim Import)

import\_errors.append(f"Fehler beim Ausführen von load\_translations: {e}. Fallback-Texte aktiv.")

# print(f"DEBUG: Exception beim Ausführen von load\_translations: {e}") # Debugging

# loaded\_texts behält die Fallback-Texte (oder das Ergebnis des vorherigen Merges)

pass

# Jetzt die Hauptfunktion der Anwendung aufrufen und das vorbereitete texts Dictionary übergeben.

main(loaded\_texts) # Übergib das geladene oder gemergte Dictionary

**locales.py:**

# locales.py (Modul zur Text-Lokalisierung)

import json

import os

from typing import Dict, Any, Optional

# Pfad zum Verzeichnis der Sprachdateien (im selben Ordner wie locales.py)

# Annahme: Sprachdateien liegen im selben Ordner wie locales.py ODER in einem 'locales' Unterordner

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

LOCALES\_DIR = BASE\_DIR # Annahme: .json Dateien liegen direkt neben locales.py

# Alternativ: Annahme, sie liegen in einem Unterordner

# LOCALES\_DIR = os.path.join(BASE\_DIR, 'locales')

def load\_translations(lang\_code: str) -> Optional[Dict[str, str]]:

"""

Lädt die Übersetzungen für den gegebenen Sprachcode aus einer JSON-Datei.

Dateinamen erwartet: <lang\_code>.json (z.B. 'de.json', 'en.json')

"""

file\_name = f"{lang\_code}.json"

file\_path = os.path.join(LOCALES\_DIR, file\_name)

# Füge den globalen Fehler-Handler aus gui.py hinzu, falls verfügbar

import\_errors = []

try:

from gui import import\_errors as global\_import\_errors

import\_errors = global\_import\_errors # Nutze die globale Liste

except ImportError:

#print("locales.py: Globaler Fehler-Handler in gui.py nicht gefunden.") # Debugging

pass # Tue nichts, wenn der globale Handler nicht importiert werden kann

if not os.path.exists(file\_path):

error\_msg = f"Lokalisierungsdatei nicht gefunden: {file\_path}"

print(f"locales.py: {error\_msg}") # Logge den Fehler

if error\_msg not in import\_errors: # Vermeide Duplikate

import\_errors.append(error\_msg) # Füge Fehler hinzu

# st.error(error\_msg) # Kein Streamlit Aufruf auf Top-Level!

return None # Gib None zurück, wenn Datei nicht gefunden wird

try:

with open(file\_path, 'r', encoding='utf-8') as f:

translations = json.load(f)

# st.success(f"Lokalisierungsdatei geladen: {file\_name}") # Kein Streamlit Aufruf auf Top-Level!

return translations # Gib die geladenen Texte zurück

except json.JSONDecodeError as e:

error\_msg = f"Fehler beim Parsen der Lokalisierungsdatei {file\_name}: {e}"

print(f"locales.py: {error\_msg}") # Logge den Fehler

if error\_msg not in import\_errors: # Vermeide Duplikate

import\_errors.append(error\_msg) # Füge Fehler hinzu

# st.error(error\_msg) # Kein Streamlit Aufruf auf Top-Level!

return None # Gib None zurück bei Fehler

except Exception as e:

error\_msg = f"Ein unerwarteter Fehler ist beim Laden von {file\_name} aufgetreten: {e}"

print(f"locales.py: {error\_msg}") # Logge den Fehler

if error\_msg not in import\_errors: # Vermeide Duplikate

import\_errors.append(error\_msg) # Füge Fehler hinzu

# st.error(error\_msg) # Kein Streamlit Aufruf auf Top-Level!

return None # Gib None zurück bei Fehler

# # Beispiel: Testfunktion zum Laden der deutschen Texte

# if \_\_name\_\_ == "\_\_main\_\_":

# print("Teste Lokalisierungsmodul:")

# # Erstelle eine Dummy de.json für den Test

# dummy\_texts = {"test\_key": "Dies ist ein Testtext", "another\_key": "Noch ein Text"}

# dummy\_file\_path = os.path.join(BASE\_DIR, 'de.json')

# with open(dummy\_file\_path, 'w', encoding='utf-8') as f:

# json.dump(dummy\_texts, f, indent=2, ensure\_ascii=False)

# print(f"Dummy de.json erstellt unter {dummy\_file\_path}")

#

# loaded = load\_translations('de')

# if loaded:

# print("\nGeladene Texte (de):")

# print(loaded)

# print(f"Testtext für 'test\_key': {loaded.get('test\_key', 'NICHT GEFUNDEN')}")

# else:

# print("\nFehler beim Laden der Texte.")

#

# # Aufräumen

# # os.remove(dummy\_file\_path) # Optional: Dummy Datei wieder löschen

# # print(f"\nDummy de.json gelöscht unter {dummy\_file\_path}")

**pdf\_generator.py:**

# pdf\_generator.py (Placeholder Modul)

# Imports für zukünftige Funktionen

# from typing import Dict, Any

# from skins import get\_skin\_details # Benötigt Skin-Details für Branding

# from reportlab.pdfgen import canvas # Beispiel für PDF-Bibliothek

# import io

# Dieses Modul erstellt die PDF-Angebote und Verträge

# Beispiel: Funktion zum Generieren eines Angebots-PDFs

def generate\_offer\_pdf(project\_data: Dict[str, Any], analysis\_results: Dict[str, Any], skin\_id: int) -> Optional[bytes]:

"""Placeholder Funktion zur Generierung eines Angebots-PDFs."""

print(f"pdf\_generator: Placeholder generate\_offer\_pdf called for Project: {project\_data.get('project\_details', {}).get('project\_name', 'N/A')}, Skin ID: {skin\_id}") # Debugging

st.warning("PDF-Angebotsgenerierung ist ein Platzhalter.") # Info

# Hier kommt die Logik zur Erstellung des PDFs

# Nutzt project\_data, analysis\_results, skin\_details (von get\_skin\_details)

# Gibt den PDF-Inhalt als Bytes zurück

# Dummy PDF-Inhalt

dummy\_content = b"Dies ist ein Platzhalter-PDF Angebot.\n"

dummy\_content += f"Projekt: {project\_data.get('project\_details', {}).get('project\_name', 'N/A')}\n"

dummy\_content += f"Skin ID: {skin\_id}\n"

dummy\_content += "Echte Inhalte folgen.\n"

return dummy\_content # Gib Dummy Bytes zurück

# Beispiel: Funktion zum Generieren eines Vertrags-PDFs

# def generate\_contract\_pdf(...): pass # Kommt später

**Product\_db.py:**

# product\_db.py

# Modul zur Verwaltung der Produktdatenbank (SQLite)

import sqlite3

import pandas as pd

import json

from typing import Dict, List, Optional, Any

import traceback # Import traceback for detailed error logging

import os

import re # Import regex for parsing numerical values

# Importiere die Funktion, die die Datenbankverbindung bereitstellt, aus database.py

# Stelle sicher, dass database.py importierbar ist

try:

# WICHTIG: Verwende absolute Imports oder stelle sicher, dass dein PYTHONPATH korrekt ist.

# 'from database import ...' geht davon aus, dass das Modul 'database' im Python-Pfad ist.

from database import get\_db\_connection, init\_db, load\_admin\_setting # Benötigen get\_db\_connection, init\_db wird woanders gerufen, load\_admin\_setting wird nicht direkt hier genutzt aber gut zu wissen ob DB geht

# Optional: Dummy-Funktionen für den Fall, dass database.py nicht geladen werden kann

if not callable(get\_db\_connection): raise ImportError("Imported get\_db\_connection is not callable.")

# init\_db wird typischerweise in database.py beim Laden oder in der Hauptanwendung gerufen

# load\_admin\_setting wird hier nicht direkt genutzt, aber product\_db braucht eine funktionierende DB

# Wenn der Import und Check erfolgreich, weisen wir die realen Funktionen zu

get\_db\_connection\_safe\_pd = get\_db\_connection

# load\_admin\_setting\_safe\_pd = load\_admin\_setting # Diese wird hier eigentlich nicht direkt gebraucht, nur zum Test im \_\_main\_\_

except ImportError as e:

# Bei Importfehler weisen wir Dummy-Funktionen zu

def get\_db\_connection\_safe\_pd():

print(f"product\_db.py: Importfehler für database.py: {e}. Dummy DB-Verbindung.")

return None

# def load\_admin\_setting\_safe\_pd(key, default=None):

# print(f"product\_db.py: Importfehler für database.py. Dummy load\_admin\_setting.")

# return default # Gib immer den Standardwert zurück

print(f"product\_db.py: Importfehler für database.py: {e}. Dummy DB Funktionen werden genutzt.")

except Exception as e:

# Bei anderen Fehlern beim Laden weisen wir Dummy-Funktionen zu

def get\_db\_connection\_safe\_pd():

print(f"product\_db.py: Fehler beim Laden von database.py: {e}. Dummy DB-Verbindung.")

return None

# def load\_admin\_setting\_safe\_pd(key, default=None):

# print(f"product\_db.py: Fehler beim Laden von database.py. Dummy load\_admin\_setting.")

# return default # Gib immer den Standardwert zurück

print(f"product\_db.py: Fehler beim Laden von database.py: {e}. Dummy DB Funktionen werden genutzt.")

# --- Produktdatenbank Funktionen (mit SQLite) ---

def add\_product(category: str, brand: Optional[str], model\_name: str, price\_euro: float,

capacity\_w: Optional[float] = None, capacity\_kwh: Optional[float] = None,

power\_kw: Optional[float] = None, warranty\_years: Optional[float] = None,

length\_m: Optional[float] = None, width\_m: Optional[float] = None,

weight\_kg: Optional[float] = None, efficiency\_percent: Optional[float] = None,

origin\_country: Optional[str] = None, description: Optional[str] = None,

pros: Optional[List[str]] = None, cons: Optional[List[str]] = None,

rating: Optional[float] = None) -> int: # Rückgabe der ID oder -1 bei Fehler

"""Fügt ein neues Produkt in die Datenbank ein."""

conn = get\_db\_connection\_safe\_pd() # Nutze die '\_safe' Funktion

if conn is None:

print("add\_product: Keine Datenbankverbindung verfügbar.")

return -1 # Indicate failure

cursor = conn.cursor()

try:

# SQL-Befehl mit allen Spalten (auch den optionalen)

# Stelle sicher, dass die Spaltennamen genau denen in database.py CREATE TABLE entsprechen!

cursor.execute("""

INSERT INTO products (category, brand, model\_name, capacity\_w, capacity\_kwh, power\_kw,

price\_euro, warranty\_years, length\_m, width\_m, weight\_kg,

efficiency\_percent, origin\_country, description, pros, cons, rating)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (category, brand, model\_name, capacity\_w, capacity\_kwh, power\_kw,

price\_euro, warranty\_years, length\_m, width\_m, weight\_kg,

efficiency\_percent, origin\_country, description,

json.dumps(pros) if pros is not None else None, # JSON speichern, None wenn Liste leer oder None

json.dumps(cons) if cons is not None else None, # JSON speichern, None wenn Liste leer oder None

rating))

conn.commit()

# print(f"Produkt '{model\_name}' hinzugefügt.") # Etwas weniger gesprächig machen

return cursor.lastrowid # Gibt die ID des eingefügten Produkts zurück

except sqlite3.IntegrityError:

# Dies geschieht, wenn model\_name UNIQUE ist und bereits existiert

# print(f"Warnung: Produkt '{model\_name}' existiert bereits.") # Etwas weniger gesprächig machen

# Hier könnte man stattdessen ein Update durchführen oder -1 zurückgeben

return -1 # Indicate exists/failed insertion

except Exception as e:

print(f"Fehler beim Hinzufügen von Produkt '{model\_name}': {e}")

traceback.print\_exc() # Detaillierten Traceback ausgeben

conn.rollback() # Änderungen rückgängig machen

return -1 # Indicate failure

finally:

if conn: conn.close() # Verbindung schließen!

def update\_product(product\_id: int, data: Dict[str, Any]) -> bool:

"""Aktualisiert die Daten für ein vorhandenes Produkt anhand seiner ID."""

conn = get\_db\_connection\_safe\_pd() # Nutze die '\_safe' Funktion

if conn is None:

print("update\_product: Keine Datenbankverbindung verfügbar.")

return False # Indicate failure

cursor = conn.cursor()

try:

# Erstelle das SQL UPDATE Statement dynamisch

set\_clauses = []

values = []

for key, value in data.items():

# Ignoriere die ID und automatisch generierte Felder bei Updates

if key in ['id', 'created\_at', 'updated\_at']:

continue

# Stelle sicher, dass der Schlüssel eine Spalte in der Tabelle ist (Optional, aber sicherer)

# Man könnte hier eine Liste der erlaubten Spalten haben.

# Vorerst gehen wir davon aus, dass die übergebenen Schlüssel gültig sind.

set\_clauses.append(f"{key} = ?")

# Handle JSON fields specifically if needed, otherwise standard types

if isinstance(value, (list, dict)):

values.append(json.dumps(value))

# Handle None Werte, die als NULL in SQL gespeichert werden sollen

elif value is None:

values.append(None)

else:

values.append(value)

if not set\_clauses:

# print(f"Keine gültigen Felder zum Aktualisieren für Produkt ID {product\_id} gefunden.") # Etwas weniger gesprächig machen

return False # Nichts zu aktualisieren

# Füge updated\_at automatisch hinzu (falls die Spalte existiert)

# set\_clauses.append("updated\_at = CURRENT\_TIMESTAMP") # Spalte existiert schon in CREATE TABLE

# values.append() # Kein Wert für CURRENT\_TIMESTAMP nötig im VALUES-Tuple

sql = f"UPDATE products SET {', '.join(set\_clauses)} WHERE id = ?"

values.append(product\_id) # Die ID für die WHERE-Klausel

cursor.execute(sql, tuple(values))

conn.commit()

# print(f"Produkt ID {product\_id} aktualisiert.") # Etwas weniger gesprächig machen

return cursor.rowcount > 0 # True, wenn mindestens eine Zeile aktualisiert wurde

except Exception as e:

print(f"Fehler beim Aktualisieren von Produkt ID {product\_id}: {e}")

traceback.print\_exc()

conn.rollback() # Änderungen rückgängig machen

return False # Indicate failure

finally:

if conn: conn.close() # Verbindung schließen!

def delete\_product(product\_id: int) -> bool:

"""Löscht ein Produkt anhand seiner ID."""

conn = get\_db\_connection\_safe\_pd() # Nutze die '\_safe' Funktion

if conn is None:

print("delete\_product: Keine Datenbankverbindung verfügbar.")

return False # Indicate failure

cursor = conn.cursor()

try:

cursor.execute("DELETE FROM products WHERE id = ?", (product\_id,))

conn.commit()

# print(f"Produkt ID {product\_id} gelöscht.") # Etwas weniger gesprächig machen

return cursor.rowcount > 0 # True if a row was deleted

except Exception as e:

print(f"Fehler beim Löschen von Produkt ID {product\_id}: {e}")

traceback.print\_exc()

conn.rollback() # Änderungen rückgängig machen

return False # Indicate failure

finally:

if conn: conn.close() # Verbindung schließen!

def get\_product\_by\_id(product\_id: int) -> Optional[Dict[str, Any]]:

"""Sucht ein Produkt anhand seiner ID und gibt es als Dictionary zurück."""

conn = get\_db\_connection\_safe\_pd() # Nutze die '\_safe' Funktion

if conn is None:

print("get\_product\_by\_id: Keine Datenbankverbindung verfügbar.")

return None # Indicate failure

cursor = conn.cursor()

try:

cursor.execute("SELECT \* FROM products WHERE id = ?", (product\_id,))

row = cursor.fetchone()

# conn.close() # Verbindung am Ende schließen

if row:

# Konvertiere sqlite3.Row zu Dictionary und parse JSON-Felder zurück

product\_data = dict(row)

for field in ['pros', 'cons']:

# Nur versuchen zu parsen, wenn das Feld existiert und nicht None ist

if field in product\_data and product\_data[field] is not None:

try:

product\_data[field] = json.loads(product\_data[field])

except (json.JSONDecodeError, TypeError): # Fange auch TypeError ab, falls Datenformat falsch ist

product\_data[field] = [] # Bei Fehler leere Liste

return product\_data

return None # Produkt nicht gefunden

except Exception as e:

print(f"Fehler bei get\_product\_by\_id ({product\_id}): {e}")

traceback.print\_exc()

return None # Indicate failure

finally:

if conn: conn.close() # Verbindung schließen!

def get\_product\_by\_model\_name(model\_name: Optional[str]) -> Optional[Dict[str, Any]]:

"""Sucht ein Produkt anhand seines Modellnamens und gibt es als Dictionary zurück."""

if model\_name is None or not model\_name.strip():

return None

conn = get\_db\_connection\_safe\_pd() # Nutze die '\_safe' Funktion

if conn is None:

# print("get\_product\_by\_model\_name: Keine Datenbankverbindung verfügbar.") # Zu gesprächig

return None # Indicate failure

cursor = conn.cursor()

try:

cursor.execute("SELECT \* FROM products WHERE model\_name = ?", (model\_name,))

row = cursor.fetchone()

# conn.close() # Verbindung am Ende schließen

if row:

product\_data = dict(row)

for field in ['pros', 'cons']:

# Nur versuchen zu parsen, wenn das Feld existiert und nicht None ist

if field in product\_data and product\_data[field] is not None:

try:

product\_data[field] = json.loads(product\_data[field])

except (json.JSONDecodeError, TypeError): # Fange auch TypeError ab

product\_data[field] = []

return product\_data

return None # Produkt nicht gefunden

except Exception as e:

print(f"Fehler bei get\_product\_by\_model\_name ('{model\_name}'): {e}")

traceback.print\_exc()

return None # Indicate failure

finally:

if conn: conn.close() # Verbindung schließen!

def list\_products(category: Optional[str] = None) -> List[Dict[str, Any]]:

"""Listet alle Produkte auf, optional gefiltert nach Kategorie."""

conn = get\_db\_connection\_safe\_pd() # Nutze die '\_safe' Funktion

if conn is None:

# print("list\_products: Keine Datenbankverbindung verfügbar.") # Zu gesprächig

return [] # Leere Liste bei Fehler

cursor = conn.cursor()

try:

if category:

# Suche nach Kategorie, case-insensitive

cursor.execute("SELECT \* FROM products WHERE LOWER(category) = LOWER(?) ORDER BY brand, model\_name", (category,))

else:

cursor.execute("SELECT \* FROM products ORDER BY category, brand, model\_name")

rows = cursor.fetchall()

# conn.close() # Verbindung am Ende schließen

# Konvertiere sqlite3.Row Objekte und parse JSON-Felder

product\_list = []

for row in rows:

product\_data = dict(row)

for field in ['pros', 'cons']:

# Nur versuchen zu parsen, wenn das Feld existiert und nicht None ist

if field in product\_data and product\_data[field] is not None:

try:

product\_data[field] = json.loads(product\_data[field])

except (json.JSONDecodeError, TypeError): # Fange auch TypeError ab

product\_data[field] = [] # Bei Fehler leere Liste

product\_list.append(product\_data)

return product\_list

except Exception as e:

print(f"Fehler bei list\_products (Kategorie: {category}): {e}")

traceback.print\_exc()

return [] # Leere Liste bei Fehler

finally:

if conn: conn.close() # Verbindung schließen!

def list\_product\_brands(category: Optional[str] = None) -> List[str]:

"""Listet alle eindeutigen Marken auf, optional gefiltert nach Kategorie."""

conn = get\_db\_connection\_safe\_pd()

if conn is None:

print("list\_product\_brands: Keine Datenbankverbindung verfügbar.")

return []

cursor = conn.cursor()

try:

if category:

cursor.execute("SELECT DISTINCT brand FROM products WHERE LOWER(category) = LOWER(?) AND brand IS NOT NULL ORDER BY brand", (category,))

else:

cursor.execute("SELECT DISTINCT brand FROM products WHERE brand IS NOT NULL ORDER BY brand")

rows = cursor.fetchall()

return [row['brand'] for row in rows]

except Exception as e:

print(f"Fehler bei list\_product\_brands (Kategorie: {category}): {e}")

traceback.print\_exc()

return []

finally:

if conn: conn.close()

def list\_product\_models\_by\_brand(brand: str, category: Optional[str] = None) -> List[str]:

"""Listet alle Modellnamen für eine bestimmte Marke auf, optional gefiltert nach Kategorie."""

conn = get\_db\_connection\_safe\_pd()

if conn is None:

print("list\_product\_models\_by\_brand: Keine Datenbankverbindung verfügbar.")

return []

cursor = conn.cursor()

try:

if category:

cursor.execute("SELECT model\_name FROM products WHERE brand = ? AND LOWER(category) = LOWER(?) AND model\_name IS NOT NULL ORDER BY model\_name", (brand, category))

else:

cursor.execute("SELECT model\_name FROM products WHERE brand = ? AND model\_name IS NOT NULL ORDER BY model\_name", (brand,))

rows = cursor.fetchall()

return [row['model\_name'] for row in rows]

except Exception as e:

print(f"Fehler bei list\_product\_models\_by\_brand (Marke: {brand}, Kategorie: {category}): {e}")

traceback.print\_exc()

return []

finally:

if conn: conn.close()

def list\_product\_models\_by\_category(category: str) -> List[str]:

"""Listet alle Modellnamen einer bestimmten Kategorie auf."""

conn = get\_db\_connection\_safe\_pd() # Nutze die '\_safe' Funktion

if conn is None:

print("list\_product\_models\_by\_category: Keine Datenbankverbindung verfügbar.")

return [] # Leere Liste bei Fehler

cursor = conn.cursor()

try:

# Suche nach Kategorie, case-insensitive

cursor.execute("SELECT model\_name FROM products WHERE LOWER(category) = LOWER(?) AND model\_name IS NOT NULL ORDER BY model\_name", (category,))

rows = cursor.fetchall()

return [row['model\_name'] for row in rows]

except Exception as e:

print(f"Fehler bei list\_product\_models\_by\_category (Kategorie: {category}): {e}")

traceback.print\_exc()

return [] # Leere Liste bei Fehler

finally:

if conn: conn.close()

# --- Import/Export Funktionen (z.B. aus Excel für Admin F) ---

def import\_products\_from\_excel(excel\_path: str) -> Dict[str, int]:

"""Importiert Produkte aus einer Excel-Datei in die Datenbank. Gibt Zähler zurück."""

results = {"imported": 0, "skipped\_errors": 0, "skipped\_no\_data": 0}

conn = None # Verbindung initial None

try:

# Initialisiere DB, falls noch nicht geschehen (sollte schon beim Start der App passiert sein)

# init\_db() # Besser nicht hier aufrufen, um Dopplung oder Timing Probleme zu vermeiden.

conn = get\_db\_connection\_safe\_pd() # Nutze die '\_safe' Funktion

if conn is None:

print("import\_products\_from\_excel: Keine Datenbankverbindung verfügbar.")

results["skipped\_errors"] = -1 # Indiziere DB nicht verfügbar

return results

df = pd.read\_excel(excel\_path)

# Stelle sicher, dass die Spaltennamen bereinigt sind (Whitespace entfernen)

df.columns = [col.strip() if isinstance(col, str) else col for col in df.columns]

# Mapping der Spaltennamen: Versuche gängige deutsche/englische Namen

# Beinhaltet alle Spalten, die in der 'products' Tabelle sind

col\_mapping = {

'category': ['category', 'Typ', 'Kategorie'],

'brand': ['brand', 'Marke', 'Hersteller'],

'model\_name': ['model\_name', 'Modell', 'Modulname', 'Name'], # WICHTIG: Muss eindeutig sein

'capacity\_w': ['capacity\_w', 'Leistung\_W', 'Leistung (W)', 'Watt'], # Für Module, WR (in W)

'capacity\_kwh': ['capacity\_kwh', 'Kapazität\_kWh', 'Kapazität (kWh)'], # Für Speicher (in kWh)

'power\_kw': ['power\_kw', 'Leistung\_kW', 'Leistung (kW)'], # Für WR, Wallbox (in kW)

'price\_euro': ['price\_euro', 'Preis\_Euro', 'Preis (€)'],

'warranty\_years': ['warranty\_years', 'Garantie\_Jahre', 'Garantie (Jahre)', 'Garantie'], # Hinzugefügt 'Garantie'

'length\_m': ['length\_m', 'Länge\_m', 'Länge (m)'],

'width\_m': ['width\_m', 'Breite\_m', 'Breite (m)'],

'weight\_kg': ['weight\_kg', 'Gewicht\_kg', 'Gewicht (kg)'],

'efficiency\_percent': ['efficiency\_percent', 'Wirkungsgrad\_Prozent', 'Wirkungsgrad (%)', 'Wirkungsgrad'],

'origin\_country': ['origin\_country', 'Herkunftsland', 'Land'],

'description': ['description', 'Beschreibung'],

'pros': ['pros', 'Vorteile'], # Erwartet kommaseparierten String oder JSON

'cons': ['cons', 'Nachteile'], # Erwartet kommaseparierten String oder JSON

'rating': ['rating', 'Bewertung']

}

# Finde die tatsächlichen Spaltennamen im DataFrame basierend auf dem Mapping

actual\_cols = {}

for key, possible\_names in col\_mapping.items():

for name in possible\_names:

if name in df.columns:

actual\_cols[key] = name

break # Nehme den ersten gefundenen Namen

# Prüfe, ob kritische Spalten gefunden wurden

critical\_missing = [key for key in ['category', 'model\_name', 'price\_euro'] if key not in actual\_cols]

if critical\_missing:

print(f"Import abgebrochen: Kritische Spalten fehlen in der Excel-Datei: {critical\_missing}")

# Füge Fehler zu den Ergebnissen hinzu, ggf. für Anzeige in Admin-GUI

results["error"] = f"Kritische Spalten fehlen: {', '.join(critical\_missing)}"

return results

cursor = conn.cursor()

# Iteriere über jede Zeile im DataFrame

for index, row in df.iterrows():

row\_num = index + 2 # Zeilennummer in Excel (Header = 1, Index startet bei 0)

# Versuche, den Modellnamen zu bekommen (strippen und als String sicherstellen)

model\_name = row.get(actual\_cols.get('model\_name', ''))

if pd.notna(model\_name):

model\_name = str(model\_name).strip()

else:

model\_name = '' # Setze auf leeren String, wenn NaN oder None

try:

# Überspringe leere Zeilen oder Zeilen ohne Modellname/Kategorie

category = row.get(actual\_cols.get('category', ''))

if pd.notna(category):

category = str(category).strip()

else:

category = '' # Setze auf leeren String

if not model\_name or not category:

# print(f"Zeile {row\_num} übersprungen: Modellname oder Kategorie leer.") # Zu gesprächig

results["skipped\_no\_data"] += 1

continue # Springe zur nächsten Zeile

# Daten für INSERT vorbereiten

product\_data = {

'category': category,

'brand': str(row.get(actual\_cols.get('brand', ''))).strip() if pd.notna(row.get(actual\_cols.get('brand', ''))) else None,

'model\_name': model\_name,

# Konvertiere numerische Felder sicher, ersetze Kommas durch Punkte

# pd.to\_numeric mit errors='coerce' versucht die Umwandlung und setzt bei Fehler NaN

'capacity\_w': pd.to\_numeric(row.get(actual\_cols.get('capacity\_w', None)), errors='coerce'),

'capacity\_kwh': pd.to\_numeric(row.get(actual\_cols.get('capacity\_kwh', None)), errors='coerce'),

'power\_kw': pd.to\_numeric(row.get(actual\_cols.get('power\_kw', None)), errors='coerce'),

'price\_euro': pd.to\_numeric(row.get(actual\_cols.get('price\_euro', None)), errors='coerce'), # price\_euro darf 0.0 sein, nicht None

'warranty\_years': pd.to\_numeric(row.get(actual\_cols.get('warranty\_years', None)), errors='coerce'),

'length\_m': pd.to\_numeric(row.get(actual\_cols.get('length\_m', None)), errors='coerce'),

'width\_m': pd.to\_numeric(row.get(actual\_cols.get('width\_m', None)), errors='coerce'),

'weight\_kg': pd.to\_numeric(row.get(actual\_cols.get('weight\_kg', None)), errors='coerce'),

'efficiency\_percent': pd.to\_numeric(row.get(actual\_cols.get('efficiency\_percent', None)), errors='coerce'),

'rating': pd.to\_numeric(row.get(actual\_cols.get('rating', None)), errors='coerce'),

# Textfelder

'origin\_country': str(row.get(actual\_cols.get('origin\_country', ''))).strip() if pd.notna(row.get(actual\_cols.get('origin\_country', ''))) else None,

'description': str(row.get(actual\_cols.get('description', ''))).strip() if pd.notna(row.get(actual\_cols.get('description', ''))) else None,

# Annahme: Pros/Cons sind eventuell kommasepariert in Excel oder fehlen

'pros': [p.strip() for p in str(row.get(actual\_cols.get('pros', '')) or '').split(',') if p.strip()] if pd.notna(row.get(actual\_cols.get('pros', ''))) else None,

'cons': [c.strip() for c in str(row.get(actual\_cols.get('cons', '')) or '').split(',') if c.strip()] if pd.notna(row.get(actual\_cols.get('cons', ''))) else None,

}

# Behandle numerische Felder, bei denen Umwandlung mit coerce fehlschlug (NaN)

# und solche, die aus Excel kommen und String-Bereinigung brauchen (z.B. '10 Jahre')

# Diese zusätzliche Logik wird nur ausgeführt, wenn pd.to\_numeric zu NaN führte

for num\_col in ['capacity\_w', 'capacity\_kwh', 'power\_kw', 'price\_euro',

'warranty\_years', 'length\_m', 'width\_m', 'weight\_kg',

'efficiency\_percent', 'rating']:

if pd.isna(product\_data[num\_col]): # Wenn Umwandlung mit coerce fehlschlug (Wert war kein reiner Zahlentyp)

original\_value = row.get(actual\_cols.get(num\_col, None))

if pd.notna(original\_value):

try:

# Versuche, den String zu bereinigen (Leerzeichen entfernen, Komma zu Punkt)

cleaned\_value\_str = str(original\_value).strip().replace(',', '.')

# Versuche, die Zahl am Anfang des Strings zu extrahieren (z.B. '10 Jahre' -> 10)

# Erlaubt nun auch Komma als Dezimaltrennzeichen VOR dem replace

num\_match = re.match(r'^\d+([.,]\d+)?', str(original\_value).strip().replace(',', '.')) # Regex für Zahl am Anfang (erlaubt . oder ,)

if num\_match:

# Nutze die geparste Zahl, konvertiere zu float

product\_data[num\_col] = float(num\_match.group(0))

else:

# Wenn keine Zahl am Anfang gefunden wird, setze auf None

product\_data[num\_col] = None

if num\_col not in ['rating', 'warranty\_years']: # Rating und Warranty Years dürfen leer sein ohne Warnung

print(f"Warnung Zeile {row\_num} ('{model\_name}'): Konnte '{actual\_cols.get(num\_col, num\_col)}' Wert '{original\_value}' nicht als Zahl parsen/extrahieren.")

except Exception:

product\_data[num\_col] = None

if num\_col not in ['rating', 'warranty\_years']:

print(f"Warnung Zeile {row\_num} ('{actual\_cols.get(num\_col, num\_col)}'): Fehler beim String-Parsing für '{actual\_cols.get(num\_col, num\_col)}' Wert '{original\_value}'.")

else: # Originalwert war schon NaN oder None

product\_data[num\_col] = None

# Stelle sicher, dass price\_euro nicht None ist, wenn er eigentlich da sein sollte (Standard 0.0)

# Preis sollte immer gesetzt sein.

if product\_data.get('price\_euro') is None:

product\_data['price\_euro'] = 0.0

# INSERT OR REPLACE: Versucht einzufügen, bei UNIQUE Konflikt wird ersetzt.

# Nutze REPLACE, um Aktualisierungen per Import zu ermöglichen

cursor.execute("""

INSERT OR REPLACE INTO products (category, brand, model\_name, capacity\_w, capacity\_kwh, power\_kw,

price\_euro, warranty\_years, length\_m, width\_m, weight\_kg,

efficiency\_percent, origin\_country, description, pros, cons, rating)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (product\_data.get('category'), product\_data.get('brand'), product\_data.get('model\_name'),

product\_data.get('capacity\_w'), product\_data.get('capacity\_kwh'), product\_data.get('power\_kw'),

product\_data.get('price\_euro'), product\_data.get('warranty\_years'), product\_data.get('length\_m'),

product\_data.get('width\_m'), product\_data.get('weight\_kg'), product\_data.get('efficiency\_percent'),

product\_data.get('origin\_country'), product\_data.get('description'),

json.dumps(product\_data.get('pros')) if product\_data.get('pros') is not None else None,

json.dumps(product\_data.get('cons')) if product\_data.get('cons') is not None else None,

product\_data.get('rating')))

results["imported"] += 1

except Exception as e:

print(f"Fehler beim Importieren von Zeile {row\_num} ('{model\_name}'): {e}")

traceback.print\_exc()

results["skipped\_errors"] += 1 # Zähle als übersprungen bei Fehler

conn.commit()

# print(f"Import abgeschlossen. Erfolgreich: {results['imported']}. Übersprungen (leer): {results['skipped\_no\_data']}. Übersprungen (Fehler): {results['skipped\_errors']}.") # Zu gesprächig

except FileNotFoundError:

print(f"Fehler: Excel-Datei nicht gefunden unter {excel\_path}")

results["error"] = f"Excel-Datei nicht gefunden unter {excel\_path}"

except pd.errors.EmptyDataError:

print(f"Fehler: Excel-Datei ist leer oder hat keine lesbaren Blätter: {excel\_path}")

results["error"] = "Excel-Datei ist leer oder hat keine lesbaren Blätter."

except Exception as e:

print(f"Ein Fehler ist während des Excel-Imports aufgetreten: {e}")

traceback.print\_exc()

results["error"] = f"Ein unerwarteter Fehler ist aufgetreten: {e}"

finally:

if conn:

conn.close()

return results # Gibt das Ergebnis-Dictionary zurück

# --- Helferfunktion zur Suche nach Produkten ---

# Diese Funktionen nutzen die get\_product\_by\_model\_name oder list\_products

# und handhaben automatisch die Datenbankverbindung über die get\_db\_connection Funktion (oder ihre Dummy-Version).

def lookup\_product\_price(model\_name: Optional[str]) -> float:

"""Sucht den Preis eines Produkts anhand des Modellnamens."""

if model\_name is None or not model\_name.strip():

return 0.0

product = get\_product\_by\_model\_name(model\_name) # Nutze die get\_product\_by\_model\_name Funktion (ist im Modul definiert)

if product and product.get('price\_euro') is not None:

return float(product['price\_euro'])

return 0.0 # Gib 0 zurück, wenn Produkt nicht gefunden oder Preis fehlt/ungültig ist

def lookup\_module\_details(model\_name: Optional[str]) -> Optional[Dict[str, Any]]:

"""Sucht Details eines Solarmoduls anhand des Modellnamens."""

if model\_name is None or not model\_name.strip():

return None

product = get\_product\_by\_model\_name(model\_name) # Nutze die get\_product\_by\_model\_name Funktion (ist im Modul definiert)

if product and product.get('category', '').lower() == 'modul':

return product

return None # Nicht gefunden oder keine Modul-Kategorie

# Füge weitere lookup Funktionen hinzu (WR, Storage, etc.)

def lookup\_inverter\_details(model\_name: Optional[str]) -> Optional[Dict[str, Any]]:

"""Sucht Details eines Wechselrichters anhand des Modellnamens."""

if model\_name is None or not model\_name.strip():

return None

product = get\_product\_by\_model\_name(model\_name)

if product and product.get('category', '').lower() in ['wechselrichter', 'inverter']:

return product

return None

def lookup\_storage\_details(model\_name: Optional[str]) -> Optional[Dict[str, Any]]:

"""Sucht Details eines Batteriespeichers anhand des Modellnamens."""

if model\_name is None or not model\_name.strip():

return None

product = get\_product\_by\_model\_name(model\_name)

if product and product.get('category', '').lower() in ['batteriespeicher', 'storage']:

return product

return None

def lookup\_wallbox\_details(model\_name: Optional[str]) -> Optional[Dict[str, Any]]:

"""Sucht Details einer Wallbox anhand des Modellnamens."""

if model\_name is None or not model\_name.strip():

return None

product = get\_product\_by\_model\_name(model\_name)

if product and product.get('category', '').lower() in ['wallbox', 'emobility']: # Annahme Kategorie 'wallbox' oder 'emobility'

return product

return None

def lookup\_ems\_details(model\_name: Optional[str]) -> Optional[Dict[str, Any]]:

"""Sucht Details eines Energiemanagementsystems anhand des Modellnamens."""

if model\_name is None or not model\_name.strip():

return None

product = get\_product\_by\_model\_name(model\_name)

if product and product.get('category', '').lower() in ['energiemanagementsystem', 'ems']: # Annahme Kategorie 'energiemanagementsystem' oder 'ems'

return product

return None

def lookup\_optimizer\_details(model\_name: Optional[str]) -> Optional[Dict[str, Any]]:

"""Sucht Details eines Leistungsoptimierers anhand des Modellnamens."""

if model\_name is None or not model\_name.strip():

return None

product = get\_product\_by\_model\_name(model\_name)

if product and product.get('category', '').lower() in ['leistungsoptimierer', 'optimizer']: # Annahme Kategorie 'leistungsoptimierer' oder 'optimizer'

return product

return None

def lookup\_carport\_details(model\_name: Optional[str]) -> Optional[Dict[str, Any]]:

"""Sucht Details eines Carports anhand des Modellnamens."""

if model\_name is None or not model\_name.strip():

return None

product = get\_product\_by\_model\_name(model\_name)

if product and product.get('category', '').lower() == 'carport': # Annahme Kategorie 'carport'

return product

return None

def lookup\_notstrom\_details(model\_name: Optional[str]) -> Optional[Dict[str, Any]]:

"""Sucht Details einer Notstromlösung anhand des Modellnamens."""

if model\_name is None or not model\_name.strip():

return None

product = get\_product\_by\_model\_name(model\_name)

if product and product.get('category', '').lower() in ['notstrom', 'backup']: # Annahme Kategorie 'notstrom' oder 'backup'

return product

return None

def lookup\_tierabwehr\_details(model\_name: Optional[str]) -> Optional[Dict[str, Any]]:

"""Sucht Details einer Tierabwehrlösung anhand des Modellnamens."""

if model\_name is None or not model\_name.strip():

return None

product = get\_product\_by\_model\_name(model\_name)

if product and product.get('category', '').lower() in ['tierabwehr', 'schutz']: # Annahme Kategorie 'tierabwehr' oder 'schutz'

return product

return None

# list\_product\_models\_by\_category etc. sind bereits oben definiert und nutzen get\_db\_connection\_safe\_pd

# --- Optional: Skript zum Hinzufügen/Importieren von Daten (für Tests oder Admin-Setup) ---

if \_\_name\_\_ == "\_\_main\_\_":

print("--- Datenimport/Testlauf für product\_db.py ---")

# Stelle sicher, dass die DB existiert und initialisiert ist.

# Die init\_db() Funktion in database.py wird beim Import von database.py gerufen (wenn database.py nicht schon lief).

try:

# Importiere database explizit für den Test

import database

database.init\_db() # Stelle sicher, dass die Tabellen da sind und Migrationen laufen

print("\nDatenbank Initialisierung/Check im Test abgeschlossen.")

except ImportError:

print("FEHLER: Kann database.py nicht importieren, Datenbank-Check/Init nicht möglich.")

# Ohne database können wir nicht weitermachen, brechen ab oder springen zu Excel-only Logik (nicht implementiert)

exit() # Beende das Skript, wenn database nicht importiert werden kann

except Exception as e:

print(f"FEHLER bei Datenbank Initialisierung/Check: {e}")

traceback.print\_exc()

# Bei einem Fehler in init\_db() brechen wir den Import ab

exit()

# --- Optionale Aktionen: Testdaten hinzufügen oder Excel importieren ---

# Wähle EINE Aktion, die du hier testweise ausführen möchtest.

# AKTION 1: Füge einige Testprodukte hinzu (für manuelle Tests)

# Entkommentiere den Block unten, um Testdaten hinzuzufügen

# print("\nFüge Testprodukte hinzu (wird bei doppelten Namen ignoriert)...")

# add\_product("Modul", "TestBrand", "TestModul 400W", 150.0, capacity\_w=400.0, warranty\_years=25.0, length\_m=1.7, width\_m=1.0)

# add\_product("Modul", "TestBrand", "PremiumModul 450W", 180.0, capacity\_w=450.0, warranty\_years=30.0, length\_m=1.9, width\_m=1.1)

# add\_product("Wechselrichter", "TestWR", "HomeWR 5KW", 1200.0, power\_kw=5.0, warranty\_years=10.0)

# add\_product("Batteriespeicher", "TestStorage", "HomeStorage 10kWh", 4000.0, capacity\_kwh=10.0, warranty\_years=10.0)

# add\_product("Wallbox", "TestWB", "SmartBox 11kW", 500.0, power\_kw=11.0, warranty\_years=5.0)

# add\_product("Energiemanagementsystem", "TestEMS", "SmartHome EMS", 800.0, warranty\_years=2.0)

# add\_product("Leistungsoptimierer", "TestOpt", "Optimizer", 50.0, warranty\_years=25.0)

# add\_product("Carport", "TestCarport", "Holz Single", 3000.0) # Beispiel ohne viele Details

# add\_product("Notstrom", "TestBackup", "BackupBox", 1500.0, power\_kw=3.0)

# add\_product("Tierabwehr", "TestProtection", "Vogelabwehr", 20.0)

# print("Testprodukte hinzugefügt/gecheckt.")

# # Springe zum Anzeige-Teil nach dem Hinzufügen von Testdaten

# all\_products\_after\_import = list\_products() # Lade die Produkte neu

# print("\n--- Produkte in der Datenbank nach Hinzufügen (erste 10) ---")

# if all\_products\_after\_import:

# for i, p in enumerate(all\_products\_after\_import[:10]):

# print(f"ID: {p.get('id')}, Kategorie: {p.get('category')}, Modell: {p.get('model\_name')}, Preis: {p.get('price\_euro', 0.0):.2f}€, Leistung(W): {p.get('capacity\_w')}, Kapazität(kWh): {p.get('capacity\_kwh')}, Garantie(Jahre): {p.get('warranty\_years')}")

# if len(all\_products\_after\_import) > 10:

# print(f"... {len(all\_products\_after\_import) - 10} weitere Produkte nicht angezeigt.")

# else:

# print("Keine Produkte in der Datenbank gefunden nach dem Hinzufügen.")

# print("\n--- Testlauf beendet ---")

# exit() # Skript hier beenden, wenn nur Testdaten hinzugefügt werden sollen

# AKTION 2: Importiere Produkte aus deiner Excel-Datei

# Empfohlen, nachdem die Excel bereinigt wurde (nur Zahlen in numerischen Spalten)

# Optional: Lösche vorher alle Produkte, um sauber neu zu importieren

confirm\_delete = input("Sollen ALLE Produkte in der Datenbank gelöscht werden, bevor aus Excel importiert wird? (j/n): ")

if confirm\_delete.lower() == 'j':

conn\_del = get\_db\_connection\_safe\_pd()

if conn\_del:

try:

cursor\_del = conn\_del.cursor()

cursor\_del.execute("DELETE FROM products")

conn\_del.commit()

print("Alle vorhandenen Produkte gelöscht.")

except Exception as e:

print(f"Fehler beim Löschen aller Produkte: {e}")

traceback.print\_exc()

conn\_del.rollback()

finally:

conn\_del.close()

else:

print("Keine Datenbankverbindung verfügbar, Produkte können nicht gelöscht werden.")

else:

print("Vorhandene Produkte werden NICHT gelöscht vor dem Import.")

print("\nStarte Import aus Excel-Datei...")

# PASSE DEN DATEINAMEN HIER AN DEN NAMEN DEINER EXCEL-DATEI AN!

excel\_file\_path = os.path.join("data", "MeineProdukte.xlsx") # Beispielname, anpassen!

import\_results = import\_products\_from\_excel(excel\_file\_path) # Rufe die Importfunktion auf

print("\n--- Import Ergebnis ---")

print(f"Erfolgreich importiert/aktualisiert: {import\_results.get('imported', 0)}")

print(f"Übersprungen (leere Zeilen): {import\_results.get('skipped\_no\_data', 0)}")

print(f"Übersprungen (Fehler beim Verarbeiten): {import\_results.get('skipped\_errors', 0)}")

if 'error' in import\_results:

print(f"Kritischer Fehler während des Imports: {import\_results['error']}")

print("\n--- Produkte in der Datenbank nach Import (erste 10) ---")

all\_products\_after\_import = list\_products() # Lade die Produkte neu

if all\_products\_after\_import:

for i, p in enumerate(all\_products\_after\_import[:10]): # Zeige nur die ersten 10, wenn es viele sind

# Drucke relevante Spalten, nutze .get() für Sicherheit

print(f"ID: {p.get('id')}, Kategorie: {p.get('category')}, Modell: {p.get('model\_name')}, Preis: {p.get('price\_euro', 0.0):.2f}€, Leistung(W): {p.get('capacity\_w')}, Kapazität(kWh): {p.get('capacity\_kwh')}, Garantie(Jahre): {p.get('warranty\_years')}")

if len(all\_products\_after\_import) > 10:

print(f"... {len(all\_products\_after\_import) - 10} weitere Produkte nicht angezeigt.")

else:

print("Keine Produkte in der Datenbank gefunden nach dem Import.")

print("\n--- Testlauf beendet ---")